Multi-task Asynchronous Deep Reinforcement Learning

Chaoyang Wang,
chaoyanw@andrew.cmu.edu

Luona Yang
luonayl@andrew.cmu.edu

Abstract

Learning to act upon different tasks and transfer knowledge in between is a crucial ability for intelligence in real life. One approach to acquire such ability for an agent is through multi-task reinforcement learning. However, due to the sequential exploring and model updating nature of many reinforcement learning methods, it's challenging to train the agent to perform multiple different tasks simultaneously. In this work, we propose a computationally efficient multi-task asynchronous learning framework, which requires only multi-threading on CPU for training, generalizable to a wide variety of RL methods, and in theory, scales well to the number of tasks. We test the proposed method with Atari games. Through the experiment, we not only show that asynchronous multi-task training is able to compress the policy and value networks for the agent through aggressive parameter sharing, but also observe significant performance gain on several games.

1 Introduction

Living in a complex world, humans are simultaneously doing all sorts of different tasks, and gain experiences from that. It's usually the case that the more experienced a person become, the better insights he has for problems which he has never encountered before. This kind of argument is used to justify the benifits of transfer learning and multi-task learning, and numerous works from different fields have shown that transfer/multi-task learning can indeed improve either the performance, or relax the size requirement of training dataset for the target task. Hence, we believe that exploring the idea that knowledge can be shared among different tasks(or environment in terms of RL) is very promising to improve the intelligence capability of the agent.

There have already been several prior works [1, 2, 3, 4, 5] also move in this direction, which we will discuss in section 1.1. Unlike those works, in this project, we want to explore the possibility to learn a multi-task value/policy network directly from scratch, and to see how much knowledge we can share among different tasks in the form of parameter sharing. We choose to do direct multi-task training, because it's more computationally efficient compared to knowledge distilling [1, 3, 6], which requires you to have multiple teacher models beforehand; and the model complexity is potentially less complex compared to model-based transfer learning [4], since it's not required to explicitly model a real environment.

Prior arts in multi-task training usually involve a combination of loss function for different tasks, and update the shared parameters synchronously respect to the fused loss function. However, this kind of approach is unscalable in previous deep RL methods like deep Q-learning, since they require exploring the environment, aggregate training data, and update the model parameters in a sequential manner, which results in a spatial/temporal cost proportional to the number of tasks. As a result, directly apply these methods for multi-task training is infeasible when the number of tasks becomes large.

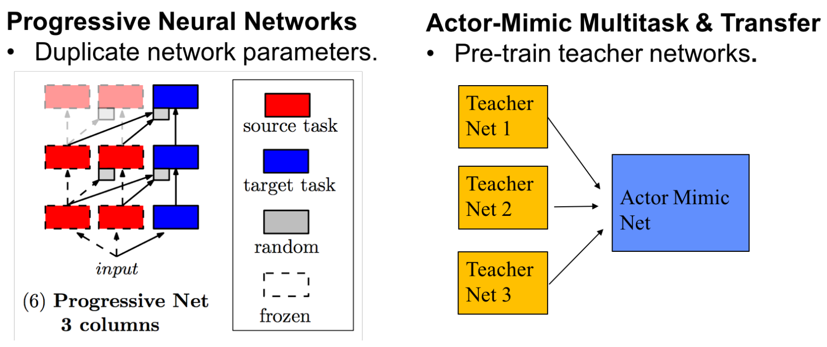


Figure 1: Illustration of previous transfer learning methods in reinforcement learning: progressive neural network(left), and Actor-mimic multi-task transfer RL(right).

To make training scalable, we look into the asynchronous method for deep reinforcement learning [7], which has proven to be a computationally efficient, generizable to both on/off policy methods, and performance-wise superior approach. It turns out that incorporating multiple different game environments within this framework is natural and efficient: we perform knowledge transfer through sharing model parameters across different tasks, and we achieve scalable multi-task learning through launching multiple threads to simultaneously explore different environments, and update model parameters asynchronously.

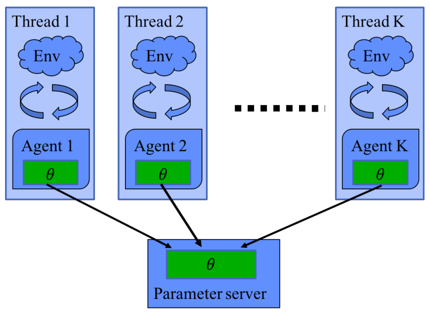
In the experiments, we train an agent to simultaneously play a set of Atari games through multi-task asynchronous actor advantage critic(M-A3C). To investigate what level of abstract knowledge can be shared among Atari games, we also experiment with different network parameter sharing strategies. We find out that the proposed asynchronous multi-task training is able to compress the policy and value networks for the agent through very aggressive parameter sharing. This indicates that there indeed exists a wide range of transferable knowledge among playing Atari games. Moreover, compared to training the agent with A3C, we observe significant performance improvement on several games by M-A3C. This result strongly supports the benefits of multi-task learning in RL, and inspires future research.

1.1 Related Works

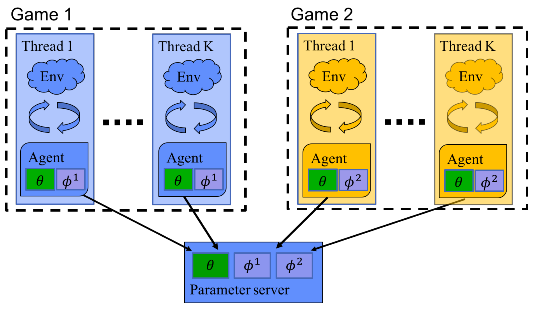
Transfer Learning

Transferring knowledge learned by tasks with abundant data and computational resources to new tasks is critical to an intelligent agent that can learn and evolve over time. Knowledge distillation [6] is a model compression technique that allows for transfer of knowledge from an ensemble of ”teacher” networks to a ”student” network that is smaller in size(illustrated in Fig. 1 right). In the realm of reinforcement learning, Actor-Mimic [1] is one such transfer learning scheme that distills knowledge from several pre-trained expert DQNs into a generic Actor-Mimic network that achieves comparable results to expert networks in a series of Atari games. Our method is different in that we intend to train multiple tasks simultaneously from scratch asynchronously, so that no computational resources used for training expert networks would be wasted.

Progressive Neural Networks [2] is another multitask and transfer learning scheme that, instead of transferring knowledge by finetuning on top of a pretrained model, learns the new tasks with a randomly initialized network that leverage prior knowledge via lateral connections to previously learned features(illustrated in Fig. 1 left). Although the model is immune to forgetting and makes one step forward life-long learning of the intelligent agent, the model is not scalable as the complexity goes up linearly with the number of tasks. Our model, in contrast, will focus on exploiting the similarity between source tasks and target tasks and aim at learning a succinct model that can easily generalize to a new task.



(a) Asynchronous RL.



(b) Multi-task asynchronous RL.

Figure 2: Illustration of asynchronous RL(left) & multi-task asynchronous RL(right).

Asynchronous Deep RL

Asynchronous deep reinforcement learning [7] takes advantage over parallel computation and achieved significant improvement over training on a single GPU both in terms of convergence speed and performance in a variety of Atari games. Leveraging the fact that DQN needs a stable target and uncorrelated gradient to stabilize training, asynchronous parallel update fits naturally into the framework of deep reinforcement learning and achieved remarkable improvements with little hurt.

Moreover, the parallelism of asynchronous deep RL also decorrelates the agents' data into a more stationary process. This enables a much larger spectrum of on-policy RL algorithms, such as Sarsa, n-step methods, and actor critic methods, as well as off-policy RL algorithms such as Q-learning, to be applied effectively and robustly using deep neural networks.

In our project, we will take advantage of asynchronous parallel methods to facilitate training and combat the heavy computational requirements by multitask and transfer learning.

2 Multi-task Asynchronous learning

Due to the asynchronous data collection mechanism, the modification needed to extend asynchronous deep RL for multi-task learning is rather straight-forward. In asynchronous deep RL(illustrated in Fig. 2a), agents explore the environment and aggregate data in multiple threads, and the model parameter θ is updated asynchronously. To adapt for multi-task training, we do the following modification:

- First, in order to enforce the agents to learn shared knowledge across different tasks, we simply share the part of network weights for each agents, while keeping the rest (including the output layer) to remain game-specific. Mathematically, the policy, value network for game k now becomes:
$$\pi_k(s_t; \theta, \phi_\pi^k), \quad V_k(s_t; \theta, \phi_v^k),$$
where parameter vector θ is shared across different games, and ϕ_π^k, ϕ_v^k are output layer parameters specific to game k. These kind of parameter factorization is in fact common practice in multi-task learning, and pre-train & fine-tune styled deep learning. For example, in computer vision, deep convolution feature pre-trained for image classification is directly used for other tasks like segmentation, detection, etc. Moreover, it can also be understood as learning a feature representation which enables the agent using simple linear dynamics. This is potentially useful since linear controller has been systematically studied and has good optimization properties.
In experiments(see Section 3), we investigate different level of weight sharing for actor advantage critic network with LSTM(see Fig. 3). We try two scenarios: share the CNN feature extraction module(see Fig. 3 left), and share CNN module together with the LSTM module(see Fig. 3 right).

Algorithm 1 Multi-task asynchronous advantage actor critic for each actor-learner thread

```
1: //Assume parameter vector $\theta$ is shared across different games.
2: //Assign game $k$ to the current thread.
3: //Assume $\phi^k_\pi, \phi^k_v$ are output layer parameters of policy and value networks for game $k$.
4: //Assume $\tilde{\phi}^k_\pi, \tilde{\phi}^k_v, \tilde{\theta}$ are thread-specific parameters, and $\phi^k_\pi, \phi^k_v, \theta$ are globally shared.
5: Initialize thread step counter $t \leftarrow 1$
6: **repeat**
7:     Reset gradients: $d\theta \leftarrow 0$, $d\phi^k_\pi \leftarrow 0$ and $d\phi^k_v \leftarrow 0$
8:     Synchronize thread-specific parameters $= \theta$, $\tilde{\phi}^k_\pi = \phi^k_\pi$ and $\tilde{\phi}^k_v = \phi^k_v$
9:     $t_{start} = t$ Get state $s_t$
10:    **repeat**
11:        Perform $a_t$ according to policy $\pi_k(a_t|s_t; \tilde{\theta}, \phi^k_v)$
12:        Receive reward $r_t$ and new state $s_{t+1}$
13:        $t \leftarrow t + 1$
14:        $T \leftarrow T + 1$
15:    **until** terminal $s_t$ or $t - t_{start} == t_{max}$
16:    **if** $s_t$ is terminal **then**
17:        $R = 0$
18:    **else**
19:        $R = V(s_t; \tilde{\theta}, \tilde{\phi}^k_v)$
20:    **end if**
21:    **for** $i \in t - 1, ..., t_{start}$ **do**
22:        $R \leftarrow r_i + \gamma R$
23:        Accmulate gradients wrt $\tilde{\theta}$: $d\theta \leftarrow d\theta + \nabla_{\tilde{\theta}} \log \pi_k(a_i|s_i; \tilde{\theta}, \tilde{\phi}^k_\pi)(R - V(s_i; \tilde{\theta}, \tilde{\phi}^k_v))$
24:        Accmulate gradients wrt $\tilde{\phi}^k_\pi$: $d\phi^k_\pi \leftarrow d\phi^k_\pi + \nabla_{\tilde{\phi}^k_\pi} \log \pi_k(a_i|s_i; \tilde{\theta}, \tilde{\phi}^k_\pi)(R - V(s_i; \tilde{\theta}, \tilde{\phi}^k_v))$
25:        Accmulate gradients wrt $\tilde{\phi}^k_\pi$: $d\phi^k_\pi \leftarrow d\phi^k_\pi + \partial(R - V(s_i; \tilde{\theta}, \tilde{\phi}^k_v))^2/\partial\tilde{\phi}^k_v$
26:    **end for**
27:    Perform asynchronous update of parameters $\phi^k_\pi, \phi^k_v, \theta$
28: **until** $T > T_{max}$
```

- Next, to simultaneously train the agent play multiple games, we divide a number of threads into groups. Agents within the same group of threads play the same type of game, and thus share the same model parameters, while agents from different groups share part of the parameters θ, but differs in ϕ^k_π, ϕ^k_v as they're trained to play different games. In our experiments, we assign equal number of threads to each game, but better thread allocation methods can be explored.

- Finally, as illustrated in algorithm 1 and figure 2b, gradients with respect to model parameters are accumulated asynchronously in each thread, and the model is updated asynchronously every fixed number of steps. In experiment, we find that the asynchronous model update can steadily improve the performance for all the games which are trained jointly, and have the same if not faster convergence speed compared to training separately for each Atrai games.

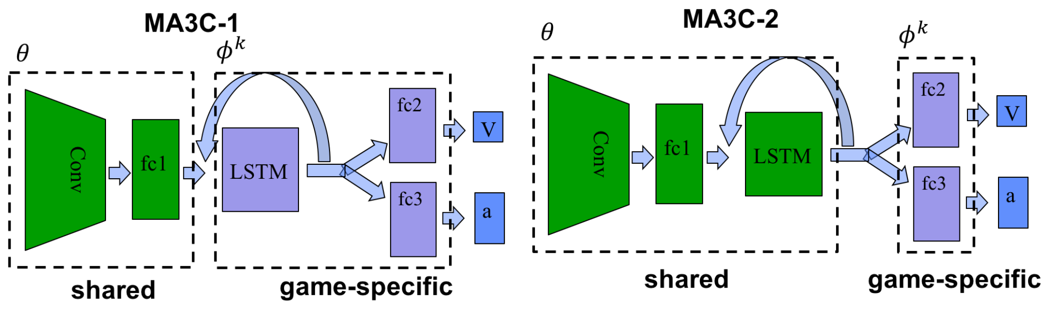


Figure 3: Network Architectures for Multi-task Asynchronous Advantage Actor Critic. left, MA3C-1, which shares the feature extraction module; right, MA3C-2, in addition to feature extraction module, LSTM module is also shared.

3 Experiments

3.1 Network Architecture

We first implemented asynchronous advantage actor critic method proposed in [7] as a baseline for our experiments. On top of that, we implemented two versions of network architecutre for our Multi-task synchronous advantage actor critic (MA3C) algorithm, as illustrated in figure 3. In the first architecture (MA3C-1), convolutional and fully connected hidden layers before the Long-short term memory cells are shared across games, while the LSTM and the fully connected hidden layers after that are game specific. In the second architecture (MA3C-2), only the fully connected hidden layers after the LSTM cells are game specific. We wanted to investigate how different portions of shared parameters influence the performance of MA3C. We expected that MA3C-1 would outperform MA3C-2 in terms of final convergence score, as intuitively different games would have difference state update mechanisms, and sharing the LSTM cells would hurt the performance.

3.2 Game Selection

Due to limited computational resources, we cherry-picked three pairs of games as our testbed for MA3C:

1. Pong and Breakout;
2. Space-Invaders and Phoenix,
3. Pong and Space-Invaders

The first two pairs of games are conceptually similar, i.e. both Pong and Breakout requires a paddle hitting a moving object, and both Space-Invaders and Phoenix requires accurate shooting of moving objects. On the contrary, the last pair is composed of two conceptually dissimilar games. We want to investigate whether (and how) similarity between games will affect the quality of knowledge transfer.

3.3 Experiment setup

We compare the average episode score reached by MA3C-1 and MA3C-2 with that reached by the baseline (standard A3C). To make a fair comparison, we use same number of threads for each game in either MA3C or A3C. More concretely, we assign 16 threads for MA3C training, and assign half of the threads to each of the two games. The baseline A3C is trained with 8 threads for a single game.

3.4 Results

Our experiment result shows that MA3C gains significant improvement over Breakout and Phoenix when trained with conceptually similar games. As shown in figure 4, both MA3C algorithms reach two times as much score as baseline A3C for breakout and phoenix. (Note that the number of iterations for MA3C is counted for both games, so the actual number of each game should be divided by two to match with the number of iterations in A3C.)

Another interesting phenomenon is that, though suspected to perform worse than MA3C-1, MA3C-2 actually managed to converges to a very similar score to MA3C-1 for all the games, despite experiencing slower convergence at the beginning. This might imply that much of the critical features for playing the game can be extracted by the same set of filters, and only the last controlling parts need to be modified to fit the traits of a specific game. In other words, models capable of playing multiple games can be significantly compressed. Different than actor-mimic algorithm where compression is achieved via distilling knowledge from pre-trained teacher networks to a versatile student network, our experiments shows that similar results can be obtained by directly training a network for multiple games, evading the time and computationally expensive process of training teachers and knowledge distilling.

4 Discussion

In our multi-task asynchronous learning approach, we do not assume access to the teacher policies which have been fully learned in their respective source domains, and this is the major difference between our method with knowledge distilling. To fully support the argument that there exists shared knowledge between different games, and multi-task training is sufficient enough to capture those knowledge, and enable better performance, we experiment on Atari games, and compared with A3C which does not do transfer learning. We show that aggressive weight sharing(MA3C-2) and multi-task asynchronous training can achieve equal if not better performance compared to learning separate agents for each game.

In the future, we'd like to give a more concrete empirical comparison with other approaches, e.g., does transfer from fully learned teachers [3], and from progressive neural network [2]. To compare those, we'd like to do the following experiments:

Simultaneous learning to play N games

Due to the restriction of computation resource and time, we only performed multi-task training with two games. In the future, we'd like to joint train more games to see the full potential of our asynchronous multi-task learning. In this experiment, we'd like to compare the speed of convergence between training the agents separately, versus training them with multi-task asynchronous learning proposed in Sec.2.

Learning to play novel games

After learning a model by simultaneously interacted with multiple source environments, we'd like to see if the learned knowledge from source tasks is applicable to a novel target task. To evaluate this, we apply the learned network on the held-out set of Atari games to see if by fine-tuning just a subset of the network parameters,can the agent learn faster, or play the game better.

In addition, we have to compare against policy distilling [3] as it's another popular transfer learning approach which is applicable for reinforcement learning.

References

[1] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

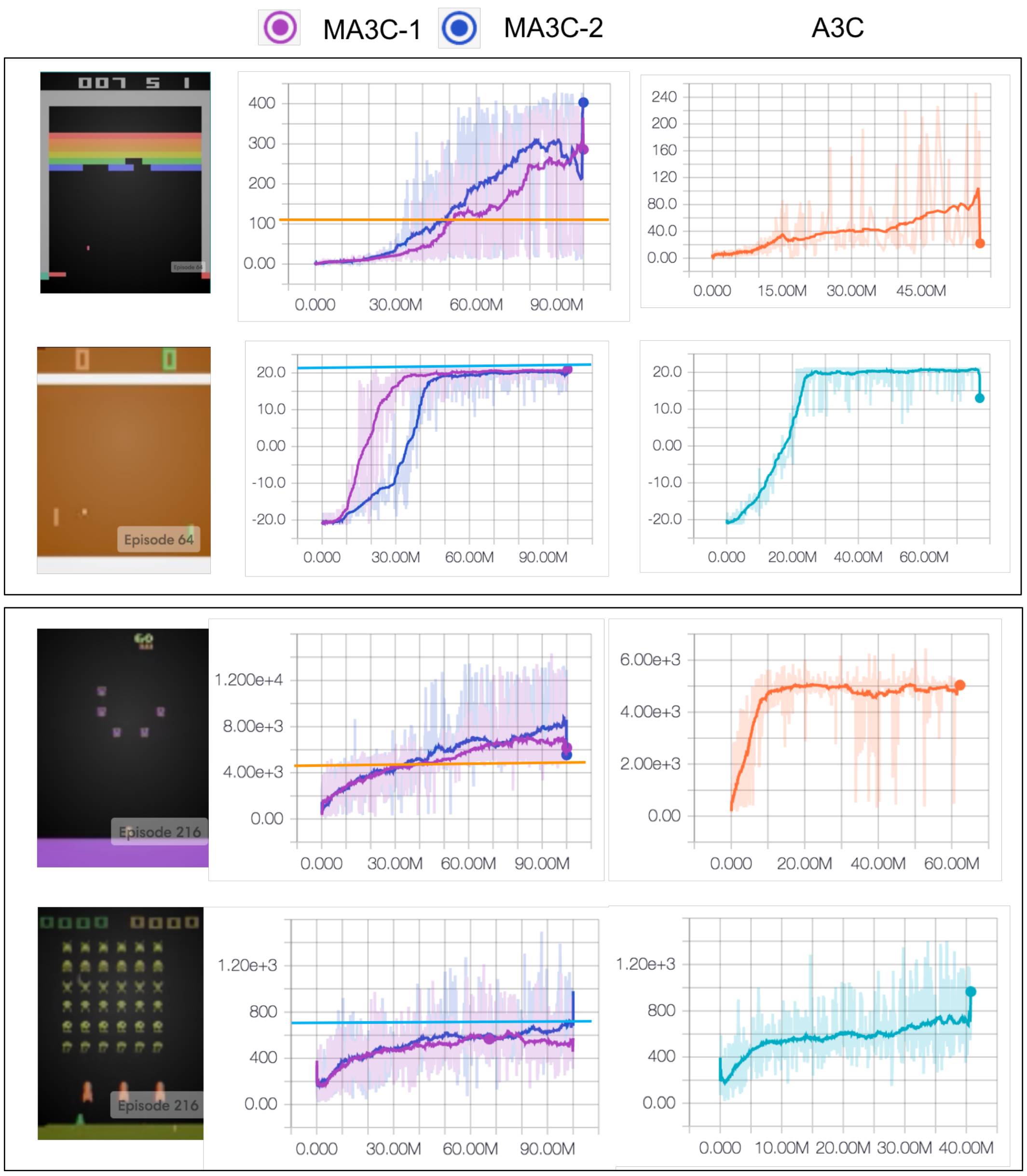


Figure 4: MA3C training curve on two conceptually similar games. Top row: co-training Breakout with Pong. Both MA3C-2 and MA3C-1 achieves significantly higher score on Breakout compared to A3C, and achieves the highest possible score on Pong; Bottom row: co-training Phoenix and SpaceInvaders. MA3C-1 and MA3C-2 gets higher score on Phoenix, and comparable performance on SpaceInvaders, in comparison to A3C.

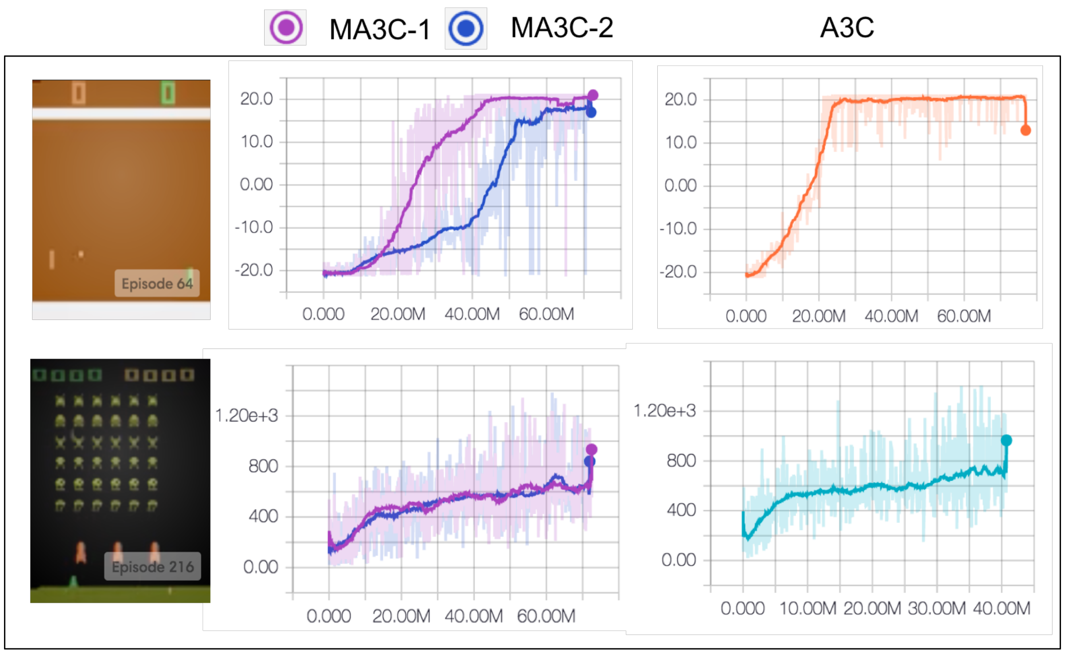


Figure 5: MA3C training curve on two conceptually dissimilar games. MA3C gets similar performance as A3C when co-training on Pong and SpaceInvaders.

[2] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[3] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[4] Asier Mujika. Multi-task learning with deep model based reinforcement learning. *arXiv preprint arXiv:1611.01457*, 2016.

[5] Joshua Romoff, Emmanuel Bengio, and J Pineau. Deep conditional multi-task learning in atari. 2016.

[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.