Multi-Model based Actor-Critic

Haoran Wei
Department of Computer and Information Sciences
University of Delaware
nancywhr@udel.edu

Keith Decker
Department of Computer and Information Sciences
University of Delaware
decker@udel.edu

Abstract

Reinforcement Learning (RL) is useful for solving complex sequential decision-making problems, but the convergence typically requires a very large number of environment interactions. The learning efficiency problem is worsened in model-free RL methods. When a high-fidelity simulation is available, speeding up single agent learning by parallel cooperative agents is a promising solution, such as the synchronous Actor-Critic method. When the environment model is not known, a predictive model can be trained as an environment simulator. However, the model mismatch may cause the RL policy trained from such a model to be highly biased. In this paper, we use a multi-model system to simulate an unknown environment and generate synthetic data for a master RL agent. This work contributes to domains where no high-quality environment simulator is available and the real-environment interactions are costly (e.g., finance problems). To overcome model mismatch, we ensemble multiple environment models to blend biased synthetic data. We experimentally compare our proposed method with the vanilla Actor-Critic where the environment model is available. Results show that our proposed method reaches the asymptotic learning bound with fewer true, non-simulated environmental interactions.

1 Introduction

Reinforcement Learning (RL) is a very natural way to solve a sequential decision-making task, unfortunately, it suffers from dealing with large scale real-world tasks with a large number of states. Many works have focussed on scaling up RL, such as balancing exploration and exploitation[Mann and Choe, 2011] or parallelization with a decomposed state space[Wei *et al.*, 2018]. In recent years, combining neural networks with RL (DRL) has received many breakthroughs in many challenging domains such as game playing [Mnih *et al.*, 2015] and robotic control [Levine *et al.*, 2016]. Based on ideas from both Dyna-Q and Advantage Actor-Critic approaches, we explore an approach to use multi-model parallel off-line learning, backing an on-line model-free learner, to improve the online RL agent's learning efficiency in this study.

One widely-used DRL category is model-free RL (MF-RL) that learns a policy from data samples in a trial-and-error fashion without building an explicit environment model. The biggest obstacle to scaling up such an approach is its low sampling efficiency, especially in large-scale domains. Parallel agents can be used to improve data-sampling efficiency by simultaneously exploring in an environment simulator and aggregating their learning results via a centralized controller (e.g., Asynchronous Actor-Critic). Another solution is model-based RL (MB-RL) where the environment model is built iteratively and the policy is obtained through planning based on the environment model (e.g., Monte Carlo Tree Search) [Guo *et al.*, 2014]. The model-based methods increase sampling efficiency significantly with the help of the environment model, but an imperfect model that doesn't accurately match the real world can mislead the RL policy and import biases.

In this study, we introduce an idea to combine model-free and model-based methods where an online RL agent learns the policy with a model-free approach and multiple models are built as background simulators to provide synthetic data backing to the online RL agent. We call the online RL agent the "master learner" and the multiple environment models as "assistant models" or "assistant system." There could be more than one master learner, and we explore this in the experiment part. The master learner uses Actor-Critic as its learning method due to its advantage in reducing the policy variance by combining policy-based and value-based methods, and it's ability to surpass the state of the art on the Atari domain. This work also can be combined with Asynchronous Actor-Critic (A3C)[Mnih *et al.*, 2016] framework where all agents are master learners and they update the with assistant system's synthetic data. Assistant models predict the next state from the given current state and action and they build from scratch with randomly sampled data from the environment. Assistant models also keep iteratively updating every N steps with the master learner's sampled data to be close to the real environment. An ensemble method is used to aggregate the synthetic data generated from all assistant models to reduce the bias caused by imperfect model prediction. We use the asynchronous advantage actor-critic algorithm [Mnih *et al.*, 2016] as the baseline where the environment is assumed to

be known, and all agents learn with their environment copies. Due to limited resources, we implement the proposed method on the OpenAI Gym Cart-Pole and Lunar-Lander domains. Even though the OpenAI gym environments themselves are simulators, we want to show the potential of scaling up the RL in real physical environments where the interactions are expensive, and the number of learning agents are limited. Overall, our contributions include 1) a new approach to combine a multi-model system with a model-free on-line RL agent; 2) a method to increase RL efficiency and shed some light on scaling up RL in an unknown environment.

2 Related Work

MF-RL learns a policy from domain rewards and bootstrapping the future estimated utility. As well known, MF-RL eventually outperforms in finding an optimal policy. However, MF approaches usually require a large number of sample data (e.g., Deep Q-Networks (DQN) [Mnih *et al.*, 2015]) so that they are impractical in large scale environments, sparse reward scenarios, or when environmental interactions are expensive. On the other hand, Model-based(MB) approaches can use samples efficiently because they model the environment explicitly so that more explorations can be carried out through the model and agents have longer on action selections [Guo *et al.*, 2014; Deisenroth *et al.*, 2015; 2011]. Recent studies have explored more possibilities towards general applications with model-based RL algorithms [Depeweg *et al.*, 2016; Mishra *et al.*, 2017]. However, insufficient training data limits most works due to the model training on a fixed dataset beforehand. Besides, a biased environment model might then cause a strongly biased policy. State of the art approaches that integrate MF and MB RL [Weber *et al.*, 2017; Peng *et al.*, 2018; Bansal *et al.*, 2017; Pong *et al.*, 2018] provide a promising way to improve RL learning efficiency by leveraging each side's advantages and overcoming the challenges of each approach.

The work of Dyna-Q [Sutton, 1990] inspires our work, a classic method combining model-free learning and model-based planning where each Q-learning step in the real environment is followed by N steps planning on the simulated model to expand the policy update data samples. Lately, Deep Dyna-Q [Peng *et al.*, 2018] has been developed to scale-up Dyna-Q's application domains with neural networks and also addressed the benefit of having a simulation model. Using Dyna-Q methods to accelerate model-free RL is clearly beneficial when the learned simulation model perfectly matches the domain ground truth. However, an imperfect simulation model may degrade the policy's stability and optimality rapidly [Gu *et al.*, 2016]. Instead of learning the domain model with a fixed sampling dataset, in our study the models are continually updated with new real environment sample data. Similar multi-model learning can be seen in more general AI studies including natural language processing and image processing [Baltrušaitis *et al.*, 2019].

Our work is also similar to work [Foerster *et al.*, 2017; Kurutach *et al.*, 2018] where ensemble methods are used to blend the prediction biases from imperfectly trained environment models. Differently, our study merges the ensembled multi-model learning with MF-RL. The master MF-RL agent suggests the assistant models' exploration and also aggregates the exploring trajectories. Another related work [Lowe *et al.*, 2017] proposes a decentralized version of multiagent Actor-Critic (AC) for continuous tasks. For the same reason, we believe actor-critic has better learning stability compared to value-based methods (e.g., Q-learning) because it associates the policy update with one-step state utility value. We also implement Actor-Critic for the master learner.

3 Preliminaries

This paper sets the environment as a discrete-time, finite-horizon Markov Decision Process (MDP). An MDP can be defined as $\langle \mathcal{S}, \mathcal{A}, r, \rho_0, \mathcal{P} \rangle$, where $\mathcal{S} \in \mathbb{R}^n$ is the state space, n is the dimension of each state. $\mathcal{A}$ is the discrete action space. r is the reward function: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow r \in \mathbb{R}$. ρ_0 is the initial state distribution. $\mathcal{P}$ is the transition model, mapping state-action pairs to a probability distribution across next states: $\mathcal{P}(s'|s,a) \rightarrow [0,1]$. We denote a stochastic policy $\pi_\theta(a|s)$ as the probability of taking action a in state s, where θ is the policy's parameter. The total discounted reward following a policy π within a certain time horizon is $\eta^\pi(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)]$, where γ is the discount factor and T is the time horizon. $\mathbb{E}_\tau[\cdots]$ indicates the empirical expectation of rewards within a certain time horizon over a finite batch of samples. Our goal is to find the optimal policy π^* by maximizing the total reward η^π. In this study, we assume the $\mathcal{S}$, $\mathcal{A}$ and reward function are known, and the transition function $\mathcal{P}$ is unknown.

Our work targets domains where there is no high-fidelity simulator is available and the reward function is assumed to be known in this work. Each assistant model is a prediction model, $f(s,a)$, that predicts the next state based on a given previous state and action, built with supervised learning. Shuffled random trajectory samples are used to differentiate the multiple assistant models. The master learner uses Actor-Critic as the learning method which updates the RL policy directly via the policy network. In this section, we provide a brief overview of the model-learning method and Actor-Critic methods.

3.1 Model Learning

In small-scale environments with discrete state and action spaces, the transition model can be learned with the occurrences of transitions [Hwang *et al.*, 2015]:

$$f(s'|s,a) = \frac{C(s,a,s')}{C(s,a)} \tag{1}$$

where $C(s,a,s')$ is the count of transition (s,a,s'). The severe drawback is that it relies on a large number of transition samples from the real environment and requires large space to memorize directly. Instead, the transition model can be represented by a feed-forward neural network. The model predicts either the probability of change to a state or predicts the next state directly [Kurutach *et al.*, 2018; Nagabandi *et al.*, 2018]. The environment model is optimized by minimizing the average absolute squared error of $(\varepsilon(\phi))$,

shown as:

$$\varepsilon(\phi) = \frac{1}{|\mathcal{D}|} \sum_{(s,a,s') \in \mathcal{D}} \frac{1}{2} \left\| s' - f_\phi(s,a) \right\|^2 \quad (2)$$

where $\mathcal{D}$ is the training dataset of state-action-state transitions from the real environment. Optionally, the prediction loss also can include the loss of reward prediction and the terminal state prediction.

With MB-RL, when the environment model is available, a sequence of actions can be planned based on the model. The simplest but very popular method is called "random shooting" in which a randomly sampled action is taken at each step following a certain distribution, such as uniform distribution. More advanced methods include Monte Carlo Tree Search (MCTS) [Chaslot *et al.*, 2008], which selects an action that hasn't been visited yet or results in the maximal estimated value at each time step. The environment model also can be updated iteratively with new sample data to move closer to the real environment. The vanilla MB-Learning algorithm is summarized in Algorithm 1.

Algorithm 1 vanilla Model-Based (MB) Learning

Require:
  Initialize predictive models f with random weights
  Initialize a base policy π_0
  Initialize an empty sampling dataset $\mathcal{D}$
  repeat
    collect random sample set $\mathcal{D}$ with π_0
    update model $f(s,a)$ with Eq 2
    for step in N_STEPS **do**
      plan through $f(s,a)$ to get actions
      execute the first action, observe transition (s,a,s')
      add (s,a,s') to $\mathcal{D}$
    end for
  until the performance stops improving

3.2 Actor-Critic

Policy gradient (PG) is a category of model-free method, and it parameterizes the policy as a function represented by a neural network. The goal is to maximize the discounted total reward, $\eta^\pi(\theta) = \sum_{t=0}^{T} \gamma^t r_t$, by doing gradient ascent on the policy parameters, represented as $\nabla_\theta \eta^\pi(\theta)$. Compared to value-based methods (e.g., Q-learning), PG updates the policy directly and avoids some drawbacks of value-based methods, such as value over-estimation, etc. One well-known PG method is REINFORCE [Williams, 1992] that uses the trajectory total reward R in the policy update objective function: $\eta^\pi = \mathbb{E}[\log \pi(\tau) R]$, where τ is a trajectory following the current policy and $\pi(\tau) = \rho_0(s_0) \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)$. $\mathbb{E}(\cdot)$ is the mean operation based on a mini batch of trajectories. With a good overall reward, all actions along the trajectory are assumed to be good, so that it ignores the adverse actions and results in higher variance.

Actor-Critic (AC) [Mnih *et al.*, 2016] has an additional value network besides the policy network compared to the vanilla PG methods, and the value network can help reduce the policy variance. The value network is called the "critic," and we use w to represent its parameters. The critic approximates the Q value for each input state and action pair. The policy network is called the "actor," and is parameterized by θ. In this work, we implement these two networks with shared fully connected layers but separated output layers. The actor updates by maximizing the expected total rewards over a batch of trajectories:

$$\eta^\pi(\theta) = \mathbb{E}[\log \pi(s,a;\theta) Q(s,a;w)] \quad (3)$$

Meanwhile, the critic network optimizes with supervised learning, minimizing the mean square error on the Q-value prediction:

$$e = \mathbb{E}[(y - Q(s,a;w))^2] \quad (4)$$

where y is the target value, and it estimates the Q-value for current state and action with the immediate reward as well as bootstrapping the best estimated Q-value from next states:

$$y = r + \max_{a' \in \mathcal{A}} Q(s', a'; w) \quad (5)$$

Compared to the regular PG methods, where using the trajectory reward R may ignore some adverse actions once the overall summation of reward is high, the critic measures the policy performance at every single step with Q-value so that the probability of taking a bad-performing action is reduced and policy variance is lowered. To reduce the variance even more, an advantage value ($A(s,a)$) is introduced that is the difference between the Q-value of a state-action pair and the utility of the same state ($V(s)$), representing how much better an action can do in a given state compared to the average performance of all actions. We use the mean value of $Q(s,a)$ over all actions to represent the $V(s)$. We use the advantage value to update the actor-network in this study, indicated as:

$$\eta^\pi(\theta) = \mathbb{E}[\log \pi(s,a;\theta) A(s,a;w)] \quad (6)$$

The AC method also can be executed in a set of parallel environments (e.g. Asynchronous Actor-Critic (A3C)) with their environment copies. Agents explore asynchronously, and they are very likely to explore different parts of the environment each time. A3C improves performance by increasing the diversity of training data, and no experience replay is needed. Notably, a good environment simulation model is required to make copies for multiple RL agents in this setting.

4 Multiagent Model-based Actor-Critic (MMAC)

We propose a framework, **M**ulti-**M**odel based **A**ctor-**C**ritic (MMAC), to increase RL agent learning efficiency and scale up RL application to domains where no high-fidelity simulator is available. This framework comprises two parts: master learner(s) and assistant models. The master leaner is an online RL agent as well as a centralized aggregator for assistant models. The assistant models learn the environment model in parallel and explore with master learner's exploration suggestion. Each assistant model predicts the next state based on the input state and action and updates with real environment data samples provided by the master learner. With a predicted state, a "done" status is decided with some prior knowledge of

Algorithm 2 Multiagent Model-based Actor-Critic (MMAC)

Require:
A base(random) exploration policy π_0
Initialize the master learner with random weight θ
Initialize M assistant models $\{f_i\}_{i=1}^{M}$ with random weights
Initialize a dataset $\mathcal{D}^p$
for epoch in NUM_EPOCH **do**
  collect random sample with π_0 and add to $\mathcal{D}^p$
  for $i = 1, \cdots, M$ **do**
    random sample a mini batch $\mathcal{B}$ from $\mathcal{D}^p$
    update model f_i with $\mathcal{B}$ as Eq 2
  end for
  if epoch $< \mathcal{H}$ **then**
    for for i in M **do**
      generate trajectories τ^i from model f_i
    end for
    update π and value function $Q(s, a)$ as Eq 7
  end if
  sample trajectories from environment and add to $\mathcal{D}^p$
  update π and value function $Q(s, a)$ as Eq 3, 4
end for

the environment (done status represents if a task has been finished or if the agent has failed. If done is True, an initial state is selected to start over at next time step). Take the OpenAI cartpole task as an example, if the predicted angle between the pole and the cart is within a small degree threshold, the pole has fallen so that the task has been done, here the degree threshold is the prior knowledge, and the angle between the pole and the cart is part of the prediction. An assistant model generates synthetic trajectories by iteratively selecting an action and predict the next state. The action selection follows the master learner's current policy with a certain probability, otherwise, randomly select one. The synthetic data are backed to update the master learner. This is a win-win work cycle between the master agent and the assistant models, and we depict it in the Fig 3. The entire working flow is demonstrated in Figure 1. The way that the assistant model system works is very similar to the state of art work[Kurutach *et al.*, 2018]. We use an ensemble method to balance the imperfect M assistant models' biases, by averaging their sample trajectories cumulative rewards, Eq 7. The ensemble method serves as an effective regularization for policy learning stability.

$$\eta^{\pi} = \frac{1}{M} \sum_{i \in M} \mathbb{E}_{\tau_i}[\log \pi(s, a) Q(s, a)] \tag{7}$$

The proposed algorithm is summarized as Alg 2. The master learner(s) explores independently from the multi-model system and update with both aggregated data from assistant models and real environment sampled data. The M assistant models $\{f_i(s, a)\}_{i=1}^{M}$ start learning with random samples drew from the dataset $\mathcal{D}^p$ where all data are collected with the random exploration policy π_0. Assistant models differentiate with randomly sampled training data and update with minimizing the difference between the model's prediction and the ground truth, shown as Eq 2. The assistants' synthetic trajectories that are used for master learner's updates are sampled with the master learner's current policy:

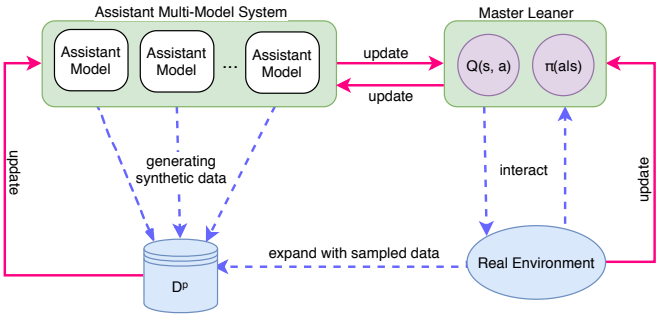


Figure 1: Multi-model Actor-Critic (MMAC) framework. Master agent interacts with the real environment and updates with both real environment data and assistant model's synthetic data. Assistant models are initialized with random samples and are optimized with the master learner's trajectory data.

Figure 2: Control Tasks. Left:LunarLander-v2 task. Right: CartPole-v1 task

$s_k \leftarrow f_i(s_{k-1}, \pi(a_{k-1}|s_{k-1})$ for $k = 1, \cdots, T$, where i is the index of assistant model and $\pi(\cdot)$ is the master learner's policy. The initial states for all assistant models are randomly picked in each training epoch. In this study, we use neural networks as the assistant models. Alternatively, linear Gaussian regression and Gaussian Process(GP) are also commonly used, and we'll compare them in future work. Besides, we also set a threshold $\mathcal{H}$ as the trigger to start using the assistant model for backing the synthetic data to the master learner. The assistant models are suspended in the beginning because we observe a negative influence from the severely imperfect trained assistant models and the insufficient random environment samples may cause it. This threshold also can be set according to the assistant model prediction errors, and we'll discuss this setting in our future work. Notice that the prediction models converge much faster than the policy and value networks, policy tends to exploit certain area repeatedly eventually, which is an overfitting issue. We use early stopping and value clipping to alleviate this issue.

To summary, our proposed framework uses decentralized model-learning execution to accelerate a centralized learning process and this idea has been used to speed up RL in other works [Clemente *et al.*, 2017; Wei *et al.*, 2018]. The benefit can be concluded as 1) parallelization is an efficient approach to scale up the RL with more varied information bootstrapped. 2) Model-learning broader the RL application domains to where the environment is not fully known.

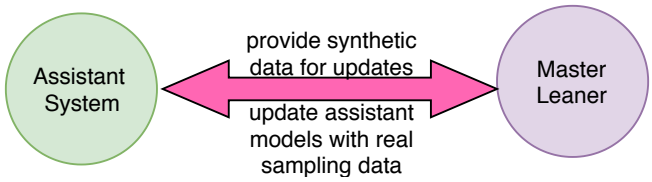


Figure 3: A Win-win interactive cycle: assistant system provides more update data to the master learner, and the master learner reciprocates real data for assistant models to update as well.

5 Experiment

A3C[Mnih *et al.*, 2016] algorithm is the baseline in our experiment ("baseline" and "A3C" are interchangeable in the following article) which has a similar parallel structure as our proposed work. In contrast, A3C works with a high-fidelity discrete time environment simulator and all parallel agents learn with their own environment copies. We expect that with the same total number of agents, the MMAC can provide asymptotic or even better solution compared to the A3C without an explicit environment model.

5.1 Experiment Tasks and Setup

We implement A3C and MMAC on two OpenAI Gym control tasks: Cart Pole and Lunar Lander, shown in Fig 2. With A3C, each agent has an OpenAI task model copy. In contrast, MMAC only observes the state and action without calling the transition model so that the environment remains uncertain. In both tasks, the state and the action space are well defined. After each step, there is reward feedback from the environment. The performance metric is the average of cumulative rewards along trajectories from the initial states until the task terminal states.

We have a grid search with single agent's AC on the hyper-parameters for AC network and the master learner(s) in MMAC and all agents in A3C share the same network setting: three linear connected layers with 64 neuron units each layer (128 for Lunar-Lander). A Relu layer is used at the end of each Linear layer. The policy output layer (Linear Layer) provides probability distribution of the whole action space, and the critic output layer provides the value for the input state and action. The assistant models are also three-layer neural networks (64 neurons each layer) and one extra output layer for the state prediction.

The assistant models update with supervised learning as Eq 2. The master learner updates by minimizing a weighted combination of three objective functions (shown as Eq 8): value prediction loss (L_v), action gain (L_a), and a policy cross-entropy (L_e). λ_1,λ_2 and λ_3 are the linear weights.

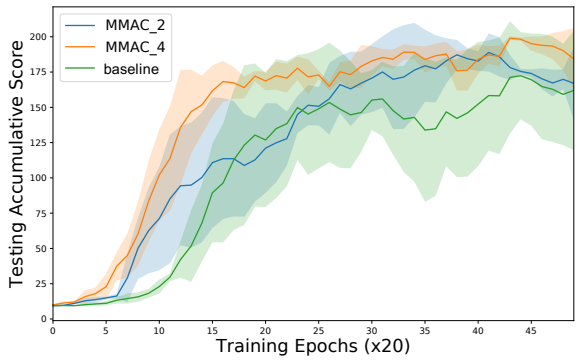
$$\mathcal{L} = \lambda_1 L_v - \lambda_2 L_a + \lambda_3 L_e$$

where

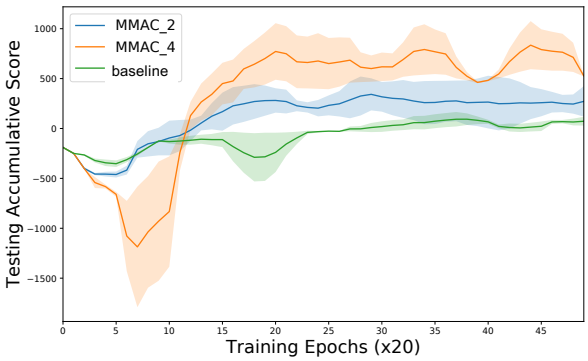
$$L_v = \mathbb{E}_\tau[(r + \max_{a' \in A} Q_w(s', a') - Q_w(s, a))^2] \quad (8)$$

$$L_a = \mathbb{E}_\tau[\log \pi_\theta(a|s) Q_w(s, a)]$$

$$L_e = \mathbb{E}_\tau[\log \pi_\theta(a|s) \pi_\theta(a|s)]$$



a. CartPole-v1 Task



b. LunarLander-v2 Task

Figure 4: An assistant model system improves the learning speed.

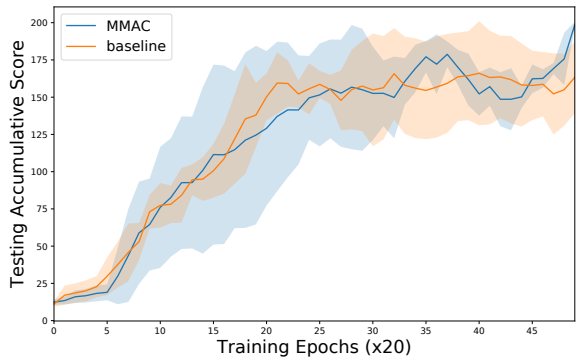
Where L_v is the critic value optimization objective function in Eq 3. L_a is the actor-network's loss function. L_e is the policy cross-entropy encourages the exploration. τ represents trajectories either from the real environment or generated by an assistant model. θ and w are the actor and critic network parameters respectively.

5.2 Experiment Analysis

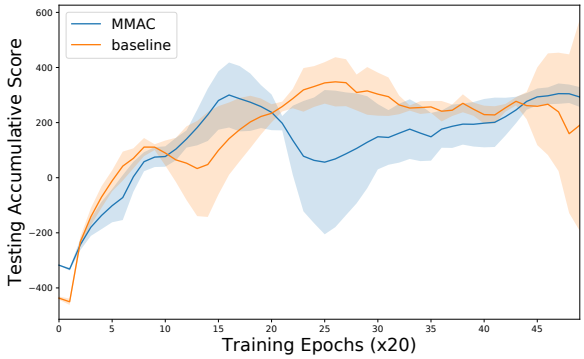
Through the experiments, we want to answer two questions:

1. Can an assistant multi-model system improve the master learners' learning efficiency?
2. Can the number of real-environment interactions be reduced by MMAC while still provide a similar learning performance compared to A3C?

To answer the first question, we compare the learning speed of one master learner MMAC with 0, 2 and 4 assistant models respectively and the baseline is one master agent with zero assistant models. We use "MMAC_2" to represents the MMAC experiment with 2 assistant models and "MMAC_4" with 4 assistant models. Three independent experiments are conducted under each method and 1000 training epochs in each experiment. The result is shown in Fig 4(a) with the task of CartPole task and Fig 4(b) with LunarLander task. The x-



a. CartPole-v1 Task



b. LunarLander-v2 Task

Figure 5: With the asymptotic performance, real-environment interactions is reduced by MMAC

axis represents the training iteration ($\times 20$ epochs), and the y-axis represents the corresponding cumulative rewards per trajectory. We set the threshold $\mathcal{H}$ as 100, meaning the assistant model are postponed from generating synthetic data until the 100th iterations. According to the learning curves, the one with 4 assistant models has a significant lift until convergence in both tasks. With 2 assistant models, the learning speed also has a noticeable increase in the first $\sim$ 300 iterations. In the LunarLander task, with the assistant model system, the policy quality is improved as well represented by the increased cumulative rewards. The shadow represents the standard deviation on the cumulative rewards, and it also reflects the policy variance. With the assistant system, the policy variance is reduced efficiently, especially inCartPole task, and we need to give credit to the ensemble method. In the LunarLander task, there is a big drop in the learning performance between the 100th iteration and 200th iteration. This drop is caused by the imperfect assistant model and the threshold $\mathcal{H}$ may need to be set higher in a complex environment.

The second question intends to show the ability of MMAC on scaling up the RL, especially in some circumstances where the number of RL agents are limited, or the environment-agent interactions are costly. We compare A3C with three parallel agents with the MMAC that includes one master learner and 2 assistant models. Even though the total number of agents is the same with both methods, the number of real-environment interactive agents are different. The result is shown in Fig 5. With a third of the number of real-environment interactions of A3C's, MMAC provides asymptotic performance.

6 Conclusion

In this study, we explored an RL scaling and acceleration method, Multi-model based Actor-Critic (MMAC). A decentralized, multi-modal system is used to provide synthetic data to an RL agent. Meanwhile, the RL agent's data can be used to optimize the environment models. MMAC encourages this win-win work cycle (shown in Fig 3) to speed up the learning process as well as to reduce the dependency on real environment interactions. This method extends RL to domains where the environment is unknown, and true environment interactions are limited. This approach also bridges multi-model learning with DRL. The two comparison experiments show that MMAC efficiently speeds up RL agents' learning with respect to cumulative rewards. Fewer interactive agents are needed with MMAC to reach asymptotic policy performance compared to A3C. This work also can be viewed as an adaptation of A3C for an unknown environment.

References

[Baltrušaitis *et al.*, 2019] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2):423–443, 2019.

[Bansal *et al.*, 2017] Somil Bansal, Roberto Calandra, Kurtland Chua, Sergey Levine, and Claire Tomlin. Mbmf: Model-based priors for model-free reinforcement learning. *arXiv preprint arXiv:1709.03153*, 2017.

[Chaslot *et al.*, 2008] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[Clemente *et al.*, 2017] Alfredo V Clemente, Humberto N Castejón, and Arjun Chandra. Efficient parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1705.04862*, 2017.

[Deisenroth *et al.*, 2011] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. 2011.

[Deisenroth *et al.*, 2015] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2015.

[Depeweg *et al.*, 2016] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

[Foerster *et al.*, 2017] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[Gu *et al.*, 2016] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[Guo *et al.*, 2014] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014.

[Hwang *et al.*, 2015] Kao-Shing Hwang, Wei-Cheng Jiang, and Yu-Jen Chen. Model learning and knowledge sharing for a multiagent system with dyna-q learning. *IEEE transactions on cybernetics*, 45(5):978–990, 2015.

[Kurutach *et al.*, 2018] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[Levine *et al.*, 2016] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[Lowe *et al.*, 2017] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[Mann and Choe, 2011] Timothy Arthur Mann and Yoonsuck Choe. Scaling up reinforcement learning through targeted exploration. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[Mishra *et al.*, 2017] Nikhil Mishra, Pieter Abbeel, and Igor Mordatch. Prediction and control with temporal segment models. *arXiv preprint arXiv:1703.04070*, 2017.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[Nagabandi *et al.*, 2018] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[Peng *et al.*, 2018] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2182–2192, 2018.

[Pong *et al.*, 2018] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[Sutton, 1990] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 1990.

[Weber *et al.*, 2017] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[Wei *et al.*, 2018] Haoran Wei, Kevin Corder, and Keith Decker. Q-learning acceleration via state-space partitioning. In *International Conference on Machine Learning on Applications*, 2018.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.