Multi-agent World Models

CSCI 7000 Deep Reinforcement Learning Final Project Report

Joe Miceli
joe.miceli@colorado.edu
University of Colorado Boulder

Mohammed Adib Oumer
mohammed.oumer@colorado.edu
University of Colorado Boulder

***Abstract*—Many systems in the real world are multi-agent in nature. Recent exciting advances in single agent reinforcement learning algorithms bring forth the question of their scalability to multi-agent systems. In this project, we attempted to build on one such single agent algorithm, *World Models* and test its performance on the multi-agent *Multiwalker* problem. Though we did not achieve all our goals before the submission deadline, we did make significant progress towards them. We justify the adaptations we made to the original approach and explain the implications they have on real-world applications of reinforcement learning. Finally, we discuss outstanding work remaining for the project and potential avenues for leveraging our work in future research projects.**

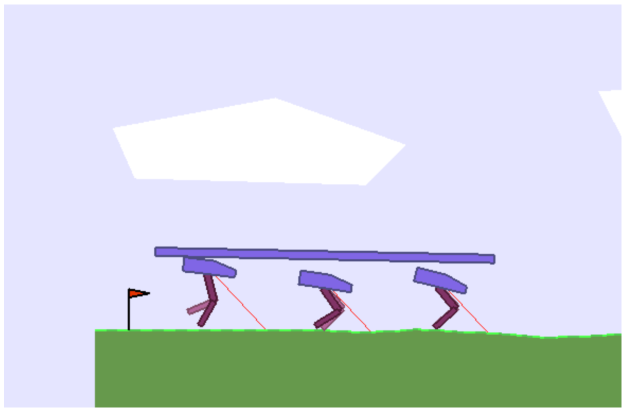


Figure 1. Rendering of Multiwalker Problem

I. Introduction

With advances in reinforcement learning (RL) for single agents comes questions regarding their applicability to multi-agent systems. Many of the most exciting applications of RL involve multi-agent systems. In fact, Google Scholar indexed over 9,400 papers regarding multi-agent reinforcement learning (MARL) in 2020 and at this point, MARL has become a ubiquitous term in the community [13]. MARL systems involve multiple agents either working collaboratively or adversarially to learn about themselves and the environment in which they operate. In this paper, we leverage the work in [8] in an attempt to teach a set of cooperative agents a model of the world. This model is then used to teach the agents how to interact in a manner that allows them to accomplish some task together. Unfortunately, due to time constraints guiding this report, we were not able to accomplish all the goals we initially outlined.

This project investigates the application of the World Models RL approach to the *Multiwalker* problem which was developed by the Stanford Intelligent Systems Laboratory (SISL) as part of the Farama Foundation's PettingZoo API [7], [13]. It is based on the *bipedal-walker* problem in OpenAI gym [3]. In the Multiwalker problem, a set of cooperative bipedal robots (the number of which is configurable) attempts to carry a single package placed on top of them as far to the right as possible. The agents have no understanding of their environment but are given detailed (yet noisy) state observations in the form of joint angles, LIDAR readings, two dimensional velocity, relative positions of neighbor walkers, relative position of the package, and angle of the package. Each agent can act independently by exerting force on its joints. See Figure 1 for a sample rendering.

Hierarchical solutions to the Multiwalker problem have been developed [12] and World Models has been applied to multi-agent systems [11]. However, to our knowledge, World Models has never been applied to the Multiwalker problem. Although not particularly complex in its observation space, the Multiwalker problem represents a different class of problem for the World Models algorithm. Originally, the algorithm was applied to a single agent with a complicated observation space that involved pixelated image frames from renderings of the environment. Instead of images, the Multiwalker problem uses sensor-provided data regarding the physical orientation, position, and velocity of the agents in the environment - a state description representative of many real world systems such as industrial robots or constellations of spacecraft.

For humans, learning frequently involves interaction with other humans, often yielding our greatest accomplishments. So, in mankind's pursuit for artificial intelligence, it is only logical that we seek to teach our artificial counterparts the same behavior. While the possible benefits of applying RL to multi-agent applications are large, so too is the computational complexity of doing so. Not only do these problems exponentially increase in complexity with states and actions, they also increase exponentially in the number of agents [4]. Therefore, solutions for multi-agent problems must not only be effective, they must also be computationally efficient. The World Models algorithm may be one of these solutions.

II. Background/Related Work

World Models uses a deep reinforcement learning setup that learns to create a simplified model of the environment through spatial and temporal "compression" with a variational autoencoder (VAE) and a mixture density recurrent neural network (MDN-RNN) [8]. The simplified model is then used in a single training iteration to process images to aid the agent learning in the actual environment. Our project extends the setup to the more realistic multi-agent scenario where the agents learn to maximize their cumulative reward working collaboratively without colliding and acting with minimal effect to the other agents. This might entail the use of iterative training procedures. It is relevant to note that in the original paper, the "agent" is defined as the combination of the VAE, MDN-RNN and controller. The VAE and MDN-RNN essentially act as the processing units of the agent's observations while the controller is the component that deals with the environment, actions, and rewards directly (so in this sense, the controller is more suited to the traditional agent definition in RL). The algorithm schematic of the World Models approach is shown in Figure 2.

As stated earlier, World Models has been applied to multi-agent problems before. SMAWM employs a reinforcement learning algorithm as the agent of World Models, and studies it in a multi-agent mixed cooperative-competitive setting [11]. Our project focuses on deploying multiple agents in a simplified environment model that seeks some form of cooperation among agents.

Additionally, the mutliwalker problem is not unsolved. FLE introduces a commander-agent hierarchical setup for coordinated MARL [12]. It uses a model-free policy gradient algorithm to optimize the policies for the agents and the latent command structures for the commander. Our project intends to use same-level (i.e. non-hierarchical) agents with the possibility of testing model-free approaches in the controller part of the agent.

III. Methods

A. Technical Contribution

The most appealing characteristic of the World Models algorithm is that it is intuitive. Its approach first compresses an observation into features that matter the most and then allows an RL agent to use those features to predict how its actions will impact the environment. This very much mirrors human behavior. And, because humans often deploy this kind of learning behavior in cooperative environments, it is our desire to understand how applicable this approach is to multi-agent problems. Our project will mainly investigate the scalability of the World Model algorithm to multi-agent systems. Through this work, we hoped to identify ways to make the algorithm scalable to a multi-agent environment of interest and the other was to identify and mitigate the key differences between single agent and multi-agent systems with respect to the algorithm. We specifically chose the multi-walker environment for this work because we believe it is representative of many real-world industrial applications.

B. Architecture

We build on the architecture used in World Models algorithm to work with multiple agents. In general, we tried to make as few changes to the original implementation as possible in order to test its supporting theory in a multi-agent domain. However, there are several

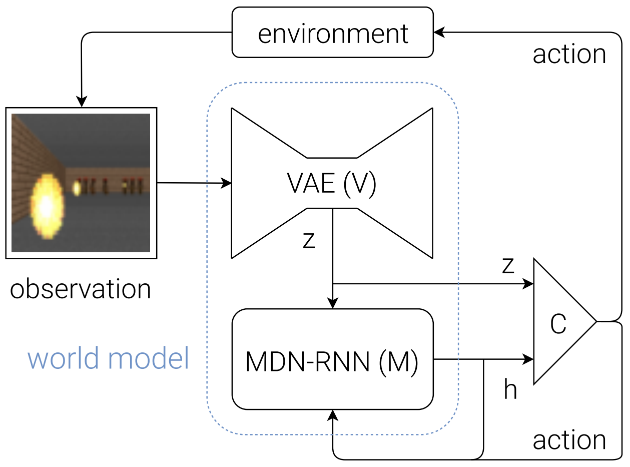


Figure 2. Schematic of the original World Models algorithm [8]

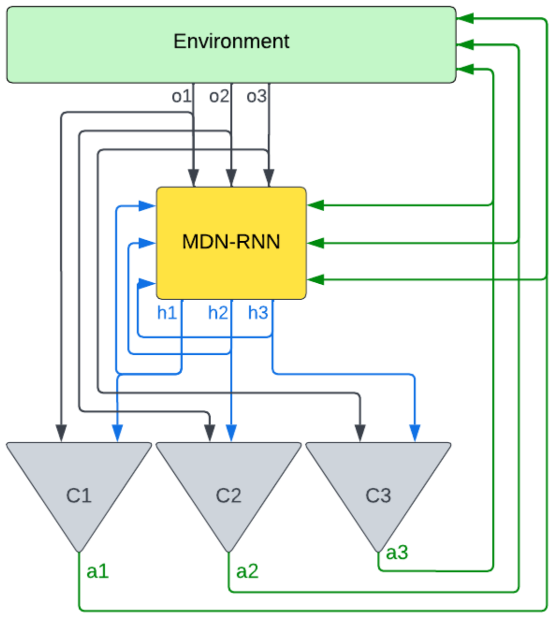


Figure 3. Schematic of our proposed simplified Multi-agent World Models algorithm

key differences between our implementation and the original implementation. The algorithm schematic of our implementation is shown in Figure 3.

In our approach, three (the default number for the Multiwalker problem) agents (i.e. controllers) each output an action to the environment. The environment responds by outputting three observations which are fed into the MDN-RNN and the controllers. The MDN-RNN takes each agent's action, observation, and previous hidden state to produce a new hidden state. The controllers used to determine each agent's action use the observation and MDN-RNN hidden state to do so. The hidden state was used since it has smaller number of parameters than the output distribution of the MDN-RNN and because the hidden state was shown empirically to result in comparable performances as the actual output of the MDN-RNN (i.e. hidden state was a sufficient substitute of the actual output predictions for the controller to use).

1) Updates to VAE: We do not make use of the VAE in our implementation since the observation space of the Multiwalker environment is not complex. To put it in perspective, the observation space in the original algorithm was an RGB image of dimensions 64x64x3. With the use of the VAE, they were able to compress this input observation into a latent vector of dimensions either $\mathbb{R}^{32}$ in their car race experiment or $\mathbb{R}^{64}$ in their doom game experiment. In the Multiwalker environment, the observation space is of dimension $\mathbb{R}^{31}$, which is not complex enough to warrant the use of the VAE. However, the action space for each agent in the Multiwalker problem is continuous and more complex. Therefore, in future experiments, we may need to reconfigure the VAE to deal with complex action spaces as opposed to complex observation spaces.

2) Updates to MDN-RNN: We use an LSTM RNN paired at its output with an MDN. The purpose of the LSTM is time-series data processing and prediction. The MDN helps in learning the distribution of the predictions instead of limiting them to deterministic target values. The mixed nature is a more realistic interpretation of assumptions on predicted values. Many problems have multiple modalities that cannot be solved by directly predicting target values so instead of assuming a single Gaussian distribution, we mix them up with different proportions. Thus, the MDN-RNN takes in observations and outputs a distribution for the predicted future observation. We then sample from this distribution to continue the training. Mathematically, the output distribution is given by the formula:

$$p(y|x) = \sum_k \pi_k(x)\mathcal{N}(y|\mu_k(x), \sigma_k^2(x))$$

where k identifies the number of Gaussians to mix, π is the mixing coefficient (summing to one), μ is the mean, σ is the standard deviation, and x and y stand for the input and predicted observations respectively. The loss function for the MDN-RNN is given by the negative log-likelihood formula as follows.

$$L(w) = (-1/N)\sum_{n=1}^{N} \log\{\sum_k \pi_k(x_n, w)\mathcal{N}(y_n|\mu_k(x_n, w), \sigma_k^2(x_n, w))\}$$

In the Multiwalker environment, each of the agents have their own observations depending on their position carrying the load. A straight forward approach to training the agents is then to collect training data for each agent from a random policy on the environment, and perform independent learning. We can then combine the results from each training, deploy them together and hope they perform well. This adds training complexity, lead to instabilities, and more critically, the agents are not coordinated, which is the opposite of our objective. Given this information, it is important to question how we could use our resource and time efficiently. The way we approached this is by implementing parameter sharing.

With parameter sharing, we will be able to train a single MDN-RNN to learn about all the agents. We re-organized the training data by adding one-hot-encoded agent ID before each observation for each agent, thereby augmenting our observation space from $\mathbb{R}^{31}$ to $\mathbb{R}^{34}$. This helps us to train a considerably less number of parameters and has been shown to be a sound, efficient method [5].

In implementation, we used five mixtures and the MDN is built using the output of the RNN (with 256 nodes) and produces a dense layer output (of dimension 3×number of mixtures (5)×augmented observation space(34) $= 510$). This layer is given by the equation:

$$Y = W_o \tanh(W_h X + b_h) + b_o$$

where X is the output from the RNN, Y is the dense layer vector, and W and b are the weights and biases respectively. The 510 length vector is divided into three parts, with the first one third part representing the π, the second one third part representing the μ and the last one third part representing the σ. We apply softmax to the π part (with some normalization to avoid float overflow) to keep the sum to 1, and exponentiate the σ part to ensure they are all positive. The number of trainable parameters at the end comes out to be 433,150.

3) Updates to Controller: Our controller still utilizes the linear combination approach that was used in the original paper. More specifically, the controller for each agent has the following structure.

$$a = W[z, h] + b$$

where z is the same observation that is input into the MDN-RNN and h is the hidden state of the MDN-RNN.

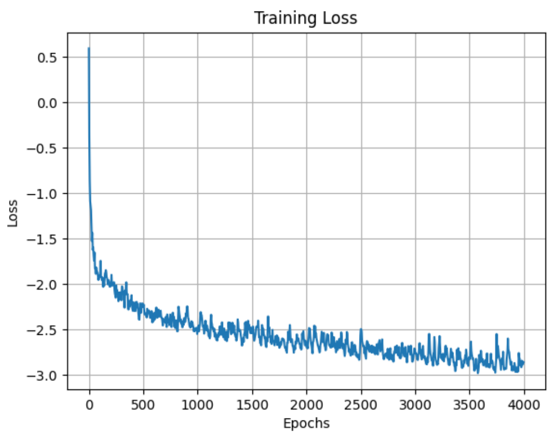
To find the weights and biases for each controller, we utilized the same optimization scheme discussed in the original paper - covariance matrix adaptation evolution strategy (CMA-ES). CMA-ES is a derivative-free covariance matrix-based method for continuous parameter optimization of non-linear, non-convex functions [9]. We attempted to use CMA-ES as opposed to traditional RL methods considering the benefits of ES and the fact that the parameter count is very small compared to the MDN-RNN. With CMA, there is no focus on the underlying objective function and rather we choose the best options in the optimization process. CMA-ES is also easier to parallelize and run over multiple CPU cores with multiple rollouts of the environment [10]. In our case, the "objective function" was a rollout simulation of the Multiwalker environment. A key observation to make in this implementation is that although we are trying to optimize three sets of parameters (and therefore using three instances of the CMA-ES optimizer), we have a single objective function because all three controllers are required to conduct an evaluation.

IV. Results/Accomplishments

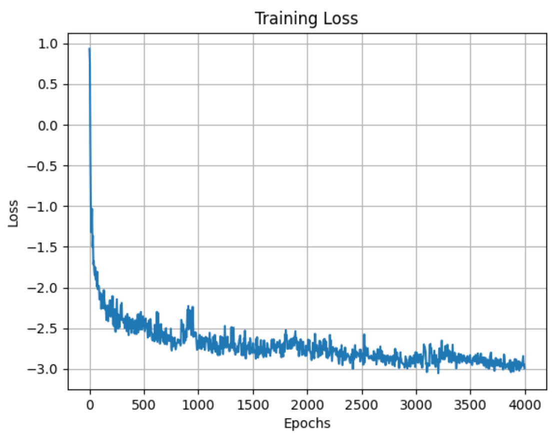
A. MDN-RNN

For the evaluation of the MDN-RNN, we first compared the training losses from the intended three-agent Multiwalker experiment with a single agent Multiwalker experiment as shown in Figure 4. We trained the MDN-RNN for 4000 epochs with 10000 random policy rollouts of the environment (the maximum we could before timing out of Google Colab GPU usage limit but also happened to be sufficient to see plateauing in the loss).

We then evaluated the trained multi-agent MDN-RNN model on 2000 random rollouts of the environment and



(a)



(b)

Figure 4. Training loss for (a) Three agent Multiwalker (b) Single agent Multiwalker

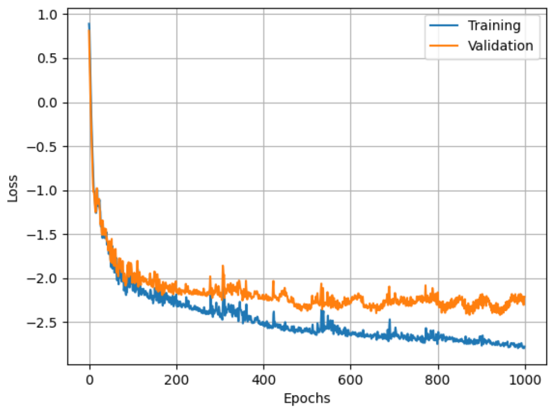


Figure 5. Training and validation losses over 1000 epochs

compared this validation loss with the training loss as shown in Figure 5.

B. Controller

Unfortunately, we were unable to complete the updates to the controller. We were able to modify the implementation to support the Multiwalker environment in a single process but ran out of time when attempting to parallelize. As result, we were limited in the amount of optimization we could perform and ultimately could not find controller parameters that performed well in the environment. We ran the CMA-ES optimizer with a population size of 8 which was significantly smaller than the original paper's population size of 64. The optimizer performed 100 generations (which was not enough for convergence) and required close to 3 hours of execution time. Every 20 generations, an evaluation was performed using a rollout simulation. The returned reward was stored for plotting and if this reward was the best so far, the parameters were also stored.

Figure 6 shows the evolution of the performance of the controllers during the optimization process. Our goal with the Multiwalker environment was to achieve a total cumulative reward of 300 so this figure clearly shows that more optimization is required. We believe that with added parallelization, we can run the optimizers for a sufficient amount of time to achieve the desired performance.

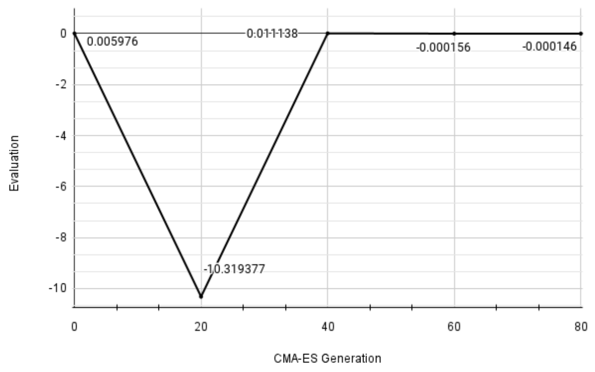


Figure 6. Progress of controller optimization over 100 generations

V. Conclusion

A. Achievements

We were able to train the MDN-RNN to the best of our abilities and to what turned out to be a sufficient number of iterations. Figure 4 shows that the training process resulted in similar losses for both the single agent and multi-agent setups of the Multiwalker environment, which we used as one indication of a successful training process. In addition, the trend in the plots is consistent with several tutorials we found online [2], [6], [1].

Figure 5 shows the training and validation losses, which generally agrees with what we expect. The validation loss generally tracks the training loss closely but it seems to be larger at higher iterations. This might be an indication of underfitting and will require more training iterations to close the gap between the losses.

Finally, as we tried to mention earlier, the controller will need more parallelized training to improve the optimization progress but we were at least able to show that despite the nice characteristics of ES being derivative-free and being parallelization friendly, it still requires extensive training process to be a reliable alternative.

In general, we were able to experience the need for time and fast hardware to facilitate the training and implementation of deep reinforcement learning algorithms. This will be a valuable lesson for us going forward and venturing into other related projects.

B. Future work

This project has identified many avenues for future work, most notably being the completion of controller optimization. After optimizing the parameters for the controllers, other opportunities for follow on work include updates to the software implementation and additional areas of research.

1) Future Software Implementation Updates: There are several updates to be made to our implementation of Multi-Agent World Models. Rendering of the Multiwalker environment is currently not supported in our implementation but would aid in verification of the results of the algorithm. In the future, we would like to modify our implementation to support an arbitrary number of agents in the Multiwalker environment. We would also like to increase the configurablity of the algorithms to make it easier to use for future research. Moreover, to enhance the speed and effectiveness of the algorithm, we are interested in exploring if the optimization process of controllers can be combined with the training process of the MDN-RNN. Currently, these steps are performed in serial: first training data is generated for the RNN, then the MDN-RNN is trained, then the controllers are optimized. There may be elegant ways to parallelize this process or structure it in an iterative method between controller and MDN-RNN that does not negatively impact the performance of any step. Finally, we would like to add the option of using the VAE model, which could be relevant for some set of more complex environments.

2) Future Research: Several interesting questions were raised during the completion of this project that could form the foundation for future research. The first is regarding the impact of sharing rewards. In our implementation, we configured the Multiwalker environment such that rewards were the same for each agent regardless of whether their independent actions resulted in different rewards. This allowed us to use a single objective function during the optimization process but the impact of doing so is not well understood and should be the focus of a future research project. Additionally, we would like to study the robustness of our implementation on varying dynamics. The Multiwalker environment can be configured to continue even if one agent falls. In other words, the number of agents in the environment can change during execution. We would like to study the the effectiveness of Multi-Agent World Models in this kind of application because it has significant real world implications. Finally, while completing this project, we were introduced to the concept of centralized training with decentralized control (CTDE). We would like to explore this MARL paradigm and if it would improve the performance of our Multi-Agent World Models implementation.

C. Acknowledgements

The authors would like to thank Professor Alessandro Roncone, Joewie Koh, and Kaushik Metha for their guidance, support and overall a challenging and rewarding semester.

References

[1] "Edward – mixture density networks," http://edwardlib.org/tutorials/mixture-density-network, (Accessed on 12/11/2022).

[2] "Mixture density networks with tensorflow | ," https://blog.otoro.net/2015/11/24/mixture-density-networks-with-tensorflow/, 2015, (Accessed on 12/11/2022).

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[4] L. Buşoniu, R. Babuška, and B. De Schutter, *Multi-agent Reinforcement Learning: An Overview*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 183–221. [Online]. Available: https://doi.org/10.1007/978-3-642-14435-6_7

[5] F. Christianos, G. Papoudakis, A. Rahman, and S. V. Albrecht, "Scaling multi-agent reinforcement learning with selective parameter sharing," 2021. [Online]. Available: https://arxiv.org/abs/2102.07475

[6] M. Dusenberry, "Mixture density networks," https://mikedusenberry.com/mixture-density-networks, 2017, (Accessed on 12/11/2022).

[7] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[8] D. Ha and J. Schmidhuber, "World models," 2018. [Online]. Available: https://worldmodels.github.io

[9] N. Hansen, "The CMA evolution strategy: A tutorial," *CoRR*, vol. abs/1604.00772, 2016. [Online]. Available: http://arxiv.org/abs/1604.00772

[10] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[11] N. J. Jie and W. Yujin, "Structured multi-agent world models." [Online]. Available: https://jetnew.io/assets/pdf/new2021structured.pdf

[12] X. Liu and Y. Tan, "Feudal latent space exploration for coordinated multi-agent reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2022. [Online]. Available: https://ala2020.vub.ac.be/papers/ALA2020_paper_22.pdf

[13] J. K. Terry, B. Black, A. Hari, L. S. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, C. Horsch, and P. Ravi, "Pettingzoo: Gym for multi-agent reinforcement learning," *CoRR*, vol. abs/2009.14471, 2020. [Online]. Available: https://arxiv.org/abs/2009.14471