World Models for Deep Reinforcement Learning

Shawn Manuel[1], Xiaobai Ma[2], and Mykel J. Kochenderfer[2]

Abstract— **Deep reinforcement learning (RL) has been used recently to solve control tasks with image inputs. Model-free RL algorithms are capable of learning a robust policy, but typically require many samples to learn an optimal policy. On the other hand, model-based methods improve sample efficiency, though there are challenges in applying them to high-dimensional observation spaces. The recent model-based approach for image domains known as "World Models" attempts to address the issue of sample complexity. It uses a variational auto-encoder and recurrent neural network to transform sequences of image observations to a latent feature vector and optimizes a simple controller network using an evolutionary algorithm. However, evolutionary algorithms often struggle to optimize policies with many parameters due to the exponential increase in the parameter search space. In this paper, we provide a general framework for incorporating the World Model architecture to train policies with deep RL, where the size of the network representing the policy is no longer constrained. We evaluate the efficiency of training RL agents using the World Model to compress high-dimensional image observations to low-dimensional feature representations. Results show that the World Model achieves more stable improvements in average reward as well as significantly decreasing training time. However, it was found that the World Model may not always learn a feature representation that is optimal for learning an optimal policy and hence may require fine tuning to the policy being learned.**

I. INTRODUCTION

Recent advances in computer vision with neural networks have enabled many real-world tasks in robotics [1]–[4], autonomous driving [5], [6], and navigation [7] to be learned with Deep Reinforcement Learning (RL) from high-dimensional images instead of low-dimensional sensor measurements. However, because RL agents trained in these domains receive a single image at a time, they are unable to infer temporal information such as velocity. As a result, RL agents often have to rely on recurrent neural networks (RNNs) or stacking sequences of recent images [8] to learn a feature representation of the underlying spatial and temporal dynamics. Both of these methods require a significant amount of computation for an RL agent to learn compared to simpler low-dimensional state spaces that can be learned with feed-forward networks. Hence, identifying a method to efficiently reduce the dimensionality of image inputs is an important challenge in improving the efficiency of Deep RL in image-based domains.

Model-free algorithms such as Deep Q-Networks (DQN) have achieved human-level performance in the image-based Atari2600 arcade game suite [8], [9]. Deep Deterministic Policy Gradients (DDPG) [10], Soft Actor Critic (SAC) [11], and Proximal Policy Optimization (PPO) [12] are state-of-the-art model-free methods for continuous control tasks, a domain in which most robotics and autonomous control applications operate. However, these algorithms reduce the dimensionality of image inputs with convolutional neural networks (CNNs) learned as a part of the policy to map images to actions, which requires many samples to sufficiently represent relevant features of the environment dynamics, leading to a high sample complexity [13]. In contrast, model-based approaches learn an explicit model of the environment dynamics and have been used for model-predictive control (MPC) in multiple-joint control [14]–[17], autonomous aircrafts [18], and autonomous driving [19]. However, these methods have primarily focused on low-dimensional state spaces. Since the applicability of RL in robotic and vehicle domains is heavily constrained by sample efficiency [2], [19], [20], it is of great interest to improve their sample efficiency of model-free RL methods in these complex image domains.

Inspired by the successful applications in image generation [21], [22], auto-encoders have been used to model environments with image observations [23]. Kaiser *et al.* used a VAE to learn an environment model for Atari games and applied PPO in the learned model, which significantly improved sample efficiency [24]. The learned model can also be used for state compression, which reduces the learning load on the policy side. Wahlström *et al.* used an auto-encoder to map visual inputs to low-dimensional features and applied MPC in the feature space [25]. Corneil *et al.* trained a VAE-based environment model encoding the observation history to a belief vector over discrete states, where the policy was represented using tabular Q learning [26]. The compressed latent space from a VAE could also be combined with state-of-the-art deep RL approaches. Igl *et al.* used the environment model as a particle filter, whose output is then encoded to serve as the state space for the A2C [28] learner [27]. However, besides the complex structure of the VAE-based particle filter, a RNN is still needed on the policy side to encode the belief particles to the state vector.

Ha and Schmidhuber introduced the "World Models" architecture, which decouples the environment representation task from the policy learning task [29]. This architecture incorporates a visual compressor modeled by a Variational Auto-Encoder (VAE) that maps high dimensional images to a lower dimensional latent vector. Then, a memory component modeled by an RNN is trained to predict the temporal dynamics of the environment from an input action and latent state. A simple decision-making module determines

[1] Shawn Manuel is with the Department of Computer Science at Stanford University, Stanford, CA. Email: `sman64@stanford.edu`

[2] Xiaobai Ma and Mykel J. Kochenderfer are with the Department of Aeronautics and Astronautics at Stanford University, Stanford, CA. Email: `mykel@stanford.edu`, `maxiaoba@stanford.edu`

the action an agent should take based on the compressed spatial and temporal representations. Using Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [30] to train the decision-making module, this system achieved state-of-the-art performance for a challenging simulated car racing environment with purely visual observations. However, a major limitation of CMA-ES is its limited scalability to networks with many parameters due to the exponential increase in search space with each additional parameter. On the other hand, training a model-free RL agent on the simplified state representation from the World Model allows for efficiently learning an optimal policy network of any size where the feature representation of the high dimensional states is already handled by the World Model.

In this paper, we incorporate the World Model architecture to reduce the dimensionality of image observations for training various model-free RL algorithms. We then evaluate the performance of agents trained with and without a World Model on image-based environments on the OpenAI and ViZDoom RL platform. We show that decoupling the environment representation task from the policy learning task significantly improves the sample efficiency training time of model-free RL algorithms.

II. BACKGROUND

We consider the standard partially observable reinforcement learning setting where an agent interacts with an environment $\mathcal{E}$ receiving image observations which do not contain temporal information [31]. At each time step t, the agent receives an observation s_t and selects an action a_t according to a policy π, which is a mapping from observations to actions. The agent then receives a next observation s_{t+1}, a scalar reward $r_t \in \mathbb{R}$, and a Boolean flag d_t indicating whether the agent reached a terminal state. An experience tuple at time t is $(s_t, a_t, s_{t+1}, r_t, d_t)$. A sequence of experience tuples from an initial state to a terminal state is known as a trajectory. The goal of an RL agent is to maximize the total expected cumulative reward at each time step, defined as $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ for a discount rate $\gamma \in (0, 1]$ [32].

A. Visual-based RL Environments

The three environments explored in this paper are OpenAI Gym's `CarRacing-v0`, and the `take_cover` and `defend_the_line` scenarios from the ViZDoom platform.

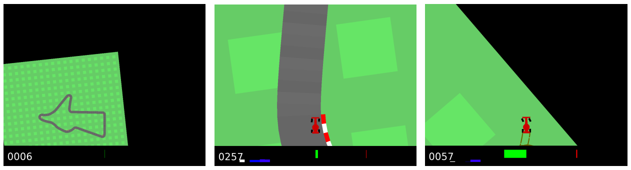

Fig. 1. Random Map (left), State Image (middle), Edge of map (right)

1) CarRacing-v0: The `CarRacing-v0` environment on OpenAI Gym [33] is a continuous action-space environment where the goal is to control the steering, acceleration, and braking of a car to race it around a randomly generated circuit track in as few time steps as possible. Each observation is an image of the top view of the car on the map as shown in Figure 1. The agent then needs to provide an action consisting of three continuous values for the steering, acceleration, and brake. The track is overlaid with n (usually between 230 and 320) rectangular tiles shown in Figure 1 (middle) where the agent receives a reward of $1000/n$ upon visiting a tile for the first time. A trajectory terminates after 1000 time steps, after the agent visits every tile, or after the agent drives off the edge of the map (right of Figure 1). The agent also receives a penalty of -0.1 at each time step, encouraging faster completion of the circuit.

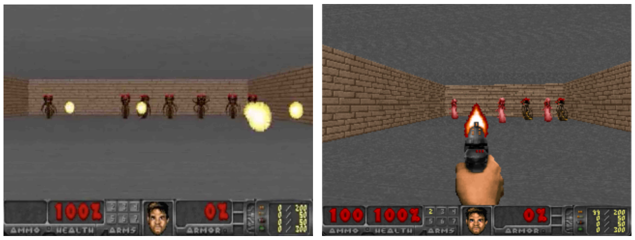

Fig. 2. The take_cover (left) and defend_the_line (right) scenarios

2) take_cover and defend_the_line: The scenarios in ViZDoom [34] are discrete action-space environments where the agent's goal is to survive in a room of monsters. The `take_cover` scenario (left of Figure 2) requires the agent to select between moving left or right to avoid fireballs launched towards the agent by opposing monsters. The agent receives a reward of $+1$ at every time step where the episode continues until the agent loses all of its health from being hit by fireballs. In the `defend_the_line` environment (right of Figure 2), the agent selects between turning left, right or shooting a pistol with limited ammunition straight ahead with the goal of eliminating as many approaching monsters in order to survive. The agent receives $+1$ for each monster eliminated and -1 for losing all health from monster attacks.

III. DEEP RL WITH WORLD MODELS

In this section, we reintroduce the World Models architecture [29] in the context of reinforcement learning. A trained World Model can be incorporated in the RL training loop as shown in Figure 3 to transform an image observation into a compressed representation that is then passed to the agent. This output consists of a latent encoding z_t of the image from a VAE and the hidden state h_t from a Gaussian Mixture Density RNN (MDRNN). The concatenation of these two vectors becomes the state feature representation, which the agent can treat as the environment's underlying state when learning an optimal policy.

A. VAE Model

A VAE consists of an encoder network that compresses an input image s_t into a latent feature vector z_t, and also a decoder network to reconstructed the image from the preserved features as shown in Figure 4. The VAE parameterized by θ is trained to minimize the difference

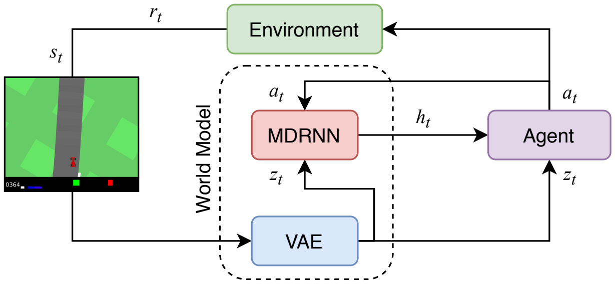

Fig. 3. MDP Loop with a World Model

between the reconstructed image and the input image with the constraint of minimizing the KL divergence between the latent encoding space and the normal distribution which was found to suitable prior over the latent variable z for maximum-likelihood feature representation [35]. This is achieved by minimizing the following loss function:

$$L_{\text{VAE}}(s_t;\theta) = (s_t - \hat{s_t})^2 + D_{\text{KL}}(p(z_t \mid s_t) \,\|\, \mathcal{N}(0,1)), \tag{1}$$

where the first term minimizes the difference between the reconstructed and input image and the second term penalizes large shifts in the conditional latent state distribution $p(z_t \mid s_t)$ from a unit normal. A VAE is ideal for this task as it learns to encode the most important sources of variation from the distribution of training images within a small linear vector. The KL divergence term ensures that the model is able to sufficiently represent less frequent states without overfitting to a particular region of the state space.

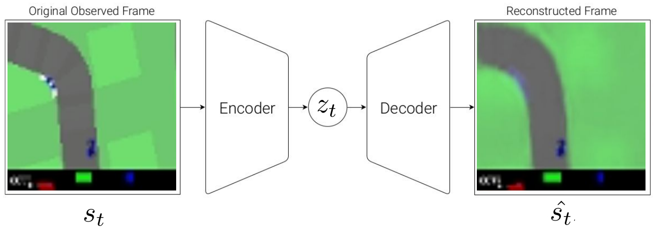

Fig. 4. Flow diagram of a VAE [29]

B. MDRNN Model

While the VAE transforms a high-dimensional image into a low dimensional latent vector, it is only a static representation at a single point in time and lacks information about the temporal dynamics of the environment. Therefore, an RNN is trained to encode temporal information over a sequence of latent vectors. Ha and Schmidhuber utilized a Long-Short-Term-Memory (LSTM) to model the temporal changes in the next state through its hidden LSTM state h_{t+1} calculated from the current latent vector z_t, chosen action a_t and its current hidden state h_t [29]. As the distribution of next states is often stochastic, the output of the LSTM is passed to a Gaussian-Mixture Density Network (MDN) [36] represented in Figure 5 as a linear layer which outputs the mean (μ), standard deviation (σ), and mixing weights ($\mathbf{w}$), for a specified number of Gaussian distributions. The predicted next latent state can be sampled by first sampling a Gaussian index from the mixture weights w and then sampling the latent vector $\hat{z}_{t+1}$ from the selected Gaussian.

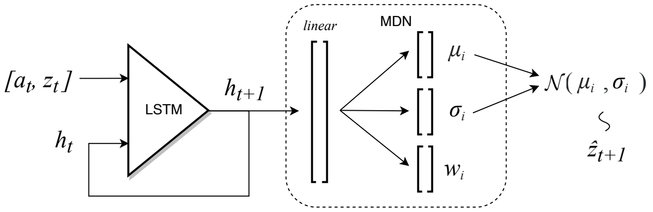

Fig. 5. Flow diagram of a MDRNN

The MDRNN can be modeled as a function parameterized by β defined as $M(a_t, z_t; h_t; \beta) \rightarrow h_{t+1}, \hat{z}_{t+1}$, where z is sampled from $\mathcal{N}(\mu_i, \sigma_i)$, and i is sampled from a categorical distribution with probabilities $\mathbf{w}$ [37]. Given an experience tuple $(s_t, a_t, s_{t+1}, r_t, d_t)$, the latent states z_t and z_{t+1} can be obtained as the output of a trained VAE given inputs s_t and s_{t+1}. Then, with the probability $p(z_{t+1} \mid \mu, \sigma; \beta)$ of sampling z_{t+1} from μ and σ, the MDRNN M can be trained by minimizing the following loss function:

$$L_{\text{MDRNN}}(\beta) = -\mathbb{E}_{(z,z')\sim\text{VAE}(\mathcal{E}),\ a\sim\mathcal{A},\ (\mu,\sigma,w)\sim M(a,z)} \left[\log\left(\sum_{i}^{n} w_i \ p(z'|\mu_i,\sigma_i;\beta)\right)\right] \tag{2}$$

C. Controller Model

The controller model is a simple linear mapping of the latent state from the World Model to an action. It was noted in [29] that the hidden state of the MDRNN h_t was sufficient for encoding temporal information of the current state since it is used to predict $\hat{z}_{t+1}$ from z_t and a_t. As a result, the controller can be represented as a linear function $a_t = W_c \cdot [z_t, h_t] + b_c$ where W_c is the weight and b_c is the bias for the single-layer network. As the controller is designed to have the minimum number of parameters for a linear function of $[z_t, h_t]$, it can be optimized with the CMA-ES algorithm. In this paper, the controller is not considered part of the World Model, but instead as a baseline agent for benchmarking deep RL agents trained with the World Model.

D. Training The World Model

Since the variation in the image observation space is stationary over time for a fixed policy, this allows the VAE to be trained independently from the MDRNN. Similarly, as the controller is responsible only for policy optimization, it can be trained with simplified states from a pre-trained World Model. Algorithm 1 outlines the procedure for training each component of the World Model. Upon initializing the parameters of the VAE V, MDRNN M, and Controller C with the appropriate InitParameters function, a dataset $\mathcal{D}$ is extended with N trajectories sampled from the environment $\mathcal{E}$ using C as the policy. Then, with learning rate α, V is trained on batches of images from $\mathcal{D}$. M can then be trained using the previously trained V to compress images to latent vectors. C is then optimized using CMA-ES given a trained World Model (V,M). It is possible that one iteration of this training process is not enough to encounter optimal states of the environment since the initial trajectories are sampled with a random controller. Hence, this loop is repeated using the previously trained controller and World Model for a specified S iterations depending on the complexity of the environment.

IV. Deep RL with World Models

This section describes the process of training RL agents using a trained World Model, as well as the architecture of the RL algorithms to be trained with the World Model.

A. *Training RL Agents With The World Model*

Given a pre-trained World Model as the output from Algorithm 1, the RL agent can focus solely on identifying the optimal mapping from latent features to actions that maximize total reward. Algorithm 2 describes the procedure for stepping through the environment $\mathcal{E}$ with a World Model consisting of the VAE V and MDRNN M that are either trained using Algorithm 1 or loaded from a previously trained World Model on the same environment $\mathcal{E}$. Then, the training of an RL policy π towards an optimal policy occurs over a specified number of training trajectories. At the start of each training trajectory, the MDRNN hidden state h is initialized with InitHidden, the initial observation s is sampled from the environment with InitState, and the latent encoding z of the initial image observation is sampled from the VAE. The flag d signifying reaching a terminal state is initialized to false. Then, until a terminal state is reached, the RL training loop proceeds with the Step function applying each agent action a to the environment and outputting the next observation s', reward r and updated terminal flag d. The observations s and s' are replaced with the latent state $[z, h]$ and next latent state $[z', h']$, where z and h are the outputs of the VAE V and MDRNN M respectively. The agent is then given the updated experience tuple with latent states using the where it can store or use in a training update through the Train function implemented according to a specified RL algorithm.

B. *RL Agent Architecture*

The RL algorithms to be trained with the World Model include Double DQN (DDQN) [38], DDPG [10], PPO [12], and SAC [11]. These agents will be tested first with raw grey-scale image inputs to evaluate the baseline performance of learning an environment representation along with the learning of the optimal policy. For the baseline, each state input will consist of the current image observation stacked on top of the two previous observations. Then, the architecture of the baseline actor and critic will consist of a convolutional component to reduce the 2D image to a 1D vector which is then passed to a feed-forward component which will output the desired action or state-value. The convolutional component will share the same network architecture as the VAE encoder having four convolutional layers. The feed-forward component will consist of three layers to map the output of the convolutional component or world model to the desired output size of the actor or critic.

Algorithm 1 Training Loop for World Model

```
procedure $\textsc{TrainWorldModel}(\mathcal{E}, S, N)$
    $V(s;\theta) \leftarrow \textsc{InitParameters}(\theta)$
    $M(z,a;\beta) \leftarrow \textsc{InitParameters}(\beta)$
    $C(z,h;W_c,b_c) \leftarrow \textsc{InitParameters}(W_c, b_c)$
    $\mathcal{D} \leftarrow \emptyset$
    for $i$ in $\{1, \ldots, S\}$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \textsc{SampleTrajectories}(\mathcal{E}, C, N)$
        for $batch$ in $\mathcal{D}$
            $\mathcal{L}_\theta \leftarrow L_{\text{VAE}}(batch;\theta)$
            $\theta \leftarrow \theta - \alpha\nabla\mathcal{L}_\theta$
        for $batch$ in $\mathcal{D}$
            $\mathcal{L}_\beta \leftarrow L_{\text{MDRNN}}(V, batch;\beta)$
            $\beta \leftarrow \beta - \alpha\nabla\mathcal{L}_\beta$
        $C \leftarrow \text{CMA\_ES}(C, V, M)$
    return $V, M$
```

Algorithm 2 RL Training With Trained World Model

```
procedure $\textsc{TrainRLAgent}(\mathcal{E}, S, N)$
    $V, M \leftarrow \textsc{TrainWorldModel}(\mathcal{E}, S, N)$
    $\pi(z,h;\phi) \leftarrow \textsc{InitParameters}(\phi)$
    for each trajectory do
        $h \leftarrow \textsc{InitHidden}(M)$
        $s \leftarrow \textsc{InitState}(\mathcal{E})$
        $z \leftarrow V(s;\theta)$
        $d \leftarrow$ false
        while $d =$ false
            $a \leftarrow \pi([z,h];\phi)$
            $s', r, d \leftarrow \textsc{Step}(\mathcal{E}, a)$
            $h' \leftarrow M(z, a; h; \beta)$
            $z' \leftarrow V(s';\theta)$
            $\textsc{Train}(\pi, ([z,h], a, [z',h'], r, d))$
            $[z,h] \leftarrow [z',h']$
```

All agents will be trained on batches of experience tuples from 16 parallel environments as done in [32]. An experience replay buffer of a maximum size 100000 samples will be used for off-policy algorithms DDQN, DDPG and SAC. DDQN and DDPG rely on explicit random exploration schedules which will be implemented with a Brownian noise process where the scale of noise is reduced from 1.0 to a minimum of 0.02 at a decay rate of 0.98 after each trajectory.[1]

V. Experimental Setup

To evaluate the effectiveness of the World Model, we trained each RL algorithm for 500000 steps on each of the three environments (except DDQN on `CarRacing-v0` due to continuous state space) with and without the World Model. The World Model was trained over two iterations for each component with 1000 sampled trajectories added to the data

[1] The complete implementation is available on Github at https://github.com/sisl/WorldModelsForDeepRL

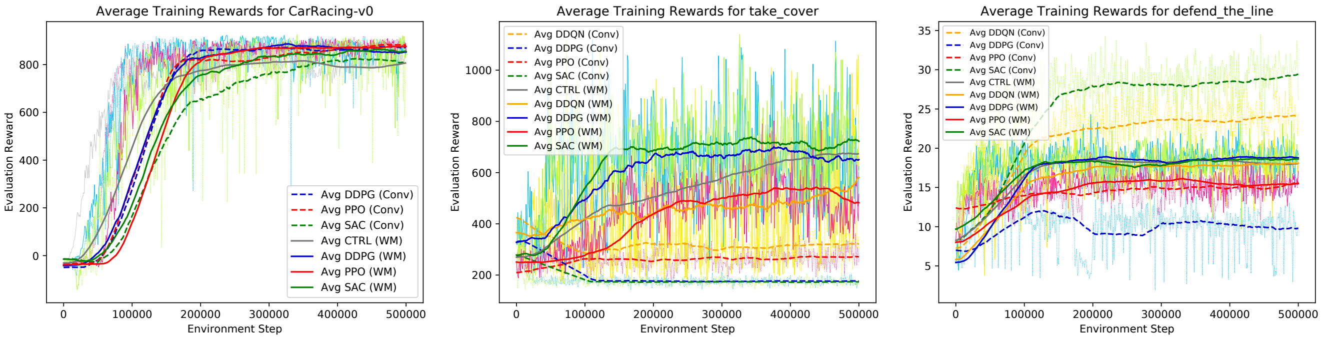

Fig. 6. The rolling average evaluation rewards over the previous 100000 steps for different RL algorithms trained in the CarRacing-v0 (left), take_cover (mid), and defend_the_line (right) environments. In each plot, the solid thick lines represent training with the pre-trained World Model and dashed thick lines represent training using convolutional layers. The thin lines represent evaluation rewards every 1000 steps.

set at each iteration, where the controller was trained for 500 generations using CMA-ES. The training plots in Figure 6 show the variability in the effect of the World Model on training efficiency. For most RL algorithms, training with the World Model outperformed the controller trained using CMA-ES in terms of maximum average reward.

The results for the `CarRacing-v0` environment in Figure 6 (left) show that agents trained with the World Model appeared to achieve the same overall performance as agents trained using a convolutional network with stacked images. Although training PPO and DDPG without the World Model were able to achieve faster improvement in average reward, the agents trained using the World Model were able to maintain this performance with less variation in rewards at each training iteration. The time required to train agents with the World Model was significantly less, taking approximately 7 hours in contrast to the 11 hours required to train a new convolutional network for each agent.

The `take_cover` environment rewards shown in Figure 6 (middle) exhibit the most variation between agents using the Model and those without. It is clear that only the agents trained with the World Model were able to improve their average score while the agents training a convolutional network collapsed to a degenarate policy. Investigating playbacks of the trained agents revealed that policies converged to only move left. Ha and Schmidhuber referred to this being the result of the credit assignment problem where backpropagation of gradients through a large neural network with many layers and parameters can hinder convergence [29]. However, the World Model enables the agent to use a shallower network for policy learning.

In the `defend_the_line` environment, training with the world model achieved a constant but stable reward compared to the agents using convolutional networks as seen in Figure 6 (right). However, an outlying observation of SAC and DDQN converging to a much higher average reward when trained without a World Model suggests that the World Model may not always capture the optimal variational features for optimization of the policy for fine variations in the image states. To address this issue, the pre-trained World Model could be further trained along with the policy to allow for finer optimization at the image processing level.

As the plots show, the World Model has many benefits for deep RL. We demonstrated that a single pre-trained World Model can generalize the environment representation for use with different agents learning in the same environment, which can be applied to various domains with a stationary state distribution. It reduces the number of parameters that the RL agent needs to update, which reduces the time taken for each update, equivalently leading to faster training. Along with saving time, the World Model reduces the memory requirement of off-policy agents using a replay buffer where high dimensional states can be directly mapped to a latent state with the World Model before being stored.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the improvement in the efficiency of training reinforcement learning agents using the World Models architecture to compress high dimensional image observations to low dimensional feature representations. We proposed a general procedure for integrating World Models within any reinforcement learning task, allowing the environment representation to be learned offline, independently from the training of the policy. We also show that the resulting approach generalizes to different RL algorithms learning within the same environment. However, it was found that World Model may not encode the full scope of variation in the image states of the environment, leading to convergence to suboptimal policies in some environments.

For future work, the pre-trained World Model can be further optimized during policy optimization for fine-tuning of the latent features that are most important for action selection. Another direction is to improve the accuracy of the World Model with attention mechanisms. Overall, we found that the World Model architecture has potential to improve stability and speed of training RL policies in complex environments, in addition to generalizing to different algorithms.

REFERENCES

[1] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018.

[2] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *arXiv preprint arXiv:1710.06542*, 2017.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[4] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 2219–2225.

[5] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3357–3364.

[6] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," *arXiv preprint arXiv:1811.11329*, 2018.

[7] D. Li, D. Zhao, Q. Zhang, and Y. Chen, "Reinforcement learning and deep learning based lateral control for autonomous driving," *arXiv preprint arXiv:1810.12778*, 2018.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] M. Volodymyr, K. Koray, S. David, G. Alex, A. Ioannis, W Daan, and R Martin, "Playing Atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, 2018, pp. 1861–1870.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 8224–8234.

[14] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[15] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," *arXiv preprint arXiv:1803.00101*, 2018.

[16] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv preprint arXiv:1802.10592*, 2018.

[17] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *Conference on Robot Learning*, 2018, pp. 617–629.

[18] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[19] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1714–1721.

[20] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Multi-step retrieval of a target object occluded by clutter," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 1614–1621.

[21] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin, "Variational autoencoder for deep learning of images, labels and captions," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 2352–2360.

[22] X. Yan, J. Yang, K. Sohn, and H. Lee, "Attribute2image: Conditional image generation from visual attributes," in *European Conference on Computer Vision (ECCV)*, Springer, 2016, pp. 776–791.

[23] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese, "Deep visual mpc-policy learning for navigation," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3184–3191, 2019.

[24] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[25] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *arXiv preprint arXiv:1502.02251*, 2015.

[26] D. Corneil, W. Gerstner, and J. Brea, "Efficient model-based deep reinforcement learning with variational state tabulation," *arXiv preprint arXiv:1802.04325*, 2018.

[27] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, "Deep variational reinforcement learning for pomdps," *arXiv preprint arXiv:1806.02426*, 2018.

[28] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, *Openai baselines*, `https://github.com/openai/baselines`, 2017.

[29] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 2450–2462.

[30] N. Hansen, "The CMA evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[31] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[34] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A Doom-based AI research platform for visual reinforcement learning," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations*, vol. 1, 2014.

[36] C. M. Bishop, "Mixture density networks," 1994.

[37] K. O. Ellefsen, C. P. Martin, and J. Torresen, "How do mixture density RNNs predict the future?" *arXiv preprint arXiv:1901.07859*, 2019.

[38] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, vol. 48, 2016, pp. 1995–2003.