World Models for Deep Reinforcement Learning

Shawn Manuel[1] and Mykel J. Kochenderfer[2]

***Abstract*— Policy optimization with reinforcement learning has many applications for control tasks such as robotics or autonomous navigation where observations are usually in image form. However, current approaches to learning an optimal policy directly from images require the reinforcement learning agent to first learn a feature representation of the image state distribution which is usually independent of the policy being learned. In this paper, we propose an architecture that decouples the reinforcement learning task into two separate phases; learning a representation of the spatial and temporal features of the environment dynamics followed by training a policy to select actions given a simplified state representation. We test the generalizability of the architecture for environment state simplification for different reinforcement learning algorithms on OpenAI's CarRacing-v0 environment and two ViZDoom first-person environments. Results show that training policies with simplified state representations from a pre-trained environment model significantly improves the speed of learning an optimal policy.**

I. INTRODUCTION

Recent advances in deep learning capabilities have enabled the application of deep reinforcement learning (RL) to many real-world tasks such as robotics [1], [2], autonomous driving [3], [4] and navigation [5] where observations are usually in the form of high dimensional images. In order to test deep reinforcement learning approaches for training agents from image states, platforms such as Atari 2600, ViZDoom and OpenAI Gym were developed as a simplified variants of these applications [6], [7], [8]. However, a single image state and only contains static information about the true state of the environment and fails to satisfy the markov property that is a key assumption of reinforcement learning algorithms [9]. Previous approaches to solve this problem involve either training a recurrent neural network (RNN) to store components of the state history, or stacking a sequence of images as input at each time step [8]. Both of these approaches increase the computational load on deep RL where it has to learn a low dimensional embedding of the underlying environment dynamics from the high dimensional image states in addition to learning the optimal policy.

State-of-the-art RL algorithms have enabled effective learning of optimal policies for simulated tasks in robotics [10], [11], multiple-joint continuous control [12], [13] and arcade gaming with discrete actions [14], [8]. However, most of these algorithms are model-free approaches, where the policy is trained via policy gradients guided by a state-value estimate, or value-based where the state-value is modeled by a function that is trained with the agent. In contrast, model-based approaches involve learning an explicit model of the environment dynamics and have been used for model-predictive control in autonomous aircraft control [15] and autonomous driving [16]. However, model-free approaches suffer from a higher sample complexity, requiring many interactions with the environment to sufficiently represent the relevant features of the environment dynamics. Since the applicability of RL in most practical robotic and vehicle domains is heavily constrained by sample efficiency [16], [2], [17], it is of great interest to reduce the computational load of model-free RL methods to improve their applicability for complex visual control domains.

As one approach to address this challenge for model-free policy optimization, citeauthorha2018recurrent introduced the "World Models" architecture that decouples of the environment representation task from the policy learning task. 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 the action an agent should take based on the compressed visual and temporal representations. Using Covariance-Matrix Adaptation Evolutionary Strategy (CMA-ES) [18] to train the decision-making module, this system achieved a new state-of-the-art performance for a challenging simulated car racing environment with purely visual observations. However, CMA requires many iterations to search for a policy and is also only effective when the parameter space is small. 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 compress the dimensionality of image observations received from the environment to allow a reinforcement learning agent to focus on maximizing rewards associated with this latent feature representation. We then evaluate the performance of such agents relative to the controller trained with parameter evolution in [19] on OpenAI's CarRacing-v0 environment and two scenarios from the ViZDoom platform. We show that decoupling the environment representation task from the policy learning task significantly improves the performance and sample efficiency of reinforcement learning agents.

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

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

II. BACKGROUND

We consider the standard reinforcement learning setting where an agent interacts with an environment $\mathcal{E}$ having a state space $\mathcal{S}$ and action space $\mathcal{A}$ for a number of discrete time steps [9]. At each step t, the agent receives a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ according to a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which is a mapping from states to actions. The agent then receives a next state $s_{t+1} \in \mathcal{S}$, a scalar reward $r_t \in \mathbb{R}$ and a flag $d_t \in \{True, False\}$ indicating whether the agent reached a terminal state. We will refer to each time step's terms $(s_t, a_t, s_{t+1}, r_t, d_t)$ as an experience tuple. A sequence of experience tuples from an initial state up to a terminal state is known as a trajectory. The goal of a reinforcement learning 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 $\gamma \in (0, 1]$ [20].

Deep reinforcement learning involves representing a policy with a neural network and using gradient-based optimization algorithms to train it towards an optimal policy. Previous work in policy optimization can be categorized as value-based, policy-based and Actor-Critic approaches.

1) Value-Based Methods: Deep Q-Learning (DQN) is a value-based algorithm for discrete action spaces that models the expected value distribution $Q(s, a)$ of taking each discrete action a from a given state s. The value function is trained towards an optimal Q-function $Q*(s, a)$ that satisfies the Bellman equation defined in [14] as $Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_a Q^*(s_{t+1}, a)]$ for all states and actions. Given this optimal value function, a Q-Learning policy $\pi^*(s_t)$ will select the action with the maximum Q-value where $a_{t+1} = \pi^*(s_t) = \arg\max_a Q^*(s_t, a)$.

2) Policy-Based Methods: Policy search algorithms are generally gradient-free approaches that aim to find a policy as a direct mapping from states to actions by evaluating the performance of a population of policies with different network parameters, then perturbing the network parameters of the best-performing agents to generate the next population to evaluate.. The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [18] is popular for use in evolving neural network parameters as it is capable of searching solution spaces of up to a few thousand parameters [19]. However, this places an implicit limit to the size and expressiveness of the neural network, where larger networks require larger population sizes in order to adequately explore the policy space [9].

3) Actor-Critic Methods: Actor-critic methods include a policy (actor) that maps states directly to actions as well as a value function (critic) which is used to train the policy towards selecting optimal actions with respect to the value function. Three popular actor-critic approaches trained in this paper are Deep Deterministic Policy Gradients (DDPG) [21], Proximal Policy Optimization (PPO) [13] and Soft Actor-Critic (SAC) [12].

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.

B. OpenAI Gym's CarRacing-v0

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

The `CarRacing-v0` environment is a continuous action-space environment where the goal is to control the steering, acceleration, and braking of the car to race around a randomly generated circuit track in as few time steps possible. Each state is a 96x96 RGB image of the top view of the car on the map seen in Figure 1 in the middle. The agent then needs to provide an action consisting of three continuous values for the steering (between -1 and 1) and the acceleration and brake (between 0 and 1). The track is overlaid with n (usually between 230 and 320) rectangular tiles (seen in the middle of Figure 1) and the agent receives a reward of $1000/n$ upon visiting a tile for the first time. A rollout terminates after 1000 time steps, if the agent visits every tile, or if it drives off the edge of the map (right of Figure 1). The agent also receives a penalty or -0.1 at each time step, encouraging faster completion of the circuit. The environment is considered solved for an agent that achieves an average reward of at least 900 over 100 consecutive trials.

C. ViZDoom's take_cover and defend_the_line

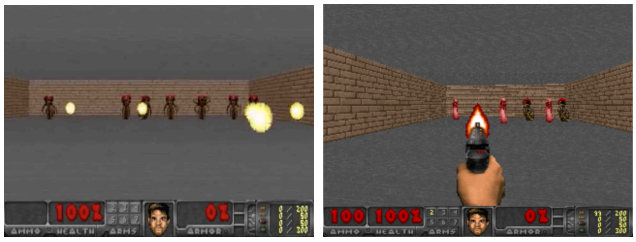

Fig. 2. The take_cover (left) and defend_the_line (right) scenarios

The scenarios in ViZDoom are discrete action-space environments where the agent's goal is to survive in a room of monsters. The `take_cover` scenario shown on the 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 its health from being hit by fireballs. In the `defend_the_line` environment, the agent selects between turning left, right or shooting a pistol with limited ammunition straight ahead with the aim of killing as many approaching monsters to survive. The agent receives +1 for each monster killed and -1 for losing all health from monster attacks.

III. DEEP RL WITH WORLD MODELS

In this section, we reintroduce the World Models architecture from [19] 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 state into a compressed representation which is then passed to the agent. This output consists of the 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 for learning an optimal policy.

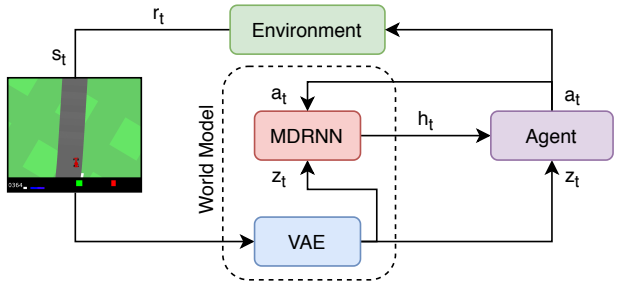

Fig. 3. MDP Loop with a World Model

A. VAE Model

As the states sampled from the environment are high dimensional images, a VAE is used to compress an input state image s_t into a latent feature vector z_t with an encoder component which can also be reconstructed with a decoder component to observe the preserved features as shown in Figure 4. The VAE parameterized by θ is trained to minimize the difference 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 [22]. This is achieved by minimizing the following loss function:

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

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.

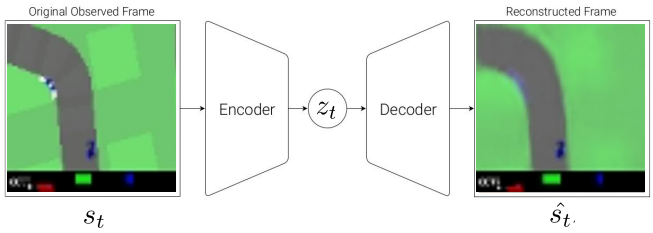

Fig. 4. Flow diagram of a VAE [19]

B. MDRNN Model

While the VAE transforms a high dimensional image state into a low dimensional latent vector, it is only a static representation at a single point in time which lacks information about the temporal dynamics of the environment, such as the velocity of a car. Therefore, an RNN is trained to encode dynamic information over a sequence of latent vectors. In [19], a Long-Short-Term-Memory (LSTM) was used to model the temporal component of 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. As the distribution of the possible next states is often stochastic, the output of the LSTM is passed to a Gaussian-Mixture Density Network (MDN) [23] represented in Figure 5 as a linear layer which outputs the mean (μ), standard deviation (σ), and mixing weights (w), for a specified number of Gaussian distributions forming a mixture of Gaussians according to the mixing weights. 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 that Gaussian. The

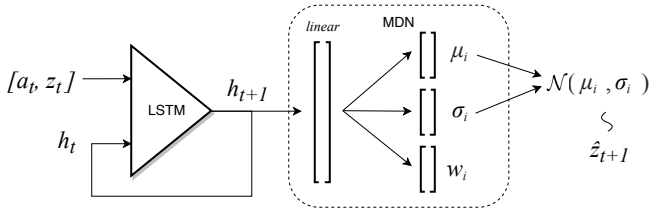

Fig. 5. Flow diagram of a MDRNN

MDRNN model can be considered 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 $\mathbf{w}$ [24]. 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 $p(z_{t+1}|\mu, \sigma; \beta)$ of sampling z_{t+1} from μ and σ, the MDRNN M can be trained by minimizing the following loss function:

$$L_{MDRNN}(\beta) = -\mathbb{E}_{(z,z')\sim VAE(\mathcal{E}),\ a\sim\mathcal{A},\ (\mu,\sigma,w)\sim M(a,z)} \left[\log \left(\sum_i^n w_i \ \ p(z'|\mu_i, \sigma_i; \beta) \right) \right] \tag{1}$$

C. Controller Model

The controller model is a simple linear mapping of the simplified state from the World Model to an action. It was noted in [19] 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. The controller can be represented as a 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

Algorithm 1: Training Loop for World Model

Function $TRAIN\_WORLD\_MODEL(\mathcal{E}, S, N)$**:**
- $V(s;\theta) \longleftarrow InitParameters(\theta)$
- $M(z, a;\beta) \longleftarrow InitParameters(\beta)$
- $C(z, h;W_c, b_c) \longleftarrow InitParameters(W_c, b_c)$
- $\mathcal{D} \longleftarrow Set()$
- **for** i ***in*** $1..S$ **do**
  - $\mathcal{D} \longleftarrow \mathcal{D} \cup sample\_trajectories(\mathcal{E}, C, N)$
  - **for** $batch$ ***in*** $\mathcal{D}$ **do**
    - $\mathcal{L}_\theta \longleftarrow L_{VAE}(batch;\theta)$
    - $\theta \longleftarrow \theta - \alpha\nabla\mathcal{L}_\theta$
  - **for** $batch$ ***in*** $\mathcal{D}$ **do**
    - $\mathcal{L}_\beta \longleftarrow L_{MDRNN}(V, batch;\theta)$
    - $\beta \longleftarrow \beta - \alpha\nabla\mathcal{L}_\beta$
  - $C \longleftarrow$ *CMA_ES*(C, M, V)
- **return** $V(s;\theta), M(z, a;\beta)$

Algorithm 2: RL Training With Trained World Model

Function $TRAIN\_RL\_AGENT(\mathcal{E}, S, N, T, E)$**:**
- $V(s;\theta), M(z, a;\beta) \longleftarrow$ $TRAIN\_WORLD\_MODEL(\mathcal{E}, S, N)$
- $\pi(z, h;\phi) \longleftarrow InitParameters(\phi)$
- **for** $episode$ ***in*** $1..E$ **do**
  - $h \longleftarrow init\_hidden(M)$
  - $s \longleftarrow init\_state(\mathcal{E})$
  - $z \longleftarrow V(s;\theta)$
  - $d \longleftarrow False$
  - **while** d ***is*** $False$ **do**
    - $a \longleftarrow \pi([z, h];\phi)$
    - $s', r, d \longleftarrow step(\mathcal{E}, a)$
    - $h' \longleftarrow M(z, a; h;\beta)$
    - $z' \longleftarrow V(s';\theta)$
    - $train(\pi, ([z, h], a, [z', h'], r, d))$
    - $[z, h] \longleftarrow [z', h']$

linear function of $[z_t, h_t]$, it can be trained with the CMA Evolutionary Strategy. In this paper, the controller is not considered as 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 state 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 from simplified states of a trained World Model. Algorithm 1 outlines the procedure for training each component of the World Model. Upon initializing the parameters of all components, a dataset of N trajectories is sampled from the environment $\mathcal{E}$ using the current controller C. Then, with learning rate α, the VAE V is trained on batches of state images from the dataset. The MDRNN M can then be trained using the previously trained VAE to compress images to latent vectors. The controller is then trained using the CMA Evolutionary Strategy given a trained World Model. 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

Upon completion of the training of the World Model, it can now be considered as part of the environment with the simple function of mapping sequences of non-markov images to a markov sequence of low dimensional features. This section describes the process of training reinforcement learning 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

The goal of using a World Model for training RL agents is to separate the learning of an optimal policy from the learning of a feature representation of the environment. 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 described the procedure for stepping through the environment $\mathcal{E}$ with a World Model for RL training. First, the VAE V and MDRNN M forming the World Model are either trained as per Algorithm 1 or loaded from a previously trained World Model on the same environment $\mathcal{E}$. Then, for a specified number of training trajectories E, the MDRNN hidden state h is initialized and the initial VAE encoding z is sampled from the environment's initial state s. Then, the standard RL loop follows with each agent action a being taken in the environment, producing an experience tuple (s, a, r, s', d). However, the state s and next state s' are replaced with the latent state $[z, h]$ and next latent state $[z', h']$ where z is the latent feature vector of the image s and h is the next hidden LSTM state of the MDRNN when computed on z and a and its previous hidden state. The agent is then given the updated experience tuple with latent states via the *train* function where it can store or use in a training update depending on the RL algorithm.

B. RL Agent Architecture

The RL algorithms to be trained with the World Model include value-based Double DQN (DDQN) [25], and actor-critic DDPG [21], PPO [13] and SAC [12]. These agents will be tested first with raw 64x64x3 images s_t as inputs to evaluate the baseline performance without a World Model providing a simplified state representation. Each individual image will be converted to a single channel 64x64 grayscale frame and stacked on top of the previous two frames to form an input size of 64x64x3, allowing for temporal information

to be included in the difference between frames in sequence. Then, the architecture of the baseline actor and critic will consist of a convolutional component to reduce the 2D image to a vector which is then passed to a feed-forward component representing the respective objective output function. The convolutional component will have four convolutional layers matching the VAE's encoder network in the VAE with the final output having a size of 512. When trained with the World Model, the convolutional component will be replaced with a single fully connected layer transforming the input to a size of 512 to have the same output size of the convolutional component. The feed-forward component of the actor and the critic will consist of an input layer of size 512 followed by a hidden layer of size 256 in the actor network or 1024 in the critic network. This layer will then be passed through a final layer which will have the required output size of the given actor or critic.

All agents will be trained by selecting actions in 16 parallel environments and training on batches of experience tuples [20]. An experience replay buffer of max length of 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 episode. The complete implementation of the networks is available on Github at [26].

V. EXPERIMENTAL SETUP

To evaluate the effectiveness of the World Model, we trained each RL algorithm over 500000 steps on each of the three environments (except DDQN on CarRacing-v0 due to continuous state space) with and without the World Model. When training without the World Model, the agent used an untrained convolutional network to process a stack of 3 sequential images at each state. The World Model was trained over two iterations of each component on 1000 trajectories added to the data set at each iteration, where the controller was trained for 500 generations of the CMA-ES algorithm.

A. Results

Figure 6, 7 and 8 on the right show the average evaluation rewards of each agent evaluated every 1000 steps. In each plot, the solid lines represent training with the pre-trained World Model and dashed lines represent training using convolutional layers. The average evolution rewards of the controller at each generation are shown as the grey line on each plot.

B. Discussion

It can be observed from the training plots for each environment that the effect of the World Model on training efficiency varies with the type of environment. For most RL algorithms, training with the World Model outperformed the controller trained using an evolutionary algorithm in terms of maximum average reward at end of training.

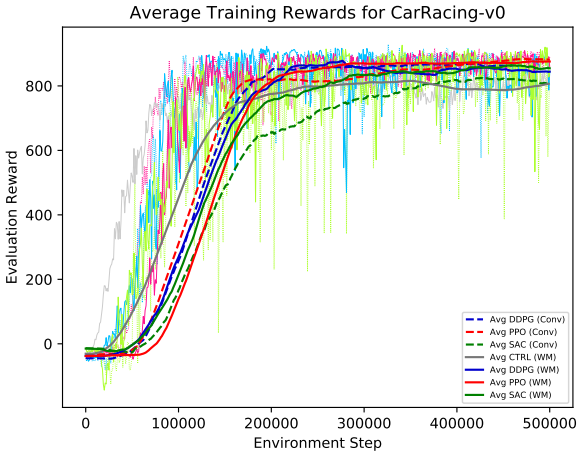

Fig. 6. Evaluation Rewards for CarRacing-v0

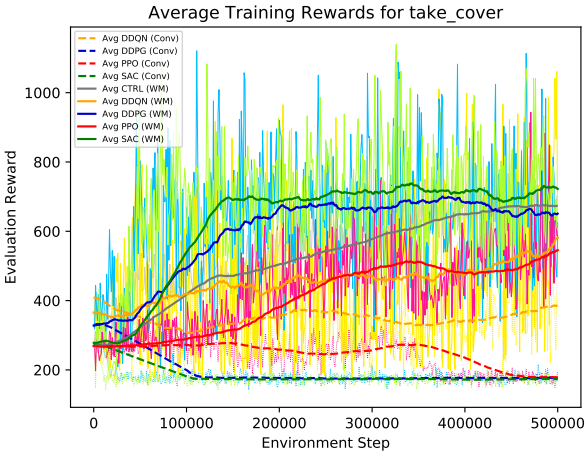

Fig. 7. Evaluation rewards for take_cover

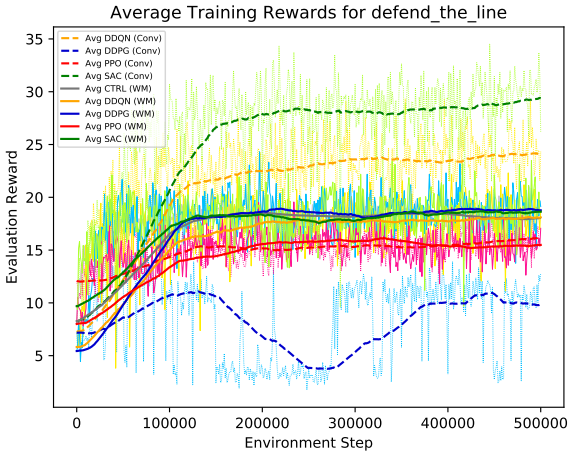

Fig. 8. Evaluation rewards for defend_the_line

1) CarRacing-v0: As the `CarRacing-v0` environment's state representation is a top view of the car on the racing track with distinct colour contrast between the track and grass, the variation in the underlying feature space can be more easily inferred from the high dimensional image. Therefore it is unsurprising that agents trained with the World Model appeared to achieve the same overall performance as agents trained using stacked images and a convolutional network from Figure 6. However, the clock 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.

2) take_cover: The `take_cover` environment results in Figure 7 show the most variation in rewards. 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 minimum constant reward. Investigating playbacks of the trained agents showed that policies converged to only move left, hence being easily hit by a fireball. One possible cause of this could be in the backpropagation of rewards through a large neural network with many convolutional layers as this suffers from the credit assignment problem [19]. However, the World Model enables the agent to use a shallower network for policy learning which eliminates this problem.

3) defend_the_line: In the `defend_the_line` environment, training with the world model achieved a constant but stable reward compared to the agents using convolutional networks. However, an outlying observation of SAC and DDQN trained without a World Model converging to a much higher average reward suggests that the World Model may not always capture the optimal variational features for a particular algorithm, which can restrict agents from fully optimizing the policy for the fine variations in the image states. In order 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 observed from the resulting plots, the World Model has many benefits for deep reinforcement learning. Firstly, 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. Secondly, 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 applied the recent World Models architecture to the field of deep reinforcement learning to simplify a high dimensional image state into a compressed feature vector for a reinforcement learning agent to learn an optimal policy. We proposed a general procedure for integrating World Models within any reinforcement learning task, allowing the environment representation to be learned offline, and independently from the policy training. 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, enabling the agent to fine-tune the features encoded in the latent space that are most important for action selection. Another direction is to improve the accuracy of the World Model with attention mechanisms, or allowing the World Model to be optimized during policy optimization. Overall, we found that the World Model architecture has potential to improve stability and speed up the training of reinforcement learning agents in complex high dimensional environments, along with reducing the required size of the policy.

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] 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 *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

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

[5] 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.

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

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

[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] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

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

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

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

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

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

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

[16] 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 *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[17] 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 *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1614–1621.

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

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

[20] 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*, 2016, pp. 1928–1937.

[21] 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.

[22] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

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

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

[25] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[26] S. Manuel, "Project Repository: WorldModelsForDeepRL," https://github.com/shawnmanuel000/WorldModelsForDeepRL, published: 2019-12-03.