World Models for Deep Reinforcement Learning

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

***Abstract*— We consider an application of the recently successful 'World Model' architecture using generative neural networks in reinforcement learning to abstract high dimensional image observations into compressed spatial and temporal vector representation. We propose to incorporate the World Model with a deep reinforcement learning agent to learn an action-selection policy from the high-level features of the environment represented by the World Model. We tested two policy gradient actor-critic learning algorithms, Deep Deterministic Policy Gradients and Proximal Policy Optimization, trained using the World Model on OpenAI Gym's CarRacing-v0. We found that these agents were able to achieve scores at the level of the current state of the art evolutionary algorithms for training an agent. It was also observed that the simplified state representation of the World Model improved the speed and stability of the training process as well as enabling simpler neural network architectures than equivalent agents trained from raw image states.**

I. INTRODUCTION

Deep Reinforcement Learning is a growing field where neural networks are employed to learn a general policy for selecting actions in a given environment. State of the art reinforcement learning algorithms such as Q-Learning have evolved to allow neural networks to learn effective policies for simple control tasks such as balancing a cartpole or swinging a pendulum. In these cases, the agent often has access to the true dynamics of the environment as a low dimensional observation and is therefore able to learn optimal policies using a simple feed forward neural network. However, tasks such as learning to control a simulated car and then navigate a track given only raw images are much more difficult to learn with standard Deep Q-Learning algorithms. Furthermore, as dynamic information such as velocity and acceleration cannot be inferred from a single image, requiring the agent to either train an RNN to store the state history or receive input of stacked image frames to translate to optimal actions to select. This increases the computational load on deep RL where it has to learn a representation of the temporal structure in addition to learning the optimal policy. Therefore, this leads to the motivation to decouple the reinforcement learning task into (1) environment simplification, and (2) optimal action selection.

The task of learning an optimal policy from high dimensional images has many practical applications in real-life situations where the true dynamics are often unknown. Autonomous driving using computer vision control systems is a growing area of study where sensory data is often in image form [1]. For these tasks, the ability to learn a stable control policy for safely driving a vehicle is a challenging optimization problem for deep reinforcement learning agents using large convolutional layers to process these images. Robotics is another application where deep reinforcement learning agents have been applied to train a robot arm to manipulate objects on a table from visual observations [2]. This often requires thousands of training episodes to adequately learn the task, where the agent's representation of the environment may not even be transferable to other similar tasks. Since the latent features of a state space are generally constant, there is potential to improve training efficiency with offline learning of a mapping from high-dimensional images to a simplified environment state space, then using this model to compress the visual observation into a smaller dimensional input to train a policy via reinforcement learning. By modeling the environment separately, the learned model can be transferred to other agents learning different tasks from the same environment observations.

Previous approaches to addressing this problem have use a convolutional neural network to compress the state to latent vector and then have a separate RNN to learn the temporal history, however, the representation of the dynamics is coupled with the policy learned. As a result, using the trained convolutional network and RNN to train a new policy from scratch would introduce irrelevant features for a policy to learn from without retraining the CNN and RNN when learning. Ha and Schmidhuber [3] recently introduced the "World Models" architecture which decouples of the environment representation from policy learning. This involved a visual module that maps high dimensional images to a lower dimensional code, a memory module that modeled temporal dynamics of the environment, and a simple decision-making module that determines the action an agent should take based on the compressed visual and temporal representations. Using Covariance-Matrix Adaptation evolutionary strategy (CMA) [4] 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. Using RL algorithms to train using the world model is an unexplored approach for efficiently learning a policy for a model with more parameters. In this paper, we incorporate the World Model architecture to simplify the dimensionality of the image observations received from the environment to allow a reinforcement learning agent to focus on maximizing reward using the relevant features of the environment observations.

[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`

We then evaluate the performance of such agents relative to the agent in [3] on the same car racing environment. Our approach achieves state of the art performance on this environment without requiring any domain-specific modifications to the environment outputs.

This report will be structured as follows: Section II provides a background to the field of reinforcement learning and the car racing environment as well as previous approaches to solve it; Section III outlines the architecture and approach for training the reinforcement agents using the World Model; and Section IV provides an experimental evaluation of the agents trained.

II. BACKGROUND

In this section we provide an overview of the field of Reinforcement Learning and related work in designing algorithms to maximize an agent's reward in a given environment. We will then introduce the car racing simulation environment as well as past approaches to train agents to optimally navigate this environment.

A. Reinforcement Learning

The reinforcement learning task typically involves an agent interacting with an environment $\mathcal{E}$ having a state space $\mathcal{S}$ and action space $\mathcal{A}$ for a finite number of time steps where the agent can alter its behavior upon observing the consequences of its actions [5]. At each time step t, the agent receives the current observable state $s_t \in \mathcal{S}$ from the environment and selects an action $a_t \in \mathcal{A}$ to take in state s_t. Upon receiving a_t, the environment then presents with agent with the next state $s_{t+1} \in \mathcal{S}$, a scalar value $r_t \in \mathbb{R}$ as feedback representing the reward or utility of taking action a_t in state s_t. As an agent usually interacts with the environment in finite time intervals, the agent also receives a boolean flag d_t representing whether the interaction has ended, usually due to reaching the time limit for each interaction, or from taking an action that lead to a terminal state. In this report, we will refer to the set of terms for each time step $(s_t, a_t, s_{t+1}, r_t, d_t)$ as an experience tuple, where a sequence of experience tuples from the initial time step to a terminal state is known as a rollout.

The goal of a reinforcement learning agent is to learn a mapping from states to actions in order to select the optimal actions based on the given input state such that the total reward over an entire rollout is maximized. This mapping is referred to as a policy $\pi(s) \rightarrow a \in \mathcal{A}$ where the next action to take is derived based on an input state. Such a policy is often learned by exploring the environment and tuning an initially random policy according to the reward signal observed from each action. 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 has explored value-based approaches such as Deep Q-Learning, policy-based approaches such as neuroevolution, and hybrid approaches such as Actor-Critic algorithms.

1) Deep Q-Learning: Deep Q-Learning aims to model the expected value distribution of taking each action where the action with the highest expected value is selected. The value of taking a particular action is modeled using the optimal value function $Q^*(s, a)$ known as the Bellman equation and is defined in [6] as:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{E}} \left[r_t + \gamma \max_a Q^*(s_{t+1}, a) \right] \quad (1)$$

This approximates the expected discounted long-term reward (known as the Q-value) of taking action a_t in state s_t where γ is the discount rate for future rewards and is typically between 0.9 and 0.99. Given this optimal value function, an optimal policy $\pi^*(s_t)$ will select the action with the maximum Q-value where $a_{t+1} = \pi^*(s_t) = \arg\max_a Q^*(s_t, a)$. The value function $Q(s, a)$ can be modeled using a neural network parameterized by θ denoted as $Q(s, a; \theta) \approx Q^*(s, a)$ and can be trained iteratively to minimize the following loss function $L_i(\theta_i)$ using stochastic gradient descent for the i_{th} iterative update:

$$L_i(\theta_i) = \mathbb{E}_{s \sim \mathcal{E}, a \sim \pi(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right], \quad (2)$$

where $y_i = r + \gamma \max_a Q(s', a; \theta_{i-1})$ is the target value computed using the currently observed reward r and next state s'. The Q-Learning algorithm is suited to environments with discrete actions and was able to achieve super-human performance on many Atari games where there are usually 4 possible actions [7]. However, this approach is less effective for environments with continuous action spaces due to the policy's discretization of Q-values modeled for a certain number of actions. One solution is to discretize the continuous state space [8], however, this can lead to an exponential increase in the number of discrete actions [9].

2) Policy Search: Policy search algorithms are generally gradient-free approaches that aim to find the optimal policy by evaluating the performance of a population of agents with different network parameters, then perturbing the network parameters of the best-performing agents to generate the next population to evaluate. Unlike Deep Q-Learning, this approach learns a direct mapping from states to actions without relying on finding the maximum over the action-value space. These are known as evolutionary algorithms and are effective for smaller network architectures as fewer parameters leads to a smaller policy search space over possible combinations of parameter values [5]. The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) is an effective approach for modeling non-linear stochastic functions as it uses a multivariate Gaussian distribution for parameter sampling [4]. It is popular for use in evolving neural network parameters as it is capable of searching solution spaces of up to a few thousand parameters [3]. 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.

3) Actor-Critic Methods: Actor-critic methods combine direct policy representation from policy search with gradient optimization enabled by value-based approaches. This consists of an "actor" network (policy) which is trained according to the feedback, also referred to as policy gradients, from a "critic" network (value function). Two popular actor-critic approaches explored in this paper are Deep Deterministic Policy Gradients (DDPG) [9] and Proximal Policy Optimization (PPO) [10].

For the DDPG algorithm, the critic network is trained to approximate the same optimal value function with the loss function from Deep Q-Networks where the target value is instead $y_i = r + \gamma Q(s', \pi(s'; \phi_i); \theta_{i-1})$, where $\pi(s'; \phi_i)$ is the direct policy of the actor network parameterized by ϕ. The actor is trained simultaneously to maximize the value predicted by the critic of a selecting action a_t taken in state s_t according to the following loss function [9]:

$$L_i(\phi_i) = -\mathbb{E}_{s \sim \mathcal{E}, a \sim \pi(s)} \left[Q(s, \pi(s; \phi_{i-1}); \theta_i) \right] \tag{3}$$

Recent improvements to the DDPG algorithm involve training the actor according to the predicted value of an action offset by a baseline value of the input state. This is known as the advantage and is calculated in [11] as:

$$A(s, a) = Q(s, a) - \mathbb{E}_{a' \sim \pi(s)} \left[Q(s, a') \right] \tag{4}$$

The actor can then be trained to maximize the advantage function according to a modified loss function:

$$\begin{aligned} L_i(\phi_i) = -\mathbb{E}_{s \sim \mathcal{E}, a, a' \sim \pi(s)} [&Q(s, \pi(s; \phi_{i-1}); \theta_i) \\ &- Q(s, a'; \theta_i)] \end{aligned} \tag{5}$$

For PPO, the critic network uses the optimal action value function to model the baseline state value, $V(s; \theta)$, independent of action as where $y_i = r + \gamma V(s'; \theta_{i-1})$. The actor network, $\pi(s; \phi) = \mathcal{N}(\mu(s; \phi), \sigma(\phi)^2)$, is a mapping from states to a probability distribution, over actions where the action $a_t \sim \pi(s_t; \phi_i)$ is sampled from this distribution, giving an associated probability of that action denoted $\pi_{\phi_i}(a_t|s_t)$. Then, the actor is trained to minimize the following loss function [10]:

$$L_i(\phi_i) = \mathbb{E}_t \left[\min \left(p\hat{A}(s_t), \ \text{clip}\left(p, 1-\epsilon, 1+\epsilon\right) \hat{A}(s_t) \right) \right], \tag{6}$$

where $p = \frac{\pi_{\phi_i}(a_t|s_t)}{\pi_{\phi_{i-1}}(a_t|s_t)}$ $\hat{A}(s_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the advantage estimate. This loss function aims to adjust the probability of sampling actions from the next policy π_{ϕ_i} relative to the last policy $\pi_{\phi_{i-1}}$ based on the advantage of taking that action. The derivation of this loss function is available in [10].

B. *OpenAI Gym's CarRacing-v0*

The `CarRacing-v0` environment is an open-source car racing simulator on the reinforcement learning platform OpenAI Gym [12]. The goal is for an agent to control the steering, acceleration, and braking of the car to drive it along a randomly generated circuit track in as few time steps possible. At each time step t, the state s_t 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), in which case it receives a penalty (negative reward) of -100. The agent also receives a penalty or -0.1 at each

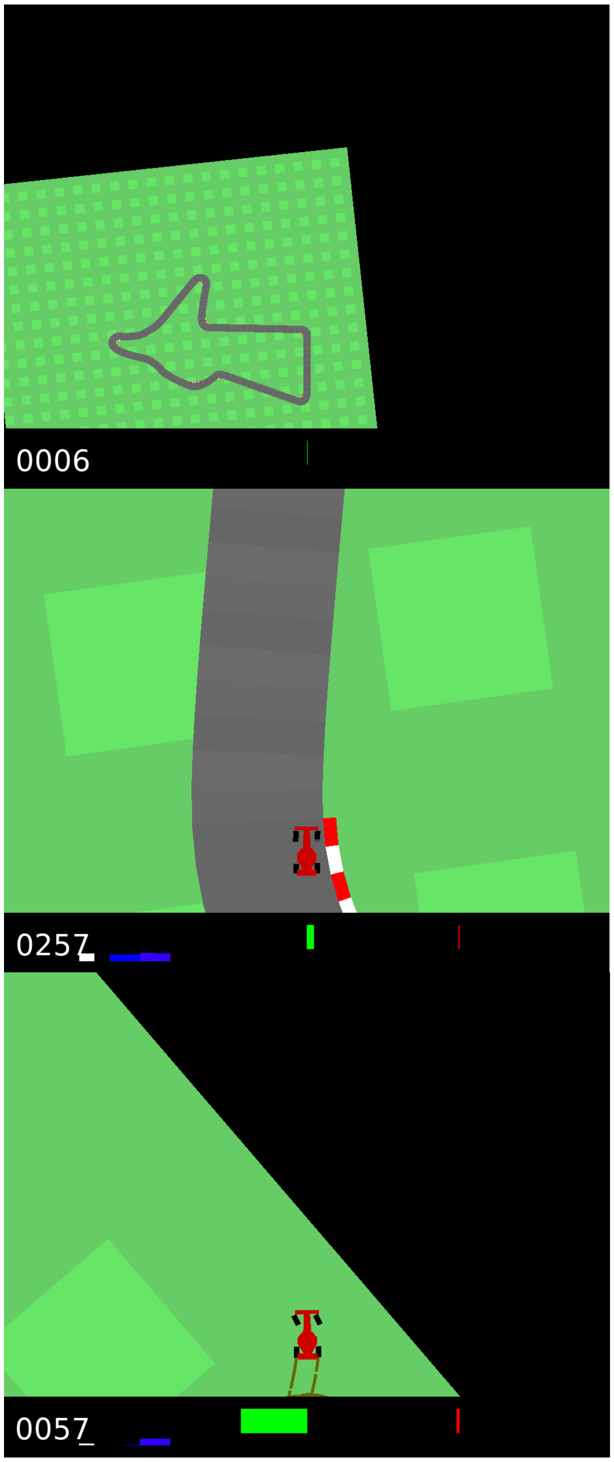

Fig. 1. Random Map (left), State Image (middle), Edge of map (right)

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. *Previous Approaches to Solve CarRacing-v0*

Recent work by Ha and Schmidhuber [3] uses Variational Auto-Encoders (VAEs) to learn a compressed latent feature vector from the image states sampled from 10000 random rollouts. This simplified representation of states in the environment is then passed through a Gaussian Mixture Density Recurrent Neural Network (MDRNN) to model the temporal dynamics of the environment (known as the World Model) by predicting the next latent vector from a given action, the current latent vector, and the cumulative internal memory state. Finally, a simple feed-forward network is trained using the CMA Evolutionary Strategy to output an action given the latent representation and the internal memory of the MDRNN. This approach is the current state of the art for solving OpenAI Gym's `CarRacing-v0` environment achieving an average score of 906 over 100 trials. In the original paper [3], it was found that the state representation of the world model trained on random rollouts may not accurately represent the reward producing states of the environment. This can be improved with iterative training of the VAE and MDRNN with additional sequences of experience tuples sampled from rollouts using a trained world model agent.

Previous work in solving `CarRacing-v0` with Deep RL has also explored using Deep Q-Networks (DQN) and DDPG. Jang et al [13] achieved a maximum score of 591 with a continuous asynchronous DDPG with edge detection preprocessing on input images. Gerber et al [14] were able to achieve an average reward of 907 with a discretized DQN using the dropout regularization method. However, both of these approaches used manual discretization of the continuous action space to 5 discrete actions based on prior knowledge of operating a car which eliminates the original challenge of the task of independently selecting the steering, acceleration and braking from a continuous range. Therefore, this paper will focus on the effectiveness of the world model architecture to simplify the input states an agent receives in order to prioritize optimal continuous action selection without manual discretization or specialized image preprocessing.

III. World Models

In this paper, we aim to reproduce the world model developed by Ha and Schmidhuber [3] for the `CarRacing-v0` environment and adapt the trained VAE and MDRNN to simplify the input state an Actor-Critic agent receives when trained online in the environment. Then, we will evaluate the performance of the Actor-Critic agent when trained with state simplification. This will be compared to the baseline for Actor-Critic agents trained from raw image states as well as a controller trained via CMA-ES from the original paper. We will also investigate the resulting improvement from a second iteration of training the VAE and MDRNN on additional policy sampled rollouts and then using this World Model to train Actor-Critic algorithms and the controller from before.

A. *VAE Model*

As the states sampled from the `CarRacing-v0` 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 [15] as shown in Figure 2. A VAE is ideal for this task as it learns to encode the most important sources of variation from the set of images trained on into a limited size representation of 32 units. We will use the same network architecture defined in the original paper [3] for both iterations of training from random and policy sampled rollouts.

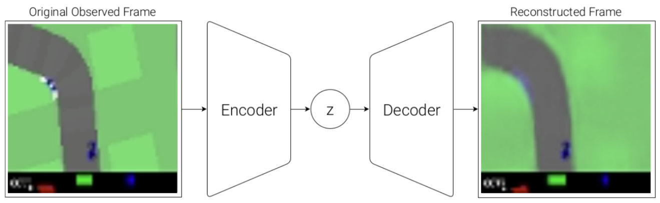

Fig. 2. Flow diagram of a VAE

B. *MDRNN Model*

While the agent can access more important feature information about a state s_t from a compressed latent vector z_t, it is only a static representation at a single point in time and doesn't contain temporal information such as the velocity of the car. Therefore, a Recurrent Neural Network (RNN) is used to encode dynamic information over a sequence of latent vectors. In [3], the RNN was used to predict the next state's latent representation z_{t+1} from the current latent vector z_t, chosen action a_t and its own internal hidden state h_t. As the distribution of the possible next states is often stochastic, the output of the RNN is passed to a Mixture-Density Network (MDN) [16], forming a Mixture-Density RNN (MDRNN) as shown in Figure 3.

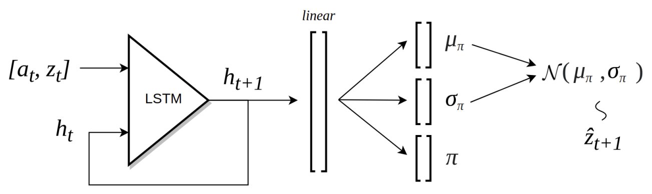

Fig. 3. Flow diagram of a MDRNN

Here, a Long-Short-Term-Memory (LSTM) network with a hidden size of 256 will be used as the RNN where the output hidden vector h_{t+1} is passed through a fully connected linear layer and then split into three 5-length vectors representing the mean (μ), standard deviation (σ), and mixing weights (π), for each of 5 Gaussian distributions forming a mixture of Gaussians according to the mixing weights. Therefore, the MDRNN can be thought of as a function $M(a_t, z_t; h_t; \beta) \rightarrow h_{t+1}$, μ, σ, π parameterized by β [17]. 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 from a trained VAE's output from 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 loss function for training the MDRNN is:

$$L_M(\beta) = -\mathbb{E}_{(z,z')\sim VAE(\mathcal{E}),\ a\sim\mathcal{A},\ (\mu,\sigma,\pi)\sim M(a,z)} \left[\log\left(\sum_i^n \pi_i\ p(z'|\mu_i, \sigma_i; \beta)\right)\right] \tag{7}$$

The MDRNN is trained at each gradient step with a batch of 16 sequences of experience tuples having a sequence length of 32. Then, for predicting the hidden state from a single experience tuple, an LSTM cell is loaded with the trained weights and used within the MDRNN cell. The trained MDRNN along with the trained VAE make up the World Model network.

C. Agent Model

The role of the agent is to map input states to actions to select in a given input state. The original state s_t produced from the environment is a 96x96x3 dimensional image which will be resized to 64x64x3, having a total of 12288 inputs. With the World Model consisting of the VAE and MDRNN, the compressed state vector $x_t = [z_t, h_t]$ is the concatenation of the latent vector z_t of size 32 from the VAE and the hidden state h_t of size 256 from the MDRNN, totaling 288 inputs. We will model three different approaches of training an agent to select actions optimally. The first will be a simplified implementation of the controller network in the original World Models paper [3] using the compressed state vector x_t. Then, DDPG and PPO agents with similar network architectures will be trained first from raw image inputs s_t as a baseline, and then using the compressed state vector x_t.

1) CMA-ES Agent: The original paper used a simple feed-forward network with a single layer mapping the compressed state input x_t directly to the action vector a_t of 3 continuous values for steering, acceleration, and braking. This can be represented as $a_t = W_c \cdot x_t + b_c$ where W_c is the weight and b_c is the bias for the single-layer network. The controller will be trained with the CMA Evolutionary Strategy with a smaller population size of 16 instead of 64 used previously in order to match the number of parallel agents per rollout across all agent models.

2) DDPG and PPO Agent: 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. Similar to the approach for learning Atari games from image inputs [7], each individual image will be converted to a single channel 64x64 grayscale frame which is then stacked on top of the previous frame to form an input size of 64x64x2, allowing for temporal information to be included in the difference between the two frames in sequence. Then, the architecture of the baseline actor and critic will consist of a convolutional component to reduce the 2D input 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 with the same architecture as the encoder network in the VAE with the final output having a size of 512 representing the compressed state, denoted as $\hat{x}_t$, of the convolutional component. This is to mimic the encoder of the VAE in the World Model which can be thought of as being trained with the rest of the network. When trained with the World Model, the convolutional component will be replaced with a single fully connected layer transforming the input x_t with 288 units 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 take the rectified (ReLU) output of the convolutional or single layer of size 512 as input. For PPO, the critic representing the state value $V(s_t)$ will pass the input to a hidden layer of size 1024 with ReLU activation followed by a second hidden layer of size 1024 with ReLU activation, then finally to an output layer of size 1, being the scalar value v_t. For DDPG, the critic representing the state-action value function $Q(s_t, a_t)$ will also take an input action a_t and pass it to a layer of size 512 before concatenating it with the state input which is then passed to the remaining layers as defined for PPO to obtain q_t. For the actor, the state input is passed through a hidden layer of size 256 with ReLU activation followed by a second hidden layer of size 256 with ReLU activation. The output to this layer will then be passed to two layers of size 3, representing the mean (μ_a) and standard deviation (σ_a) for the Gaussian parameters of the next action to take. For DDPG, the resulting action will be calculated as $a_t = \mu_a + \epsilon\sigma_a$ where $\epsilon \sim \mathcal{N}(0, I)$. For PPO, the action is sampled from the Gaussian as $a_t \sim \mathcal{N}(\mu_a, \sigma_a^2)$.

Both the DDPG and PPO agents will be trained asynchronously with 16 agents acting in 16 different environments with parallel gradient updates using a learning rate of 0.0001 as this is known to speed up training efficiency [18]. For DDPG, an experience replay buffer of max length 100000 will be used to store experience tuples which are sampled in batches of 32 for each gradient step. A Brownian noise process is used to ensure exploration where the scale of noise is reduced from 1.0 to a minimum of 0.1 at a decay rate of 0.99 after each episode. For PPO, the clip ratio ϵ is decayed from 0.1 to a minimum of 0.01 at a decay rate of 0.995 after each batch update. Gradient updates are taken every 20 time-steps with a batch size of 5 over 4 epochs. An entropy prioritization weight of 0.005 is used for exploration. The complete implementation of the networks is available on Github at [19].

IV. EXPERIMENTAL RESULTS

The experiments in this work focus on comparing the performance of a DDPG and PPO agent trained to select actions using compressed states from a World Model network to the performance of the baseline implementation of DDPG and PPO trained using the convolutional component to process raw state images. In order to evaluate the performance of the World Model for deep reinforcement learning over the current evolutionary method, we also require a baseline for the simple controller trained with CMA-ES. We will evaluate the effectiveness of the World Model trained only on 10000 random rollouts of the environment as done in [3], then retrained on 1000 random and 1000 policy rollouts using the previous iteration's trained controller network. The list of experiments is as follows:

1) Train a baseline controller with an untrained VAE and MDRNN using the CMA-ES strategy.
2) Sample 10000 rollouts using a random (Brownian noise) agent, then train the first iteration of VAE and MDRNN on experience sequences from these rollouts. Then train the first iteration controller with CMA-ES using the first iteration World Model.
3) Sample 1000 rollouts using a random (Brownian noise) agent and 1000 rollouts using the first iteration controller, then train the second iteration of VAE and MDRNN on experience sequences from these 2000 rollouts. Then train the second iteration controller with CMA-ES using the second iteration World Model.
4) Train a baseline DDPG and PPO agent with stacked grayscale images of dimension 64x64x2 as input to the convolutional component.
5) Train the DDPG and PPO with the compressed state vector of size 288 from the first iteration World Model as input to the single-layer before the feed-forward component.
6) Train the DDPG and PPO with the compressed state vector of size 288 from the second iteration World Model as input to the single-layer before the feed-forward component.

A. Results

The above experiments were run on a 3.6GHz 8-core Linux desktop with 64GB RAM and a NVIDIA RTX 2070 GPU with 8GB memory. Figure 4 below shows the best score (thin line) and rolling 100-generation average (thick line) of the population at each of the 250 generations of CMA-ES parameter evolution for training the first and second iteration controller. The green line represents the baseline from experiment 1, the blue lines represent the best and average score for the first iteration controller from experiment 2 and the red lines represent the best and average score for the second iteration controller from experiment 3. The first iteration controller achieved a maximum average of 681 and a maximum best score of 796 while the second iteration controller achieved a maximum average of 885 and a maximum best score of 909.

Figure 5 below shows the training scores (thin line) and rolling 100-rollout average (thick line) of the DDPG agents trained for 500 rollouts (left) and PPO agents trained for 250 rollouts (right). In each graph, the green lines represent the baseline from experiment 4, the blue lines represent training the agent using the first iteration World Model from experiment 5 and the red lines represent training the agent using the second iteration World Model from experiment 6. For the baseline, the DDPG agent achieved a maximum score of 929 and maximum average of 816 and the PPO agent achieved a maximum score of 212 and maximum average of 45. Using the first iteration World Model, the DDPG agent

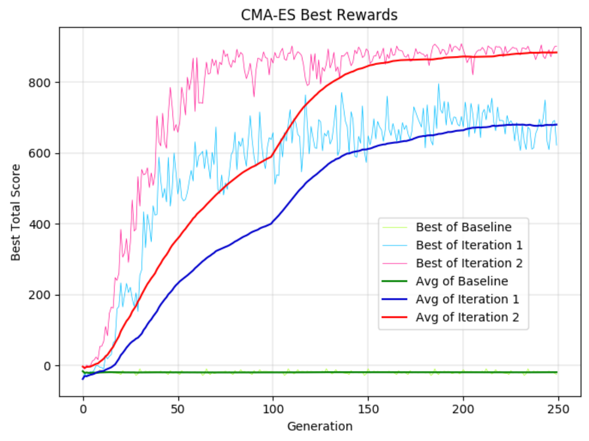

Fig. 4. Best Population Scores for CMA-ES with World Models

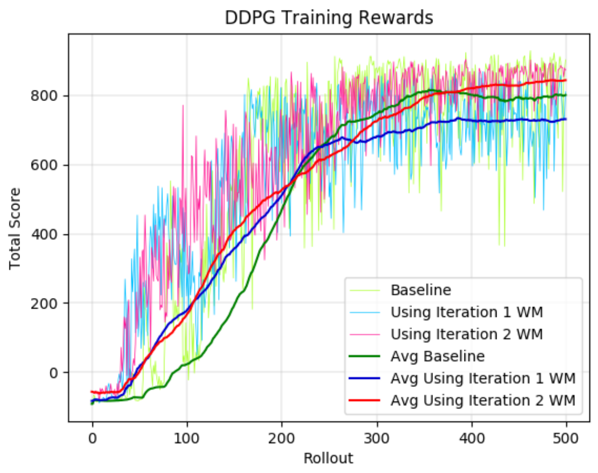

Fig. 5. DDPG Training Scores (left) and PPO Training Scores (right)

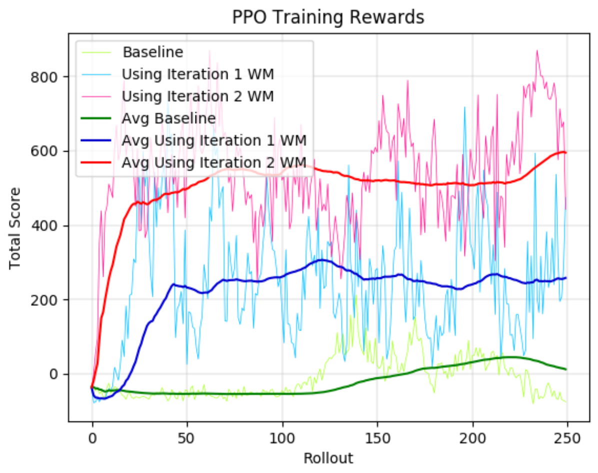

Fig. 6. DDPG Training Scores (left) and PPO Training Scores (right)

achieved a maximum score of 890 and maximum average of 734 and the PPO agent achieved a maximum score of 734 and maximum average of 306. Using the second iteration World Model, the DDPG agent achieved a maximum score of 904 and maximum average of 844 and the PPO agent achieved a maximum score of 871 and maximum average of 597.

Following the training of all agents for the baseline and both iterations of World Model, the model of each agent was saved at the point in training where it had achieved the highest score over 10 test rollouts. Then, for each saved model, the agent was evaluated over 100 consecutive trials providing a mean score and standard deviation. Table I below shows the mean ($\pm$ standard deviation) of each agent over the baseline, and using each iteration of World Model. The best average trial score for each agent is highlighted in bold.

TABLE I
AVERAGE 100-TRIAL SCORES ACROSS AGENTS

Agent	Baseline	World Model 1	World Model 2
CMA-ES	-46 $\pm$ 13	632 $\pm$ 184	**825 $\pm$ 150**
DDPG	**845 $\pm$ 167**	710 $\pm$ 206	842 $\pm$ 78
PPO	153 $\pm$ 137	739 $\pm$ 186	**812 $\pm$ 158**

B. Discussion

There is a clear difference between the relative performance of benchmarks for each agent algorithm and the best scoring trial for that agent. Considering the baseline, we found that a simple controller trained with the CMA-ES strategy using an untrained VAE and MDRNN was effectively a random agent that simply moved forward along the initial track. Comparing the first and second iteration of training the controller with a World Model, the improvement in average 100-trial score from 632 to 825 indicated that training on policy sampled rollouts was necessary to accurately represent reward bearing states. Furthermore, the second iteration was only trained on 2000 total rollouts compared to the 10000 total rollouts in the original paper [3]. While our simplified implementation for training the controller with a population size of 16 instead of 64 didn't achieve an average above 900, this may still provide a contrast in the performance variation over different training configurations.

For the DDPG agent, the baseline was able to achieve the highest 100-trial average score of 845, however when compared to the second-best performing trial averaging 842 using the second iteration World Model, we observe the score standard deviation when using the World Model is less than half of that of the baseline. This is also seen in Figure 5 where the variation in scores at each rollout is reduced using the World Model. Considering both iterations of the World Model for training the DDPG agent, Figure 5 also shows faster and more stable improvement in scores. Therefore, with a smaller state representation given by the World Model, this allows the agent to more quickly learn a mapping from the relevant state information to an optimal action. However, as the baseline was also able to control the training of the convolutional component representing the encoder of a World Model VAE, where the VAE in the World Model was fixed after initial training, this could suggest that an independently trained World Model may not encode the most relevant features for the action selection task. For future work, this could be addressed by using the policy gradients from training an agent in the environment to further optimize a pre-trained World Model for the particular task.

As PPO involves more stochastic parameter optimization, there is a large fluctuation in the training scores observed. We therefore evaluated the PPO agent using the saved parameters at the rollout that achieved the highest training score. For the baseline, the agent was unable to learn a stable policy and eventually diverges back to a random policy. This could be due to the large number of parameters from the convolutional component that needs to process the high dimensional image state input. In contrast, when using the World Model, Figure 6 shows the PPO agent was able to learn a policy within 50 episodes that was able to achieve a maximum training score of 734 and 871 using the first and second iteration World Models respectively. Therefore, the World Model has potential to significantly improve the effectiveness and efficiency of the PPO algorithm for learning complex tasks from high dimensional image states.

It can be observed in Table I that all agents using the second iteration World Model were able to achieve an average score within the 800s over 100 consecutive trials, being the highest scoring trials for that agent, except in the case of the DDPG agent which scored 3 points below its baseline. While these average scores are within 10% of the minimum threshold of 900 for solving the environment, it is important to consider the implications of this requirement. With the maximum cumulative reward of 1000 from reaching every tile, the agent also experiences a maximum cumulative negative reward of -100 (-0.1 over 1000 time steps). Therefore, in order to score above 900, the agent needs to visit every tile in a single round of the circuit due to the time limit. Ha and Schmidhuber reported in the original paper that the simple controller trained with the CMA-ES strategy was able to achieve an average of 906 $\pm$ 21, which suggests that the agent prioritized visiting every tile in the 1000 time step limit. In the case of DDPG, it was found that the learned policy would reach a local optimum where the agent would prioritize the greedy action of speeding up in order to complete the track faster. However, this would then cause the agent to cut sharp turns and miss a tile, where it was then unable to reach the missed tile in a second round of the circuit in the given time limit. In rollouts where the agent slowed down to navigate turns, it was then able to complete the track. In other cases, a small mistake would cause the car to spin, where if the agent was able to regain control, it would proceed in the opposite direction where the tiles had already been reached. In the case of PPO, since this algorithm is stochastic in nature, this increases the chance of taking a suboptimal action. These examples emphasize the challenge of the car racing environment for reinforcement learning agents, where approximating the value of each state in a rollout can lead to learning a greedy suboptimal policy. Ha and Schmidhuber refer to this as the credit assignment problem, which motivated the use of evolutionary algorithms which optimize a policy based on the total reward from a rollout. Addressing this problem in deep reinforcement learning would be an interesting area for future work.

C. Future Work

The reinforcement learning agents trained using the World Model were close to solving the `CarRacing-v0` environment. A simplified training process with 16 parallel agents was used due to time constraints, however, with more fine-tuning of hyperparameters, it may be possible to establish achieve a new state of the art in reinforcement learning algorithms for solving `CarRacing-v0` which have not been able to achieve an average 100-trial score of more than 600 without discretizing the action space [13], [14], [20].

In another area of 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, such as identifying which tiles have not been visited. Ha and Schmidhuber also suggested incorporating attention mechanisms into the MDRNN to focus on important temporal events and their consequences on future states. A recent paper incorporated visual attention within the VAE and MDRNN for a controller trained with the CMA-ES strategy [21]. This could be adapted for a deep reinforcement learning agent to train the VAE and MDRNN during policy optimization.

V. Conclusion

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 focused on training two algorithms, DDPG and PPO, on OpenAI Gym's `CarRacing-v0` environment. It was found that the World Model significantly improved the stability of the PPO algorithm, allowing it to achieve an average score of 812 using the World Model compared to 153 without it. The DDPG algorithm achieved an average score of 842 with the World Model with a standard deviation of less than half of that without the World Model. Overall, our approach achieved the highest average score for the DDPG and PPO algorithm compared to previous approaches without having to simplify the continuous action space.

Future work could focus on addressing the credit assignment problem prevalent in reinforcement learning by incorporating generalized exploration schemes or adjusting the hyperparameters based on approaches for solving similarly complex environments. 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] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," *arXiv preprint arXiv:1811.11329*, 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] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, 2018, pp. 2450–2462.

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

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

[6] 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.

[7] 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, p. 529, 2015.

[8] C. Tallec, L. Blier, and Y. Ollivier, "Making deep q-learning methods robust to time discretization," *arXiv preprint arXiv:1901.09732*, 2019.

[9] 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.

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

[11] 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.

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

[13] S. W. Jang, J. Min, and J. H. Kim, "Reinforcement car racing with a3c," *https://goo.gl/58SKBp*, 2017.

[14] D. van der Wal, B. O. K. Intelligentie, and W. Shang, "Solving openai's car racing environment with deep reinforcement learningand dropout," 2018. [Online]. Available: https://github.com/AMD-RIPS/RL-2018/blob/master/documents/nips/nips_2018.pdf

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

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

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

[18] 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.

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

[20] P. Gerber, J. Guan, E. Nunez, K. Phamdo, T. Monsoor, and N. Malaya, "Advantage actor-critic methods for carracing," 2018.

[21] P. Chadha and D. Bablani, "Incorporating attention in world models for improved dynamics modeling."