World Models

David Ha, Jurgen Schmidhubert

***Abstract*—Reflecting on the mental model humans build based on perception, the paper presents a simple reinforcement learning agent incorporated with a "World Model" which is capable of learning to model its real world in a latent space, obtained using a Generative Neural Network to learn the compressed spatial and temporal representation of the environment. The prediction of the world model coupled with the latent space observation is utilized by the agent to learn a simple and compact policy. Experiments in which the agent was trained in the real environment and policy transferred to the hallucinated environment, as well as where the agent was trained in the hallucinated environment with varying levels of complexity and the transferred policy evaluated in the real environment, were conducted and the results were analyzed. The authors also provide an interactive version of the paper which allows visualization of vectors in the latent space along with a tunable temperature parameter in the hallucinated environment (https://worldmodels.github.io).**

I. Motivation

HUMANS take their actions and decisions by making use of an internal model of their perceived environment created by our brain, either consciously or in their subconscious mind when it comes to performing fast reflexive behaviours. This is true in the case of a professional cricket player knowing exactly when and where to hit the incoming ball to even in the case where our body acts instinctively at the time of danger. Thus humans benefit from maintaining their own predictive model of their surroundings. The paper was motivated by the idea that even an AI agent could perform better upon maintaining a robust representation of past and present states of the world to create a good predictive model for the future.

II. Model

Inspired by our cognitive system, the model has been segregated into 3 components : Vision model (V), Memory model (M) and Controller (C). Upon receiving an observation from the environment at each time step :

1) **Vision Model** : Generates a lower dimensional representation of the input observation
2) **Memory Model** : Predicts the next latent state observation by using the encoded history of previous set of observations
3) **Controller Model** : Takes an action based on the encoded history from memory model and the current observation from the vision model

The action affects the environment, which produces the next observation and the cycle continues. It is to be noted that the world model comprising of the vision model and memory model has an architecture capable of capturing the necessary information about the environment so that the controller model could be kept simple and compact such as to find policies in a smaller search space. The paper states *"A small controller lets the training algorithm focus on the credit assignment problem on a small search space, while not sacrificing capacity and expressiveness via the larger world model"*. This aspect is important since sparse reward problem is common in the field of reinforcement learning and training a convoluted network with a large number of parameters in such a scenario can be cumbersome. This is possible because the proposed architecture is such that only the Controller (C) Model has access to the reward information from the environment and the Vision and Memory models have no knowledge about the actual reward signals from the environment and are trained in an unsupervised fashion.

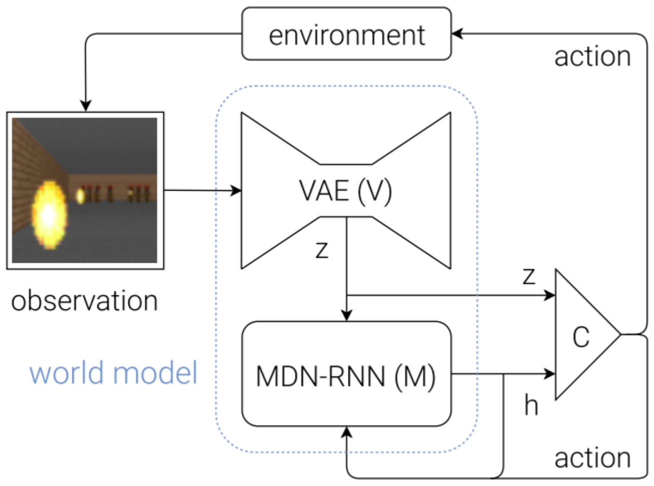

Fig. 1. Complete Architecture flow diagram: Agent comprises of Vision, Memory and Controller working together to give enhanced performance

The pseudo-code for one rollout of the agent model is as follows :

Algorithm 1 Rollout

```
1: procedure DEF ROLLOUT(controller) :
2:     obs ← env.reset()
3:     h ← Memory_Model.initial_state()
4:     done ← False
5:     cumulative_reward ← 0
6:     while not done do
7:         z ← Vision_Model.encode(obs)
8:         a ← Controller.action([z, h])
9:         obs, reward, done ← env.step(a)
10:        cumulative_reward ← cumulative_reward+reward
11:        h ← Memory_Model.forward([a, z, h])
       return cumulative_reward
```

The Vision and Memory models were separately trained beforehand using 10,000 rollouts using a random policy. The authors argue that training them separately is more practical, and also achieves satisfactory results.

III. Choice of Model Components

A. VAE as Vision Model

Variational Auto Encoder is an encoder-decoder based generative model. The encoder takes the observation as input and creates a lower dimensional representation while the decoder tries to reconstruct the input from the latent representation. VAE was chosen for the vision model since it can be used for both ABSTRACTION as well as GENERATION with a bit of applied randomness using re-parameterisation trick. This means that the decoded output is always some slight variation of the input. The encoder learns a probabilistic distribution $Q(z_t|X_t)$ over the latent variables given the input observation, while the decoder learns to model $P(\hat{X}|z)$ where $\hat{X}$ is the reconstructed output.

It is to be noted that, the encoder learns the mean and sigma of the Gaussian distribution over possible values of the latent representation. Although the enforcement of a Gaussian prior over the latent representation z was suspected to be a constraint on the information capacity of the encoding, the authors argue that it also helps reduce unrelated values of z from being generated by the Memory model.

B. MDN-RNN as Memory Model

They have used a Long Short Term Memory network coupled with a Mixture Density Network at the output of the LSTM as the architecture for the Memory model, which is responsible for keeping track of changes in space through time caused by actions taken by the agent. The LSTM takes the current state z_t along with the action executed by the agent a_t and it's own hidden representation h_t to produce h_{t+1} to have a representation of both time and space. The Memory module is responsible for predicting the next state of the environment in the latent space by modelling a probability distribution $P(z_{t+1}|a_t, z_t, h_t)$, which is sampled to give z_{t+1}. The Mixture density Network outputs the parameters for k Gaussian probability distributions (the authors have used $k = 5$), along with a probability vector, depicting contribution of each Gaussian towards the prediction of z_{t+1} (the predicted next state of the agent in latent space). In addition to the output of the LSTM, temperature parameter τ is given as input to the Memory Density Network for controlling the uncertainty in the prediction of z_{t+1}.

C. Linear Model for Controller

Controller module is responsible for predicting what action to take, inorder to maximize the expected reward of the agent during the rollout. It is a simple feed forward neural network (with a single layer : $a_t = W_c[z_t, h_t] + b_c$). It takes the current representations of observation from Vision model and the hidden representation of the Memory model as its input. Since the Memory model's prediction of z_{t+1} is produced from the LSTMs hidden representation, it would subsume the necessary information to give predictive power to the agent.

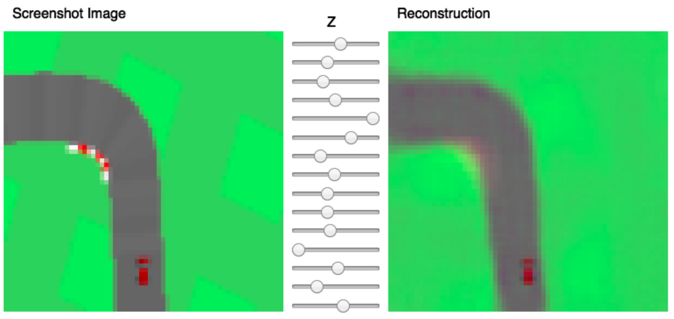

Fig. 2. Comparison of real observation from the environment v/s information retained in the encoding of Vision model for `CarRacing-v0`

The authors reason out that the simplicity of the controller helps it traverse the policy space more efficiently and is easier to optimize. The lack of capacity of the controller is made up by the information obtained from the world model which has been trained separately, implying that most of the agent's complexity resides in the World Model. This helps combat the difficulty in training large neural networks in sparse reward scenarios.

Since the Controller parameters are minimal, this allowed the authors to use Covariance Matrix Adaptation Evolution Strategy (CMA-ES), an unconventional method for optimizing the Controller. They used this evolutionary technique in order to find the set of parameters that would make the agent perform best in the environment.

IV. Experiments

A. Car Racing

`CarRacing-v0` environment generates tracks randomly and the agent has control over 3 continuous actions : steering left/right, acceleration, and brake. The agent is rewarded for visiting as many tiles as possible in the least amount of time. This particular game is considered challenging because the task involves getting an average reward of 900 over 100 consecutive trials inorder to be solved. The Vision Model (VAE) has been used to visualize (Fig. 2) the amount of information available to the agent while making decisions.

They experimented using 3 different models for the controller to verify the benefits of the world model:

- V Model : Here, the controller has access to information from only the Vision model and the the controller is a single linear layer ($a_t = W_c[z_t, h_t] + b_c$). The agent was found to wobble around and miss the track on sharp turns.
- V Model with Hidden Layer : Here, the previously single layered controller is incorporated with a hidden layer, with information available only from the Vision model. Although the performance improved, it wasn't enough to satisfy the requirement for the challenge to be declared as solved.
- Full World Model : The agent was given inputs from both Vision model as well as the Memory model. This improved the performance of the agent since it was given a predictive power in addition to the the information about

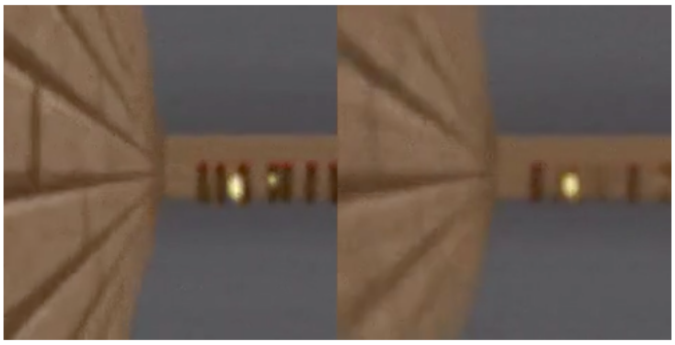
Fig. 3. Comparison of real observation from the environment v/s information retained in the encoding of Vision model for `VizDoom: Take Cover`

the current state. The driving was found to be more stable and the agent was able to effectively take sharp turns.

Inorder to evaluate the effectiveness of the model, the predictive capacity of the model was tested by utilising the the predictions for creating a hallucinated environment which they referred to as being analogous to the concept of dreams and called it *Car Racing Dreams*. The current action a_t and the observation z_t was used to produce the next observation z_{t+1} and the trained controller's policies were evaluated on this virtually generated environment. The dream environment was evaluated for various values of the temperature parameter inorder to study the agent's behaviour (upon transferring the policy learned in the real environment) as the uncertainty in the dream environment changes.

B. Vizdoom

Inorder to test the architecture's capabilities further, they trained the agent inside its own dream(generated by the world model) and transferred that policy into the actual environment. In short, they trained the AI inside a simulated latent space dream world.

They used the game `VizDoom: Take Cover` for the purpose of their experiments. This game requires the agent to stay alive for the maximum time steps as possible, with monsters shooting fire balls with the sole motive of killing the agent, and will be rewarded accordingly (Fig. 3). The agent has access to 3 discrete actions : move left, move right and stay put. The task involves the agent managing an average survival time of atleast 750 time steps over 100 consecutive rollouts, to be considered solved. In this environment, in addition to the next observation prediction, the model also needs to predict whether the agent will die or not. Hence the Memory model is trained to predict $P(z_{t+1}, d_{t+1}|a_t, z_t, h_t)$.

The agent is trained entirely in the latent space environment and the policy is transferred to the real environment. This task requires the model to learn all aspects of the real game environment including alignment of the the directional pathways of the fireballs as the agent moves in the environment as well as keeping track of the activities of all the monsters and restraining the agent within the 2 boundary walls.

Inorder to evaluate the policies learned by the agent as the level of difficulty of the task changes, they varied the temperature parameter τ (during the sampling process of z_{t+1}) to introduce uncertainty into the environment as well as to compensate for the imperfections in the learned model. The results of the same can be found in Table I.

TABLE I
TAKE COVER SCORES AT VARIOUS TEMPERATURE SETTINGS

Temperature τ	Virtual Score	Actual Score
0.10	2086 ± 140	193 ± 58
0.50	2060 ± 277	196 ± 50
1.00	1145 ± 690	868 ± 511
1.15	918 ± 546	1092 ± 556
1.30	732 ± 269	753 ± 139

The dream environment can thus be made more difficult than the real environment by increasing the temperature parameter. As τ increases, the incoming fireballs started taking random trajectories, the number of monsters became ambiguous and in certain rollouts, then agent was found to be dead without any explanation. Hence, as can be seen from Table I, the agent can learn policies that in fact perform better when transferred to the real environment.

An interesting aspect of the experiment to notice is that the agent started to learn adversarial policies by the utilisation of the imperfections in the hallucinated environment. But these policies probably won't work in the real environment since those states are away from the training distribution. For example, in certain rollouts, the agent learned to move in such a way that none of the monsters fire or magically extinguish incoming fireballs. The fact that the agent has access to the internal states and memory of the game model through the hidden representation of the Memory model might be a crucial reason for the same. This gives the agent an unfair advantage over human players who learn to play the game by the use of only observations as the input.

The authors have argued that making the dream environment stochastic and more uncertain with respect to the real environment can help combat some of these issues and hence control the trade-off between realism and exploitability.

V. ITERATIVE TRAINING PROCEDURE

In more difficult tasks where the controller needs to explore more parts of the environment as the game progresses, the authors have also proposed an iterative training procedure :

Algorithm 2 Iterative Training Procedure

```
1: Initialize M, C with random model parameters
2: Rollout to actual environment N times
3: Save all actions $a_t$ and observations $x_t$ during rollouts
4: Train M to model $P(x_{t+1}, r_{t+1}, a_{t+1} d_{t+1}|x_t, a_t, h_t)$
5: Train C to optimize expected rewards inside of M
6: Go back to (2) if task has not been completed
```

This procedure requires the Memory model to predict the next action and reward in addition to the next state and whether the agent is alive or not. This method is useful in sophisticated Environments where the agent needs to strategically navigate to parts of the environment that are revealed at a time. This can also be thought of as curriculum Learning, allowing the CM model to develop a natural hierarchical way of learning.

VI. Related Work

A. Hallucination with RNN

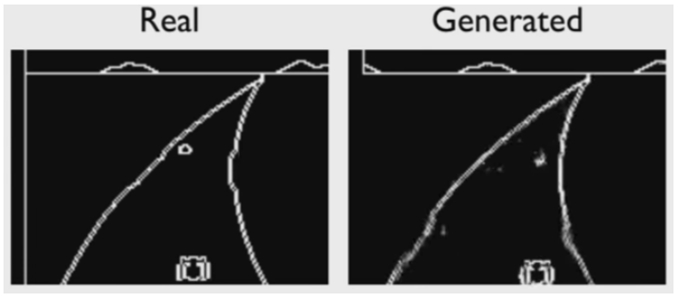

Fig. 4. Probabilistic model of Enduro (Atari) learned using an RNN

During a guest lecture by Alex Graves of Google Deepmind at the University of OXFORD, he demonstrated the visual predictions of the game using a trained RNN on the game of Enduro, a racing video game, a simpler domain than `CarRacing-v0`. The trained RNN learned the structure of the game and could hallucinate similar game levels on its own. (Link to the lecture : https://www.youtube.com/watch?v=-yX1SYeDHbg&t=49m33s)

B. From Pixels to Torques

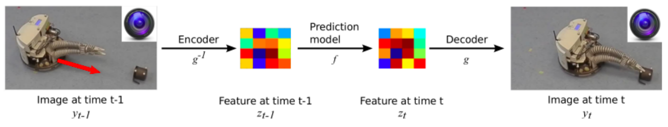

Fig. 5. Feature learning and model prediction using reduced feature space

The paper talks about an architecture capable of learning in a lower dimensional feature space and predicting the future and reconstructing the predicted observation from the latent space. The agent learns a closed-loop control policy solely from the pixel information. They also utilize deep autoencoders to learn a low-dimensional embedding of images jointly with a predictive model such as the proposed model above. But their architecture is end-to-end trainable and this joint learning ensures that not only static but also dynamic properties of the data are accounted for in the learned feature representation in the latent space.

C. PILCO

Probabilistic Inference for Learning COntrol, abbreviated as PILCO [2] has a similar concept as the temperature parameter utilized in [5] for adding uncertainty into the environment and making it stochastic. This paper involves learning a probabilistic dynamics model and explicitly incorporating model uncertainty into long-term planning. They use this to reduce the model bias and also cope with very little data (credit assignment problem). Here, a Gaussian process (GP) model is used to learn the system dynamics, and sample trajectories for training the controller.

VII. Conclusion

The paper is very well written with an available interactive version of the paper online. But it is to be noted that the authors could have experimented with an end-to-end training of the complete model which would enable the Vision and Memory models to encode more task-relevant features. In the current scenario, the dynamics of the environment are not taken into account to the full potential. Another aspect that must be paid attention to is the fact that the temperature parameter τ is a tunable hyperparameter that has significant effects on the performance of the agent in the virtual environment, especially when the training occurs in the dream environment. The controller has an unfair advantage of having access to the hidden states of the game which also has an adverse effect of promoting the learning of adversarial policies. But the paper introduces a very strong idea that we can train the agent in a latent space environment generated by its own model and not depend on expensive game engines, as long as we have generated rollout trajectories from the actual game that can be utilized to train the world model. This essentially means that once we have a trained Vision model that can encode observations and a Memory model that can predict the environmental changes following an action by utilization of knowledge of previous history, we can train the agent using the Memory model instead of the actual game and also visualize the process using the Vision model.

References

[1] Christopher M Bishop. *Mixture density networks*. Tech. rep. Citeseer, 1994.

[2] Marc Deisenroth and Carl E Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472.

[3] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908* (2016).

[4] David Ha and Douglas Eck. "A neural representation of sketch drawings". In: *arXiv preprint arXiv:1704.03477* (2017).

[5] David Ha and Jürgen Schmidhuber. "World models. arXiv preprint". In: *arXiv preprint arXiv:1803.10122* (2018).

[6] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[7] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes, 2013". In: *URL https://arxiv.org/abs/1312.6114* (2013).

[8] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. "From pixels to torques: Policy learning with deep dynamical models". In: *arXiv preprint arXiv:1502.02251* (2015).