Semi-Supervised World Models

Shahd Safarani*, Yiyao Wei*

*{8safaran, 9wei}@informatik.uni-hamburg.de
Knowledge Processing Seminar, WTM
Informatics, University of Hamburg

***Abstract*—A world model is a model-based approach to reinforcement learning, consisting of three separate models (V, M and C) responsible for feature extraction, future prediction and planning. C is a small linear controller trained to perform the task by using the compressed representation of the world from V and representation of time from M. Furthermore, future events generated by M can substitute real observations to fully train C which is called training-in-the-dream. That structure enables the agent to learn inside its own hallucinated dream. In this paper, we compare the performance of different versions of world models, including a novel combined one that leverages AutoEncoders and Echo-State-Networks (ESN) for modeling the environment as well as PPO for policy optimization, referred to as *VRC*. It is shown that VRC can achieve a decent score (762±288), even when it uses untrained RNN. Also, this work showcases that ESNs can be used for RL tasks. To this end, different world model versions are compared and investigated to see how each component [V,M,C] contributes to the general performance of the world model including stability of learning, resources, reaching good scores.**

I. Introduction

Reinforcement learning (RL) is one type of learning theories from the perspective of psychology. Artificial intelligence pioneers brought the idea of RL to design artificial learning systems that can solve complex tasks which are used to be seen as only solvable by human intelligence. In the past two decades, benefiting from rapid improvements and the growing accessibility and development in computational power, RL researchers proved that RL algorithms in combination with deep learning can master tasks which require human-level intelligence such as classic 1-player Atari games [11], multiplayer video game Dota 2 [1], and the high-complexity board game GO [14]. However, these algorithms highly rely on huge computational power.

To reduce the computational cost of training an RL agent and enhance its efficiency, researchers turn their attention to the cognitive neuroscience domain to find inspiration. Dreaming about real events is a cognitive property that takes place in many living beings. It helps them process such events via imagination to handle them better when they re-occur. Recent literature has shown that dreaming can be applicable to reinforcement learning problems and can improve the agent's behavior in the real environment. Furthermore, it turned out that training can happen while dreaming with no requirement for the interaction with an actual environment.

[6] introduces a model-based approach, so-called *World Models*, that hands out specific duties of solving the RL task to three separate components. The first and second components combined represent a "dream" model that can simulate the dynamics of the environment in a low-dimensional space since they are generative models of the actual environment. In summary, the framework can achieve state-of-the-art performance in a continuous control task while maintaining a high level of efficiency and robustness.

In this work, we are interested in investigating the performance of the existing frameworks that support the training-in-the-dream concept. In addition, we propose an additional novel framework as a combination of previous works and test its performance in comparison to the other approaches.

II. Related Work

Ha and Schmidhuber claim that an RL agent can achieve the start-of-the-art score in a fast and cheap way, even can train entirely inside its dream. They introduce the first world model approach in [6]. In a follow-up study, Tallec et al. investigate the importance of RNN when training a world model to master the `CarRacing-v0` environment, and they report that the training of the MDN-RNN does not play an important role in learning [15]. Surprisingly, the performance of the full world model architecture with a trained MDR-RNN (860 ± 120) is even worse than the performance when we do not train it (870 ± 120) [15]. Their interpretation of this counter-intuitive result is that the agent is able to learn some dynamics of the environment from recurrent states, although the recurrent model is incapable of predicting the next state. That implies that using only the temporal information from a random weight-fixed RNN is sufficient to learn a comparable strategy on the task.

This observation is also evident in the work of Chang et al. when using a CNN and an ESN whose internal weights are frozen during the training process. [3] argues, their model achieves state-of-the-art results in the `CarRacing-v0` environment as well. This raises the question of whether training the "dream" model yields benefits for learning an optimal policy.

III. Background

A. Reservoir computing

Reservoir computing [10] refers to a family of RNNs with random fixed weights. Echo state network (ESN) is a kind of reservoir computing. Unlike the normal RNNs that are highly expressive, ESNs contain random fixed weights. In other words, ESN is an efficient approach for processing time sequence inputs since the weights are randomly assigned and do not require optimization, unlike large RNNs that have millions of parameters need to be updated. On the other hand, the randomness of its connectivity and weight parameters have also been criticized.

Reservoir computing models need an extra readout layer for solving any task. In this case, future state prediction needs a linear regression layer and action selection (controller) needs an extra layer that uses an optimization algorithm for learning an optimal policy, e.g. in [3]. For more details about ESNs, refer to [10] and [3] for their RL applications.

B. World models

World models are generative models of gaming environments. The main contribution of world model is that it introduces a model-based approach to RL problems. A typical world model consists of three components, which are responsible for different jobs. They are also trained separately in a sequence. What those parts do and how they work together are explained below. First, let us give a name for them, as shown in Figure 1.

- **Vision (V)** for environment feature extraction.
- **Memory (M)** for future latent vectors prediction.
- **Controller (C)** for action selection.

In the world model framework, V model serves as the integration of encoding and decoding. It takes the pixel images of every time step from the environment and compresses these high-dimensional inputs into a low-dimensional latent space $z \in \mathcal{R}^{32}$ (i.e. the shape of the road, left/right turn, etc.). On the other hand, V model also serves as a decoder that reconstructs the original RGB frame from the compressed latent vector. Often time, to reconstruct the environment for later analysis is a good idea for human researchers.

After the encoder outputs the sequence of latent vectors, M model uses the past observations given by z to predict the probability distribution of the latent vectors for the next frame. The input fed to M model is the output of V model which could be in form of z vectors or flattened/raw extracted features by V. Inside M we have a hidden state h, which is the representation of time because it is updated dependently on past and current observations. The predicted probability distribution $p(z_{t+1})$ is defined by the following function

$$P(z_{t+1}|a_t, z_t, h_t)$$

where a_t is the action taken of the agent, z_t is the latent vector of the environment, and h_t is the hidden state of the RNN.

Finally, we feed the low dimensional representation of space z (or raw features output by V model) and the low dimensional representation of time h into the small linear controller C to find the action that can maximize the accumulated reward. In other words, the simple linear combination matrix C outputs the next best action by using the information about the environment learned from V and M in a long time range. Let a_t be the action vector at time t, the following formula represents how the linear model produces it.

$$a_t = W_c[z_t h_t] + b_c$$

Here, the input vector is $[z_t h_t]$ where z_t is the latent vector from V and h_t is the hidden state from M at each time step. The weight matrix W_c maps $[z_t h_t]$ with bias vector b_c to compute a_t.

Table I shows the parameter counts for each component. Apparently, the majority of the complexity takes place in V and M. Also, the controller is indeed very small.

Note that the number of parameters is positively proportional to the complexity of the environment. Here we get the Statistical data and the equations from the simple `CarRacing-v0` environment since we also use it in this paper. If a new degree is introduced in a more challenging task, the setup of V, M and C should be redefined accordingly. For instance, the setup of V dimensionality for `VizDoom` environment is adjusted to $z \in \mathcal{R}^{64}$ in [6].

Furthermore, the prediction of the world (z_{t+1}) generated by M can be used as the input to train the controller C. Thus, we say the agent can learn how to solve the task in its own hallucination. More interestingly, with hallucinated training inside the dream, the agent can also solve the actual environment, which implies that the policy learned in the generated environment can be transferred back into the real world.

Altogether, we have introduced the general idea of world models, but the concrete example of each model is still unclear. You can find the details of two world models that are highly related to our proposed model below.

Table I: Number of parameters of each component [6].

Model	Parameter Count
VAE (V)	4,348,547
MND-RNN (M)	422,368
Controller (C)	867

1) VM model: The VM model [6] is the original world model that uses VAE, MDN-RNN, and a simple linear combination as the instantiation of V, M, and C.

Variational Auto-Encoder (VAE) [9] is a stochastic variational inference that follows an encoding-decoding scheme. In the middle of the structure of VAE, a latent vector with a much smaller dimension ($z \in \mathcal{R}^{32}$) is generated by matching each of its elements to a unit Gaussian distribution. The Gaussian sampling step makes VAE very efficient while preserving the richness of information stored in latent variables.

MDN-RNN stands for Mixture-Density Recurrent Neural Network. It is a newly discovered sequence model that can generate continuous signals follows a Gaussian mixture model at each time step [5][4]. In essence, MDN-RNN is a recurrent neural network using mixture density network as the output layer for temporal feature extraction and future events predication.

To find the optimal parameters in the controller C in a fast and cheap manner, [6] uses Covariance-Matrix Adaptation evolution strategy algorithm (CMA-ES) [7] for action selection. The evolution strategy (ES) is a family of black-box optimization technique. For the comparison between ES and standard RL methods, see [12].

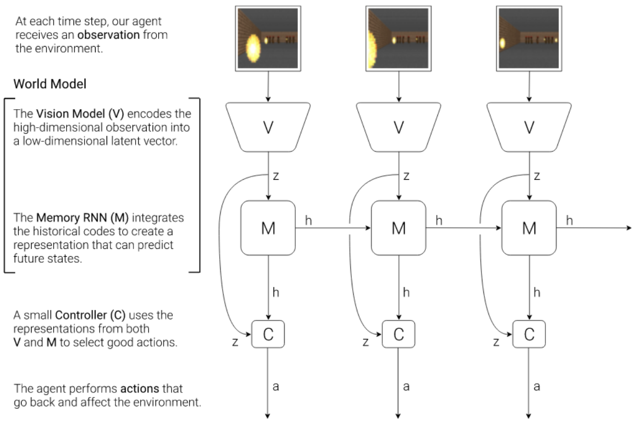
At each time step, our agent receives an observation from the environment.
World Model
The Vision Model (V) encodes the high-dimensional observation into a low-dimensional latent vector.
The Memory RNN (M) integrates the historical codes to create a representation that can predict future states.
A small Controller (C) uses the representations from both V and M to select good actions.
The agent performs actions that go back and affect the environment.
V
z
M
h
C
a

Figure 1: An overview of the structure of the VM model [6].

2) RCRC model: RCRC stands for reinforcement learning with convolutional reservoir computing which is provided by [3]. As you can see in Figure 2, RCRC uses a random fixed weight layer as V and M for feature and temporal extraction. A CNN with depth of three is used to learn the representation of the space in addition to an ESN for extracting temporal features. In the experiments of this paper, the size of filters is slightly modified to 32, 8, 2 in that order. The combination of the random fixed CNN and ESN can extract features very fast since it is only a feedforward layer (training is not required) and can be seen as a simple linear combination matrix. For optimizing the weights in C, CMA-ES is used and trained while feeding the output of the convolutional reservoir layer as C's input.

IV. Novel Combined Approach

We propose a modified version of world models. Instead of using MDN-RNN, we use ESN as M. In other words, MDN-RNN is simply replaced with a random fixed-weights RNN as in the RCRC model. As mentioned in Section III-A, one advantage of ESN is the low cost since weights are fixed

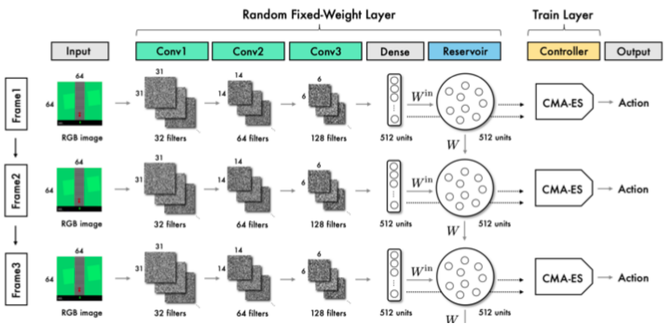

Figure 2: An overview of the structure of the RCRC model [3].

and also initialized in an efficient way to present very rich dynamics.

Our hypothesis is that this world model version can further reduce the computational cost of training an RL algorithm in a continuous environment because there is no need to train the RNN. Still, the trained VAE is supposed to stabilize and guide the processing within the ESN to output future representations that convey information about the actual environment. Since C can be any policy optimization algorithm, Proximal Policy Optimization Algorithms (PPO) [13] is used for C in our proposed model. Because CMA-ES was always leveraged for policy optimization in world models and we aim to leverage a new optimization method for C, PPO is selected for C as it proved to be robust and well-performing in many reinforcement tasks.

In short, this model is an additional version we want to investigate and has VAE, ESN, and PPO as the instantiation of V, M, and C. To differentiate it from the other world model versions, it is called *VRC*, referring to VAE Reservoir Computing model.

V. Experimental Studies

In order to investigate the performance of all three versions of world models (Vm, RCRC and VRC), several experiments are conducted. These models are applied to the `CarRacing-v0` environment which is available through the OpenAI Gym Framework [2]. The detail experimental procedure is in Appendix VIII. We open-source the code used to run experiments in this paper, see *github.com/Shahdsaf/Semi-Supervised-World-Models*.

VI. Results

VM model training is repeated for two separate runs. In both runs the model is stuck and reaches a maximum score of less than 400, despite of training over 10000 episodes for each run. Meanwhile, RCRC model reached a maximum score of 835 in its first run after 16k episodes but this learning process could not be reproduced after conducting 4 more separate runs. This may imply that the learning process of RCRC is highly dependent on the initial conditions (initial

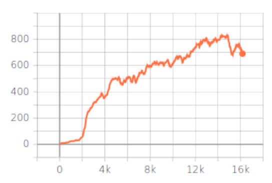

Figure 3: RCRC learning curve of the best run.

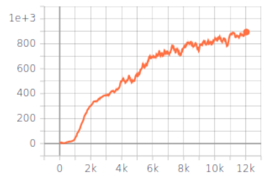

Figure 4: VRC learning curve of the best run.

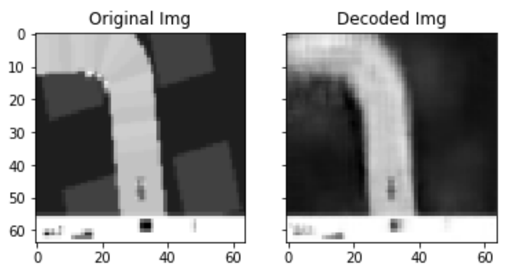

Figure 5: Example of VAE's decoding.

weights etc.) which makes it unstable and requiring several runs to obtain a good result as shown in Table II.

Regarding our additional proposed VRC model, the learning curve is constantly increasing and reflects more stable and smooth learning process than in the case of RCRC, as shown in Figures 3 and 4, where the y-axis shows moving averaged score while x-axis for the number of episodes. In addition, it is noticed that VRC could reach a maximum score of 900 only after 12k episodes which also indicates that VRC learns faster than RCRC straightforwardly reaching the maximum score. Another run is executed for VRC which asserted the previously mentioned observations.

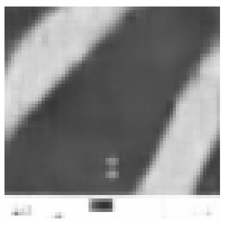
Figure 6: ESN predicted state, decoded via VAE.

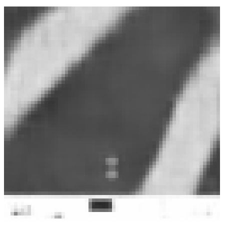
Figure 7: Original Future state.

Table II: `CarRacing-v0` scores that different versions of world model achieved.

Model	Avg. Score
VM	446 ± 101
RCRC	852 ± 221
VRC	762 ± 288

Those results indicate that having untrained RNN in the world model (ESN in RCRC and VRC) can achieve quite decent results. On the other hand, utilizing a trained VAE (a CNN in general) preserves the stability of the agent's policy learning process. The reason maybe due to the extraction of features that are more related to the actual environment which guides C's training in an efficient and informative manner. In order for VRC to achieve better average test score, it is believed that training the VRC model slightly longer would boost its performance and make it more robust and well-performing.

Investigating the low performance of VM, VAE could encode and decode frames quite well as shown in Figure 5. Also, MDN-RNN is checked and could predict future states well. Furthermore, the implementation of PPO was checked that it works well with no bugs before using its source code. In addition, PPO works well with RCRC and VRC, so the question is left open, why VM performs so badly with PPO.

Training-in-the-dream: due to time constraints, training in simulated dream environment could not be fully finalized and applied to RCRC or VRC models. In order to build such an environment, the world model should be able to predict reward signals and future states. For RCRC and VRC, this can be achieved by a linear regressive layer on the top of ESN to train it, given the outputs of the ESN and VAE/CNN concatenated.

In Figures 6 and 7, it is demonstrated that ESNs are capable of future state prediction when using VRC world model. Developing the full train-in-the-dream concept with VRC is left as future work.

VII. Conclusion

In this work, our main contribution is to test the ability of ESN in the RL domain. We show that trained RNN is not necessary in world models as well as PPO can be used as an alternative to CMA-ES for the controller. In addition, it is noticed that adding a pre-trained VAE/CNN to RCRC yields stability and robustness when training the agent C. In other words, VRC version can be seen as more optimal than the original world model version [6], as supported by [3] and [15]. Finally, this work argues that reservoir computing is an underestimated area of research.

References

[1] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
[3] Hanten Chang and Katsuya Futagami. Convolutional reservoir computing for world models. *arXiv preprint arXiv:1907.08040*, 2019.
[4] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
[5] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.
[6] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[7] Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[12] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[14] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[15] Corentin Tallec, Léonard Blier, and Diviyan Kalainathan. Reproducing ”world models”: Is training the recurrent network really needed ? https://ctallec.github.io/world-models, 2019.

VIII. Appendix

A. *Experimental Setups*

In this section, the experiment design is explained in addition to model configurations used for comparing the performance.

1) Environment: We test our agent in `CarRacing-v0`, a top-down racing environment. It is a simple example of learning a continuous task from pixels and is used as the benchmark task in the related literature [6] and [3], thus, the perfect choice for our experiments. The state space of `CarRacing-v0` is $96 \times 96 \times 3$, while the action space is 3-dimensional representing steer, gas, and brake. The default value of reward in every frame is -0.1. When the agent has reached the destination, which means that every track tile has been visited, $+1000/N$ will be added to the total rewards, where N is the total number of tiles in the track. For instance, if the agent solves the track in 700 frames, the final reward will be $1000 - 0.1 * 700 = 930$.

To restrict the learning process to be more efficient, a GymWrapper is integrated into the original `CarRacing-v0` environment. GymWrapper is a class that wraps the original environment class to overwrite functions or add some customized steps on the top of the environment functions. In our wrapper, the state of the environment is defined to be a stack of the three last frames (3 channels of gray-scaled frames resized to 64×64) instead of having one RGB-colored frame at a time. This conveys more useful information about motion, direction, etc. Furthermore, the wrapper penalizes for being in the green space (outside the track) and ends the episode if there is no recent reward. Finally, the total reward per step is the summation of that action repeated 8 times. This way of calculating the total reward reflects more the continuity and stochasticity of the environment. In addition, the training process of the reinforcement learning agent becomes easier and more smooth.

2) Target Models: We aim to compare the performance of different versions of world models on the `CarRacing-v0` environment in order to determine which version is best performing and more efficient. Also, we want to address which components in the world model contribute the most to its performance. For all models, we changed CMA-ES to PPO due to the limited computational resources. Hence, these versions vary in V, M or both. More specifically, the comparison includes VM, RCRC, and VRC.

Hyperparameters: PPO's hyperparameters are set to default values: max norm gradient equals 0.5, clipping parameter is 0.1 in addition to having a memory buffer that represents an experience replay. That buffer has a capacity of 2000 step while the maximum length for an episode is 1000 steps. Each time the buffer is full, the PPO agent is trained and updated for 10 epochs using Adam optimizer [8] and a learning rate of 0.001. Configurations related to VRC, RCRC and VM models such as the size of ESN or filter size of VAE are set up similarly to their original default values as mentioned in [6], [15], [3] etc. Except that filter sizes of the CNN in RCRC are slightly modified to 32, 8 and 2 in order.

B. *Measurement*

All experiments run on a RTX 2060 GPU. 2-5 separate training runs for at least 10k episodes are executed for each version. Repeating the training process multiple times provides information about how the learning process, whether it reaches the same results each run, for how long it lasts, etc. More runs are tried out when there is a clear difference every time the world model version is trained because we want to see how runs are different and why. However, when the same learning pattern repeats itself given one version, there is no need for more runs.

Regarding each version, the best run is determined by the training curve and the maximum score it reached. The best model of each world model version is tested by running it in the actual environment for 100 episodes while randomly initializing the track every time. The final score averaged over those 100 episodes is calculated and shown in Table II.