Developing Improved Perception for Reinforcement Learning Agents in Complex Environments

D. Malathi[1*] Sankalp Sanand[2]
[1] *Professor, Department of Computer science and Engineering, SRM Institute of Science and Technology, Kattankulathur – 603 203*
[2] *B.Tech Student, Department of Computer science and Engineering, SRM institute of Science and Technology, Kattankulathur – 603 203*
[*] *Corresponding Author Mail ID:malathi.d@ktr.srmuniv.ac.in*

Abstract

In this project, the plan was to develop a method of perception for reinforcement learning agents which may allow them to operate within seemingly complex environments. In a recent paper, one such architecture of perception was proposed called the World Models. This method constitutes of three components namely, Vision(V), Memory(M), and Controller(C) models. The V model uses a variational autoencoder made from of CNNs to create a compressed representation of the environment. The M model uses mixture density networks made from RNNs in order to give the whole architecture the ability to learn from its previous states. The C model uses evolutionary strategies to control the agents actions according to the repre- sentation it receives from the previous two models. This whole architecture has already been implemented on two relatively simpler gaming environments, CarRacing and DoomTakeCover, both by the creators of the World Models. Implementing this in a more complex environment and observing the results obtained is the objective of this project.

Keywords: *Reinforcement Learning, Convolutional Neural Network, World Models, Sonic's environment, Doomtakecover*

I. INTRODUCTION

Humans have remained an inspiration for designing ma- chines which mimic their sophistication and perform highly complex generalized as well as specific tasks with equal ease. This level of efficiency which is inherently present in us comes from a highly advanced perception model which we take for granted. How we perceive our environment plays a major role in evaluation of our decisions that we are planning to take. This is why in a recent paper[1] a methodology of training reinforcement learning agents using a similar approach was proposed. This architecture uses 3 separate but connected components, collectively known as the World Models.

A. World Models

The underlying idea of this architecture is quite simple. We humans tend to create a high-level abstract representation of the environment and its dynamics inside of our mind and utilise that representation to judge our surroundings. More often than not, we also tend to perceive time spatially, for example, when a football is coming towards us we observe how close the ball is getting towards us instead of measuring the seconds remaining till it reaches us. These aspects in combination with an attention mechanism and long term dependency modelling of human perception provided inspiration for this architecture.
The three components introduced are Vision, Memory, and Controller model which combine to for the World Models. These are explained below:
1) Vision: The Vision model receives the original raw environment images or frames as input and tries to create a similar but compressed representation of that frame. It learns

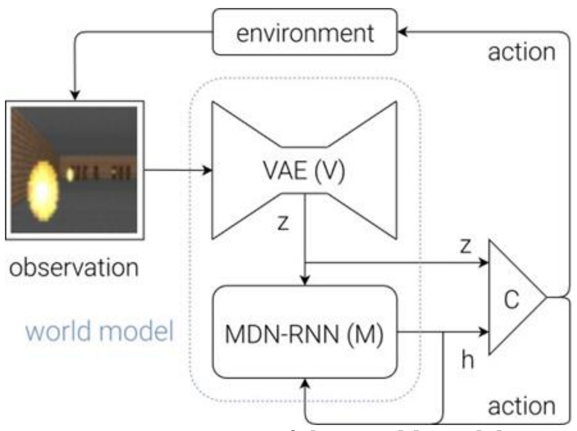

Fig. 1. Overview of the World Models

what parts of the frame are necessary and what is being repeated in order to create a latent vector which will be fed to the components in further steps.

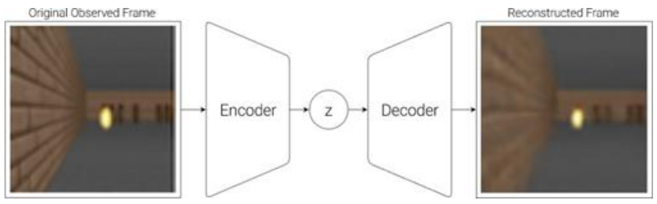

Fig. 2. Variational Autoencoder encoding the original image into a latent vector z and decoding z to recreate that particular frame

This learning takes place through a Variational Autoen- coder which first encodes the frame obtained to a low dimensional latent vector z. Then it tries to recreate the original frame from that vector z and compares the original with the recreated version while adjusting the weights in its neural network. CNN are an obvious choice in this case since the operations are being performed on an image. This z vector is then provided as input to the Memory model.

2) *Memory:* The Memory model learns about the changes in the environment as a result of the action performed and keeps those in the agent's memory so that it can learn from those alterations and improve its decision making process. That information about modifications and its subsequent re- sults is stored in the hidden variable h which is concatenated with z in the later stages and sent to the Controller model.

3) This component is used as a predictive model of future z vectors that will be produced by the Vision model, and thus it gradually learns to predict the content of the next frame. Since most of the complex environments are stochastic in nature, this model produces a probability density distribution instead of a deterministic value of z.

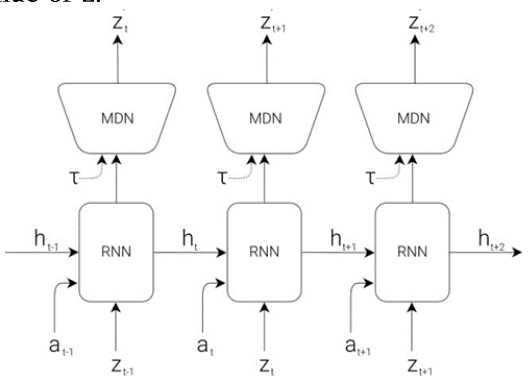

Fig. 3. Memory model uses RNNs in combination with Mixture Density Neural Networks for remembering earlier events

As there are several shortcomings of basic RNNs, for example, they are not able to model long term dependencies and perform poorly if the context had occurred far back in the sequence, an LSTM is used to efficiently address this issue. By doing this, long term dependencies can now be taken into account by the network and the problem of vanishing gradient will also be solved.

Since the agent now has a way to learn from its previous states and predict what might happen in the next states it has become capable enough anticipate the danger or reward beforehand and take necessary decisions accordingly.

4) Controller: The controller model takes the respon- sibility of determining what course of action would be most rewarding to the agent. Since the previously discussed models perform most of the heavy lifting in this architecture and are quite complex, it was better to keep this model as simple as possible. Which is why a single layer linear model was used which maps the outputs of previous two models z and h to an action a.

In order to train the controller model and optimize its parameters to give the best performance in its environment, a type of evolutionary strategy algorithm was used known as Co-variance Matrix Adaptation. The uniqueness of this algorithm lies in its dynamic standard deviation which makes the search space to adaptively increase or decrease depending upon the situation. A fitness shaping function is also applied to the fitness score received by each set of parameters so that the algorithm doesn’t converge to local minima.

This project is also inspired by the work of Dylan Djian[3]. His blog helped provide the ground for experimentation in this project.

II. PREVIOUS WORK

The implementation of this architecture has already been done in two environments by the authors who introduced this. Both of them are discussed below:

A. CarRacing Environment

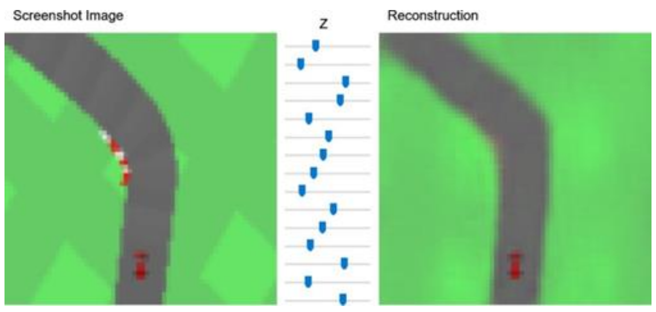

Fig. 4. The agent learns to create a representation of its original environment from which it trains to improve its score

In this environment the agent has access to the pixels it is receiving and the 4 actions it can take namely, Accelerate, Brake, Turn Left, and Turn Right. The agent is rewarded for visiting as many black tiles as possible within a given amount of time. This environment is considered solved if a score of 900, averaged over 100 random trials, is achieved. Despite the best efforts of earlier reinforcement learning agents, the agent based on the world models architecture proved to be the first one to solve this environment.

B. DoomTakeCover Environment

This environment is 3 dimensional in nature and has rather simple controls of moving left, moving right, or standing still. The objective in this game is to dodge the fireballs hurling towards the agent from the other side by enemies and remain alive for as much time as possible. The number of frames through which the agent manages to stay alive is calculated as the score. Due to this mechanics of the game, a more necessary parameter is also given to the agent as input, which is a boolean value indicating whether the agent is dead in the current frame or not.

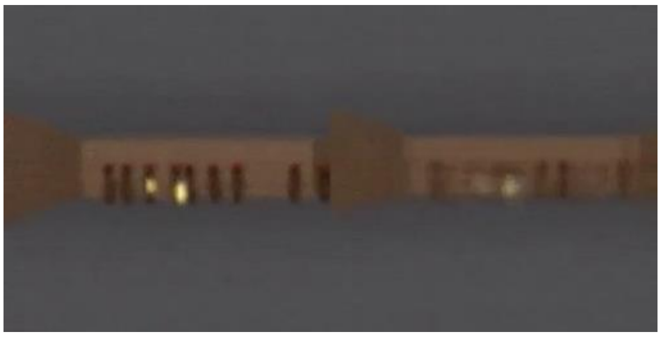

Fig. 5. In this image the first part is of the original environment and the second one is of the ”dream” environment which was created by the agent itself

The difference between this environment’s implementation and the CarRacing one’s is that a variant of the agent in this environment is completely trained inside of its own ”dream” environment which was sampled from the V model’s com- pressed representation. This makes the training much harder as the uncertainty of any fireball’s location increases. But this also means that now when the agent will be tested in the original environment then a considerable increment in its score will be observed due to its extremely hard training regime.
Now, for implementing this whole architecture in a much more complex environment the recently launched reinforce- ment learning library - Retro by Open AI, was used and the new gaming environment was Sonic The Hedgehog.

III. IMPLEMENTATION IN SONIC’S ENVIRONMENT

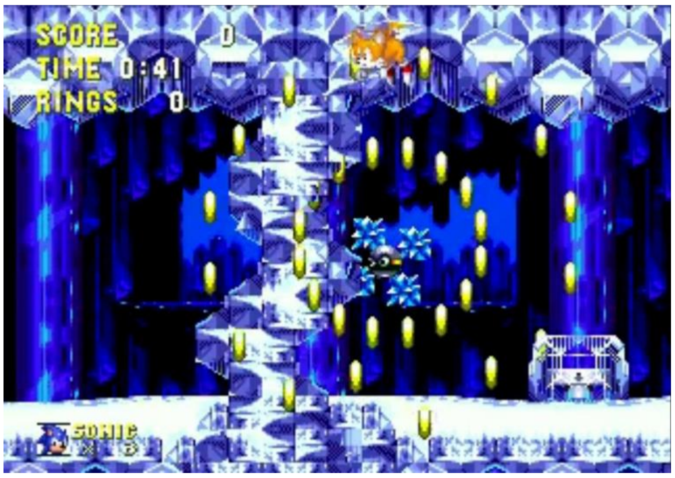

Fig. 6. Just observe the sheer number of elements present in a single frame. Only a small amount of those actually contribute to the cumulative reward.

As can be seen in the image above, this environment is a lot more complex than the earlier ones just by virtue of quantity of content available to the agent. Moreover, this environment has quite a few controls which in turn further increase the complexity of solving it.
Here, as proposed in the blog post[3], some modifications to the world models were made for a coherent implementa- tion. These changes include:

- Temperature parameter which was the randomization variable was removed as that will not benefit the agent’s understanding of the environment in this case.
- β-Variational Autoencoder was used since it will pro- vide a robust representation of Sonic’s Levels

A particular pipeline was required to be followed in order to successfully train the agent to play in any new environment by learning from only a few dozen rollouts. The initial steps before letting the agent play in the new environment include generating necessary frames of the the similar functioning environment and training all of the 3 models in that, then testing the agent in the new unfamiliar environment.

A. Generating Frames

The initial decision for generating the necessary frames so that the agent can form its own representations of it was to by randomly letting it roam around the environment. This led to the agent getting stuck very frequently which is not good if we are trying to cover as much variety in frames as possible. Which is why actual recordings of humans playing the game were used as released by OpenAI here[2].
Thus, in order to let the agent explore parts of the environment which were inaccessible when using the random policy called JERK[2], the provided human rollouts were stored in a MongoDB database of 2500 tuples each with their corresponding values of (frames, actions, reward).

B. Training

Each model was separately trained and a learning rate of
was used with adam optimizer for both the variational autoencoder and the LSTM.
The VAE uses the following configuration: batch size of 300 frames with each frame having dimensions as 128x128x3, β is used as 4 for better enforcement of latent representation z in spite of the quality loss of reconstructed image.
The LSTM network used had 1 layer of LSTM cells, 1024 hidden units, 8 Gaussians, and 1024 units in the first linear layer. A sequence of 500 latent vectors was also used for capturing time dependency.
For greater efficiency the mean and standard deviation of each frame is stored instead of using the raw frames when constructing batches. A rotating buffer was also used to replace small portions, around 5-10%, of the dataset.
In the case of the controller model, the hyperparameters used in co-variance matrix adaptation are: 80 population size with 20 solutions being evaluated in parallel with evaluation of score being averaged over 5 rollouts in a limited amount of time.

IV. OBSERVATIONS: AGENT ROLLOUT IN NEW ENVIRONMENT

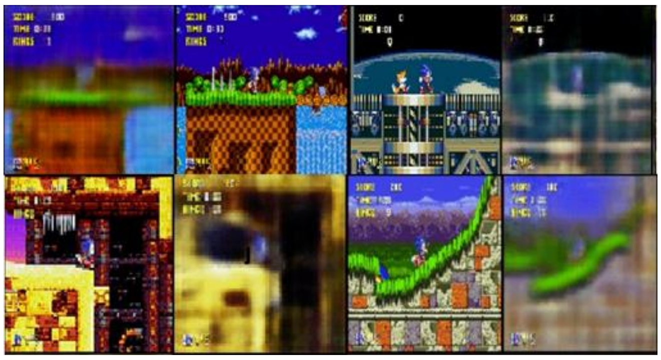

Fig. 7. Comparison of original and reconstructed environments

The vision model was able to successfully compress and recreate the frames with necessary details and it seems to understand that Sonic is a recurring character in every frame of the environments.

The MDN model also seems to have understood some of the basic mechanics of the game, for example it has started predicting in the frames that whenever Sonic is attacked it falls slowly back to the ground, which is a promising achievement. One more thing that happens is that the M model gradually becomes less sure of the long term future since when a prediction has to be made about a very late stage game from current time then the predicted frames start to get a lot more blurrier due

to this uncertainty.
The results obtained from the controller were satisfactory at best and may have been improved if more time was given for training and optimising the hyperparameters. The controller also seems to have understood some basic ideas like moving to the right and jumping is generally a profitable step to take when stuck. It also seems to have understood the concept of enemy units and is able to avoid them if they are present in the current scene. This behaviour of the agent seems to be heavily influenced by the early-stopping mechanism present in Retro library for Sonic which pushes the agent to complete a level as fast as possible and save computation. The best agent was able to achieve an average score of around 3500 out of 10000 over 60 levels in 3 different Sonic games. This may seem a small number but it has more to do with the time available for training, testing, and optimizing than with the perception mechanism.

V. APPLICATIONS

Wherever an abstract understanding of the environment is required while excluding the non-contributing components in decision making, this approach will prove to be quite successful. In designing intelligent systems there is a need of efficient and robust perception mechanisms which can be used utilised to enable devices to perform more sophisticated tasks. These models will prove to be beneficial in scenarios mentioned below,

- During the training of agents for self driving cars reinforcement learning is used. By utilising this method of perception, a significant amount of on-road driving time can be reduced as now the agent will be able to create its own environment in which it can learn to drive a car.
- Using interpolations between multiple environments in order to pretrain reinforcement learning agents reduce the training time required in new environments consid- erably
- The essence of an environment can be extracted and understood when a distributed network of devices util- ising the same perception model are used inside similar environments.

VI. DISCUSSIONS

Even though this architecture is state of the art for some environments and provides a unique new perspective of look- ing at the way reinforcement learning agents learn from their surroundings, it is not without its own share of drawbacks. Firstly, internalizing this architecture is difficult at best as it requires to change the way we are accustomed to perceiving objects in our environment. To further complicate matters, when trying to implement this we came across a variety of dependency issues ranging from incorrect CUDA version combination with Tensorflow or PyTorch to unsupported operating systems.
Moreover, a flaw in this architecture which even the authors discussed, is that it still requires an augmentation or extension in its memory model as long term dependencies, which may be useful to the agent in the future, still cannot be mapped and saved. In order to address this issue a whole new research project is required which completely dissects and reconstruct the memory model used here. Nonetheless, this architecture remains a key stepping stone in achieving artificial intelligence due to its highly generalized nature of perception and the flexibility it provides in implementation.

VII. CONCLUSIONS

Through the means of this project we were able to grasp the advantages and limitations of a highly promising method of perception for reinforcement learning agents. This project also allowed us to experiment with the baseline provided in this post[3]. We also learnt how even by changing the way an agent looks at its environment much improved results can be obtained.
This project also spans over the ideas of meta-learning and transfer learning. Both of which are promising techniques in cases where artificial agents are to be deployed in unfamiliar environments with restrictions on time and availability of information. This method of doing more with less is akin to how humans function, for example, if someone knows how to use a pen then even if a pencil is given to him he will be able to use it without any trouble, which is basically what happens in transfer learning. Perception models like the one discussed in this paper will pave way for far more

intelligent systems which hopefully someday will reach the level of sophistication we humans are accustomed to and allow us to take a step further in evolution.

REFERENCES

1. David Ha, Jurgen Schmidhuber, World Models, arXiv:1803.10122
2. Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, John Schulman, Gotta Learn Fast: A New Benchmark for Generalization in RL, arXiv:1804.03720
3. Dylan Djian, World Models applied to Sonic, https:// dylandjian.github.io/
4. Jaan Altosaar, Variational Autoen-coder Tutorial, https://jaan.io/ what-is-variational-autoencoder-vae-tutorial/
5. Christopher Olah, LSTM Networks Article, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
6. Mike Dusenberry, Mixture Density Networks Tutorial, https:// mikedusenberry.com/mixture-density-networks
7. David Ha, Mixture Density Networks in PyTorch, https: //github.com/hardmaru/pytorch_notebooks/blob/ master/mixture_density_networks.ipynb
8. David Ha, A Visual Guide for Evolutionary Strategies, http://blog.otoro.net/2017/10/29/ visual-evolution-strategies/
9. Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, Ilya Sutskever, Evolution Strategies as a Scalable Alternative to Reinforcement Learn- ing, arXiv:1703.03864
10. Nikolaus Hansen (Inria), The CMA Evolution Strategy: A Tutorial, arXiv:1604.00772Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, Alexander Lerchner, Understanding disentangling in -VAE, arXiv:1804.03599
11. Nikolaus Hansen, Pycma, https://github.com/CMA-ES/ pycma
12. OpenAI Retro, https://github.com/openai/retro
13. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Py-Torch, https://github.com/pytorch/
14. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schnei- der, John Schulman, Jie Tang, Wojciech Zaremba, OpenAI Gym, arXiv:1606.01540
15. Bishop, Christopher M. Mixture density networks.Technical Report, 1994. http://publications.aston.ac.uk/373/
16. Alexey Dosovitskiy, Vladlen Koltun, Learning to act by predicting the future, arXiv:1611.01779
17. Oleg Klimov, Carracing-v0, https://gym.openai.com/envs/ CarRacing-v0/
18. Philip Paquette, DoomTakeCover-v0, https://gym.openai.com/envs/DoomTakeCover-v0/
19. Nadav Bhonker, Shai Rozenberg, Itay Hubara, Playing SNES in the Retro Learning Environment, arXiv:1611.02205