Review Of World Models

Gunjan Arora

2019-07-08

1 Introduction

In the real world, we make quick decisions and do actions by making future predictions based on what we have experienced before. For instance, if we take baseball example the player on strike is able to predict the position of the ball based on his experience. This is what this reinforcement learning is based on i.e. future predictions.
Most of the model-based Reinforcement Learning approaches learn a model of the RL environment but still train on the actual environment. On the other hand, World model fully replaces the actual Reinforcement Learning environment with a generated one and train the agent's controller only inside of the environment which is generated by its own internal model and transfer this policy into the actual environment to perform the action.
In the approach of World Models, agent generates environment on its own which may lead to imperfections. To control imperfections, we adjust a temperature parameter of the internal world model to control the amount of uncertainty. Also, the author trained the agent's controller inside of a noisier and more uncertain version of its generated environment. This approach helped to prevent the agent from taking advantage of the imperfections of its internal world.

2 Agent Model

Their are three components of the agent:

1. Visual Sensory Component(V)
2. Memory Component(M)
3. Decision making Component(C)

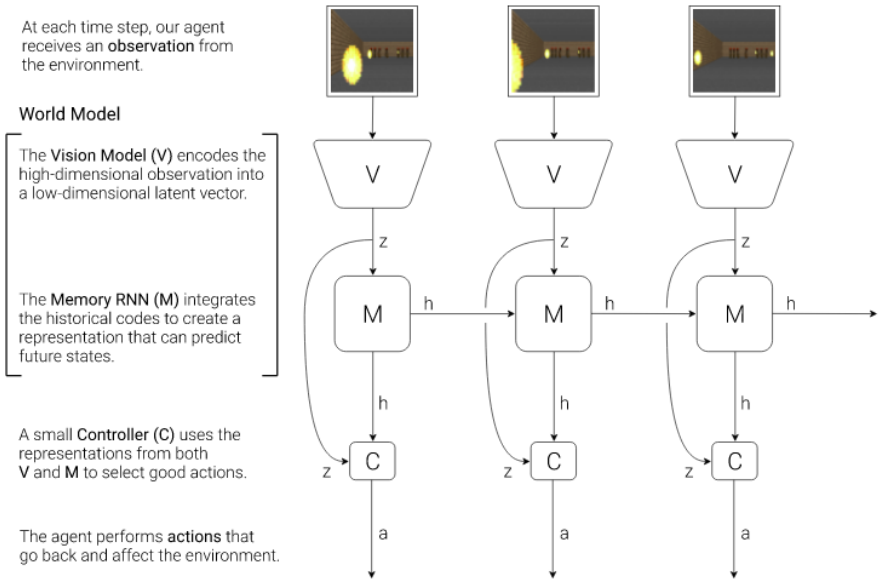


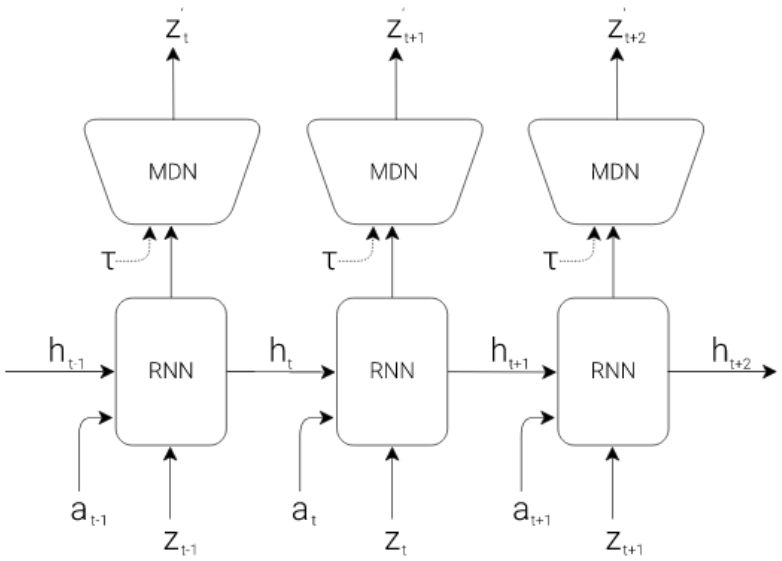
Figure 1: Agent shown pictorially

3 VAE(V) Model

Role of V model is to learn an abstract, compressed representation of each observed input frame. Variational Autoencoder (VAE) was used as the V model in experiments. A frame that is received at time t is converted to low dimensional latent vector which can be used to reconstruct the original image.

4 MDN-RNN (M) Model

Role of M model is to predict future. M model takes the input from the V model generated at time t and past predictions and actions of the agent and use all these inputs to predict future z vectors which V is expected to produce after time t. All the input(past actions, history and present latent vector z) are fed into RNN model, the output of which is passed on to MDN whose other input is temperature(to control uncertainity). Temperature parameter can be adjusted.



RNN with a Mixture Density Network output layer. The MDN outputs the parameters of a mixture of Gaussian distribution used to sample a prediction of the next latent vector z.

Figure 2: M Model

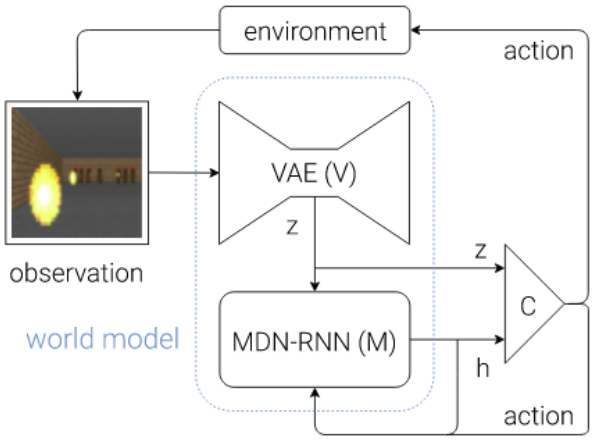
5 Controller(C) Model

The C model is responsible for determining the course of actions to take in order to maximize the expected cumulative reward of the agent during a rollout of the environment.
C is kept simple and small as possible and trained seperately from V and M model, so that most of the agent's complexity resides in V and M model.
C is a simple layer that maps z_t and h_t directly to action a_t at each time step:

$$a_t = W_c \left[z_t \; h_t \right] + b_c$$

6 Combining Models

The following flow diagram illustrates how V, M, and C interacts with the environment:



Flow diagram of our Agent model. The raw observation is first processed by V at each time step t to produce z_t. The input into C is this latent vector z_t concatenated with M's hidden state h_t at each time step. C will then output an action vector a_t for motor control. M will then take the current z_t and action a_t as an input to update its own hidden state to produce h_{t+1} to be used at time $t+1$.

Car racing experiment performed by the author is used in this review to explain how C, V and M models are put together

To train the V model, a dataset of 10,000 random rollouts of the environment are first collected. First agent acts randomly to explore the environment multiple times and a_t actions are recorded and also the observations from the environment are recorded.

Then VAE is trained to encode each frame into low dimensional vector z by minimizing the difference between the given frame and the reconstructed version of the frame produced by the decoder from z.

After this model M is trained using the pre-processed frame z_t. Using this pre-processed data along with the recorded random actions a_t taken, MDN-RNN architecture can be trained to model $P(z_{t+1}|\ a_t,\ z_t,\ h_t)$ as a mixture of Gaussians.

After this the controller C is evolved to maximize the expected cumulative reward of a rollout.