World Models Writeup

For SforAiDl Summer Assignment

Arijit Gupta

16 July 2019

Aim of the article

The goal of this article[1] *is to distill several key concepts from a series of papers 1990– 2015 on combinations of RNN-based world models and controllers. In this article, a simplified framework that can be used to experimentally demonstrate some of the key concepts from these papers, and also suggest further insights to effectively apply these ideas to various RL environments is presented*

1 Components of the Agent Model

1.1 Vision Model (V)

The role of the V model is to learn an abstract, compressed representation of each observed input frame that is part of a sequence. A simple Variational Autoencoder(VAE) is used to compress each image frame into a small latent vector z. The input 2D image frame that is part of a video sequence, goes into the VAE, is encoded into a compressed form z and then decoded and reconstructed for the model to train. In essence the VAE is used by the model to construct its own input, and the error between the reconstructed and actual input, usually cross entropy loss, is known as the reconstruction loss.

1.2 Memory Model (M)

The M model serves as a predictive model of the future z vectors that V is expected to produce using a Mixture Density Network combined with a RNN(MDN-RNN). Since many complex environments are stochastic, the RNN is trained to output a probability density $p(z)$ instead of a deterministic prediction z. The RNN models $P(z_{t+1}|a_t, z_t, h_t)$, where a_t is action taken at time t and h_t is the *hidden state* of the RNN at that time.

In essence, the RNN takes a_t and h_t as parameters, then passes the result to a MDN along with a *temperature* parameter τ which is used to control model uncertainity to return the probability density z_{t+1}

1.3 Controller Model (C)

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 roll-out of the environment. C is a simple single layer linear model that maps z_t and h_t directly to the action a_t at each time step.

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

Here W_c and b_c are the weight matrix and the bias vector that map the concatenated vector $[z_t h_t]$ to the output action vector a_t.

[1] Jürgen Schmidhuber David Ha. *World Models*. `https://arxiv.org/abs/1803.10122v4`. Accessed on 16-07-2019. May 2018.

These V, C and M models interact with the environment together to form the agent model[2]. 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 at for motor control, and will affect the environment. M will then take the current z_t and action at as an input to update its own hidden state to produce h_{t+1} to be used at time $t + 1$.

2 Testing the model on different environments

2.1 Car Racing Experiment

The agent is tested on a car racing experiment for feature extraction. The images are gathered from the video feed and then sent to the V component, and while the reconstructed input is blurry due to lossy compression, it captures the essence of each video frame. When the C model is handicapped with access to only V and not M, it results in wobbly and shaky driving. On being given access to M the driving is more stable.

To summarize the Car Racing experiment, below are the steps taken:

1. Collect 10,000 rollouts from a random policy.

2. Train VAE (V) to encode frames into z $\in R^{32}$.

3. Train MDN-RNN (M) to model $P(z_{t+1}|a_t, z_t, h_t)$.

4. Define Controller (C) as $a_t = W_c[z_t ht] + b_c$.

5. Use CMA-ES to solve for a W_c and b_c that maximizes the expected cumulative reward.

The features extracted by the V are then used by the M to generate a *dream environment* in which the model now continues to train without further use of the V component. Here, the τ component is used to control the uncertainity in the model generated in the dream. We can ask it to produce the probability distribution of z_{t+1} given the current states, *sample* a z_{t+1} and use this *sample* as the real observation. We can put our trained C back into this hallucinated environment generated by M

[2]Normally state of the art models have parameters in the order of 10^8 to 10^9 while the training models looked at here, have 10^7 parameters. This is because it is based on model-free RL models which have 10^3 to 10^6 parameters.

2.2 VizDoom Experiment

It is also tested on a Doom experiment for learning inside a dream. We saw in the car experiment that the policy learned in the real environment somewhat worked int he dream environment, but now we check whether the policy learned in the dream can be transferred to the real world environment The procedure is largely the same as the car racing experiment. The key change is that in the car experiment the M model is only predicting the z_t while here the predictions are used to simulate a full RL environment.
When the model trains in the simulated environment the V model is not needed to encode pixels, so the model trains completely in a latent space environment. This is done using a iterative training process as follows:

1. Initialize M, C with random model parameters.

2. Roll-out to actual environment N times. Agent may learn during roll-outs. Save all actions a_t and observations x_t during roll-outs to storage device.

3. Train M to model $P(x_{t+1}, r_{t+1}, a_{t+1}, d_{t+1} \mid x_t, a_t, h_t)$

4. and train C to optimize expected rewards inside of M.

5. Go back to (2) if task has not been completed.

3 Results of the Tests

The main focus was to implement iterative training using the C-M model to learn in dreams and then transferring it to the real world model and it had a good success rate in the above experiments, making it a really exciting topic

References

David Ha, Jürgen Schmidhuber. *World Models*. `https://arxiv.org/abs/1803.10122v4`. Accessed on 16-07-2019. May 2018.