Comparison of Model-based RL Approaches

Abhishek Maiti (2016005), Suryatej Reddy (2016102)
Vishaal Udandarao (2016119)

Instructions

Add detail to every slide that follows. Only add to the provided text box using the default font and size (Arial 18) and do not exceed the default text box of the slide.

Source Code

Push your code to GitHub and keep it public and paste the repository link here. Make sure to keep your code ready. You can be asked to demonstrate the learning.

https://github.com/vishaal27/Model_Based_Reinforcement_Learning

Goal of the Project

1. We can represent a state in many ways. Are some ways more efficient than others?
2. Can we represent the states in a more compact fashion retaining the same features to deduce a similar optimal policy as the original world model.
3. There are many ways to search for the optimal parameters. Are some ways better than others?
4. Is imposing a prior belief on the state latent justified?

Referenced Papers

Ha, David, & Schmidhuber, Jürgen. (2018). World Models (Version 1.1). http://doi.org/10.5281/zenodo.1207631

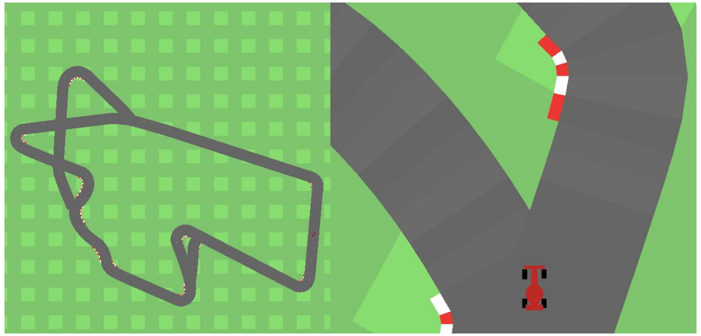
Novel Contribution by the Team

1. We removed the prior latent space constraint to see if the state representations improved.
2. We compared different ways to search the parameter space. Which one converged faster and which one converged optimally?

Environment, Agents, Rewards, and etc.

1. We used the `car-racing-v0` environment by OpenAI gym to assess our modifications to maintain a fair comparison with the reference paper.
2. Agent was a car trying to follow a predefined track.
3. Every timestep is penalized by 0.1. On the event of the car finishing the race, it is awarded 1000.
4. State is the top-level view of the track with the car in it. A sample image:

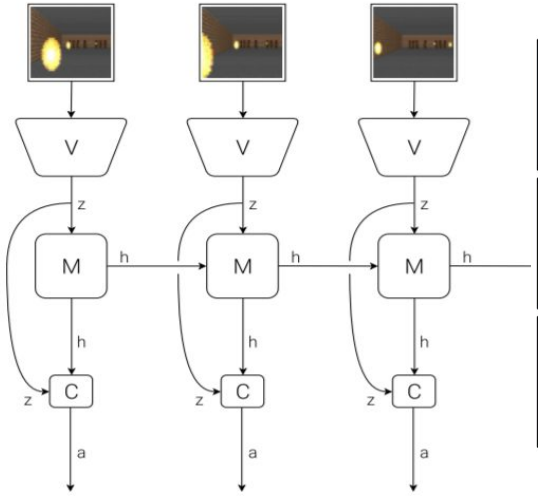
Left image is the entire track

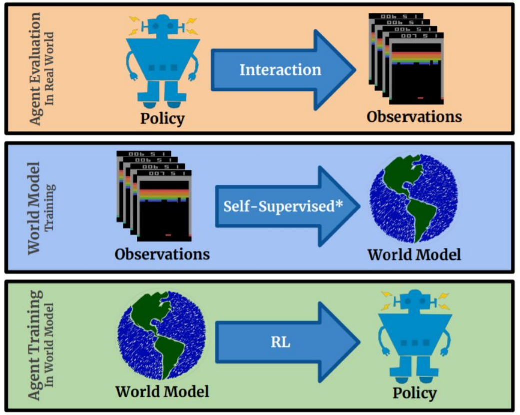


Right image is of our agent

RL Algorithm

1. Our pipeline consists of 3 parts.
   a. Vision (V)
   b. Memory (M)
   c. Controller (C)





2. **V** is responsible for encoding the state into a low dimensional latent vector.
3. **M** has the knowledge of the previous state and create a representation that predicts the future state.
4. **C** takes in the output from both **V** and **M** and performs the action and observes the next state.

Instructions for Results (4 - 5 slides max, use plots as far as possible)

<Show comparison if any with referenced papers>

<Demonstrate impact of any novelty you introduced>

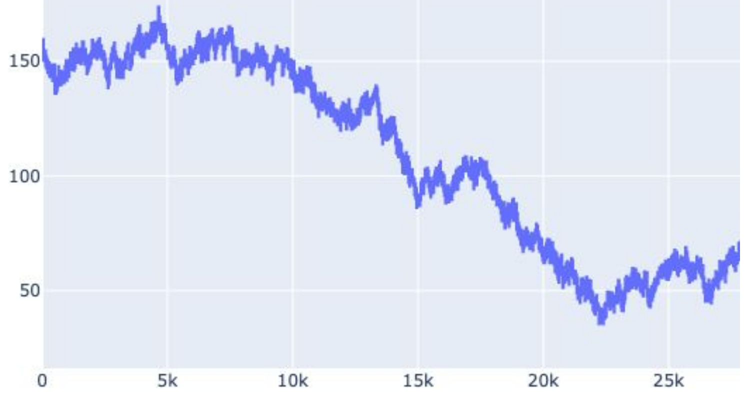
<Average cumulative reward or episodic reward averaged over many episodes>

<Standard deviation or variance of the average cumulative reward>

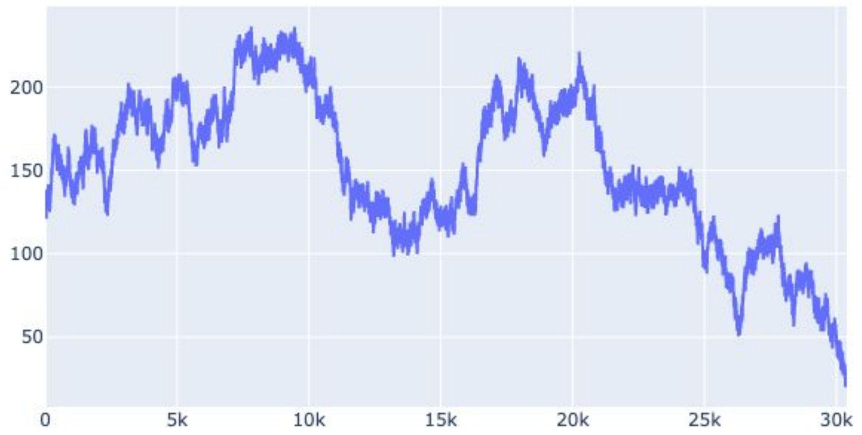
<Training Curves - Showing the progress of the training over time.>

<Any other plots or performance indicators>

Results - 1: Training Loss Plots

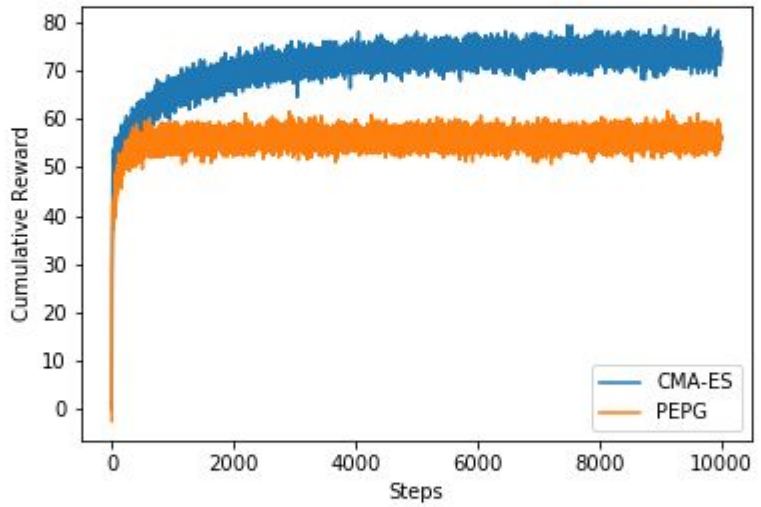


Variational AutoEncoder

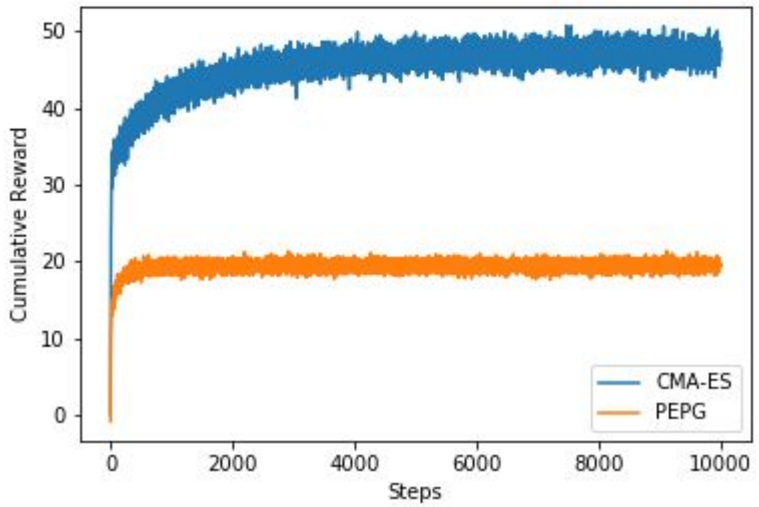


AutoEncoder

Results - 2: Cumulative Rewards Plots



Variational AutoEncoder

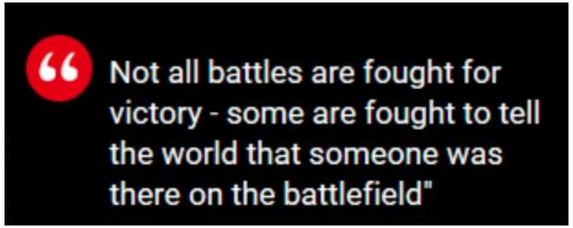


AutoEncoder

Results - 3: Cumulative Rewards obtained

Table: Cumulative reward for different model settings

Encoder Model/Parameter Search Method	Covariance Matrix Adaptation Evolution Strategy (CMA-ES)	Parameter-Exploring Policy Gradients (PEPG)
Variational Autoencoder	74.67 +/- 10.12	60.94 +/- 6.17
Vanilla Autoencoder	47.34 +/- 6.37	20.36 +/- 3.80


Not all battles are fought for victory - some are fought to tell the world that someone was there on the battlefield"

Results - 4: Analysis and Reasoning

1. We found out that Autoencoder performed sub-optimally compared to Variational Autoencoders to encode the state.
2. We think this is because
   a. Imposing some sense of structure by constraining the latent space with a Gaussian prior is better adapted to capturing the state
   b. The predictions of states generated by the MDN-RNN network can be unreliable or very noisy if the latents are left unconstrained
3. We also found that Covariance Matrix Adaptation Evolution Strategy (CMA-ES) converged slower but more optimally than Parameter-Exploring Policy Gradients (PEPG).
4. We think this is because
   a. The PEPG evolution considers all the solutions at every generation rather than considering the top n% of solutions like CMA-ES
   b. According to us, this might have caused a faster convergence to a sub-optimal solution