Progress Report

Jingyuan Liu
1012871910, jyuan.liu@mail.utoronto.ca
Github Repo
VIsualization Results

Project Description

A world model predicts future states based on historical and current state representations and actions. The World Model introduced in (Ha and Schmidhuber, 2018) leverages this capability to better solve tasks in game-playing environments. It consists of three modules: a vision module, a predictor module, and a controller. The vision module is a Variational Autoencoder (VAE) that encodes observations into a latent space. The predictor module, an RNN with a Gaussian Mixture Model as its output head, predicts future states in the latent space, taking historical state information and the current action as input. Finally, the controller (an MLP) outputs an action based on past and predicted state information.

In this project, I aim to modify the architecture with more modern designs. Specifically, I replace the VAE in the vision module with a Vector Quantized-Variational Autoencoder (VQ-VAE) (van den Oord et al., 2017), which constructs the latent space using discrete vectors. Another modification is to replace the RNN in the predictor module with a Transformer encoder, which is considered more effective at handling sequential input.

Overall, the goal of this project is to implement a model that generates actions based on observations. A simple but strong approach would be to create a neural network for end-to-end, observation-to-action mapping(Chi et al., 2023; Black et al., 2024; Kim et al., 2024). However, the most compelling aspect of this project is that the model can not only generate actions but also predict future states. In other words, the model begins to understand how the world operates (albeit in a limited setting), rather than merely approximating the function from observation to action.

Deep learning is an ideal approach for this project because it allows the model to learn implicit patterns within the data. For the tasks in this project—which require constructing complex, non-linear mappings from observations to a lower-dimensional latent space, and from past states and actions to future states—it is extremely difficult to manually engineer feature representation functions or use traditional machine learning methods.

Data Processing

To date, this project has focused on the car racing environment. The dataset was collected in this environment, where a new random action was generated every 20 steps. Information such as observations, actions, and rewards was recorded at every timestep. This collection pipeline is in alignment with the implementation in (Ha and Schmidhuber, 2018). Table 1 provides basic statistics about the collected dataset, and 16 sample episodes are available in video format at this URL for a qualitative overview.

Table 1: Dataset Statistics

Num. Episodes	192	Num. Frames	192000
Max Episode Length	1000	Avg. Episode Length	1000
Image Resolution	(96, 96)	Action Dimension	3

Once collected, the dataset must be processed to create training samples. The action values required no cleaning and were directly converted to tensors. For the images, the status bars at the bottom were cropped out to prevent distracting information from affecting the model. Then the images were converted to tensors and normalized to a range of to ensure stable training.

For the predictor module specifically, the final frame and action of each episode are excluded during predictor training, as there is no subsequent frame to predict. The dataset was split into training, validation, and test sets with a ratio of 80%, 10%, and 10%, respectively. The test set is intended for evaluating the model's performance offline, though it has not yet been utilized.

To thoroughly test the model's performance, it will be deployed to play the game in the simulation environment. A trained model that performs well on the collected data may still encounter unseen scenarios in the live game environment, which serves as a suitable test of its generalization capabilities.

Baseline Model

The baseline model is the original implementation from (Ha and Schmidhuber, 2018). As there is no official PyTorch version, I adjusted a third-party implementation from this GitHub repository. The training process consists of three stages. For specific parameter settings, please refer to my GitHub repository.

In Stage 1, the vision module (VAE) is trained using only images from the dataset. The VAE is composed of a CNN encoder, a CNN decoder, and a latent space defined by a normal distributio. The CNN encoder has four convolutional layers and one linear layer to flatten the input, while the decoder is symmetric to the encoder, with four convolutional transpose layers.

In Stage 2, an image is first processed by the trained VAE from Stage 1 to obtain its latent representation. Then, the predictor module (MDNRNN) is trained in this latent space. Specifically, the predictor models the conditional probability $p(z_{t+1}|z_t, a_t)$, where z_t is the latent representation of observation image at timestep t and a_t is the action. First, an LSTM takes z_t and a_t as input and outputs both a prediction for timestep $t + 1$ and a hidden state h. This output is then passed to a Gaussian Mixture Model (with 5 components) to generate a probabilistic distribution over z_{t+1}.

In Stage 3, the controller is trained to generate and execute actions in the game environment. The controller is a relatively simple network, consisting of only a single linear layer. It takes the hidden state from the predictor (Stage 2) as input and projects it to an action space. The training method is CMA-ES, an evolutionary algorithm for optimization problems. This evolutionary algorithm allows for online adaptation, which is more flexible than supervised learning, especially since acquiring ground-truth actions is difficult for a randomly collected dataset. Furthermore, because the controller is a simple linear layer, evolutionary algorithms are sufficient to achieve good performance.

The performance of this baseline model in the live game environment is shown in Fig 2. Qualitative results for Stage 1 and Stage 2 can be obtained by passing the predicted latent representation through the trained VAE decoder. An example of the MDNRNN model's prediction is shown in Fig 1. Additional training curves, qualitative images, and videos of the agent playing the game are available at this URL. Checkpoints are also provided.

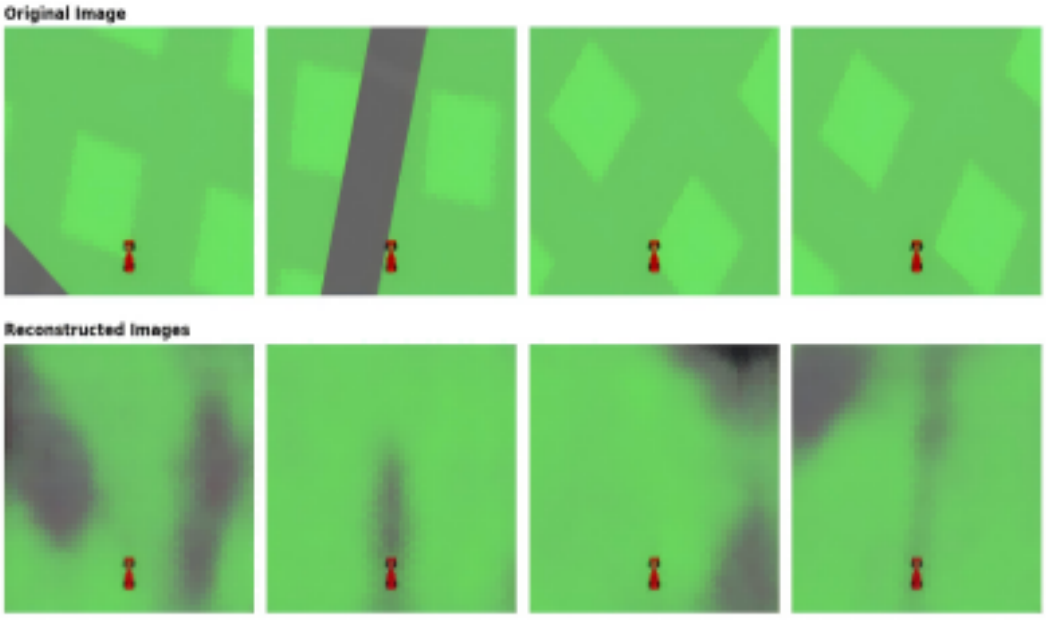

Figure 1: Reconstructed images from the predicted latent representations of the RNN. The top row shows the ground-truth image of the next state, while the bottom row shows the predicted image.

Primary model

The primary model's architecture is similar to the baseline, with the exception of the backbones used for the vision and predictor modules. For the vision module, a codebook for vector quantization replaces the normal distribution in the latent space of the VAE.

For the predictor module, a Transformer encoder replaces the RNN for modeling the sequences of states and actions. The decoder part of the Transformer is omitted for simplicity. The encoder consists of 6 multi-head self-attention layers, with 8 heads in each layer. First, two separate linear projectors map the state sequence $(z_0, z_1, \ldots, z_t)$ and the action sequence $(a_0, a_1, \ldots, a_t)$ into the Transformer's embedding dimension. These two projected sequences are then interleaved to form a single input sequence: $z'_0, a'_0, \ldots, z'_t, a'_t$)[1] In addition to positional embeddings, type embeddings are added to distinguish between state and action tokens, where states are assigned type ID 0 and actions are assigned type ID 1.

The Transformer encoder processes this interleaved sequence using a causal mask. This mask is a lower-triangular matrix that allows each token to attend only to previous tokens and itself. After the Transformer encoder produces the output sequence, a Mixture Density Network takes the outputs corresponding to the action tokens a'_t as input, similar to the baseline.

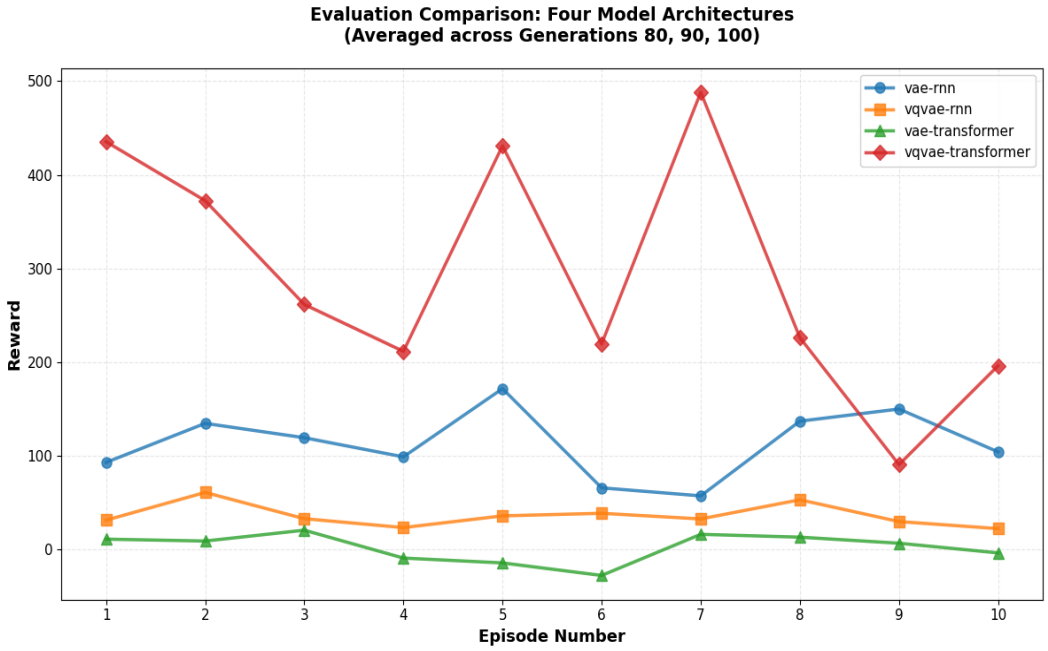

Figure 2: Performance comparison of different model combinations. The Y-axis represents the average reward achieved in the game environment. The X-axis represents 10 episodes for testing. For the vision and predictor modules, the models from the final training epoch were used. For the controller, performance was averaged over generations 80, 90, and 100 of the CMA-ES algorithm.

The results are shown in Fig 2, where the four lines represent the performance of four different combinations of vision and predictor modules. Interestingly, substituting only one of these components into the baseline architecture resulted in degraded performance. This may be due to the discrete nature of the VQ-VAE's latent space, which could be particularly well-suited to the autoregressive, token-based processing of the Transformer. A definitive explanation for this phenomenon has not yet been established. Further investigation, including comprehensive evaluation and hyperparameter tuning for each of the three architectural components, is planned for the upcoming month. Additional information is in this this URL[2].

[1]The prime symbol (') indicates that z and a have been projected and are no longer in their original vector space.

[2]The training loss for Stage 2 may become negative. This is expected, as the loss function is the Gaussian Negative Log-Likelihood (NLL) for the Mixture of Gaussians, which is not bounded to be positive. A negative loss value does not indicate an error or negatively impact performance.

References

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., Jakubczak, S., Jones, T., Ke, L., Levine, S., Li-Bell, A., Mothukuri, M., Nair, S., Pertsch, K., Shi, L. X., Tanner, J., Vuong, Q., Walling, A., Wang, H., and Zhilinsky, U. (2024). π_0: A vision-language-action flow model for general robot control.

Chi, C., Feng, S., Du, Y., Xu, Z., Cousineau, E., Burchfiel, B., and Song, S. (2023). Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc. `https://worldmodels.github.io`.

Kim, M., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E., Lam, G., Sanketi, P., Vuong, Q., Kollar, T., Burchfiel, B., Tedrake, R., Sadigh, D., Levine, S., Liang, P., and Finn, C. (2024). Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*.

van den Oord, A., Vinyals, O., and kavukcuoglu, k. (2017). Neural discrete representation learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.