World Models

Yuchen Liu (yl5680), Su Lei Win (sw5205), Abhishek Kalra (ak7468)

Task – 1

Read the paper and write your own 4-page report of the technique (a model-based RL), in a tutorial like fashion so computer scientists can still understand it.

Model-Based Reinforcement Learning Technique

Model-Based Reinforcement Learning

Model-based Reinforcement Learning (Model-based RL) uses environment, action, and reward to get the most reward from the action. It focuses on the model than the policy, so the quality of the model is crucial for success. It uses machine learning models like neural networks, random forest, and others.

Unlike other reinforcement learning, Model-based RL does not require specific prior knowledge. However, having prior knowledge can speed up the agent's learning. Model-based RL requires fewer data to learn a policy and less interaction with the environment to train models. It does not need to learn directly from interactions with the real environment. The sample is used more efficiently in model-based learning because we do not need to sample again to optimize the policy once the model and the cost function are known.

The agent captures the transition function and the reward function during the interaction with the environment. If the observation environment is high dimensional during the training, the agent will not have a clear reward function, so the agent could not know whether the task is succeeded or not. We have to define the reward function by providing an image of the goal. Based on the transition and reward function, the agent can predict the next state and the reward.

Dealing with high dimensional observations

In the World Model paper, the agent has only access to high dimensional observations during the training. For that situation, we can choose to work from three different approaches: learning a model in latent space or learning a model directly from the high dimensional observations or learning inverse models. In the paper, they learn their model in latent space and apply it to the Model-Based RL, so we will be only explaining the latent approach.

Using Latent Models with the Model-Based RL for the high dimensional observations

In Latent Models, unlike the other Markov Decision Process (MDP) approach, all the MDP elements: state, action, transition, and rewards do not work on the single model. They have their own models. Latent models are also not connected directly to the input and output. Instead, they are connected to other models and signals. It captured the interaction in different models: observation, transition, representation, and rewards, and they are either trained in self-supervised like variational autoencoder or unsupervised or recurrent networks. Basically, we learn the model in latent space and then apply it to the standard model-based reinforcement learning.

Let's look at how the latent model algorithm works. First of all, we have to run the base policy like a random policy to collect the sample data, and then we use that sampled data to learn latent embedding observations and learn dynamics model in that latent space. After that, we use that model to optimize over a sequence of actions. Then, we execute those planned actions and append visited tuples: action, observation, and the next observation.

World Model's Agent

World Model's agent has three main components: Vision (V), Memory (M), and Controller (C). They are trained separately. Vision components help the agent to encode the high-dimensional input image frame into a small latent vector z . Memory components help the agent to make the prediction about the future based on the previous information. A controller is a decision-making component which helps the agent to make the decision and take action based on the information from the vision and memory component.

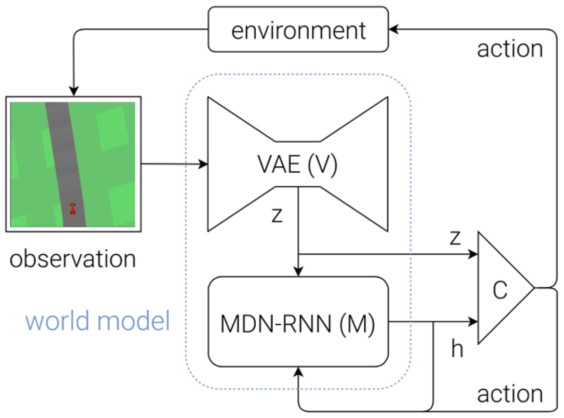


Figure 1 Flow diagram of how V, M, and C interacts with the environment in World Model

Car Racing Experiment Procedure

Step 1: Generate Random Rollouts

They let their agent explore the environment multiple times to collect 10,000 rollouts. In the meantime, they save their agent's random actions at each time step.

Step 2: Train VAE

After they have collected 10,000 rollouts, they train their agent's vision (V) model. The agent has only access to high dimensional observations during the training, so they use Convolutional Variational Autoencoder (ConvVAE) to learn their model in latent space. It helps to encode each frame into a low dimensional latent vector z. It also helps them to compress the space.

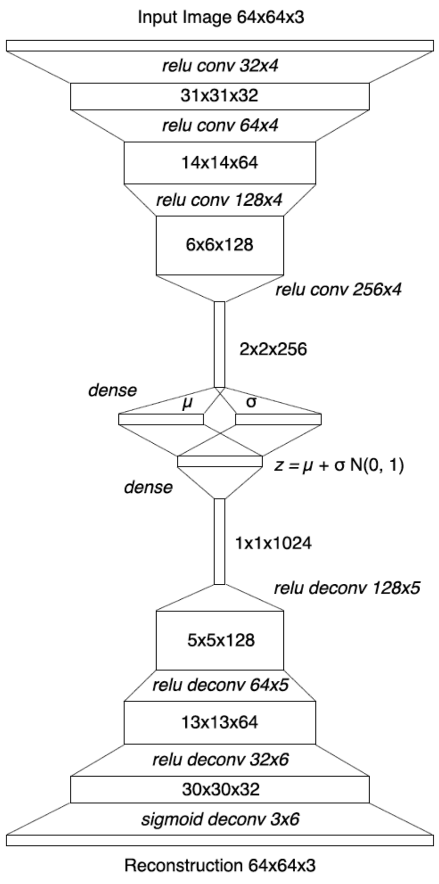


Figure 2 Convolutional Variational Autoencoder (ConvVAE)

First of all, they resize each of the high dimensional input images into 64x64 pixels and stored each of the pixels as three floating points to represent each of the RGB channels. The ConvVAE takes resized image: 64x64x3 (width, height and RGB depth), and encodes it into low dimensional vectors latent vector z of size 32 by passing through four ReLU convolutional layers. Now, the agent does not need to deal with higher representation, so the agent can learn more efficiently. After that, they decode and reconstruct the image by passing through the latent vector into three ReLU deconvolution layers and a sigmoid deconvolution layer. The output has to be between 0 and 1, so they used sigmoid in their output layer.

Step 3: Train MDN-RNN

To train their agent's M Model, they use a Mixture Density Network (MDN) combined with an RNN approach. It predicts the future vector z that will be produced by the V model. They use long short-term memory (LSTM) recurrent neural network for the RNN because it helps to resolve the RNN's vanishing gradient problem. It remembers the past data in memory and trains the model by using backpropagation.

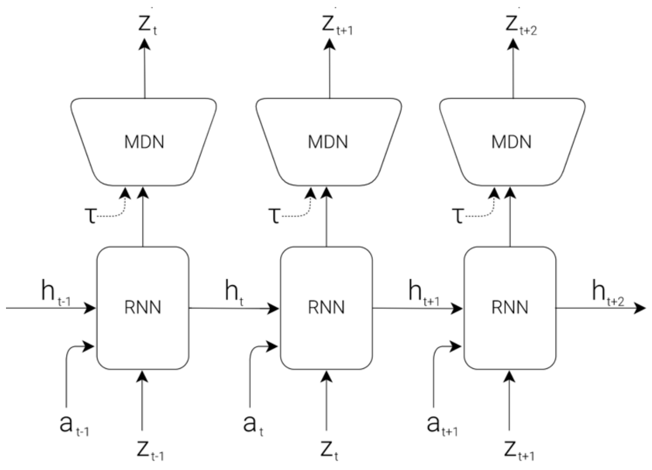


Figure 3 Recurrent Neural Network with Mixture Density Network

To generate RNN, they use the latent vector z from the VAE model, the agent's recorded random actions a, and the hidden state h of the RNN. After that, they fed it to the MDN with the temperature parameter τ. They approximate the probability density function (pdf) as a mixture of Gaussian distribution and train the RNN to get the probability distribution of the next latent vector z_{t+1}.

Step 4: Train the Controller

They trained the controller model separately from the vision and memory model. Their agent's vision and memory do not know the actual signal from reward. Only the controller (C) model knows the reward information from the environment. Their agent controller controls three continuous actions: steering left or right, acceleration, and brake.

For their agent's controller components, they use a simple single-layer linear model:

$$a_t = W_c[z_t \mathrm{h}_t] + b_c$$

In that linear model, the weight matrix W_c and bias vector b_c maps the concatenated latent vector at time t: z_t and the hidden state of the RNN at time t: h_t to the action taken at time t: a_t . They use Covariance Matrix Adaptation – Evolution Strategy (CMA-ES) to find the optimal parameters of the weight matrix W_c and bias vector b_c. CMA-ES creates multiple random initialized copies and then tests each of them inside the environment. It saves the state of the process from each generation and produces the best weight after each generation.

Task – 2

Add to the report of a section that documents your own results and comment on the speed of learning a policy that drive the car around the track.

In this Project, we implemented the World Models (Ha et al., 2018), a recent model-based reinforcement learning paper that achieves surprisingly good performance on the challenging CarRacing-v0 environment.

World Models compromise of the following:

A) Variational Auto-Encoder (VAE, Kingma et al., 2014)(V): a generative model with both an encoder and a decoder. The encoder's task is to compress the input images into a compact latent representation(z). The decoder's task is to recover the original image from the latent representation.
B) A Mixture-Density Recurrent Network (MDN-RNN, Graves, 2013)(M): predicts the latent encoding of the next frame given past encodings and actions. The mixture-density network outputs a Gaussian mixture for predicting the distribution density of the next observation.
C) Linear Controller (C): It takes inputs from both the VAE (latent encoding of the current frame) and from the hidden state of the MDN-RNN (given past latents and actions) and outputs an action. It is trained to maximize the cumulative reward using the Covariance-Matrix Adaptation Evolution-Strategy (CMA-ES, Hansen, 2006).

Below is a figure from the original paper explaining the architecture.

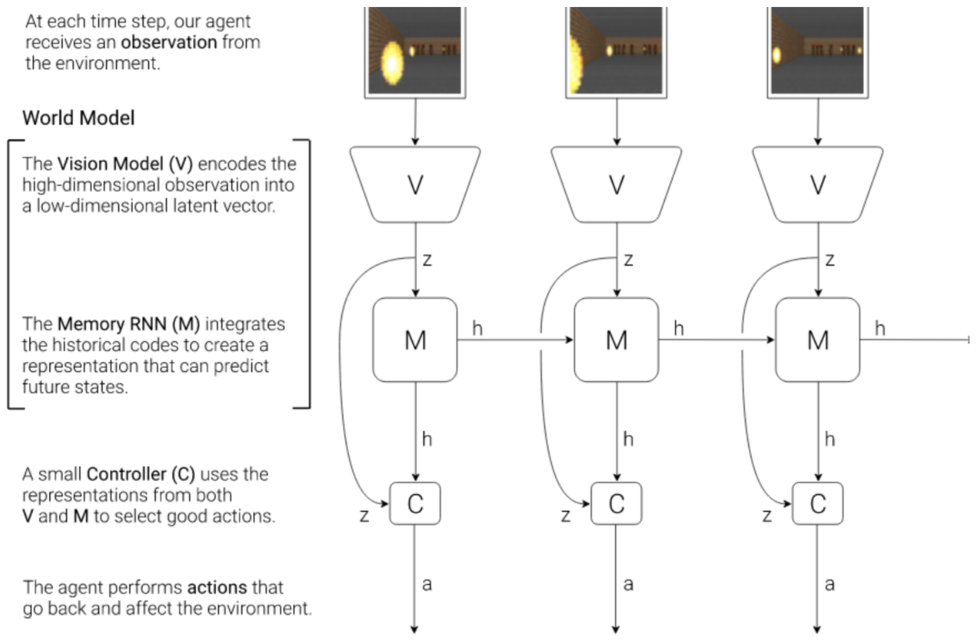
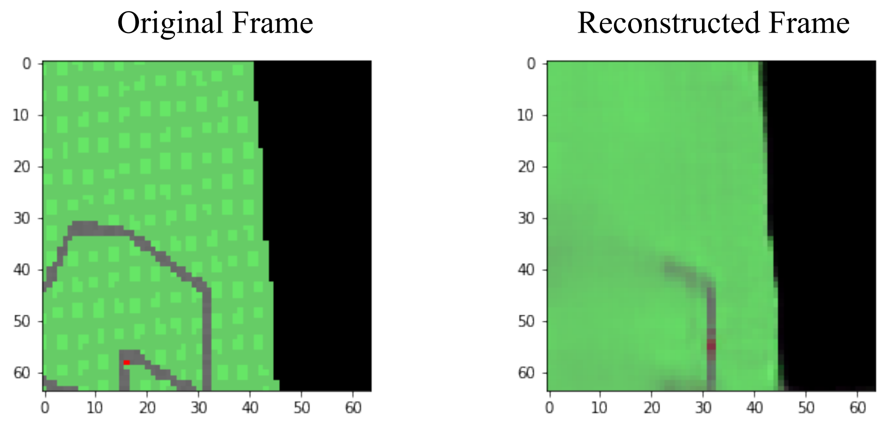

At each time step, our agent receives an observation from the environment.
World Model
The Vision Model (V) encodes the high-dimensional observation into a low-dimensional latent vector.
The Memory RNN (M) integrates the historical codes to create a representation that can predict future states.
A small Controller (C) uses the representations from both V and M to select good actions.
The agent performs actions that go back and affect the environment.
V
M
C
z
h
a

Figure 4 World Model Architecture

During our re-implementation, we trained the Full world model (V and M) in unison. Following steps were performed:

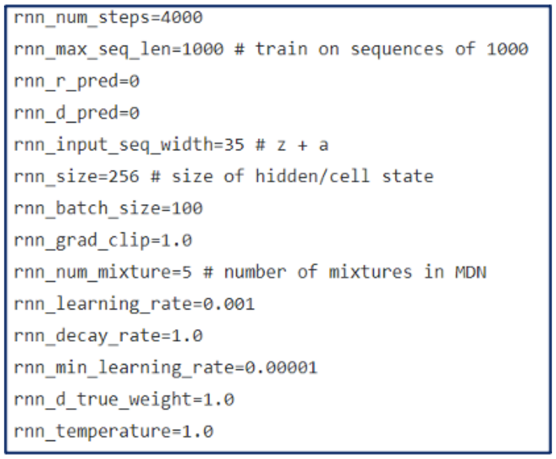
D) Collected 640 rollouts from a well-suited random policy.

E) V was trained using the collected dataset by encoding each frame into low dimensional latent vector representation (z) for 10 epochs. The trained model's decoder outputs the reconstructed frame from z and uses the difference from the original frame for optimizing its training.



As noticable the reconstructed frames do lose some degree of information in reconstructure frame however the latent vector (z) does captures the essence of the image which could be attributable to smaller rollout size of 640 as compared to 10,000 in the original implementation.

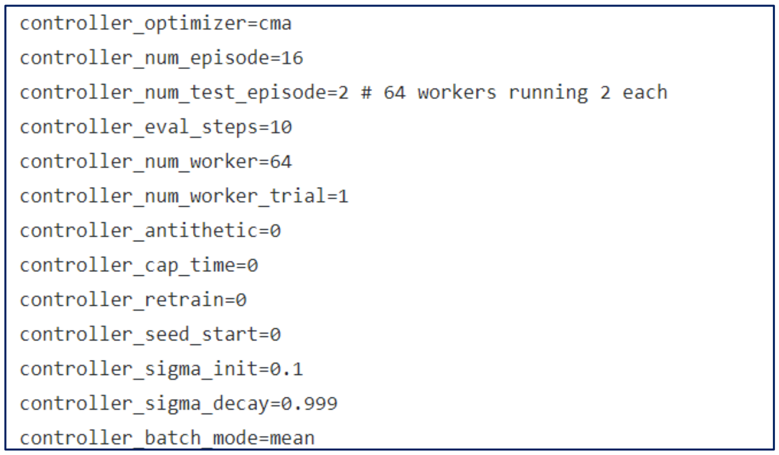
F) The MDN-RNN was now trained on the frames pre-processed by VAE. Using this pre-processed data MDN-RNN was trained to output the probability distribution of the next latent vector z_{t+1} given the current and past information made available to it. To reduce computational load, we trained the MDN-RNN on fixed sized subsequences (1000) of the rollouts. The MDN-RNN's LSTM used 256 hidden units and used 5 gaussian mixtures for sampling z. Other important model parameters settings used have been illustrate below.



```
rnn_num_steps=4000
rnn_max_seq_len=1000 # train on sequences of 1000
rnn_r_pred=0
rnn_d_pred=0
rnn_input_seq_width=35 # z + a
rnn_size=256 # size of hidden/cell state
rnn_batch_size=100
rnn_grad_clip=1.0
rnn_num_mixture=5 # number of mixtures in MDN
rnn_learning_rate=0.001
rnn_decay_rate=1.0
rnn_min_learning_rate=0.00001
rnn_d_true_weight=1.0
rnn_temperature=1.0
```

Figure 5 Model Parameters

G) Training the controller (C) using the VAE & memory: Controller is trained while interacting with the environment using Covariance Matrix Adaptation Evolution Strategy(CMA-ES). At each time step, the controller takes as input both the encoded current frame and the recurrent state of the MDN-RNN, which contains information about all previous frames and actions. Figure below shows the values used for controller's parameters during training.



```
controller_optimizer=cma
controller_num_episode=16
controller_num_test_episode=2 # 64 workers running 2 each
controller_eval_steps=10
controller_num_worker=64
controller_num_worker_trial=1
controller_antithetic=0
controller_cap_time=0
controller_retrain=0
controller_seed_start=0
controller_sigma_init=0.1
controller_sigma_decay=0.999
controller_batch_mode=mean
```

Our implementation reached a best score of 500 over 100 generations, below the 906 reported in the paper(over 1800 generations).

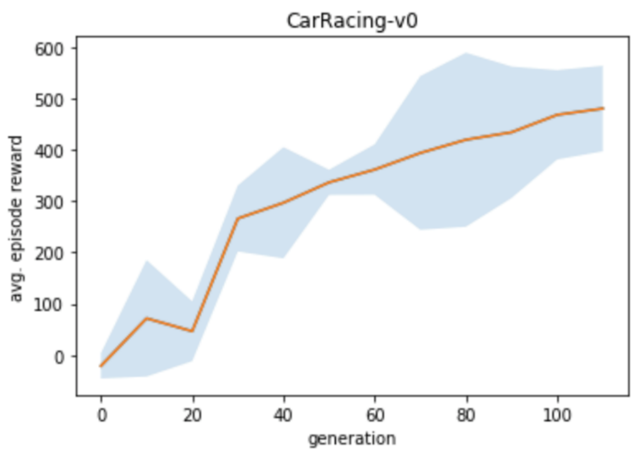


Figure 6

We believe the gap in the results is related to our reduced computational power, resulting in tamed down hyperparameters for CMA-ES compared to those used in the paper.

Conclusion

We reproduced the paper “World Models” on the Car Racing environment. The results were easily reproducible. However, we noted muted performance due to our time and computational resource constraints.

Task – 3

Document a new approach that combines the VAE with a GAN.

The term VAE-GAN was introduced by A. Larsen et. al (2016) in their paper "Autoencoding beyond pixels using a learned similarity metric". The authors noting the criticality of choice of similarity metric in generative models like Variational Auto Encoder (VAE) explored usage of alternative architectures & similarity metrics. Through their experiments authors noted a combination of VAE and generative adversarial networks (GAN) to outperform traditional VAEs. The following sections discuss the architectures of generative models VAEs, GANs and the improvements made by assimilation of their architectures.

Variational Auto Encoders(VAEs)

Variational Autoencoders (VAEs) are powerful generative models with applications as diverse as from generating fake human faces, to producing purely synthetic music. The textbook defines VAEs as "providing probabilistic descriptions of observations in latent spaces." Intuitively, this means VAEs store latent attributes as probability distributions. This ability to encode the latent attributes of the input in a probabilistic manner (distribution) instead of a deterministic manner (single value) like standard autoencoder provides them a superior ability for recreation.

A typical variational autoencoder is nothing other than a cleverly designed deep neural network, which consists of a pair of networks: the encoder and the decoder. The encoder can be described as a variational inference network responsible for mapping of input x to posteriors distributions q $\theta(z|x)$. Likelihood $p(z|x)$ is then parametrized by the decoder, a generative network which takes latent variables z and parameters as inputs and projects them to data distributions $p\phi(x|z)$.

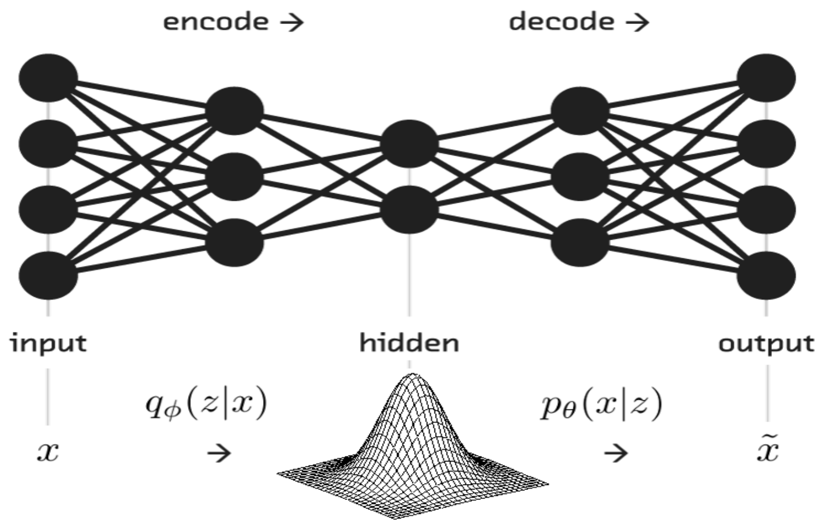


Figure 7 VAE Architecture

The VAE regularizes the encoder by imposing a prior over the latent distribution p(z). Typically, z N(0, I) is chosen. The VAE loss is minus the sum of the expected log likelihood (the reconstruction error) and a prior regularization term:

Equation 1: VAE loss Function

$$\mathcal{L}_{\text{VAE}} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})}\right] = \mathcal{L}_{\text{llike}}^{\text{pixel}} + \mathcal{L}_{\text{prior}}$$

Generative Adversarial Networks (GANs)

GANs just like VAEs belong to a class of generative algorithms that are used in unsupervised machine learning. A GAN architecture (Figure 8) consists of two neural networks, a generative neural network and a discriminative neural network engaged in a ‘zero-sum’ game. A generative neural network is responsible for taking noise as input and generating samples. The discriminative neural network is then asked to evaluate and distinguish the generated samples from training data. Much like VAEs, generative networks map latent variables and parameters to data distributions.

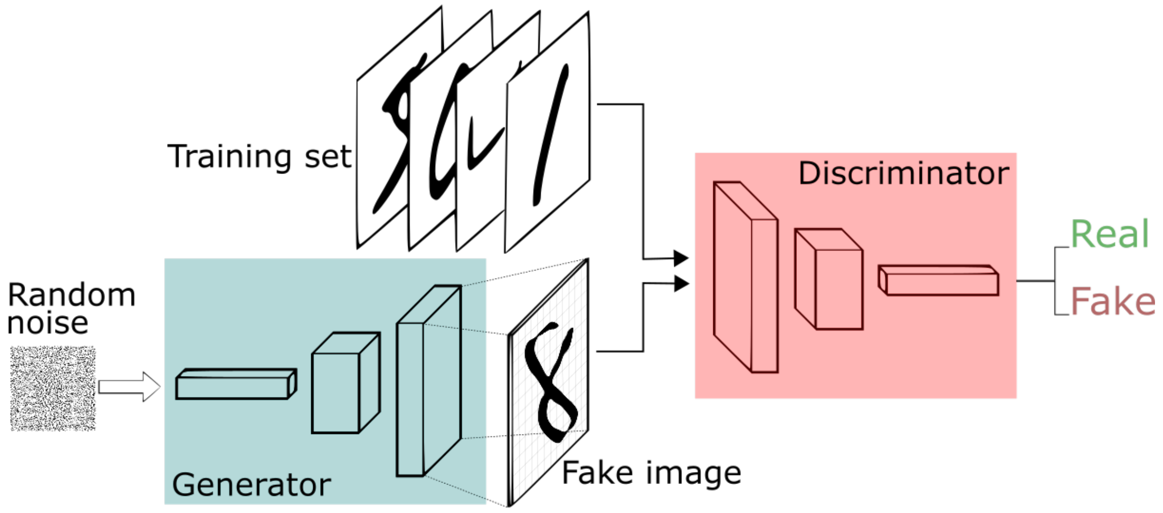


Figure 8 GAN Architecture

The GAN objective is to find the binary classifier that gives the best possible discrimination between true and generated data and simultaneously encouraging Generator to fit the true data distribution. We thus aim to maximize/ minimize the binary cross entropy:

Equation 2: GAN Loss Function

$$-\mathcal{L}_{GAN} = E_{x\sim p_{data}} \log(D(x)) + E_{x\sim p_{model}} \log(1 - D(x))$$

Combining VAE-GANs Architectures

A major drawback of VAEs is the unrealistic and blurry outputs that it generates (Dosovitskiy & Brox). This has to do with the way data distributions are recovered and loss functions are calculated in VAEs. VAEs use element-wise measures like the squared error (equation 1) which although simple but are not suitable for training requiring usage of complex datasets like images. A. Larsen et. al therefore, recommend using a higher-level and sufficiently invariant representation to measure image similarity.

The authors accomplish the same by jointly training a VAE and a generative adversarial network (GAN) (Goodfellow et al., 2014) and use GAN's discriminant networks property to learn a similarity metric.

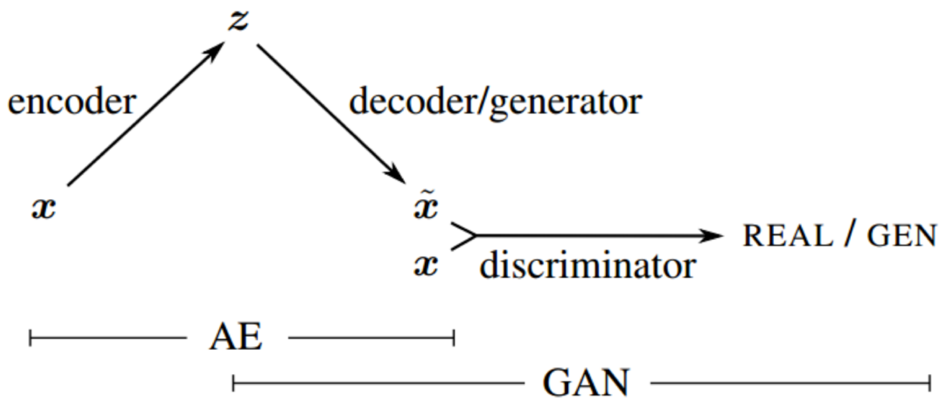


Figure 9 VAE-GAN Combine Architecture

The assimilated VAE-GAN architecture aggregates the VAE decoder and the GAN generator by letting them share parameters and training them jointly. Simultaneously VAE decoder is replaced with GAN discriminator allowing a more abstract reconstruction error for the VAE by transfer of the properties of images learned by the GAN's discriminator. VAE's loss term (L Pixel-llike) in equation 1 is therefore, updated using the following:

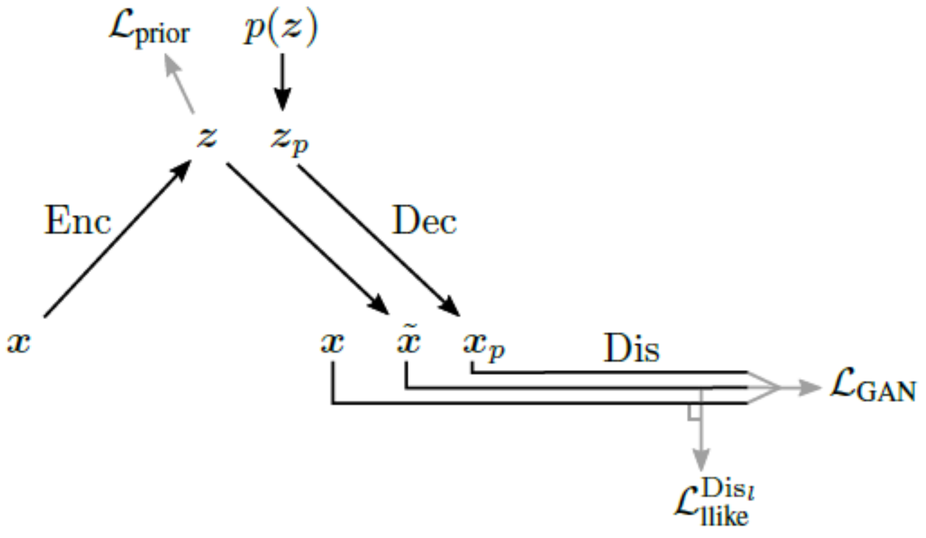
$$\mathcal{L}_{\text{llike}}^{\text{Dis}_l} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\text{Dis}_l(\boldsymbol{x})|\boldsymbol{z})\right]$$

The combined model is therefore, trained using the following loss function.

Equation 3: VAE-GAN Loss

$$\mathcal{L} = \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l} + \mathcal{L}_{\text{GAN}} \ .$$

Figure 10 shows the information flow in VAE-GAN architecture.



$\mathcal{L}_{\text{prior}}$
$p(z)$
z
z_p
Enc
Dec
x
x
$\tilde{x}$
x_p
Dis
$\mathcal{L}_{\text{GAN}}$
$\mathcal{L}_{\text{llike}}^{\text{Dis}_l}$

Figure 10 VAE-GAN Information Flow

The end result is a model that combines the advantage of GAN as a high-quality generative model and VAE as a method that produces an encoder of data into the latent space z. The architecture provides the following benefits:

Limiting error signals to relevant part of networks: As both VAE and GAN train simultaneously not all network parameters as per the combined loss are updated. The parameters are updated as shown below:

$$\begin{aligned}
\boldsymbol{\theta}_{\text{Enc}} &\overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Enc}}}(\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l}) \\
\boldsymbol{\theta}_{\text{Dec}} &\overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dec}}}(\gamma\mathcal{L}_{\text{llike}}^{\text{Dis}_l} - \mathcal{L}_{\text{GAN}}) \\
\boldsymbol{\theta}_{\text{Dis}} &\overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dis}}}\mathcal{L}_{\text{GAN}}
\end{aligned}$$

Discriminator minimizes GAN’s loss function and does not minimize LDisl-llike (Decoder’s loss function) as this would collapse the discriminator to 0. Further by not backpropagating the error signal from LGAN to Encoder better results are achieved.

Repeat the experiment in the car racing environment. There are multiple implementations of the VAE/GAN approach. Add to the report what improvements (if any) you observed from the transition to GAN in terms of performance (20 points)

We implemented the combined VAE/GAN architecture using the implementation done by Larsen et al. (2016). The code for the same was sourced from the following GitHub repository https://github.com/leoHeidel/vae-gan-tf2. As discussed, the combined model aims to improve the VAE's ability for reconstruction of images from the latent representation(z) generated by VAE's encoder. Using the rollouts generated as part of the earlier task, we retrained the aggregated VAE-GAN model in the following steps:

A) Collected 640 rollouts from a well-suited random policy.

B) Combined VAE-GAN model was trained using the collected dataset by encoding each frame into low dimensional latent vector representation (z).

C) The latent vector representation (z) was fed into the assimilated VAE-GAN architecture that aggregates the VAE decoder and the GAN generator by letting them share parameters and training jointly. Simultaneously VAE decoder is replaced with GAN discriminator allowing a more abstract reconstruction error for the VAE by transfer of the properties of images learned by the GAN's discriminator. The model was trained using the following loss function:

$$\mathcal{L} = \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l} + \mathcal{L}_{\text{GAN}} \,.$$

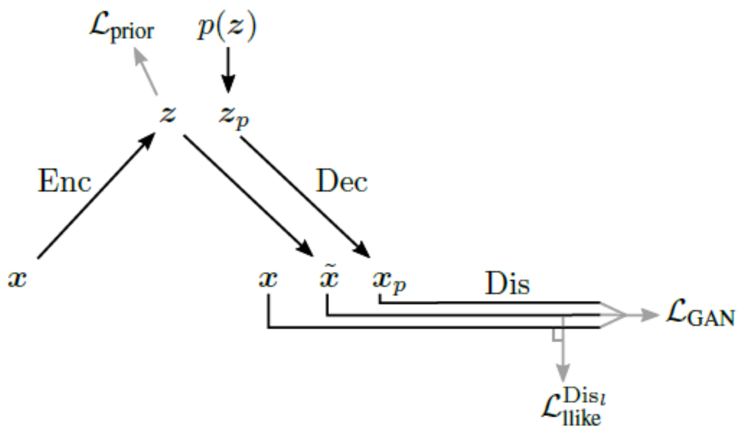
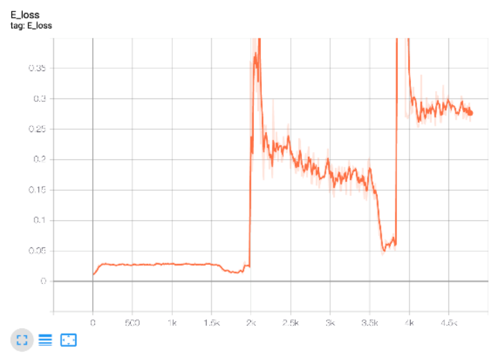

$\mathcal{L}_{\text{prior}}$
$p(z)$
z
z_p
Enc
Dec
x
x
$\tilde{x}$
x_p
Dis
$\mathcal{L}_{\text{GAN}}$
$\mathcal{L}_{\text{llike}}^{\text{Dis}_l}$

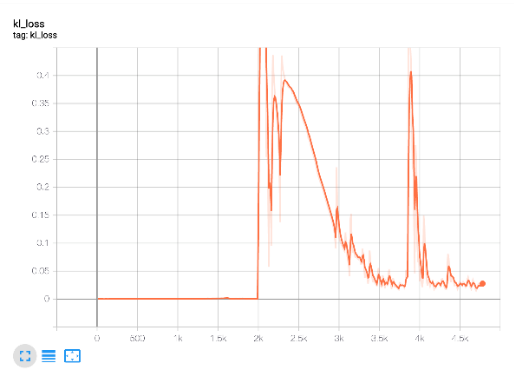
Figure 11 VAE-GAN Information Flow

The graphs below shown the overall VAE-GAN loss function and its components generated as part of training the VAE-GAN model on random rollouts in Car-racing-v0 environment.

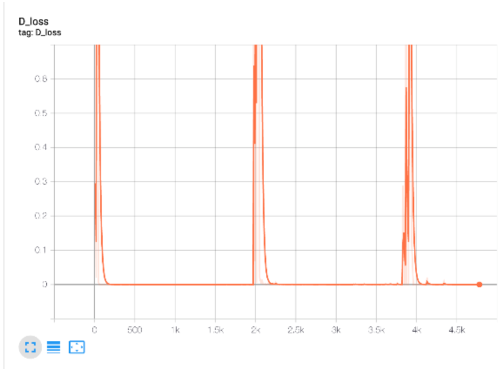
Graph (E_Loss) shows the encoder's loss that after staying low for initial iterations and suddenly spikes up after 2k iterations and then gradually reduces. It again shoots up after 4K iterations. Encoder's loss comprises of both prior regularization (KL Divergence) (Graph kl_loss) as well as discriminator's loss from hidden layer of GAN.

E_Loss KL_Loss

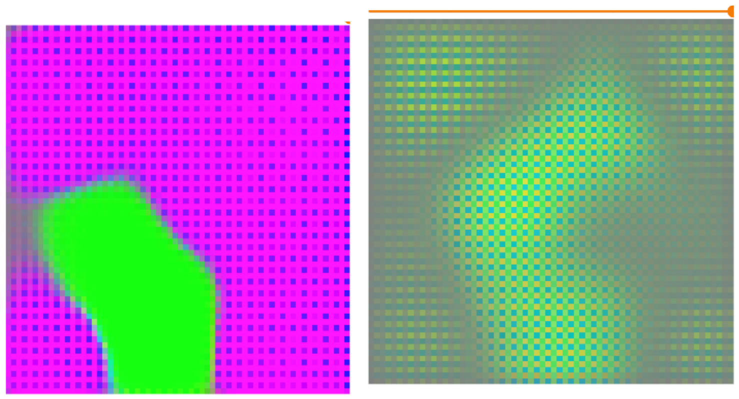




The discriminator's loss which is equivalent to GAN's loss is shown below. While GAN's loss increases at the start and after intervals of 2k iterations it seems to coincide with the introduction of new frames leading to GAN's discriminator labelling it falsely.



A sample of the reconstructed image by VAE-GAN is shown below(on right). The image misses quite a few features of the original image, however, seems to be producing an image of sharper quality as compared to VAE.



Conclusion

Combined VAE-GAN does seem to provide interesting results in terms of the reconstructed images, and we would have loved to do a comprehensive comparison using a larger sample size. We would have loved to be able to integrate into the complete world-models pipeline with results from VAE-GAN being fed into MDN-RNN and controller units however, owing to time constraints performance the same was not possible. In future, we would love to train the combined model and compare the scores achieved by VAE vs VAE-GAN integrated world models architecture.

References

1. Arulkumaran, K., Deisenroth, M.P., Brundage, M. and Bharath, A.A., 2017. IEEE Signal Processing Magazine, Vol 34(6), pp. 26-38. DOI: 10.1109/MSP.2017.2743240
2. Dosovitskiy, A., & Brox, T. (2016). Generating images with perceptual similarity metrics based on deep networks. *arXiv preprint arXiv:1602.02644*.
3. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.
4. Graves, A., Generating Sequences With Recurrent Neural Networks, 2013
5. Ha, D. and Schmidhuber, J. World Models, 2018
6. Hansen, N., The CMA evolution strategy: a comparing review, 2006
7. Henderson, P. Islam, R. Bachman, P. Pineau, J. Precup, D. Meger, D., Deep Reinforcement Learning that Matters, 2017
8. Irpan, A., Deep Reinforcement Learning Doesn't Work Yet, 2018
9. Kingma, D., Auto-Encoding Variational Bayes, 2014
10. Larsen, A. B. L., Sønderby, S. K., Larochelle, H., & Winther, O. (2016, June). Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning* (pp. 1558-1566). PMLR.
11. Plaat, A., Kosters, W., & Preuss, M. (2020, December 1). *Deep model-based reinforcement learning for High-dimensional problems, a survey*.
12. *Stanford CS330: Multi-Task and Meta-Learning, 2019 | Lecture 8 - Model-Based Reinforcement Learning*. https://www.youtube.com/watch?v=NBjcWPcCccA