Learning to Simulate Dynamic Environments with GameGAN

Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, Sanja Fidler

https://arxiv.org/abs/2005.12126
https://arxiv.org/pdf/2005.12126.pdf

TL;DR

GameGAN is a generative model that learns to visually imitate a desired game by ingesting screenplay and keyboard input. The GAN renders the next screen using a GAN each time a key is pressed. GameGAN has an optional memory module that builds an internal map of the environment to enable long-term consistency. GameGAN can also distinguish between static and dynamic components within an image.

Introduction

Designing good simulators for artificial agents is extremely important but challenging. It is traditionally done with time consuming and difficult manual labour. Learning to simulate by simply observing is the most scalable way going forward. This paper presents GameGAN, a generative model that learns to imitate a desired game through only screenplay and keyboard actions. GameGAN has higher quality results and has memory to allow the agent to return to previously visited locations with high visual consistency.

GameGAN

GameGAN should model both the deterministic and stochastic nature of the environment. GameGAN predicts the next image given the history of images along with actions and a stochastic variable that corresponds to randomness in the environment. It is composed of multiple parts. The dynamics engine maintains an internal state variable with past actions, randomness, and inputs. Environments that require long-term consistency, can use an external memory module. The rendering engine produces the output image given the state of the dynamics engine. Adversarial losses along with a proposed temporal cycle loss is used to train GameGAN.

Dynamics Engine

GameGAN has to learn how various aspects of an environment change with respect to the given user action. For example, walking through a wall is not possible., and how other objects behave because of the action. The dynamics engine learns these relations. It has access to past history, and is thus implemented as a LSTM. It computes action, randomness, the image, and the memory vector if the memory module is used.

Memory Module
Long-term consistency is required when the simulated scene should not change when the agent comes back to the same location. This is challenging as the model must remember every scene in the past. An external memory module is used to store the info.

Rendering Engine
The rendering engine is responsible for rendering the simulated image. It can be implemented with standard transposed convolution layers. However, a specialized rendering engine architecture is used to ensure long term consistency when required.

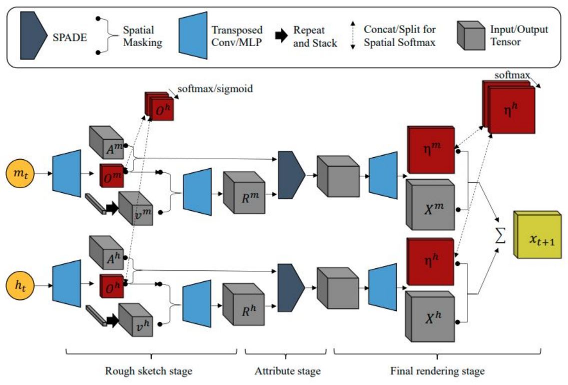


Figure 6: *Rendering engine* for disentangling static and dynamic components. See Sec 3.3 for details.

Training
GameGAN leverages adversarial training to learn environment dynamics and to produce realistic temporally coherent simulations. For long-term consistency, cycle loss is used to diesentagle static and dynamic components to learn to remember what it has generated.

To ensure each generated frame is realistic, the single image discriminator and gameGAN play an adversarial game.

GameGAN also has to reflect the actions taken by the agent. Three pairs are given to the action-conditioned discriminator. (x_t, x_{t+1}, a_t), $(x_t, x_{t+1}, \bar{a}_t)$ and $(\hat{x}_t, \hat{x}_{t+1}, a_t)$. . x denotes real image, $\widehat{x}$ the generated image, and a an action. $\bar{a}_t \in \mathcal{A}$, a sampled negative action. The job of the discriminator is to judge if two frames are consistent with respect to the action.

A Temporal discriminator is implemented as a 3D convolutional network. It takes several frames as input and decides if the sequence is generated or real. This ensures time is considered.

GameGAN is trained end-to-end. It employs a warm-up phase where real frames are fed into the dynamics engine for the first few epochs, then slowly reduces the number of real frames to 1.

Experiments

Training Data for pacman was generated from a trained DQN

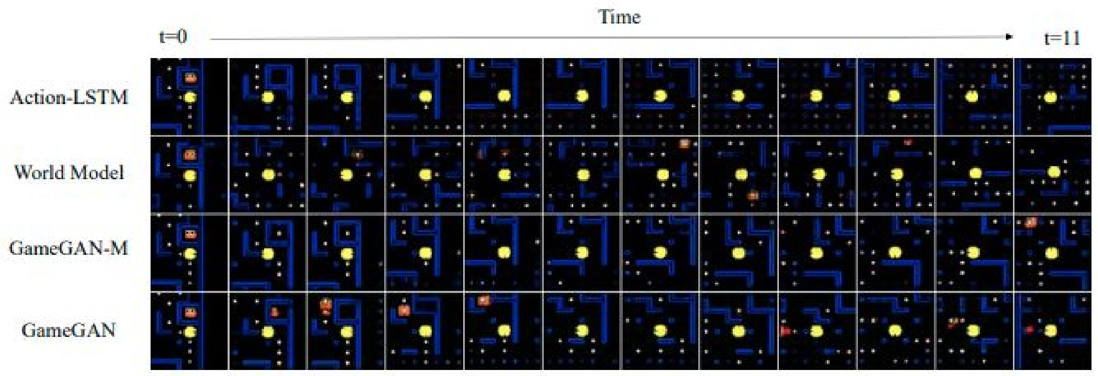
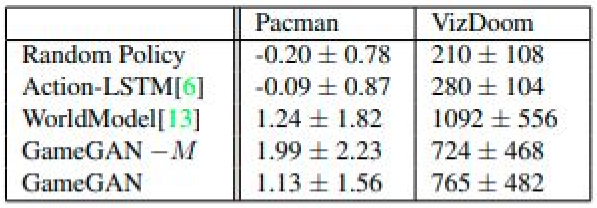


Figure 8: Rollout of models from the same initial screen. Action-LSTM trained with reconstruction loss produces frames without refined details (*e.g.* foods). World Model has difficulty keeping temporal consistency, resulting in occasional significant discontinuities. GameGAN can produce consistent simulation.

To test the simulated environment, a RL agent can be trained on the simulated environment then tested in the real environment.



	Pacman	VizDoom
Random Policy	-0.20 $\pm$ 0.78	210 $\pm$ 108
Action-LSTM[6]	-0.09 $\pm$ 0.87	280 $\pm$ 104
WorldModel[13]	1.24 $\pm$ 1.82	1092 $\pm$ 556
GameGAN $-M$	1.99 $\pm$ 2.23	724 $\pm$ 468
GameGAN	1.13 $\pm$ 1.56	765 $\pm$ 482

Table 1: Numbers are reported as mean scores $\pm$ standard deviation. Higher is better. For Pacman, an agent trained in real environment achieves 3.02 $\pm$ 2.64 which can be regarded as the upper bound. VizDoom is considered solved when a score of 750 is achieved.

Conclusion

GameGAN leverages adversarial networks to simulate games by observing screenplay and does not require access to the game's logic or engine. GameGAN has a memory module to ensure long-term consistency.