Learning to Paint With Model-based Deep Reinforcement Learning

Report

Vinayak Gupta 4th August 2021

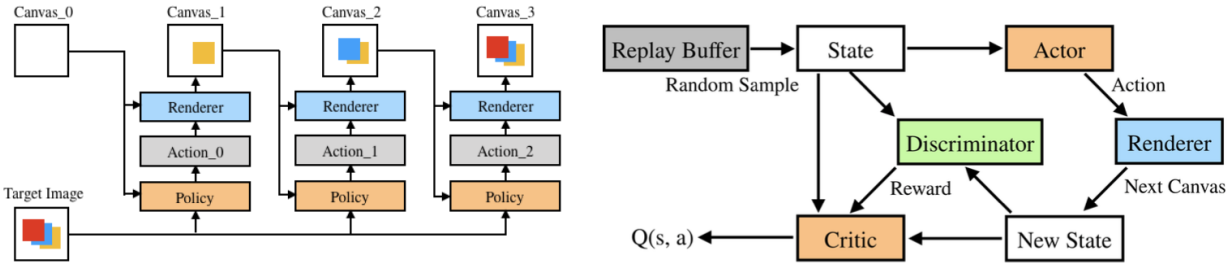
Introduction

Teaching a machine how to paint using only strokes is a difficult task since it needs to decide both on past history as well as for future strokes. To make the model decide based on future rewards, the authors decide to use model-based Deep Reinforcement Learning since it helps the model to learn using the future outcomes as well as using the history. They use a neural renderer to simulate the canvas for the model so that it can look into every possible outcome for a single action. Basically on each stroke on the canvas, the agent has to predict the stroke location, shape, stroke thickness and colour. They define a differentiable neural renderer that can back-propagate gradients for the model to learn. It does not require any supervision since it learns on its own using the simulation. This paper is a combination of GAN + RL.

Previous Works

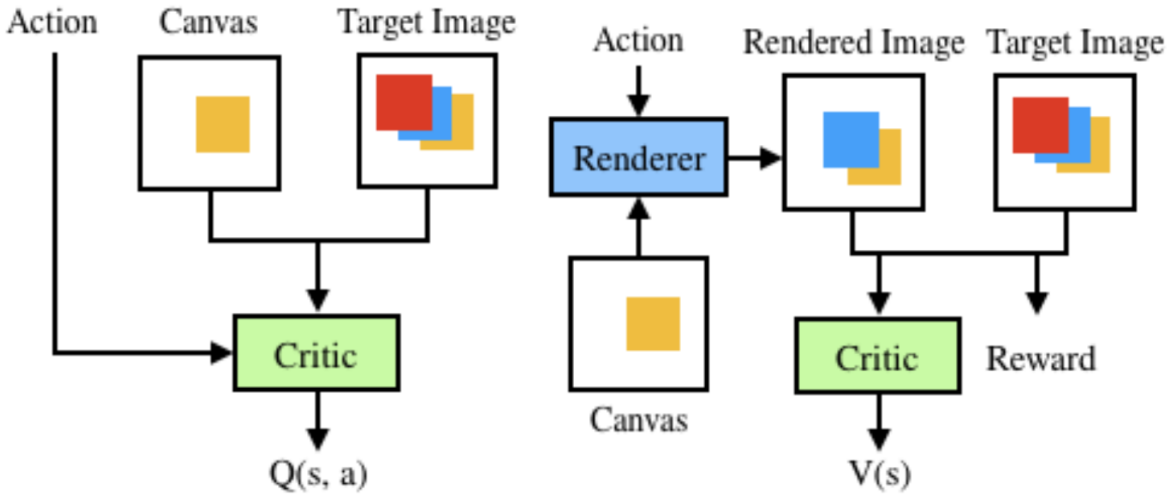
- **Synthesizing programs for images using reinforced adversarial learning: Paper**

Model



State space in this environment is made up of 3 parts: The current state of canvas, target Image and step number.
The next state is defined by s(t+1) = trans(s(t), a(t)), where the trans function is the function which applies the action a(t) on the current canvas state s(t)
The action space of the agent is made up of 4 parameters namely colour, shape, thickness and location. Basically the agent acts according to a policy function π which maps from state space to action space.
The reward is modelled such that it can measure the difference between current canvas and the next canvas.
$r(s(t), a(t)) = L(t) - L(t+1)$ where L(t) is the loss between the current canvas and the target image and L(t+1) is the loss between the next canvas and the target image. To make the final canvas resemble the target image, the agent should be driven to maximize the cumulative rewards in the whole episode.
We use Deep Deterministic Policy Gradient(DDPG) to help the actor and the critic develop a best policy and also to predict the estimated reward correctly respectively. Usually the data is actually the data from the replay buffer but it would be tough for the agent to just learn from the replay buffer since it has to model complex

structures and hence they design a neural renderer so that the agent can observe a modeled environment. Then it can explore the environment and improve its policy efficiently



(a) Original DDPG (b) Model-based DDPG

At step t, the critic takes s(t+1) as input rather than both of s(t) and a(t). The critic still predicts the expected reward for the state but no longer includes the reward caused by the current action. The new expected reward is a value function V(s(t)) trained using discounted reward:

$$V\ (s(t)) = r(s(t), a(t)) + \gamma V\ (s(t+1))$$

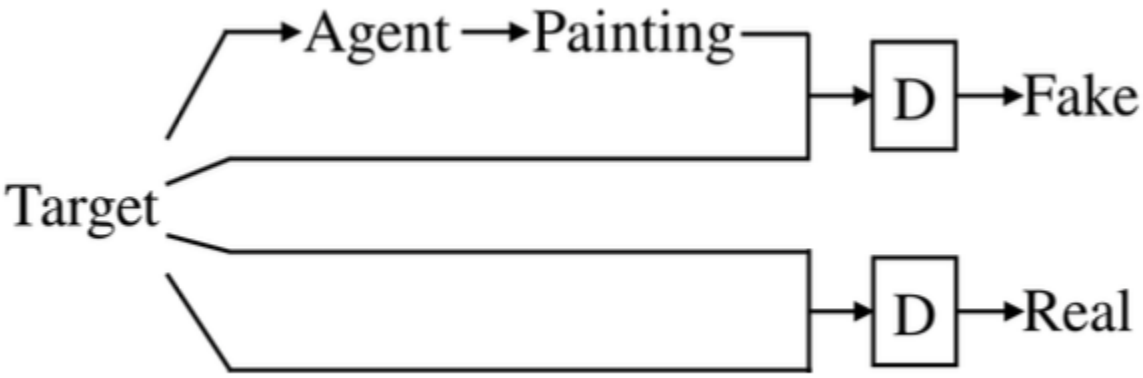
Action Bundle: To improve accuracy and also to decrease computation cost, they propose action bundle. Rather than the agent observing the environment and perform the action each frame. We could let the agent observe the environment for k frames using the same previous action and then after k frames it acts. This practice encourages the exploration of the action space and action

combinations. The renderer can easily render k strokes at a time rather than k strokes every single time individually.

They use Wasserstein GAN loss to predict the reward for every time step. The objective or loss for WGAN is defined as follows

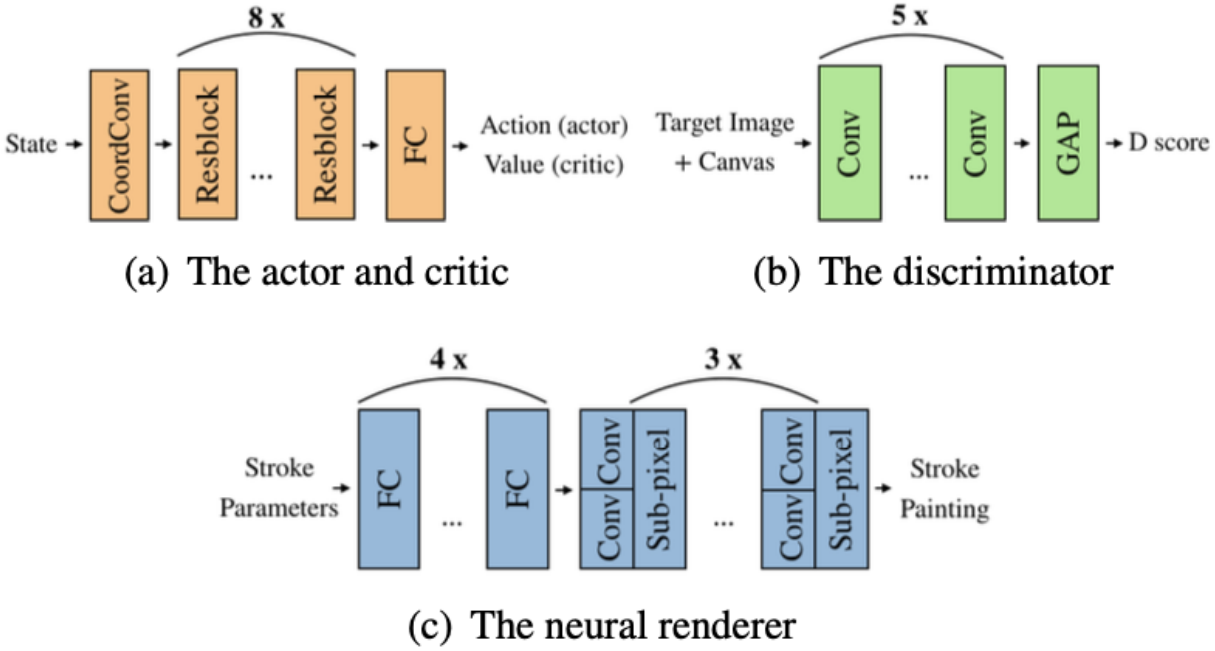
$$\max_D \mathbb{E}_{y\sim\mu}[D(y)] - \mathbb{E}_{x\sim\nu}[D(x)]$$

The fake examples are the pairs of the canvas paintings by the agent and the target image whereas the real images are the pairs of same target image. They also constraint that D should be under the constraints of 1-Lipschitz.



This is the loss used to calculate the reward which we saw earlier.

Neural Render: The neural renderer is made up of some fc layers and sub pixel upsampling layers where the input is the stroke parameters and the output is the rendered stroke image.



(a) The actor and critic

(b) The discriminator

(c) The neural renderer

The stroke design they choose is Quadratic Bezier curve(QBC) with thickness to simulate the effect of brush. The stroke is defined as:

$$a_t = (x_0, y_0, x_1, y_1, x_2, y_2, r_0, t_0, r_1, t_1, R, G, B)_t$$

where (x0,y0,x1,y1,x2,y2) are the coordinates of the three control points of the QBC. (r0, t0), (r1, t1) control the thickness and transparency of the two endpoints of the curve, respectively. (R, G, B) controls the color. The formula of QBC is

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, 0 \leq t \leq 1$$

Dataset

- MNIST
- SVHN

- CelebA
- ImageNet

Training

To learn more about the training paramerters and stuff check out the paper. They have nicely explained the architecture with neat diagrams and have given the hyperparameters very organised

Results

End Note

To know more about Painting with RL, check out the paper: [Painting with RL]

To check out the Pytorch Implementation of the model, check out my GitHub Repo: [GitHub Repo]