

Vinci – A first step towards Creativity in Robots

Omkar Joglekar – 932731094

omkarj@mail.tau.ac.il

Abstract

The early signs of creativity in human toddlers can be observed through the sketches they make. There exist many classifications and definitions people have attempted in the past for creativity [\[10\]](#). However, the topic of human creativity as a fundamental aspect of human thinking, can be best understood through a “7 C’s” approach [\[1, 2\]](#): Creators (person-centered characteristics), Creating (the creative process), Collaborations (co-creating), Contexts (environmental conditions), Creations (the nature of creative work), Consumption (the adoption of creative products) and Curricula (the development and enhancement of creativity). Through this project, we will aim to create a system able to show qualities relevant to these 7 C’s.

In general, our first step towards creativity in robots relies on the existence of a human-like “Creator” capable of following a certain reward function to generate a sketch. In this project we attempt to create a deep learning architecture that “learns how to be creative” by adapting an existing sketching algorithm from supervised learning to deep reinforcement learning. To get this architecture to learn to be creative, we need to eventually design a reward function which will promote creative thinking in the creator. As a first step, we tested the functionality of the model on two simple reward functions – “maximizing pixels” and “maximizing black pixels”

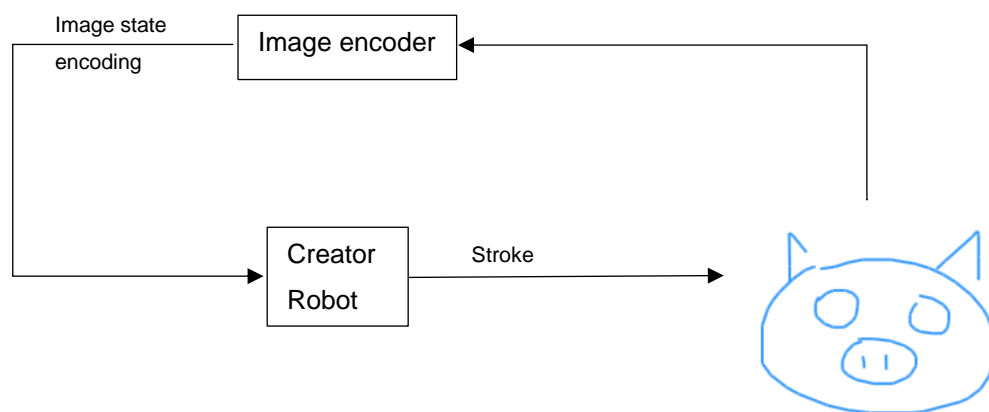


Figure 1: Block Diagram of a general creative agent

Introduction

Creators refer to all those who engage in creative thinking. In fact, every human being can be characterized as a creator and as “creative” to some degree. We tend to think spontaneously of great, eminent creators such as Leonardo da Vinci or Pablo Picasso. However, these eminent creators represent the pinnacle of a much larger set of creative people, who deploy their original thinking in their everyday lives and work [\[3\]](#).

Creativity is one of the most important and pervasive of all human abilities. However, it seems to decline during school age years, in a phenomenon entitled “creative crisis”. As developed societies are shifting from an industrialized economy to a creative economy, there is a need to support creative abilities through life. It is shown that toddlers interacting with social robots show boosted creative abilities [\[4\]](#).

According to the 7 C’s, Collaboration is an important creativity feature. The goal of this project is to make a creative social robot and let it interact with human toddlers to observe the impact on the creativity of the toddler. This project discusses creativity related to sketching, specifically.

The first step is to have a “Creator” agent that can learn to be creative. This creator will eventually be trained using a specifically engineered reward function. First, we need to verify whether the current architecture can follow reward functions and converge to an optimal policy. In this work we have tested two different reward functions – “pixel maximiser” and “black pixel maximiser”. We will show that the agent succeeds in learning an optimum solution for either of these, with some given constraints. This also presents the “Curricula” and “Creating” features of 7 C’s.

The major contribution of this work is adapting the legacy Sketch-RNN architecture that is capable of learning creative sketching through supervised learning, to a reinforcement learning setup (DDPG) which can successfully follow specific reward functions. This adaptation of the architecture will eventually enable us to make the agent follow much more complicated reward functions to achieve the goal of thinking creatively.

We allowed the agent a maximum of 200 strokes with a restriction on the length of the stroke. The stroke width was fixed to a single line. Each sketch was

padding with a 50 white pixels "border". Given these constraints, the best solution for the first reward function is the longest possible diagonal line and that for the second reward is a the longest possible 1-D line.

Related Work

Our work is strongly based on SketchRNN [5] – a sequence-to-sequence variational autoencoder (VAE) for generating sketches in a stroke-by-stroke manner; and Deep Deterministic Policy Gradients (DDPG) [6] – an off-policy, model-free, deep Q network (DQN) based deterministic policy gradients (DPG) based reinforcement learning algorithm. We also use some ideas from the sketch-pix2seq [7] – a modified version of the SketchRNN, which uses a CNN to encode the latent space.

1. Reinforcement learning and DDPG

DDPG is an actor-critic based reinforcement learning algorithm, where the actor performs actions on a simulated environment based on the current state of the environment and learns the best policy, $\mu(s, t)$, where μ is a deterministic policy based on ‘s’ – the current state and ‘t’ – time. The critic learns a Q-value function $Q(s, a)$ which assigns a value to the current action taken based on current state and action. The algorithm also implements two supplementary networks called the “target networks” – target actor and target critic. After each training step of the actor and critic networks, these networks are updated “softly”, meaning with a factor $\tau (< 1)$. These networks help in the convergence of the algorithm.

In each episode of the DDPG, the actor generates the data tuples (s, a, r, s') where ‘s’ is the current state, ‘a’ is action taken, ‘r’ is the reward received from the environment and ‘s’ is the updated state. These tuples are stored in a replay buffer. Since the policy is deterministic, the exploration is done using a stochastic action noise process. The paper [6] recommends using Gaussian Ornstein-Uhlenbeck (OU) noise process for this purpose. After the data are collected, a step of training is done on a randomly sampled mini batch from the replay buffer. A step of training involves updating the critic with the loss calculated according to the equation –

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$$

Where y_i is given by –

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'})$$

Where N is the number of samples in the minibatch, r_i are rewards, γ is the discount factor, Q is the critic network, Q' is the target critic network, s_i are the states, s_{i+1} are next states, μ' is the target actor and $\theta^Q, \theta^{Q'}, \theta^{\mu'}$, are the weights of the respective neural networks. The actor is then updated by using the policy gradient –

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s | \theta^{\mu}) |_{s_i}$$

All notations same as above, with μ being the actor network. After this training step, the target networks undergo a soft update –

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^{\mu} + (1 - \tau) \theta^{\mu'} \end{aligned}$$

Where τ is the update factor ($\ll 1$). The full algorithm is shown below in pseudocode (figure 2)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a | \theta^Q)$ and actor $\mu(s | \theta^{\mu})$ with weights θ^Q and θ^{μ} .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process \mathcal{N} for action exploration
 Receive initial observation state s_1
 for $t = 1, T$ **do**
 Select action $a_t = \mu(s_t | \theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'})) | \theta^{Q'}$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s | \theta^{\mu}) |_{s_i}$$

Update the target networks:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^{\mu} + (1 - \tau) \theta^{\mu'} \end{aligned}$$

end for
end for

Figure 2: DDPG pseudocode [6]

2. Sketch-RNN

Sketch-RNN uses a Bidirectional LSTM RNN based encoder to encode the stroke order into a latent space distribution 'z'. 'z' is generated by sampling a gaussian distribution with mean ' μ ' and variance ' σ^2 '. The RNN encoder learns to generate these distribution parameters. The sampled z is used to generate the hidden states of the LSTM decoder RNN and is also used as an input, concatenated with the stroke at time 't', s_t . The decoder RNN then starts generating the sketch, stroke by stroke, with the input to the next LSTM cell being the previous stroke concatenated with z.

Each stroke is generated by sampling a Mixture Density Network [8] that is generated by the decoder. The full architecture of the Sketch-RNN is summarized in figure 3 below. As a result of the architecture and the training procedure described in the paper, Sketch-RNN learns to create a sketch "subject to" a given input sketch. This concept is used in our creator to observe the process of sketching which it performs. In the future, we wish to use this agent for multiclass sketching, which is a major flaw in Sketch-RNN. The improved model in sketch-pix2seq gets rid of the KL divergence-based loss in latent space encoding, to improve multiclass sketching. We will use these findings in our agents.

We have utilized the sketch RNN as the connection of two components – the encoder (which generates the states (within the environment)) and the decoder (which performs the action (a stroke)).

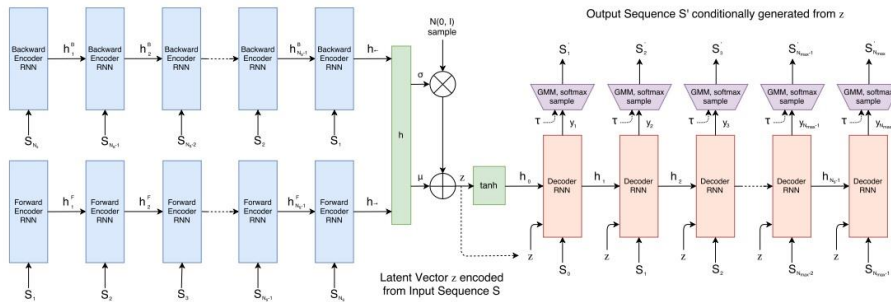


Figure 3: Sketch-RNN architecture [5]

Data

In the implementation of the DDPG algorithm, data is generated by the actor performing actions on the environment based on a current state and receiving the reward and next state from the environment. This data is recorded in the Replay Buffer as a tuple (s, a, r, s') . In each episode, we first generate these tuples for T timesteps and then sample a minibatch and run the training step. In our implementation, we defined the action(a) as an individual stroke, encoded as $(dx, dy, p1, p2, p3)$, where dx and dy are spatial offsets in the x and y direction. $p1, p2, p3$ are pen states signifying – pen touching the paper, pen not touching the paper and end of sketch respectively (as in the Sketch-RNN paper). We defined the states as the latent space encoded by the encoder RNN concatenated with the previous action stroke.

Methods

Our whole model can be summarized in the block diagrams below (figure 4 and 5). We created a gym environment for the simulation of the sketching process. In our setup, the actor and critic networks are both based on the decoder RNN networks adapted from the Sketch-RNN architecture. The actor generates mixture density coefficients while the critic generates the respective Q-values. The encoder situated inside the Sketch Environment is based on the encoder RNN network of the Sketch-RNN. This encoder is used to determine the latent state space for the input of the actor in the next time step. The interactions of these three networks follows the procedure where the actor and critic interact with this environment, generate the data tuples, get rewards, and learn according to the DDPG algorithm. We defined the action space as a 5-D stroke vector (dx , dy , $p1$, $p2$, $p3$) where ' dx ' and ' dy ' are the stroke offset components that are continuous coordinates in nature. ' $p1$ ', ' $p2$ ', ' $p3$ ' are the pen-states denoting "touching the paper", "not touching the paper" and "finished current sketch" states respectively. The most crucial motivation for using the DDPG was the fact that it works with a continuous action space that is required to learn the offset components of our action space.

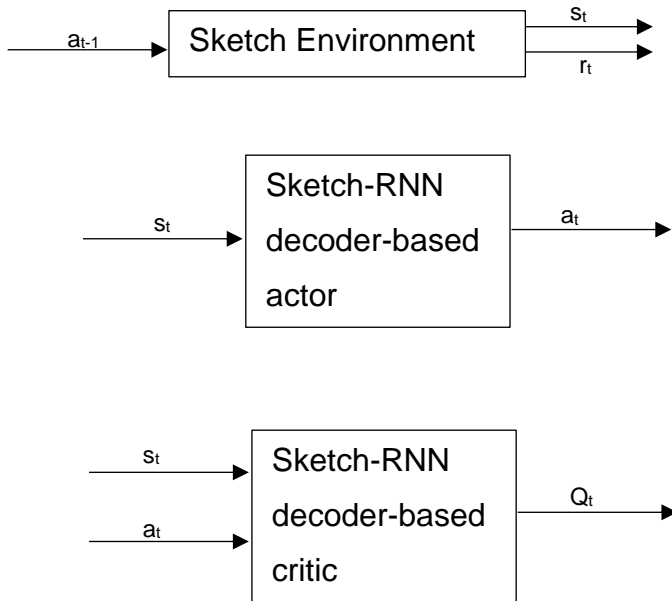


Figure 4: Experiment setup

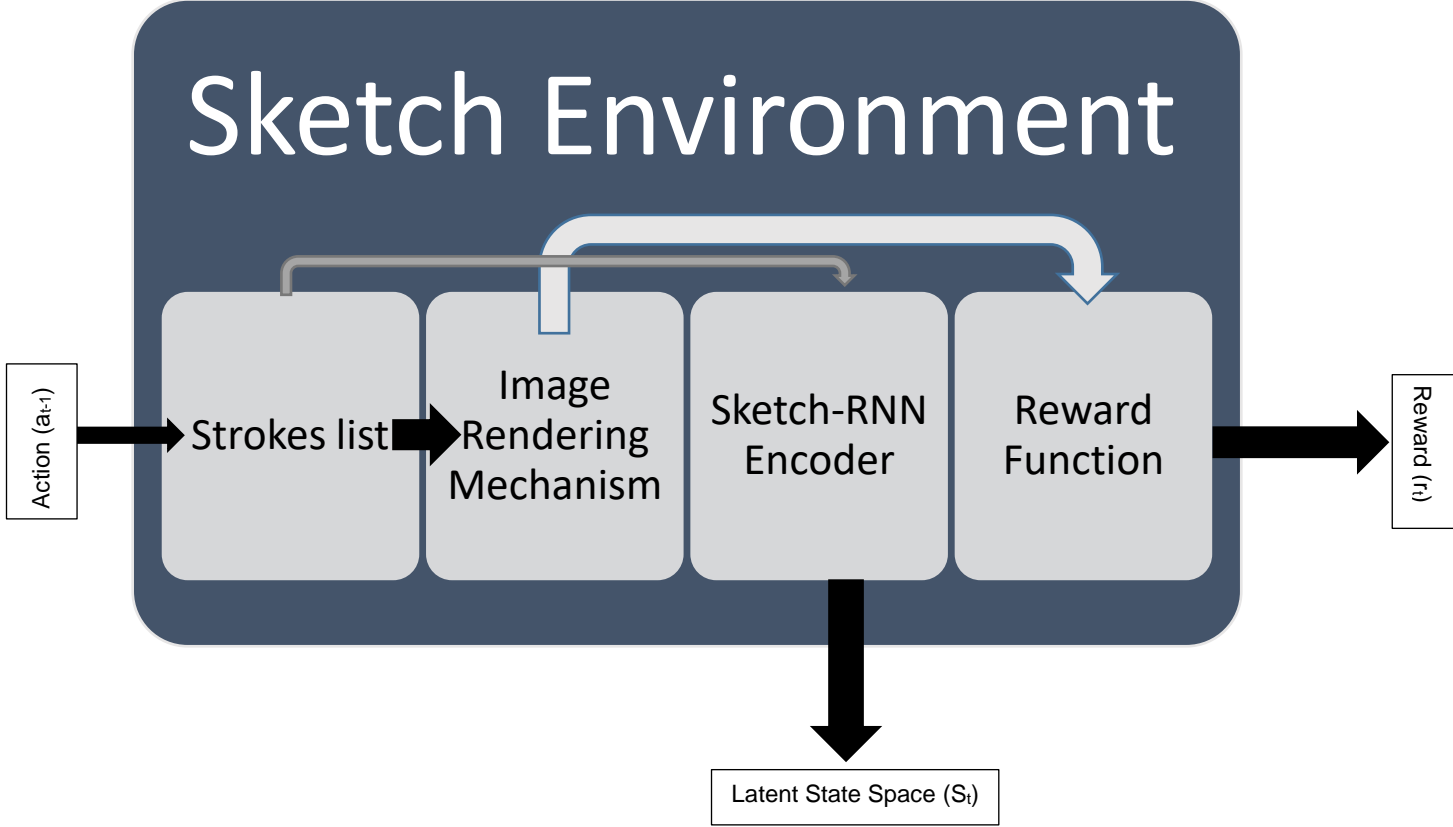


Figure 5: Block diagram of Sketch Environment

The simulation environment is a custom environment created using the “gym” libraries in python. According to the setup, the “state” of the environment is given by the latent space representation generated by the Sketch-RNN encoder based on previous strokes. Each “action” is the 5-dimensional stroke vector generated by the Sketch-RNN decoder. The “done” boolean is the pen state p3 according to our stroke definition above. The “reward” functions we are using are summarized as follows –

1. The Pixel Maximizer reward function:

$$reward = (N_{gen} - N_{ref}) / N_{ref}$$

Where N_{gen} is the number of pixels in the generated sketch and N_{ref} is the number of pixels in a reference image. We set the reference image to be –



Figure 6: Reference image for pixel maximizer reward

With dimensions – 90x158 (total 14,220 pixels).

2. The Black Pixel Maximizer reward function:

$$reward = (2 * N_{gen}^B - N_{gen}) / N_{gen}$$

Where, N_{gen}^B are the number of black pixels in the generated sketch and N_{gen} are the total number of pixels in the generated sketch (minus the white “border” pixels).

All rewards are then clipped to lie between 1 and -1.

Sketch-RNN already incorporates a lot of creative functionalities which we aspired to harness. The major contribution of our method is adapting the Sketch-RNN model to a reinforcement learning setup capable of providing a similar if not better quality of performance on creative sketching applications.

Experiments

We carried out two simple experiments on the models we constructed. In one of the experiments, we made the creator sketch an image that has a high number of pixels and in the other experiment we made it maximize the number of black pixels. Tuning the reward function to fit each experiment objective, according to the methods discussed above.

We defined 200 to be the maximum number of strokes per episode, with each episode being an individual sketch. The maximum offset in a stroke was limited to 12.5 and 12.5 in each dimension. The sketch was reset at the beginning of each episode. We chose to use the OU action noise as the DDPG exploration noise as recommended by the authors. We chose the process to have zero mean and 0.2 standard deviation. We chose actor and critic learning rates to be 0.01 and 0.02 respectively. The discount factor(γ) was 0.99 and the target networks update factor(τ) was 0.01. The encoder LSTM has 256 cells, and the decoder LSTM has 512 cells. Batch size was 64 and total number of episodes was 20 for the first reward and 100 for the second.

At the beginning of each episode, the environment is reset with a blank sketch. In each episode the decoder outputs an action, which is fed into the environment, a sketch is generated, the latent space is updated, and a reward is calculated. Each episode is of a maximum of 200 strokes in length, with early stopping if $\text{done}=1$. This environment tries to simulate the actual behavior of the creator robot.

We trained the model for 20 episodes for the first reward and 100 episodes for the second reward. After the training, we ran the target actor model for 1 episode in the environment, to generate the ground truth sketch. The reward at the end of each episode was plotted (since this is the reward the final sketch received, it is the only one relevant to us), vs the total number of episodes. The resulting ground truths and reward graphs are attached below (figure 7, 8).

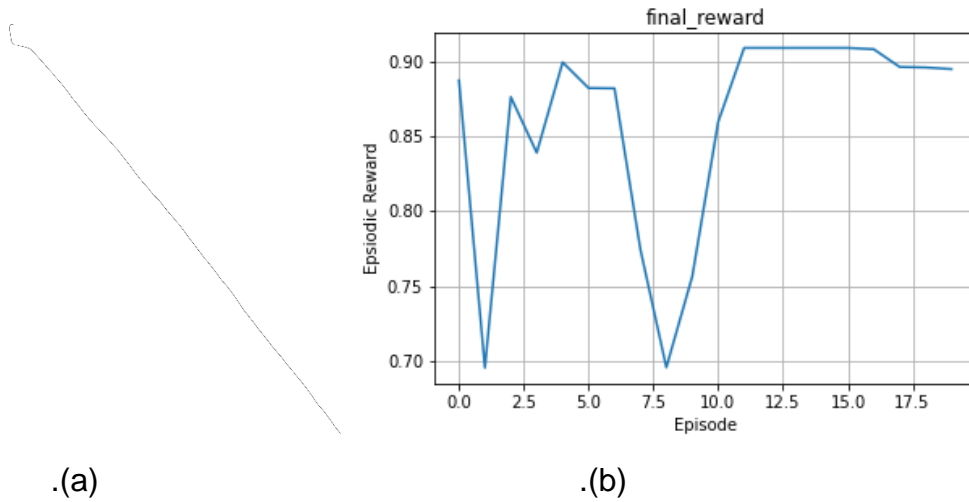


Figure 7: Pixel Maximizer (a) Ground Truth sketch (Dimensions – 1160x1421 (total 16,48,360 pixels)); (b) Reward Graph

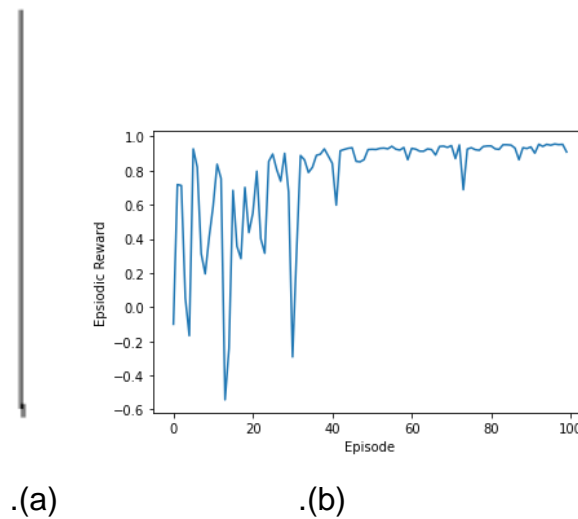


Figure 8: Black Pixel Maximizer (a) Ground Truth Sketch; (b) Reward Graph

We tested the agents on two different reward functions. The best possible policy which the actor can learn in case of Pixel Maximizer reward is a diagonal line at 45 degrees incline. We saw that the target actor eventually learned this policy and the sketch it outputs is a diagonal line. This was achieved in only a matter of 20 episodes, with more training it should achieve even better results. The reward for this training session is also increasing monotonically, except one point. This is a very common phenomenon seen in model-free deep reinforcement learning (DRL) algorithms [9]. DDPG being a model-free DRL algorithm, also faces these issues especially for sparse reward functions.

Eventually, however, the target actor did converge to a policy giving high reward of ~ 0.9 at the end of an episode.

The Black Pixel Maximizer reward function gives a higher reward for a higher ratio of black pixels in an image relative to the total pixels in the image. The ideal, best-case solution for this problem is a black shaded square in the middle of the frame. This is however, a very intricate policy to learn given the limitation on the number of strokes, size of strokes and the extent of exploration which can be done in a single episode. The closest optimal solution is the 1-D solution consisting of a single line. This policy is learned by the target actor by the end of 100 episodes. The maximum reward achieved by this policy is ~ 0.9 . This graph also exhibits the instability in the training process, like the above reward function.

Conclusions

We see that the created model was able to follow two basic reward functions after training for as little as 20-100 episodes. We observed some drawbacks of the model-free implementation during training and according to [\[9\]](#), weight averaging is a probable solution of the stability of the training process. This agent and training procedure we used, can be modified to follow much more complex reward functions such as sketching a day-to-day object (for ex: a carrot).

A possible improvement to the model is to replace the Sketch-RNN encoder with a CNN based encoder as in [\[7\]](#). Their results have shown that this provides significant improvement over the classical RNN-based encoder. Another possible experiment could use CNN+RNN-based encoders, which would provide temporal as well as spatial encoding. In the current implementation, the encoder weights are pretrained and unchanged during training. Changes need to be made to train it with the actor.

There also exists work related to the importance of curiosity in enhancing human creativity [\[11\]](#) and on teaching robots to be curious using hierarchical curiosity loops [\[12\]](#). These powerful tools can also be adapted to make robot agents learn to be creative.

Other than the implementation improvements, eventually the goal of this Creator is to generate meaningful day-to-day objects given an input sketch as a starting point. For example, given a circle as an input, it will sketch a pig's face, an alarm clock, a baseball, etc. There is a very well-known creativity test created by Dr. E. Paul Torrance [\[13\]](#) which contains a shapes task, in which subjects are given simple shapes and are asked to use them or combine them in a picture or to complete a partial picture. The more unique but complete pictures a subject can generate from a single input is supposed to be directly correlated with the extent of their divergent thinking capabilities which are a very significant part of creative thinking. The eventual goal of this project is to participate and perform well in this test.








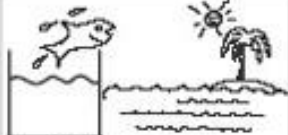

	Starting Shapes	Completed Drawing	
		More Creative	Less Creative
Use		 Mickey Mouse	 Chain
Combine		 King	 Face
Complete		 A fish on vacation	 Pot

Figure 9: Torrance test of Creative Thinking

References

- [1] Lubart, T. (2017). The 7 C's of creativity. *Journal of Creative Behavior*, 51(4), 293–296. doi:10.1002/jocb.190
- [2] Lubart, T., Thornhill-Miller, B.J. (2019). "Creativity: An Overview of the 7C's of Creative Thought". Sternberg, R.J., & Funke, J. (ed), *The Psychology of Human Thought: An Introduction*. Heidelberg: Heidelberg University Publishing
- [3] Kaufman, J. C., & Beghetto, R. A. (2009). Beyond big and little: The four c model of creativity. *Review of General Psychology*, 13(1), 1–12. doi:10.1037/a0013688
- [4] Alves-Oliveira, Patrícia & Arriaga, Patricia & Hoffman, Guy & Paiva, Ana. (2016). Boosting Children's Creativity through Creative Interactions with Social Robots. 10.1109/HRI.2016.7451871
- [5] Ha, D., & Eck, D. (2017). A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*
- [6] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*
- [7] Chen, Yajing & Tu, Shikui & Yi, Yuqi & Xu, Lei. (2017). Sketch-pix2seq: a Model to Generate Sketches of Multiple Categories.
- [8] Bishop, C. M. (1994). Mixture density networks.
- [9] Nikishin, E., Izmailov, P., Athiwaratkun, B., Podoprikin, D., Garipov, T., Shvechikov, P., ... & Wilson, A. G. (2018). Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*.
- [10] Mekern, Vera & Hommel, Bernhard & Sjoerds, Zsuzsika. (2019). Computational models of creativity: a review of single-process and multi-process recent approaches to demystify creative cognition. *Current Opinion in Behavioral Sciences*. 27. 47-54. 10.1016/j.cobeha.2018.09.008.
- [11] Gross, Madeleine & Zedelius, Claire & Schooler, Jonathan. (2020). Cultivating an understanding of curiosity as a seed for

creativity. Current Opinion in Behavioral Sciences. 35. 77-82.
10.1016/j.cobeha.2020.07.015.

- [12] Gordon, G., Ahissar, E.: Hierarchical curiosity loops and active sensing. Neural Networks 32, 119–129 (2012)

Links:

- [13] Torrance test of creative thinking:
https://en.wikipedia.org/wiki/Torrance_Tests_of_Creative_Thinking#Non-verbal_tasks
- [14] The Sketch-RNN implementation in Keras was inspired by the code repository—
<https://github.com/MarioBonse/Sketch-rnn>
- [15] The Custom gym environment was created using the guide—
<https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e>
- [16] The DDPG code reference is—
https://keras.io/examples/rl/ddpg_pendulum/

Appendix

All code is located on the drive link:

https://drive.google.com/drive/folders/1cB588cdAPZa-UP45BskYgnA4_YxvFOWQ?usp=sharing