# CS 285 Final Report: Reinforcement Learning and Reconstruction

Kevin Ma, Edward Zeng

## Abstract

Advances in deep reinforcement learning algorithms in recent years have led to impressive results. These algorithms have been applied to the field of robotics, computational biology, natural language processing, computer vision, and more. However, there has been a notable void in the applications of deep reinforcement learning in the field of engineering design. This is primarily due to the fact that key components of engineering design, which involve sketching, constructing, and ideating, are tasks that are easy for humans but very difficult for computers. In our final project, we attempt to bring deep reinforcement learning into the field of engineering design by identifying whether a deep reinforcement learning algorithm can learn to reconstruct in a similar manner as engineering designers.

In our final project, we went through two iterations of a Computer Aided Design (CAD) reconstruction gym environment. One environment does CAD reconstruction through the use of different variants of an intersection of union (IOU) for our reward function to calculate the goodness of a CAD reconstruction. The second environment utilizes generative adversarial networks (GANs) as our reward function to determine the goodness of the results for each CAD reconstruction sequence. In both cases, we trained the environments on a deep reinforcement learning actor-critic algorithm.

In the first environment, we attempted several variants of the IOU function as our reward system. However, we determined that the action space for the IOU function is too large, and we were unable to generate any sensible results. For our second environment, we narrowed the action space significantly by constraining the construction of rectangles to be only two specific shapes that can only translate either vertically or horizontally by 8 units or less. We also changed our reward system to utilize a GAN and a variant of a GAN that utilizes the Wasserstein distance. The second environment yielded better results when trained on the deep reinforcement learning algorithm, but it was still unable to produce any resemblance of our target construction result.

While our project was unable to generate realistic results, we accomplished a great deal by gathering invaluable insights and discoveries as a result of our experiments. For example, when the goal of the agent is to perform 2D CAD reconstruction sequences, an environment that makes use of a discriminator network as a reward function does a significantly better job than the more traditionally used reward function of an IOU. Additionally, different variants of GANs and their own unique forms of a discriminator network can drastically improve the performance of the results. Finally, our work can be used as a preliminary study for larger projects with 3D CAD reconstructions that reconstruct models in real time to generate large scale generative 3D CAD models. Future work that builds on our work can be critical for advancing the field of engineering design.

## I. Background

Computer aided design (CAD) programs are used by design engineers to prototype ideas. In CAD softwares, designs are created by specifying a sequence of 2 dimensional and 3 dimensional (2D and 3D) modeling operations. Recovering this sequence from a final design is known as CAD reconstruction.

A natural research topic is the application of learning-based methods to CAD reconstruction. Important recent work by Willis et. al. used imitation learning to train an agent in reconstruction [1]. To do so, Willis et. al. created a Fusion 360 Gallery gym environment and utilized a dataset [2] of ~8000 designs and their corresponding sequences of modeling operations.

However, imitation learning is only effective when human experts provide data that machine learning agents can learn from, which is a time consuming and costly process. We wanted to apply deep reinforcement learning to 3D CAD reconstruction, and our initial idea was to use the Fusion 360 Gallery gym as our environment and train a reinforcement learning agent on it.

Unfortunately, in our conversations with Karl Willis, the author of the Fusion 360 gym environment [1], we found out that the Fusion 360 Gallery gym required the use of a computationally expensive software, AutoCAD. He suggested we simplified our problem statement because we did not have the computing resources to use the Fusion 360 Gallery gym.

Our remaining option was to create toy environments for CAD reconstruction. These environments are much simpler than Fusion 360 Gallery, but they capture the basic idea and challenge behind CAD reconstruction. We created two environments: Environment A and Environment B. The rest of this report goes into more detail about these environments and how well the deep reinforcement learning agents performed on these environments.

## II. Environment A

In Environment A, the agent was given a picture and had to find the fewest number of rectangles that approximated the picture. We specified the agent's action as such:

1. Environment: 10x10 matrix of 1s and 0s (i.e. a bitmap) representing a picture, which we call the original picture, along with a 10x10 matrix of 1s and 0s representing a reconstructed picture.
2. Action: Draw a rectangle (a block of 1s) on the reconstructed picture.
3. Reward: See later section.
4. Rollout length: 5.

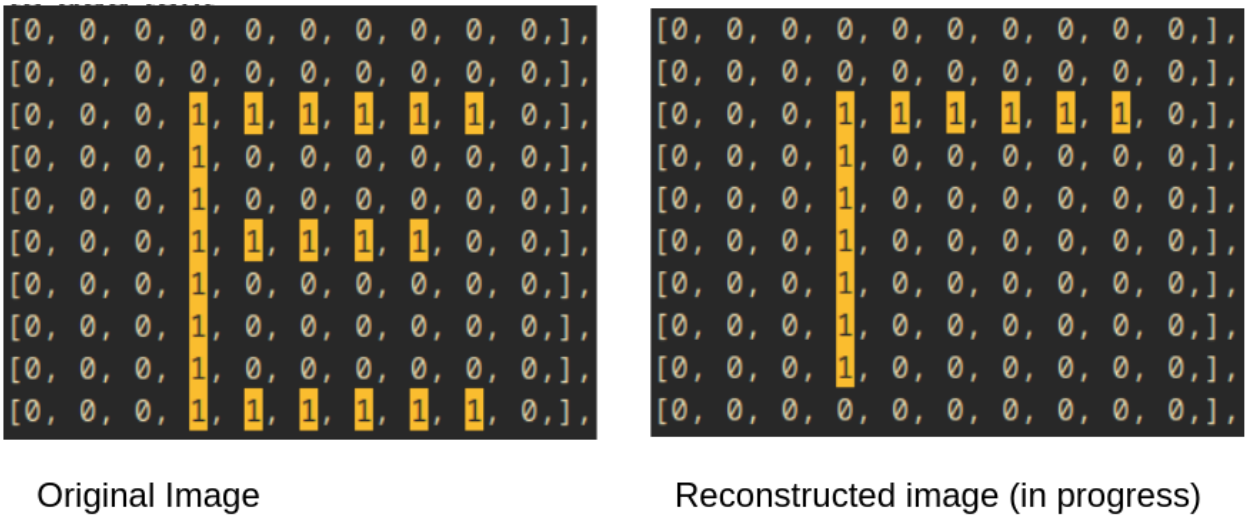


**Figure 1.** Possible environment state.

Although this may seem very different from normal CAD reconstruction, it captures the same spirit. In CAD reconstruction, one looks for a small number of commands to recreate a given final picture or geometry. The commands that CAD provides include complicated actions like sketch and extrude. In our environment, we allow only one kind of action – draw a rectangle. In doing so, we restrict ourselves to 2D geometries and a smaller set of possible actions, making our environment much easier to approach with deep reinforcement learning as well as easier to debug.

We had to limit the rollout length to a small number (like 5) to force the agent to find a small number of rectangles to approximate the picture. If we allowed an arbitrary number of rectangles, then one simple way to obtain perfect reconstruction is to draw the original picture pixel-by-pixel. Then the reconstruction is just as complicated as the image itself, defeating the purpose of CAD reconstruction.

The code for the environment can be found at [3].

## III. Environment A Results

For training and evaluation, we used an Asynchronous Advantage Actor Critic deep reinforcement learning algorithm (A2C) from stable baselines [4]. In this section we will go through some of the reward functions we attempted and their corresponding performance results.

The first reward function we used was the intersection-over-union (IOU) reward function that was also used in [1]. IOU is defined in equation 1 below.

$$IOU = \frac{area(reconstruction\ image \cap original\ image)}{area(reconstructed\ image \cup original\ image)} \quad (1)$$

Note, IOU is a function of the time step, and, for each action, we defined the reward function to be the increase in IOU.

$$\Delta IOU = IOU(current\ time\ step) - IOU(previous\ time\ step) \quad (2)$$

When we ran our agent with this reward function with 1 million timesteps, we found out that the agent would output a rectangle equal to the size of the entire picture. In other words, the reconstruction would set all the entries in the 10x10 reconstruction matrix to 1. We believed the agent settled on this output because this simple strategy always provided some reward.

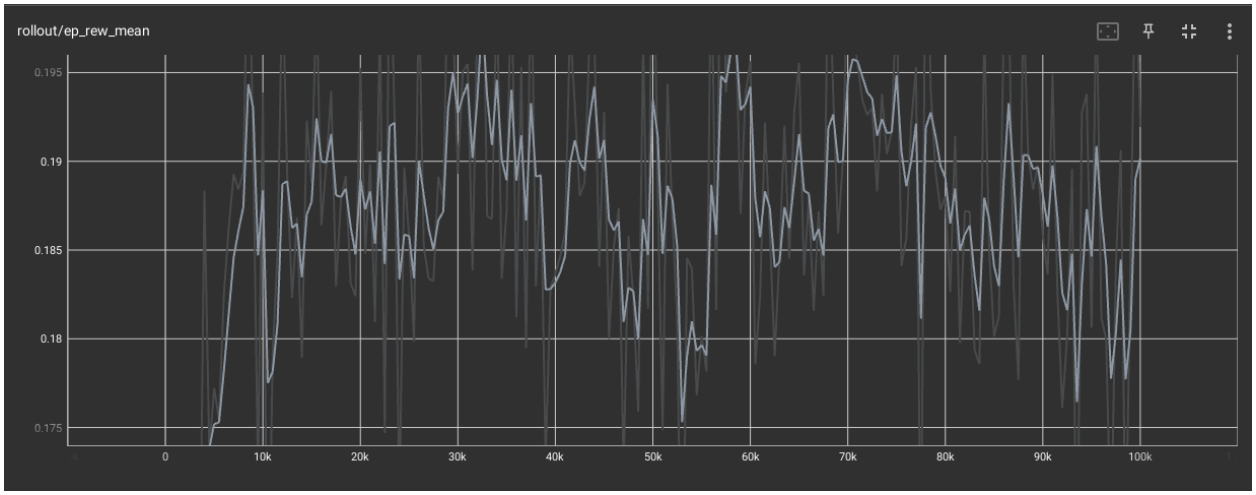


**Figure 2.** Agent reward over training for 100,000 steps with $\Delta IOU$ reward

To fix this, we decided to modify our IOU reward by penalizing the agent when they set all the entries to a value of 1. The equation for the modified IOU is shown in equation 3 below.

$$IOU_{\alpha} = \frac{area(reconstructed\ image \cap original\ image) - \alpha \times area(reconstructed\ image - original\ image)}{area(reconstructed\ image \cup original\ image)} \quad (3)$$

For each action, we defined the reward function to be $\Delta IOU_{\alpha}$. Here, the α term is used to adjust the weight of the penalty for drawing 1s outside the original picture. A variety of α terms were tested, but, in each case, the agent would output a constant reconstructed image:

**Table 1.** Results from each α value

| α value | Result after training for 1 million timesteps |
|---|---|
| 0.1 | Cover **most** pixels |

| | |
|---|---|
| 0.3 | Cover **few** pixels |
| 0.5 | Cover **no** pixels |
| 0.8 | Cover **no** pixels |

We concluded that that agent had difficulty learning a reconstruction method that would depend on the input's original picture. This task is not difficult for humans, but it is possible that the large search space was detrimental to the agent for coming up with a useful strategy.

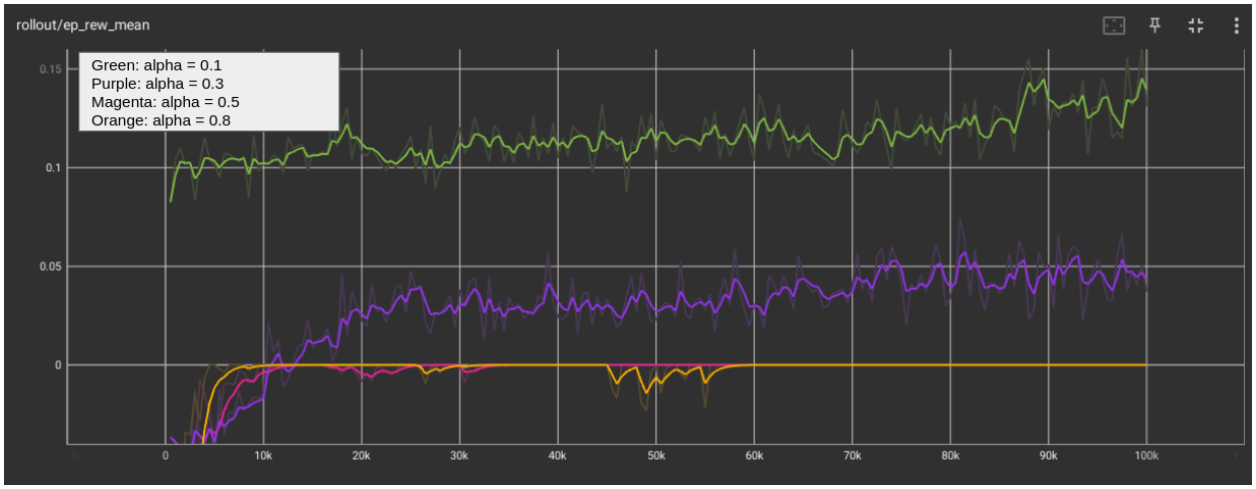


**Figure 3.** Agent reward over training for 100,000 steps with $\Delta IOU_{\alpha}$ reward.

Thus we decided to change our reward function, again, to try to guide the A2C algorithm to a good strategy in a different manner. The third reward function that we use is shown in the equation below.

$$IOU'_{\beta} = \begin{cases} 0 & IOU < \beta \\ IOU & IOU \geq \beta \end{cases} \quad (4)$$

In a certain sense, this is the correct reward function when $\beta = 1.0$ because what matters the most is whether or not the final reconstruction is correct. For our purposes, however, we set $\beta = 0.9$ to give the agent some leniency. Unlike the first reward function we considered, this function gives much less partial rewards for partial reconstructions, so there is less incentive to output a constant reconstructed image. However, even after training for 10 million iterations, the agent would still output a rectangle that covered the entire picture. We found out that the agent was unable to obtain any reward at all during training as shown in figure 4 below.

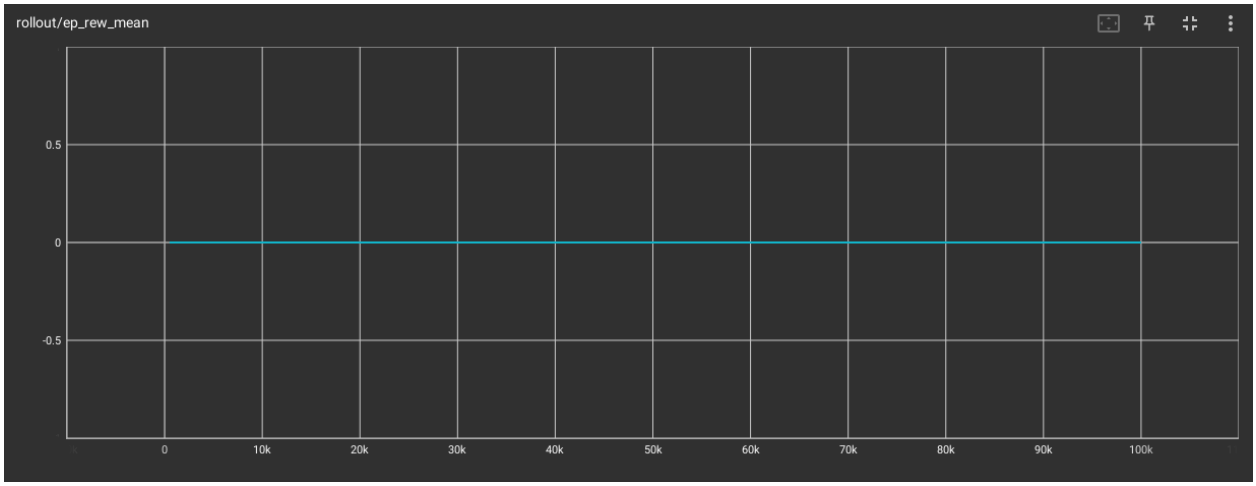


**Figure 4.** Agent reward over training for 100,000 steps with $IOU'_{0.9}$ reward.

From these setbacks, we deduced that the action space for the environment was still too big to efficiently look for a good reconstruction strategy. It is possible that increasing the number of train iterations (perhaps to 100 million or 1 billion) would yield better results, but we did not have the time or resources to do that. Instead, we decided to build a different environment to see if the A2C agents would perform better there.

## IV. Environment B

In Environment B, the agent was given a picture and had to recreate a sequence of rectangular reconstructions to match the picture. The environment is a 10x10 matrix of 1s and 0s similar to Environment A where the location of the 1s and 0s represents a picture within the 10x10 matrix. In environment B, the agent controls the placement of rectangles of size 1x3 or 3x1 (width x height) and the location of the placements. The first action, $a_1$, consists of 2 discrete decisions: 0 informs the agent to create a 1x3 vertical rectangle and 1 informs the agent to create a 3x1 horizontal rectangle. When an agent has been informed to make either a vertical or a horizontal rectangle, the agent can control the placement of the rectangle through actions $a_2$ and $a_3$, which both contain 8 discrete units that inform the agent how many units horizontally or vertically the rectangle should be translated.

Like Environment A, this is a very simplified version of an actual CAD reconstruction where a user would be able to create rectangles, circles, and other more complex geometric shapes. Likewise, in an actual CAD reconstruction, the user can control the placement of the geometric shape and make modifications to the dimensional size of the shape. However, due to the high combinatorial value of the action spaces (an important problem in Environment A), we narrowed the scope of our project to only create vertical or horizontal rectangles that can translate a maximum of 8 units horizontally or 8 units vertically.

Finally, due to issues with the Environment A, we scrapped the use of the IOUs in our reward functions and created a new reward system that made use of a Generative Adversarial Network model (GAN). This idea was inspired by a paper written by Deep Mind called SPIRAL [5].

In a GAN, there are two parts: a generator and a discriminator [6]. The generator learns to generate realistic images based on the discriminator's ability to distinguish the generator's fake images and real images. The discriminator will continually penalize the generator every time it successfully discriminates against the generator's results.

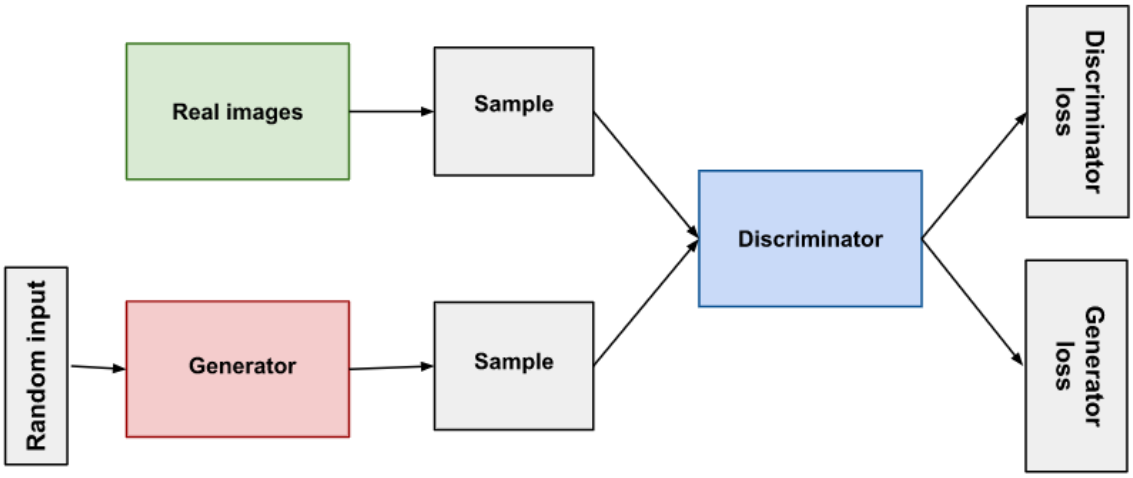


**Figure 5:** Image of the GAN model [6]

The objective of the discriminator is to minimize its respective loss function, which can be observed in the equation 5 below [7].

$$L_D = E_x[log(D(x))] + E_z[log(1 - D(G(z)))] \quad (5)$$

where $E_x$ is the expected value over the real data distribution, D(x) is the discriminator's estimate of the probability of whether or not the real data distribution, x, is real or fake, G(z) is the generator's output given noise z, D(G(z)) is the estimate of the probability of whether or not the fake data is real or fake, and $E_z$ is the expected value over all the random inputs to the generator [7]. This objective has shown to be difficult to optimize, and the results are discussed in Section 4. We also experimented with a variation of a Wasserstein value function that makes use of gradient penalties rather than weight clipping to mitigate the difficulties that normal GANs are known to have, such as vanishing gradients and mode collapse [8]. The objective for the Wasserstein variant of the discriminator is shown in equation 6 below [8]

$$L_D = - E_{x\sim fake}[D(x)] + E_{x\sim real}[D(x)] + R \quad (6)$$

where $E_{x\sim real}$ is the expected value of the discriminator sampled from the real data distribution, $E_{x\sim fake}$ is the expected value of the discriminator sampled from the fake data distribution, and R is the regularization term containing the gradient penalty weighted by a lambda value. The discriminator here will be the reward function, so the advantage actor critic algorithm will maximize the negative reward, which will minimize the loss values generated by the discriminator. This will work in conjunction with the generator, which is the policy, to optimize the algorithm to select actions that will reconstruct real images given to them.

Usually the generator is a neural network that takes in random noise to transform the noise into meaningful data as it gets classified as real or fake by the discriminator. In this case, the generator is usually defined with a loss function as shown in equation 7 below [5].

$$L_G = -E_{x\sim generated}[D(x)] \qquad (7)$$

In Environment B, however, the generator acts as a policy $\pi$ that predicts a distribution over all possible parameters and current observations. We employed a stable baselines asynchronous actor critic algorithm to optimize the policy based on the loss function shown in equation 8.

$$L_G = -\sum_t log\pi(a_t|s_t;\theta)[R_t - V^{\pi}(s_t)] \qquad (8)$$

The equation above is based on a variant of a deep reinforcement learning algorithm called actor critic where $V^{\pi}$ is an approximation to the value function and $R_t$ is the sum of all the Monte Carlo 1-sampled estimated rewards [9]. Optimizing the actor-critic loss function should recover the solution to the generator's loss function if the rewards are set to a piecewise function as shown in equation 9 below [5].

$$r_t = 0,\ if\ t < N$$
$$r_t = D(R(a_1, a_2, \ldots a_N)),\ if\ t = N \qquad (9)$$

What we found was that the inclusion of this reward modification caused the entropy loss to never change while the exclusion of the reward caused entropy loss to change very often. We present examples of both instances in the Section 4 of the report.

## V. Environment B Results

There were several instances where the agent would get stuck creating only horizontal rectangles or only vertical rectangles, so we included a forced action exploration to mediate this issue where

the reward will be penalized if the agent repeats the same actions too many times. We conducted 2 sets of experiments on 3 different combinations of parameters that include the inclusion of the reward trick shown in equation 9 and a forced action exploration. The 2 sets of experiments are divided between the use of a GAN that utilizes a Wasserstein distance and a normal GAN reward function.

We tested all our results using 100,000 training steps with an episode length of 20. Each generation was unconditional, and the goal of the agent is to reconstruct a tree from a 10x10 pixel as shown in the Figure 6 below.

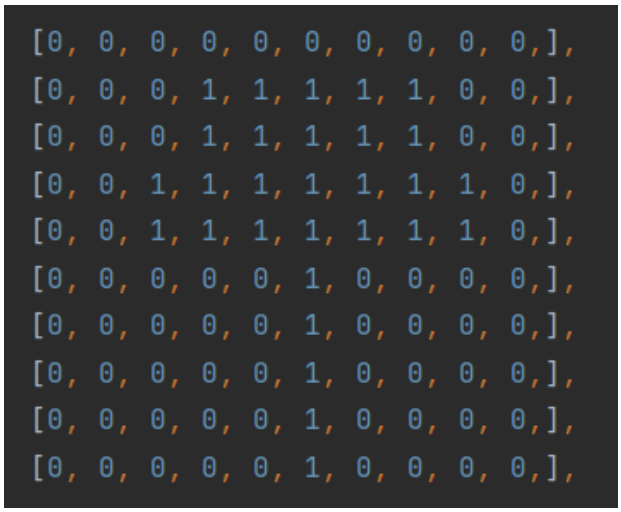


**Figure 6:** Image of the tree the agent is trying to reconstruct

The agent will learn how to reconstruct the tree by optimizing the actions as the actor critic algorithm minimizes the losses of the rewards calculated by the discriminator. Figure 7 shows the reconstructions of the 3 parameters from an actor critic algorithm that minimizes the losses of a generic discriminator.

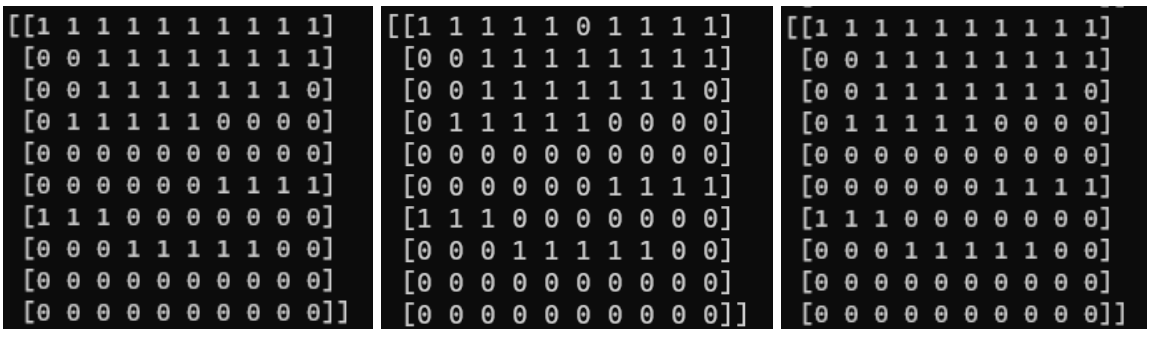


**Figure 7:** Reconstruction results from Environment B using the reward trick and action trick (furthest left), reward trick only (middle), and no reward trick or action trick (furthest right)

Using only the normal GAN discriminator reveals that the agent is incapable of doing any sensible reconstructions, and the rectangles are congregated in the top half of the observation space. However, just from a qualitative study, the results between all three parameter settings were very similar. Below we also plotted the results to investigate the different mean reward and entropy losses plotted against the timestep.

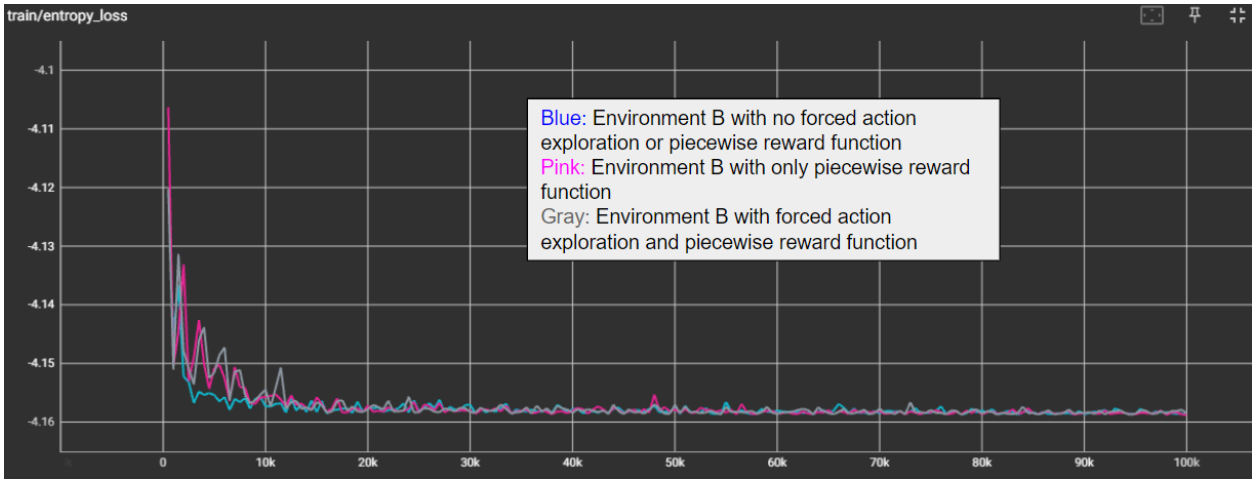


**Figure 8:** Entropy losses from the environment that utilizes a reward function with a normal GAN discriminator

Figure 8 above reveals that the entropy losses are very similar, and there are minimal changes with the entropy losses over the time step. However, when the average reward was plotted against the timesteps, something interesting happened where the average reward goes all the way to zero as shown in the figure below.


**Figure 9:** Average reward from the environment that utilizes a reward function with a normal GAN discriminator

We suspected the results of the average reward was due to a known issue with GANs called mode collapse and vanishing gradients. To resolve this issue, we opted to use a variant of a GAN that employs a Wasserstein distance as a measure of distance between the fake and real distributions [8]. In this variant of the GAN, we induced a gradient penalty on the discriminator to improve the stability of the discriminator function. We ran the same parameters as the environment with the normal GAN reward function, and the results can be shown in the figure below.


**Figure 10:** From left to right: Wasserstein with no reward or action trick, Wasserstein with reward trick only, and Wasserstein with reward and action trick

The reward function with the Wasserstein variant of the GAN still failed to produce any sensible reconstructions. However, it does have a marginal visual improvement over the reward function with the normal GAN discriminator, and the rectangles are able to explore different regions of the observation space. Below is a figure of a plot of the entropy loss of the environment with the Wasserstein variant of the GAN.

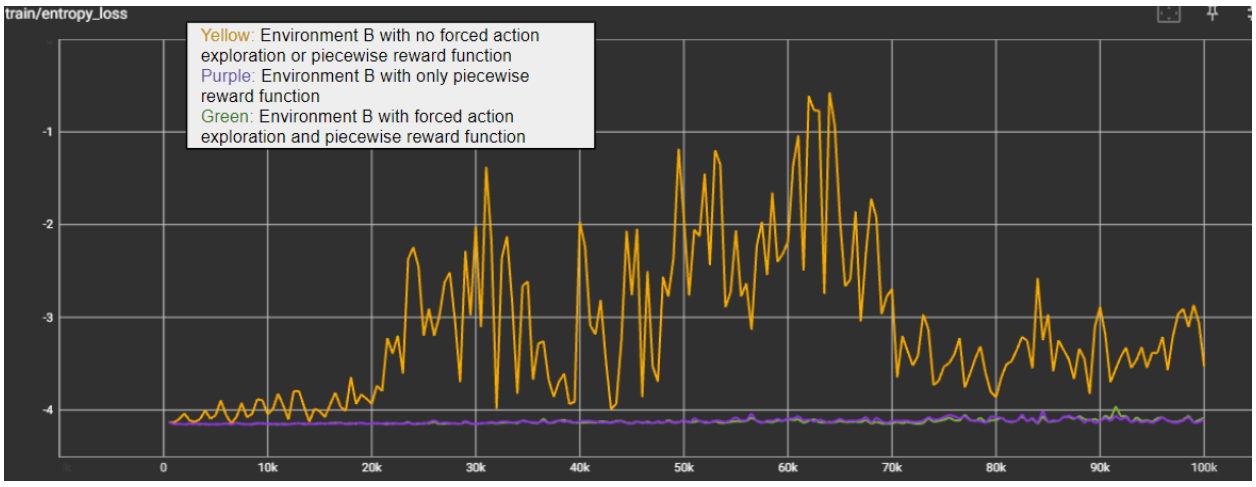


**Figure 11:** Entropy loss from the environment that utilizes a reward function with a Wasserstein variant of the GAN discriminator

It can be observed that the parameter with the highest entropy loss instability was the one without any reward or action trick while the added action trick to the reward trick had little to no effects on its entropy loss. Below is a plot of the average reward of all three parameters.


**Figure 12:** Average reward from the environment that utilizes a reward function with a Wasserstein variant of the GAN discriminator

Likewise, the Wasserstein curve without the piecewise reward function and forced action exploration had the most unstable average reward. Our findings showed that Environment B performed much better than Environment A, but the results from Environment B still did not reconstruct a reasonable depiction of a 2D tree. We note that the introduction of a piecewise reward function improved the stability of the reconstruction sequences, and, qualitatively, the environment using a Wasserstein was able to explore in different regions along the observation space rather than get stuck in the same region in the observation space.

The code for the environment can be found in our Github [10].

## VI. Conclusion and Summary

In both environments we built, the RL agent had trouble reconstructing simple images. From this, we conclude that reconstruction might not be suited for reinforcement learning due to the high number of combinations of action spaces the agent must choose from. However, a definitive answer to the applicability of RL to CAD reconstruction remains to be seen.

In Environment A, despite the use of a wide variety of reward functions, the RL agent was not able to progress beyond extremely basic reconstructions.

We had a bit more luck with Environment B, where our results indicate that using a reward function with a Wasserstein variant of the GAN that includes a reward trick produces the best

results for CAD reconstruction. While the agent was unable to produce any meaningful results, we suspect higher amounts of training steps and iterations along with modifications of other parameters in the actor critic algorithm could lead to better results. We also suspect that since the discriminator is being trained at every time step, it is possible that the discriminator is overfitting and getting stuck between specific actions, leading to poor action exploration. To mediate this issue, we attempted to do forced action exploration, but the parameter chosen for this forced action exploration could be more widely explored since the impact of the forced action exploration had little to no effect on the final results. For future work, a more thorough investigation into how to break this overfitting would help turn the reconstruction results into something more sensible. Some possible solutions to this could be introducing a replay buffer to store the trajectories of experiences in a policy to optimize the discriminator network at a higher rate than the policy since the original paper in WGAN updates the discriminator much more frequently than the discriminator [8]. This solution has been tested in a paper called SPIRAL, but it has not been tested in the domain of engineering design yet [5].

## **Contributions:**

Kevin Ma and Edward Zeng both contributed equally in this project. Edward Zeng's largest contribution is the construction of Environment A, setting up the baselines, and conducting all the experiments for the different reward functions for Environment A. Kevin Ma's largest contribution is the construction of Environment B, conducting the experiments for the different reward functions for Environment B, and performing literature reviews relevant to the construction of Environment B.

## References


[1] K. D. Willis, Y. Pu, J. Luo, H. Chu, T. Du, J. G. Lambourne, A. Solar-Lezama, and W. Matusik, "Fusion 360 gallery," ACM Transactions on Graphics, vol. 40, no. 4, pp. 1–24, 2021.

[2] Dataset website: https://github.com/AutodeskAILab/Fusion360GalleryDataset

[3] Github: https://github.com/ewzeng/cs285-final-project/tree/old

[4] Stable baselines website: https://stable-baselines3.readthedocs.io/en/master/index.html

[5] Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, A., and Vinyals, O., 2018, "International Conference on Machine Learning," Synthesizing programs for images using reinforced adversarial learning, PMLR, pp. 1666–1675.

[6] "Overview of gan structure | machine learning | google developers," Google [Online]. Available: https://developers.google.com/machine-learning/gan/gan_structure. [Accessed: 14-Dec-2022].

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2020, "Generative Adversarial Networks," Communications of the ACM, **63**(11), pp. 139–144.

[8] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A., 2017, "Advances in neural information processing systems," Improved Training of Gasserstein GANs, pp. 5767–5777.

[9] Vijay, K., and Tsitsiklis, J., 2000, "Advances in neural information processing systems," Actor-critic algorithms, pp. 1008–1014.

[10] Github: https://github.com/ewzeng/cs285-final-project/tree/master