Offline Deep Reinforcement Learning with BCQ GAN

I. Introduction

Large data sets have helped drive the growth of Machine Learning. In Supervised Learning (SL) with Deep Learning, large networks have benefited from access to huge data sets. However Reinforcement Learning (RL) typically does not use large, already gathered data sets. Data is usually gathered from scratch while training. In RL an agent is trained to perform a task by interacting with an environment. Based on the state of the environment, the agent performs actions and receives rewards from the environment. Agents often require millions of interactions with the environment, and typically discard that data after each training run. This type of RL is called online RL. In offline RL, an agent is trained with already gathered data and does not interact with the environment. However leading RL algorithms like DQN and DDPG are unable to learn effectively with offline data and require regular interactions with the environment.[1] Additionally in environments where data collection is costly, like manipulating a robotic arm, or safety concerns are prevalent, such as health care data, training an agent in an online manner is often infeasible. The need for online data prevents RL from taking advantage of huge, already gathered data sets. If RL could exploit the data-driven approach that has led to the great advances in SL, large data sets could drive progress in the field. The goal of offline RL (also known as batch RL) is to train an agent on an already existing data set, potentially allowing RL experience the data-driven revolution that has boosted SL.[2]

The central issue in offline RL is distributional shift. The agent is trained on one set of previously collected data and then tested on the actual environment. However the data the agent experiences in the environment during testing likely has a different distribution than the training set. It is difficult for the RL agent's function approximator, typically a neural network, to effectively generalize to the environment's data distribution after being trained with the offline

data. An additional issue is the counterfactual nature of the question posed to the RL agent: *what would have been the outcome of an action not taken or a state not visited?* The agent is asked to some extent not only how to value the state and action but how to value an action that was not taken.[2] These issues can take the form of extrapolation error where the agent incorrectly overvalues or undervalues a state or action.[3]

The Deep RL algorithm Batch Constrained Q Learning (BCQ) attempts to combat extrapolation error. The algorithm trains two neural networks and a Variational Auto Encoder (VAE) on the offline data in an attempt to constrain actions into a safe range as learned from the data. At test time, the agent attempts to perform actions within that constrained action range. In the paper that introduced BCQ, BCQ trained on offline data from the Mujuco simulation environments. When evaluated against other RL algorithms, BCQ was the only algorithm to succeed at all tasks and matched or outperformed the other benchmarks.[1]

The VAE used for BCQ is a vanilla VAE. Asked about why a VAE was chosen the creator of BCQ noted that a Generative Adversarial Network (GAN) may also work but noted the difficulties in training GANs and getting the GANs to generalize as well as the VAE.[3] GANs are state of the art in many Deep Learning areas when it comes to generating synthetic data similar to a given data set. GANs utilize game theory principles while training a generator neural network to produce new data that tries to fool a discriminator neural network that is trained to discriminate between the real and created data. GANs have overtaken VAE in other Deep Learning applications but it is unclear if GANs can replace the VAE used in BCQ. This paper is an attempt to improve the performance of BCQ with BCQ GAN, a modification of BCQ that replaces the VAE with a GAN.

II. Descriptions of Data

Data is gathered from three RL environments from the OpenAI gym package. In Pendulum, the agent attempts to swing a pendulum so it remains in an upright position. In LunarLander the agent attempts to safely land a machine by firing the machine's rockets. In BipedalWalker, the

agent attempts to navigate the walker across the landscape's obstacles.[4] To gather data, train and test agents are trained using the Deep Deterministic Policy Gradient algorithm.[5] The agent is trained until it reaches acceptable performance. The train and test agents then gather three collections of data for each environment. In each of the collections of data, the proportion of the time the agent behaves randomly varies. In one collection the agent does not behave randomly, in the second collection the agent behaves randomly 20% of the time, and in the third collection the agent behaves randomly 50% of the time. The three collections signify three different types of agents: near optimal (0% random action), suboptimal but with occasional optimal sequences of actions (20% random action), and a suboptimal agent (50% random action). The offline data gathered by the train and test agent is used as the offline RL data set. The train set is used for tuning hyperparameters on BCQ GAN. The test set is used for evaluation.

Ideally, an offline RL algorithm can train an agent from occasional or suboptimal actions to produce a higher performing agent. The hope is that not only will an agent trained under BCQ GAN be able to mimic an optimal agent, but that a well trained agent can learn to perform the good actions and avoid the poor actions found in a suboptimal agent's data set. An offline RL agent can 'stitch' together actions. For example in Figure 1, the offline data set has an agent going from point A to point B and from point B from point C. A successful agent can learn from that data set how to go from point A to C even though going from point A to C is not in the data set.[6]

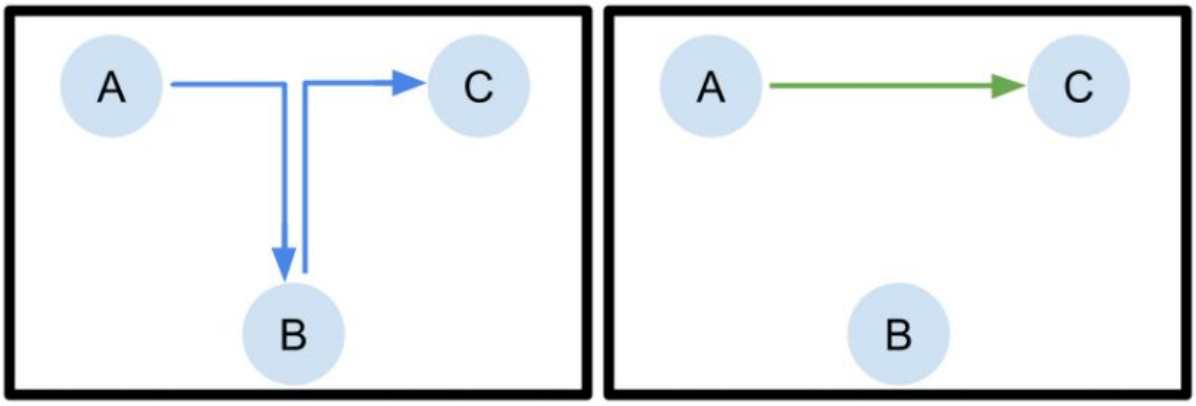


Figure 1: On the left, an the offline data set shows an agent going from point A to B and from B to C. On the right, the hope is that an agent is able to 'stitch' together different actions from the offline data set to learn a better policy.[6]

III. Model Selection and Assessment

The original BCQ algorithm contains 3 main parts. An Actor-Critic architecture in which the actor selects actions and the critic, based on the reward received, evaluates the action. Both the Actor and Critic consist of a three layer neural network. The second aspect of BCQ is a VAE, which helps constrain actions to good actions seen within the data set while avoiding poor performing actions. The third part is a perturbation model which adds a small factor to the Actor network to help expand the range of sampled actions to hopefully find better actions at train and evaluation time.

In this project, I replace the VAE with a GAN. I try three different varieties of GANs. The first is a Conditional GAN (CGAN). In a CGAN, the generator tries to generate data and the discriminator discriminates conditioned on a given piece of data. In the offline RL case, the GAN tries to generate action data conditioned on the state data thus making CGAN a natural choice. I also attempt to use Wasserstein GAN with Gradient Penalty (WGAN-GP). The WGAN uses the Wasserstein distance as the loss function, ideally providing a gradient signal where a standard GAN may fail to. WGAN-GP adds a gradient penalty to the discriminator's loss function to help improve stability. The third GAN modification is the Spectral Normalization GAN (SN-GAN). SN-GAN provides weight normalization on the discriminator to help with the training of the GAN.[7] The CGAN is tested by itself, tested with WGAN-GP, tested with SN-GAN, and tested with WGAN-GP and SN-GAN.

I also run trials with and without BCQ's Actor-Critic architecture. Some trials are run with the actor, critic, generator, and discriminator networks each being trained on their own neural network (BCQ GAN ACGD). In others trials I combine the actor and generator into one network with two separate heads: one produces actions (actor head) and the other generates actions

(generator head). The critic and discriminator are combined with multiple separate heads: the discriminator head and two critic heads. There is a natural analogue between the actor-critic set up and the generator-discriminator set up as in each architecture one neural network produces actions and the other judges them. This architecture is labelled as BCQ GAN GD in the results below.

The third modification I make is to allow BCQ's perturbation model to be replaced by an approach inspired by Random Network Distillation (RND). An RND model is typically used to improve exploration in RL algorithms by being a proxy for how familiar the agent is with a certain state. RND consists of two neural networks. One neural network is fixed and never changes. The second neural network tries to predict the output of the first neural network, given a state variable which both network receives. The second neural network is trained based on the difference between its output and the first neural network's output. The idea being that the trained neural network will be able to better estimate the fixed neural network's output as the agent visits the state more often (and thus the trained neural network is trained more frequently on that state).[8] In this project the RND is used at evaluation time when an agent selects an action. The original BCQ chooses the argmax action, BCQ GAN with the RND option enabled chooses the argmax action given that the action satisfies an RND score for familiarity with that action given that state. RND could also be used to augment the BCQ GAN with action selection during training time but due to this projects already large number of ablations to test, this was not implemented.

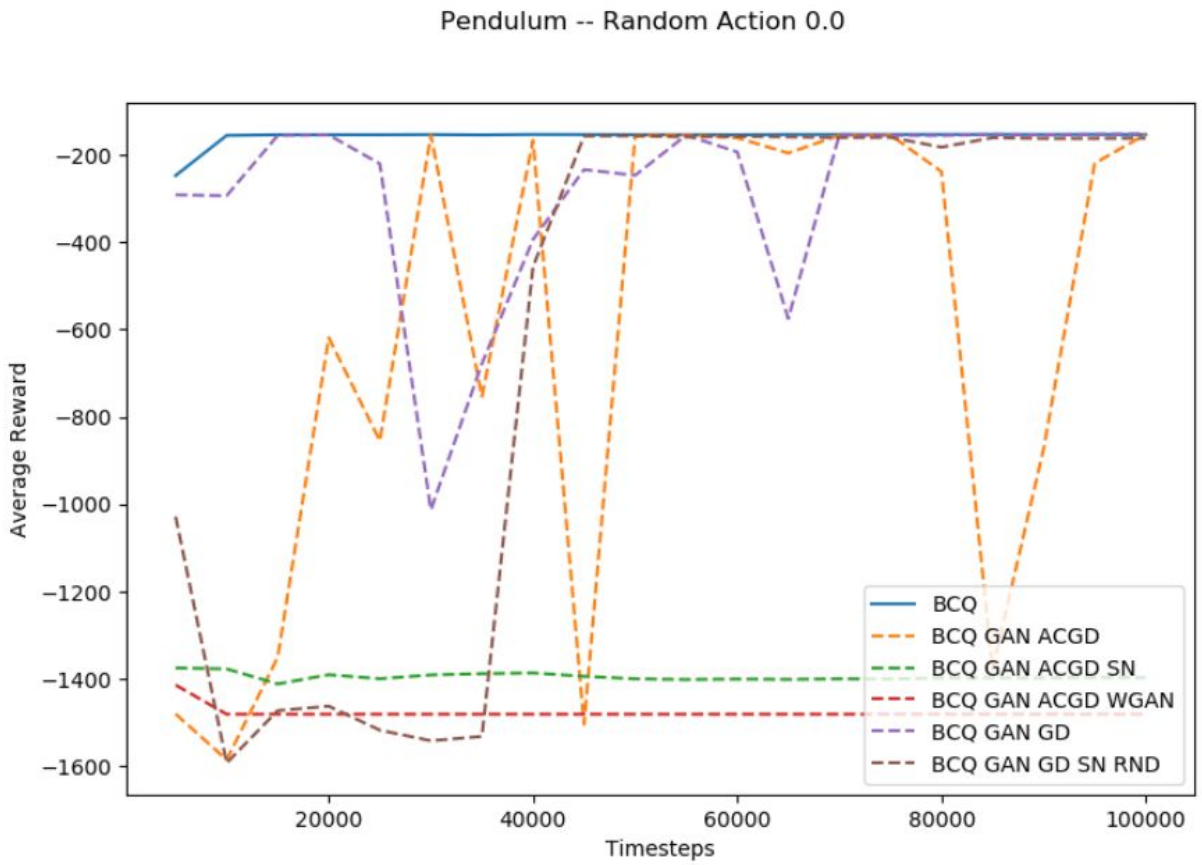
Evaluation of models consists of average reward produced by that agent, evaluated at every 5,000 timesteps of the agent's training process. During evaluation, the agent interacts with the environment and the average reward of the agent is collected. The agent receives no signal from the evaluation and continues to train only on offline data. The agent is trained on the test set data for 100,000 timesteps on Pendulum, 200,000 timesteps on LunarLander, and 300,000 timesteps on BipedalWalker.

In total there 16 different ablations this project seeks to test: with or without separate Actor-Critic networks, with or without WGAN-GP, with or without SN-GAN, and with or without RND action selection. BCQ is evaluated on all three environments and all three collected data

sets for each environment on the collected test data. The default hyperparameters for BCQ are used. BCQ GAN's hyperparameters are partially tuned on the train data for each environment and evaluated on the test data. Due to the large number of ablations (16), the number of environments (3), the number of collected data sets (3), and the number of hyperparameters to tune (dozens); an exhaustive sweep of hyperparameters was unable to be performed. Instead I attempted to tune hyperparameters for various trials, starting with the easiest environment of Pendulum with random action probability of 0.0. For successful ablations, more hyperparameter tuning occurred for more difficult random action probabilities (0.2 and then 0.5) and then more difficult environments (LunarLander and then BipedalWalker). Due to time constraints, unsuccessful ablations are gradually dropped from further hyperparameter tuning and further test runs in more difficult trials. For example, I was unable to get any successful results for WGAN despite dozens of attempts at different hyperparameters. I include WGAN results for the first trial (Pendulum with p=0.0) and then do not include any further ablations including WGAN.

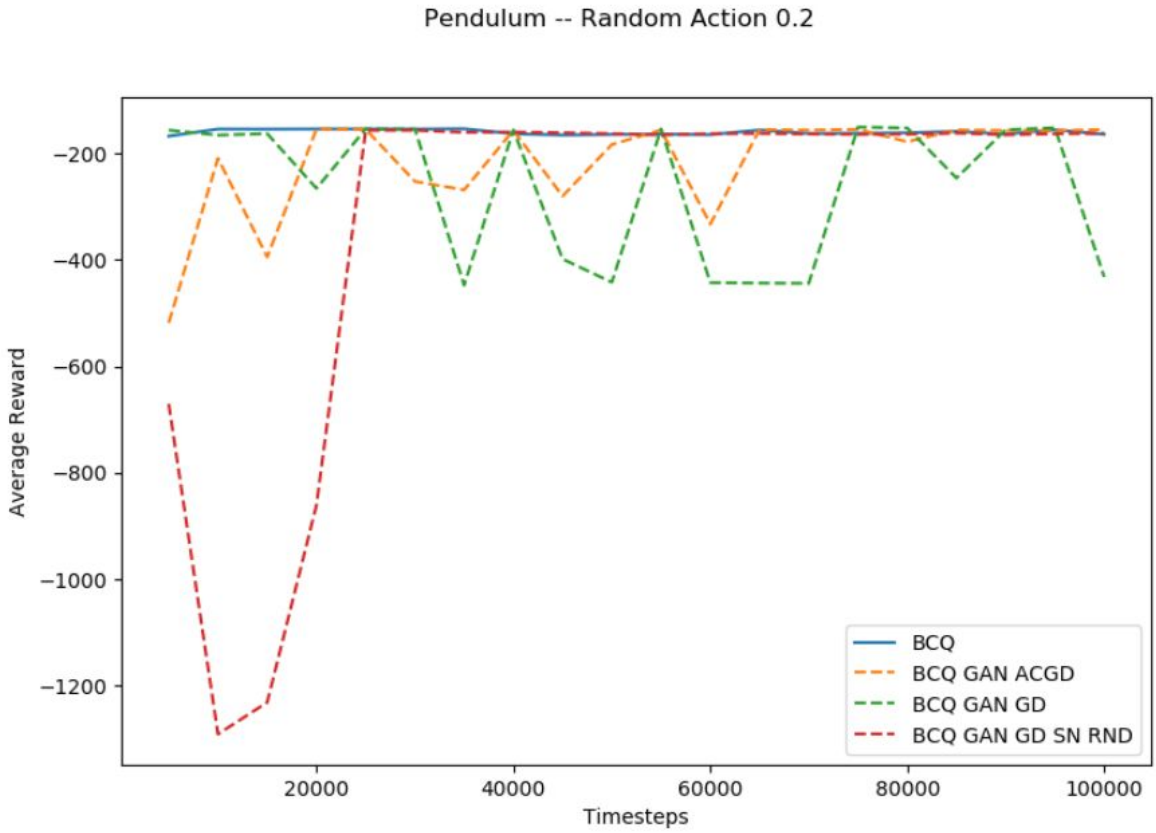
IV. Findings

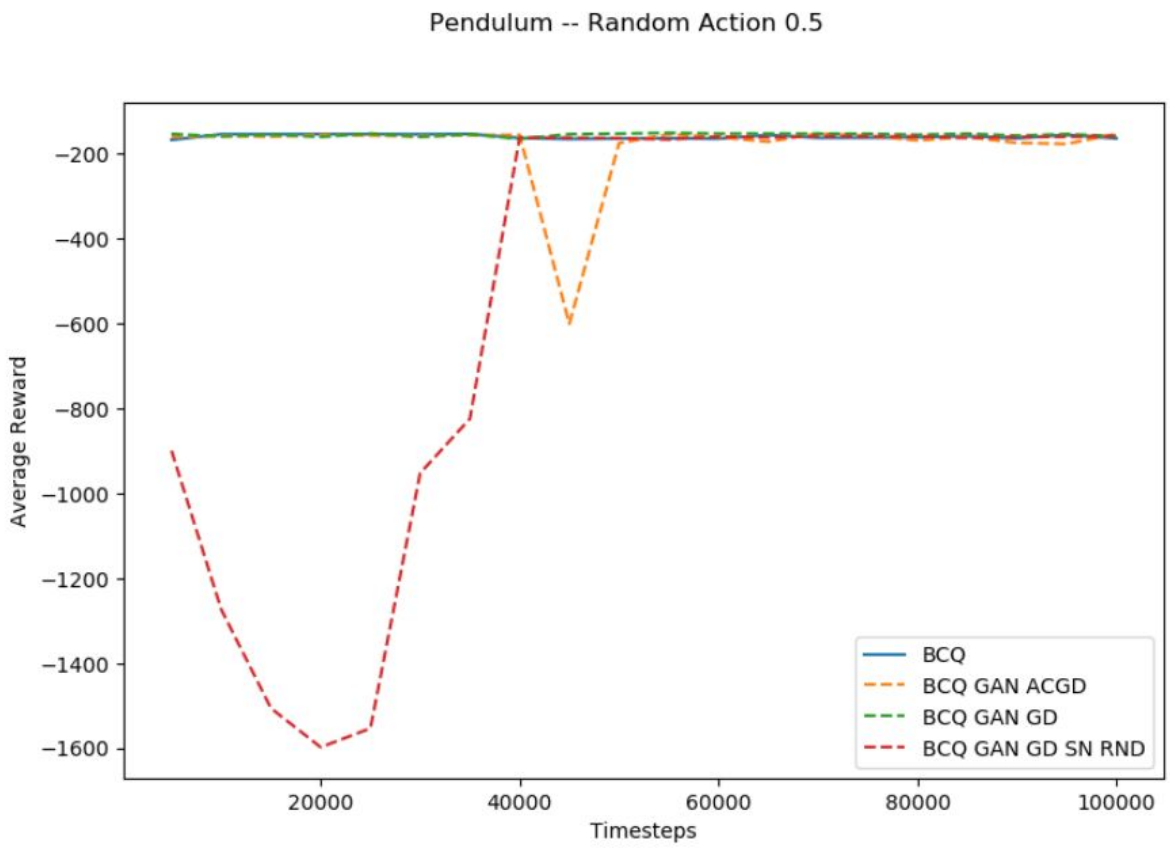
Included in this section are the average reward plots for various ablations of the BCQ GAN agent. The BCQ benchmark was a strong benchmark on all three tasks, performing above the solved level for LunarLander and BipedalWalker (above 200 and 300 average reward) and near optimally for Pendulum. As can be seen from the figure 'Pendulum -- Random Action 0.0' WGAN performs poorly. WGAN was unable to produce any results with any ablations or hyperparameters and was dropped from further charts. BCG GAN with ACGD and SN-GAN also performs poorly and were dropped. However SN-GAN performs well in the GD architecture (which combines the Actor-Critic with the Generator and Discriminator) in all levels of Pendulum and some levels of LunarLander.



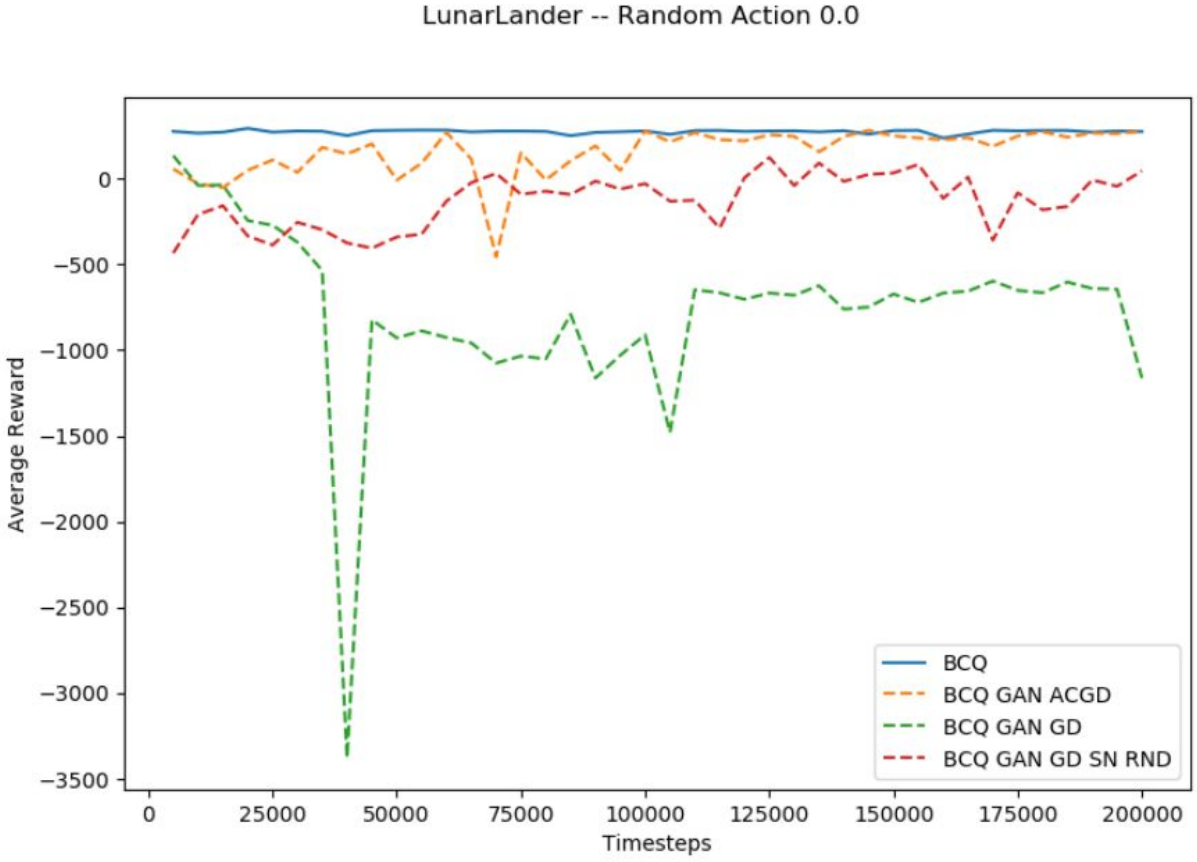
For ‘Pendulum -- Random Action 0.2’ and ‘Pendulum -- Random Action 0.5’, the remaining ablations continue to perform strongly. BCQ GAN with the GD architecture and SN-GAN and RND action selector performs at the level of the BCQ benchmark after the GAN is successfully

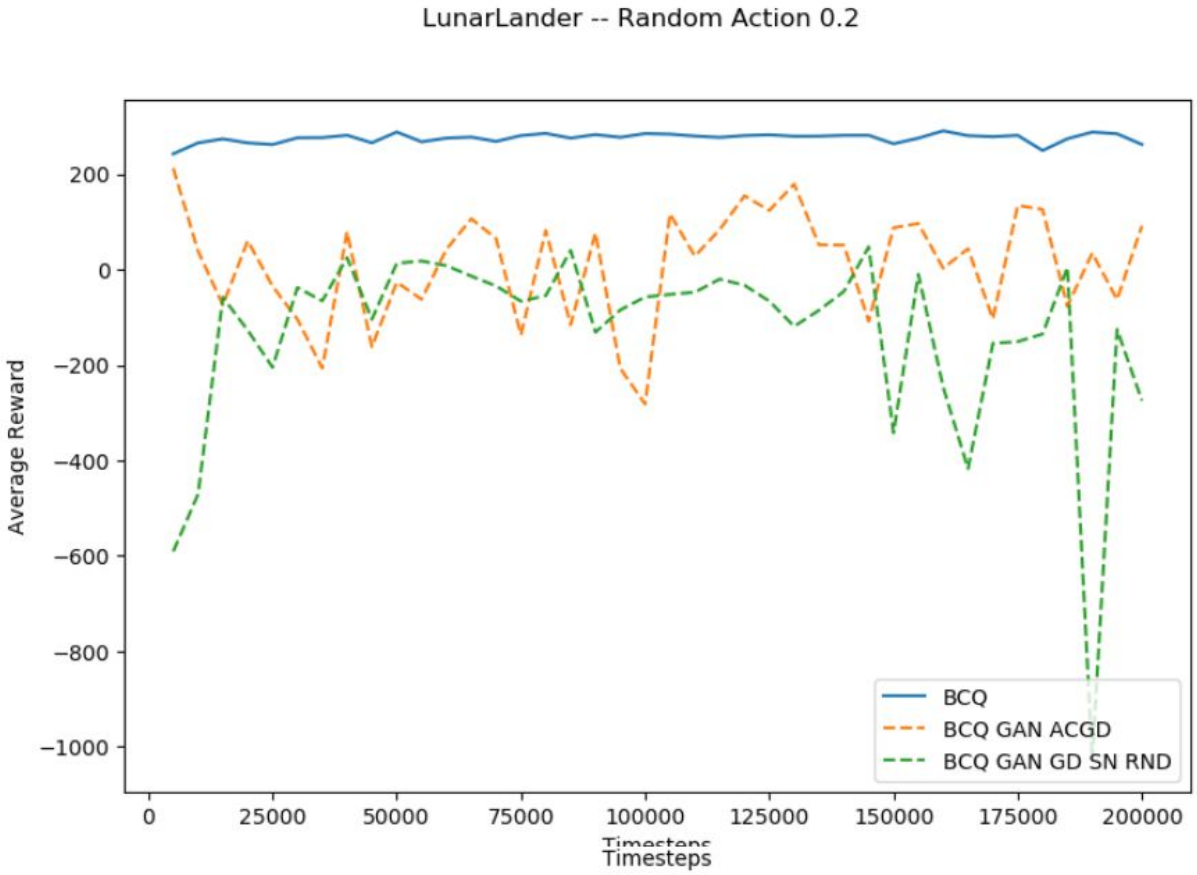
trained.

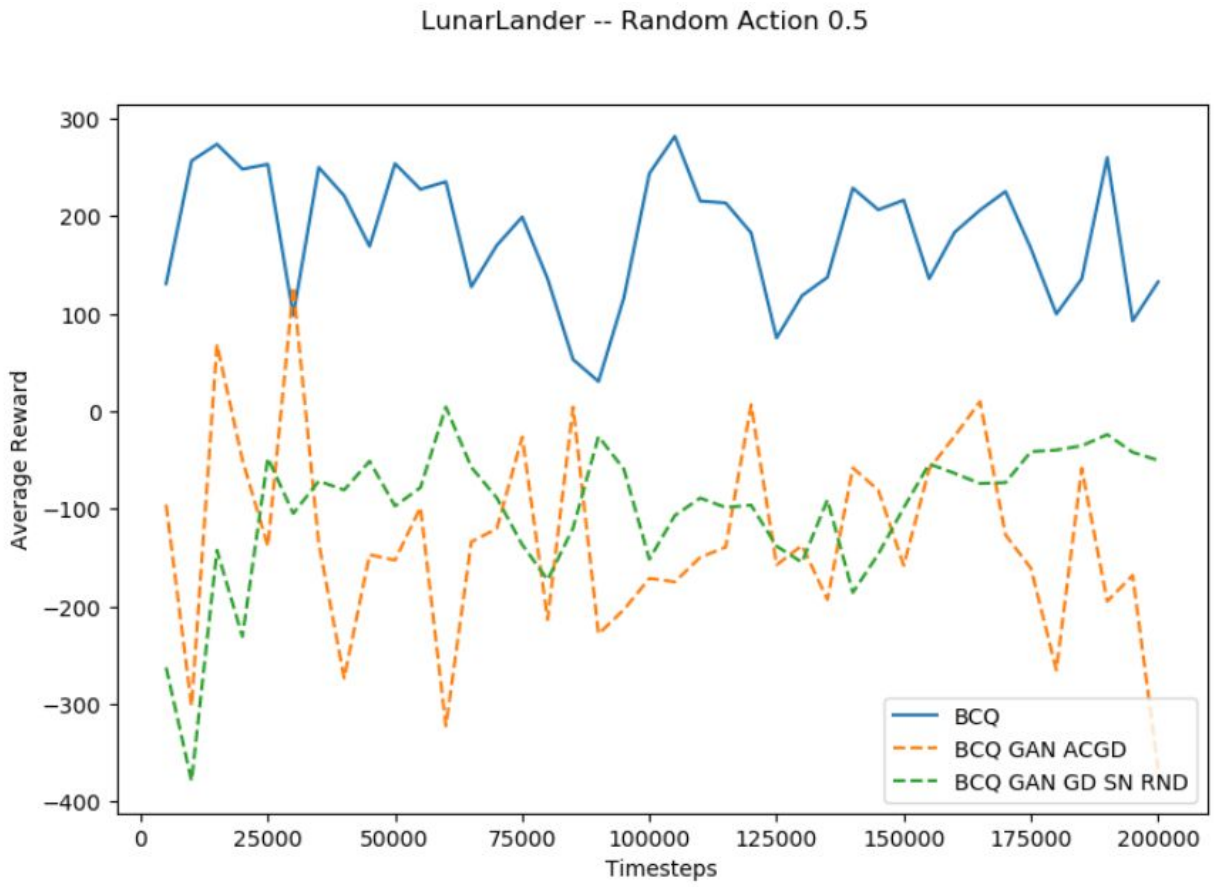

Pendulum -- Random Action 0.2
Average Reward
−200
−400
−600
−800
−1000
−1200
20000
40000
60000
80000
100000
Timesteps
BCQ
BCQ GAN ACGD
BCQ GAN GD
BCQ GAN GD SN RND



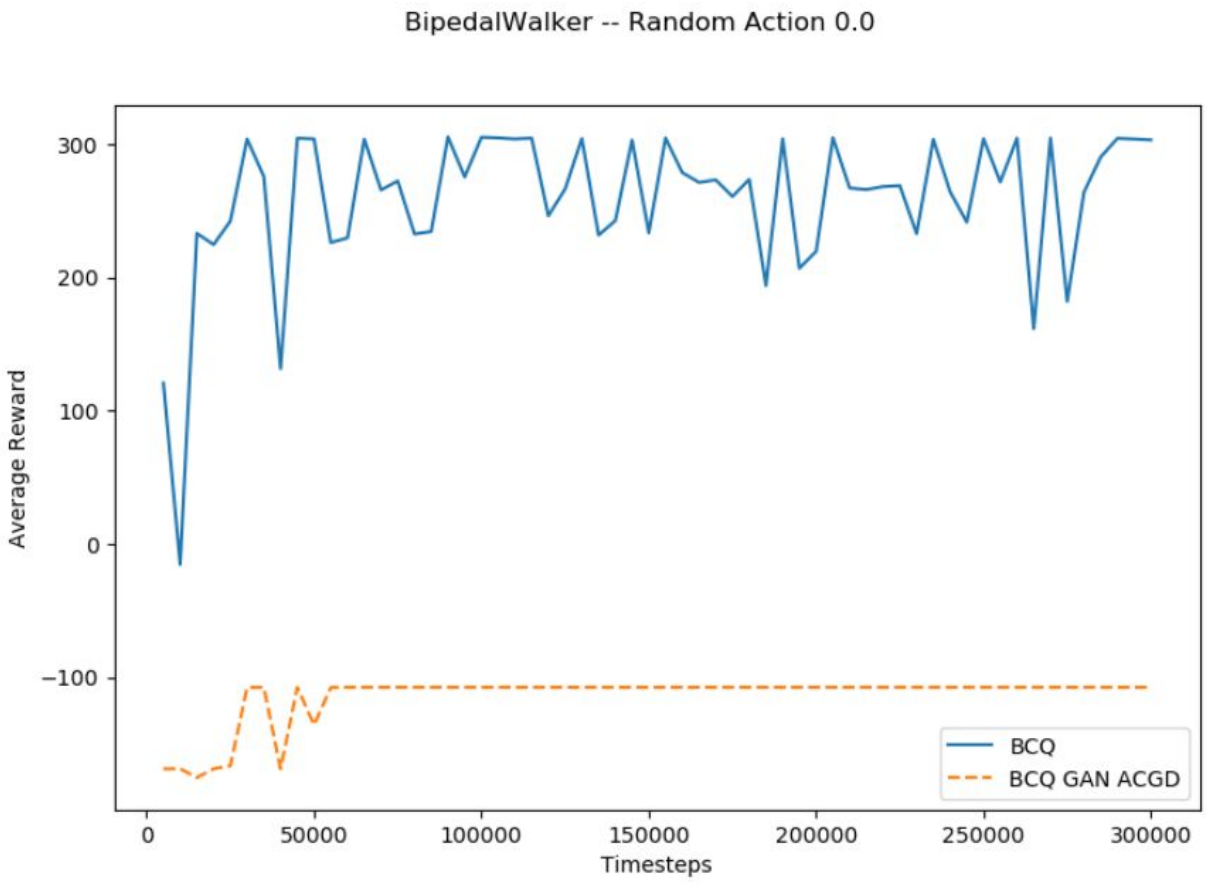
For LunarLander, the BCQ benchmark greatly outperforms all ablations of BCQ GAN. BCQ GAN maintains limited results for 'LunarLander -- Random Action 0.0' but performs worse and worse when the random action probability increases. BCQ benchmark also decreases, as would be expected since the agent is trained on more suboptimal data and has less access to successful action trajectories, but decreases at a much slower rate than BCQ GAN.


LunarLander -- Random Action 0.0
Average Reward
0
−500
−1000
−1500
−2000
−2500
−3000
−3500
0
25000
50000
75000
100000
125000
150000
175000
200000
Timesteps
BCQ
BCQ GAN ACGD
BCQ GAN GD
BCQ GAN GD SN RND


LunarLander -- Random Action 0.2
Average Reward
200
0
−200
−400
−600
−800
−1000
0
25000
50000
75000
100000
125000
150000
175000
200000
Timesteps
BCQ
BCQ GAN ACGD
BCQ GAN GD SN RND



For the BipedalWalker environment, the BCQ benchmark performs well. No tuning of BCQ GAN hyperparameters was able to get any positive results for BipedalWalker. Generally the BCQ GAN gets stuck producing the same defective action over and over again. Some hyperparameter settings were able to produce non defective actions but average reward was similarly poor to the results in 'BipedalWalker -- Random Action 0.0'



V. Conclusions and Future Work

Under the settings of this project's experiments, VAE in the BCQ algorithm is superior to training offline RL agents than a GAN using the BCQ algorithm. While a GAN had limited success in training offline RL agents in Pendulum and LunarLander in some experiment setups it was only achievable after much hyperparameter tuning and many ablations and testing of various types of GANs and neural network architectures. The SN-GAN had limited success when using the GD architecture with RND while the ACGD architecture with no SN-GAN or RND performed similarly.

GANs may hold promise in the field of offline RL. Offline RL is a difficult field and this experiment was limited to simple environments with too many ablations to sufficiently perform full hyperparameter sweeps. For future work I am interested in how a GAN can be fully incorporated into an algorithm built around the GAN. Using image data may also highlight the

strengths of GANs in the offline RL setting as GANs continue to produce outstanding results in the field of image generation.

References:

1. Scott Fujimoto, David Meger, and Doina Precup. “Off-Policy Deep Reinforcement Learning without Exploration.” 2019; https://arxiv.org/abs/1812.02900.
2. Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. “Offline Reinforcement Learning: Tutorial, Review, and Perspective on Open Problems. 2020; https://arxiv.org/abs/2005.01643.
3. Robin Ranjit Singh Chauhan, host. “Scott Fujimoto.” *TalkRL*. 2020; https://www.talkrl.com/episodes/scott-fujimoto.
4. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” arXiv: 1606.01540 [cs.LG], June 2016.
5. Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In International Conference on Learning Representations, 2016.
6. Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. “Datasets for Deep Data-Driven Reinforcement Learning.” 2020; https://sites.google.com/view/d4rl/home.
7. Z. Wang, Q. She, and T. Ward, “Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy,” arXiv: 1906.01529 [cs.LG], June 2019.
8. Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by Random Network Distillation,” arXiv: 1810.12894 [cs.LG], June 2019.