Generalizing and Testing Robustness of Augmentation Techniques in Reinforcement Learning

Anjali Dhabaria
Georgia Tech
adhabaria3@gatech.edu

Rohit Sridhar
Georgia Tech
rsridhar37@gatech.edu

Shashwat Shivam
Georgia Tech
sshivam6@gatech.edu

Abstract

Pixel-based Reinforcement Learning tasks have traditionally been difficult to train using Q-learning. The widely accepted approach involves using CNNs to encode the states in the state-space. One approach, building upon this, proposes using data augmentation techniques to improve the performance of RL algorithms. We expand on their work and test the generalizability using GANs and architecture modifications [1]. *We also test the robustness of the technique through the use of Adversarial observations. Notably, we find that the models trained on data augmentations are somewhat robust to Adversarial attacks, even with high attack probabilities. Our attempts to improve generalizability are mixed. Changing the underlying model to a sequential neural model produces stable, but lengthy training. We also find some data augmentation techniques significantly improve performance during training, while others implemented by us do not improve performance by any quantifiable metric.*

1. Introduction

The currently accepted paradigm for RL involving visual observation is using CNNs providing a spatial inductive bias [8]. A recent paper [7] proposed using data augmentation techniques to improve performance of RL in such scenarios. The paper provides several augmentation techniques, which when applied to the data while training gave a measurable improvement in performance. Alternatively, this can also be used to reduce training time to achieve the same level of performance or improve data efficiency of training. Further, due to augmentation, the learnt model would generalize better to new environments. With simple augmentation techniques, the authors' simple RL model outperforms the current state-of-the-art methods, proving the efficacy of their proposed technique.

One of our goals in this paper is to test robustness of the generalizability claims in the augmentations paper using adversarial observations. We believe this is a good method of testing robustness since data augmentation techniques should make the model more robust to game states that are visually identical to the original game state, but slightly modified. While adversarial methods have been used to stress test Reinforcement Learning models [10] and Data Augmentations have been tested for generalization in the past [5] [9], we haven't seen work that stress tests Reinforcement Learning models trained on Data Augmentations using adversarial observations, so what we attempt here is novel. We also modify the original network architecture by replacing the MLP with a sequential model. We run robustness tests against adversarial observations to see how it compares to the original set of models in the same scenario. We hypothesized that this model would perform better since sequential models naturally map to the sequential nature of game playing, even when data augmentations are to enhance the original model.

Next, we wish to expand the original authors' approach by introducing more augmentations based on image manipulation techniques. This would also check whether all data augmentation techniques can provide a benefit during training or if only a select few do. The benefit of this is twofold: it may help uncover a potential augmentation technique which performs better than the best the authors [7] could find, and may provide insights into the reasons behind why some techniques help in training more than others.

Other than simple data augmentation techniques, which are thematically similar to the ones explored in the original paper, we also use GANs for data augmentation. This has the added advantage of not being constrained to a certain type of augmentation, which is a drawback of the technique presented by the authors. Using the GAN during training is expected to improve generalization far more than any particular data augmentation technique. The drawback, of course, is the need to pre-train the GAN for creating new training data.

Solving and understanding generalizability is an important task. An approach which easily generalizes is applica-

[1] https://github.com/rohit-sridhar/rad/

ble in a wide variety of applications. Further, it may provide a deeper understanding behind generalizability problems outside of reinforcement learning. It is a well known and important problem concerning the field of Machine Learning as a whole. Finally, it can improve training times, as good performance may be attained using fewer iterations. All of these reasons form the motivation behind exploring the generalizability of these models.

2. Background and Setup

Since the focus of this paper was stress-testing the robustness and generalizability of the paper, we started with the implemented base code for the base paper present at https://github.com/MishaLaskin/rad and used PyTorch for the parts we wrote. Due to resource constraints, we train just the Soft Actor Critic model described in the original paper [7] on the Cart pole Swing Up task for each of our experiments. The original actor network consists of pixel inputs of game states being passed into a 4 layer CNN which encodes the states. These are passed into a 2 layer MLP, which outputs the corresponding action. An action in this domain is a continuous value in the interval $[-1.0, 1.0]$. The critic and corresponding Loss functions are described in Section 3 of the original paper [7]. The full configuration is given in the Appendix Section 8.1.

3. Testing Robustness of Data Augmentations with Adversarial Observations

In this section, we observe the impact of Adversarial examples on models trained using Data Augmentations. To run these experiments, we trained Actor Critic Reinforcement Learning models from the RAD repository for $30,000$ training steps and saved the model every $5,000$ training steps starting from the $10,000^{\text{th}}$ step. We train a baseline model and two other models using two data augmentations, namely "flip" and "crop". Both augmentations are described in detail in the Data Augmentations paper that we use as our starting point [7]. "crop" extracts a random patch from the original frame, and expands it to fit the original image size. "flip" randomly flips an image over the vertical axis. Next, we test the trained models on adversarial examples, that are generated by shifting the observation in the direction of the positive gradient of the loss function for the actor. We use the actor loss, since the actor directly outputs the action. Thus, for some learning rate $\alpha > 0$, actor loss function L_a and observation X we compute the actor loss gradient, $\nabla_X L_a$, and then update the observation using the following equation:

$$X = X + \alpha * (\nabla_X L_a / ||\nabla_X L_a||_2) \qquad (1)$$

We iterate the above computation 10 times and fix $\alpha = 0.1$ as these settings produce clean adversarial outputs without requiring too much compute time. The resulting X is our adversarial observation.

Each evaluation step consists of several episodes, with each episode being a full game played to completion. Within each episode, we choose whether or not to alter a given observation using some attack probability. For these experiments, we use the attack probabilities, $0.0, 0.25, 0.50$, and 0.75. We run 15 evaluation steps, with each step consisting of 10 episodes. The reward of each step is computed as the average reward over the episodes in the step. To assess generalizability, we note whether mean rewards and the standard deviations increase or decrease as the attack probability changes. The intuition behind this approach is to note how resilient the models are towards observations that could plausibly be a part of the game state, but are purposely altered using the Loss Function. In an ideal scenario, the rewards reaped by the trained model would not deteriorate significantly until observations start to become visibly distant from plausible game states. An example of a frame stack modified by the algorithm described here is shown in Figure 1. Our results are shown in Table 1, Table 2, and Table 3. The full configuration of the network and GPU is given in the Appendix Section 8.1.

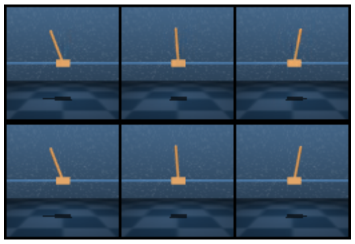

Figure 1: Example of frame stack modified by the Adversarial Algorithm. The top row is the modified frame stack, while the bottom row is the original.

Before discussing the results, we first note that even $30,000$ training steps was not sufficient to entirely train the models. We did not have the compute power or time to train any further. Thus, as expected in the early stages of training an RL algorithm, the rewards at the given training steps are still quite noisy. Regardless, there are some interesting trends in the data. For every data augmentation that was tested, the expected rewards decreased as the attack probability increased. On the other hand, both flip and crop saw smaller decreases relative to the No Augmentation case as the number of training steps increased. In fact, we even see that crop achieves an increase in expected reward when the model is trained to both $25,000$ and $30,000$ steps. Thus while the models seem to generalize better due to data augmentations, there is still a gap in training which the adversarial observations are able to exploit. This could be due to the small number of training steps, but it's more

	Attack Probability			
Model	0.0	0.25	0.50	0.75
10K	427.51 (11.61)	412.90 (17.85)	412.59 (18.44)	407.19 (17.03)
15K	494.56 (11.65)	474.21 (16.25)	459.48 (14.73)	448.86 (19.07)
20K	500.96 (20.94)	492.76 (19.03)	486.63 (21.17)	476.89 (22.42)
25K	523.66 (14.94)	505.37 (21.23)	498.33 (24.86)	472.66 (21.32)
30K	598.82 (38.86)	564.36 (46.60)	535.54 (58.32)	501.99 (45.61)

Table 1: Mean (Std Dev) Rewards per Model (Identified by Number of Training Steps) and Attack Probability with No Data Augmentation

	Attack Probability			
Model	0.0	0.25	0.50	0.75
10K	241.05 (10.81)	230.36 (14.14)	222.97 (20.05)	214.67 (8.42)
15K	253.55 (16.14)	249.50 (19.88)	242.64 (16.85)	239.59 (16.02)
20K	265.74 (12.37)	260.30 (14.86)	254.62 (12.74)	251.07 (11.28)
25K	289.69 (17.05)	272.43 (19.87)	258.02 (17.53)	248.51 (16.12)
30K	278.67 (11.81)	269.93 (13.07)	251.82 (17.30)	243.18 (17.52)

Table 2: Mean (Std Dev) Rewards per Model (Identified by Number of Training Steps) and Attack Probability with "Flip" Data Augmentation

likely that Data Augmentations are insufficient to close the gap entirely on their own. It is unsurprising that crop works best, given that it outperformed the other Data Augmentations in the original paper. Another possible explanation is that adversarial observations for the cart pole environment in general tend to add orange to the empty regions of the observations. The color of the cart pole is also orange; thus the adversarial observations may fundamentally mimic shifting the cart pole. The authors of the original paper note that crop imbues the agent with "translation invariance" [7], which may explain its success on the adversarial observations.

We can also note the increase in standard deviation of rewards across all Data Augmentations tested. This makes intuitive sense, since the use of an attack probability ensures that only some of the observations are altered while others remain intact. On the other hand, flip sees the least change

	Attack Probability			
Model	0.0	0.25	0.50	0.75
10K	274.20 (1.01)	286.27 (12.92)	271.36 (7.58)	278.19 (13.14)
15K	542.62 (21.48)	507.28 (27.67)	433.00 (27.15)	337.49 (25.77)
20K	577.77 (15.17)	553.01 (39.95)	524.50 (57.06)	457.22 (52.93)
25K	389.20 (36.01)	386.35 (38.22)	405.34 (34.22)	398.38 (40.75)
30K	315.52 (16.31)	365.57 (37.89)	363.73 (41.32)	337.29 (28.98)

Table 3: Mean (Std Dev) Rewards per Model (Identified by Number of Training Steps) and Attack Probability with "Crop" Data Augmentation

in standard deviation, while crop sees the most change. While it isn't clear why crop would see the most change in standard deviation, it is possible that flip simply builds more stability into the model generally, since it actually flips the observation (thus inverting the required action). Thus, the more drastic augmentation possibly affords flips more stability than the baseline or crop in the face of adversarial observations as well.

4. Testing an LSTM Architecture with Adversarial Observations

In this section, we test to see if an architecture change to a sequential model is robust to adversarial input. Specifically, we use a Long Short-Term Memory (LSTM) architecture and test it against adversarial observations. The original network in the paper uses a 4 layer CNN encoder and 2 linear layers with ReLU activations [7]. We keep the CNN encoder, but replace the MLP with an LSTM. Thus, we feed our LSTM a sequence of encoded representations of recent game states. We use an LSTM since it has seen success in RL settings in the past [3] [6], and it seems reasonable to hypothesize that LSTMs should be able to encode more information about the game compared to MLPs, since the former can assess sequences of game states whereas the latter can only consider one game state at a time. The expected input (pixels of observations) and outputs (actions) remain the same.

Due to resource constraints, we use just a 1 layer LSTM and use sequences of size 5 when training the LSTM. We tune the learning rate and ultimately find that a learning rate of 1.5×10^{-5} produces stable training. Due to resource constraints, we were not able to train beyond $1,500$ steps, but even so we see interesting results. We evaluate the model every 500 training steps and perform 10 evaluation steps,

	Attack Probability			
Model	0.0	0.25	0.50	0.75
500	68.91 (0.22)	62.21 (0.34)	62.24 (0.27)	62.28 (0.34)
1K	93.51 (0.46)	83.11 (0.25)	83.13 (0.37)	83.31 (0.42)
1.5K	102.63 (0.49)	88.65 (0.63)	88.15 (0.73)	87.51 (0.70)

Table 4: Mean (Std Dev) Rewards per LSTM Model (Identified by Number of Training Steps) and Attack Probability with No Data Augmentation

each consisting of 5 episodes and compute mean rewards over the steps. As in the previous section, we iterate over the following attack probabilities: $0.0, 0.25, 0.5$ and 0.75. Our results are given in Table 4. The full configuration of the network and our hyper parameter selection is given in the Appendix Section 8.2

The rewards are on average lower than in the previous cases, due to the decreased number of training steps. Even so, we can note some facts about the stability of the results. Regardless of whether or not adversarial attacks are employed, the standard deviations are low. While the percentage changes in standard deviation are comparable to the original network, the absolute magnitudes are much smaller. This may be due to the fact that using an LSTM over-parametrizes the model, improving generalization capabilities [4]. It may also be due to the sequential nature of the model. Perhaps due to the simpler nature of the cart pole problem, the sequences can be easily generalized over. It would be interesting to implement a fully trained sequential model with attention heads to see if we can in fact identify specific game states that lead to model decisions and to try it on a variety of different games.

The expected rewards do not decrease much after the initial increase in attack probability (from $0.0 \rightarrow 0.25$). This is an interesting phenomenon, but it is not immediately clear why it is the case. It is possibly related to the low standard deviations in the rewards across trials. In this case, initial attack probability of 0.25 is enough to induce a sufficiently large change in the Expected Reward which does not get impacted much with further increases in attack probability.

5. Testing Generalization and Performance with More Data Augmentations

The original paper introduces several augmentation techniques which can be used to increase training data. Of these, 'crop' is observed to perform best after training for the Cart Pole Swing Up task. Some of the other techniques do not converge to a solution which would result in the pole reaching the upright position after training for 200,000 steps. Instead, a few we tested had an interesting behaviour of shooting off to one end of the frame and then trying to swing the pole upward. The 'crop' method found the correct solution in the least number of time-steps and was able to balance the pole in an upright position, which validates the paper's claim of the reward for 'crop' being the highest.

To test whether crop is indeed the best augmentation technique, we implemented 4 additional augmentations techniques not present in the paper. This would also help determine if the claim that reinforcement learning can be improved with augmentation techniques generalizes to other techniques and not just 'crop'. To this end, the 4 augmentation techniques implemented were:

- Median Blur: Here, a 3x3 window was passed over the padded image and the median value in the window was selected to replace the pixel at the center of the window. This filter results in rounded corners and reduces salt-and-pepper noise.
- RGB Shift: For this technique, we randomly shift the RGB values for each image by a random amount. The shift for each channel is chosen separately but from the same probability distribution.
- Channel Shuffle: This augmentation involves swapping the values for the 3 channels amongst themselves. Here, we replace the values for the red channel with those from the green channel, the ones in green with the values in blue and finally blue with red.
- Random Inversion: First a random subset of images is selected from the batch and for those, the color value of each channel is inverted. To achieve this we simply subtract the value from the maximum value, i.e. 255.

We tested the hypothesis about generalization on the 4 augmentations mentioned above. However, we observed similar behaviour of the cart moving to one side and the pole being unsuccessful in staying vertical. While this is not conclusive proof, it does raise questions on the generalizability of the paper's claim of augmentations improving RL performance. This could also be a result of the reduced training steps we performed the experiments with, due to resource constraints. Some sample results showing the reward at 200,000 steps for RGB shift, Random inverse, flip and crop is tabulated below. We do not observe good reward values for the RGB shift and Random inverse techniques, which means that RL does not always benefit from augmentation.

The above behaviour could be attributed to one of two reasons - necessity of more training or some augmentations being inherently better at improving performance with RL. For instance, 'crop' would let the Neural Network learn

	Final Reward	
Model	Mean	Best
RGB Shift	177.54	180.63
Random Inverse	108.62	110.18
Flip	625.84	838.10
Crop	877.66	878.56

Table 5: Mean and best Rewards for different augmentation techniques. Crop has best mean and best rewards after full training of 200,000 steps.

from the relevant parts of the image. This would definitely improve performance. On the other hand, some augmentation techniques like RGB shift may not help with generalization at all.

6. Data Augmentation with GANs

Image data augmentation techniques are based on image manipulation or deep learning approaches. In the original paper as well as in Section 5 many data augmentation techniques were analysed which either perform pixel-level transforms or spatial-level transforms on input training images. In this section we explore deep learning based approaches (typically GANs) to augment the observations of an agent to learn a better policy. In recent years, Generative Adversarial Networks (GANs) have proved to be quite successful for data augmentation. Examples include CycleGAN for data augmentation on medical images based on image to image translation [12], DAGAN to generate synthetic images based on image conditioned GAN [1] etc. In this work we generate synthetic observations for the agent using noise and then use these generated observations to augment the data.

6.1. Objective

Similar to the traditional GAN model, the objective of discriminators is to discriminate between the generated images and the real images. We used the Wasserstein loss (W-Loss) format [2] and not the BCE loss because W-Loss does not suffer through vanishing gradient problem. This mitigates against the mode collapse issue, resulting in better performance. When training GANs using this loss, the critic needs to be 1-Lipschitz Continuous because it assures that the W-Loss function is continuous and differentiable i.e makes the loss function valid in approximating the underlying Earth Movers Distance Also, this condition doesn't allow the loss function to grow too much and thus maintains some stability during training. For this we used a gradient penalty by adding a regularization term to the loss function which penalizes the critic when the gradient norm is greater than 1. Thus, the overall objective becomes -

$$\min_g \max_c [\mathbb{E}(c(x)) - \mathbb{E}(c(g(z)))] + \lambda\,\mathbb{E}(\|\nabla c(\hat{x})\|_2 - 1)^2 \tag{2}$$

where g is generator, c is critic, z is the input noise vector, $\hat{x}$ is the interpolated image between generated and real images with ϵ and λ is a hyperparameter to weigh regularization.

Hyperparameter	Value
Augmentation	GAN - cart pole
Observation rendering	(100, 100)
Replay buffer size	100000
Initial steps	1000
Stacked frames	1
Action repeat	8
Batch Size	128
Evaluation episodes	10

Table 6: Hyperparameters for training the agent. Other parameters not in the table are taken same as in the original paper

Hyperparameter	Value
n_epochs	400
z_dim	64
batch_size	4
lr	0.0002
beta_1	0.5
beta_2	0.999
c_lambda	10
crit_repeats	5

Table 7: Hyperparameters for training GAN. z_dim: the dimension of the noise vector, lr: learning rate, beta_1, beta_2: momentum terms, c_lambda: weights of the gradient penalty, crit_repeats: number of times to update the critic per generator update

6.2. Network Configuration

The generator and critic architectures are drawn from DCGAN [11]. We used convolutions without any pooling layers, batchnorm in both the generator and the critic, no fully connected hidden layers, ReLU activation in the generator for all layers except for the output, which uses a Tanh activation and LeakyReLU activation in the critic for all layers except for the output, which does not use any activation. The Generator is built using 6 layers (5 hidden layers and 1 output layer) and the critic is built using 5 layers (4 hidden and 1 output layer)

6.3. Training procedure

To build the training set we first played an episode and collected 10,000 observations. We then trained the GAN following the procedure as in [2]. We trained the critic crit_repeats steps, then one step on generator. At each iteration, the generator is not trained until the critic has been trained for crit_repeats steps. This enables the critic to provide more reliable gradient information. We employed Adam for gradient descent optimization.

We first used a batch size of 128 and 64 but it resulted in learning more than one location of agent in the synthetic observations. We then resorted to much lower batch size of 4 to ensure no noticeable differences in the observations in a batch.

We generated a stack of three images as a observation similar to the original paper. Since, the GAN model wasn't trained to generate three temporally related images, we did not observe good rewards. So, we decided to generate a single image observation for the rest of our experiment.

After we trained the GAN, we fixed the generator and started training the agent. For every batch of observation, we generated a batch of noise vectors and passed through the generator model which in turn produced synthetic observations to be added to the replay buffer. The training of the agent is followed as in the original paper for 200K steps. See table Table 6 and Table 7 for the hyperparameter values used.

6.4. Evaluation

There are two reasons that we think explain the observed behavior. First, it could be due to the fact that the initial observations collected to train the GAN network are imbalanced. Most of the images are of the cart pole balancing an upright position and there are few images of the cart pole in any other position; thus the GAN network learned to produce one cart position more often resulting in augmentation of one state more often than the rest. We note that due this imbalance, the agent is unable to learn well.

Secondly, this could be due to the fact that the observations are generated from random noise rather than conditioned on the input image. We believe that conditioning on input images would have produced more close to real synthetic observations.

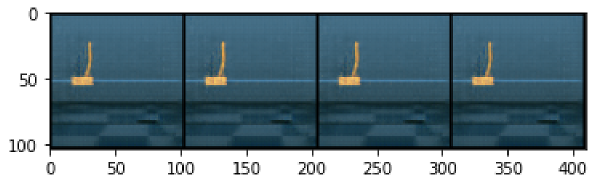

Figure 2: Example of a batch of generated images by the trained GAN model

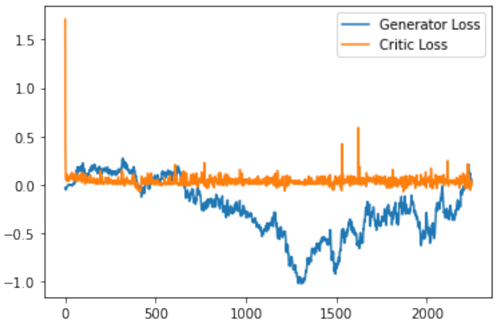

Figure 3: Generator and Critic Loss vs steps: Generator mean loss: -0.184, critic mean loss: 0.019

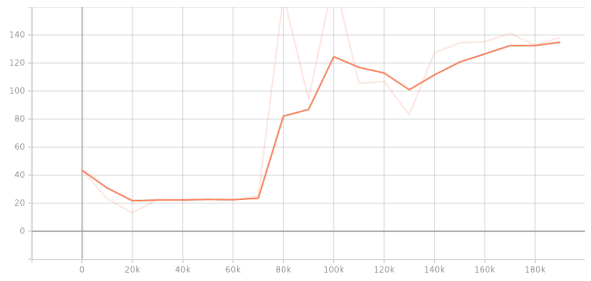

Figure 4: Maximum Episode Reward vs steps

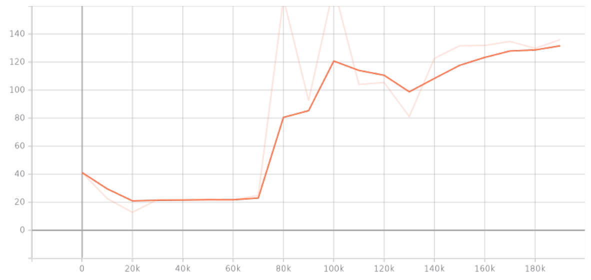

Figure 5: Mean Episode Reward vs steps

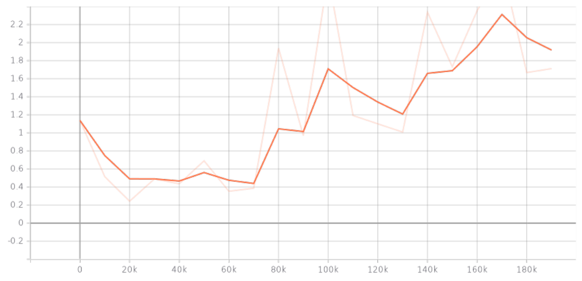

Figure 6: Standard Deviation of Episode Rewards vs steps

7. Concluding Remarks

We first tested robustness of the original paper's model using adversarial observations. Though we were limited in training time due to resource constraints, our results were more or less in line with what we expected. Both "crop" and "flip" were somewhat more resilient to adversarial observations than the "No Data Augmentation" case. Next we tinkered with the architecture and used a sequential model (LSTM, specifically), instead of the original MLP. We found that it stabilized the model, even in the face of adversarial observations. We also implemented additional data augmentations, but did not find the significant improvements we expected. These results suggest that data augmentations may not generally improve performance for RL models, but rather that specific augmentations may help for some specific problems. Lastly, we built a GAN model as another means of tweaking the training procedure. We note that the model trained on observations generated by the GAN, in addition to standard observations, did not perform as well as we expected. This was likely due to the fact that input images were not sufficiently varied and the fact that that generated observations are conditioned on random noise, rather than on proper input.

Thus, we both tested robustness of the existing data augmentations and tried to improve upon it through various means (architecture changes, more augmentations and generator models). While we came upon interesting results, much of it was also limited by our resource constraints and the large amount of training time required for Reinforcement Learning models using pixel based inputs.

7.1. Future Work

We think there is potential for more work in every area we looked at. We'd like to train models for several thousand more time steps and then test on adversarial examples. We'd like to train across a wider range of data augmentations, environments and tasks in order to better understand and contextualize the results of Section 3. It would also be fruitful to expand the methods of generating adversarial examples. We would also like to try training our LSTM using different data augmentations (we just looked at the "No Data Augmentation" case in this project), and see if we can reap the benefits of both. Finally, it would be interesting to run our sequential models using Attention modules to see if specific game states, generally lead to better rewards and how (if at all) different data augmentations can help here.

For GANs, the current model is limited by the type of the environment. For this we would like to design and train a generative model conditioned on input images that learns to take any data(consisting of observations across all the environments as classes) item and generalize to novel unseen classes of data. Also, in the original paper consecutive frames are stacked together to infer temporal information. It would be interesting to see how well the agent performs if we design a generative model such that it generates observation as a stack of synthetic images related by temporal information learned during the training.

References

[1] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017. 5

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. 5, 6

[3] B. Bakker. Reinforcement learning by backpropagation through an lstm model/critic. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 127–134, 2007. 3

[4] Alon Brutzkus and Amir Globerson. Over-parameterization improves generalization in the XOR detection problem, 2019. 4

[5] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289, Long Beach, California, USA, 09–15 Jun 2019. PMLR. 1

[6] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017. 3

[7] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data, 2020. 1, 2, 3, 8

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[9] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning, 2020. 1

[10] Björn Lütjens, Michael Everett, and Jonathan P. How. Certified adversarial robustness for deep reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1328–1337. PMLR, 30 Oct–01 Nov 2020. 1

Encoder Attribute	Attribute Value
Layers	4
Slide (first layer)	2
Slide (other layers)	1
Filters	32
MLP Feature Dim	50
Output Dim	50

Table 8: Encoder Settings

MLP Attribute	Attribute Value
Input Dim	50
Hidden Layers	2
Activation (both layers)	ReLU
Hidden Dim (both layers)	1024
Output Dim	2

Table 9: MLP Settings

[11] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 5

[12] Z. Xu, C. Qi, and G. Xu. Semi-supervised attention-guided cyclegan for data augmentation on medical images. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 563–568, 2019. 5

8. Appendix

8.1. Adversarial Observation Test Setup

The details of the implementation are given here. The original models were trained using default settings from the RAD paper [7]. The specific settings are given in Table 8 and Table 9.

The training took place for $30,000$ steps and took approximately 6 hours per model. We ran our training procedures on the Google Cloud server (Debian GNU/Linux). We used precisely 1 GPU, the Tesla V100. We fixed the learning rate for both the actor and critic at 1×10^{-3}.

When running the adversarial observation test, we ran it per the specifications in the main paper, for each attack probability (0.0, 0.25, 0.50, and 0.75). The approximate time taken to run the evaluation for each attack probability is given in Table 10.

8.2. LSTM Test Setup

We trained an LSTM on the same Google Cloud server described in the previous section. The LSTM replaces the MLP that takes in the output from the encoder in the previous section. The LSTM settings are given in Table 11. We attempted to test it with learning rates, in the set $\{10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001\}$. We found that 0.00001 produced the most stable training. Additionally, we experimented with 2 layer LSTMs and found that training was much too slow with dropout regularization. Finally, we tried look back sizes between 5 and 10, as well as 128 (a maximum length sequence in this domain and task). We found that 5 produced training that was quick enough and stable enough. We also experimented with the batch size, varying it within the set: $\{2, 5, 8, 10, 128\}$. Lower batch sizes produced unstable training, hence we decided that the increased training time was worth the increased stability. Due to increased resources required for training, we only ran this model for 1500 steps.

When testing the LSTM, we followed the testing procedure described in the main section. We used the same attack probabilities as in the previous section. The approximate time needed was much larger and is given in Table 12.

Attack Probability	Time Taken
0.0	3 minutes
0.25	5 minutes
0.50	15 minutes
0.75	30 minutes

Table 10: Time Taken for Adversarial Obs Tests

LSTM Attribute	Attribute Value
Num Layers	1
Hidden Dim	1024
Max Sequence Size	5
Input Dim (per element of seq.)	50
Output Dim	2

Table 11: LSTM Settings

Attack Probability	Time Taken
0.0	5 minutes
0.25	20 minutes
0.50	40 minutes
0.75	90 minutes

Table 12: Time Taken for LSTM Adversarial Obs Tests