# Analyzing Stability and Properties of Generative Adversarial Networks

**Harsh Bandhey**
Department of ECE
Duke University
`hb170@duke.edu`

**Adrian Lopez**
Department of ECE
Duke University
`adrian.lopez@duke.edu`

**Joshua Rabinowitz**
Department of ECE
Duke University
`joshua.rabinwoitz@duke.edu`

## Abstract

We explore and analyze the stability, properties, and problems associated with different Generative Adversarial Models. We focus our attention on three particular types of models: an auxiliary classifier GAN (AC-GAN), a Deep Convolutional Generative Adversarial Network (DCGAN), and the Wasserstein GAN (WGAN). We trained these models on the CIFAR-10 dataset. We sought to train these models to the degree that we could qualitatively observe that the generator of each GAN model was producing realistic samples for their class. After the sample quality was assured, we sought to perform latent space arithmetic on the CIFAR-10 dataset. Lastly, we wished to demonstrate the instability of traditional GAN models and how WGAN can remedy this instability. The method we use to demonstrate the stability of WGANs is via analysis of the model's gradients and loss curves.

## 1 Background

A Generative Adversarial Network[1] is implemented by training neural networks and having them compete against one another. The first network is a generative network that, from random initialization, outputs generated images. The second model is a discriminator model which predicts if the images provided by the generator are real or fake. When the discriminator model guesses correctly we leave the discriminator model parameters undisturbed but update the parameters of the generative model. If the discriminator guesses incorrectly then we update the discriminator model parameters and leave the generative model parameters undisturbed.

This can be defined as an optimization problem with a discriminator network $D$ and a generator network $G$ to solve the adversarial min-max problem:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{train}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_G}[\log(1-D(G(z)))] \tag{1}$$

This process is looped through in an adversarial game with the discriminator and generative model competing against each other and ideally lasts until the generative model can successfully output fake images that the discriminator only has an approximately $50\%$ chance of getting correctly. Thus, the discriminator is only randomly guessing at which images are real or fake. GANs are notoriously difficult to train and have problems such as mode collapse, when a generator defaults to outputting only a subset of the classes of the initial data set, training instability, and vanishing gradients. As such, we decided to explore different solutions to this problem.

ECE 661 (2021 Fall) final report.

## 2 Method

We trained different types of GANs on the CIFAR-10 dataset. The CIFAR data set contains 60000 32x32 images with 10 classes. We then compared the stability and the results of each of these GANS. We used a DCGAN as a standard GAN model for a point of comparison. We then compared the results of this DCGAN to a WGAN and an ACGAN in order to compare how and if those GAN models improved training and image generation.

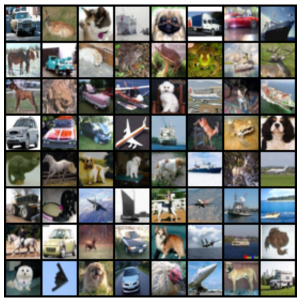


Figure 1: CIFAR-10 Data Set

### 2.1 DC-GAN

We followed the proposed discriminator, generator and hyper-parameters structure from Radford et. al.[2] for optimal performance, with some small modifications in the Convolution and the Transposed Convolution transpose layers in order to make it compatible with 32X32 images [2].

The Discriminator had a loss function composed of an average between the binary cross entropy loss from a batch of real images with values of 1s and images generated from the generator with values of 0. Note this loss function was used in Radford et. al., and is a very similar distance metric to the Jansen-Shannon distance [2]. The Generator had a loss function composed of the binary cross entropy between its output, and a vector of all ones when passed to the discriminator. Reference sections B.1 and B.2 of the appendix for a table of the generator and discriminator structures[1].

### 2.2 WGAN

The WGAN uses the same basic model as the DC-GAN. The only difference between the WGAN and DC-GAN model is that there is no Sigmoid function attached to the last convolution layer of the discriminator when using the WGAN algorithm. The primary differences are the WGAN's novel loss utilizing the Wasserstein Distance and slight modifications to the DC-GAN algorithm[2]. We followed the algorithm for training WGAN original proposed by Arjovsky et al. [3]. Specifically we trained for 100 epochs, using a batch size of 64, a learning rate of 5e-4 and used the RMS Prop optimizer instead of the ADAM optimizer. The full model structure and hyperparameters for the WGAN are in section B.3 and B.4 of the appendix. Unlike the DC-GAN where we only train the discriminator once per batch, in the WGAN we train the discriminator (called the critic in the original paper but we shall call it the discriminator for simplicity) five times on the same batch with the generator outputting a new set of images every instance. These generated images and images from the data set are then fed to the discriminator in order to compute loss and perform backpropagation. The generator is only trained once per batch following the procedure from Arjovsky et al. [3]: The algorithm proposed by Arjovsky et al. is listed below:

[1]We loosely reference eriklindernoren/PyTorch-GAN/ for our project

[2]Our Pytorch implementation of the WGAN algorithm largely referenced aladdinpersson/Machine-Learning-Collection/ML/Pytorch/GANs/3. WGAN/

**Algorithm 1** WGAN Algorithm Proposed by Arjovsky et al.

```
1:  while $\theta$ has not converged do
2:      for $t = 0, ..., n_{critic}$ do
3:          Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data
4:          Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples
5:          $g_w \leftarrow \nabla_w \left[\frac{1}{m}\Sigma_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\Sigma_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
6:          $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
7:          $w \leftarrow clip(w, -c, c)$
8:      end for
9:      Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\Sigma_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

The other difference between the WGAN and the DC-GAN is a novel loss computation involving the Wasserstein Distance. The Wasserstein Distance provides more informative gradients than the Jensen-Shannon (JS) Divergence, which is mathematically similar to the loss function used to train the DC-GAN [3]. The Wasserstein distance is a more direct way of representing the discriminator's goal: separating the probability distribution of the images produced from the generator from the probability distribution of naturally occurring images. As the discriminator seeks to maximize the Wasserstein difference, the generator seeks to minimize the Wasserstein distance. A constraint on using the Wasserstein distance is that the function that maximizes this distance must lie in a compact space $\mathcal{W}$ and be Lipschitz. We ensure this condition by clipping all the weights in the discriminator for each instance of back-propagation.

$$\max_{||D||_l \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[D(x)] \tag{2}$$

Discriminator Use of Wasserstein Difference

## 2.3 ACGAN

We also implement an ACGAN[3] which is a variant of GANs employing label conditioning for generating classes of images with global coherence and better stability. The structure of the model implemented and hyperparameters is followed from the proposed model referenced in [4] and is and ny differences are outlined below Section B.5 and B.6 in the appendix. The model is trained for a 100 epochs. In the ACGAN, every generated sample has a corresponding class label, $cp_c$ in addition to the random noise $z$. The generator uses both the class label and the random noise to generate fake images images. The discriminator gives both a probability distribution over sources and a probability distribution over the class labels. Thus, the discriminator outputs two predictions with the first being whether the image is real or fake and the second which class it seems to belong to.

$$L_S = \mathbb{E}[log(\mathbb{P}(S = real|X_{real}))] + \mathbb{E}[log(\mathbb{P}(S = fake|X_{fake}))] \tag{3}$$

$$L_C = \mathbb{E}[log(\mathbb{P}(C = real|X_{real}))] + \mathbb{E}[log(\mathbb{P}(C = fake|X_{fake}))] \tag{4}$$

Loss functions for AC-GAN

The objective function has two parts: the log-likelihood of the correct source, $L_S$, and the log-likelihood of the correct class, $L_C$. Discriminator is trained to maximize $L_S + L_C$ while Generator is trained to maximize $L_C - L_S$. AC-GANs learn a representation for that is independent of class label which we show through vector manipulation.

[3]Our training loop implementation and several functions for ACGAN came from clvrai/ACGAN-PyTorch/

## 3 Experiment results

Below we discuss the results of the various GAN models. All of the hyper-parameters for the models are taken from the literature and were not changed during training. The model structures and hyperparamters are listed in the appendix.

### 3.1 DC-GAN

Below is the output for the DC-GAN images. The DC-GAN was trained for 100 epochs implementing the models in Appendix B.1 and the hyper-parameters in Appendix B.2. From the results below we can see that the DC-GAN is beginning to output realistic images though more training is needed in order to produce images that could fool a human. We did not train the DC-GAN for more epochs as we wished to compare the DC-GAN and WGAN for the same numbtr of epochs.

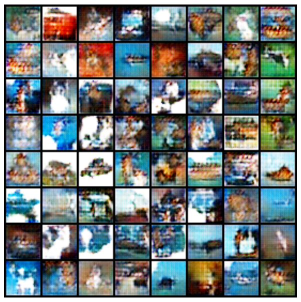


Figure 2: Samples from the DC-GAN Generator at Epoch 100

### 3.2 WGAN

Below is the output for the WGAN images. The WGAN was trained for 100 epochs implementing the models in Appendix B.3 and the hyperparameters in Appendix B.4. We see that the model doesn't give very clear images, took more time to train, and did not converge by a 100 epochs. As the WGAN is computationally expensive to train due to the inner loop of the discriminator and the overall algorithm we were unable to train for more epochs.

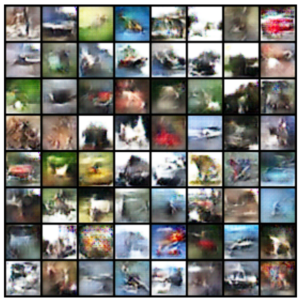
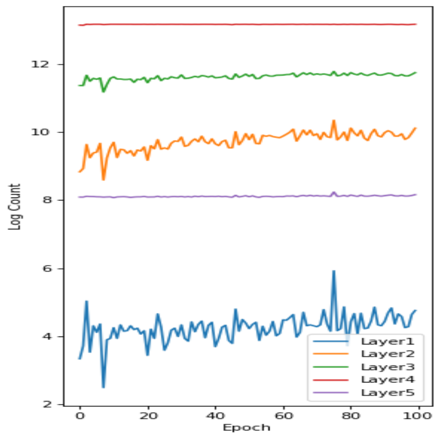


Figure 3: Samples from the WGAN Generator at Epoch 100

#### 3.2.1 Vanishing Gradients

From the below plots we can see that in the DC-GAN (left) an increasing number of the gradients in the model trend to 0 as the training progresses while in WGAN (right) the number of zero gradients stays about constant from initialization. This provides clear evidence that the WGAN algorithm fights the vanishing gradient problem.



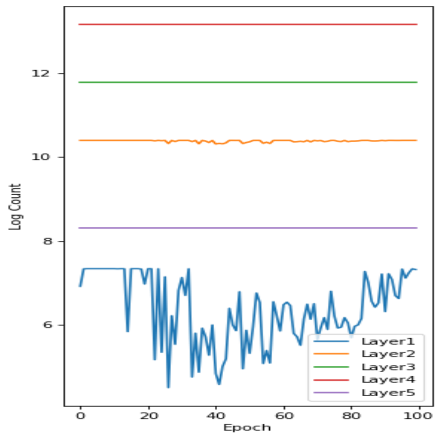


Figure 4: Log Count of Gradients below 0.01

### 3.3 AC-GAN

Below is the output for the AC-GAN images. The AC-GAN was trained for 100 epochs implementing the models in Appendix B.5 and the hyper-parameters in Appendix B.6. From the results below we can see AG-GAN started giving meaning full images from the output results, as well from loss curves in section C.2 of the appendix.

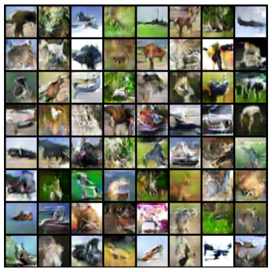


Figure 5: Samples from the AC-GAN Generator at Epoch 100

### 3.4 Vector Space Manipulation

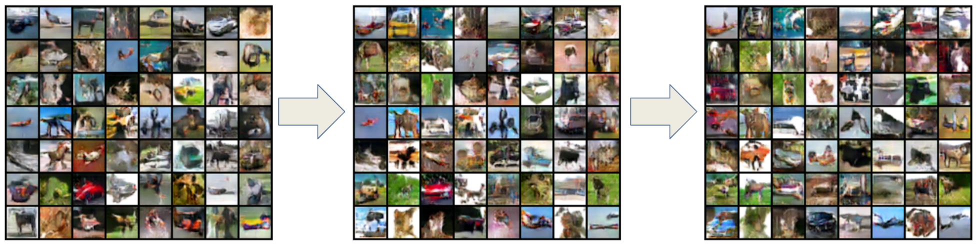


Figure 6: DC-GAN Vector Space Interpolation

We take a Random Isotropic Gaussian ($\mu = 0, \sigma = 0.02$) $= R_1$ and another Random Isotropic Gaussian ($\mu = 0, \sigma = 0.02$) $= R_2$. We then generate 21 images using the discriminator with noise $Zn = \lambda R_1 + (1 - \lambda) R_2$ with 20 $\lambda$ linearly spaced between 0 to 1. Figure 7 shows the image generated with $Z_0$ on the left the figure generated with $Z_{20}$ on the right and the centre shows the middle image generated with $Z_{11}$. The objects in $R_1$ slowly change into $R_2$ which can be more clearly seen over the 20 images, we show the middle interpolation between the transition in the center in this figure [4]

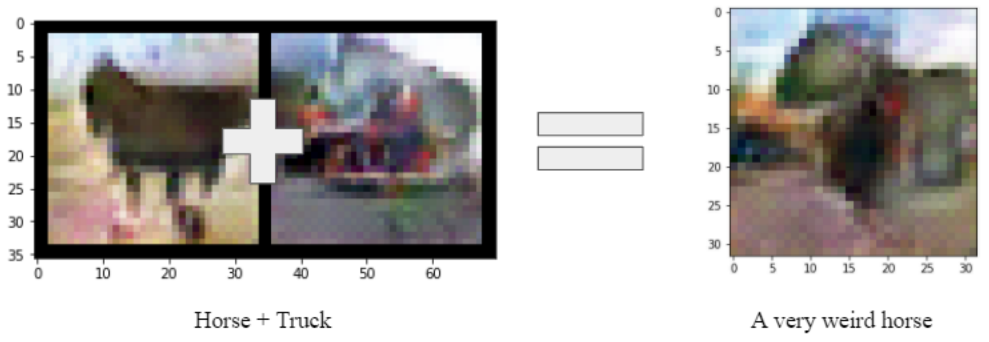


Figure 7: AC-GAN Vector Space Interpolation

We also generate two images form two random noises and then generate a third image from a linear combination of the first two noises. The generated images is not a great representation for the class it was classified for by the AC-GAN. This shows AC-GAN is not dependent on the label for the image generation but more heavily on the latent space representation.

Moreover these are clear observations of how latent space representation affects the generation of the output and how they can be manipulated. More importantly it lays the intuition that how GANs can be used for domain/style transfer and generation of new images.

## 4 Conclusions

Our results in Figure 4 show that the WGAN was more resilient to the vanishing gradient problem than the DC-GAN. As clearly shown, the earlier layers, specifically layers 1-3 in the discriminator of the DC-GANs, had an increasing number of gradients going to zero as training progressed. Both models were trained for the same number of epochs, have similar model structures, and thus this can largely be attributed to the loss function and training procedure. Another issue with GAN training is the oscillatory and unpredictable nature of the loss for both the discriminator and the generator. In section C.1 of the appendix, one can easily see that the WGAN makes the loss much smoother for the discriminator when comparing it to that of the DC-GAN. Theoretically, this is because the loss generated from the discriminator in the DC-GAN gives less information to the generator than the loss function in the WGAN [3]. It is important to note that the WGAN does take longer to train and the weights in the discriminator need to be clipped to enforce Lipschitz constraint. Thus, while it is more stable and sidesteps many of the problems facing other GAN models, the WGAN is very computationally expensive to train and we were unable to bring it to convergence, suggesting that the results could have been even better! The AC-GAN also greatly improved performance in comparison to the baseline DC-GAN as one can easily see from image quality. In the AC-GAN, the generator uses conditional generation with labels as inputs, and the discriminator predicts a particular image's class label instead of receiving it as an input. This additional input caused the training process to steadily progress and provided some form of stabilization. From the results of our different GANs we can see their training process is highly complex and hard to manipulate and more work in the field is needed.

[4]For implementing latent space interpolation we referenced https://machinelearningmastery.com/how-to-interpolate-and-perform-vector-arithmetic-with-faces-using-a-generative-adversarial-network/

## References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[4] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR, 06–11 Aug 2017.

## A Timeline and task allocation

We first implemented and trained an ACGAN model and made sure the ACGAN was beginning to output realistic images. Our next step was to implement the basic DC-GAN model and made sure the DC-GAN was beginning to output realistic images. Lastly, we implemented the WGAN model and ensured the images were realistic. From this point we performed latent space analysis on the DC-GAN and ACGAN and plotted images for loss curves and gradient information in order to perform comparison between models. We then created the poster and drafted the report.

Adrian Lopez worked on the implementation for the DC-GAN according to the specifications of [2] but fit to produce an output of a 32x32 image instead of the 64x64 images produced for Image-Net. Josh Rabinowitz worked on modifying the DC-GAN to run the WGAN algorithm and plot loss curves and generated images from the WGAN[3]. Harsh Bandhey and Adrian Lopez worked for running of AC-GAN according to the specifications of [4]. Josh Rabinowitz and Harsh Bandhey worked towards showing the effects of vector manipulation. Harsh Bandhey and Adrian Lopez worked towards showing evidence towards the vanishing gradients problem.

## B Implementation detail

### B.1 DC-GAN Model Structures

Table 1: DC-GAN Discriminator layers and parameters

| Operation | Kernel | Strides | Padding | Feature maps | BN | Nonlinearity |
|---|---|---|---|---|---|---|
| *D(x) – 128 x 3 × 32 × 32 input* | | | | | | |
| Convolution | $4 \times 4$ | 2 | 1x1 | 32 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 64 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 128 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 256 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 1 | No | Sigmoid |

Table 2: DC-GAN Generator layers and parameters

| Operation | Kernel | Strides | Padding | Feature maps | BN | Nonlinearity |
|---|---|---|---|---|---|---|
| *G(z) – 128 x 100 × 1 × 1 input* | | | | | | |
| Transposed Convolution | $4 \times 4$ | $1 \times 1$ | 1x1 | 1024 | Yes | ReLU |
| Transposed Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 512 | Yes | ReLU |
| Transposed Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 256 | Yes | ReLU |
| Transposed Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 3 | No | Tanh |

### B.2 DC-GAN Hyperparameters

Table 3: DC-GAN Hyperparameters

| Hyperparameter | Value |
|---|---|
| Optimizer | Adam ($\alpha = 0.0002$) |
| Batch size | 128 |
| Iterations | 200 |
| Leaky ReLU Slope | 0.2 |
| Weight | Isotropic gaussian ($\mu = 0, \sigma = 0.02$) |
| Bias initialization | Constant(0) |

### B.3 WGAN Model Structures

Table 4: WGAN Discriminator layers and parameters

| Operation | Kernel | Strides | Padding | Feature maps | BN | Nonlinearity |
|---|---|---|---|---|---|---|
| *D(x) – 128 x 3 × 32 × 32 input* | | | | | | |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 32 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 64 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 128 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 256 | Yes | Leaky ReLU |
| Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 1 | No | 0.0 |
| None | | | | | | |

Table 5: WGAN Generator layers and parameters

| Operation | Kernel | Strides | Padding | Feature maps | BN | Nonlinearity |
|---|---|---|---|---|---|---|
| *G(z) – 128 x 100 × 1 × 1 input* | | | | | | |
| Transposed Convolution | $4 \times 4$ | $1 \times 1$ | 1x1 | 1024 | Yes | ReLU |
| Transposed Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 512 | Yes | ReLU |
| Transposed Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 256 | Yes | ReLU |
| Transposed Convolution | $4 \times 4$ | $2 \times 2$ | 1x1 | 3 | No | Tanh |

### B.4 WGAN Hyperparameters

Table 6: ACGAN Hyperparameters

| Hyperparameter | Value |
|---|---|
| Optimizer | RMSProp ($\alpha = 5e - 5$) |
| Batch size | 64 |
| Iterations | 100 |
| Leaky ReLU Slope | 0.2 |
| Weight | Isotropic gaussian ($\mu = 0, \sigma = 0.02$) |
| Bias initialization | Constant(0) |

### B.5 ACGAN Model Structures

Table 7: ACGAN Discriminator layers and parameters

| Operation | Kernel | Strides | Feature maps | BN | Dropout | Nonlinearity |
|---|---|---|---|---|---|---|
| D(x) – 32 × 32 × 3 input | | | | | | |
| Convolution | 3 × 3 | 2 × 2 | 16 | No | 0.5 | Leaky ReLU |
| Convolution | 3 × 3 | 1 × 1 | 32 | Yes | 0.5 | Leaky ReLU |
| Convolution | 3 × 3 | 2 × 2 | 64 | Yes | 0.5 | Leaky ReLU |
| Convolution | 3 × 3 | 1 × 1 | 128 | Yes | 0.5 | Leaky ReLU |
| Convolution | 3 × 3 | 2 × 2 | 256 | Yes | 0.5 | Leaky ReLU |
| Convolution | 3 × 3 | 1 × 1 | 512 | Yes | 0.5 | Leaky ReLU |
| Linear | N/A | N/A | 11 | No | 0.0 | Soft-Sigmoid |

Table 8: ACGAN Generator layers and parameters

| Operation | Kernel | Strides | Feature maps | BN | Dropout | Nonlinearity |
|---|---|---|---|---|---|---|
| *Gx(z) – 110 × 1 × 1 input* | | | | | | |
| Linear | N/A | N/A | 384 | No | 0.0 | ReLU |
| Transposed Convolution | 5 × 5 | 2 × 2 | 192 | Yes | 0.0 | ReLU |
| Transposed Convolution | 5 × 5 | 2 × 2 | 96 | Yes | 0.0 | ReLU |
| Transposed Convolution | 5 × 5 | 2 × 2 | 48 | Yes | 0.0 | ReLU |
| Transposed Convolution | 5 × 5 | 2 × 2 | 3 | No | 0.0 | Tanh |

### B.6 AC-GAN Hyperparameters

Table 9: AC-GAN Hyperparameters

| Hyperparameter | Value |
|---|---|
| Optimizer | Adam ($\alpha = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999$) |
| Batch size | 100 |
| Iterations | 100 |
| Leaky ReLU Slope | 0.2 |
| Weight | Isotropic gaussian ($\mu = 0, \sigma = 0.02$) |
| Bias initialization | Constant(0) |

## C Additional experiment results

### C.1 Loss Instability

From the below plots of the loss of the DC-GAN and WGAN discriminator we can see that the WGAN loss curves are much more constrained and trend in a more stable manner. The DC-GAN loss curve fluctuates wildly and indicates training instability.

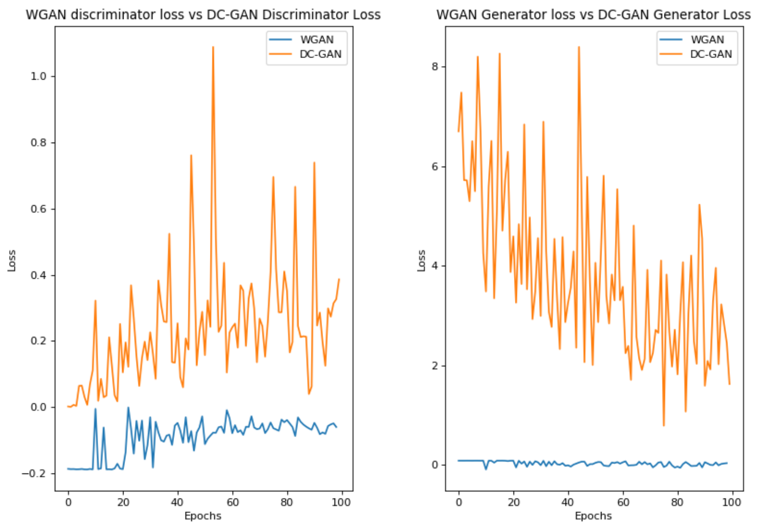


Figure 8: Loss Curves for the WGAN and DCGAN Discriminator And Generator

A note about the WGAN algorithm is that in order to implement $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ we had to negate the Wasserstein distance. This is due to the fact that PyTorch natively performs $w \leftarrow w - \alpha \cdot \text{RMSProp}(w, g_w)$. Thus the loss for the WGAN appears negative. Note however that the volatility of the WGAN discriminator loss curve is much less than the DC-GAN loss curve, suggesting a much smoother training process most likely due to informative gradients being passed.

### C.2 ACGAN Loss Curves

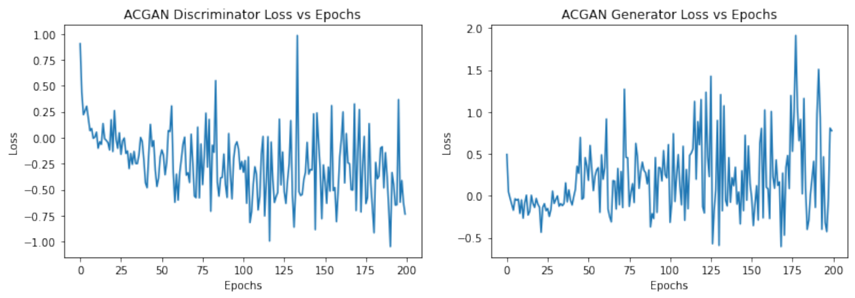


Figure 9: Loss Curves for the ACGAN Discriminator and Generator