Performance Comparison of GAN Variants

Jimmy Hickey
Department of Statistics
North Carolina State University
Raleigh, NC 27606
jhickey@ncsu.edu

David Elsheimer
Department of Statistics
North Carolina State University
Raleigh, NC 27606
dbelshei@ncsu.edu

Abstract

Generative adversarial networks (GANs) are a relatively new development in generative modeling. GANs work via the training of two models in tandem; a generative model G which attempts to simulate the actual data distribution, and a discriminative model D which estimates the probability of a sample coming from training data rather than G. Since its inception, many GAN variants have been developed. We will demonstrate the benefits of using a convex variant, the Wasserstein GAN, over the regular GAN. We will compare these variants based on performance metrics such as discriminator prediction accuracy and how the performance scales with time and problem size.

1 Introduction

A brief understanding of the GAN framework is important. The generator follows a distribution p_g defined on data x. As input, the generator takes in a noise variables with prior distribution $p_z(z)$. The generator then attempts to map this input to the real data distribution, p_{data}. This is done by a differentiable function, $G(z, \theta_g)$, that is represented by a perceptron with multiple layers and parameters θ_g. Similarly, the discriminator, $D(x, \theta_d)$, is represented by θ_d. The output of the $D(x)$ is the probability that its input came from the true data, and not the generated data. The discriminator is being trained to maximize the probability of correctly assigning labels to training examples and fake examples from G. The generator is being trained in tandem to minimize $\log(1 - D(G(z)))$. In essence, the generator is trying to output values that are similar to the actual input data distribution. As such, G and D are working against one another in a minimax game with the following value function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}\Big(\log(D(x))\Big) + \mathbb{E}_{z \sim p_z(z)}\Big(\log(1 - D(G(z)))\Big).$$

(Goodfellow et al. 2-3) While GAN variants will each have key differences, this adversarial training idea persists throughout. For more detailed information, the seminal paper *Generative Adversarial Nets* by Ian Goodfellow and others should be considered.

Preprint. Under review.

Algorithm 1 Minibatch stochastic gradient descent for training GANs. Number of steps applied to discriminator = k

for number of training iterations **do**
  for k steps **do**
    Sample m noise samples from $p_g(\boldsymbol{z})$.
    Sample m examples from $p_{data(\boldsymbol{x})}$.
    Update discriminator by ascending gradient:

$$\nabla_{\theta_d}\left(\sum_{i=1}^{m}(\log(D(x_i)) + 1 - \log(D(G(z_i))))\right)$$

  end for
  Sample m noise samples from $p_g(\boldsymbol{z})$.
  Update generator by descending the stochastic gradient:

$$\nabla_{\theta_g}\left(\sum_{i=1}^{m}(1 - \log(D(G(z_i))))\right)$$

end for These updates can follow any rule that is a standard gradient-based rule.

(Goodfellow et al. 4)

The discriminator loss function is measured by how accurately it can distinguish generated data from real data. If the generator is performing well, these accuracies should be similar. Asymptotically, one would hope to see the discriminator accuracy for real and fake data meet at 0.5, meaning that the generator is doing so well that the discriminator is essentially guessing.

1.1 Wasserstein GAN

In the scope of this analysis, basic GAN methods are compared to a particular variant of GANs: The Wasserstein GAN (WGAN). We will examine the weight clipping variant and discuss the gradient penalty variant. A WGAN minimizes the Wasserstein (Earth-Mover) Distance, a convex measure defined as follows:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} E_{(x,y)\sim\gamma}\Big(\|x - y\|\Big) \tag{1}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of all joint distributions of the form $\gamma(x, y)$, which respectively have the marginal distributions $\mathbb{P}_r, \mathbb{P}_g$. $\gamma(x, y)$ shows how much "mass" or "earth" needs to be moved from x to y to transform its marginal $\mathbb{P}_r$ to $\mathbb{P}_g$. (Arjovsky et al. 4) The proof of its convexity can be found in the Supplemental Materials.

The weight clipping WGAN has a similar algorithm to the basic GAN, with some key changes, as seen below. The function in the gradient f_w is known as the critic function (this takes the place of the discriminator). (Arjovsky et al. 7)

Algorithm 2 Minibatch stochastic gradient descent for training GANs. Number of steps applied to discriminator = k

Require: α (learning rate), c (clipping parameter), m (batch size), n_{critic} (number of iterations of the critic per generator iteration)

Require: w_0 (initial critic parameters), θ_0 (initial generator)

for number of training iterations **do**
  while θ not converged **do**
    for $t \in [1 : n_{critic}]$ **do**
      Sample m noise samples from $p_g(\boldsymbol{z})$.
      Sample m examples from $p_{data(\boldsymbol{x})}$.
      Update gradient: $g_w \leftarrow \nabla_w \left(\sum_{i=1}^{m} (f_w(x_i) - f_w(G_\theta(z_i)) \right)$
      Update w
      $w \leftarrow \text{clip}(w, -c, c)$
    end for
    Sample m noise samples from $p_g(\boldsymbol{z})$.
    Update generator: $g_\theta \leftarrow -\nabla_{\theta_g} \left(\sum_{i=1}^{m} f_w(g_\theta(z_i)) \right)$
    Update θ
  end while
end for

(Arjovsky et al. 8)

There are some drawbacks to weight-clipping that gave rise to the development of another variant of WGAN which uses a gradient penalty. Weight clipping introduces a problem where the weights that are computed are at extremes. By instead substituting in a gradient penalty in place of clipping, this problem is avoided (Hong et al. 7) We will show the effect of weight clipping in the Discussion section. The primary difference between these approaches is in how their discriminator and generator loss functions are defined. These can be seen below.

Figure 1: Loss Functions

Type	Discriminator	Generator
Basic	$\mathcal{L}_{\text{D}}^{\text{GAN}} = - \mathop{\mathbb{E}}_{x \sim p_d} \Big(\log(D(x)) \Big) - \mathop{\mathbb{E}}_{\hat{x} \sim p_g} \Big(\log(1 - D(\hat{x})) \Big)$	$\mathcal{L}_{\text{G}}^{\text{GAN}} = \mathop{\mathbb{E}}_{\hat{x} \sim p_g} \Big(\log(1 - D(\hat{x})) \Big)$
WC	$\mathcal{L}_{\text{D}}^{\text{WGAN}} = - \mathop{\mathbb{E}}_{x \sim p_d} \Big(D(x) \Big) - \mathop{\mathbb{E}}_{\hat{x} \sim p_g} \Big(D(\hat{x}) \Big)$	$\mathcal{L}_{\text{G}}^{\text{WGAN}} = \mathop{\mathbb{E}}_{\hat{x} \sim p_g} \Big(D(\hat{x}) \Big)$
GP	$\mathcal{L}_{\text{D}}^{\text{WGANGP}} = \mathcal{L}_{\text{D}}^{\text{WGAN}} + \lambda \mathop{\mathbb{E}}_{\hat{x} \sim p_g} \Big((\|\nabla D(\alpha x + (1-\alpha)\hat{x})\|_2 - 1)^2 \Big)$	$\mathcal{L}_{\text{G}}^{\text{WGANGP}} = \mathop{\mathbb{E}}_{\hat{x} \sim p_g} \Big(D(\hat{x}) \Big)$

(Lucic et al. 3)

One major difference between the GAN and WGAN is how the methods assess performance. We can think of the WGAN critic loss function as the average critic score on real data minus the average critic score on generated data. Notice that these are scores and no longer label predictions, so the critic will give a positive score for a sample is suspects is real and a negative score for one it thinks is fake. Also the generator's loss function can be interpreted as the negative of the critic score on generated samples. A benefit of using this loss function is that it more directly relates to increasing generated data quality while training. A lower critic loss for generated samples (the orange line in our plots) will trend downward as generated data quality increases.

It can be hard to compare the proficiency of these models to one another. We use both qualitative and quantitative metrics to differentiate the GAN and the weight clipping WGAN. These ideas can be applied to the gradient penalty WGAN when implementation is complete and other forms of GAN as well.

2 Methodology

The data set used in this analysis is KANNADA MNIST. This is a data set of handwritten numeric digits; however, these are not the Arabic numerals widely used in the MNIST data set but instead these are the digits used by those who speak and write in Kannada, a language used in India.

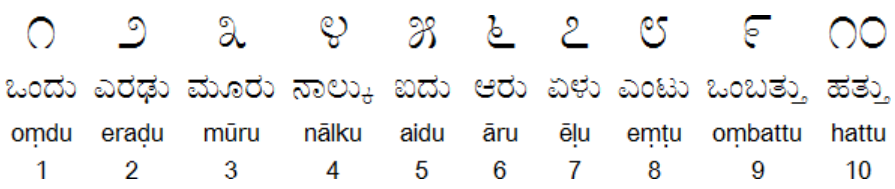


Figure 2: Kannada Numerals 1-10

(Omniglot)

2.1 Performance for different epochs

First, performance of the each GAN variant was assessed as the number of epochs increases. Changing no other factors, changing the epochs will result in a linear change in computation time. This is true for all variants being considered. One of the key differences across variants is that a basic GAN does not perform well when tasked with generating multiple different numbers. If more than one number is given, performance does not generally improve greatly even as the number of epochs is increased. Even when training on a subset of the data related to a specific number, performance for a basic GAN does not demonstrate convergent behavior in accuracy percentages. Ideally, as the number of epochs and thus iterations increase, there would be some convergent trend. Such behavior is present to an extent with the weight clipping variant of a Wasserstein GAN. A comparison of the two methods can be seen below, where the basic GAN is trained using only data with true response as the Kannada equivalent of 7. Wasserstein is trained using all Kannada numbers, to demonstrate its improved capabilities.

2.2 Performance for different batch sizes

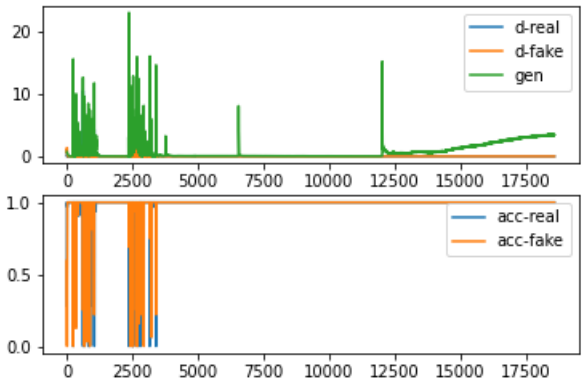
We can also measure differences in performance as we adjust batch size. Batch size denotes the number of samples drawn from the true data and the number of samples that the generator will create. The combination of these samples will then be given to the discriminator to judge. In the algorithms listed this quantity is denoted by m. In the code n_{batch} denotes the total batch size, which is made of half real and half generated data. The batch sizes considered here are 16, 32, 64, and 128. The number of epochs is fixed at 20 for each scenario. It should be noted that while epoch number remains fixed across these examples, a change in batch size results in a change in the number of iterations per epoch. The total number of iterations has an inverse relationship with batch size.

3 Analysis and results

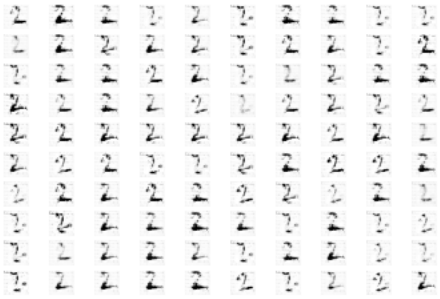
All of our analysis was run using Google Colab with a GPU for hardware acceleration.

3.1 Performance for different epochs

As the number of epochs increases the performance of the basic GAN does not display a convergence in the loss functions, nor in the accuracy rates. Accuracy rates near 1 for the generated data indicate the generator is performing poorly. As such it is ideal to have accuracy rates equivalent to one another at 0.5.



Loss functions and Accuracy rates



Generated Results (Epoch 27)

Figure 3: 20 Epochs of basic GAN

As the number of epochs is increased, the performance does not increase. This can be seen below from the output for a 200-epoch run (the results of the iteration 9200).

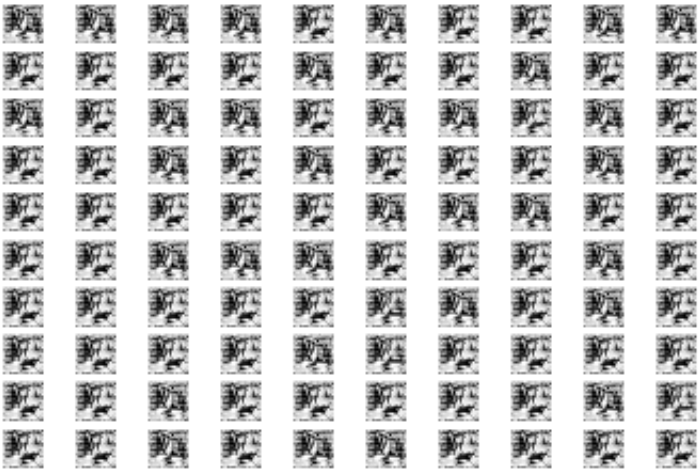
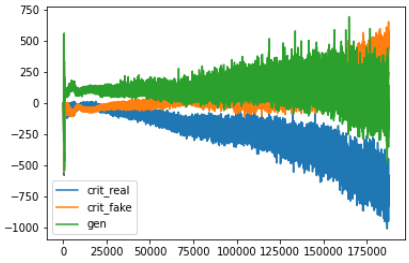
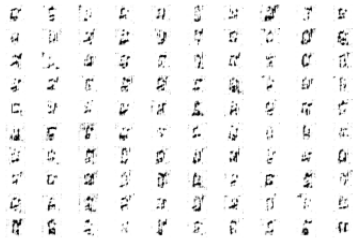


Figure 4: Basic GAN Epoch 200 (Iteration 9200)

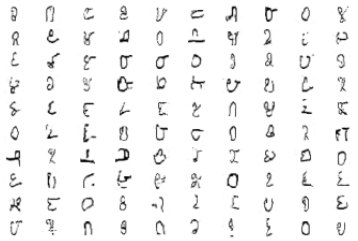
This lack of convergent behavior is contrasted by the weight-clipping WGAN variant, which demonstrated good performance even in the first epoch.



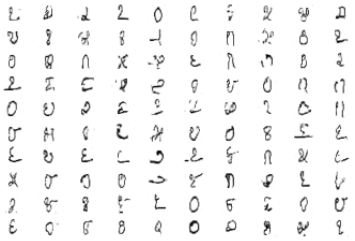
Loss functions



Generated Results (Epoch 1)



Generated Results (Epoch 100)



Generated Results (Epoch 200)

Figure 5: WGAN Sample Output

3.2 Performance for different batch sizes

The computational time as it scales with batch size is shown below for the basic GAN. For the loss function graph and generated output from these runs, please see the Supplemental Material.

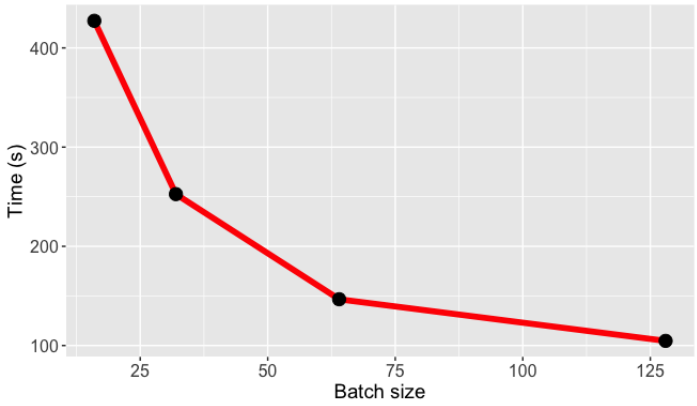


Figure 6: Batch Size vs. Comp. Time

As batch size increases, computation time decreases, but performance also appears to decrease. Based on the accuracy rates for the different batch sizes, it seems to be the case that decreasing the batch size results in more cases where the generator produces results indistinguishable from the real data, but at the expense of computation time.

For the weight-clipping WGAN variant, results were produced for batch sizes 16, 32, 64, and 128 with number of epochs fixed at 20. These results are provided below and again the loss function and generated outputs can be found in the Supplemental Material.

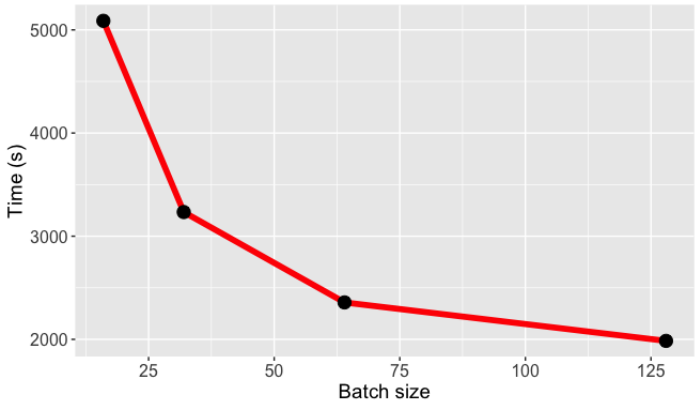


Figure 7: Batch Size vs. Comp. Time

The relationship between computation time and batch size is roughly similar to that of a basic GAN. However, with decreases in batch size, the loss functions tend to vary considerably more between iterations. The end result for each of these GANs is fairly reasonable, with all of the results showing easily distinguishable digits. As such, WGAN-WC demonstrates clear benefits over a basic GAN, with better results for larger batch sizes, and the ability to produce clear results even when training on multiple different classes.

4 Discussion

4.1 Drawbacks of weight clipping

To demonstrate the effect of clipping the weights on the distribution of the weights, see the histogram below.

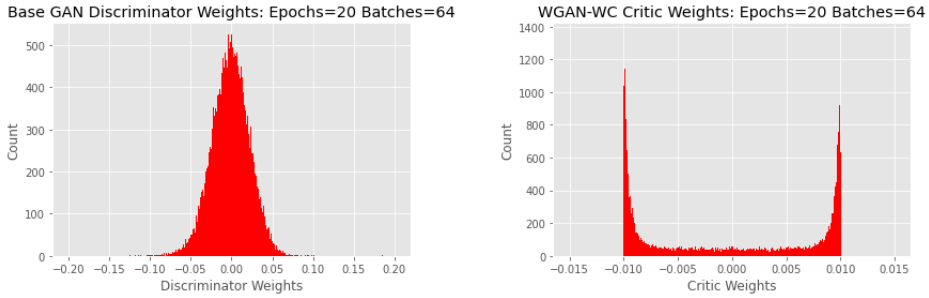


Figure 8: Discriminator Weight Distribution Comparison

As seen above, weight-clipping has the undesired effect of producing critic/discriminator weights that move the majority of the weights to the extremes. This undesired effect is remedied when a gradient penalty is added to the discriminator loss function.

4.2 Alternative analysis methods

Our analysis focuses on adjusting epoch and batch size for two different GAN variants. This has useful practical applications, however there are other metrics that could be used in measuring GAN performance. We later found the FID (Fréchet Inception Distance) (Heusel et al. 6) metric which is often used to measure the generative performance of a GAN. The study by Lucic et al. uses this measure to compare many GAN varieties including the basic GAN, WGAN with weight clipping, WGAN with gradient penalty, and more. They apply the GAN flavors across multiple standard data sets. This is definitely a better study in addressing output image quality.

Visually examining the images we generated for effect of epoch and batch size, it can can clearly be seen that the GAN performs worse than the WGAN-WC. However, using a metric like FID would be a more quantitatively rigorous way to measure how much worse the GAN is performing.

4.3 Conclusions and areas for further study

The Wasserstein GAN has clear benefits over just using a basic GAN. First and foremost, WGANs can generate good fakes using all classes of a response and a relatively low number of epochs. WGANs do not encounter the same inconsistency problem where the loss functions seem to follow no particular trend over the course of iterations. However, due to the weight-clipping that occurs in a normal WGAN framework, the weights are pushed to the extremes of the clipping range, as seen previously. This is not an issue encountered in a basic GAN, and is something which a gradient penalty modification to the discriminator network accounts for. The implementation of the gradient penalty is currently in progress.

In the future, we hope to compare a WGAN-GP framework to these two networks with respect to their computation time and performance for different epoch and batch sizes. Another variant worth study is the loss sensitive GAN (LS-GAN). One of the key differences between a LS-GAN and a WGAN is the fact that the LS-GAN makes pairwise comparisons between real and generative loss, ensuring that the generator coordinates with the real data in an effort to learn the optimal loss function. (Qi 10) Another area of interest is applying these methods to different types of data, where the data types are not in strict class form such as the numbers 0-9, but different art styles for a given image.

It would also be interesting to combine our methods with those used in Lucic et al. by furthering their exploration into hyperparameter search and computational budget (Lucic et al. 7) and combining it with our performance metrics. Knowing which generates the best images is helpful, but in practice we also need to know how long (at least relatively) training will take. This would offer more stratification across GAN flavors, allowing someone to choose which to use based on required image quality and time constraint.

References

[1] Arjovsky, Martin, et al. “Wasserstein GAN.” Arxiv, 6 Dec. 2017, arxiv.org/abs/1701.07875.

[2] Brownlee, Jason. “How to Develop a Wasserstein Generative Adversarial Network (WGAN) From Scratch.” Machine Learning Mastery, 17 July 2019, machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/.

[3] Brownlee, Jason. “How to Identify and Diagnose GAN Failure Modes.” Machine Learning Mastery, 8 July 2019, machinelearningmastery.com/practical-guide-to-gan-failure-modes/.

[4] Goodfellow, Ian J, et al. “Generative Adversarial Networks.” Arxiv, 10 June 2014, arxiv.org/abs/1406.2661v1.

[5] Hong, Yongjun, et al. “How Generative Adversarial Networks and Their Variants Work.” ACM Computing Surveys, vol. 52, no. 1, 2019, pp. 1–43., doi:10.1145/3301282.

[6] Huesel, Martin, et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium" Arxiv, 12 Jan, 2018 https://arxiv.org/abs/1706.08500

[7] “Image of First 10 Kannada Numbers.” Omniglot, https://www.omniglot.com/images/writing/kannada_num.gif.

[8] Lucic, Mario, et al. “Are GANs Created Equal? A Large-Scale Study.” Arxiv, 29 Oct. 2018, arxiv.org/abs/1711.10337.

[9] Prabhu, Vinay. “Kannada MNIST.” Kaggle, www.kaggle.com/c/Kannada-MNIST/overview.

[10] Qi, Guo-Jun. “Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities.” Arxiv, 19 Mar. 2018, arxiv.org/abs/1701.06264.