Least Squares Generative Adversarial Networks (LSGAN's) for Image Generation

Nabeel Sarwar, Ziv Schwartz, Elliot Silva
ns3429@nyu.edu, zs1349@nyu.edu, egs345@nyu.edu
Center for Data Science, New York University

Abstract – Generative Adversarial Networks (GAN) have been proven successful in generating images which seem realistic. However, typical GAN's utilize a sigmoid loss function, which can often lead to mode collapse and the failure of the model to produce authentic images. In this paper, we reproduce the successes of previous work demonstrating that using a least squares loss function will lead to improved stability in generating realistic images. We evaluate the success of this improved model by training over the Fashion-MNIST dataset.

1. Introduction

Our assigned paper focused on ways to improve the success of unsupervised learning methods in computer vision using generative adversarial networks (GAN). GAN's utilize both generator and discriminator components, wherein the generator seeks to produce images indiscernible in authenticity to a set of real images, while the discriminator seeks to identify which images are real and which are fake [1]. Since training the generator means improving its ability to trick the discriminator, the system can be thought of as an *adversarial neural network*.

Regular GAN's are implemented using the sigmoid cross entropy loss function, but this paper introduces the idea of implementing GAN's with a least squares loss function (LSGAN's) instead [2]. The reasoning for implementing a least squares loss function is due to the fact that GAN's implemented with sigmoid cross entropy loss tend to exhibit the vanishing gradient problem. The vanishing gradient problem describes the phenomenon found in artificial neural networks where the gradient becomes so miniscule that the weight effectively does not change values during training. This phenomenon does not occur when the GAN is implemented with a least squares loss function. In our project we experimented with implementing regular GAN's and LSGAN's, both with and without batch normalization. To evaluate these models we used the Frechet Inception Distance [3], a metric evaluating the similarity between the real and generated images (where lower is better). The FID is a distance that compares the Fretchet distance between two sets of Inception scores modeled as multivariate Gaussiains. Inception scores are outputs of network called Inception V3 that compare conditional labeled distribution with the marginal label distribution [4].

2. Data and Methodology

We used the PyTorch package to implement both our GAN and LSGAN, and trained both over the Fashion-MNIST dataset, a set of 60,000 images of shoes, shirts, dresses, pants, and other articles of clothing. In addition to the sigmoid vs least squares loss function for GAN vs LSGAN, we also experimented with architectures of various complexities, as well as batch normalization. We used GPU's configured on Google Colab to train each model for 200 epochs, then manually examined the resulting images produced, assessing them for image quality, blurriness, and how much they resemble actual articles of clothing. Additionally, for each model we calculated the Frechet Inception Distance (FID), a metric evaluating the similarity between generated images and the real images they were trained on.

3. Model Architectures

To implement these models, we adapted architectures from the LSGAN paper as closely as possible, making minor simplifications in reducing the number of layers in order to decrease run-time and prevent mode collapse, discussed further below. The architecture of our GAN's and LSGAN's are nearly identical, with the main

difference being the choice of sigmoid vs least squares loss function. The implementation of the loss function for the GAN utilizes the sigmoid function, as introduced in the Goodfellow et al., GAN paper [1]. The loss functions of the LSGAN is slightly more complex because of the square loss, as shown below:

$$\min_D V(D) = \frac{1}{2}E_{x\sim p_{\text{data}}(x)}[(D(x)-b)^2] + \frac{1}{2}E_{z\sim p_z(z)}[(D(G(x))-a)^2]$$

$$\min_D V^*(G) = \frac{1}{2}E_{x\sim p_{\text{data}}(x)}[(D(x)-c)^2] + \frac{1}{2}E_{z\sim p_z(z)}[(D(G(x))-c)^2]$$

If $b-c=1$ and $b-a=2$, then the LSGAN minimizes the Pearson χ^2 Divergence between $2p_g(x)$ and $p_{\text{data}}(x)+p_g(x)$. This could suggest choosing parameters like $a=-1$, $b=1$, and $c=0$, but since we used the architecture suggested by Mao et al. [2], which has the discriminator with a sigmoid as its final layer, this would lead to mode collapse as the a-value is unreachable. Instead, we used parameters $a=0$, $b=c=1$. This choice of parameters does not minimize the f-divergence, but it still enables the generator to generate realistic images, and has similar performance. This choice of parameters is also utilized in an implementation of the LSGAN listed by the authors of the paper.

A diagram of the architecture is shown in the Appendix. Our deconvolutional layers (ConvTranspose2D in PyTorch) were kernel size of 3, stride of 2, padding of 1, and output padding of 1; so they all upsampled the incoming image by a factor of two. Our convolutional layers (except the one in the generator) were kernel size of 3, stride of 2, and padding of 1; so they downsampled the incoming image by a factor of two.

We also tested each model with and without batch normalization, to observe the effects on stability and mode collapse. We experimented with the default momentum value of 0.1 from the LSGAN paper, as well as a value of 0.05, in order to attempt to diminish the instability we observed in some of the training runs.

Models	**No Batch Norm**	**Batch Norm (momentum = 0.1)**	**Batch Norm (momentum = 0.05)**
GAN	66.73	144.43	347.65
LSGAN	48.72	74.12	144.11

Table 1: FID scores for each model. The distance is evaluated between a set of generated images and the real images from the Fashion-MNIST test set.

4. Discussion

In terms of both quality of images generated and FID score, we observed that the LSGAN without batch normalization performed best. We observed that overall, the GAN was more likely to exhibit "mode collapse", wherein the generator is unable to improve itself. In fact, in many runs it would generate blurry images which didn't look like articles of clothing whatsoever. When this happened, the generator loss typically began increasing without ever improving, and the discriminator loss converges to zero, meaning the model is failing to produce images which could be mistaken as true images. To remedy this, we tried decreasing the complexity of our model by reducing the number of convolutional and deconvolutional layers, particularly in the discriminator. This yielded better results, though the LSGAN still performed best overall.

Generally, batch normalization (BatchNorm) improves the performance of GAN's, but we did not observe this effect. After various tunings, we experimented with adding two additional deconvolutional layers to the generator and one additional convolutional layer to the generator. This allowed the LSGAN with BatchNorm layers with default parameters to generate meaningful images with an FID score of 94.23. This implies that the generator and discriminator architectures above were not deep enough to benefit from BatchNorm. In addition, we used BatchNorm before the activation layers as suggested by the implementation of the LSGAN paper and in lecture. However, an experiment with BatchNorm done after activation suggests that this is a promising approach. Refer to Figure 5 to see images generated by this deeper LSGAN.

5. Conclusion and Future Work

Our experiments confirmed that LSGAN's do in fact lead to more stability when training a model used to generate images. Our LSGAN model was more likely to successfully finish model training without exhibiting mode collapse, produced images that were less blurry and more recognizable as articles of clothing, and had lower FID scores when compared to real images.

For future work, our group is interested in seeing if it is possible to extend this analysis to more complex image datasets. Additionally, applying a supervised (or even semi-supervised) approach by passing image metadata or labels to the model may prove beneficial. Lastly, our experiments with more layers suggests that it may be beneficial to pursue models with deeper architectures.

References

[1] Goodfellow et al., "Generative Adversarial Networks" (2014) https://arxiv.org/abs/1406.2661
[2] Mao et al., "Least Squares Generative Adversarial Networks" (2016) https://arxiv.org/abs/1611.04076
[3] Huesel et al., "GAN's Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium" (2016) https://arxiv.org/abs/1606.03498
[4] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X. (2016). Improved techniques for training gans. In Advances in Neural Information Processing Systems (pp. 22342242).

Appendix

Generator and Discriminator Loss:

$$\min_D V(D) = \frac{1}{2}E_{x\sim p_{\text{data}}(x)}[(D(x)-b)^2] + \frac{1}{2}E_{z\sim p_z(z)}[(D(G(x))-a)^2]$$

$$\min_D V^*(G) = \frac{1}{2}E_{x\sim p_{\text{data}}(x)}[(D(x)-c)^2] + \frac{1}{2}E_{z\sim p_z(z)}[(D(G(x))-c)^2]$$

We can ignore the expectation over the data when optimizing G, so the final cost for the generator is:

$$\min_D V(G) = \frac{1}{2}E_{z\sim p_g(z)}[(D(G(x))-c)^2]$$

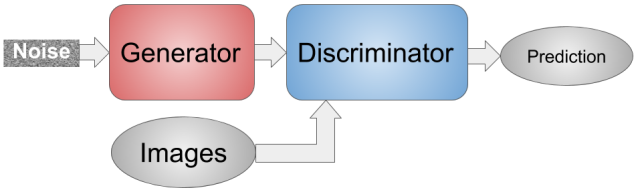

Figure 1: GAN Architecture

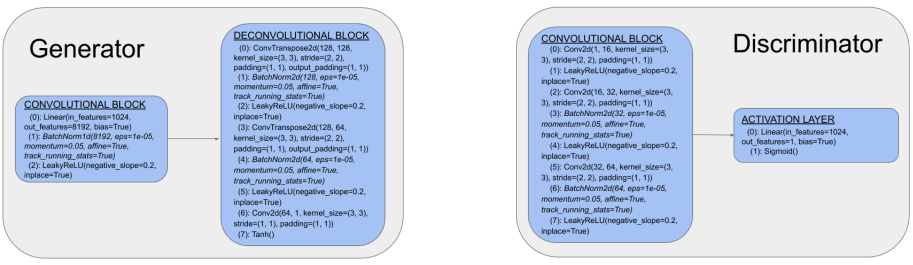

Figure 2: Generator architecture (left); Discriminator architecture (right)

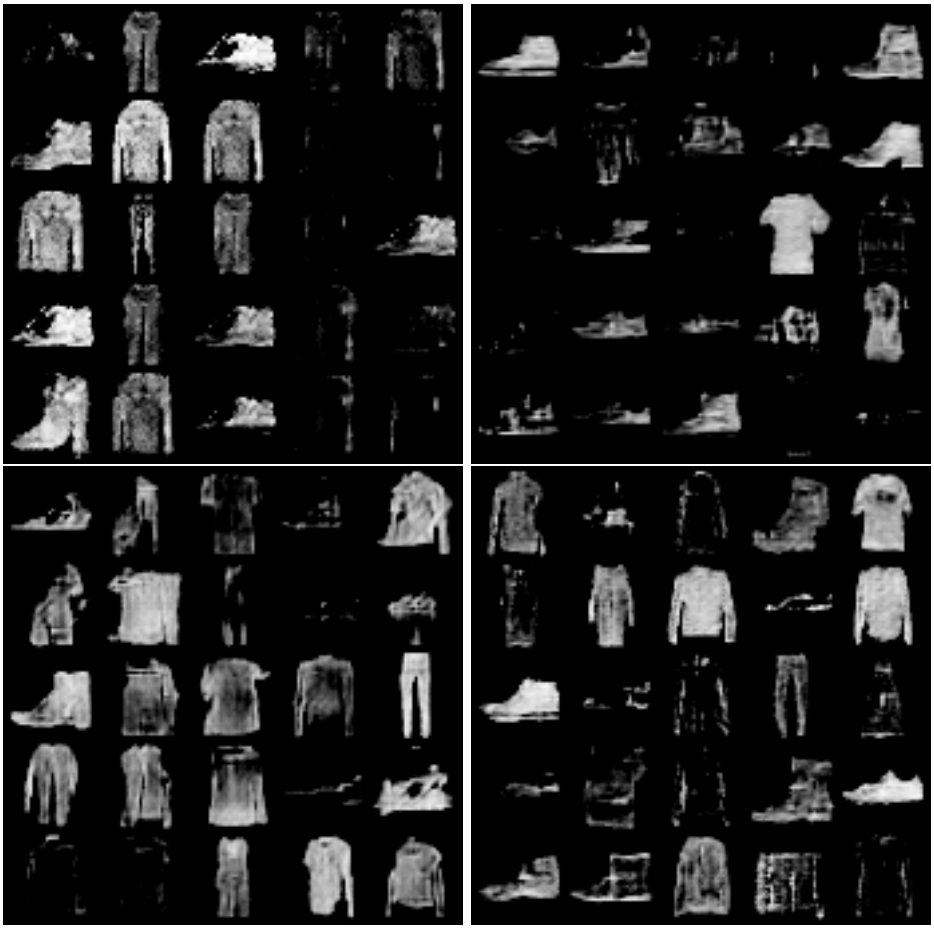

Figure 3: Images generated by our model. Top left: GAN with batch normalization. Top right: GAN without batch normalization. Bottom left: LSGAN with batch normalization. Bottom right: LSGAN without batch normalization.

Figure 4: Images generated by an LSGAN with batchnorm with two extra deconvolution and one extra convolutional layer in generator and discrimiantor respectively. They are better than the images generated by a GAN and LSGAN with momentum using our shallower architectures.