Conditional Generative Adversarial Networks

Parth Sagar Hasabnis, Master of Science in ECE[1]

Abstract

This paper discusses the implementation of a variation of Generative Adversarial Networks (GANs), namely Conditional GANs (cGANs). This technique was developed in an attempt to control the output of a Generative network by imposing a condition on the input. We have tested multiple implementations of this technique to generate images of FashionMNSIT dataset conditioned on the class labels.

1. Introduction

Generative Adversarial Networks is a technique for generative modelling which utilizes two agents - a Generator and a Discriminator (usually constructed using deep neural networks), to map inputs from a lower dimensional latent space to a higher dimensional data space. The training is set up as a two-player minimax game, where the discriminator attempts to identify fake data created by the generator, while the generator attempts to minimize the divergence between the fake and the real data.

Traditional GANs lack control over the generated output, making it difficult to traverse and identify patterns in the latent space necessary to generate the desired output. However, by conditioning the model on additional information, such as a desired class label, it is possible to direct the data generation process and achieve greater control over the output of the generator.

In this paper, we have implemented the conditioning by transforming the class label into an embedding, which served as an additional input to the generator along with the latent space variable. We have used the FashionMNSIT dataset to generate artificial images of clothing items and accessories. The code along with the saved models and other details can be found at https://github.com/parth-hasabnis/phasabni-ECE50024

2. Literature Review

This section will go over some of the mathematical background regarding the minimax problem introduced in the previous section.

2.1. Generative Adversarial Networks

Generative adversarial networks were proposed as a new approach for training generative models. They comprise two opposing models: a generative model G that models the underlying data distribution, and a discriminative model D that assesses the likelihood that a sample is real or generated by G. Both G and D are typically non-linear mappings, such as multi-layer perceptrons.

The goal is to train G and D to be ideal mappings, i.e. G produces samples indistinguishable from the training data, and D is able to perfectly identify the data as real or fake. This is achieved by training G and D in a two-player minimax game, where G tries to fool D by generating samples that are indistinguishable from real data, and D tries to correctly classify samples as real or generated

To learn the distribution p_g over the training data x, the generator tries to build a mapping function $G(z; \theta_g)$ from the latent space Z (usually a low dimensional Gaussian vector) to the data space. The discriminator tries to classify the input data as either being from the training space x, or from the distribution p_g by outputting a singular value between $[0, 1]$ which represents the probability of it belonging to training data rather than p_g.

Hence the loss function for a GAN consists of two parts: The generator loss and the discriminator loss. The generator loss L_G is a measure of how well the generator fools the discriminator. It is defined as the negative log-likelihood of the discriminator outputting 1 for a generated sample. The generator aims to minimize this loss, as it means that the discriminator is being fooled by the generated samples.

$$L_G =\equiv E[1 - log(D(G(z)))] \tag{1}$$

The discriminator loss L_D measures how well the discriminator can distinguish between real and generated samples. It is given by the sum of the negative log-likelihood that D classifies a generated sample as fake, and a training sample as real. The discriminator aims to maximize this loss, as it means that it can accurately distinguish between real and generated samples.

$$L_D = E[D(x)] + E[1 - log(D(G(z)))] \tag{2}$$

Hence G and D are trained together: G is trained to minimize $D(x)$, and D is trained to maximize $(1 - D(G(z))$. Combining equations (1) and (2), we get the final objective function of our problem.

$$\min_G \max_D V(D,G) = E_{x\ p_{data}(x)}[log\ D(x)] + E_{x\ p_z(z)}[log\ (1 - D(G(z)))] \quad (3)$$

2.2. Conditional GANs

The architecture of a GAN can be extended to a generative model by conditioning both the Generator and the Discriminator with some extra information y. This can be achieved by including y as additional input to G and D. As for the value of y, it can be encoded class labels for class-based data generation or a type of style for image-to-image translation. The objective function given by (3) would be modified to include this extra imposed condition.

$$\min_G \max_D V(D,G) = E_{x\ p_{data}(x)}[log\ D(x|y)] + E_{x\ p_z(z)}[log\ (1 - D(G(z|y)))] \quad (4)$$

3. Methodology

3.1. Implementing the objective function

The cGAN was implemented in Python using the PyTorch module. It offers a flexible environment with methods and functions to implement the cGAN using multi-layer perceptrons.

The loss function described in the previous section is implemented using the "Binary Cross Entropy Loss" in the PyTorch module. This loss is defined as:

$$L(\vec{x}, \vec{y}) = \sum_{i=1}^{n} -[y_i \cdot log(x_i) + (1 - y_i) \cdot log(1 - x_i)] \quad (5)$$

for training data $(\vec{x}, \vec{y})$. This loss can be used to calculate both of the components $log\ D(x|y)$ and $log\ (1 - D(G(z|y)))$ of the objective function, by specifying the value of y.

A mini-batch training approach was adopted for training the GAN. The set of images, consisting of both real and generated, is denoted by x_n. Adopting the convention of [1], we assign a label of $y = 1$ for the real images, and $y = 0$ for the generated images.

3.2. Training the discriminator

The goal of training the discriminator is to maximize the probability of a correct prediction. Hence, we aim to maximize $log\ D(x|y)$ + $log\ (1-D(G(z)))$. For each mini-batch, we first create a batch of real images from the training set, complete a forward pass through D, calculate the loss $[log\ D(x|y)]$ and then accumulate the gradients through a backward pass. Then the process is repeated by constructing a batch of fake images using the generator and calculating the loss $[log\ (1 - D(G(z)))]$. Now that the gradients from both the real and fake samples have been accumulated, the discriminator's optimizer is called to take a step.

3.3. Training the generator

The goal of training the generator is to maximize the probability that the discriminator misclassified the fake samples as real. Hence we aim to minimize $log\ (1 - D(G(z)))$. As demonstrated by [1], the direct approach of trying to minimize this loss does not provide large enough gradients, especially early in the training process. Instead, we aim to maximize $log(D(G(z)))$. This is accomplished by assigning a label $y = 1$ to the fake images as they are passed to the discriminator. In this way, the $log(x_i)$ part of the Binary Cross Entropy loss is used to calculate the gradients. Then we call the generator's optimizer to take a step.

3.4. Applying conditionality to the input data

The dataset in use is the Fashion MNIST dataset [4], which was developed as an alternative to MNIST [5] to benchmark machine learning algorithms. The dataset consists of 70000 grayscale images, each of dimension 28x28. The images denote 10 different clothing items and accessories, like shirts, trousers, bags, etc. The label of each image is encoded as an integer from $[0, 9]$.

This label serves as a condition to input data. In practice, we encode the label and input it to the generator along with the noise vector (and to the discriminator with the image data). There are multiple ways to encode this label, one popular strategy is to encode it as a one-hot vector and concatenate it with the input data. In our approach, we have encoded the label information using a learnable embedding, so that the network itself can decide the optimal way to encode the information.

4. Experimental Results

Many different architectures were chosen for the generator G and the discriminator D. In the subsequent sections, we will go over some of them. Both G and D were constructed as multi-layer perceptrons. In each experiment, they were trained with the same following parameters: The input layer for each generator is the same, which is a Linear layer that takes in a noise vector and the label embedding as the input. Similarly, the output layer for each discriminator is a Linear layer followed by a Sigmoid activation, which gives the

Hyperparameter	Value
Optimizer	ADAM
Learning Rate	0.0001
Embedding dimension	5
Epochs	50
Z size	100
Batch Size	128

Table 1. Values of the parameters used in training. Z size is the dimension of the input noise vector

probability of the input image belonging to training data rather than p_g. We vary the hidden layers for the networks to observe their impact on the output.

4.1. Constructing the generator and discriminator using dense layers

This approach was adopted by [3] in implementing a cGAN. They used 3 dense hidden layers of size [256, 512, 1024] for the generator. A similar architecture was used for the discriminator, which consisted of 3 dense hidden layers of sizes [1024, 512, 256]. Hence the resulting networks are Fully Connected Networks (FCN). This architecture will act as the baseline with which we would compare our results. The generated images are shown in Figure 1.

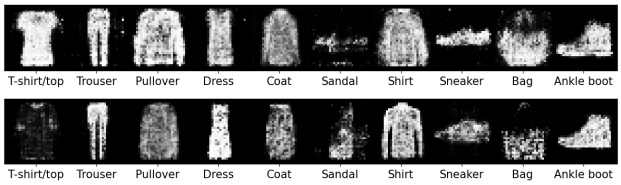

Figure 1. Output images generated after 30, 50 Epochs using only Dense Layers

The images generated using only dense layers roughly resemble the target classes but have significant salt and pepper noise. Their backgrounds are not perfectly black. This approach works only for targets that can be represented with high contrast and less detail (Eg: trousers, pullovers). It fails for low contrast high detail targets (Eg: sandals, sneakers) where the image is riddled with noise.

4.2. Constructing the generator and discriminator using convolutional layers

Fully Convolutional Networks have shown to be better than FCNs [6] for image classification and generation problems. We have experimented with networks of different depths to try and gauge the effect of the number of layers on the perceived quality of the generated images.

In the first iteration, our network architecture was inspired by [7], in which they used a modified VGG architecture [8] for the generator and the discriminator. The generator consists of 4 Transpose Convolution (Conv-T) Layers, with [128, 64, 32, 16] corresponding channels. The first three Conv-T Layer uses a kernel size of (3x3), padding of (1,1) and a stride of (2,2), while the final layer has a (4x4) kernel, a stride of (1,1) and no padding. The parameters of the final layer were chosen to get the final image size to (28x28). We perform batch normalization after each Conv-T layer, followed by a Leaky ReLu activation $\alpha = 0.2$. After the final ConvT layer, we apply a Hyperbolic Tangent activation function to get the outputs between [0, 1]. For the Discriminator, we use a single Convolutional (Conv) Layer with 16 channels, a stride of (2,2) and a kernel size of (3,3), followed by two Dense Layers with [512, 256] neurons. The stride was chosen as (2,2) to avoid a Pooling operation, with the downsampling now being carried out during convolution. After each Dense layer, we use a Leaky ReLu activation $\alpha = 0.2$ and a Droput of 30%. The generated images are shown in Figure 2.

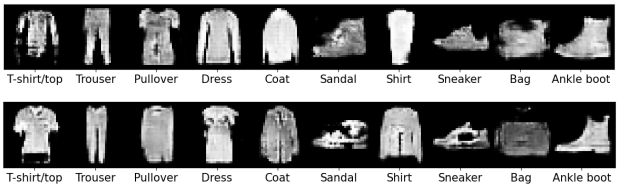

Figure 2. Output images generated after 30, 50 Epochs using a combination of Convolutional and Densec Layers

The images generated using Conv layers are a much closer representation of the training images when compared to the ones generated using Dense layers. They do not have the same salt and pepper noise and can generate images with a greater degree of detail in them. However, we can observe many visual artefacts in the generated images, like the Pullover does not have proper sleeves, and the T-shirt is quite deformed.

We tried another experiment by increasing the number of layers in the discriminator in an attempt to better its performance. We added a Conv Layer with the same parameters and a Dense Layer with [128] neurons. The generated images are shown in Figure 3. The resulting images bear the same characteristics as before but with less deformation in the generated images. Compared to the baseline, this architecture performs much better with imperceptible noise and significant variation in the generated images.

In our final experiment, the architecture of the generator was used to mirror the discriminator. Hence instead of using a hybrid of Conv and Dense Layers, the discriminator

Figure 3. Output images generated after 30, 50 Epochs after adding additional layers to the Discriminator

was constructed using solely 4 Conv Layers, followed by a singular Dense Layer. While the generated images resemble the Fashion objects, they do not agree with the label they were conditioned with, hence defeating the purpose of using conditionality. We suspect that the input data was over-compressed after using the 4 convolutional layers. The output data (4x4) after convolution did not represent the input image well, leading the discriminator to learn mismatched embeddings. The generated images are shown in Figure 4.

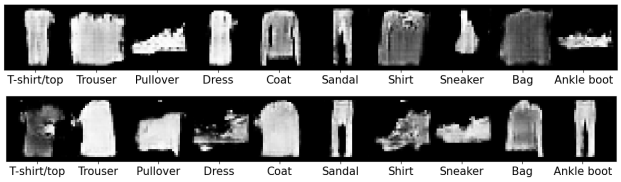

Figure 4. Output images generated after 30, 50 Epochs after using solely Convolutional Layers in both networks

5. Conclusion

Based on the conducted experiments, we conclude that the Generator model with 4 Conv-T Layers and the Discriminator model with 2 Conv Layers and 3 Dense Layers together form a satisfactory cGAN. We successfully demonstrate the application of conditionality to generate images with the required labels. When compared to the baseline, the generated images are more uniform and have significantly less salt-and-pepper noise

The model was trained multiple times using different weight initializations to ensure that it does not get stuck in a local minimum. The Loss curve (Figure 7) demonstrates the successful implementation of the training strategy, in that the Generator Loss is minimized and the Discriminator Loss is maximized at the end of training. It can also be observed that the numerical value of the loss stops changing after 43 epochs, and the model has reached its global optimum.

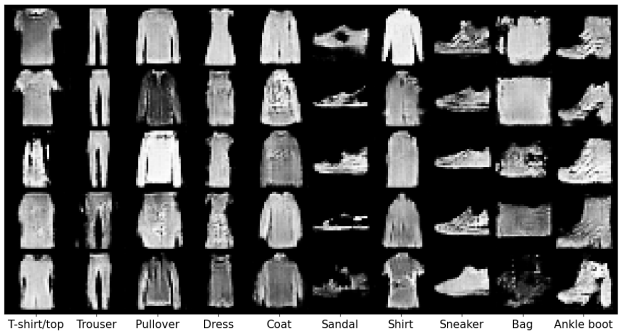

Figure 5. Images generated using the model described in the third experiment

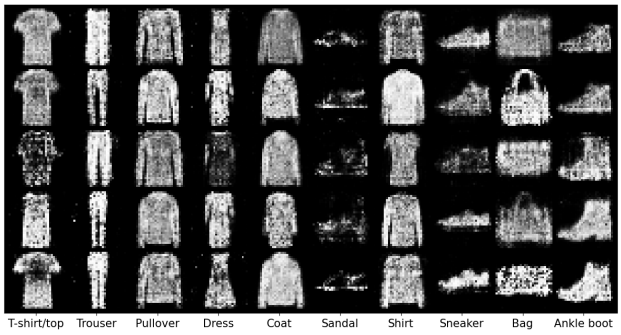

Figure 6. Images generated using the Baseline

6. Limitations and Future Work

The results presented in this paper were selected from the top 4 of the overall 10 experiments conducted. The results are quite preliminary and employ only two types of network architectures. The best-performing model sometimes produces distorted and unusable images with missing patches. Occasionally, it produces ”blobs” that do not resemble any label. The model also fails at generating satisfactory images of the ”Bag” label when compared to the baseline. More sophisticated network models using multiple Generators and Discriminators, as demonstrated in [9] can be used to achieve superior performance. The current hyperparameters were chosen mostly based on convention (Eg: Size of Noise vector) and minimal experimentation. Their values can be optimized by employing a grid-search method.

7. Acknowledgements

This paper was written as a part of the final class project for Purdue ECE 50024: Machine Learning. We would like to thank the Instructor Dr Qi Guo and the teaching assistants for their help throughout the course. I would also like to thank OpenAI and ChatGPT which aided me in refining my report by correcting grammatical errors and simplifying my language.

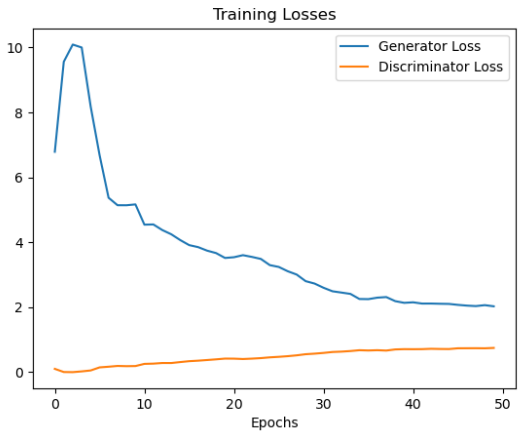

Figure 7. Training Loss Curves

8. References

[1] Goodfellow, I. J., Mirza, M., Xu, B., Ozair, S., Courville, A., Bengio, Y. (2014). Generative Adversarial Networks. ArXiv. /abs/1406.2661

[2] Mirza, M., Osindero, S. (2014). Conditional Generative Adversarial Nets. ArXiv. /abs/1411.1784

[3] https://github.com/qbxlvnf11/conditional-GAN

[4] Xiao, H., Rasul, K., Vollgraf, R. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. ArXiv. /abs/1708.07747

[5] Deng, L., 2012. The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), pp.141–142.

[6] Long, J., Shelhamer, E., Darrell, T. (2014). Fully Convolutional Networks for Semantic Segmentation. ArXiv. /abs/1411.4038

[7]Pytorch Tutorial - DCGAN https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html

[8] Simonyan, K., Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. ArXiv. /abs/1409.1556

[9] Zhu, J., Park, T., Isola, P., Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ArXiv. /abs/1703.10593