Generative Models for Handwritten Digits

Evan Hernandez, David Liang, Sahib Pandori,
Viswesh Periyasamy, Shantanu Singhal

Abstract - Ordinary deep networks are effective for creating discriminative models, but sometimes a generative model is necessary, particularly if one wishes to construct new data according to a target distribution. Generative adversarial networks have been shown to realize such generative models and to generate realistic, structurally sound images. We implemented a deep convolutional generative adversarial network and applied it to the MNIST dataset. We qualitatively discuss the results and analyze the convergence of the GAN.

1 INTRODUCTION

Since their resurgence in 2012 by Krizhevsky et al. [4], convolutional neural networks have experienced great success as discriminative models. They are especially popular in the computer vision domain, where they are used primarily for image labeling and object recognition.

In some tasks, the goal is not to distinguish between existing data points but to generate new data points according to some target distribution. For example, suppose we wished to generate images of bedrooms. It is not immediately clear how a convolutional neural network would go about doing this. A model for this task must capture the semantics of a bedroom: it must understand that bedrooms coincide with beds, blankets, pillows, nightstands, lamps, windows, doors, etc., and it must understand how each of these components connect in terms of visual structure.

Many non-parametric methods, such as texture synthesis [1], attempt this task but ultimately fall short when complex input data are provided. Continuing the previous example, bedrooms do not exhibit uniform patterns that could be exploited by a non-parametric model. It would

seem that any good model for this task should leverage the availability of training data and the representational power of parametric methods like neural networks.

Generative adversarial networks (GANs) do just that. GANs consist of two neural networks, a generator and a discriminator, playing a game of cat and mouse. The generator creates new data from by drawing from and manipulating a noise distribution, while the discriminator distinguishes between generated and real data drawn from a target distribution. Through this adversarial process, the generator learns to produce structurally realistic images, and the discriminator learns to distinguish real vs. fake, until the latter can no longer make confident distinctions.

We implemented a deep convolutional generative adversarial network (DCGAN) using TensorFlow. In this report, we describe our DCGAN implementation in detail and display our results on the MNIST datasets. We then analyze the efficacy of the generator by monitoring its (and the discriminator's) outputs during the training process. We also discuss the effects of network hyperparameters on the results. Finally, we describe room for future work with DCGANs.

2 Background

Most classifiers, such as ordinary convolutional neural networks, learn discriminative models, meaning they learn the conditional probability distribution $P(y|x)$, where y is the class label and x is the example. Other models are generative, meaning they learn the full joint distribution $P(x, y)$. By virtue of estimating the full joint distribution, generative models offer more information about the relationships between variables than do discriminative models, making them especially powerful for creating new data.

While traditional convolutional neural networks are used in the classification setting, and thus learn discriminative models, GANs, first described by Goodfellow et al. in [2], learn a generative model by pitting two networks against each other in competition. One network, the generator, maps a prior noise distribution to an image in an effort to mimic to true data distribution. The other network, the discriminator, distinguishes between generated data and real data, outputting the probability that the given image came from the real distribution. These two networks compete in a minimax game, until the discriminator can no longer distinguish real data from generated data.

More formally, given a noise distribution Z and a real distribution T, the generator is a differentiable function $G(z;\theta_G)$ represented by a neural network with parameters θ_G mapping elements from the noise distribution to the target distribution. The discriminator is a differentiable function $D(x;\theta_D)$, also represented by a neural network, returning the probability that x belongs to T. The optimal solution to the minimax game, where we define $V(G,D)$ to be the value function, is given by the following:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_T(x)}[\log D(x)] \; + \; \mathbb{E}_{z\sim p_Z(z)}[\log(1-D(G(z)))]$$

Those familiar with game theory might appreciate the fact that optimizing this objective is equivalent to realizing a Nash equilibrium between the generator and the discriminator.

In practice, since D and G are represented by neural networks, we can approximate the optimal solution via backpropagation. The loss function for the discriminator, given datapoints x from the target distribution and z from the noise distribution, is given by

$$L_D(x,z) = \log D(x) + \log(1 - D(G(z)))$$

and the loss function for the generator is given by

$$L_G(z) = \log(1 - D(G(z)))$$

Goodfellow et al. showed in [2] that the equation above describes the optimal discriminator and that backpropagation always converges to the optimal solution, given enough time and enough data.

Goodfellow et al. demonstrate in their seminal paper the generative power of GANs. They successfully use GANs to generate both realistic simple images, like digits akin to MNIST digits, as well as complex images with recognizable structure and form, like faces and bedrooms [2].

Radford et al. extend GANs to utilize deep convolutional structures [6]. In particular, they establish a number of architectural guidelines to increase stability during training. These guidelines include using fractional strided convolutions (i.e., reverse convolution), batch normalization, and no fully connected layers. Additionally, they show that the learned filters from DCGANs can be used to construct richer feature sets to improve image classification. They demonstrate state-of-the-art performance on the CIFAR dataset using this method.

Other literature, namely Gregor et al. with their so-called DRAW network, have improved GAN image generation further by using recurrent networks and LSTM models [3]. Such networks use a spatial attention mechanism to generate the image one section at a time, in a sense "drawing" it as a human would. While recurrent GANs such as these have generated images that are nearly indistinguishable from their real counterparts, they are beyond the scope of our project and are mentioned here for completeness.

3 Implementation

We implemented two networks: an ordinary GAN, akin to the one described by Goodfellow et al. in [2], and a DCGAN, akin to the one described by Radford et al. in [6].

Our GAN includes a generator that drew samples from a 100-dimensional standard normal distribution with two layers of 150 and 300 hidden units respectively and a final output layer that is reshaped into the appropriately sized image. All hidden units use rectified linear activations, and the output layer applies a hyperbolic tangent activation. The discriminator

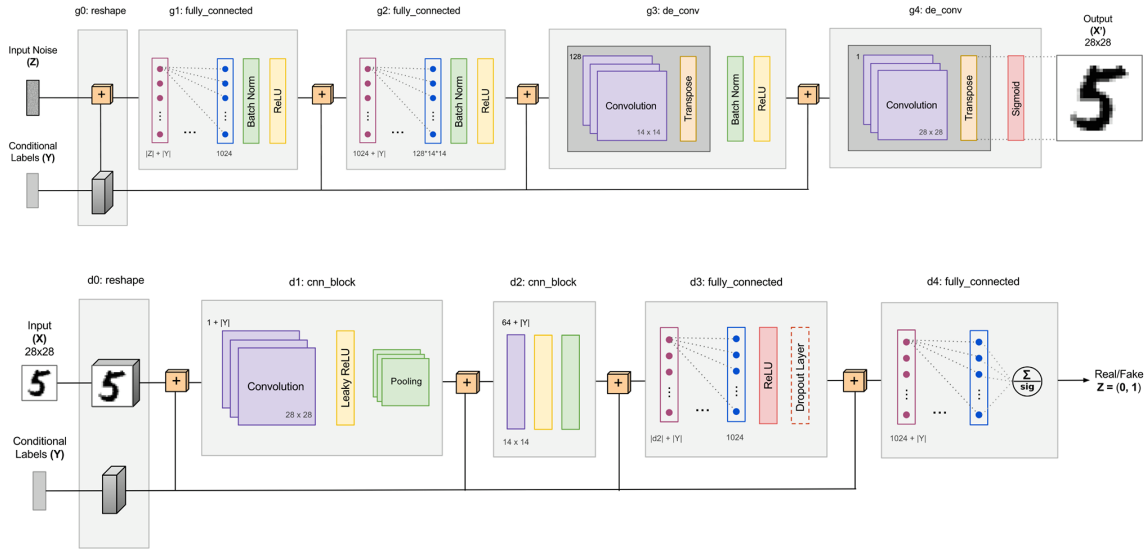

Figure 1: Network architectures for the generator (top) and discriminator (bottom).

network has the same structure but reversed, and the output layer consists only of two nodes: one representing the confidence that the image was generated, and the other representing the confidence that the image was real. The network was trained by sampling one minibatch of real images and random noise at a time, forward propagating this batch through the discriminator and generator, and then computing the respective losses for each network for each sample and backpropagating.

Our DCGAN follows the structure described by figure 1. In addition to swapping fully connected hidden layers with convolutional/deconvolutional layers, the DCGAN also accepts a conditional label as input to guide the training process. The label is a 1-of-n encoded vector that determines which type of image the generator should generate (e.g., for MNIST, it describes what digit to generate). This information is also passed to the discriminator. The other components of the training process are nearly identical to that of our ordinary GAN.

All code was written in python and all network components were implemented with numpy and TensorFlow. The hyperparameters were varied depending on the dataset at hand and are described in more detail in the following section.

4 Experimental Results

We now discuss our experimental results with our DCGAN. We omit the results of our original GAN because the focus of our project was to implement the DCGAN. The original GAN was simply a stepping stone in the development process, which we described in the previous section for sake of completeness.

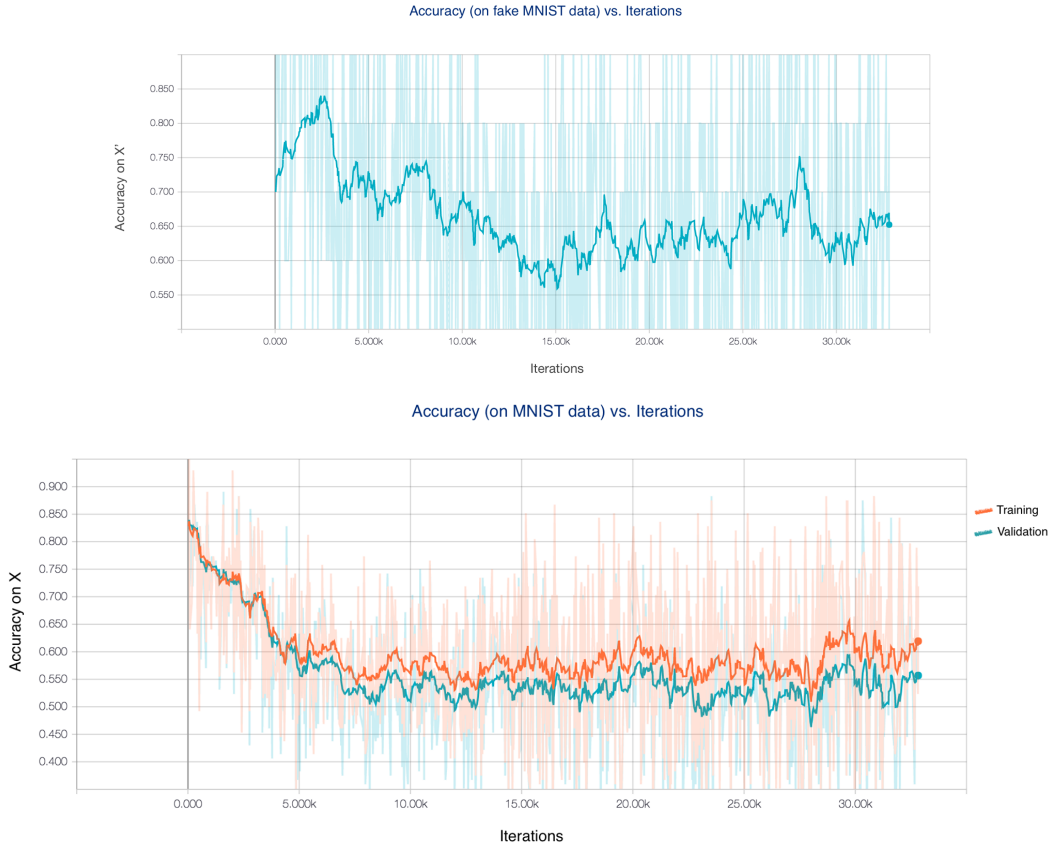

Figure 2: Discriminator confidence on generated images (left) and real MNIST data (right) as functions of training iteration. As the generator improves its output, the discriminator struggles to distinguish between real and fake. Note that the data points for the left plot were taken during the validation step.

We tested our DCGAN on the MNIST dataset. MNIST is a collection of 70000 labeled handwritten digits, including the numbers 0 through 9, and is a popular testbed for classifiers and, in our case, generative models. Hand-written digits are complex enough that non-parametric methods cannot generate realistic duplicates, but simple enough (and with enough available data) that a GAN can generate interesting results. Since our goal was to generate realistic images, the results we describe are mostly qualitative.

We trained our networks for 100 epochs using the Adam optimizer. We used a learning rate of 0.0002, a batch size of 128, with a learning rate decay (used by Adam) of 0.5. Each epoch used all samples in the training partition of MNIST, so 60,000 examples in total.

Figure 3 shows the generator's output after 0, 5, 10, 15, 20, and 25 epochs, in that order. Surprisingly, after only one epoch the generator's output adopts the structure of the training

data. By epoch 25, the digits are nearly indistinguishable from the MNIST data. One can see that as training progresses, the digits obtain more structure with thinner lines and less noise. Interestingly, the network even picks up on variations in the ways that each digit is drawn. For example, the network outputs some 2s with loops and some without loops. This emphasizes that the DCGAN is learning a high-level representation for the digit 2, and is not just duplicating the training data.

Figure 3: Generator output for each digit every 5 epochs, starting from the first epoch on the left.

From a more quantitative angle, figure 2 displays the value of the discriminator confidence when given fake data and real MNIST data as a function of iteration, where we treat the forward and backward propagation of one batch as an iteration. As training continues, the discriminator steadily becomes less confident because the generator produces more realistic images. Figure 2 also shows that the discriminator's confidence degrades to the optimum of $\frac{1}{2}$ (with respect to minimax loss).

Despite appearances, sometimes the network exhibited divergent behavior. The network began by generating shapes that resembled handwritten digits, but these shapes quickly became noise after a few epochs. We solved this by using a smaller learning rate, and by training the DCGAN for the full 100 epochs. We experimented with a number of decay parameters for the learning rate, but ultimately found that the rate we described above produced the cleanest results. Anything larger (say, 0.001) produced oscillations or divergence.

It is also worth mentioning that the speedy convergence displayed in the previous two figures did not hold for the ordinary GAN. Rather, to achieve the same degree of results, the ordinary GAN had to train for the full 100 epochs. Indeed, for the first 25 epochs, the generated images from the ordinary GAN look very similar to the output from the first epoch with the DCGAN. This points to the representational power of convolutional neural networks for image processing: where an ordinary, fully connected neural network likely overfits the training data and oscillates over its many degrees of freedom, a convolutional neural network must fit the data with many fewer parameters and thus learn a higher level representation of the concept at hand. As such, convolutional neural networks function more effectively as generative models.

5 Conclusions and Future Work

We demonstrated that generative adversarial networks, and in particular their deep convolutional variants, function exceptionally well as generative models. We described our GAN and DCGAN implementations and showed the realistic handwritten digits they produced.

There are a number of ways we could improve our GAN if we had more time. We especially would like to experiment with recurrent networks and LSTM models because they have been shown to outperform even DCGANs on image generation tasks [3]. Using these networks would also allow us to better visualize the image generation process, because generative recurrent networks use a selective attention mechanism and generate images one small chunk at a time.

Another interesting extension of this project would be to generate alternative representations of input data. For example, we could try to generate doodles from the corresponding images in the Sketchy database [5]. We did not pursue this idea for our project because it would have required a significantly more complicated network architecture, or perhaps a different model altogether, and moreover Sketchy does not provide enough training data by category to make this work with a neural network. Nevertheless, it seems like an interesting pursuit.

It is clear that generative models offer more representational power than their discriminative counterparts. We foresee great future success with GANs, DCGANs, and generative models in general.

REFERENCES

[1] Efros, A. A., Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *SIGGRAPH '01*.

[2] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets.

[3] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., Wierstra, D. (2015). DRAW: A recurrent neural network for image generation.

[4] Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks.

[5] Patsorn, S., Burnell, N. (2016). The Sketchy database: learning to retrieve badly drawn bunnies. In *ACM Transactions on Graphics (proceedings of SIGGRAPH)*.

[6] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR 2016*.