Experiments with GANs

Amol Kapoor
Department of Computer Science
Oxford University

Bernardo Orozco
Department of Computer Science
Oxford University

Abstract

Generative models are capable of approximating the joint distribution of a data set, allowing for a full probabilistic understanding of all variables. Generative models can be used for many of the same tasks as discriminative models, but can also be used to generate additional data and express more complex relationships between features defining the data set. Generative Adversarial Networks (GANs) are a popular framework for building generative models. In this paper, we examine and compare multiple GAN architectures on the MNIST generation task. We find that though deep single-pixel stride convolutional networks have the potential to perform better than linear networks and are more stable, linear networks train faster and have sharper outcomes within that training period.

1 Introduction

Deep learning aims to find detailed models that are capable of representing complex and potentially obscure relationships between features in large data sets. Discriminative deep learning models are capable of approximating conditional distributions of a data set $p(x|y)$. These models have made significant progress in many complex yet vital tasks, from object recognition to reinforcement learning. However, discriminative models cannot model joint distributions $p(x, y)$, and so struggle at tasks like speech synthesis or unsupervised data representation and compression. Generative models are capable of modelling joint distributions of features in a data set, providing a more full representation of the relationships between variables. Generative models and discriminative models excel at unique tasks, so we cannot rely on discriminative models alone. Therefore, analysis and improvement of generative models is an important field of study[1][2]. Further, following the intuitive idea that the ability to create (a data set) indicates an ability to understand (a data set), generative models also promise to play an important role in general artificial intelligence.

Generative adversarial networks, or GANs, are a popular method for training generative models. In the GAN framework, the generative model attempts to fool a counterpart discriminative model. The discriminative model, in turn, attempts to determine whether a sample is from the generative model or from the data distribution. Because the rewards are mutually exclusive, both models are driven to improve their understanding of the data set until the generated data is indistinguishable from the real data. Conveniently, backpropagation can be applied through the entire model[3].

Though the GAN framework was introduced with the simple multilayer perceptron[3], the generator and discriminator models can both be constructed using any combination of differentiable functions - in practice, combinations of convolutional[4][5], LSTM[6][7][8], and linear layers. In this paper, we train and compare the success of various architectures that use combinations of convolutional, LSTM, and linear layers on the MNIST generation task.

2 Related Work

Modelling joint probability distributions with generative models is a well studied problem in machine learning. Probabilistic generative models are efficient and effective . These methods include Hidden Markov Models and other mixture models[9], Gaussian Processes[10], and others. However, probabilistic models can struggle with large data sets and scale poorly. Further, they often require either computationally intractable calculations or significant assumptions about the data.

Deep generative models are built to train on large data sets and can model relationships in the data without any prior assumptions. Autoencoders are a class of deep networks that are capable of discovering a minimal representation of data from a larger data set. Autoencoders - including variational[11], sparse[12], and denoising[13] autoencoders - share a similar encoder-decoder architecture. They are unsupervised learning models that, in their most basic form, learn to reconstruct larger dimensional data from a smaller central layer[14]. Because the central layer is generally of a lower dimension than the data itself, autoencoders act well as data compressors[14]. While autoencoders can act as generators, they are explicitly rewarded for memorizing the data set instead of learning conceptual representations[13]. Empirically, autoencoder-generated images can be blurry[4].

By comparison, GANs are explicitly designed for the generation task. Though they are difficult to train, they also produce sharper images.

3 Model

In this paper we examine and compare the relative success of linear GANs, convolutional GANs (DCGANs), and LSTM GANs. Each model type consists of a discriminator and a generator model of the same structure. In all of the models, the generator starts from an input array of random numbers and outputs a flattened array representing a MNIST image. The discriminator starts from a flattened input array representing a MNIST image, and outputs a single value representing the likelihood of the input image being a part of the MNIST data set. The generator model outputs into a tanh function, while the discriminator model outputs into a sigmoid function. Each model type was further tested with multiple hyperparameter configurations.

All models contained dropout. All models besides the linear model contained gradient clipping. L2 norm regularization was not used as the regularization was either unnecessary or did not stop exploding gradients in different models.

Linear
The linear model consists of a multilayer perceptron generator and discriminator. Each successive layer of the generator increases in size, while each susccessive layer of the discriminator correspondingly decreases. The model can be seen with an example architecture in Figure 1a.

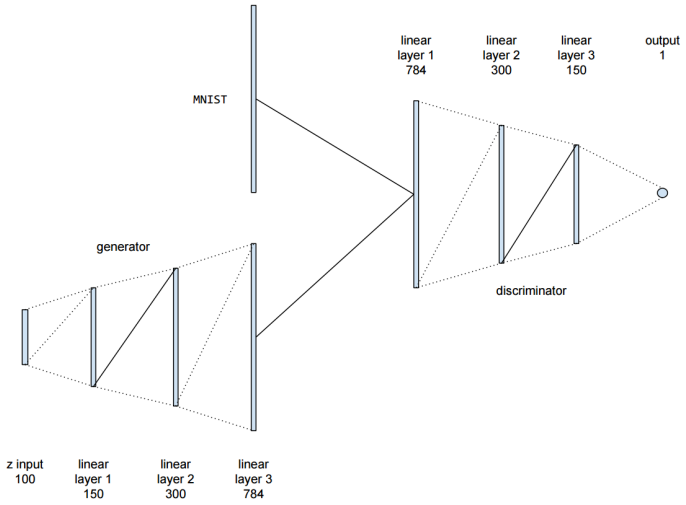
Convolutional
The convolutional model consists of a series of deconvolutional layers for the generator and convolutional layers for the discriminator. We tested the model both with and without pooling layers in between convolutional layers. The model layers are capped with a fully connected linear layer. The non-pooling model can be seen with an example architecture in Figure 1b, and the pooling model can be seen with an example architecture in Figure 1c.

LSTM
The LSTM model consists of multiple LSTM layers for both the generator and discriminator. As with the convolutional model, both the generator and discriminator are capped with a fully connected linear layer. The model can be seen with an example architecture in Figure 1d.

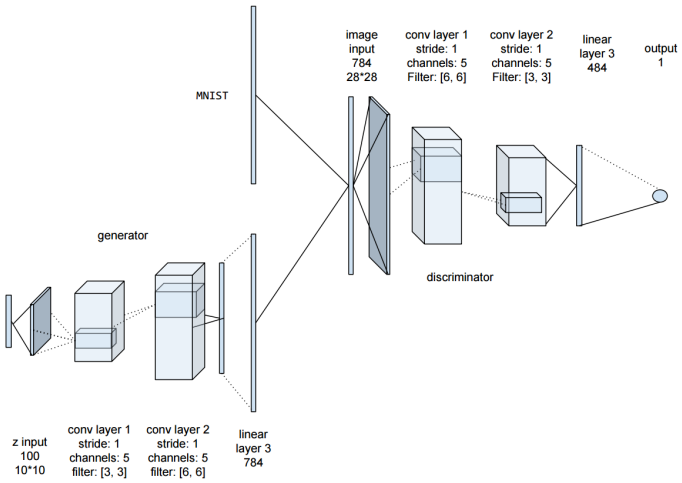
4 Experiment

4.1 Data

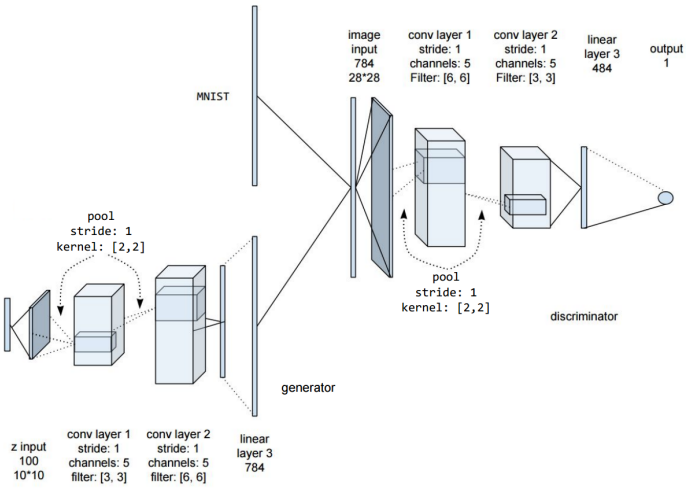
The training task was accomplished using the MNIST data set[15]. The MNIST data set contains 80000 flattened arrays split into training (60000), validation (10000), and test (10000) sets. Each array represents a 28*28 handdrawn gray scale image. Most models process the flattened data. For the DCGANs, the data was transformed back into 28*28 images.



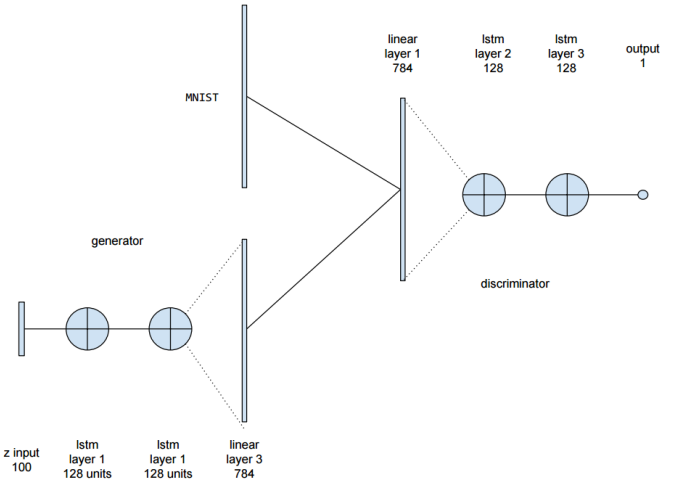
(a) Linear Model.



(b) DCGAN.



(c) DCGAN with Pooling.



(d) LSTM Model.

Figure 1: The different types of models examined in this paper, each with an example architecture.

4.2 Setup

The model was built using Tensorflow 1.0. Models were trained both locally on an ASUS ROG GL551J without CUDA, and, when available, remotely on an NVIDIA Titan X server cluster with CUDA. Convolution models used the tensorflow implementation found in *tf.nn.conv2d*; LSTM models used the tensorflow implementation found in *tf.contrib.rnn.LSTMCell*. Models were trained with backpropagation using the Adam optimizer on the MNIST training data set.

Unchanging network parameters are displayed in Table 1. For each GAN type, we attempted to rigorously test numerous architectures to find the optimal hyperparameter configuration. The relevant hyperparameters for each architecture type are displayed in Table 2. Because of a lack of computational power and a strict deadline, finding the optimal hyperparameter configuration proved difficult for each of the models we tested.

We quantitatively compare training time and relative loss between discriminator and generator; and we qualitatively compare sample outputs at the end of the training period. We continued training a few models past the maximum training epochs to observe the long term potential of the generator.

Table 1: Unchanged Network Parameters

Parameter	Value
Image Height	28
Image Width	28
Image Size	784
Batch Size	256
Z Input Size	100
Learning Rate	1e-4
Dropout Probability	0.5
Max Gradient Norm	5.0
Layers	2
Convolutional Output Channels	5
Pooling Kernel Size	[2, 2]
Pooling Stride	[1, 1]
Max Training Epochs	500

Table 2: Hyperparameters for each Model Type

Linear Model	Conv Model	LSTM Model
Layer One Size	Filter Shape	Number of Units
Layer Two Size	Stride	
	Use Pooling	

4.3 Results

4.3.1 Quantitative

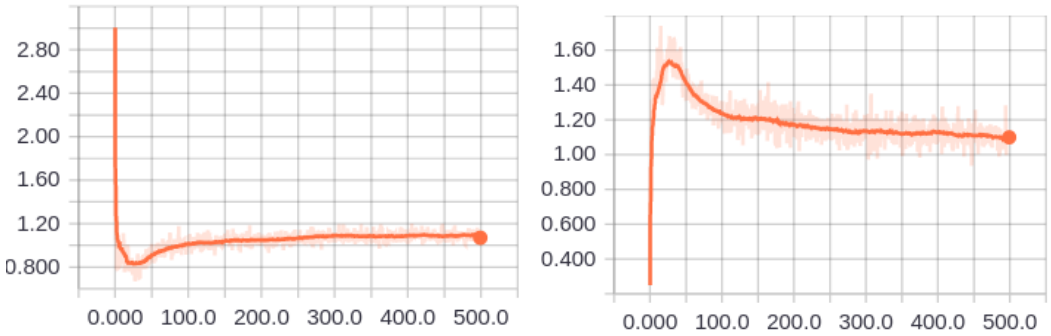
Of the model types that we examined, we found that the linear model trained the fastest but also saturated rapidly and benefited least from additional training. Generator loss began to flatten by step 300. Discriminator loss flattened even earlier. The linear loss can be seen in Figure 2. Of the numerous linear models we examined (various combinations of layer sizes), all but one suffered from exploding weights that became NaNs. In this sense, the linear models were the most difficult to train and correctly optimize.

As a class, the DCGANS without pooling layers continued to improve over thousands of steps. We therefore believe they have the potential to create more realistic outputs than the linear models. However, these models took much longer to train than linear models, both in terms of number of steps and in terms of time per step. We found that loss reduction was highly dependent on stride more than any other hyperparameter - smaller strides performed better in the short and long term. DCGAN loss can be seen in Figure 3.

We did not expect LSTM GANS or DCGANs with pooling to be particularly good at the MNIST generation task.

LSTMs excel at sequential data with deep nested patterns. While symbols are inherently patterned when drawn, the data does not lend itself to sequential examination. LSTMs take flattened data, but image patterns can appear horizontally, vertically, diagonally, and more. While some research has been done on constructing LSTM models for image generation, these require more complex preprocessing[8][6] that, in the interest of time, we did not build in.

Pooling layers, on the other hand, can be useful for larger image models by downsampling and removing unimportant details. However, because the MNIST images are already fairly small, we suspect pooling layers will actually remove important details and make the final output less accurate. The quantitative data for LSTMs and DCGANs with pooling can be seen in Figures 4 and 5 respectively. The LSTMs did not show any consistent improvement. Though the pooling DCGANs seem to begin to improve, further observation of the model over time (not shown) showed that this was not the case.



(a) Layer sizes: 300, 150. D-Loss.

(b) Layer sizes: 150, 300. G-Loss.

Figure 2: Comparative loss in the linear architecture. Note that these models began to improve as early as step 50. However, they flatten early as well.

4.3.2 Qualitative

We show for each model the final MNIST output.

As we expected given the quantitative loss, the linear models performed the best with the sharpest outputs. In all but a few cases, the numbers are clearly legible. The outputs from the linear model seem to accurately reflect the MNIST control images. The DCGAN models seemed to have more long term potential and stability - more of these models produced some kind of readable output data, and over a longer training time improved visibly (not shown). Further, unlike in the linear case, the DCGANs did not result in exploding gradients. The DCGANs were highly dependent on their hyperparameters, making optimization difficult.

The best DCGAN was the one with the smallest stride and filter. We suspect that success of smaller strides and filters is due to the small size of the MNIST images - with larger filter sizes, the data is likely compressed too much and important information is lost.

As expected, the LSTM and DCGAN with pooling models performed fairly poorly throughout.

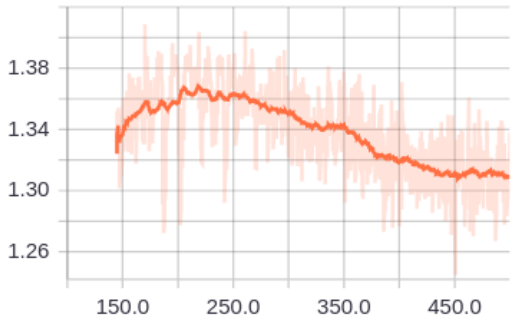
The qualitative data can be seen in Figure 6.

5 Conclusion and Future Work

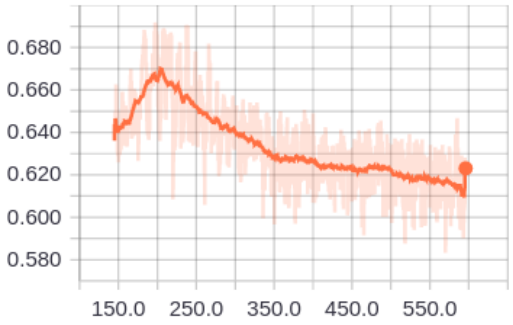
In this paper we developed and tested a series of generative adversarial networks. We found that linear models performed the best and trained the fastest, but were extremely sensitive and would often suffer from exploding gradients. DCGANs took longer to train but were the next best class of model. DCGANs also proved to be far more stable than any of the other model types. DCGANs with pooling performed slightly better than LSTM models, but both overall produced results that were decidedly less than satisfactory.

The models that we implemented were all of a single class. We did not experiment with the type and number of hidden layers. Given more time and computational ability, we propose experimenting with various kinds of hidden layers in the architectures. We also further propose more rigorously examining all possible hyperparameters for each architecture, including some parameters that we left as constants for the purposes of this paper.

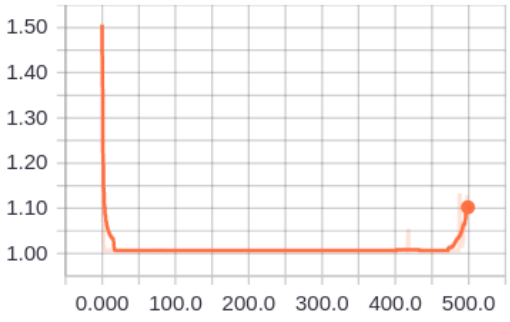
Finally, the models that we implemented were not state of the art. We already mentioned improved LSTM models, such as the DRAW algorithm[8]. There are also other DCGAN implementations, such as the softmax DCGAN[16]. In order to improve this experiment, the comparisons made in this paper should be done with the improved state of the art architectures.



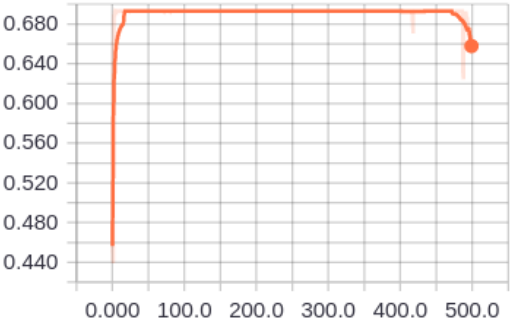
(a) Stride: 1. Filter: [4,4], [2,2]. D-Loss.



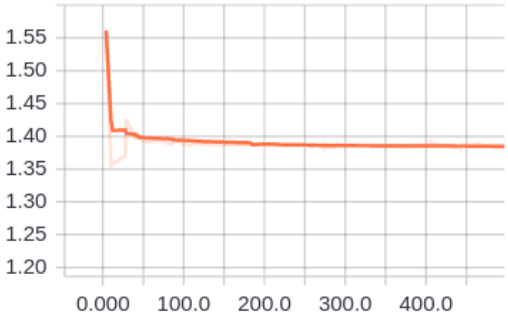
(b) Stride: 1. Filter: [2,2], [4,4]. G-Loss.



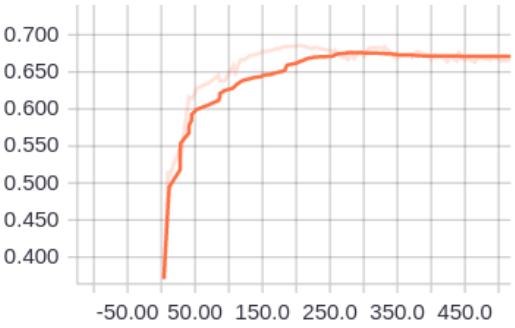
(c) Stride: 1. Filter: [6,6], [3,3]. D-Loss.



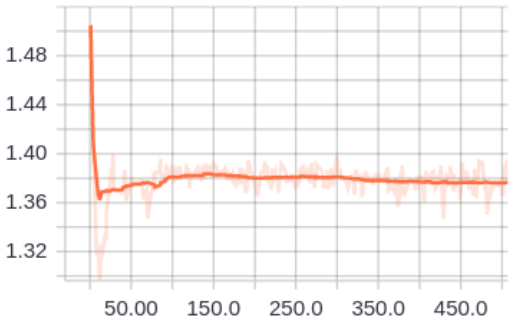
(d) Stride: 1. Filter: [3,3], [6,6]. G-Loss.



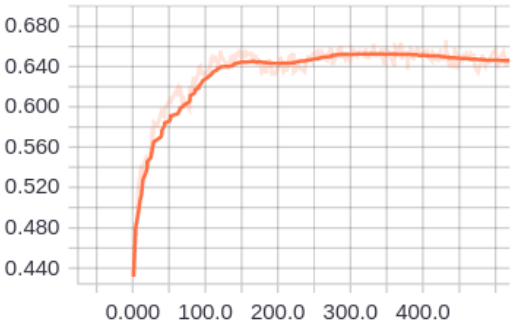
(e) Stride: 2. Filter: [6,6], [3,3]. D-Loss.



(f) Stride: 2. Filter: [3,3], [6,6]. G-Loss.

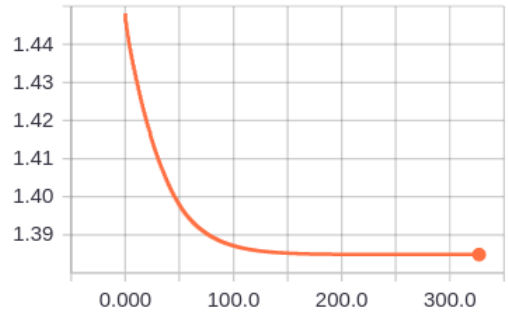


(g) Stride: 2. Filter: [10,10], [5,5]. D-Loss.

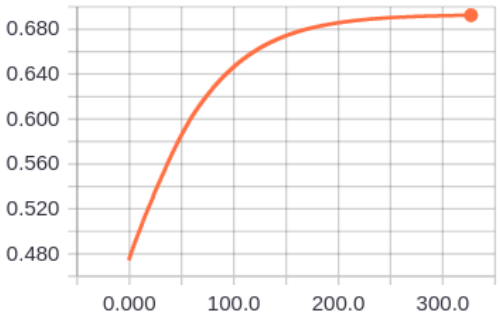


(h) Stride: 2. Filter: [5,5], [10,10]. G-Loss.

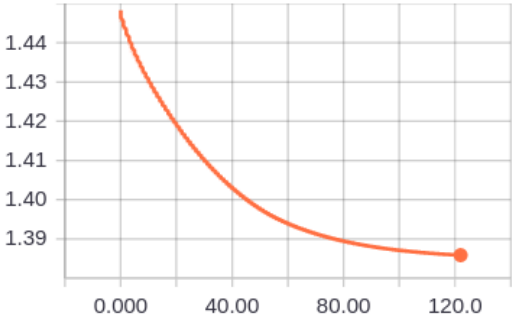
Figure 3: Discriminator loss compared to generator loss in the DCGAN no-pooling architectures. Notice that models with a stride of one ended up performing much better than those with a larger stride. Further, note that even the best DCGAN architecture (fig. A-B) did not begin to improve until after step 200.



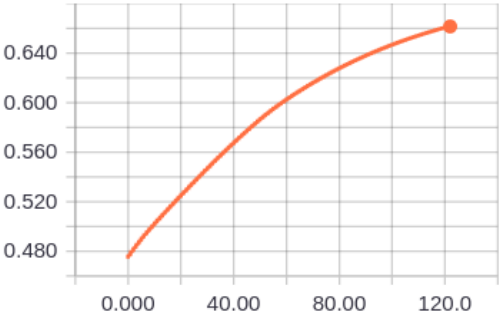
(a) 128 units per layer. D-Loss.



(b) 128 units per layer. G-Loss.



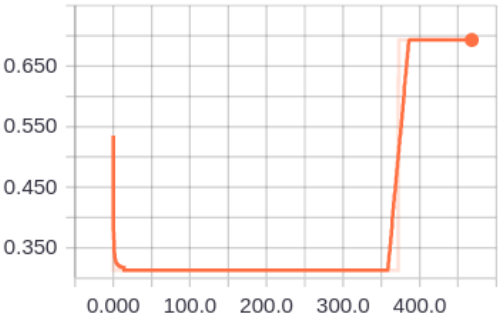
(c) 256 units per layer. D-Loss.



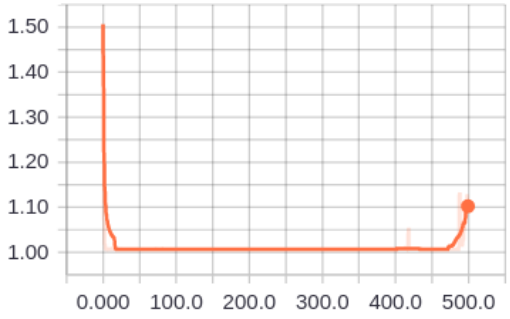
(d) 256 units per layer. G-Loss.

Figure 4: Comparative loss in the LSTM architecture. Note that these models don't seem to improve at all. We stopped training the model early after observing that the gradients had dropped to 0.

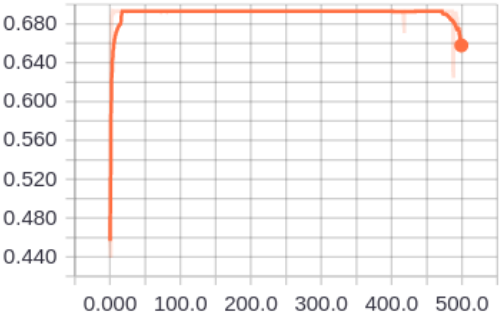
(a) Stride: 1. Filter: [4,4], [2,2]. D-Loss.



(b) Stride: 1. Filter: [2,2], [4,4]. G-Loss.

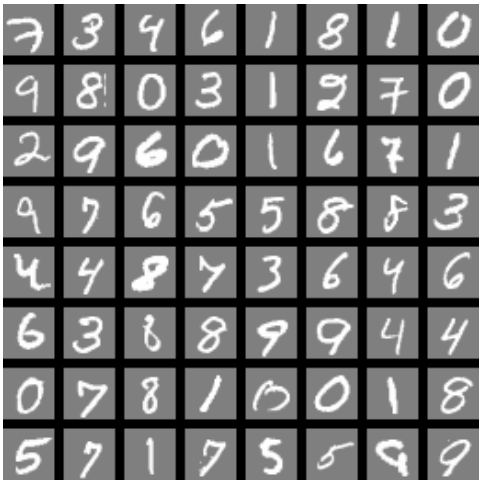


(c) Stride: 1. Filter: [6,6], [3,3]. D-Loss.



(d) Stride: 1. Filter: [3,3], [6,6]. G-Loss.

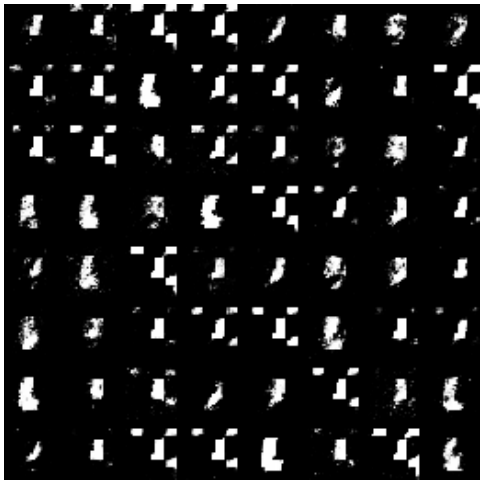
Figure 5: Comparative loss in the DCGAN pooling architectures.



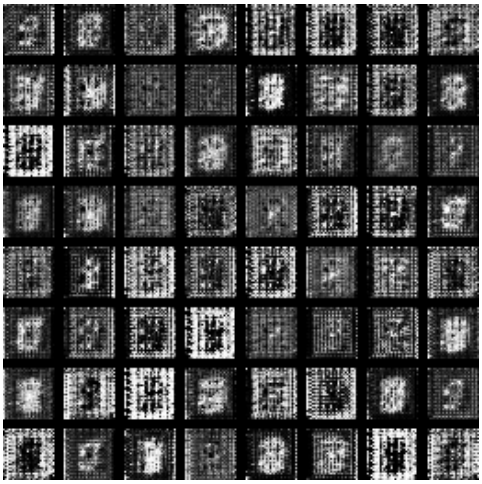
(a) MNIST

(b) Linear model. Layer sizes: 150, 300.

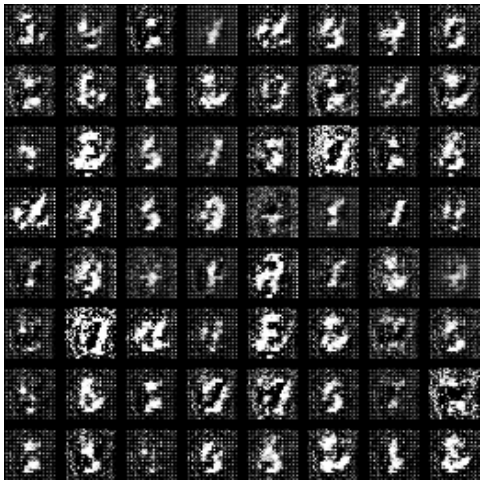
(c) DCGAN. Stride: 1. Filters: [2,2], [4,4].



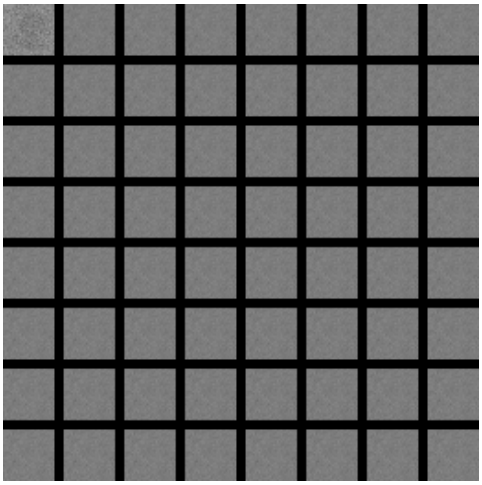
(d) DCGAN. Stride: 1. Filters: [3,3], [6,6].



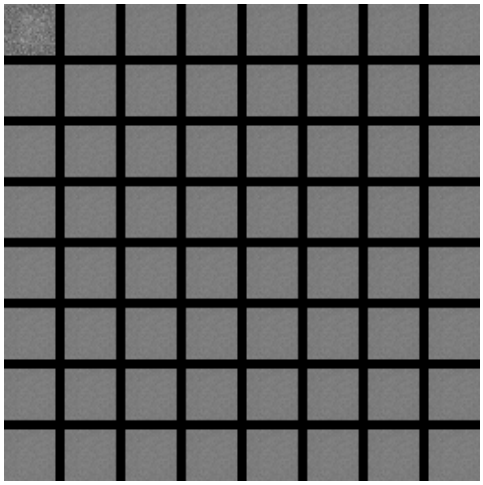
(e) DCGAN. Stride: 2. Filters: [3,3], [6,6].



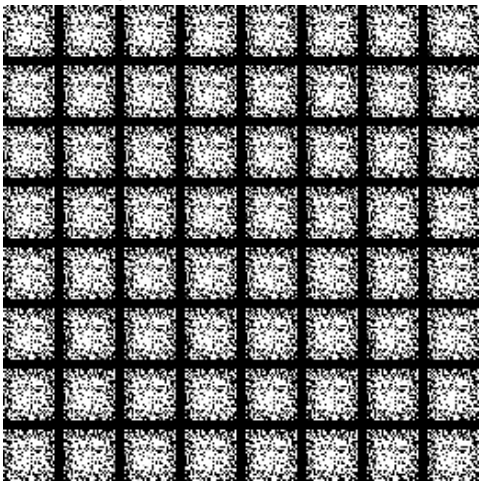
(f) DCGAN. Stride: 2. Filters: [5,5], [10,10].



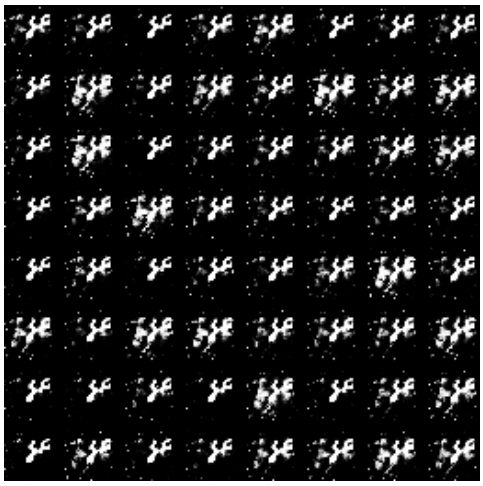
(g) LSTM. Num units: 128.



(h) LSTM. Num units: 256.



(i) Pooling DCGAN. Stride: 1. Filter: [2,2], [4,4].



(j) Pooling DCGAN. Stride: 1. Filter: [3,3], [6,6].

Figure 6: Qualitative MNIST output comparison across all tested architectures. The linear model had the sharpest outputs, followed by the DCGANs, then the DCGANs with pooling, and finally the LSTM architectures.

Acknowledgments

Thank you Bernardo for an amazing class, and for a great experience for my entire two terms here. I learned more than I could have hoped during my time at Worcester, and I look forward to applying what I have learned both over the summer and well into the future!

References

[1] Ilkay Ulusoy and Christopher M Bishop. Generative versus discriminative methods for object recognition. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 258–265. IEEE, 2005.

[2] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2:841–848, 2002.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[5] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546, 2015.

[6] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935, 2015.

[7] Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015.

[8] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[9] Xiaojin Zhu. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, 2(3):4, 2006.

[10] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[13] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[14] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[15] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[16] Min Lin. Softmax gan. *arXiv preprint arXiv:1704.06191*, 2017.

[17] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.