Project 2: DCGAN — DTSC 740 Deep Learning

Joanna Pedretti — December 2024

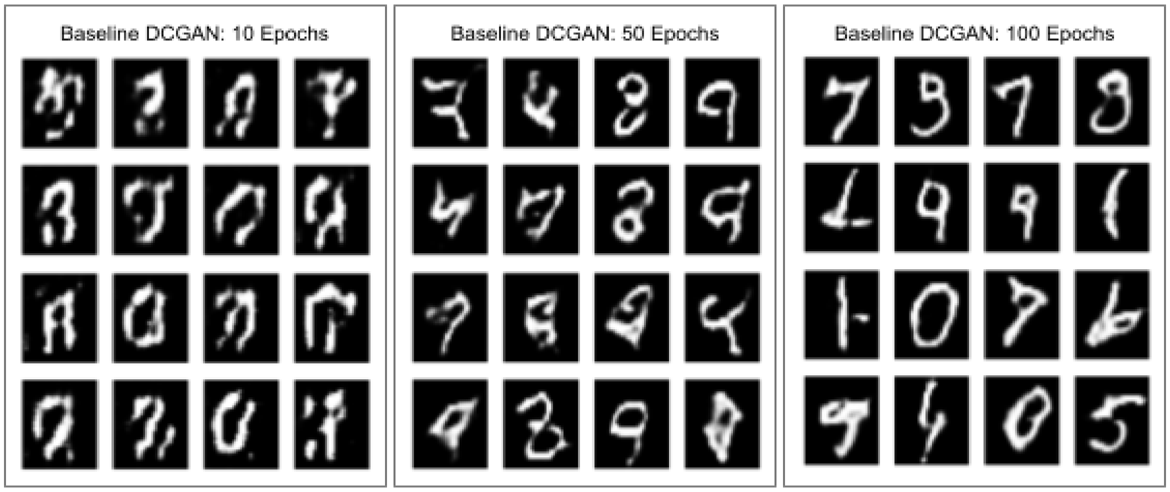
1 Introduction

Deep convolutional generative adversarial networks (DCGAN) can be used to generate images that replicate their training data. Unsupervised learning tasks can be performed with GANs [1] in order to train the model with no ground truth of what the images should look like, aside from what it learns during training. A generative adversarial network is made up of two distinct parts, the generator and the discriminator. The discriminator takes lead in making modifications to the model when calculating error and updating weights, and the generator learns based on what information is provided by the discriminator [2]. It will take inputs of both the real images and the synthetic images created by the generator, and attempt to determine if they are real or fake. The better the quality of the fake images created by the generator, the harder it will be for the discriminator to classify them accurately. Therefore the training process involves finding a balanced set of parameters which both maximize the classification accuracy of the discriminator and also maximize the quality of the fake images from the generator [2]. DCGANs specifically make use of deep convolutional layers to tune both parts of the network and optimize pattern recognition and feature extraction. These convolutional networks are efficient at feature extraction which has been proven by much research in CNNs, therefore utilizing convolution layers to perform the feature mapping provides an advantage for the generator to understand and replicate the patterns it is learning.

2 Baseline DCGAN

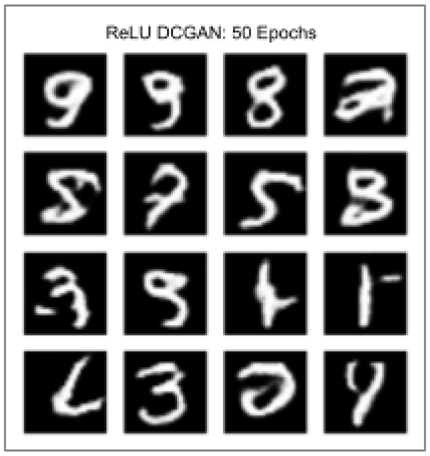
The basic DCGAN architecture uses Leaky ReLU activation functions. Leaky ReLU is a slightly different version of the ReLU activation, in which the value of the function is equal to max(alpha*x, x) [3]. Meaning that the value chosen will be either alpha*x or x, whichever number is higher. If the value of alpha is set to zero then the function works as a regular ReLU function, otherwise it can be modified by choosing different values of alpha to prevent the function from becoming zero for certain value inputs. The purpose of this activation function is to help avoid the dying ReLU problem; where values being set to zero during training could eventually prevent the neural network from continuing learning [3]. Using the leaky ReLU function could be beneficial for this deep learning network since it will train over many epochs and there is a chance of the dying ReLU problem occurring.

In this experiment, the same architecture is tested with a varying number of epochs to see the difference in the learning of the model over time. Each time, the runtime for the code was restarted so that the model did not have any previous knowledge of the dataset. This helps avoid any transfer learning that could have occurred by running the same model multiple times in the same code, and instead provides a more fair comparison between the learning abilities over different training times. At first it was trained over 10 epochs, and it is evident by the diagram that the model needed much more time to learn how to generate realistic looking fake images. It was barely able to replicate the handwriting of any numbers, and most of the outputs look the same which shows low diversity of the generator outputs. On the next attempt with 50 epochs, the numbers begin to show a bit more prominently as the model has more time to understand different handwriting patterns and learn to generate similar results. They are still nowhere near perfect but it is definitely a large margin of improvement compared to 10 epochs. Finally for the test with 100 epochs, it is evident that the quality of these output images are the most realistic out of the three tests. The lines are drawn straighter and more clearly, and there is also a more diverse representation of numbers visible. The effectiveness of longer training times is demonstrated in this case because it is very obvious to see the clarity and diversity of the outputs increasing over each of the tests. The baseline DCGAN model was able to learn the patterns of the training dataset in a more complex way over time, which allowed it to replicate these images with decently believable fakes.


Baseline DCGAN: 10 Epochs
Baseline DCGAN: 50 Epochs
Baseline DCGAN: 100 Epochs

3 ReLU DCGAN

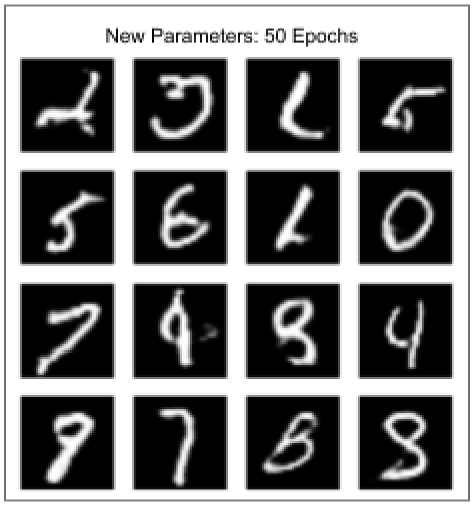
In a second architecture, the new DCGAN will use ReLU activation in the generator for all layers except for the output layer. The output layer uses a Tanh activation function which is the same as in the baseline DCGAN experiments from section 2. Everything else about the model architecture is the same (including the chosen hyperparameters), and the only change in this section is the activation function. Since leaky ReLU was removed there is a possibility of this model facing the dying ReLU problem if it is run over too many epochs and the inputs drop to zero, but it does not seem like that occurred in this situation. This model was run for 50 epochs, and compared to the 50 epoch output of the baseline DCGAN in section 2, it is clear that this model was able to generate much clearer and more diverse fake images. The lines are much more clear and mostly all of the outputs look like a hand written number. The output from this model at 50 epochs is arguably also better than the baseline model running for 100 epochs. The training time of this model took about 10 minutes using the T4 GPU on Google Colaboratory, whereas the baseline model took around 15 minutes using the same processor. Overall it seems that in both training time and learning ability, the ReLU DCGAN model outperformed the baseline model. Since it outperformed the baseline model in speed and clarity of the synthetic images, it is clear that the ReLU activation function is a better fit for this type of task.



4 Hyperparameter Tuning

Many different hyperparameters could be tuned for this model; such as the dimensionality of the noise vector, the batch size and learning rate, and the momentum terms. New hyperparameter values will be chosen for this section of the experiment to attempt to increase performance of the model. The noise vector acts as an input to the generator which helps create diverse and high quality images after training. If the dimensions of the noise vector are too low, the model may not learn complex enough patterns to generate high quality images. If the dimensions are too high, the computations of the generator will become too complex and make training difficult for the model. Hyperparameters such as the batch size and learning rate also affect the model's ability to learn on the training data. They affect performance by determining the stability of the model and how quickly it will converge on the ideal outputs. Finally, the moment terms are useful for the chosen optimizer to help find smooth gradients and quick convergence. If the momentum is too low the training will be slow and could possibly get stuck in local minimums instead of the global minimum. If it is too high it could also miss the global minimum by skipping over it by mistake while training too quickly.

In the previous models, the original dimensionality of the noise vector was [1, 100]. In order to attempt to increase the quality of the images, the dimensionality will now be decreased to [1, 50]. This may cause the model to train faster, but the realisticness of the fake images should still be improved. To balance it out, a batch size of 128 will be used, which is half the batch size of the previous models. This should help stabilize the training, along with a new learning rate of 5e-5 which could help with the faster moving model. Finally, a momentum term of 0.5 is used which is slightly lower than the default 0.9 used in the adam optimizer. A lower momentum term may also help balance stability and could lead to faster convergence on the optimal solution. This model uses the ReLU activation function since that has proven to perform better from the previous sections, and it is run over 50 epochs. For the results, the output below indicates that while the lines drawn are very clear and concise, the numbers themselves don't really look right. Many of them look like scribbles or random shapes instead of a hand written number, so although the quality of the images is very high, the realisticness of the fake numbers is not as accurate as some of the previous experiments.



5 Discussion and Conclusion

Overall, it is clear that the DCGAN with 50 epochs and ReLU activation function performs the best generation of fake images with the given architectures. By tuning the hyperparameters differently, the results became more clear but less accurate in some ways. Therefore the ReLU DCGAN with the original hyperparameters seems to be the best performing model out of the three tested in this experiment. The fact that it was able to generate clearer images in 50 epochs than the baseline model generated in 100 epochs shows that the activation function has a large impact on the accuracy of generative adversarial network models. When using the ReLU activation function and choosing other hyperparameters, the images still retained their clarity but the diversity and accuracy of the way the numbers look seems to be diminished. This could be attributed to the model learning how to replicate straight lines, but not fully learning the proper shapes that the numbers should be drawn in. If further testing was done and the hyperparameters were tuned more optimally, it has the potential to eventually generate extremely realistic fake images. It could also improve by training for more than 50 epochs, depending on if these parameters are well suited for the task. In conclusion, based on the experiments performed in this paper, a decently well performing architecture for DCGANs using the MNIST dataset has been determined. Future works would involve more hyperparameter optimization to further improve the generative output.

6 References

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *Advances in Neural Information Processing Systems, 27*, 2672–2680. https://doi.org/10.48550/arXiv.1406.2661

[2] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Deep learning for visual understanding: Part 2. *IEEE Signal Processing Magazine, 35*(4), 107–117. https://doi.org/10.1109/MSP.2018.2821461

[3] Educative. (n.d.). What is Leaky ReLU? *Educative.io.* Retrieved December 13, 2024, from https://www.educative.io/answers/what-is-leaky-relu