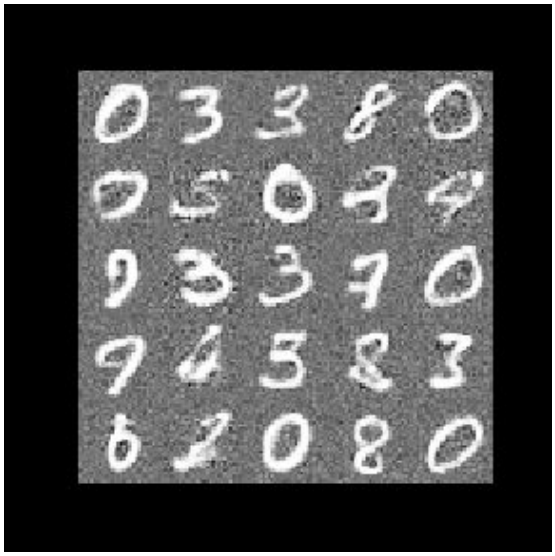
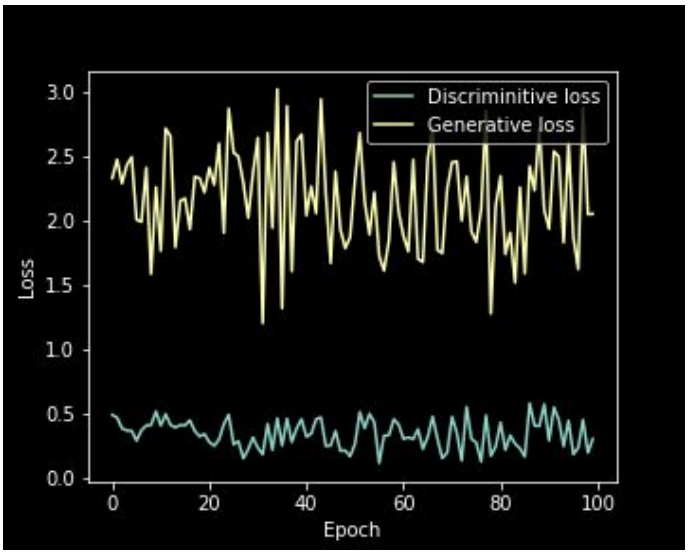
Caleb Ralphs
4/17/2019

Individual GAN MNIST Assignment

Final Image Generation and Loss



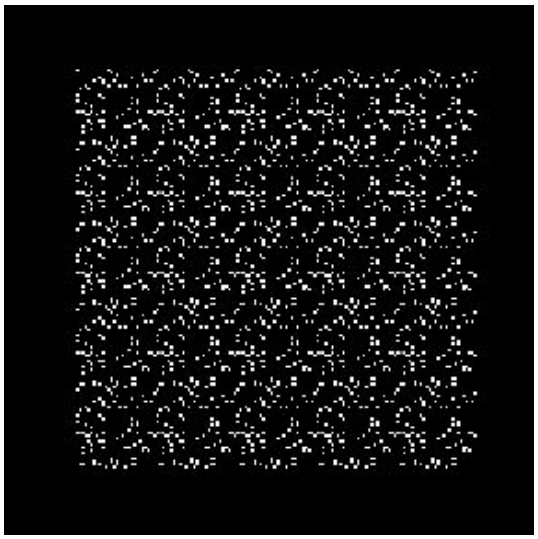


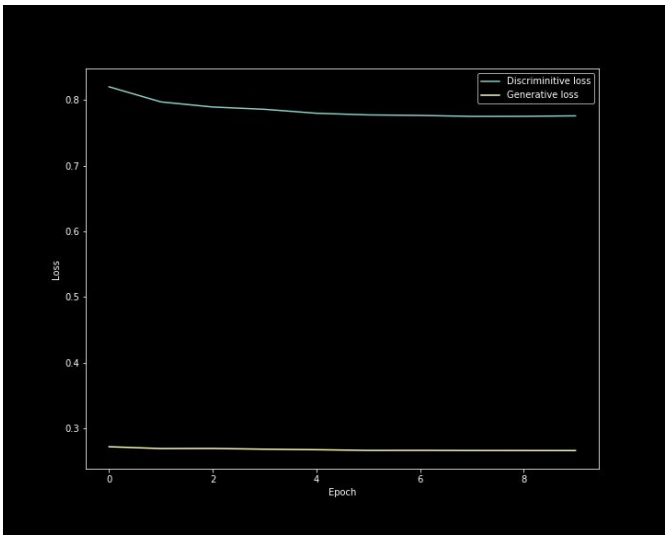
Special Skills

- Avoiding sparse gradients: replacing the ReLU layer with a Leaky ReLU layer, reducing the amount of data lost in raw ReLU activation.
- Adding layers: adding layers to generator and discriminator to increase complexity of model, either increasing or decreasing the spatial dimensions.
- Structuring complexity of generator and discriminator: increasing spatial dimensions of the generator while going from more complex to less complex for the discriminator.
- Sampling from a gaussian distribution: replaces the linear interpolation of data with a spherical linear interpolation, by way of a gaussian or normal, distribution, prevents the model from diverging to a previous distribution.
- Label smoothing: make the labels the noisy for the discriminator through replacing true labels with some noise based upon some small distribution around the true label, training the discriminator to be less likely to be “tricked” in future iterations of training.

Experiments

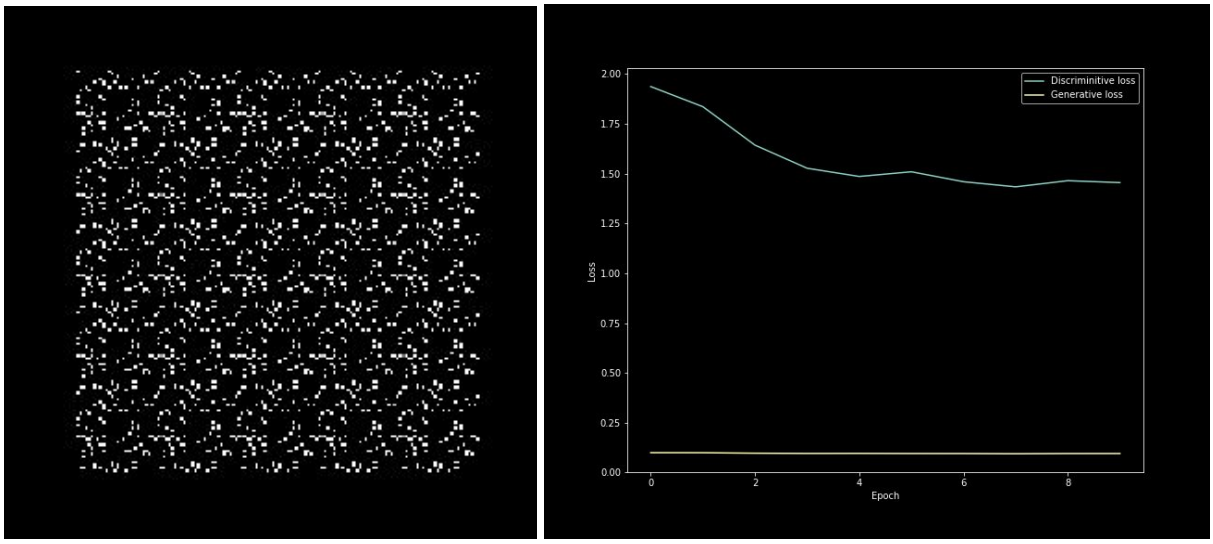
- Baseline (Epochs = 10):
    - Model Structure:
        - Generator:
          Dense(256)
          Activation(relu)
          Dense(784, tanh)
        - Discriminator:
          Dense(256)
          Activation(relu)
          Dense(1, sigmoid)





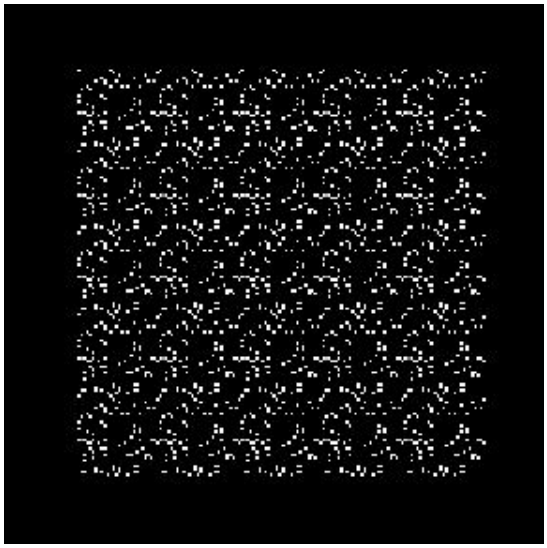
Random images and poor loss performance (i.e not much change in loss for either model)

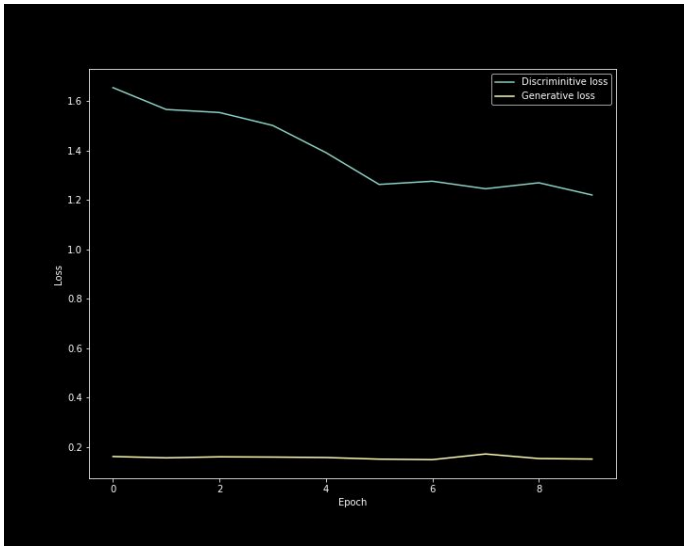
- Leaky ReLU Layer Addition to Generator and Discriminator (Epochs = 10):
    - “The stability of the GAN game suffers if you have sparse gradients” [1]. I changed the ReLU to Leaky ReLU with an another layer. Additionally, an article suggests going from lower to higher complexity layers with generator, and the opposite for the discriminator [2].
    - Model Structure:
        - Generator:
          Dense(256)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(784, tanh)
        - Discriminator:
          Dense(256)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(1, sigmoid)



The images still look completely random. There’s a better looking loss for the discriminator, but the generator is still very poor.

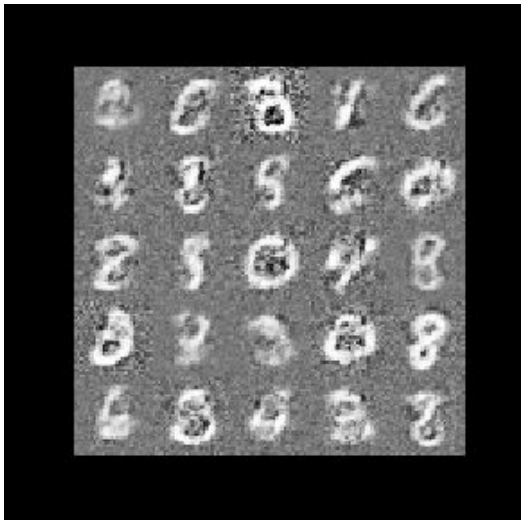
- Another Leaky ReLU Layer addition to Generator and Discriminator (Epochs = 10):
    - The Leaky ReLU layer on the discriminator made for better loss, but the generator is still lacking, so I am going to see if adding another layer with more nodes will help.
    - Model Structure:
        - Generator:
          Dense(256)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(1024)
          LeakyReLU(.2)
          Dense(784, tanh)
        - Discriminator:
          Dense(1024)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(1, sigmoid)

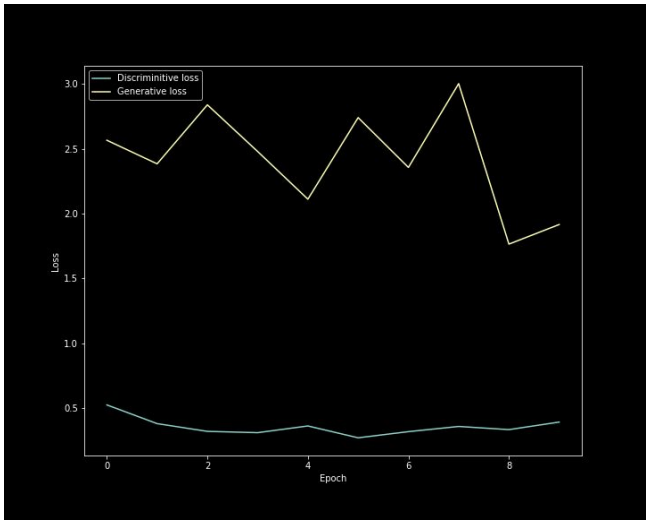




Images still look very random. The loss has gotten even better for the discriminator, but still looks poor for the generator.

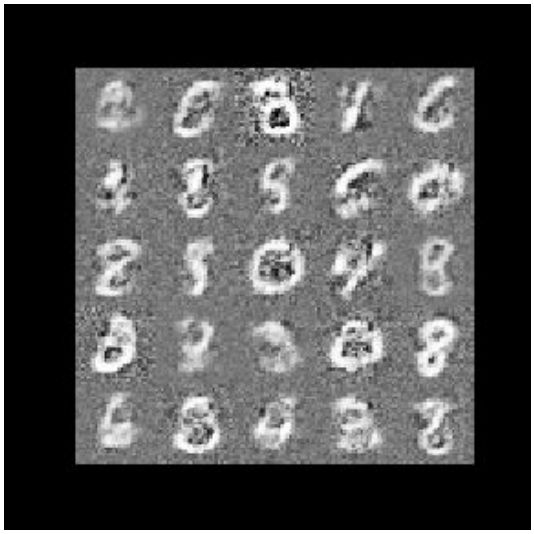
- Sampling from a normal distribution and adding dropout (Epochs = 10):
    - “Sample from a gaussian distribution” [1]. Trying dropout to provide additional noise for the model. Additionally, I specified the input dimension for the first layer of the discriminator to be that of the last layer of the generator.
    - Model Structure:
        - Generator:
          Dense(256, RandomNormal)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(1024)
          LeakyReLU(.2)
          Dense(784, tanh)
        - Discriminator:
          Dense(1024, input = 784, RandomNormal)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(512)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(1, sigmoid)

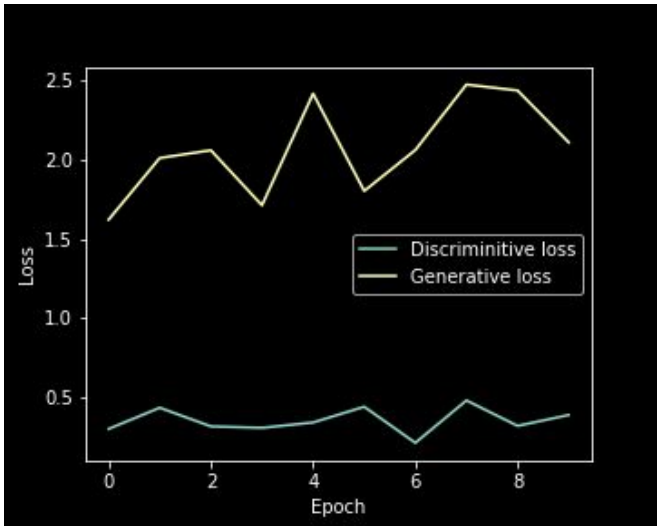




Using a normal (gaussian) distribution and adding the dropout seemed to really help the model. Even after 10 epochs, you can see the structure of the digits in the generated image. Additionally, the loss looks more normal, fluctuating up and down as each model is trained better.

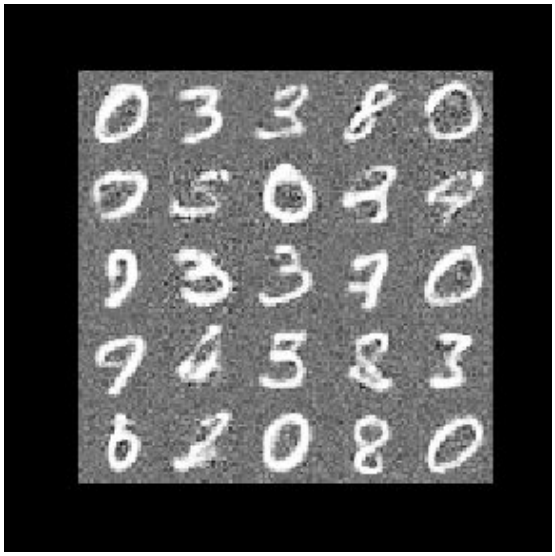
- Adding label smoothing in the training function (Epochs = 10):
    - “Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example)” [1].
    - Model Structure (same as previous experiment):
        - Generator:
          Dense(256, RandomNormal)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(1024)
          LeakyReLU(.2)
          Dense(784, tanh)
        - Discriminator:
          Dense(1024, input = 784, RandomNormal)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(512)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(1, sigmoid)

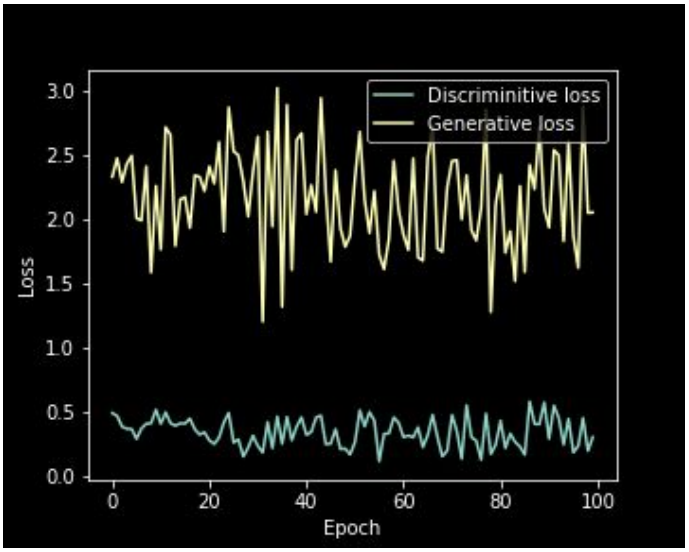




Very similar generative image results to the previous experiment. The loss seems to be showing more variance.

- Final Run with one more layer in discriminator (Epochs = 100):
    - Similar model to the previous experiment, just adding one more layer with Leaky ReLU and Dropout to the discriminator, so both models now have the same amount of layers.
    - Model Structure:
        - Generator:
          Dense(256, RandomNormal)
          LeakyReLU(.2)
          Dense(512)
          LeakyReLU(.2)
          Dense(1024)
          LeakyReLU(.2)
          Dense(784, tanh)
        - Discriminator:
          Dense(1024, input = 784, RandomNormal)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(512)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(256)
          LeakyReLU(.2)
          Dropout(.4)
          Dense(1, sigmoid)





The increased number of epochs from 20 to 100 made a significant difference on the quality of the images made by the generated. The images are still kind of noisy, but that is understandable given the noise that I added to the model. I would've gone for 200 epochs like some other implementations did, but I could not get WPI's turing tasks to work, so I was limited to my local machine.

References

1. https://github.com/soumith/ganhacks
2. https://medium.com/@UdacityINDIA/everything-you-need-to-know-about-generative-adversarial-networks-7be6773f46f4
3. https://github.com/Zackory/Keras-MNIST-GAN