Image Generation using Generative Adversarial Networks

Rawal Khirodkar  Maulik Shah  Shantanu Thakoor  Nikhil Vyas
130050014  13D100004  13D100003  130050023

Problem Statement

Setting:
Deep Convolutional Generative Adversarial Neural Networks (DCGAN)[1] is trained on a set of images (from a specific domain, eg. faces) and then images similar to the training set are generated using Generative Network.

Input:
100 dimensional vector (Z) of real values sampled randomly uniformly from [-1,1]

Output:
64 x 64 RGB image

Training:
Generative and Discriminative Networks are trained together on images belonging to a specific domain (say faces!).

- Generator Network generates images similar to the images from the training data.
- Discriminator Network's job is to distinguish between images generated by generator and between original images from training data.

Image generation from Z:
The vector Z acts as a starting point for the Generative network which converts Z into an 64 x 64 x 3 image gradually using forward propagation.
Model Structure:

![Generative Network Model Diagram]

Fig 1. Generative Network Model [1]

Major Drawbacks:

The output images generated by Generative Adversarial Neural Network (GAN) follows a continuous distribution.

- For example, if the network outputs images of a face, then an image of person wearing spectacles and that of person not wearing one should be separated significantly apart in the feature space of output images.
- However, we observed that the images generated by a standard implementation of GAN as in [1] also produced images that were in the interpolated region between these two images.
- Consider these three images generated by the network:

  ![Sample image without spectacle]

  Sample image without spectacle

  ![Sample image with half a spectacle (Not acceptable!)]

  Sample image with half a spectacle (Not acceptable!)

  ![Sample image with spectacles.]

  Sample image with spectacles.
• As can be seen in the outputs above the network generates all three types of images, with specs, without specs and something with just the outline of a spec.
• Clearly, only two out of three are desired when we are trying to generate realistic images. (Note: The training data has no instance of a face with half a spectacle)
• The problem here as we interpreted it was that the network learns to generate output in a continuous space (as a gross over simplification, say, everything between 0 to 1. 0 representing a face with no specs and 1 representing a face wearing specs), while the desired output belongs to discrete space (continuing our previous example, say, just 0 and 1 or regions within their small neighbourhoods)

**Main Goal:**

In this project, we aim to try and explore possible solutions to “discretize” the output of GAN.

**Simplification of Problem Statement**

As soon as we started working on some basic ideas, we realised that the domain of output images (Faces) was too complex (i.e discontinuity between two faces is not well defined).

Furthermore, GANs have no metric for measure improvements (like accuracy or precision). This compounded with the fact that feature vector spaces of faces is a very rich domain to visualize bolstered our decision to try simpler images.

**Simpler Domain:**

The domain consists of circles and triangles -- a domain where the improvements are obvious to naked eye.

The dataset consists of two kinds of images.

![Type 1: 64 x 64 blue bordered square containing a red circle](image-url)
Training Data:

Training Data consists of **10,000 images**.

**5000 images of Type 1 and 5000 images of Type 2.**

The coordinates of circle and triangle are selected randomly inside the square.

Fig 2. Visualising a segment of Training Data (64 images)
It is apparent from Figure 3 that the problem of producing continuous output still persists even with our simpler domain of circles and triangles. There are instances where both circle and triangle are present together (for eg: Top row, fourth cell from left).

**Proposed Solutions**

**A] Approach: Use a Deeper/Wider Neural Network**

**Intuition:**

A function with a higher derivative is able to generate outputs with sharper changes. For example, consider the sigmoid function. The higher the magnitude of weight parameter $w$, the sharper is the curve.
The assumption is that a function with higher derivative can be better represented by a more discriminative network, one way to achieve this is by using deeper/wider network.
Hypothesis:

Since we wanted the function represented by the neural network to generate outputs in separate clusters given a continuous input, we hypothesized that we wanted the network to have a higher derivative.

Implementation:

We increased the number of layers in the generator network and the discriminator network and tried for several deep network architectures. We also tried to widen the network. The training was done for 24 epochs.

Results:

Either the generator failed to give any meaningful output (due to discriminator being too strong) or there was no appreciable difference.

![Fig 4. Output with a Deeper Generator and Discriminator Network.](image)

The change in architecture disturbed the stability of DCGAN training leading to bad results.

Conclusion:

As can be seen from the results above, this approach failed to generate the desired output. Maybe having a more powerful model is not the way to approach this problem.
B] Approach: Making Z discontinuous

Intuition:

In essence the function represented by the neural network is a mapping from a continuous input space (100 dimensional real space in this case) to a continuous output space. This is because each individual component of the neural network is a continuous function. Thus, making the input Z discontinuous can result in gaining discontinuity in the output as well.

Number of discontinuities introduced has to be proportional to number of clusters present in the training data. As we have two clusters (circles and triangles), we decided to break Z into two clusters.

Hypothesis:

If we could break the input space to form discrete clusters and only take in Z values from these restricted clusters, the corresponding outputs generated would also form discrete clusters.

Implementation:

Generate 100 dimensional Z vector such that every individual component is a real number uniformly sampled from -1 to 1.

- With probability 0.5, input vector Z is used unchanged for training.
- With probability 0.5, add +3 to each component of the input vector Z and further use it for training.

While testing, same transformations are applied to the vector Z.

This would generate input vectors from two discrete clusters:

- 100 dimensional hypercube with boundaries of each dimension at -1 and 1.
- 100 dimensional hypercube with boundaries of each dimension at 2 and 4.

Fig 5. Here Z represents one of the dimension of vector Z
Results:
The outputs were not separated as clusters of triangles and circles. This is probably because we are incrementing all the input features rather than changing their relative values.

Conclusion:
This approach failed to generate circles and triangle separately. Also we tried using further separated intervals for choosing each dimensional value of Z with no better results.

C] Approach: Changing half signals of Z

Intuition:
This would be the equivalent of our network learning certain features, such as triangles and circles, but then being activated only at certain inputs, independently of each other. This independence was what we mainly wanted to capture with this idea; half the signals of Z would be set to 0 to obtain outputs corresponding to a certain cluster, and the other half of Z being set to 0 would generate outputs of the other cluster.

Hypothesis:
Again, this is a discretization of the input space. Instead of the network having to learn values which would correspond to different outputs based on the magnitude of the inputs, it would learn different features which would get activated by certain parts of the input vector.

Implementation:
Generate 100 dimensional Z vector such that every individual component is a real number uniformly sampled from -1 to 1.

- With probability 0.5, multiply first 50 dimensions of Z by 1/2.
- With probability 0.5, multiply last 50 dimensions of Z by 1/2.

While testing, same transformations are applied to the vector Z.

Results:
Fig 6. None of the output images contain a circle and triangle together

**Conclusion:**

This approach was successfully able to discretize the output space into circles and triangles.

**Future Work**

Our intention when starting this project, was to provide a broad improvement of generative adversarial networks as a general framework. Although we have used a toy dataset, we believe we have highlighted the benefits of our approach well. A good next step for this, would be automatically deciding how the input space should be discretized (ie, the number of clusters in the dataset). Some kind of unsupervised learning on the training set as a preprocessing step before starting our DCGAN framework would help with this.

Another problem we were facing, is that since features were shared between the two clusters in approach C, they could not be very complex and a lot of the time they were only reproducing images from the dataset exactly. Future work to disincentivize the
generator from only duplicating images from within the training data would also improve
the performance of our framework.

**Codebase, Experimental Details, Effort**

**Code:**

**Language:** Python, Google TensorFlow

**Starting Codebase:** [Github Repository for GAN](https://github.com/username/Repository)

**Lines of Code:** About 200 lines

**Code is hosted at:** [Github Repository of our code](https://github.com/username/Repository)

**Experimental Platform:**

**Machine:** Nvidia GeForce GTX Titan X GPU

**Execution Time for one experiment:** ~30 minutes

**Effort:**

**Most challenging part:** Too much memory in the network (ie, the network only
duplicates images present in the dataset). The cited paper has some ideas to combat this,
but they were so complex they would be a project in itself :)

**Fraction of time spent on different parts of project:**
Most of the time was spent in coming up with different discretization strategies (60%)
Trying with different datasets took a lot of time, since some were so simple that the
standard framework could learn them well, and some were so complicated that our
method could not be adequately demonstrated better with them (eg. due to many
clusters) (20%)

Actually training the DCGAN also took time, since the training is very brittle on some
settings (ie too much fluctuation, no real learning) (20%)

**Memory Issues:**
We suffered from quicker Discriminator convergence in comparison to Generator. Thus, resulting in lack of learning and overall convergence.

Fraction of work done by team members:

Rawal Khirodkar (130050014) : 25%
Maulik Shah (13D100004) : 25%
Shantanu Thakoor (13D100003) : 25%
Nikhil Vyas (130050023) : 25%

References: