DEEP LEARNING PROJECT: GAN ARTWORK

A PREPRINT

Kevin Steele
Department of Electrical and Computer Engineering
University of Iowa
kevin-steele@uiowa.edu

Brain Fiegel
Department of Industrial and Systems Engineering
University of Iowa
brian-fiegel@uiowa.edu

Stjepan Fiolic
Department of Electrical and Computer Engineering
University of Iowa
stjepan-fiolic@uiowa.edu

May 7, 2021

ABSTRACT

The Iowa Neuroscience Institute has a desire for a landscape generation process by which DNA sequences are converted to landscape images. Our project is to train a Generative Adversarial Network (GAN) to accomplish this task. Specifically, we sought out the generative power of StackGAN as inspiration for image generation, using the general architecture to build a multi-stage network of increasing image fidelity. However, given that our data does not follow the same word-sentence input structure of the original paper, we were unable to use the architecture to the full extent. Our results are well enough given the difficulty of the task, but a simpler model might suit better for this possess. There is also room to grow with information extraction from the input sequence.

Keywords Deep Learning · GAN

1 Introduction

Generative Adversarial Networks (GANs) are an approach to generative modeling using deep learning methods. Generative models are an unsupervised machine learning task. This task involves learning and discovering input data patterns in a way that the model can generate new output data that has a close relationship to the original input data. For example, if we were to feed the model with images of the human face, the model would generate a new human face that is not in the original dataset. This may be an example most people are familiar with, but models can also convert natural language text descriptions into images. This type of model is called a Text-to-Image synthesis issue. We aim to achieve a similar goal by feeding a valid DNA sequence into our model and output a landscape image.

To understand GANs, the reader should first note that there are two sub-models of a GAN. The first is the generator model. The generator model takes a fixed-length random vector as input and generates a sample in that domain. A gaussian distribution is used to create a randomized vector. That vector then becomes the seed of the generative process. The training process will then produce latent (hidden) variables to input into the generator, the collection of which is referred to as a latent space. The second model of a GAN is the discriminator model. The discriminator model takes an example from the domain as input and predicts a binary class label. This label is either authentic or generated. The authentic examples come from the training dataset, in our case images of landscapes. The generated examples come from the output of the generator model.

The best way to distinguish the two model types is to think about them as a tug of war. One model is trying to build a landscape image (generator) great enough to fool a human (discriminator) into thinking that it is a real landscape. The generator must make the landscapes as close to real landscapes while the human will point out flaws in the generated image and send it back.

There is a reward system in place between the generator fooling the discriminator and the discriminator. This distinguishes the generated images from authentic images that come from the training set. If the generator is able to fool the discriminator, then the generator is rewarded by not having to rebuild its model. In this case, the discriminator is punished by having to rebuild its model. Else-wise, if the discriminator correctly distinguishes a generated image, the generator is punished by having to rebuild its model. The discriminator is rewarded by not having to rebuild its model. At convergence, the generator will have samples that are indistinguishable from the real data, or training set.

Conditional augmentation of a GAN is an extension where both the generator and discriminator take-in additional conditioning variables, in our case DNA. The generator forms images based on the additional conditioning variables and the discriminator verifies. First question should be how to convert a DNA element to a string that represents a feature in a landscape image. We plan on using an encoder to map DNA codons.[1]

Given our particular object of Text-to-Image synthesis, we will require a specialized GAN architecture. A few architectures have been proposed in this realm, particularly StackGAN [1] for text-to-image and CycleGAN [2] for unpaired image mapping. This process can be used for text-to-image as well [3], which involves using two generators, one mapping text to the generated image, then another doing the reverse with an image mapping to a generated text. A more recent attempt at this is MirrorGAN [4], which uses a stacked generator architecture to generate an image and an auto-encoder to regenerate semantically similar text. In examining the architectures, we decided to work with StackGAN. This architecture has two stages: stage-I GAN and stage-II GAN. Stage-I GAN does a rough estimation sketch of the text input provided. For a mountain landscape this would be the equivalent of having mountains distinguishable from sky but not being able to distinguish individual trees or different mountain faces. Stage-II uses the same text input that stage-I has and the low-resolution output image of stage-I as an input. This stage is where the details are included, like being able to distinguish mountain faces and trees.

Real world applications that we discussed include a provided digital image of a landscape generated based on the participant's DNA. Although this might be a neat souvenir for the participant, some privacy concerns arise with participant DNA structure being transmitted over an unsecured connection. Also, the image would be a random conglomeration of DNA, but a clever enough person might be able to reconstruct participant data with the correct tools.

2 Problem Definition

Our goal is to convert a valid DNA input sequence into a landscape image using a Generative Adversarial Network (GAN). Examples of these can be seen in the Data section 3. This is a Text-to-Image synthesis issue, with the generator taking valid DNA codons (TCAG) as input and creating landscape 224 by 224 by 3 (RBG) images as output. Image values range within [0,255] across the 3 color channels. Similar to a more general Text-to-Image GAN, which would take text descriptions to generate images, the DNA codon sequence will be a translated language by which we generate landscapes. Valid DNA codons follow the standard generic code table with valid inputs of T, C, A, and G in groups of 3. Each sequence must also start with a start codon and end with an end codon. Noted in the original table, there are 25 different amino acids that these codons correspond to. While it is not currently our goal, this fact could be used similarly to an alphabet, and thusly a sentence or description of the image.

Looking more comprehensively at the GAN, the generator takes as input valid codon, considered the latent space of the system, and outputs the image we desire. The discriminator would take as input an image, either directly from the dataset or from the generator, and return the probability that the image is real (direct input) or fake (generated input). It is obvious that, generally, DNA does not map to images in the real world. For our purposes, we currently create input to the generator, the latent variables, by converting the DNA sequence to a normalized average value between [0, 1]. Specifically, each character is represented by a number between 0 to 3 (ACGT respectively). These numbers are averaged by taking the sum of them and dividing by the length of the sequence and the length of a codon (3). This average is what we record, and thus we convert the variable length string into a single floating-point feature. This is done to create 128 latent variables, thus creating a 128-dimension latent space. Finally, the values are normalized to [0,1], finishing our noise generation. We played around with using text embeddings using k-mer counting and Sequence Graph Transformation, but ended up leaving this out as it would not conform to the latent space defined. It is an area with room for exploration, however. As for the landscape images, all images vary in resolution, many of which are far too large to train on, and thus must be down-sampled into their 224 by 224 format. This is handled by keras preprocessing when loading the dataset into training and testing sets.

[1] sequence of three DNA nucleotides that corresponds with a specific amino acid or stop signal during protein synthesis.

Table 1: DNA Formatting

ID	Name	Description	Number of Features	Per letter annotation for:	Sequence

3 Data

The datasets used within this project are a Landscape dataset obtained from Kaggle and a DNA dataset obtained from Joint Research Center. They can be found at their respective pages:

`https://www.kaggle.com/arnaud58/landscape-pictures`
`https://data.jrc.ec.europa.eu/dataset/2abb5c2b-3ab6-4ce4-b103-cb1c5fc7349e`

3.1 Landscape Dataset

The landscape dataset contains a variety of non-descriptive landscape images. There are no classifications or metadata for any image. All images start as JPEG images of varying sizes, but are pre-processed to be 224 by 224, as can be seen in Figure 1. The images used were scraped from Flickr images

Figure 1: Pre-processed images

3.2 DNA Dataset

The DNA dataset contains various sequences of DNA codons. Collected by the Joint Research Center by sequencing genetically modified strains of bacillus subtilis used for production of vitamin B2 in feed additive.
Data is formatted in a *.fastq* file with the structure found in Table 1.

4 Method

Here we will discuss the methodology of employed on the project. All code was developed to be run on Google Colab using Jupyter Notebook formatting for Python. Code can be found at `https://github.com/KaiwenS7/DeepLearningProject`.

4.1 Relevant Equations

GANs optimize their networks around a minmax problem. The generator G is optimized to produce images that are difficult for the discriminator D to differentiate from real images. D is then optimized to know the difference between generated images and real images. This optimization comes by way of minmaxing the loss function. For our loss function, we use the binary cross entropy between the predicted labels and the true labels for the discriminator D and between predicted labels and fake labels for the generator G. Cross Entropy can be defined as:

$$H(p,q) = -\Sigma_i p_i log q_i = -ylog\hat{y}\breve{}(1-y)log(1-\hat{y}) \quad (1)$$

where $p \in y, 1-y$ or the true examples and $q \in \hat{y}, 1-\hat{y}$ or the generated examples.

We can convert the loss function into a minmax problem for G and D, where we maximize the loss of D while minimizing the loss on G. This dual-optimization can be summarized with the objective function:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}}[logD(x)] + E_{x \sim p_z}[log(1 - D(G(z))] \tag{2}$$

Where x an image from the real data p_{data}, and z is the noise from a Gaussian distribution p_z.

To evaluate images produced by our model, we use the Fréchet inception Distance (FID). This, like inception score, is used to evaluate the quality and diversity of a GAN's image production. FID in particular compares the distribution of generated images to the distribution of training or real images. This assumes that the two distributions are multidimensional Gaussians: $\mathcal{N}(\mu, \Sigma)$, the generated distribution and $\mathcal{N}(\mu_\omega, \Sigma_\omega)$ the real distribution. Using these, we can describe FID as:

$$FID = |\mu - \mu_\omega|^2 + tr(\Sigma + \Sigma_\omega - 2 * (\Sigma\Sigma_\omega)^{1/2}) \tag{3}$$

4.2 Implementation

We originally followed the StackGAN model as closely as was capable, implementing the neural networks of the stage 1 GAN and the stage 2 GAN without the conditional augmentation. In the original paper, conditional augmentation was used for word segmentation and sentence feature extraction, which is then fed into both stages. We do no such thing with DNA sequences. Both stages use an Adam optimizer with a Learning Rate of 0.0002 and a Beta of 0.5 for both the discriminator and the generator. We have also implemented a learning rate decay of 0.5 every 50 epochs.

In preparation for discussion of each stage, we will first define the blocks used to simplify explanation of the models. There are 2 repeating blocks used in building the GANs, the Up Sample block and Down Sample block. The Up Sample block involves an up sampling layer followed by a convolutional layer with a ReLU activation. The Down Sample block is a convolutional layer of at least stride 2, followed by a ReLU activation and a Dropout layer. Also as a note, all LeakyReLU activations use an alpha of 0.2, and all Dropout layers use a 0.2 dropout rate.

The stage 1 generator takes an input with the latent space size of 128. This is fed into a dense layer and reshaped with the values (4, 4, 1024). This is fed through a series of 4 Up Sample blocks, each reducing the filter size by half. Finally we use a convolution layer with 3 filters and a tanh activation to obtain a 64x64x3 image. The discriminator follows a convolutional layer of 64 filters and a kernel of 4 using a LeakyReLU activation. Following this is 4 Down Sample blocks, each doubling the number of filters, and finally the discriminator ends with a dense layer of output size 1, the probability that the input sample is real.

The stage 2 generator follows the output of the stage 1 generator, taking the 64x64x3 image as input. Since we desire a 224x224x3 output, the input image must be resized to 56x56x3 using lanczos resampling [5]. This is fed through 2 Up Sample block of kernel 4 with filter sizes of 112 and 224 respectively, and the generator finishes with a convolutional layer of 3 filters and a tanh activation output. This provides a 224x224x3 image. The discriminator opens with convolutional layer of filter size 224, kernel 4, and stride 2, followed by a LeakyReLU layer. This is then fed through 3 Down Sample blocks of kernel 4 and filters 56, 112, and 224 respectively. Finally, a dense layer of output size 1 generates the discriminator's prediction.

4.3 Training

5 Results and Discussion

5.1 Evaluation

6 Conclusion

A short summary of the work. Keep it brief and concise, and straight to the point.

References

[1] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. *arXiv preprint arXiv:1612.03242*, 2017.

[2] Zhu, Jun-Yan and Park, Taesung and Isola, Phillip and Efros, Alexei A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In IEEE International Conference on Computer Vision (ICCV), 2017.

[3] Trevor Tsue and Samir Sen and Jason Li Cycle Text-To-Image GAN with BERT *arXiv preprint arXiv:2003.12137*, 2020

[4] Tingting Qiao and Jing Zhang and Duanqing Xu and Dacheng Tao. MirrorGAN: Learning Text-to-image Generation by Redescription *arXiv preprint arXiv:1903.05854*, 2019.

[5] Lanczos Resampling. In *Wikipedia.* `https://en.wikipedia.org/wiki/Lanczos_resampling`

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford and Xi Chen Improved Techniques for Training GANs *arXiv preprint arXiv:1606.03498*, 2016.