GAN User Manual

Zach Brill, Dan Kershner, Annan Miao, Troy Schwab

December 2018

Contents

1 Introduction

1.1 Introduction

Creativity is not often viewed as a problem that needs solving. While in some sense we agree with this sentiment, that does not mean that we should not create computer programs which attempt to make inroads into explaining what creativity is and where it can manifest. It seems as though humans prefer to place only ourselves on the pedestal of creation, suggesting that a machine can never truly make anything new, or at least can't make anything new that we didn't tell it (or teach it) to. Indeed, this stance is taken by many in opposition to any credibility we may attempt to give to AI and its successes. Often times the success of the machine is attributed to those who designed it. We would like to challenge this idea by attempting to create an artificial intelligence that creates art. If we fail with this endeavor we hope to at least propose some valid questions that poke at the issues of creativity, art and the machine's place in this realm.

1.2 Background

GAN is a type of generative modeling. The idea of GAN started with Ian Goodfellow et al.'s landmark paper "Generative Adversarial Nets" (GANs), published at the Neural Information Processing Systems (NIPS) conference in 2014. The idea is to propose a new framework for estimating generative models via an adversarial process, in which the program simultaneously trains two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability for D to make mistakes. This framework constructs a two-player minimax game.

GANs were designed to avoid many of the drawbacks associated with the above models. As opposed to Fully Visible Belief Networks, GANs use a latent

code, and can generate samples in parallel. Unlike Variational Autoencoders, GANs are asymptotically consistent. Also, GANs do not require Markov chains, which is an advantage over Boltzmann Machines.

GANs have a wide range of applications. For instance, one can feed some text written with a particular handwriting as input to a generative model to generate more text in the same handwriting. Generative models can also be used in exploration in reinforcement learning where they can be used to generate artificial environments. Other applications include the conversion of sketches to images, image de-noising, conversion of low resolution images to high resolution, generation of art, and the conversion of satellite images to maps.

1.3 Motivation

Our group was particularly interested in the creation of art, and more specifically the creation of artistic images. Where most modern AI is excelling at the classification of images, the generation of images often times takes a back seat to classification. We took this project as a great opportunity to explore a less technical side of computer science, i.e. the arts. In class, however, we had only touched on the ways that AI can be used to classify certain inputs. For example, our first neural nets would take two boolean inputs and output their truth values for various logical operators. Because of this, at first it was not very clear to us how we might go about attempting to reproduce creativity in a machine or computer program.

2 Implementation

GANs implicitly estimate a data distribution using the following minimax objective loss function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Figure 1: minimax function for GAN

This loss function narratively describes a game between two opponents, a discriminator and a generator. The goal of discriminator is to judge whether that example was drawn from the true distribution $p_{data}(x)$, or from the generated fake distribution $G(z)$. Hence, proper optimization over the loss function in $Fig.$ 1 results in an equilibrium state where the discriminator and generator are evenly matched and the discriminator cannot distinguish between real and fake generated samples, assigning a half probability to any sample it receives.

The goal of the research is to understand the theory and structure of GAN, implement a GAN from scratch and apply it to generate images and possibly art. The program uses matplotlib for plotting the generated images after every 5 epochs, Tensorflow and Keras as the backend libraries and tqdm to show a command line progress bar for each epoch (iteration). The generator and discriminator are convolutional neural networks (because GANs are mainly used for image tasks). This network takes some random noise vector and outputs an image. When training the generative network, it learns which areas of the image to change so that the discriminator would have a harder time differentiating its generated images from the real ones. The generative network keeps producing images that are closer in appearance to the real images while the discriminating network is trying to determine the differences between real and fake images. The ultimate goal is to have a generative network that can produce images which are indistinguishable from the real ones.

Keras is our high-level network to help develop and evaluate deep learning models with tensorflow as the backend, matplotlib for plotting, and tqdm to show a fancy progress bar for each epoch (iteration). We use the Adam optimizer for the generator and discriminator networks. This optimizer is best used to optimize stochastic objective functions, which is especially helpful for our generator network which works functions almost entirely stochastically during

the initial epochs. For both networks we create a neural network with four hidden layers with a Leaky ReLU activation function. The Leaky ReLU helps to eliminate the likelihood of the network 'dying'. Dying occurs when the negative activations essentially have zero effect because the rectifier only properly handles positive outputs. A leaky model gives these negative outputs at least some negative bias, meaning their effects will propogate through the network, if only a little. We also add dropout layers for the discriminator to improve robustness on unseen images. The final layer of the discriminator simply outputs the probability that an image it received is real. The final layer of the generator outputs an image that it thinks should be very close to the images in the training set.

3 Instructions

To run our GAN, navigate to the topmost GAN folder and, from the command line, enter the following if your system's default python is version 3 or above:

```
python gan.py
```

otherwise, enter:

```
python3 gan.py
```