APPLICATIONS OF GENERATIVE ADVERSARIAL NETWORKS TO NATURAL HANDWRITING AND MUSIC CREATION

KYLE LITTLE, CHRISTOPHER STEVENS, AND DANA D. CLAHANE

ABSTRACT. We apply the recent concept of a generative adversarial network (GAN) to two tasks: natural handwriting emulation and music creation. Because it of great interest to mathematically generalize neural networks (NN's), along the way, we also present some abstract definitions that can serve as templates for NN's and GAN's.

1. INTRODUCTION

In this paper, we exhibit two applications of so-called generative adversarial networks (*1*), and we attempt to generalize various aspects of them, including even the definition of a neural network. Before describing GAN's, we will introduce some preliminary notation and definitions in the next sections. In the sections that follow the section below, we proceed to applications of GAN's to natural handwriting emulation and music creation. Because the generalization of neural networks is of great interest, along the way, we will take the opportunity to propose general mathematical notions. The reader will be provided with instructions for playing with our GAN's, along with links to locations where data is stored for experimentation. We'll also give some examples of outputs of our GAN's in both applications. For natural handwriting, images are provided, and for natural music creation, we provide links where MIDI files are stored, so that the reader can perhaps be impressed.

2. DEFINITIONS AND NOTATION

Notation 2.1 (Prediction ($\sim$))**.** Suppose that $p : \mathbb{R}^m \to \mathbb{R}$ and that $\alpha \in [0, 1]$. Then the notation $\mathbf{v} \sim_\alpha p(\mathbf{v})$, or simply, $\mathbf{v} \sim p(\mathbf{v})$ in the case that $\alpha = 1/2$, denotes the phrase "The value $p(\mathbf{v})$ allows us to conclude that $\mathbf{v} \in S$, with probability of truth of this statement being $\geq \alpha$." Another way we say this statement can be "$p(\mathbf{v})$" predicts $\mathbf{v}$ with probability $\geq 1/2$. The components of $\mathbf{v}$ are called *predictors.*

Example 2.2. Suppose that $S = [0, \infty)$. Suppose that $p : \mathbb{R} \to [0, 1]$ is given by

$$p(x) = \int_{-\infty}^{x} e^{-t^2}\, dt.$$

Then $7.5 \sim_{1/2} p(7.5)$ and $7.5 \sim p(7.5)$ both denote the same statement, "The value $p(7.5)$ allows us to conclude that $7.5 \in [0, \infty)$, with probability of truth of this statement being $\geq 1/2$. In this case 7.5 is a predictor.

Partially Supported by NSF-DMS-1541911 RE-Cˆ2 grant. RE-Cˆ2 Research Experiences in Community Colleges.

To improve our predictions in the specific applications of GANS discussed in this paper, we must first discuss how to quantify inaccuracy in predictions. To do so, we attempt to define an error function in the context of the above definition of prediction. Later in this paper, we will specialize this idea to the more specific applications of Generative Adversarial Networks we are interested in, to natural handwriting and music creation.

Definition 2.3 (Error Function)**.** Let $\mathbf{x}$ denote an $\mathbb{R}^n$-valued (datum) variable that we would like to classify as being or not being in a given set S, We say that a given function is an *error function induced by the variable* $\mathbf{x}$ *and* $\mathbf{p}$ and denote such an error function by $\mathbb{E}_{\mathbf{x}\sim\mathbf{p}(\mathbf{x})}$ iff the following conditions are satisfied in a given classification problem that we are interested in:

(1)
$$\mathbb{E}_{\mathbf{x}\sim\mathbf{p}(\mathbf{x})} : \mathbb{R} \to \mathbb{R},$$
(2) A given input $d \in \mathbb{R}$ for this function is the output of $f \circ P$, where f is a one-to-one function on $[0, 1]$, and P is a probability function.
(3) $\mathbb{E}_{\mathbf{x}\sim\mathbf{p}(\mathbf{x})}(d) = 0$ when the input d corresponds to the statement $\mathbf{x} \sim_1 p(\mathbf{x})$;
(4) A large value of $\mathbb{E}_{\mathbf{x}\sim\mathbf{p}(\mathbf{x})}(d)$ for some input d corresponds to the statement that $\mathbf{x} \sim_\alpha p(\mathbf{x})$ only for small values of $\alpha \geq 0$.

3. Neural Networks

In order to make predictions we must use some model. For the purpose of GANS the model we will use is based on the human brains neurons. Simply put we may break down the human brains processing into layers of neurons which we will model using vertices in a digraph as follows.

Definition 3.1 ((Generalized) Neural Network)**.** We call (V, E, W) an *neural network*, or, more briefly, an NN, iff the following statements hold:

(1) (V, E) is a digraph.
(2) W is a function whose domain is E.
(3) $\forall(u, v) \in E$ such that u is not the terminal vertex of an edge in E, we have that u is a real number.
(4) $\forall(u, v) \in E,\ W(u, v) : \text{Range}(u) \to \text{Dom}(v)$.

An NN (V, E, W) is called *real-valued* if and only if every $v \in V$ that is a terminal vertex of some $e \in E$ and not the initial vertex of some $f \in E$, is a real number.

Furthermore, the vertices of an NN that are both the terminal edge of a some vertex in V and the initial edge of another vertex in V are also called *neurons*.

Recall that graphs can be connected or disconnected, or, they can be simple or non-simple. Since NN's are partially composed of graphs, then NN's can be called, connected, disconnected, simple, or non-simple, in exactly the same manner, depending on the nature of the underlying graph.

However, practically speaking, we note that in reasonable applications, the underlying graph of an NN should be connected and simple. Since an NN is a mathematical model inspired by the way a brain works, a disconnected NN would be like two separate brains that don't interact with each other (not of interest in this paper). Furthermore, a non-simple NN would be like a brain with at least one infinite loop of a chain of neurons, which does not happen in nature, and, for algorithms, is something that is obviously undesirable. (We don't want infinite loops in our algorithms.)

For the applications that we are interested in, the vertices of a given neural network that are functions will have domains and ranges in Euclidean space.

Definition 3.2 (Euclidean NN and notation within it)**.**

(1) We call an NN (V, E, W) *Euclidean* iff each $v \in V$ is either a real constant or $v : \mathbb{R}^m \to \mathbb{R}^n$ for some positive integers m, n and, $\forall (u, v) \in E$, we have that $W(u, v)$ is either a real number or a column vector of real numbers whose number of components is the same as the dimension of the domain of v.

(2) Let (V, E, W) be an Euclidean NN, and let $v \in \mathbb{N}$ be the cardinality of V. Denote the vertices by $v_1, v_2, \ldots, v_k$.
   (a) We denote by $I := \{0 \,..\, k\}$, the set of all integers between and including 0 and k.
   (b) With respect to $m_0, m_1, m_2, \ldots, m_k, n_0, n_1, n_2, \ldots, n_k \in \mathbb{W}$, the set of all positive integers, we impose that $m_i \; \forall \, i$ is the dimension of the domain of v_i vertex and also impose that n_i is the dimension of the range of v_i.
   (c) That is, we suppose that each vertex v_i of V here satisfies
$$v_i : \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$$
   (d) If $i, j \in \{1..k\}$ and $(v_i, v_j) \in E$, then we denote $W(v_i, v_j)$ more briefly by $w_{i,j}$.

The weights we assign each neuron change the network's influence on these neurons. Sometimes, we do not want a neuron to be included in the calculation for training in order to remove dependency on one neuron. Analogously to the human brain we have certain neurons removed from the graph.

Remark 3.3 (Drop-out)*.* A weight of zero represents a vertex or neuron has *dropped-out* for some training cycle. Since the weight is a scalar multiplied by the input vector, when that weight is zero, that input will be ignored.

We have chosen to define neural networks as above to impose a general mathematical structure to the theory of a largely implementation-based algorithm. Allowing each neuron to be any function and generalizing weights allows for a general theoretical neural network definition. A GAN uses two neural networks which are consequently trained in tandem. We will define the two separate neural networks before we define the full GAN.

Definition 3.4 (Perceptron)**.** A *perceptron* is a neural network consisting of a single layer of weights and with identity functions as the vertex set.

Remark 3.5*.* As an effect of this definition, perceptrons may be modeled as linear predictors, or as functions $\mathbf{q}(\mathbf{x}) \sim \mathbf{p}(\mathbf{x})$. Later in the paper we will use this notation to refer to perceptrons.

Definition 3.6 (Discriminator)**.** A *discriminator* is a type of neural network that determines authenticity of input data. It can be modeled by a function

$$D : \mathbb{R}^n \to \mathbb{R},$$

together with the other properties of a more general NN.

The discriminator is some neural network that takes in some vector to check if it is a part of the same set as the data it was trained on, and outputs a probability that the vector is a part of its training set.

Definition 3.7 (Generator)**.** A *generator* is a type of neural network that produces new data based on training, and can be described as a function

$$G : \mathbf{z} \to \mathbb{R}^n$$

where $\mathbf{z}$ is a noise vector.

The generator is some neural network that takes in some kind of random noise and outputs things that look like the training data. Thus, when we train our generator in tandem with our discriminator we can evoke the idea of counterfeiting, in the sense that the outputs of the generator try to fool the input to the discriminator.

4. Definition of a Generative Adversarial Network

Definition 4.1 (Generative Adversarial Network)**.** We define a *generative adversarial network* (GAN) in terms of a two player min-max game between a discriminator D and a generator G with value function

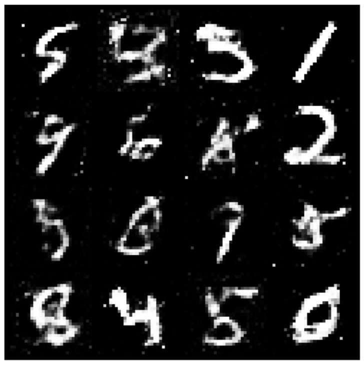
$$\underset{G}{min}\,\underset{D}{max}\; V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\,[\log(1 - D(G(\mathbf{z})))] \quad (1)$$

where we train D to maximize the probability of correctly assigning labels to both training data and data that comes from G. In turn we train G to minimize $\log(1 - D(G(\mathbf{z})))$.

5. Results

5.1. **Natural Handwriting.** The first application of GANs we attempted was to produce handwritten letters and numbers in a manner that looks human written. We achieved partial results, based upon and modified from open source work by Diego Gomez[1]. Many changes were made in order to preserve outputs long term, as well as laying foundations for cohesive data inputs. For example, work was first done using MNIST as the input data, which produced very legible numbers.

[1]https://github.com/diegoalejogm/gans



The code for the handwriting portion and instructions on how to run it can be found at https://github.com/Onionchi/GANdwriting

5.2. **Music Creation.** With the second application of GANs we attempted to produce new music based upon MIDI files from various classical composers. This involved two neural network models over the course of our research. Both of these models were useful in finding more about GANs. Outputs to both models were generally not similar to music that would generally be considered "normal," but there was significant improvement between the two models. The model is based off starting code. [2]

The first model was a naive implementation of a Generative Adversarial network. This model simply took the input MIDI files and tried to produce new MIDI files that were similar. This led to results that generally fit in one of two categories.

(1). The MIDI file output would be a jumble of notes equivalent to someone randomly pushing keys on a keyboard.

(2). The MIDI file output would degrade into two notes one very high pitched and the other very low pitched. These notes would repeat at seemingly random intervals throughout.

We think that these two outcomes are due to a phenomenon known as modal collapse. More specifically over the process of training GANs, GANs will sometimes after extensive training learn only one way of making output as indicated above.

The second model has many improvements over the first. The predictor that used to be a raw note value was changed to be a difference between notes over time. This way the neural network could try to learn some form of structure over time in music. This change was very successful and brought about new MIDI output that had segments resembling musical scales and arpeggios that might be features of human composed music.

[2]https://skymind.ai/wiki/generative-adversarial-network-gan

The code that was written is available publicly at https://github.com/Aniranth/GAN_Project Simply get Python 3 on your system and follow the instructions in the read-me file of the GitHub. You can also find sample MIDI files generated by several epochs of testing of the GAN at this link.

6. Further Research

There are many different directions that can be taken for further research. Currently for the application of handwriting we are trying to emulate the handwriting of Thomas Jefferson, as he has a large volume of annotated written work, available at the Library of Congress.

We would like to look further into the modal collapse that was experienced in the Music application. The authors would specifically like to consider a paper found late found as the research in this paper developed.(*2*)

Another project the authors have significant personal interest in will be to implement the more generalized neural network that outputs functions rather than real numbers.

7. Acknowledgements

Kyle Little would like to thank Ryan Wilcox for his suggestion to use a change in notes for the note predictor in the second version of the Music application for GANs.

Christopher Stevens would like to thank David Nuon for his helpful conversations about different types of neural networks. These conversations helped inspire thought about neural networks in the non-traditional view expressed in this paper.

All three authors would like to thank Professors Helena Noronha, Bruce Shapiro, and Adriano Zambom of CSUN for their work organizing this research project and their lectures on data science. As well, we would like to thank Kevin Scully for several wonderful talks about neural networks at the Fullerton College Mathematics Colloquium, which inspired our selection of this data science topic for the current project. The authors would also like to thank the Department of Mathematics at California State University Northridge for providing hospitality and for hosting the Math Research Experiences in Community Colleges Conference for the past three years. It is exciting to see math research beginning to sprout in community colleges, and data science, the emphasis of this NSF project for the year, is well-chosen for many reasons.

This project was completed as part of Research Experiences in Community Colleges (REC^2), which has been funded by the National Science Foundation, under Division of Mathematical Sciences Grant Number 1541911.

References

1. I. J. Goodfellow *et al.*, *Generative Adversarial Networks*, arXiv: `1406.2661`

2. K. Li, J. Malik, *Implicit Maximum Likelihood Estimation*, 2018, eprint: `arXiv:1809.09087`.

E-mail address: `kylelittle2134@gmail.com`

E-mail address: `cestevens@berkeley.edu`

E-mail address: `dclahane@fullcoll.edu`

Mathematics and Computer Science Division,
Fullerton College, 321 E. Chapman, Fullerton, CA 92832-2095