# Project Proposal: Generative Adversarial Networks

Justin Dong
Emily Reed

October 2019

## 1 Introduction

An artificial neural network is a composition of nonlinear functions trained on a set of data whose output corresponds to the approximation of a function or a classification pattern for the given data. In his 1989 paper *Approximations by Superpositions of a Sigmoidal Function*, Cybenko proved that a function of the given form,

$$f(x) = \sum_{j=1}^{n} w_j^{(2)} \sigma(w_j^{(1)} x + b_j^{(1)}) \tag{1}$$

is capable of approximating any continuous function on $\mathbb{R}^n$ under certain weak conditions on $\sigma$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is a continuous univariate function and $w^{(1)}, w^{(2)}, b^{(1)} \in \mathbb{R}^n$ [3]. Note that this function describes a one-layer neural network. We can write a multilayer feedforward network generally as follows with the goal of determining the parameters $w$ and $b$.

$$\begin{cases} x_1 = x \\ x_\ell = \sigma(w^{(1)} x_{\ell-1} \ + b^{(1)}), \ell = 1, ..., L-1 \\ x_L = w_L * x_{L-1} \ + b_L \end{cases}$$

In addition, Cybenko proved a similar result for binary classifiers. This universal approximation property has been the key in the success of fields such as natural language processing [6], speech recognition [5], and computer vision [7] among others. A more recent application of neural networks is in the creation of generative-type models. In particular, generative adversarial networks (GANs) allow us to generate samples with the same statistics as a given training set. First introduced by Ian Goodfellow et al. in 2014, the authors used GANs to generate handwritten digits and realistic human faces [4]. Recently, GANs have been used for a variety of applications such as deep fakes. These deep fakes generate faced-swapped fake videos and images by generating the same statistics of the given real videos and images [1].

Our goal for the semester will be to understand and implement a basic GAN as described in [4] in order to produce realistic handwritten digits. Time-permitting, we will also implement Wasserstein GANs [2], which utilize a different metric when training the networks which has shown improved results over traditional GANs.

# 2 Background

In this section, we introduce generative adversarial networks, or GANs. The framework for a GAN consists of two neural networks, a generative model $G$ and a discriminative model $D$. The generative model is described by the mapping $G(z;\theta_g)$, where $G$ is a differentiable function with the associated parameters $\theta_g$. On the other hand, the discriminative model is described by the mapping $D(x;\theta_d)$ where $D(x)$ represents the likelihood between 0 and 1 that $x$ came from a prespecified data distribution $p_{data}$, with the associated parameters $\theta_d$.

The goal of the generator is to map samples from a prior distribution $p_z$ (generally taken to be a Gaussian or uniform distribution) to an approximation of the target distribution $p_{\text{data}}$ while minimizing the expected value of $\log(1 - D(G(z)))$ with respect to the prior distribution. This quantity is minimized when the discriminator $D(G(z))$ outputs a likelihood close to 1, meaning that the generator is able to produce a data sample to fool the discriminator. Contrarily, the goal of the discriminator is produce a number signifying the likelihood that $x$ came from $p_{\text{data}}$ while training to maximize the expected values of $\log(D(x))$ and $\log(1 - D(G(z)))$. Note that $\log(D(x))$ is maximized when $D(x)$ is close to 1, indicating that the discriminator can correctly identify data from the target data distribution. Additionally, the value $\log(1 - D(G(z)))$ is maximized when $D(G(z))$ is close to 0, implying that the discriminator can correctly determine that the generated output from $G$ is fake.

These two models are trained against each other in a two-player minimax game in the following form which we will call the objective function $V(D, G)$,

$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x;\theta_d)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z;\theta_g);\theta_d)] \tag{2}$$

where $p_z$ is a pre-specified prior distribution on which the generator acts and $p_{\text{data}}$ is the distribution from which our training data has been generated [4]. In practice, GANs of this form are difficult to train due to the fact that optimizing $D$ within the inner loop of training is computationally impractical and can result in overfitting. This leads to the outline of an algorithm for implementing this minimax game as outlined in [4] below. Note that we complete $k$ steps of training the discriminator for every one step of the generator. Also, in order to approximate the expected values given in the objective function, Monte Carlo approximation with importance sampling is used at each iteration.

**Algorithm 1:** Minibatch stochastic gradient descent training of generative adversarial nets. The hyperparameter $k$ describes the number of steps to apply to the gradient.

**for** *number of training iterations* **do**

  **for** $k$ *steps* **do**

  - Sample minibatch of $m$ noise examples $\{z^{(1)}, ..., z^{(m)}\}$ from the noise prior distribution $p_g(z)$
  - Sample minibatch of $m$ examples $\{x^{(1)}, ..., x^{(m)}\}$ from the data generating distribution $p_{data}(x)$
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right] \tag{3}$$

  **end**

  - Sample minibatch of $m$ noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from the noise prior distribution $p_g(z)$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta d} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)}))) \tag{4}$$

**end**

# 3 Methodology

We begin the semester by coding a single layer neural network for approximating 1D functions without TensorFlow in order to fully understand the underlying mathematics fundamental to these neural networks. Next, we plan to implement a convolutional neural network in TensorFlow for image classification. Finally, we plan to implement a GAN using convolutional neural networks as the structure for our two adversarial models in order to generate realistic handwritten digits. In order to implement the GAN, we will first start with implementing the discriminator model and train it to distinguish between the samples from the data distribution and the prior distribution. Later, we will add the generator model. Finally, time-permitting we would like to implement a Wasserstein GAN (WGAN) for the same problem.

# References


[1] *faceswap-gan*, GitHub repository, (2019). `https://github.com/shaoanlu/faceswap-GAN`.

[2] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, arXiv preprint arXiv:1701.07875, (2017).

[3] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems, 2 (1989), pp. 303–314.

[4] e. a. Goodfellow, Ian, *Generative adversarial networks*, Advances in Neural Information Processing Systems (NIPS), (2014).

[5] e. a. Graves, Alex, *Speech recognition with deep recurrent neural networks*, International Conference on Acoustics, Speech and Signal Processing, (2013).

[6] M. C. Hirschberg, Julia, *Advances in natural language processing*, Science, 349 (2015), pp. 261–266.

[7] e. a. Krizhevsky, Alex, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (NIPS), (2012).