Building a Customizable GAN Package Tool in Python

Helping Users Make and Test GANs in an Easy to Use Format

Ryan Astor
Kyle Costello
Joshua DeOliveira
Alek Lewis

A thesis presented for the completion of
Major Qualifying Project

Faculty Advisor: Elke Rundensteiner
The William Smith Dean's Professor in Computer Science
Founding Director in Data Science

Worcester Polytechnic Institute
Worcester, Massachusetts
March 4, 2022

Contents

1 Introduction

Generative adversarial networks (GANs) are machine learning based generative approaches where a tandem of neural networks train in a competitive environment to effectively synthesise data from a known corpus to perform up-scaling for a data set. Training a GAN is a challenging task that has many moving parts. These all need to be considered before training a GAN properly. GANs are not widely understood by the development community and are complex in nature, which makes them difficult for new users to get accustomed to. Though extremely useful for data generation, the competitive nature of a GAN makes training one difficult and imprecise. Often times, developers may not even know where to start creating one. This makes the demand for a usable package that assists the making of GANs very high. [1]

1.1 State of the Art

Currently, there are tutorials that go over how to write a GAN. One notable tutorial is the Neural Information Processing Systems (NIPS) tutorial. The NIPS tutorial gives a great introduction as to what GANs do to give the user a strong foundation. There are figures throughout the tutorial that give more visual context to what is going on to help the user get a better understanding [2] There also exists several packages in python that help give developers new to GANs some exposure to them. The Keras-GAN package [3] and PyTorch-GAN package [4] both are available for helping get familiar with GANs. Both projects are very similar. One key difference is that they use different packages, Keras and PyTorch respectively, to create their tool. Their main purpose is to allow users to run a vast array of different GANs on their computers immediately from the over a dozen that are rebuilt into the software. It is a great tool to run a first GAN with no experience or skill needed.

1.2 Limitation of State of the Art

The example packages, while nice for seeing what a GAN is, are limited in scope. They can only facilitate the running of rebuilt GANs. Users may find it difficult to get support in building their own custom models from these packages as that is not their purpose. This means that there is a severe lack of a GAN support packages that help users make their own GANs. The available GAN tutorials, while comprehensive, are outdated. The tutorials were written in 2016, which is when GANs were new and not very widely known. GANs have evolved since they were first utilized in 2016.

1.3 Problem Statement

GANs are extremely useful for generating data and have applications in image generation, supplementing data sets for other machine learning applications, and in data inference. However, the complicated nature of GANs makes them

difficult to use for those new to the field without proper tools to assist in their creation.

Our goal is to create a tool that is easy to use for those who are getting started with GANs that will allow them to more easily get started with creating their own.

1.4 Challenges

There are a few challenges that would come from solving this problem. Firstly, we want to make a tool that is as accessible as possible for those new to GANs or inexperienced with them. This group of people is potentially highly diverse in existing coding skill set and knowledge so making a package that is helpful to the most people possible in this group is essential to meet our goals.

In addition, we must evaluate the effectiveness of our tool. Our package needs to have clear documentation that is easy to follow and code that can be understood by the average user. The code needs to be intuitive and logical for someone who has a programming background so they can more deeply understand its purpose.

Another challenge lies within the testing of our tool. It is important that our tool is tested for all of the main use cases so that users can accomplish what that are expected to do without errors. The package would not be accessible if it contains errors and inconsistencies that make it more difficult for the user to work with. It is also crucial for us to test the different edge cases that might occur in addition to common tests so that the possibility of issue is minimized. Doing a sufficient testing protocol will be difficult to ensure the smoothest experience possible.

1.5 Proposed Solution

We hope to write a user friendly package that will be accessible by all members of our expected audience of users. Through this package, we will provide a new avenue for users to learn how to make some of the less complicated GAN architectures. Specifically, the users will be able to write the architectures themselves with the guidance of our documentation and tutorials. Through this, we hope to give users a platform where they can test their GANs more easily and prepare them for building more advanced ones in the future.

1.6 Goals

- Make a GAN package that will help users make their own
- Add clear and helpful documentation and tutorials for our package to help programmers user it
- Test our package tutorial on prospective users to ensure that it understandable

- Publish the package as open-source in python to make it usable to people beyond our project

2 Background

Originally motivated in the domain of image synthesis, competition-based approaches [5] were introduced where multiple machines train against each other in so that one of the machines can create novel data: the generative adversarial network (GAN). Since then, GANs have been growing in research prominence for applications in a variety of domains. Despite being very popular in machine learning research, they are still not widely known to most programmers. Therefore, it is not easy to get started.

2.1 Simple Generative Adversarial Networks

Simple generative adversarial networks consist of two neural networks, a generator G and discriminator D, that compete in a zero-sum game in which each machine attempts to minimize or maximize the following objective function $f(G, D)$,

$$\min_G \max_D f(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z))] \quad (1)$$

Generator G attempts to minimize $f(G, D)$ by drawing a random sample of latent vectors z from a distribution defined over the latent space $\mathbb{R}^n$ and maps them to a real space $\mathbb{R}^m$. Likewise, discriminator D maximizes $f(G, D)$ by discriminating real data x from the synthetically generated data $G(z)$. The tandem trains in an iterative process where one machine's parameters are temporarily frozen and its adversary learns using the gradients derived from the cross-entropy loss of the frozen machine. This process alternates in a periodic fashion, where G finds a mapping function from the fake distribution to the real distribution, implicitly attempting to fool D, whereas D combats this process by training to learn the patterns and nuances only found in the real data. Under ideal training conditions, G's generated data is similar enough to the real data that D is forced to randomly guess whether the data is real or fake [5].

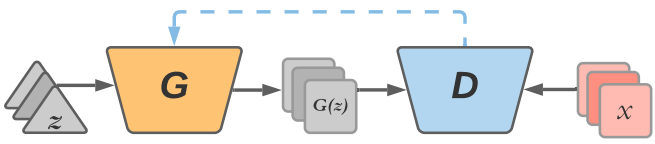

Figure 1: Simple GAN schema

Simple GANs are limited to single class data generation and are unable to allow user-adjustable generation. This schema is easily vulnerable to mode collapse: a scenario of training failure in which G learns to generate data solely mimicking the majority class and/or dominant pattern found in real data in order to minimize its loss as quickly as possible. Mode collapse results in a limited scope of synthetic data in which G can generate meaningful diversity, causing G to have little practical use post-training [6–8].

2.2 Training Instability

Machines that train in an unstable fashion can no longer be assumed to improve in their performance or loss if trained for an additional epoch. This type of behavior can be diagnosed by non-monotonically decreasing trends in loss over the course of training, such as high variance, oscillating patterns, or even divergence from the loss's goal. Highly sensitive hyper-parameters, an inappropriate or ambiguous loss function, or incompatible architectures can all result in unstable training.

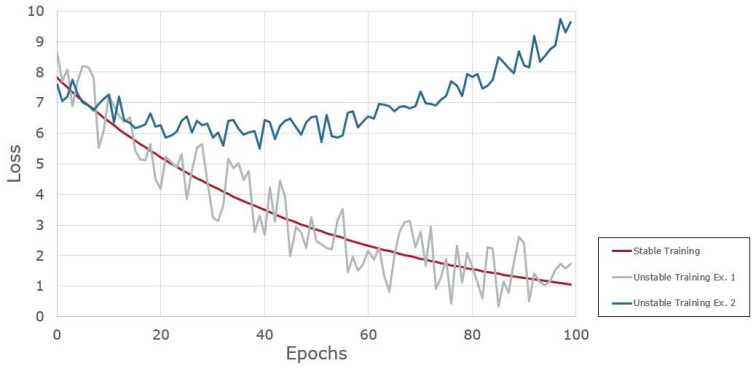

Figure 2: Stable training (red line) is monotonic and deceasing with negligible variance as the loss approaches zero. One example of unstable training (grey line) trends wildly towards zero with non-trivial variance. The most egregious instance of instability (blue line) results in a divergence from the optimal loss after a short training duration.

One theory for how GAN training becomes unstable is that it is a consequence of the discriminator becoming trained enough to perfectly discriminate between the generated and real data. In this state, all of the generated data is marked as fake and the discriminator does not pass any useful gradients down to the generator. As the remaining gradients are dominated by noise, continued training is unlikely to improve the generator and may even decrease its performance [9].

2.3 Modified Architectures

One way to customize the behavior and capabilities of a trained generator is by modifying the architectures of one or more machines in the GAN environment. Architecture modifications allow for more realistic data synthesis in the context of specific domains. In some cases, finding effective architectures reduces pitfalls that commonly arise in training which, in turn, reduce training instabilities. This can be accomplished by modifying the architecture of D or G, or by introducing entirely additional machines to the GAN.

2.3.1 Conditional GAN

One of the first GAN architecture changes ever proposed was the conditional GAN. This new type of GAN modifies the structure of G and D to include inputs from some extra piece of information. This commonly comes in the form of class labels but also can be any class modifier. For the generator, this modification comes with the initial input noise. This noise is combined with the class label to make a new hidden layer that is used to generate the new output. For the discriminator, the generated or real data is passed in along with the class labels for the correct and more complex input. In the end, this will allow the conditional GAN to generate output that corresponds to the particular labels that wered used as input making it much more targeted than the original GAN. [6].

When considering a datum, which can be either real x or fake $G(z|y)$, that lies within a single class y, then the new loss $f(G, D)$ is defined as

$$\begin{aligned}\min_G \max_D f(G,D) = \mathbb{E}_{x\sim p_{data}(x), y\sim p_y(y)}[log(D(x|y))] + \\ \mathbb{E}_{z\sim p_z(z), y\sim p_y(y)}[log(1 - D(G(z|y)|y))]\end{aligned} \tag{2}$$

Like the simple GAN, the generator will attempt to minimize the loss $f(G, D)$ while the discriminator attempts to maximize $f(G, D)$. The only difference of note here is the additional input of feature label y received in the generator and discriminator. However, this change will only change how data is handled and not impact the core logic of the simple GAN loss function.

Conditional GANs are not a significant architectural shift from the simple GAN as the only difference that is present is the feature label y. The generator G receives random noise z and label y to make some fake data based on y. Then, the discriminator D will take in this generated output $G(x|y)$ with label y as well as real data x with label y and attempt to tell which is which. Finally, the discriminator will pass data back to the generator in the form of the above loss to learn from its success or failure.

2.3.2 Controllable GAN

Controllable GANs are are a class-tunable architecture that builds upon conditional GANs with the inclusion of an additional pre-trained classifier, C, which identifies whether the synthesized data $G(z)$ best matches the class y it was

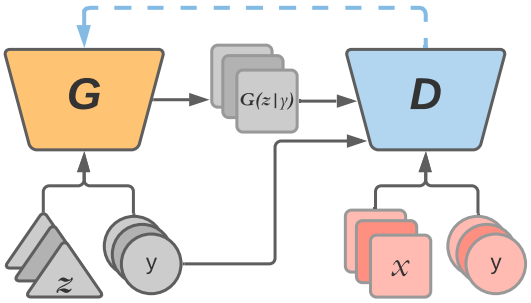

Figure 3: Conditional GAN schema

conditioned on during generation: $G(z|y)$. The third machine C is trained exclusively on real data prior to training the generator and discriminator in the context of the GAN. Fully training ahead of time on real data allows C to properly identify intra-class patterns on realistic, correctly labeled data. For multi-class problems with k independent classes, C learns by minimizing the following categorical cross entropy function $f(C)$

$$\min_C f(C) = -\sum_i^K k_i log(C(x)_i) \tag{3}$$

Consequently, if C is unable to classify the correct labels of only real data, then its poor performance will also present itself when training the GAN and potentially serve as a point of failure for the entire environment.

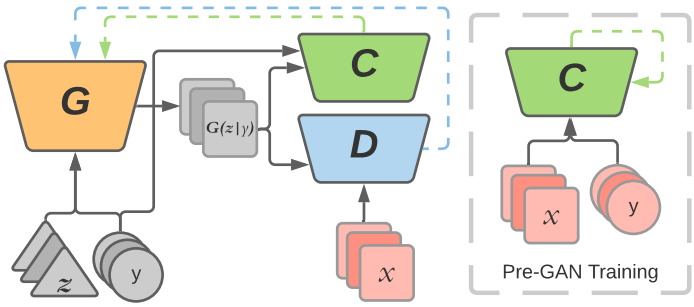

Figure 4: Controllable GAN schema

Similar to the simple GAN, the job of the discriminator D only determines whether the data it receives is real or fake without knowledge of its class label.

[7]. The generator G attempts to generate data that fools the discriminator and makes the classifier guess the correct class the data was conditioned on. Thus, the loss $f(G, D, C)$ can be described as

$$\begin{aligned} \min_G \max_D f(G, D, C) = \mathbb{E}_{x \sim p_{data}, (y) \sim p(y)}[log(D(x)] + \\ \mathbb{E}_{z \sim p_z, (y) \sim p(y)}[log(1 - D(G(z|y)))] - \\ \mathbb{E}_{z \sim p_z, (y) \sim p(y)}[log(C(G(z|y)|y))] \end{aligned} \tag{4}$$

When training, the discriminator it attempts to maximize the $f(G, D, C)$ while the generator attempts to minimize the $f(G, D, C)$. At no point does the classifier learn when the GAN is training, so C does not need to minimize nor maximize $f(G, D, C)$ and only receives data when the generator trains. Since C is also completely independent of D at no point do these two machines communicate any information with each other.

2.3.3 Explicitly Controllable GAN

Explicitly Controllable GANs are a multi-label, user tunable, machine learning architecture. For a given datum x in the real data set X_{real}, x may contain several, independent attributes that may correspond to discrete or real-valued labels. For example in the context of facial images, hair color, eye color and gender would be two discrete labels; likewise pose, hair length, and perceived age could be real-valued attributes. Unlike previous architectures, which take in real noise and generate completely novel data *de novo*, the explicitly controllable GAN takes in a real image, and modifies these real attributes to the desired attributes to generate a modified image x'. In order to accomplish this task, G functions as a variational autoencoder (VAE). G takes in a high dimensional representation of the data and its desired attributes and then transforms it into a lower dimensional latent space. The latent space is then expanded back into the encoding of the higher dimensional representation of the original, where the desired attributes have been "baked in" to the output. An encoder is trained to be able to translate human-readable attribute labels into a compressed latent representation. [8].

When training, real and modified data along with their attrite encodings are passed to a discriminator, and n different classifiers, where n is the number of independent attributes used to encode the data. The discriminator's job is purely to distinguish the given data as real or fake. The job of each attribute classifier is to determine how similar the given data matches the label it is associated with for only a single attribute. The loss from each of the $n + 1$ machines is then fed back into the generator to better encode labels into latent representations, and then decode back into a reconstructed image.

Each different type of modified architecture gives its GAN different capabilities by changing the amounts and types of input G and D take. However, these architecture modifications cannot work in isolation - they must be accompanied by fitting loss functions and training procedures.

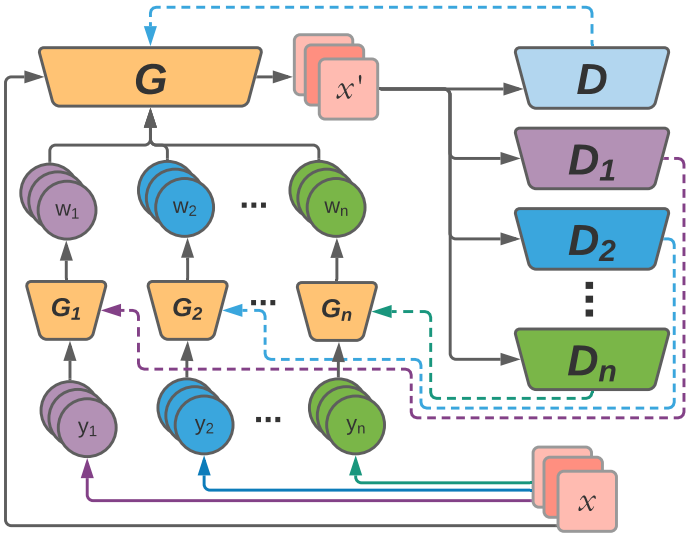

Figure 5: Explicitly Controllable GAN schema

2.4 Generative Metrics

When training models, it is necessary to numerically evaluate how those models perform. In particular, it is useful to calculate a numerical representation of how well a model generates data indistinguishable from the real data. The data set of real data and a synthesized data set of fake data can both be represented by a probability density function which represents how likely a single or very similar datum will appear if randomly drawn from one of the data sets. However, since these data sets are finite and cannot be guaranteed to perfectly follow some functional form, any value x from the probability density function can be estimated via a Parzen window with a Gaussian kernel,

$$\hat{p}(x)_{x\sim P_{data}(X)} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x_i - x)^2}{2\sigma^2}\right) \tag{5}$$

Knowing probability estimates by proxy of the Parzen window means that a divergence metric, via any standard divergence function, measuring the divergence of the fake data against the real data can then be clearly quantified. As the divergence between these two probability density functions shrinks, the generation quality generally improves. In the extreme case however, if the fake data perfectly matches the real data set, then the generator is no longer producing novel data no longer has any practical value [10].

Another metric is the Fréchet Inception Distance (FID), described in Heusel *et al.* [11]. This metric measures the similarity of a given image to a larger data

set. This metric is able to accurately measure the amount of distortion within a given image, and maps very closely to human judgement.

$$\mathbf{FID}_{\mathcal{N}(\mu_r,\Sigma_r),\mathcal{N}(\mu_s,\Sigma_s)} = |\mu_r - \mu_s|^2 + tr\left(\Sigma_r + \Sigma_s - 2\sqrt{\Sigma_r\Sigma_s}\right) \tag{6}$$

FID is especially useful in measuring the rigor of a generators generation. If small distortions in the synthetic image corresponds to a large FID, then the generation technique (or the generator in the context of GANs) weakly recognizes patterns in real data.

2.5 Divergence Based Loss

In ideal scenarios, a well trained GAN will result in a generator that can effectively produce synthesized data that is indistinguishable from its real contemporaries. Utilizing techniques such as parzen windows, synthesized and real data can both be described as probability density functions respectively within some higher dimensional space that corresponds to the data's encoding. [12] Synthesized data that is adequately indistinguishable results in the probability density functions (pdf's) of the two distributions to be extremely similar to one another. [12]

Divergence metrics give quantifiable measure of how two probability distributions differ. Thus, divergence based loss functions in the context of GANs aim at training the generator and discriminator directly to minimize the divergence (ie: improve the similarity) of the fake distribution with respect to the real distribution. The divergence between two probability distributions can be calculated in many different ways based on what properties of the distributions are considered. Kullback-Leibler divergence (equation 7) is one metric that measures how much one distribution diverges from a second. It is also important to note this divergence is asymmetric, meaning the relation $D_{KL}(P||Q) = D_{KL}(Q||P)$ is not guaranteed to always hold.

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)\ log\left(\frac{p(x)}{q(x)}\right) dx \tag{7}$$

Jensen-Shannon divergence (equation 8) is a symmetric metric who's principles are rooted in Kullback-Leibler divergence. Thus, it can be reliably assumed $D_{JS}(P||Q) = D_{JS}(Q||P)$ holds for any two arbitrary pdf's P and Q.

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \quad M = \frac{1}{2}(P+Q) \tag{8}$$

With Wasserstein divergence, GANs are typically trained by running only a fixed number of discriminator updates per generator update. Specifically, each number needs to be the same for the discriminator every time the generator gets updated once. [13]

$$D_{W_a}(P||Q) = \left(\inf_{\gamma\in\Gamma(P,Q)} \int_{M\times M} d(x,y)^a\ d\gamma(x,y)\right)^{\frac{1}{a}}, \forall a \in \mathbb{N} \tag{9}$$

There are also different algorithms for Variational Divergence Minimization (VDM). One algorithm for VDM is the alternating gradient method. With the alternating gradient method, the inner loop tightens the lower bound during the divergence, and the outer loop improves the model of the generator. [13] There is also a single-step gradient method. The single-step gradient method is less complex than the alternating gradient method, because there is no inner loop and the gradients are determined in a single-back propagation. [13]

$$D_{B_F}(P||Q) = F(p) - F(q) - \langle \nabla F, p - q \rangle, \forall F : \Omega \mapsto \mathbb{R} \tag{10}$$

With Bregman divergence (equation 10), a base measure is defined where it stays well-defined even when the data and the model distribution do not have the same support. [13] The key contribution here is to identify base measures that can play a useful role. Bregman divergence forms a measure of difference between using a function that includes a wide range of distances. These distances include the the Euclidean distance and the Kullback-Leibler divergence between finite-cardinality probability mass functions [13]. It is also important to note that some of the divergence methods are instances of Bregman. For example, the implementation of Kullback-Leibler includes a subsection of the implementation used in Bregman. [13]

$$D_{H^2}(P||Q) = 1 - \int_{-\infty}^{\infty} \sqrt{p(x)q(x)}\, dx \tag{11}$$

The Hellinger distance (equation 11) is a typf of f-divergence that is used to quantitatively identify similarities between two probability distributions. The Hellinger distance creates a bounded metric over a given probability space.

$$D_{BC}(P||Q) = -ln\left(\int_{-\infty}^{\infty} \sqrt{p(x)q(x)}\, dx\right) \tag{12}$$

Bhattacharrya distance (equation 12) measures the similarity of two distributions of probability. More specifically, the Bhattacharrya coefficient measures how much overlap there is between two statistical samples or populations. [14] Its intended purpose is to determine how close two samples are after they are measured. The Bhattacharyya distance grows as the difference between the standard deviations increase. The Bhattacharrya distance is widely used when extracting features, processing images, speaker recognition, and phone clustering. [14]

2.6 Gradient Tuning Considerations

An additional way to change how a GAN functions is by altering how the gradients that are used in training are tuned. One of the ways that gradients can be tuned differently than the most basic GAN is the use of techniques such as dropout, the batch-normalization, weight clipping, weight decay, momentum, and various propagation techniques. Dropout is the machine learning practice

of setting some of the input values to 0 throughout the training process in an attempt to mitigate the problems of over-fitting. For a GAN, it is mostly one more tool to help improve the success of training overall. Batch-normalization will attempt to fix the issue of internal covariate shift. This is when the input layers change in distribution from different training iterations. Batch-normalization solves this by shifting around the input layers to make them more stable over training.

Weight clipping is a technique used for the Wasserstein GAN where weights that are deemed to be too big are scaled down in size [15]. With it, training stays stable and within the defined best region for training. Weight decay is used to penalize a model from coming too complex in an attempt to prevent over fitting. This technique is used in a paper on GAN convergence as just one of many common machine learning tools that can be used [16]. Changing weights based on momentum is also possible. This is when gradient decent is being used to optimize the model and we find a direction that is good for a while so the model keeps going in that direction based on the past evidence of momentum. This is also used while testing in the GAN convergence paper [16].

Finally, using various propagation techniques like SGD, Rprop, and RMSprop are also popular. Stochastic gradient decent or SGD is one of the most common methods of marching through many epochs of propagation through the model training. It is used very commonly as no paper in particular is mentioned here. Rprop and RMSprop are just other propagation options that are available to the maker of a GAN.

2.7 Modification of Training Procedure

The procedure in which a GAN trains describes the method, information, and frequency in which each machine in the adversarial environment communicates with the other machine(s). Established by the first simple GAN [5], the G and D alternate training based off of a static number of epochs respectively. When G trains, a completely random sample from a latent space (often Gaussian) and passed through G, where it is then evaluated on how realistic it is via D.

3 Package Design

To achieve the goals for this project, the team set out to make a GAN helper package. The package assists in the creation, execution, and testing of GANs. The team made sure that effective documentation was created, as it will allow the users to effectively understand the package.

3.1 About the Package

The team designed a GAN Package that includes different types of GANs and how they work. These GANs are constructed using PyTorch and Python programming. The GAN package is designed to be used for several purposes, and

multiple target audiences, which will be further discussed in the sections below. There are also multiple documentation files associated with the package.

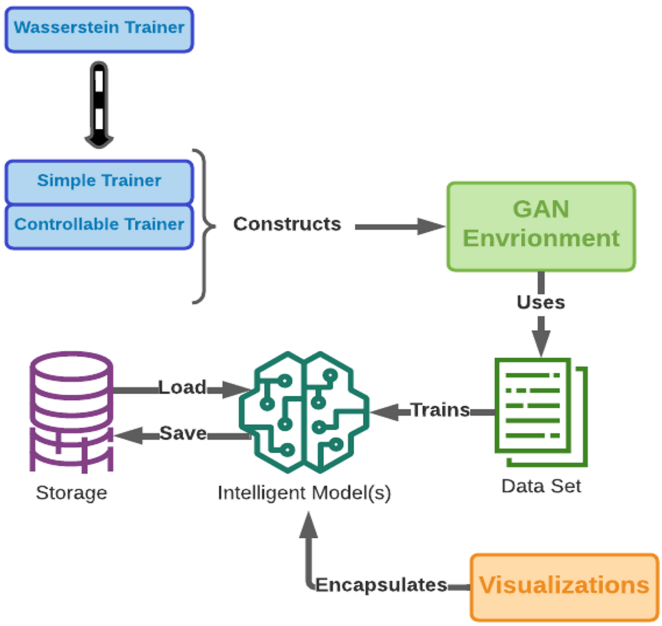

Figure 6: Package Architecture

3.1.1 Resources

The github repository can be found at:
`https://github.com/deoliveirajoshua/pytorch_GAN_Package`

3.1.2 Purpose

The purpose of making the package is to provide the wide developer community with a useful tool to help them make and test their GANs. After gaining experience with GANs, the team knows that implementing them are not always trivial. There are many different ways that they can be implemented. We hope to streamline the process by abstracting the parts that are common for the GANs that our package supports and allowing the user to specify the rest.

We believe that our package can help with those goals by giving users a tool that can be set up relatively easily compared to starting a GAN from scratch. We also have several built in visualizations to show GAN performance. Finally, the package is very usable with the preexisting Python ecosystem. To help with

this goal, we have worked to make good documentation that will help others use the package with minimal difficulty and have made an open source repository on GitHub for anyone to view.

3.1.3 Target Audience

There are multiple sets of people who represent the intended target audience. The package will specifically be tailored to our target audience in order to ensure full usability. One part of the target audience consists of Students, Engineers, and related professionals who have some experience with Python and some experience with opaque Machine Learning. In addition, this group will be familiar with GANs from a conceptual level. They will know how GANs work, but do not have much experience writing GANs themselves. The intentions behind having this group be part of the audience is to help them gain more skills and expand their current knowledge - specifically with how to write GANs themselves. Ideally, those who know the least about the implementation of GANs will gain the most out of using the package. People who have programming experience and GAN familiarity already will not have too much difficulty constructing more complex GANs. However, the more experienced the user is, the less potential gain they will receive from using the package than a fully novel user.

Another group that makes up our target audience is people who have no experience writing GANs at all. These people will gain the experience with how GANs work by specifically working with the GAN package. They will learn more skills with GANs in order to advance their research.

A third group that is part of the target audience consists of GAN subject matter experts. These researchers, developers, and students will have extensive knowledge of working with GANs already. They are looking to further expand their knowledge to write new GANs from scratch. GAN subject matter experts will benefit from the package as they will be able to identify ways to create new GANs from the existing GANs already in the package. They will be able to further enhance creativity during training and testing a model.

3.1.4 Compatibility

To use our package in its ideal state, here are some guidelines for what compatibility restrictions exist:

- Minimum Python Version: 3 (we have used versions 3.6 and higher for testing)
- Packages Used:
  - torch
  - numpy
  - pandas
  - matplotlib

  - scipy

- Must have a 64-bit system

While this is not a comprehensive list, it should give some guidance on what is needed to support our package.

3.2 Package Structure

The basic design goal of the package was to be able to train, save, and manipulate GANs in a simple and intuitive fashion. To this end, the package is designed around an object-oriented framework where the basic unit is the **Trainer object**.

3.2.1 Trainer Object

The Trainer object represents the GAN as a whole - it stores all information relevant to the GAN, and contains functions used to manipulate and train the GAN. The Trainer object stores the models which make up the GAN, the functions used to draw from the data set and the latent space, the optimizers used to train the models, metrics saved while training the GAN, and the **ToTrain object**, which is an object designed to determine which of the generator or discriminator to train on a particular epoch. The Trainer object contains methods used to train the GAN, evaluate either model on specified input, visualize training metrics, and save and load the GAN to disk.

User-Provided Objects When instantiating a Trainer object, the user must provide certain objects and methods. In particular, the user must provide the PyTorch model objects, the optimizers connected to those models, the desired loss functions for each model, a function to draw a random batch from the latent space, a function to draw from the real data, and the device on which training and evaluation should be run.

The models must be valid PyTorch model objects - that is to say, they must subclass *nn.Module* according to PyTorch conventions. Similarly, the optimizers and loss functions must be valid for use with PyTorch models but may be either built-in PyTorch objects or custom user-defined objects.

By definition, the generator is supposed to output data of the same format as the real data. As such, the shape of the generator's output should be identical to the shape of the discriminator's input.

The latent space and data set functions must accept as parameters the desired batch size and the device on which training is done. These functions must output PyTorch tensors on the specified device. The latent space function will output a tensor of shape *(batch_size, generator_input_shape)*, and the data set function will output a tensor of shape *(batch_size, discriminator_input_shape)*.

The device provided to the Trainer object must be a string representation of a device as used by PyTorch. By default, the CPU is represented by `"cpu"`

and the GPU is represented by `"cuda"`, but this may depend on the user's environment.

Training The training loop is the most important part of the Trainer object. Custom Trainer objects may override the default *.train()* function, but any custom implementations should follow this basic structure:

ToTrain Object The ToTrain object determines which model to train during each epoch. At the start of each epoch, the *.train()* function will call the ToTrain object's *.next()* function in order to determine which of the generator or discriminator will be trained. The *.next()* function takes as input the entire Trainer object.

ToTrain objects are designed to be completely interchangeable, while also being as easy as possible for a user to write their own custom ToTrain objects. To that end, the only requirements are that a ToTrain object has a *.next(trainer_object)* function, which returns some representation of the next model to be trained. The *.next()* function receives the entire trainer object as a parameter so that it can read any metric from the Trainer object while being perfectly cross-compatible with other ToTrain objects. The *.next()* function should not modify the Trainer object in any way.

Built-in ToTrain objects return the string "D" if the discriminator should be trained next, and the string "G" if the generator should be trained next. Custom ToTrain objects should respect this convention if they're meant to be compatible with built-in Trainer objects.

The package includes some built-in ToTrain objects such as *TwoFiveRule*, which trains the discriminator for two epochs and the generator for five epochs.

Built-in Trainer objects call their ToTrain object at the beginning of every epoch as follows:

```
tt = self.totrain.next(self)
# Determine which model to train - will either be "D" or "G"
```

Training the Specified Model Once the ToTrain object has determined which model to train, the training loop will then train that model. The Simple GAN Trainer object handles this step in the most simple fashion possible:

If the generator is next to be trained, the Trainer object first generates a batch from the latent space using the user-specified latent space function. The output from this function is fed into the generator to create a batch of fake data, and this fake data is fed into the discriminator. The output from the discriminator is then used to compute the loss, which is then used by the generator's optimizer to update the generator's parameters.

If the discriminator is next to be trained, the Trainer object creates a batch which is composed of half generated data from the generator and half real data from the user-specified dataset function. This batch is fed into the discriminator, and the discriminator's output is used to compute the loss and update the discriminator's parameters.

Training Metrics The Trainer object keeps track of metrics during the training loop. These metrics are stored in a single dictionary, where the keys are the name of each metric. For example, *stats["epochs_trained"]* corresponds to the dictionary {*"G":X, "D":Y*}, where X is the number of epochs the generator has been trained for and Y is the number of epochs the discriminator has been trained for.

Users may define their own metrics in custom Trainer objects, and can call them every epoch by overwriting the *do_viz()* function.

Using the Trained Models After training the generator and discriminator, the Trainer object can be used to perform operations on both models. The models are stored in the Trainer object's *models* dictionary. Built-in Trainer objects use the convention of *models["G"]* referring to the generator and *models["D"]* referring to the discriminator. Custom Trainer objects may use whatever convention makes sense for their intended use, but different conventions will not be compatible with built-in ToTrain objects.

Evaluating the Models The models can be evaluated on given input by using either of the Trainer object's two built-in evaluation functions, or by accessing the models directly through the *models* dictionary.

```
# output from trained generator
print(gan.eval_generator(lat_space(16, device)))

# Equivalent to:
print(gan.eval("G", lat_space(16, device)))

# Or:
gan.models["G"].eval()
print(gan.models["G"](lat_space(16, device)))
gan.models["G"].train()
```

Changing Device The Trainer object automatically does all operations on the specified device, including training and evaluation. This device can be changed during runtime using the *.models_to(new_device)* function.

Training Modes Trainer objects have functionality to support different training modes, which are designed to impact the manner in which GANs are trained and metrics are collected. Currently, the only training mode currently implemented is Wasserstein training.

Wasserstein Trainer objects in Wasserstein mode discard the user-provided optimizers and replace them with built-in optimizer objects designed to train the GAN based on Wasserstein distance.

Saving and Loading Trainer objects support saving and loading their state to or from disk. The `.soft_save(PATH)` and `.soft_load(PATH)` functions save and load the Trainer object respectively. These functions use Pickle to save most of the Trainer object's parameters, and use PyTorch's built-in saving functionality to save the states of the models and optimizers (PyTorch's saving functionality also just uses Pickle). Because these functions use Pickle, successfully loading a Trainer object checkpoint from disk requires that any custom objects are visible in the current scope.

3.3 Testing Protocol

In order to effectively test our GAN Package, we will investigate all possible use cases that a member of our target audience might have and focus the primary effort on testing those. Then, we can test all related edge cases and other less common activities. For our testing, we will then be focusing on making GANs, training GANs, evaluating their results, saving and loading models from disk, and visualizing their results. These broad idea encompass all main tasks a user might do and is thus as comprehensive as possible.

4 User Study

In order to actually serve the target audience, the package does not just need useful functionality but also simple and clear documentation. We created documentation in the form of a tutorial in order to describe the core functionality of the package and how to use its features.

By being the author's of the package, we were too familiar with how it worked to accurately judge the effectiveness of the tutorial. In order to evaluate the tutorial and find areas to improve, we conducted a user study composed of a coding challenge and a survey.

We sought out seven participants with solid python experience and varying levels of experience with deep learning and GANs. We had these users complete a coding challenge designed to test knowledge of the package's basic functionality, and complete a survey about how they did. To ensure that our study was done properly, we applied for and received approval for our study from the Institutional Review Board. A copy of this approval can be found in Appendix A. The study in full can be viewed in Appendix B.

4.1 Coding Challenge

The coding challenge was designed to test whether the tutorial was able to give a user a general understanding of the package's functionality. In particular, our goal when designing the challenge was to test a user's ability to use functionality described in the tutorial in a manner not described in the tutorial. To complete the coding challenge, a user would need to be able to apply skills taught by the

tutorial but the user would not be able to solve the challenge simply by directly copying code from the tutorial.

The core premise of the challenge was to use a Trainer object to create a simple GAN and train the GAN in multiple steps. The user was required to perform various operations during training, such as viewing built-in visualizers and saving a checkpoint of the model. These skills were demonstrated by the tutorial, but not in the exact fashion used in the challenge, so to successfully complete the challenge a user would need to be able to generalize the information presented in the tutorial into a working understanding of the package. If the tutorial was effective at teaching how to use our package, then all the participants should be able to accomplish this. A copy of the coding challenge can be found in Appendix B part 4.

5 Results

Through the process of making our GAN helper package, we have produced several useful resources as well as data from our tests and user study that support the package.

5.1 Testing Results

From our testing, we have determined that the package works as intended when used for its use cases. To perform this testing, we wrote various testing files of an example program a user might make to test their potential actions. These files are `SimpleGANTesting.py`, `SimpleGANSaveTesting.py`, `WassersteinGANTesting.py`, and `ConditionalGANTesting.py`.

Our testing process allowed us to find and correct bugs during the development process, and we can now confidently say that all aspects of our package work as intended.

5.2 User Study Results

Our team was able to compile results for seven different users. These users were able to work with the provided documentation. In terms of making the documentation, our team provided the users a README.md file that lays out the different steps that the users need to understand. Our documentation first talks about the SimpleGANTrainer.py file and what the purpose is behind it. Next, our documentation talks about the initial setup for the users. The initial setup is crucial in order for the trainer object to be created. After the initial setup, we show the user the necessary environment setup, including the proper imports, in order to successfully design the GAN. Next, our documentation includes the generator and discriminator models, optimizers and loss functions, and latent space and dataset functions. After showing the different pieces of code, our documentation describes the purposes behind making the ToTrain object and the discriminator positive threshold. Next, out documentation includes instructions

on how to create the trainer object, how to train the GAN, and how to evaluate the different models. Our documentation also includes instructions for saving and loading, as well as loading a checkpoint.

5.2.1 Data on our Participants

At the start of our study, we ask for some ratings that the user felt about their skills in related areas to the. The full data of our user study can be found in Appendix C. First we asked for people's python experience level. Having some knowledge of the language is a prerequisite of using our package. As shown by Figure 7, every user has at least some python experience so they should be minimally prepared.

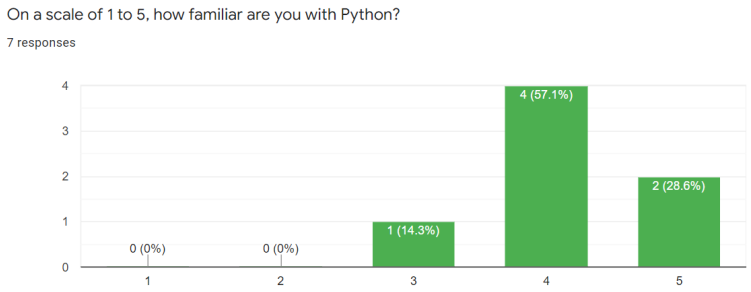

Figure 7: Python Experience Question

Next, we asked how familiar they are with PyTorch. It the main package we use to do our machine learning. As shown by Figure 8, many users were not familiar with it beforehand. This knowledge is not as important for the tutorial so this is not much of a factor.

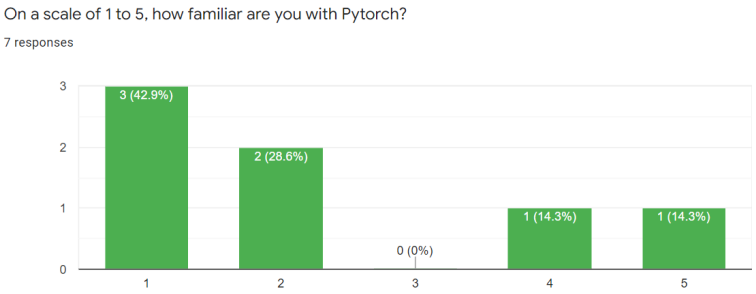

Figure 8: PyTorch Experience Question

Following this, we asked for experience level with neural networks. These are the building blocks of GANs so the user will get much more out of the tutorial

with this knowledge. This was much more split, as shown by Figure 9, so we have a wide variety of users in this regard.

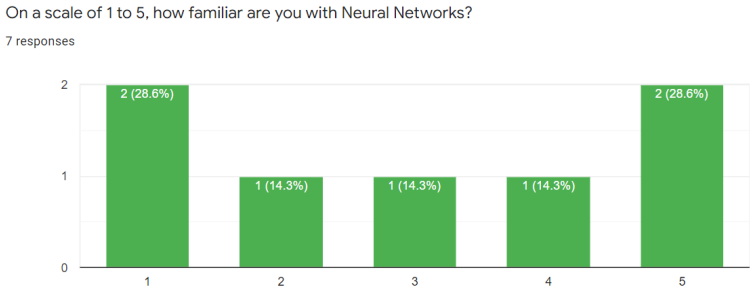

Figure 9: Neural Networks Experience Question

Finally, as this is a GAN centered package, we thought it would be good to ask their own experience with GANs. As shown by Figure 10, most did not know much about the topic which makes them our ideal target audience. The ones more familiar come from the expert users we also tested on.

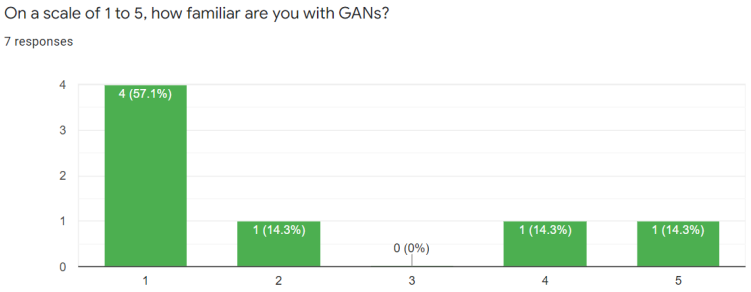

Figure 10: GAN Experience Question

5.2.2 Tutorial Feedback

After reading the tutorial and doing the programming task, we ask post-study questions to gauge understanding and performance. In terms of data, we asked several multiple choice questions. First, we asked on how long it took to read and understand the tutorial. It is crucial that this is not a huge time sink as we want it to be easy to read and follow. Luckily, a majority of respondents took the minimal time range of less that 15 minutes. The rest took 30-60 minutes. This is within an acceptable time expectation for this task.

Next, we asked for how long the programming task took. Being more involved than reading the tutorial, we anticipated it taking longer for users. It is good to see that a majority of users took 30 minutes or less to complete it.

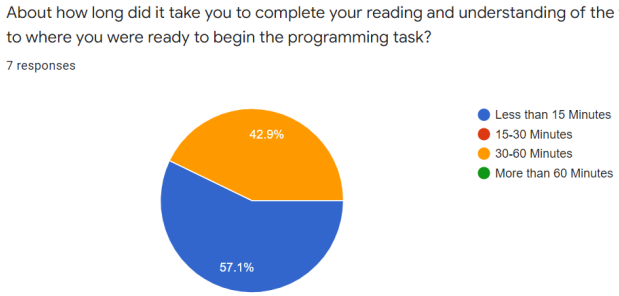

Figure 11: Time Taken Reading the Tutorial

Most of the rest took the net highest time rank which is still fair with only a single user taking over an hour. This user encountered many issues that we have addressed to hopefully prevent their troubles from happening again.

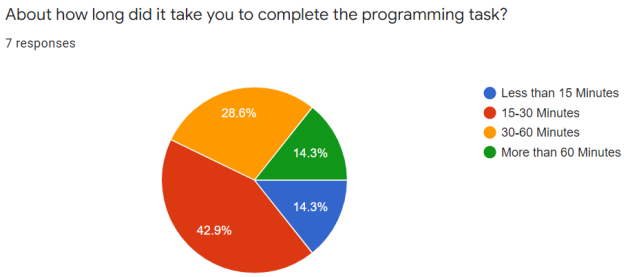

Figure 12: Time Taken Doing the Programming Task

Finally, we asked for a overall rating of the ease of following our tutorial. As this was the main objective of our test, this acts as a holistic rating of our efficacy. No one gave a perfect 5 which makes sense as our project is still being worked on and needed their feedback. However, everyone gave a 3 or 4 which are both positive scores. Though every user encountered issues, it is a boost to the project confidence that they still thought highly of the tutorial regardless.

5.2.3 Tutorial Changes

After the multiple choice questions, we provided open ended questions to give participants more room to express their opinions. Through this, we received helpful feedback on what was done well and what issues need to be fixed. In general, users were able to complete the programming task, but some users found sections of the tutorial which were poorly worded or unclear when describing how to accomplish specific tasks.

The feedback we received, and the actions we took to improve the tutorial,

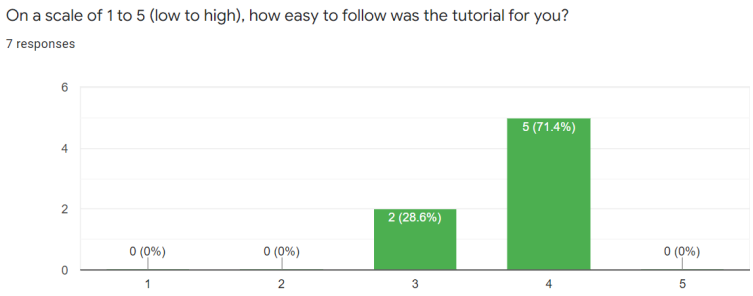

Figure 13: Tutorial Rating

was:

- The tutorial did not clearly show how to load a checkpoint from a newly instantiated Trainer object. To fix this, we updated the example in order to demonstrate this behavior.
- The tutorial did not clearly show how to use visualisers or explain what the visualiser functions do. To fix this, we elaborated on the visualiser section.
- The tutorial did not clearly show how to instantiate ToTrain objects. To fix this, we added an example showing how to do this.
- The list of dependencies incorrectly omitted SciPy, which caused the package installation process to fail to install SciPy in python environments which did not already have SciPy. We fixed this by adding SciPy to the list of dependencies.
- Trainer object checkpoints did not write files to the correct folder on Linux systems. We fixed this by correcting the format of the auto-generated filenames.
- Finally, we fixed several typos and wording problems.

While this was not every error found in the tutorial testing, it covers a majority of the major issues found. All not addressed are mentioned in the future works section.

6 Conclusion

Overall, we thought that the project was a valuable way to learn more about GANs and teach others how to work with them. In addition, we are proud of our accomplishments while learning about GANs in a relatively short period of time. In addition, we were able to stay on track with our timeline every

week, despite challenges faced along the way. We have also been able to learn a significant amount of information from each other. The ability to adapt has been crucial for our success in this project.

6.1 Future Work

Though the package contains all of its core features, there are many features that could still be added to improve it. Also, there is still work to be done in increasing its accessibility to a wide audience. Future work could go towards fixing these problems by adding new features. In particular:

- Currently the only two built-in Trainer objects are designed for Simple GANs and Conditional GANs. Future work could add more built-in Trainer objects, such as a Controllable GAN, designed to work with a much wider variety of GAN types.
- The array of built-in ToTrain objects is currently very bare. Future work could add more built-in ToTrain objects.
- The user has very little control over how long the GAN should be trained - currently, the user can only specify number of epochs. Future work could add different ways the user could control training length, such as performance benchmarks.
- Currently, built-in Trainer objects do not keep track of very many metrics. Future work could add to the list of natively supported metrics.
- Currently, the only feature which makes use of the Training Modes feature is Wasserstein Training. Future work could add additional training modes.
- There is not currently any cohesive guide for users planning on creating custom Trainer objects. Though the information required is present in this document, future work could gather it all in once place, ideally in the form of a guided tutorial.
- Currently, the only documentation of the package is the tutorial. A more traditional documentation of all the objects and functions in the package would help users make the most use of our package.
- User study feedback not addressed:
  - One user suggested making the tutorial have more payoff at the end on top of the generated graphs. This is a great thing to add, though we did not have time to make this a reality.
  - Another user recommended using the GitHub wiki tool to display the tutorial rather then the ReadMe we used. This could be great to look into, however we again did not have time to see it through

7 Individual Contributions

7.1 A Term

Kyle implemented simple and controllable GANs. In addition, he found multiple articles related to Divergence, GAN Strategies, and Training Procedures. Kyle also effectively communicated as the team leader and explained the team's goals during individual and advisor meetings. Kyle took initiative and scheduled team meetings as needed. Kyle helped the team troubleshoot errors and provided solutions that completed the GAN implementations. Kyle wrote the Introduction and the Introduction subsections, and sections of the background including Divergence Based Loss (Section 2.5), and part of Modification of Training Procedure (Section 2.7). In addition, Kyle wrote about the f-Divergent Generator in the methods (Section 3.2).

Ryan also implemented a simple and controllable GAN. He helped write notes during each of the advisor and individual meetings. Ryan created several figures that have been added to the paper. Also, he organized meetings with the team, made outlook invites for said events, and wrote or assisted in writing several sections, including 2.6, 2.3.1, and 3.3.

Alek aided with the literature review by finding research papers on GAN architectures, training procedures, and generative metrics. Alek wrote sections of the background (part of 2.2, parts of 2.3, and 2.4) and edited the introduction. Alek proposed a novel training strategy, Multiple Discriminators, and wrote the section describing it (3.1) and created the diagram in that section.

Josh performed a literature review of research done in order to improve GAN training. In doing so, organized GAN research into 4 key facets: improving the architectures, improving the loss functions, incorporating more meaningful regularizations, and improving the order in which parts of the GAN train. Also, Josh proposed a novel training strategy coined *dynamic freezing* detailed in section (3.3). Josh also assisted in introducing simple GANs and its well respected successors (conditional, controllable, explicitly controllable) to the rest of the team, as well as help improve understanding of these modified approaches in order for the team to have a good foundation for the rest of the MQP. Josh wrote or assisted in writing sub-sections in the Introduction (prelude, 1.1, 1.2, 1.6), Background (prelude, 2.1-2.5) and Methods (3.3) chapters. Lastly, Josh designed and created figures 1-5 and 7-11, as well as wrote equations 1-15.

7.2 B Term

Kyle implemented the Wasserstein GAN. The Wasserstein GAN involved modifying the Simple GAN. Specifically, Kyle was able to graph the Wasserstein distance over time as the number of epochs increases. Kyle also worked on documentation for the Wasserstein GAN. In addition, Kyle took notes during the entire term. The notes taken during every meeting helped the team stay on track to meet the project goals over the course of the timeline.

Ryan implemented the code that proves the ability for the package to sup-

port the running of a Conditional GAN. This is achieved by using the Simple GAN file and giving it extra input via the labels while also changing the passed in generator, discriminator, latent space, and sample from data functions. They also gathered and cleaned two data sets to use to test the efficacy of the Conditional and Controllable GANs. They also wrote code to record the Wasserstein distances per epoch per class and wrote a new visualization to show that data if asked. Finally, the wrote sections new sections for the paper specifically on the package purpose (4.1.1) and package compatibility (4.1.3)

Alek implemented the SuperTrainer base class and the SimpleGANTrainer class. The SimpleGANTrainer trains simple GANs with a focus on allowing the user to customize their own GAN as much as possible, making as few assumptions as possible about the structure of the data or the models themselves. This customizability was key for allowing conditional GANs to be implemented using the SimpleGANTrainer. Alek also implemented the save/load feature, the custom device training feature, and many of the present visualizations. Finally, Alek wrote the tutorials for how to use the package (4.3.2).

Josh implemented the code that proves the ability for the package to support the running of a Controllable GAN. This is achieved by writing a ControllableGANTrainer class that extends from SuperTrainer (similar to the simple and wasserstein GAN trainer objects). ControllableGANTrainer allows pre-trained classifiers to be incorperated into a generator-discriminator GAN environment, and train on multi-class tabular data sets. Josh also assisted in visualizations for GAN evaluations, as well as debugging and design in the conditional and wasserstein GAN implementations in the SuperTrainer architecture. Finally, Josh wrote sections in Package Structure (4.2.0) and User Pipeline (4.2.1)

7.3 C Term

Kyle worked on the documentation to make sure it is easy to understand for the target audience. In addition, Kyle worked on the poster that will be used during Project Presentation Day. For the study, Kyle helped to gather participants and gain more outreach to graduate level students taking Deep Learning classes. Kyle also helped on determining the study questions. Kyle tested the coding challenge and was able to identify issues that were addressed before sending out the study to participants. He also worked on rewriting the paper and writing specific sections, including information about the tutorial/documentation, user study results, and findings from the reflection as part of the conclusion.

Ryan worked on testing the package to ensure that it was in full working order through various testing files on all of the main components that a user might work on. He also helped make the questions for the user study that was sent to participants. For the study, he also tested the coding challenge and filter the feedback received into executable tasks. In addition, Ryan served as the team leader this term by keeping the team on task at all times and coordinating efficient work across the team. Finally, he contributed sections the the paper about testing done on the package and helped write those about the user study, and proofread the rest.

Alek worked on writing the tutorial explaining how to use the package. Alek also designed the coding challenge for the user study, and revised the tutorial based on feedback from the team. Alek helped fix bugs in the package code and refactored the package to simplify its structure. Alek contributed to the paper by writing sections 3.2, 4, and 5.2.2, and by helping reorganize the structure of the paper.

Josh was unable to participate on the project this term for medical reasons so he has no contributions to list here.

References

[1] Yang Zhao, Chunyuan Li, Ping Yu, Jianfeng Gao, and Changyou Chen. Feature quantization improves gan training, 2020.

[2] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2017.

[3] Erik Linder-Norén. Keras-GAN, 2021.

[4] Erik Linder-Norén. PyTorch-GAN, 2021.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[6] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[7] Minhyeok Lee and Junhee Seok. Controllable generative adversarial network. *CoRR*, abs/1708.00598, 2017.

[8] Alon Shoshan, Nadav Bhonker, Igor Kviatkovsky, and Gerard Medioni. Gan-control: Explicitly controllable gans. *arXiv preprint arXiv:2101.02477*, 2021.

[9] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks, 2017.

[10] Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Tempered markov chain monte carlo for training of restricted boltzmann machines. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 145–152, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[11] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[12] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible conditional gans for image editing, 2016.

[13] Sebastian Nowozin and Botond Cseke. f-gan:training generative neural samplers using variational divergence minimization. *CoRR*, abs/1606.00709, 2016.

[14] Wikipedia contributors. Bhattacharyya distance — Wikipedia, the free encyclopedia, 2021. [Online; accessed 13-October-2021].

[15] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[16] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.

8 Appendix

8.1 Appendix A: IRB Approval Letter

WORCESTER POLYTECHNIC INSTITUTE

100 INSTITUTE ROAD, WORCESTER MA 01609 USA

Institutional Review Board
FWA #00030698 - HHS #00007374

Notification of IRB Approval

Date:	04-Feb-2022
PI:	Elke Rundensteiner A
Protocol Number:	IRB-22-0367
Protocol Title:	GAN Trainer Package tutorial effectiveness
Approved Study Personnel:	Astor, Ryan C~Costello, Kyle A~Lewis, Alek J~DeOliveira, Joshua C~Rundensteiner, Elke A~
Effective Date:	04-Feb-2022
Exemption Category:	3
Sponsor*:	

The WPI Institutional Review Board (IRB) has reviewed the materials submitted with regard to the above-mentioned protocol.We have determined that this research is exempt from further IRB review under 45 CFR § 46.104 (d). For a detailed description of the categories of exempt research, please refer to the IRB website.

The study is approved indefinitely unless terminated sooner (in writing) by yourself or the WPI IRB. Amendments or changes to the research that might alter this specific approval must be submitted to the WPI IRB for review and may require a full IRB application in order for the research to continue. You are also required to report any adverse events with regard to your study subjects or their data.

Changes to the research which might affect its exempt status must be submitted to the WPI IRB for review and approval before such changes are put into practice. A full IRB application may be required in order for the research to continue.

Please contact the IRB at irb@wpi.edu if you have any questions.

8.2 Appendix B: User Study Questionnaire

8.2.1 Informed Consent Agreement for the User Study

Investigator: Professor Elke Rundensteiner (PI), Ryan Astor, Kyle Costello, Josh DeOliveira, and Alek Lewis
Contact Information: `gr-GAN-MQP-2021-2022@wpi.edu`
Title of Research Study: Participation in GAN Trainer Package tutorial effectiveness
Introduction:

You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks, or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.
Purpose of the study:

Our MQP project is focusing on generative adversarial networks or GANs. This is a special kind of machine learning that involves two or more neural networks competing with each other. In the simplest case, one is a generator that makes fake data and the other is a discriminator that attempts to tell the difference between the fake data made by the generator and some real data provided. The end goal is to create machines capable of making believable fake data and this is useful in various areas of research and industry alike. The goal of this study is to assess the effectiveness of the tutorial we created for our GAN python package that helps newer users of GANs get some experience with the fundamentals of creating them.
Procedures to be followed:

For this study, you will first answer a few initial questions to assess your background. Then you will read over our tutorial on the package. Then, you will do a short programming task to assess your understanding of the tutorial. Finally, you will answer some follow-up questions to reflect on your experience in the study.
Risks to study participants:

We do not foresee any risks to the user from participating in this study.
Benefits to research participants and others:

There are no benefits for the user from participating in this study.
Record keeping and confidentiality:

The data collected for this study will be done via google forms. The only ones who will have access to these records are the study personnel listed above as investigators. Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators, the sponsor or it's designee and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury:

There is no expected possibility of injury involved in taking part in our study. You do not give up any of your legal rights by signing this statement.

Cost/Payment:

Participants who completed the study will each receive a $20 gift card.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact:

The MQP Team: `gr-GAN-MQP-2021-2022@wpi.edu`

IRB Manager (Ruth McKeogh, Tel. 508 8316699, Email: `irb@wpi.edu`)

Human Protection Administrator (Gabriel Johnson, Tel. 508-831-4989, Email: `gjohnson@wpi.edu`)

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

Virtual Signature:

8.2.2 Pre-Study Questionnaire

Please answer the following before starting our tutorial

The compensation of the gift card listed above is for the full completion of this study. For compensation to occur, some evidence of actually participation in the full study is needed by your responses. This is mostly to discourage potential participants from giving little to no effort and expecting the gift card in return. If you do actually try, then there is no reason to think you would not qualify for the gift card.

Please give your email if you would be ok with potential follow questions and to be reachable for the receipt of the gift card

1. On a scale of 1 to 5, how familiar are you with Python?
2. On a scale of 1 to 5, how familiar are you with Pytorch?
3. On a scale of 1 to 5, how familiar are you with Neural Networks?
4. On a scale of 1 to 5, how familiar are you with GANs?

8.2.3 Tutorial Section

Please read our package tutorial. It can be found here in the ReadMe:
`https://github.com/deoliveirajoshua/pytorch_GAN_Package`
Move to the next section when you have read and understand our tutorial

8.2.4 Programming Task Section

Now that you have read and understand the tutorial, Please complete the following programming task. Once you think you are done, please upload the code you wrote. A google account is required for the file upload.

Task for you to complete:
Using the minimal GAN demonstrated in the tutorial:

1. Train the GAN for 100 epochs
2. At epochs 20, 60, and 100, graph the discriminator's loss by epochs
3. At epoch 60, save the trainer
4. Load a new model from the saved checkpoint, and train it for another 60 epochs. After training the new GAN, graph the generator's loss by epochs

Starter Code:
`https://drive.google.com/file/d/10VF1EwLtejyr19P05Pc3LUjWjUb9EXe0/view?usp=sharing`

8.2.5 Post-Study Questions

By now you have finished reading the tutorial and completing the programming task. Please answer the following questions about your experience in this study

1. About how long did it take you to complete your reading and understanding of the tutorial as to where you were ready to begin the programming task?

   Less than 15 Minutes

   15-30 Minutes

   30-60 Minutes

   More than 60 Minutes

2. About how long did it take you to complete the programming task?

   Less than 15 Minutes

   15-30 Minutes

   30-60 Minutes

More than 60 Minutes

3. On a scale of 1 to 5 (low to high), how easy to follow was the tutorial for you?

4. Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

5. Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

6. Do you have any suggestions as to how we could improve the tutorial?

8.3 Appendix C: User Study Responses

8.3.1 Pre-Study Results

Participant	Python Exp	Pytorch Exp	NN Exp	GAN Exp
1	5	2	4	2
2	4	1	2	1
3	4	1	3	1
4	3	2	1	1
5	4	4	5	4
6	4	1	1	1
7	5	5	5	5

8.3.2 Post-Study Tabular

Participant	Tutorial Time	Task Time	Rating
1	Less than 15 Minutes	15-30 Minutes	4
2	Less than 15 Minutes	15-30 Minutes	4
3	30-60 Minutes	30-60 Minutes	3
4	30-60 Minutes	15-30 Minutes	3
5	Less than 15 Minutes	Less than 15 Minutes	4
6	30-60 Minutes	More than 60 Minutes	4
7	Less than 15 Minutes	30-60 Minutes	4

8.3.3 Post-Study Open Ended

Participant 1:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

Yes.

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

There were no places in particular that required excessive re-reading, however the wordiness of the tutorial made it difficult to re-find information.

Do you have any suggestions as to how we could improve the tutorial?

I would recommend utilizing GitHub's wiki to properly document each function of the object, so that the tutorial / example code only needs simple explanations for each line of code. This would make it quicker to process the tutorial, and specifics of each function could be looked up as-needed. This would also make it easier to publish more examples.

Participant 2:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

SimpleGANTrainer complained that scipy wasn't installed, which wasn't stated in the tutorial. It worked fine after installing it.

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

I ended up having some trouble with pytorch and matplotlib because I'm running Ubuntu 20.04 on WSL2, but that's not really your problem. I eventually gave up on trying to get my GPU to work with pytorch, and got matplotlib to work with my X11 server using the python3.8-tk package from apt. At that point everything went fine.

I have very little experience with deep learning and its terminology, but I was able to figure out what most terms in the tutorial meant from context. The only one I never got was what a "dense layer" was, but it wasn't necessary to understand to complete the tutorial."

Do you have any suggestions as to how we could improve the tutorial?

I noticed that 'SimpleGANTrainer.soft_save' saved several files. I assume these are supposed to be in a separate folder with the name I gave, but they got saved as files named like "name\D.pt" with the backslash in the filename. They still happened to load fine, but this probably isn't the behavior you're looking for.

There are some minor formatting problems with the code in the tutorial. Some of the code blocks have inconsistent indentation, comments with inconsistent capitalization, and some double quotes look like they got turned into smart quotes.

I wish there was more of a payoff in the tutorial like being able to give some numbers to the discriminator manually to see that it's getting better at telling which ones are even. I recognize that the loss-by-epoch graphs are probably supposed to be this payoff, but they're meaningless to me as someone without prior deep-learning knowledge.

Participant 3:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

Confused on what the PATH should be, PATH to the project or any path on the file system. Also, the tutorial didn't have a gan.soft_load in the second example if you wanted a new object.

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

Confused by the .loss_by_epoch(model_name) function. I saw that "D" and

"G" were options but an example used gan.loss_by_epoch_d() which wasn't mentioned in the tutorial.

Do you have any suggestions as to how we could improve the tutorial?

Confused on why you asked for only discriminator loss for the first 3 training and only generator's loss for the loaded file. Also there was no requirement on downloading scipy which was an error I had when I ran the file for the first time even after installing all the packages including the pytorch_GAN_package.

Participant 4:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

Yes, I think it gave me all the information I needed to complete the tasks.

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

I had two small issues. One was that SciPy is a package that is used in the SimpleGANTrainer file but apparently didn't get installed when I ran the pip command, so I just installed it myself and it was fine after. Also, you should make it more clear what PATH is for soft_save and soft_load or at least what it could be.

Do you have any suggestions as to how we could improve the tutorial?

Not really other than make sure all your code example are correct. I think you're missing some key lines in some of your last examples (like evaluating on a different device). Also, you talk about ToTrain objects but don't show any example of them. It didn't impede my ability to do the tasks but it felt weird to not see code for it.

Participant 5:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

Yes - in part. I was able to write code that ran, but I do not know if the visualizations were generated anywhere. The tutorial says that the function call will "show" a visualization. Show in what sense? Should I be running this in a Jupyter notebook or something? Just running as a .py file throws no errors but I do not see where the visualizations were saved.

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

No - other than the visualization thin

Do you have any suggestions as to how we could improve the tutorial?

Some notes:

on google form, under "programming task section": please upload the cod you wrote. → please upload the code you wrote

no indication in the tutorial what it means to plot visualizations - does it save a png of a plot? where does it save it? Is it overriding the previous visualization every time I save it? I don't see the visualizations anywhere

Error message, requiring installing scipy: ModuleNotFoundError: No module named 'scipy'

I tried to save the parameters of the gan as "trained_gan". this made the system create an empty directory, and then created a bunch of files OUTSIDE

OF the directory titled "trained_gan\ ~whaterver~". Maybe this is a bug that will only arise on linux system?

Participant 6:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

No, it was unclear about a few lines of code regarding instantiating the trainer object. First, it was not clear that the variable titled "GAN_DEVICE" was supposed to be the input for the device parameter. Second, the sw input was never specified and I only found it from the programming task. You do talk about the two-five rule but it was unclear that sw was the two five rule and there was no mention of the twoFiveRule function in the tutorial. Also, I had to install the "scipy" package to use the code. When running the program, I got a ValueError that said the torch.Size([16, 1]) was not equal to torch.Size([16,7]).

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

I had to spend time at the trainer object because I didnt know what "sw" or device were.

Do you have any suggestions as to how we could improve the tutorial?

Add in the scipy and math libraries as packages you need to run the program. In the tutorial, it should say "import math" under the imports section since we used that library.

Participant 7:

Did the tutorial give you all the information you needed to complete the tasks? If no, why not?

A basic minimum, not sure how it would fare when people want to do something more than the simple tutorial style workflow.

Was there any point in reading the tutorial where you got stuck or had to spend an extended period of time figuring it out? If yes, please explain.

yes, the last part wasn't clear how to properly load and re start training

Do you have any suggestions as to how we could improve the tutorial?

make things more explicit and technical; maybe include a brief section about the terminology of gans and how they work.