Major Project Stage 2

GAN Studio

Software Design and Development

Matty Hempstead

Mr McFarlane

6/7/18

1. User Documentation

1.1. Installation Guide

1.1.1. Requirements

GAN Studio is built on HTML, CSS, and JavaScript, and will thus run in most modern browsers. It will install and run on any modern version of Windows (7/8/10). At the time of writing, the program is compatible Chrome (V67), Firefox (V61), and Edge (V42).

It is strongly recommended however that one uses the fastest browser available to them, unless performing quick tasks like generating images. The performance increase for example when switching from Edge to Firefox usually more than 3x, which can have a major impact during long training sessions. As training is an intensive process, it will quickly use battery if performed on a laptop and may suddenly run out of charge causing all progress to be lost. It is therefore recommended that you install on computers connected to a continuous power supply for extended training periods.

1.1.2. Benchmarks

To give you an idea of the training speed of each browser in GAN Studio, I have conducted a simple speed test of the three compatible browsers with default GAN settings. *The benchmarks were performed on a single Windows 10 Desktop and may vary between users.*

Training speed per browser:

- Firefox: ~2300/s
- Chrome: ~1500/s
- Edge: ~700/s

1.1.3. Installation

All that is required to run the program is to open /index.html with one of the compatible browsers, and to make sure JavaScript is enabled.

1.2. User Manual

1.2.1. Introduction to GANs and ANNs

1.2.1.1. Why am I telling you this?

Before going into depth about how to use GAN Studio to create GANs, I felt it was best to give a brief introduction into how GANs actually work. Having a basic understanding of what's happening underneath will make the creation of effective

GANs a more achievable task, as you will understand the effects of tweaking particular settings on the performance of a neural network. I have also provided explanations on the effects of altering the network parameters accessible to you, as a user, in GAN Studio. As the original aim of GAN Studio was to increase my understanding of neural networks, having to write down all my gained knowledge in a logical and sequential manner also reinforces my completion of the goal.

1.2.1.2. Understanding ANNs

While I do expect you to have some understandings of ANNs already, I have provided below a my own simplified explanation to act as refresher on the topic. I highly recommended reading how others have explained ANNs, as they will inevitably be more thorough, and provide different, and better interpretations than my attempt below.
If you wish to have a mathematical understanding of ANNs or perhaps even make your own, this excellent youtube series by 3Blue1Brown goes into great depth on the math of ANNs and gradient descent/backpropagation.

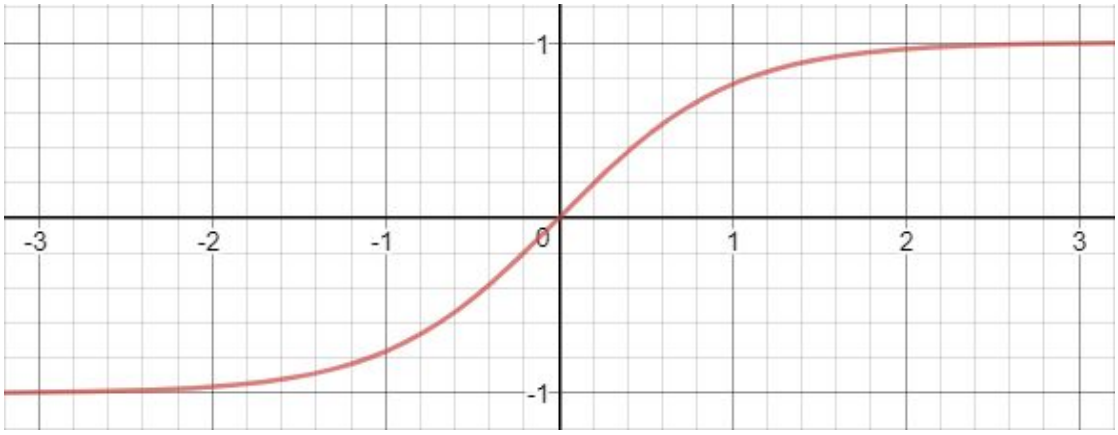
- https://www.youtube.com/watch?v=aircAruvnKk

1.2.1.3. What is an ANN?

Artificial Neural Networks (ANNs) can be thought of as universal function approximators, inspired by the inner workings of the brain. Essentially, they take a list of numbers as inputs, perform some computation, and output another list of numbers that we can interpret in a meaningful way for completing a particular task. ANNs consist of an input layer which receives the list of input numbers, hidden layer/s which essentially perform the computation, and an output layer which creates the final list of numbers. Each of these layers has a certain number of neurons (similar to the brain's' neurons), and the layers are connected through weights (like the synapses connecting real neurons).

1.2.1.4. Computation

The “computation” that occurs between layers, involves the sum of “weighted” values from each of the neurons in the previous layer, which are then passed through an activation function and fed into the next layer. The activation function essentially squishes the sum between two known values (usually -1 to 1 or 0 to 1) through some non-linear process, which allows this “function approximator” to be much more complex than a simple linear function. The activation function used for ANNs in GAN Studio is hyperbolic tangent, while this isn’t necessary for your usage of the program, it is good to know how these “squishing” functions look and work if you were to create your own ANN.



The hyperbolic tangent activation function used in GAN Studio. Inputs are squished between -1 and 1, with the quickest change in value occuring around x=0.

1.2.1.5. Choosing Layer Count

As these nonlinear functions are stacked with each additional hidden layer, the more layers, the more complex the ANN can become. Having many layers also means the network is harder to train however, as it is deeper and thus the training of “weights” which I will discuss later is less direct towards the desired output. For simple handwritten digit image recognition/generation, like that present in GAN Studio, 1 to 2 hidden layers will often suffice for the given complexity of the task. It is only when you begin creating chess bots better than any human alive that the task starts to become complex enough for dozens or even hundreds of layers to be required. Each new layer also decreases training speed quite considerably, especially when having so few layers that adding a single layer doubles the number of hidden layers that need to be trained.

1.2.1.6. Choosing Neuron Count

Instead of altering the number of hidden layers, you can also choose to change the number of neurons per layer. Just as the number of layers can be thought of as increasing the depth of the function approximator, the number of neurons per layer is somewhat like the scope of the ANN. Having more neurons allows the ANN to approximate a wider range of input/output possibilities with a higher accuracy as there are more neurons/weights to store and process information regarding the computation.
Layers size also affects computational speed, although not as much as changing the layer count, as neurons are already looped through hundreds of times so adding a few more or less from GAN Studio’s default of 16 shouldn’t affect the speed too much. Remember however that each neuron is connected with individual weights to all other neurons in the previous, and succeeding layer. This means that having many neurons in hidden layers, such as a single layer of 128 will require *(784+1)*128 + (128+1)*1 = 100609* weights compared GAN Studio’s default 2 layer

network with 16 neurons in each containing *(784+1)*16 + (16+1)*16 + (16+1)*16 + (16+1)*1 = 13121* weights.
Too few neurons however and the network may be unable to keep track of the required information for computing an accurate result. A careful balance should thus be found between the depth of an ANN, and it's width, and while you should take this all into account when choosing the structure of your networks, there are default settings that produce reasonably effective GANs which are provided in GAN Studio.

1.2.1.7. Training ANNs

Creating the neural network with your set number of layers and neurons per layer, is simply the first step in a much larger problem. To create your universal function approximator you will need to train it to accurately reflect the data you wish for it to learn (the process for effective training I will go into detail later in the user manual). This data to replicate is known as the training samples (sometimes "training data", or simply "data" as often referred to in GAN Studio), and must also be created carefully (I will also go into detail about this process later in the user manual). The process of training is completely automated by GAN Studio, as you will only need to provide the initial settings and training data before letting the program take over.

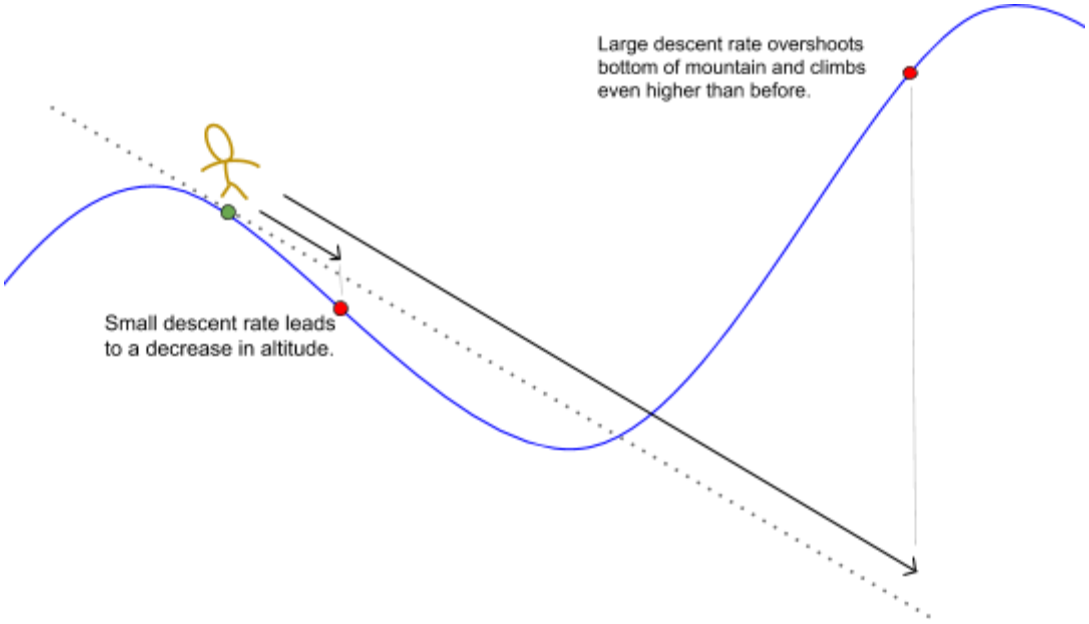
Training an ANN, means teaching the network to produce a particular output given a particular input. As the process is quite complex, and involves some calculus, I will not be providing a full explanation however the process is essentially as follows…

1. Feed the network your input, and keep track of how it produced its output. Before training, this output will be essentially random.
2. Using magical calculus (*some simplification has been made here*), start from the final layer and work your way backwards, finding the rate at which each individual weight affects the output of the network relative to the output you are trying to replicate. This process of finding the effect of weights on output error is known as backpropagation.
3. Based on this "gradient" of output error to weight value, attempt to reduce the error by descending down this line by a tiny set value (learning rate). This process is thus known as "gradient descent".
4. Repeat lots of times for lots of varying data, until the network can accurately produce a desired output given a particular input.
5. Hope that when new inputs that the network hasn't seen before are provided, the network outputs a value that reflects the training data which it had been fed.

1.2.1.8. Choosing Learning Rate

The learning rate affects the rate at which the ANN descends the gradient of error for each weight. You may be wondering why one wouldn't simply increase the learning rate to very large value, as this would lead to faster training right? Well, yes and no. Yes, because the network would descend the error gradients further, possibly

leading to a larger drop in error, but often this is not the case. To understand why you can think about training the ANN as a climber trying to reach bottom of a mountain, where the altitude of the climber represents the error of the network, which we are trying to minimise.



As you can see, the high descent rate of the climber caused them to overshoot the minimum altitude, actually leading to an increase in height rather than decrease. Similarly, you should be careful not to set your learning rate too high as the ANN may overshoot the minimum error, leading to a higher error than previous causing no actual progress.
Contrastly, the smaller the learning rate, the longer the ANN will take to train, however the training that does occur will be very careful, and will almost always lead to a decrease in error.
Finding the right learning rate for your GAN is important, I recommend starting around the default of 0.01, however if you find this too low or high you are free to experiment as that is part of the purpose of GAN Studio.

1.2.1.9. GANs in GAN Studio

Now that you understand what an ANN is and how it works, you are probably wondering what a GAN is and how it is even relevant, after all, GAN Studio is a program for creating and training GANs. Simply put, a GAN or *Generative Adversarial Neural Network* is the combination of two seperate ANN's in such a way that they are able to generate new training data similar, but not identical to those already present in the training data. GAN Studio for example, allows you to create

GANs that can generate new 28x28 pixel grayscale images of handwritten digits, or any custom images that you can draw.

1.2.1.10. How a GAN Works

As said before, each GAN consists of two separate ANNs, one is called the generator, while the other is called the discriminator. The generators role is to learn to create new training data that looks as real as possible to the discriminator, while the discriminators role is to learn tell the difference between images created by the generator (fake), and actual images from the training data (real). This means there is arms race of sorts that arises, since as the discriminator gets better at detecting fake images made by the generator, so does the generator at generating images that more closely resemble real images from the training data, and thus the harder the discriminators job of telling them apart becomes.
The optimal goal is that if you let both get better at their respective task for long enough, eventually the generator will be creating images that look almost identical to those from the training data, and the discriminator will not be able to tell them apart. This is real life however, so we never really achieve that perfect state, however we can get close, and close is usually good enough for images as humans are not perfect discriminators themselves. GAN Studio on the other hand was designed by a high school student with no formal training in ANNs, let alone GANs, so the results are not perfect.

Just as the purpose of either ANN is different, so does the structure of the generator vary to that discriminator. The input for the generator in GAN Studio is a list of random values, and it's output is the 784 long array of pixel darknesses for the 28x28 images. The random input means the each time, a new image is generated instead of the same one each time, otherwise the discriminator would learn quite quickly to tell it from the real images.
The discriminator on the other hand has the 784 pixel value array as input, and outputs a single value between -1 and 1. If the output 1, the discriminator thinks the image is 100% real, -1 means it thinks the image is 100% fake, and anything in between means the ANN is not exactly sure.

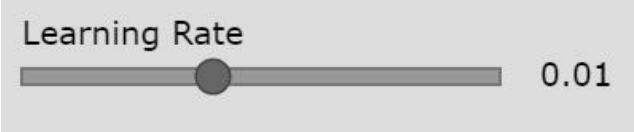
1.2.2. GAN Studio Basics

1.2.2.1. GUI Elements

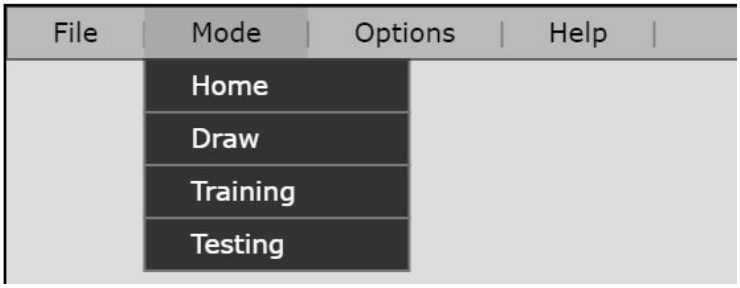
There are 4 main GUI elements used in GAN Studio, each have different applications and work in seperate ways.

Buttons are the main GUI element used to perform a particular action when the user clicks on them depending on the text inside.

Sliders allow you to set a particular value anywhere in between a minimum and maximum. To use a slider, click and hold on the circle inside of the rectangle, and 'slide' your mouse either to the left or right to change the value up or down.



Dropdowns exist at the top of the application at all times, and allow you to navigate your way around the program. To use a dropdown, hover over one of the options to reveal a list of sub-options. Without moving your mouse outside of the option and its dropdown, select one of the sub-options by clicking on it.



Scroll Bars are used to access a list of items by vertically scrolling through them. Click and hold on the scroller inside of the scroll bar, and drag your mouse up/down to navigate through the data.

Image rotated 90 degrees for convenience

1.2.2.2. Creating GANs

When using GAN Studio, the first step is to create an untrained GAN. To do this, use the dropdown menu at the top of the application to navigate to *File -> New GAN*.

Here, you will have the option of tweaking the parameters of your GAN, both the discriminator and generator ANN. You can switch between editing the generator and discriminator with the button in the top right that says either "*Edit Discriminator*" or "*Edit Generator*", depending on which you are already editing.

Both networks have the option of tweaking the number of hidden layers using a slider from 0 to 5, and also the number of neurons in each of these layers ranging from 1 to 128. The generator network also allows you to alter the size of its random input from 1 to 128. Having a larger input size generally increases performance in the long run as the generated images have less variation to begin with and can thus all be improved at a similar rate.

You can also pick the learning rate of either network independently. Be careful with this however, because if one network has a higher learning rate and improves too quickly at its job over the other, the other will eventually become so bad in comparison that it simply cannot compete, even if you completely stop training the better one. It is recommended that for most purposes the learning rate is the same for both, however the option is available if you wish to experiment.

If you already have a GAN created and choose to create a new one by opening the create GAN screen, you will be warned as your current GAN will be overwritten and cannot be retrieved. To create a second GAN whilst keeping the first you can choose to export to a file as shown next.

1.2.2.3. Importing/Exporting GANs

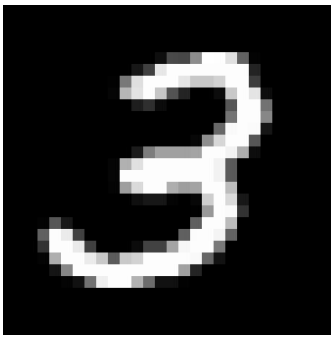
After putting time into training your GANs you may wish to save them and reuse them later, either to generate images or to continue training. To accomplish this, GAN Studio allows you to export your GANs to JSON files on your computer, and also import them back into the program. This feature is available through the dropdown menu, to save to a file, go *File -> Export GAN* and to upload a previously made GAN go *File -> Import GAN*. Importing will bring up the file explorer window and from there you can navigate and select the correct file.

1.2.2.4. Importing/Exporting Training Data

Similarly, you can also import/export lists of training data with a process almost identical to that of GAN files. *File -> Export Data* to download a JSON file containing all the current training data, and *File -> Import Data* to upload a list of pre-made training data. This is especially useful if you do not want to redraw an entire set of training data each time the program is refreshed.

1.2.2.5. MNIST Dataset

The MNIST dataset of handwritten digits is a list of 28x28 pixel grayscale images that was created specifically for machine learning purposes. In case you would like to test GAN Studio without having to draw images, I have converted ~1000 of each digits into a format readable by the GAN Studio importing feature. The files for each digit 0-9 are available with the project.



A typical image from the MNIST dataset

1.2.3. Home Mode

1.2.3.1. Introduction

When launching the program, and after creating a GAN, you will be brought to the home screen of GAN Studio. You can also visit the home mode at any time using the dropdown menu with "*Mode -> Home*". Here you can see the GAN you have created, displayed with information about the structure and learning rates of both the generator and discriminator.
The amount of training data that the application currently has stored is also shown. This information provides the user with an overview of the current state of the program, allowing them to more easily make a decision about what action to perform next.

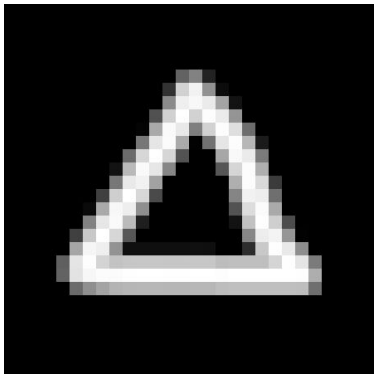
1.2.4. Draw Mode

1.2.4.1. Introduction

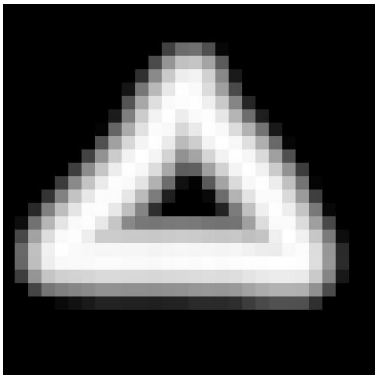
If you are up for the challenge of creating your own training data, the "Draw" mode (accessed with "*Mode -> Draw*"), will allow you to draw an entire set of data using a built-in drawing pad. To start, you can create a new image by pressing the "*New Image*" button, this will create a completely black canvas on the right half of the screen and also add a new image to the list of training data next to it.

1.2.4.2. Drawing Data

The drawing tool is similar to the of the pen tool in PhotoShop, where the strongest part of the stroke is in the center with it getting lighter towards the edges. Draw by clicking and dragging your mouse while remaining within the bounds of the canvas. If you wish to restart on an image, pressing “Clear Image” will set it back to its original blank state. A slider is also available to allow you to alter the brush size from 10 pixels in radius down to 0 to allow for different thicknesses when drawing.



Brush Size: 2



Brush size: 4

1.2.4.3. Altering Existing Data

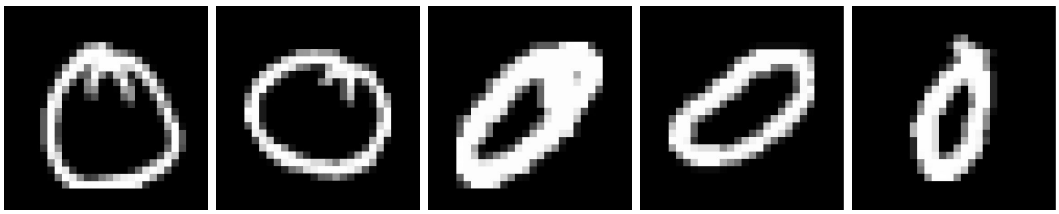
To perform actions on existing data, you must first select it by clicking on the small preview version in the image list next to the drawing canvas. The image list can only display 6 images at once, so to view all the data you can use the adjacent scroll wheel, or the buttons above and below to move up and down one image respectively. Selected images will display a green square around their preview, and by clicking on an already selected image it will be deselected. The position of the selected image relative to all others is also displayed below the drawing canvas in the form “*Selected Image: x / n*” where ‘*n’* is the total number of training images and ‘*x’* is its location within them.

You also have to option to edit existing data by first selecting it to make it appear in the canvas, such that now any edits that are made to the canvas will also be made to that particular training sample.
If you would like to duplicate the image in the training data, the “Copy Image” button will place a copy of the image in the list of training data, and select the new duplicate image.
You can also remove training samples on at a time by pressing “Delete Image” which will deselect the image and then remove it from the training data set.

1.2.4.4. Creating Good Data

Good training data is necessary if you wish to create an effective GAN that can successfully replicate what you have drawn. Firstly, for any single set of training data, all images should be of the same object (the letter A for example). You should not go mixing different images together as often what happens is the GAN will pick just one type and attempt to replicate it as there is no middle ground for it to occupy.

A wide variety of images is recommended for best results, meaning if you were to draw zeros, try to create different brush thicknesses and slants as this will give the GAN more features to mix as it attempts to generate its own. If drawing these different variations of the same thing, you should be careful not to leave large feature gaps in the dataset (for example drawing vertical A's and very slanted A's but nothing in between). Small gaps will promote the generator to learn all the possibilities and produce a wide range of images.



The first 5 images from the MNIST Dataset shows a good variety of zeros

The amount of data is also very important in training GANs, as this is often a direct reflection of the stability of training. The more data, the less likely a GAN is to collapse onto generating a single image and become essentially useless. From past experimentation, any more than 50 images is usually enough to have consistent results, but the more the better. This reliance on big data is also why I would recommend using the attached MNIST dataset for most testing purposes as you can be assured a lack of varying data is not an issue.

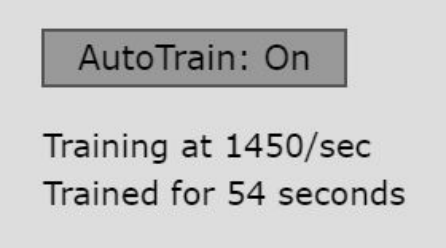
1.2.5. Training Mode

1.2.5.1. Introduction

Without training, any ANN is rendered useless and will produce meaningless outputs, GANs are no different, as training is the difference between random noise, and realistic looking images. The training mode handles everything related to training your GAN to generate images, you can visit this mode with "*Mode -> Training*". Before any training can occur, you must have a GAN and training data in the program.

1.2.5.2. AutoTrain

Pressing the "*AutoTrain*" button on the left of the screen will cause the program to take over, automatically training both the discriminator and generator to become better at their respective jobs.
The rate at which AutoTrain is functioning is displayed below with the in the form of the number of trains occurring per second. You can use this number to see the quantitative effects of certain network structures on computational speed.
The cumulative time that AutoTrain has been running in seconds is displayed below the training rate, this allows you to compare how fast different GAN configurations are able to train to your expectations. For the default GAN setup, you will only need to train for couple minutes to see recognisable results.



AutoTrain button and related stats

1.2.5.3. Training Ratio

Both the discriminator and generator are trained by default in alternating sessions of 25 trains each. This ratio of how many times the discriminator trains before the generator can be customised with a slider called the "Training Ratio". Sliding this up and down will change the ratio of trains that the discriminator has from 0 (0/50 trains), to 1 (50/50) trains, and anywhere in between. Training one more that the other is not something that you should leave in effect for long periods of time as it can lead to one becoming irreversibly better than the other. Although I do recommend playing around with the slider a little just to see the effect of training one more than the other one the performance of both.

1.2.5.4. Graphs

The large white square that takes up the majority of the screen in this mode is a graph, used to keep track of the performance of both networks. There are two graphs available which display the error of both the discriminator and generator respectively. You can switch between these two using the arrow buttons at the bottom of the graph.
While training, points will be added to these graphs which are connected by red lines. The maximum value of any point is displayed in the top left, and the number of points is displayed in the bottom right. After training for some time, the number of points will grow too large (>5000) for the graph to become readable and render smoothly, so it will automatically zoom in by a magnitude. You also have the option

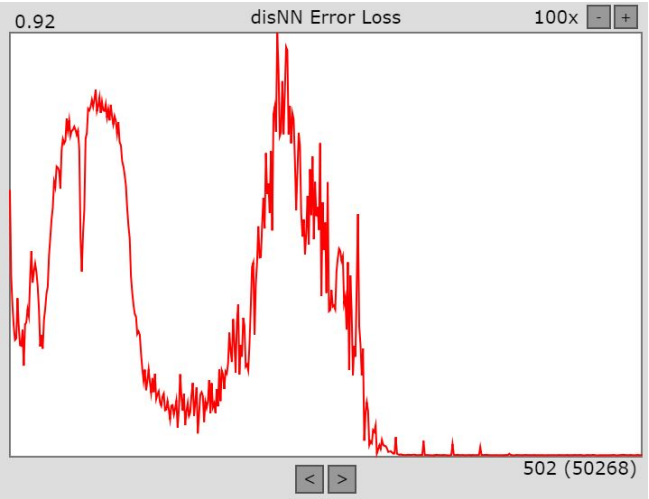
to alter this scale manually with the “+” and “-” buttons in the top right of the graph, next to which the current scale factor will be displayed, e.g. “100x”.

A typical graph while AutoTrain is On

1.2.5.5. Using Graphs

The actual meaning of these graphs in relation to performance is a little mysterious, as it is not simply “the smaller error the better” because there are two errors you are trying to minimise, each somewhat opposites of each other. You can however, use the graphs to detect certain situations which represent the failure of a GAN, which is much easier than analysing generated images and determining if a GAN is still improving or not.

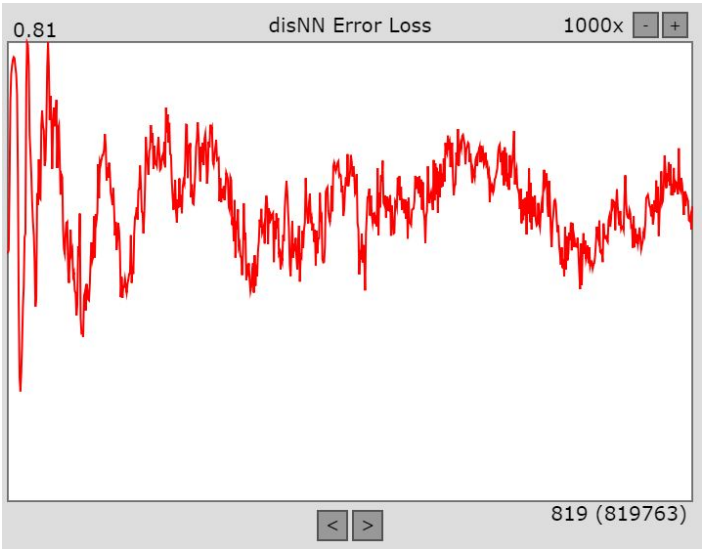
One major failure situation that is easy to detect is when the the errors of either graph completely zero out. This occurs when either ANN becomes too good at their job in comparison to the other, causing their error to reduce to zero as no matter what the other does, it always manages to adapt. It is often quite hard for a GAN to get out of this situation, and the longer you leave it in such, the harder it will become for the other to catch up. It is recommended that instead to try to minimise the chance of this happening, by ensuring neither network has a clear structural advantage, either through neuron/layer count or in learning rate.



A failed GAN (discriminator has become too accurate)

Another failure state, or atleast one that usually means training has slowed right down, is when the graph plateaus somewhere in the middle. At this point, it seems that neither the generator or discriminator is getting any better relative to one other. Otherwise we would see dips or spikes in error representing either the generator becoming better at tricking the discriminator, or the discriminator finding a new way to tell them apart from the real images.

Overall, you should look for graphs that involve lots of stable movement, both up and down, as this represents a constant cycle of one improving, and the other catching up. The graphs that seem to represent the best performance often slowly calm down over time with the spikes in performance weakening out as the generator converges to produce realistic images.



The training graph of a successful GAN.

1.2.6. Testing Mode

1.2.6.1. Introduction

No amount of analysing error graphs to track performance will ever really compete with actually viewing the images generated by your GAN. This is exactly the function of the "*Testing Mode*" ("*Mode -> Testing*"), allowing you to view and download images generated by the GANs that you have trained in the training mode.

1.2.6.2. How to test

To test your GAN, you can generate and view a single image by pressing the "Generate Image" button. This will feed the generator a set of random inputs, and place the constructed image inside of the large square on the right.
You can also download the images to your computer if you wish with the "*Download Image*" button, the images will be 28x28 pixel PNG file.
Training data is not required to generate images, however you will need to have a GAN loaded into the application. This means you can upload pre-trained GANs to generate and download images, then close the program without ever having to upload or draw any training data.
Testing mode also allows gives you the discriminators opinions on the generated images, showing you whether or not it thinks what is sees is either "Real" or "Fake", with its percentage of certainty displayed alongside.

1.2.7. Help

By opening the "Help" dropdown menu, users have access to the user manual in the form of a pdf which opened in a new tab.

1.3. Tutorial

You can access a basic video walkthrough of all the main features of GAN Studio at:
https://www.dropbox.com/s/qfr780ptgi1wq5h/Tutorial%20Video.mp4?dl=0

2. Interface Design

2.1. Intended Audience

As the intended audience for my major project is myself (and perhaps others with a similar technical interest into neural networks), I was therefore not focused on creating an “engaging” user interface, rather one that was based on functionality. This is why the interface is constructed almost entirely from text, rectangles, and lines, instead of flashy images and animations that other programs may utilise.

2.2. Ergonomic Issues

Even through the design was quite bland, that doesn’t mean I didn’t spend time taking into account a range of ergonomic issues, if anything, I probably spent too much time working on the user interface in comparison to the rest of the project.

A number of UI elements were utilised throughout the project, as each time I needed user interaction I made sure to use only the most appropriate element which included buttons, drop-down menus, scroll bars and sliders. Each element was not only interactive but also provided some feedback to the user in the form of changing colours in response to mouse hovering/clicking, as this gave the program some responsiveness.

Consistency was easy to maintain thanks to the modularity of each of the UI elements within the code. By making each mode looks similar in design to the others, users are able to re-use their knowledge of the interface throughout the program instead of having to relearn how to use the UI for each separate mode. The clearest example of this is the navigation bar up the top, as this is present for every single mode, allowing the user to navigate around the entire project by only learning how to use this single drop-down UI element.

The arrangement of UI elements throughout the project is also carefully taken into consideration, with each element being seperated by the necessary amount of whitespace as to not look clustered, while also being chunked into groups to show seperation in functionality. Elements are also aligned carefully and meaningfully, with both text and rectangles being left-aligned, centered, or right-aligned when necessary.

The font/colour scheme is quite bland, which is a reflection of the intended audience. Although I did still put effort to ensure that fonts were readable and sized correctly, and that colours were of similar shades (lots of different greys) and contrasted with the text to keep it readable.

3. Technical Documentation

All variables and functions are named carefully to give the reader of the code an understanding of its purpose.
Substantial extrinsic documentation can be found throughout the entire code base which describes the code in detail.

4. Project Blog

Blog entries for stage 2 of my major work can be found at:
- https://gan-major-project-matty.weebly.com/project-log

5. Communication

5.1. Progress Meeting Agenda

Work completed

- Neural network library
    - Backpropagation algorithm for training
    - Works quite well as a digit recogniser (90%+ accuracy)
- Program can generate somewhat working GANs
    - Images are really bad but recognisable.
    - Can very likely be made better
- Working UI with buttons/dropdown menus
- Different modes
- Uploading training data (MNIST Dataset)
- Documentation of blogs

Problems Encountered

- The terrible looking images generated by the GANs
- Lack of technical abilities to improve upon GAN algorithm
- Not sure if I will have enough time to add extra features on top of the major ones I already have.

Questions

- What is interface design consist of for criteria? Do I need sentences on each area in the final report?
- How does the criteria for testing work? Are testing methods part of coding or the final testing stages?
- Should I bother making the GAN better? Most major modes are done and all that's left is just to make the GAN better.
- How much time should I leave for rest of project?

- Interface design
- User docs
- Testing

5.2. Progress Meeting Minutes

To start the meeting we discussed my progress so far, that I had completed many of the major features and would therefore be able to complete the project in time. We also went over each topic in the marking criteria to ensure I was completing the task to the correct standards. It's during this discussion that I inserted my questions about individual components of the project.

Interface Design

- I was told that the reason others had created sentences on interface design, when it was clearly reflected in the project itself, was to convince the marker that they understood that area of software development, and had therefore taken such into consideration.
  This also applied for other areas of the project such as with the testing tools and techniques. For that we should write a small description on each to show our understanding, and then show why we used the tool/technique, or why not.

Testing

- We discussed the testing criteria in detail after I expressed my confusion. The first section on testing was actually just checking my knowledge on all areas related to testing software, and that it didn't really have a place at any particular point in development of my project.
- End user testing on the other hand was specifically for after development of the project. Here I would create tests that would analyse the project as a whole, such as a survey for other users after they use the software.

Bad Images

- This was one of the main problems in my project, as I was worried that the quality of generated images which was a reflection of the algorithm itself would not satisfy the requirements. I was told not to be worried about its horrible quality, as it technically works, and therefore completes my goal of "Learning how to create a GAN".

What to spend my time on

- As I no longer need to worry about making the GAN better, I can now focus on making the rest of the project documentation. I was told I should soon be slowing down development to focus on the rest, and should leave a reasonable amount of time for testing purposes. Only ⅓ of the project is towards the actual software so I need to spend more time on everything else.

6. Testing the software solution

6.1. Original design specifications

The original design specifications for my project involved a wide range of functionality, all of which have been kept in the final project to some extent. Here I list each of the original design specifications, and the extent to which my program achieves that goal.

Creating GANs - As a core feature of original design specs, users have the opportunity create new GANs and tweak a variety of parameters relating to its function. I had originally planned on having more than simply the structure and learning rate as changeable parameters, but I never reached a point where I was adding new training methods that allowed for more variables to customise. Batch learning for example would have provided a couple new variables to tweak, however its development only seemed to hinder performance so I ended up disabling it.

Training GANs - Users have the ability to train their GANs on a set of training data so that it can generate new images from the data set. While I was hoping that training would be more effective while writing the original design specs, I never really specified the extent to which the GANs should generate images, so this area of the project has been covered.

Viewing Graphs - As per the original design specs, users can view and analyse data through graphs on the screen.

Creating Training Data - As per the original design specs, users have the ability to create their own training data sets. I had originally planned on allowing users to upload individual training data images but due to how specific images ended up having to be in the final product (28x28 greyscale) I decided this feature would rarely be useful and stuck to only allowing users to draw their data with a drawing pad.

Generating Images - As per the original design specs, users can generate images with their GANs and export them in to the PNG file format.

Importing/Exporting Files - Users have the ability to import/export their trained GANs and sets of training data. The original design specs stated they would be text files but the final product uses JSON files, however this is a minor alteration and has no effect on the function.

Provide working example - The original boundaries stated that my project would come attached with a working GAN that can generate handwritten digits. While I have attached a GAN, its function is limited to generating only zeros as I feel this conveys well enough a working GAN. Attaching 9 more GANs for all the other digits is unnecessary to prove the functionality of GAN Studio.

Networking - Part of the reason for selecting JavaScript as the programming language was the possibility of easily integrating an entire web server based system to project. I was never sure about this feature however, and it turned out not being worth the time I would spend creating it. So this area of the original design specs, while never really official, was not completed.

6.2. Level testing

Level testing is the testing of a program at a variety of levels. This can be on the level of modules/subroutines to ensure their individual functionality, or of the entire program, to test the combination of modules that interact with one together to make a working program.

6.2.1. Module testing

To ensure the function of individual sections of the program, and the subroutines which they consist of, module testing was used throughout development. An example of which was the testing of the neural network library during and after its development.

Individual functions were tested after development, such as the code that generated the initial structure of each neural network. This part of the module was tested by printing the entire network after creation, and comparing it what I expected based on the parameters I gave it.
I also tested the functionality of the feed-forward output system, which after logging the output produced an entire array of NaN where there should have been floating point numbers. I discuss more about how I fixed this error in the blog post for v0.4.0.

To test the entire modules functionality before integrating it with other modules as shown in the blog post for v0.5.2, I created a simple addition program where the ANN learned to add two floating point numbers together. While the output to this test was entirely through the console, it allowed me to ensure the functionality of this module on its own so that when I do add it to the GUI-based program and start training it to generate/recognise images, I can be sure that any problems I encounter are not due to bugs in the NN class. I also performed the same test with a medium, and very large neural network to check boundaries of the module.

6.2.2. Program Testing

To ensure the functionality of an entire application with the interaction of different modules, program testing must occur. I performed this test for GAN Studio during end user testing, where I instructed users to perform a set of instructions that utilised each major module of the program in such a way that they all must work together successfully.

6.3. Live Test Data

6.3.1. Response Times

The response time of a program can have a large effect on the user experience, as users can become frustrated if a program is slow or unresponsive to their user inputs. This is often an important consideration when calling web servers, which I had originally considered as a possibility for my project, but never developed it so that area of response timing is not something I need to test.
I do however have a range of GUI elements that the user can interact with, from which an action is executed upon clicking after a certain delay. Under normal circumstances the time between clicking and a response is almost instantaneous, however, I found during testing that while the user is performing intensive training, the response time of GUI elements increases significantly. While I cannot fix such an issue as this computationally expensive process is a core part of GAN Studio, I did partially reduce the effect by automatically scaling down the graph whenever more than 5000 points were visible during training which caused noticeable lag.

6.3.2. Interfaces between modules

Testing of the interactions between modules is crucial in ensuring the operation of the entire project. GAN Studio constantly feeds both training data, and the GANs themselves between different modules such to accomplish tasks like drawing, training and testing. I ensure an effective operation of modules between one another during the end user testing, when I asked users if they found using a combination of all the different modes to achieve their goal a "*simple, and functioning process.*"

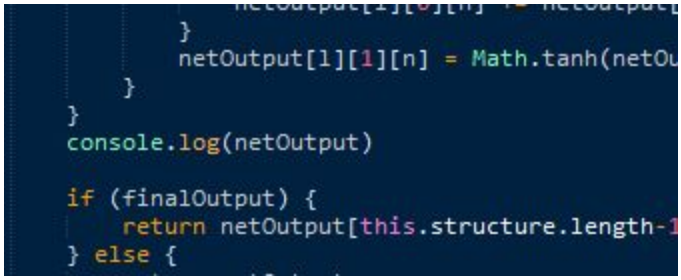
6.3.3. Volume Data and Load Testing

Especially since GAN Studio is based on intensive computation of lots of data, the programs' capability under load and when fed large file sizes is very important to test. To perform this test, which I go into detail in final stages of my blog, I created that largest possible GAN and fed it ~1000 training images to see if the neural network was still able to train. To my surprise, the network actually trained at about 100 trains/sec even with the size increase, and all other functions of the program including import/exporting of the 7mb file size for the GAN operated correctly.

6.4. Use of software techniques and tools

6.4.1 Debugging output statements

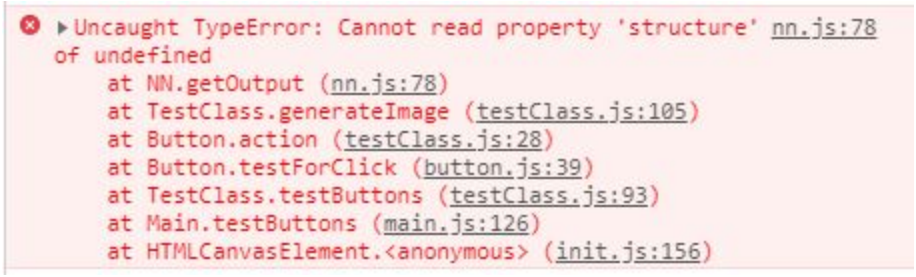
The act of inserting temporary output statements into the code during development/debugging can assist in understanding the flow of your program and in isolating errors. I used debugging output statements throughout development of my project, such as when trying to fix the feed-forward output function of my neural network library as shown in the blog post for v.0.4.0.



This logging of the networks output allowed me to pinpoint exactly where the error was in calculating the output, as the logged output had an array of NaNs in the input of the final layer. From this, I knew exactly where to look in the code and was able to fix the bug which turned out to be an off-by-one indexing error.

6.4.2. Tracebacks

As the JavaScript console supports tracebacks, they became a vital part of my debugging process, and were often the first place I looked in determining the source of any error that cause an exception. This is because tracebacks show the execution of a program leading up to the error while also displaying the code line at which the error occurred.



For example, the traceback error above told me I had an error on line 78 of the neural network library when I called the generateImage() function, as I was trying to access the structure of a variable that doesn't exist. After inspecting the line I realised that it was a spelling error and was able to fix the error.

6.4.3. Breakpoints

A breakpoint is a point that programmers can define in an IDE which will cause a program to pause execution upon reaching it. Once the program stops, it can be carefully inspected before continuing execution. This is especially useful in debugging, as developer use breakpoints to stop the execution of their program at certain key points to assist in finding the causes of errors. I did not use breakpoints in the development of my project. Partly because the IDE I was using (Sublime) does not support them, and because I found other error isolation tools/techniques (particularly debugging statements and tracebacks) satisfactory for most of my debugging purposes.

6.4.4 Single Line Stepping

Single line stepping is the process of pausing execution of a program after each line of code. They are similar to breakpoints in that they allow the user to view the state of their program during execution to assist in the isolation of errors, except they occur on each line instead of one that the developer sets prior. I did not use single line stepping in my project for similar reasons (Sublime does not support them and other tools/techniques were satisfactory).

7. End User Testing

7.1. Test Requirements

End user testing was conducted with 3 participants, each person was given a goal to complete within GAN Studio, and a list of steps to assist them through the process. After conducting the test I asked each a series of questions related to their experience with the program, also also asked for any recommendations that could be made to improve my project.

Users will require the following to conduct the tests:

- A computer and a valid browser to install the program.
- The program itself, and any attached files used for the testing purposes.

The requirements of tests themselves is that they need to perform the following:

- Determine whether new users are able to create working GANs.
- Validate the easy of use and functionality of each mode separately (draw/train/test).
- Ensure the program functions correctly as a whole with each mode working together.
- Ensure the user interface and navigation through the program is ergonomic and intuitive/easy to use.
- Discover any problems users encounter and find new ways in which the program can be improved.

7.2. Test Data

Users will need to complete a goal by following a given list of steps. After completion of the goal, I ask that they read and answer a list of questions about their experience with the program, and to suggest any changes they think would be beneficial. I have also attached a file containing the MNIST training data used in step 2.

Goal: “Construct your own GAN using GAN Studio that can generate recognisable images of zeros.”

Steps:

1. Open GAN Studio, and follow the instructions on the home screen to create your own GAN with the default settings.
2. Upload the MNIST set of training data for handwritten zeros, then navigate to the drawing mode and proceed to add 3 more to the dataset that you have drawn yourself.
3. Navigate to the training mode, and start AutoTraining for ~100000 generations (this will take ~1 minute).
4. Go to the testing mode, and generate a few images that you find recognisably zeros and download them to your computer. If nothing looks recognisable, either continue training or create a new GAN and restart training.

Questions:
1. Did you complete the task? Were you able to successfully create a GAN that generated zeros immediately or did it require multiple attempts?
2. On a scale of 1 to 10, how easy was it to use the drawing mode when you added your three images?
3. On a scale of 1 to 10, how easy was training your GAN for the suggested amount?
4. On a scale of 1 to 10, how easy was testing your GAN for recognisable zeros?
5. On a scale of 1 to 10, how simple, and functional of a process was using all the different modes together to create your final product (images of zeros)?
6. Was the user interface easy to use, and is it aesthetically pleasing enough in relation to the function of the program?
7. Did you encounter any problems during your usage of GAN Studio?
8. Do you have any recommendations that you think would improve the application?

7.3. Expected Results

I expect all users to be able to complete the steps I have provided and achieve the goal of training their own GAN. I am not sure all users will complete the task on their first try, not due to any problems with using the program, but because GAN training is not always reliable and sometimes fails randomly.
Users will probably find using each mode a simple and functional process as their are so few buttons to press and thus not many places to go wrong. The drawing mode will likely be the hardest to grasp, but most users should figure it all out within 30 seconds or less as their really isn't much to it.
I also expect users to comment on the blandness of the user interface, although I am not too worried as this is not an important feature when you consider the intended audience.

7.4. Test Results and Recommendations

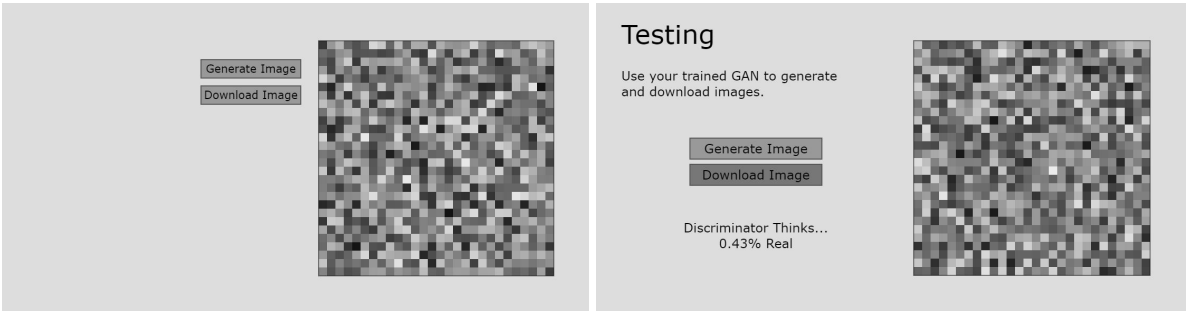
All users were able to successfully complete the task of creating a GAN that can generate images of zeros on their first attempt. Each commented on the actual look of their zeros, stating they looked fuzzy and were not always shaped correctly, but that they were recognisable. I do wish I could fix this issue of generating bad zeros, however this is simply

too hard to accomplish in the time I had available and with my current knowledge. I suspect the issue is in part due to the relatively high learning rate when compared to other peoples GANs, as most others I could find that generate MNIST digits used much faster GPUs for training and still took at least an hour to generate recognisable digits.

Users found both the training and testing mode very simple to use, probably because there are so few elements on either mode. This is likely because one is dedicated to just waiting (training) and the other is just for visually inspecting images (testing) rather than actual user interactions and thus the learning curves are much smaller.

The drawing mode was always reported as the relatively the hardest part of their experience with GAN Studio, as it took the longest time to realise how to use it relative to other modes, even though it was still quick to figure out. I asked users what could improve the mode, and one asked for some instructions on where and how to use the drawing ability. Thankfully this is what the user manual is for, which I deliberately kept from the users to test the intuitiveness of my program. As all figured out how to use it before becoming frustrated or confused, I will not be making any changes to this mode.

One user commented on how empty some of the modes (particularly testing mode) appeared in places, that their seemed to be large empty spaces that were aethetically unappealing. These spaces were originally left blank for potential future UI elements, but now that I am in the testing phase and will likely not be adding many more features, I have removed the spaces to the best of my ability. To accomplish this, I decided to add a title and a small sentence on its function to the testing mode, and for consistency, to each other mode. This filling of blank spaces and the shifting/resizing of elements has allowed me to create modes that, while no actual features have been added, look more completed and have screen space used to its potential.



The testing mode before and after changes were made to fill in blank spaces.

A bug was found by a single user, who for some reason thought they would attempt to start training before creating a new neural network. This led to the program crashing as it attempted to access a non-existent GAN. I was a bit confused as to why I had not found such a major bug, but up until very recently I had been automatically creating a default GAN in the initialisation of the program, as this meant I could test features quicker without having to go “File -> New GAN” every time I restarted the project. This shortcut led to me having a

GAN open at all times, so I was never able to test the program in situations without a GAN, such as upon launching of the program as highlighted by this test. The bug was a simple fix, with me now checking if a GAN exists before allowing users to enable training.

Overall, the end users tests were a success and led to the highlighting of multiple issues that have now been resolved. No other problems other than those stated and fixed were encountered by any users, which has made me feel quite confident that completion of the project is very close.