GAN Studio

User Manual

1. Introduction to GANs and ANNs

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. 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.

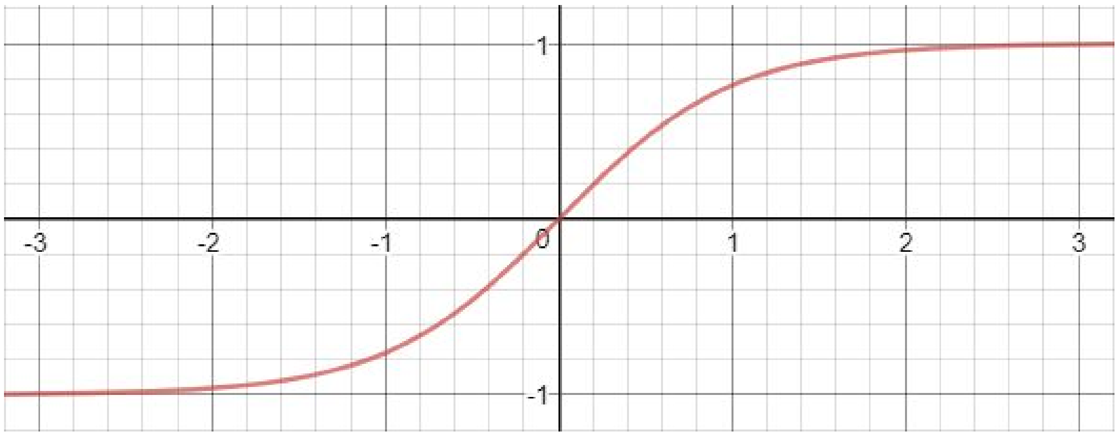
- https://www.youtube.com/watch?v=aircAruvnKk

1.3. 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).

1.4. 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.

1.5. 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.

1.6. 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.

1.7. 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…

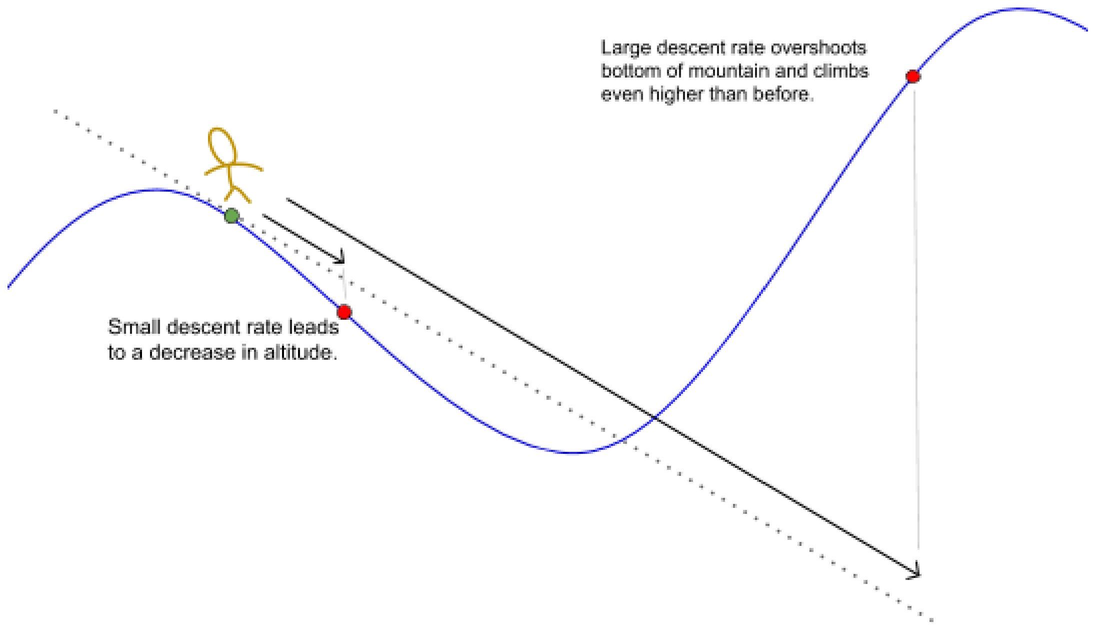
1. Feed the network your input, and keep track of how it produced its output. Before training, this output will be essentially random.
2. Using magical calculus (*some simplification has been made here*), start from the final layer and work your way backwards, finding the rate at which each individual 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.

1.8. 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.


Large descent rate overshoots bottom of mountain and climbs even higher than before.
Small descent rate leads to a decrease in altitude.

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.

1.9. 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.

1.10. 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.

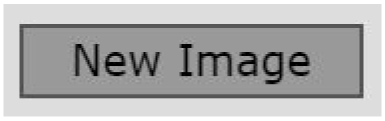
The discriminator on the other hand has the 784 pixel value array as input, and outputs a single value between -1 and 1. If the output 1, the discriminator thinks the image is 100% real, -1 means it thinks the image is 100% fake, and anything in between means the ANN is not exactly sure.

2. GAN Studio Basics

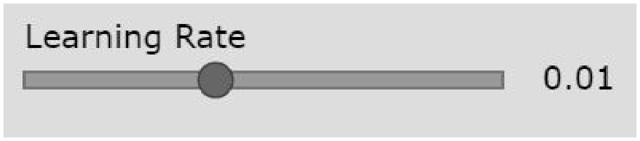
2.1. GUI Elements

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

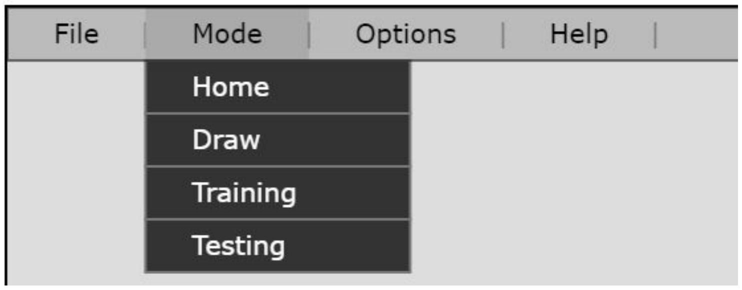
Buttons are the main GUI element used to perform a particular action when the user clicks on them depending on the text inside.



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.

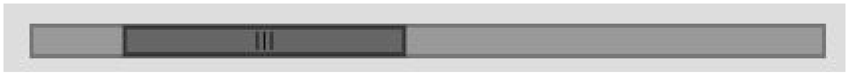


Image rotated 90 degrees for convenience

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.

2.3. Importing/Exporting GANs

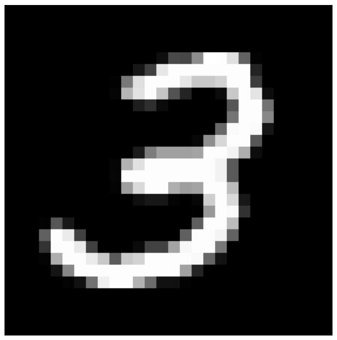
After putting time into training your GANs you may wish to save them and reuse them later, either to generate images or to continue training. To accomplish this, GAN Studio allows you to export your GANs to JSON files on your computer, and also import them back into the program. This feature is available 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.

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.

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

3. Home Mode

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.

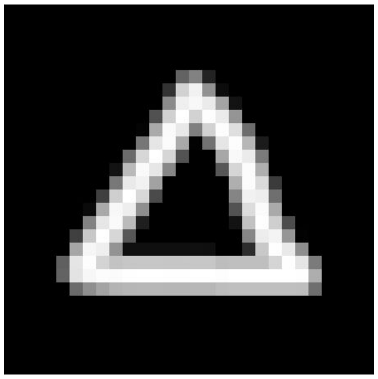
4. Draw Mode

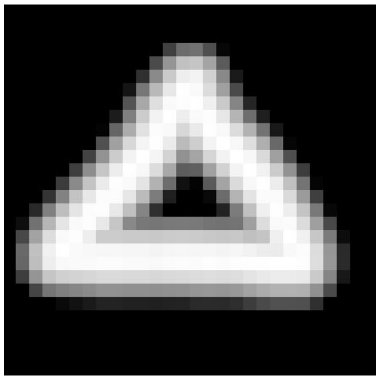
4.1. Introduction

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.

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

4.3. Altering Existing Data

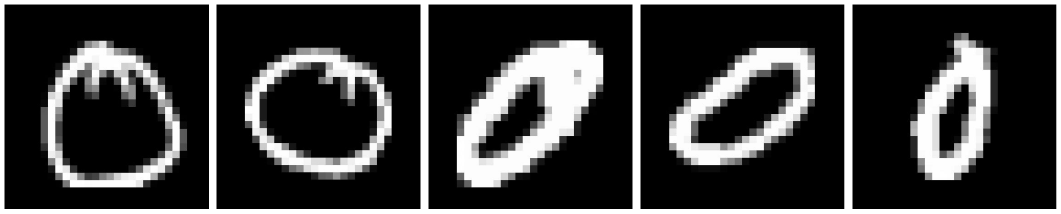
To perform actions on existing data, you must first select it by clicking on the small preview version in the image list next to the drawing canvas. The image list can only display 6 images at once, so to view all the data you can use the adjacent scroll wheel, or the buttons above and below to move 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.

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.

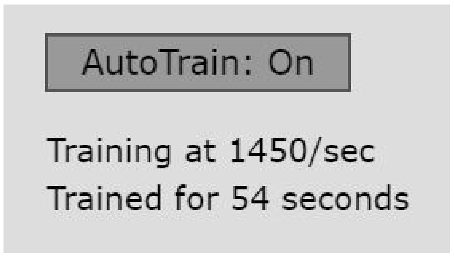
5. Training Mode

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.

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.



Training at 1450/sec
Trained for 54 seconds

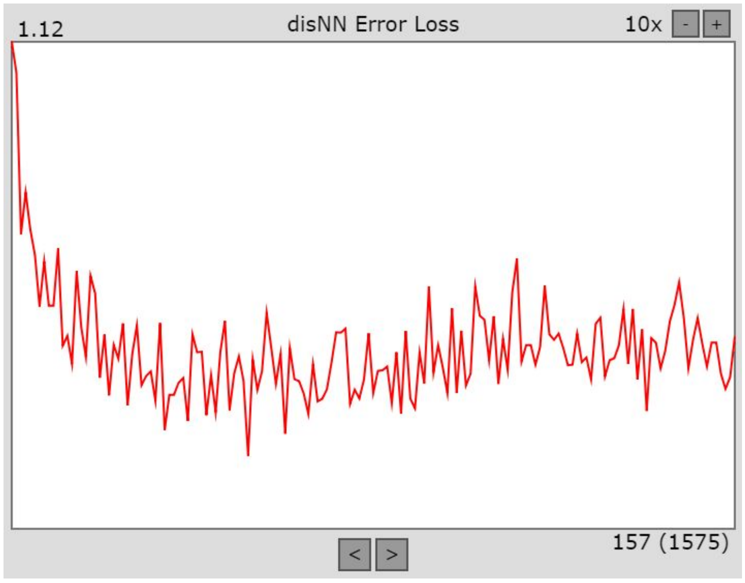
AutoTrain button and related stats

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.

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”.

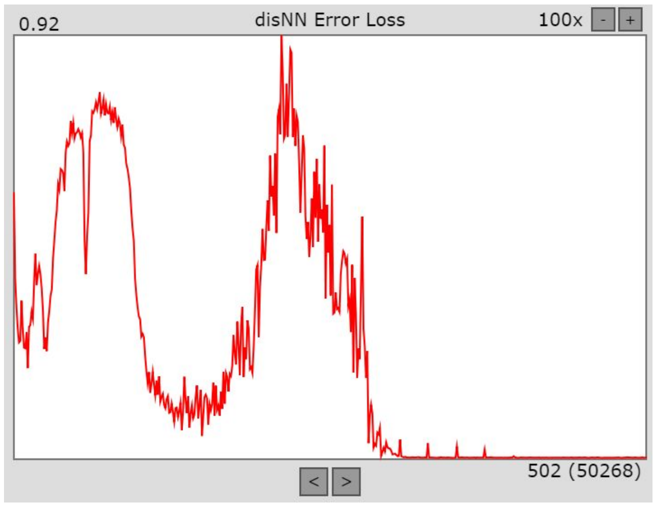


A typical graph while AutoTrain is On

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,

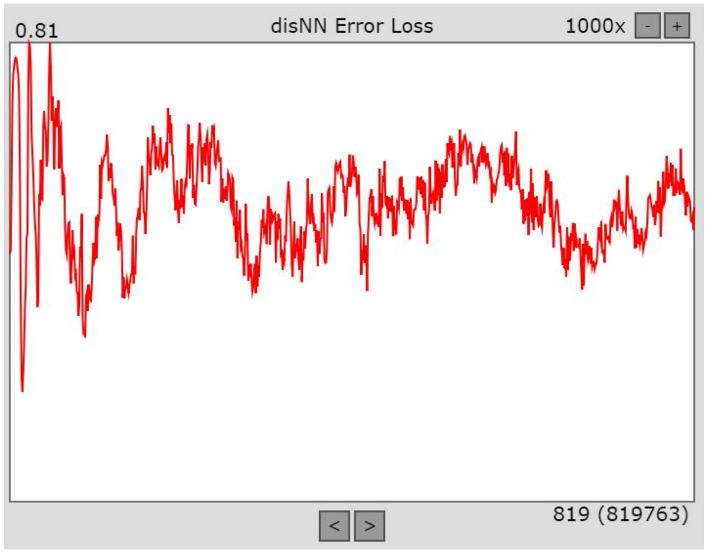
by ensuring neither network has a clear structural advantage, either through neuron/layer count or in learning rate.



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.

6. Testing Mode

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.

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.

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.