**Training GANs**

**Problems training GANs**

GANs are extremely sensitive to train. This is because the different parts of the GAN need to be train separately, but at the same time, while finding the balance between them so that none of them overfits to early. The discriminator overfits easily and should not, preferably, predict perfectly. Even though the goal of the discriminator during training is to predict real images as 1 and generated images as 0, it is desirable that this doesn’t happen, since that means the generator is generating poor images.

Additionally, we want the encoder and generator to reconstruct the real images, which is a problem itself.

* **Mode collapse**

Mode collapse in a GAN is when the model fails to explore the full distribution of training data, and the model’s generator produces a restricted set of output examples. In essence, the GAN's generator becomes trapped in a specific mode or pattern, preventing it from generating diverse outputs that span the entire spectrum of the data. This limitation can lead to repetitive, monotonous output lacking variety and detail, and in some cases, the generated content may bear no resemblance to the training data.

A typical reason for mode collapse is that the discriminator overfits. As mentioned above, the discriminator overfits easily when training a GAN, which is one of the most frequent issues when training a GAN. Other reasons for mode collapse are that during the learning process, the GAN forgets previous learning steps. This is referred to as catastrophic forgetting, which is defined as forgetting steps learned in a specific task when the model is trying to learn another task.

**Possible trick to train GANs**

* **Training generator more than discriminator**

There are multiple solutions to finding the right balance when training a GAN. One approach is to train the discriminator less often than the generator. That means for every x batch passed through the generator during training, one batch is passed through the discriminator. This approach is good for making sure that the discriminator is not overfitting too early. It is important that the discriminator gets different batches the few time it is trained.

* **Training discriminator on real images, noise and generated images**

Another approach is to pass different input through the discriminator. Typically, real images are passed through the discriminator with the intention of being classified as 1 (real images) and recreated images are passed through the discriminator with the intention of being classified as 0 (generated image). However, to train it even further, we could pass generated images, noise passed through the generator, with the intention to be classified as 0 (generated image). Adding the latest approach could lead to an overfit in the discriminator since it sees more input that should be classified as generated images, and the generated images can clearly be separated from the real images, especially early on, which will lead to early convergence.

* **Scaling loss**

Scaling losses is an additional approach to find the right balance between training the discriminator and generator in the GAN. The GAN have multiple different losses for the different parts, like a combination between a classical GAN loss and reconstruction loss in the generator. This will most likely make the generator loss significantly larger than the discriminator loss, which can affect how the model weight the training of each part. Scaling the loss, especially the reconstruction loss in the generator, can help find the balance between classification and reconstruction in the encoder/generator part, which again can improve the chances of convergence in the GAN.

* **Different latent size**

Experimenting with different latent sizes could help the model train better. If the latent space is too sparse or too big, the latent representation of the image might not be a good representation and the generator might not be able to create a decent image of the latent representation.

* **Hyperparameters**

Changing and experimenting with different combinations like learning rate and batch size could potentially alter the training efficiency.

* **Using different noise sampling distributions**

Using different noise sampling distributions could potentially affect the new generated samples. The most common methods to sample noise is either from a uniform distribution or a normal distribution. Changing the parameters of the distributions could also have an impact on the generated samples.

* **Combining solutions**

Experimenting with different combinations of the presented tricks could lead to faster convergence and better results in terms of generated images when training a GAN. However, as stated, to find the right balance or a sweet spot could be very hard due to the sensitivity of the GAN.

* **When the model has converged**

A black and blue rectangle with green lines

Description automatically generatedA blue and black rectangle with black and white lines

Description automatically generatedHow do we know when the model has converged? There are a few things to notice when the model has converged. Obviously, loss is the first thing to look at, however, it can be tricky to determine convergence just by looking at loss, since we have different losses with different scales (reconstruction vs classic GAN loss). Additionally, overfitting or mode collapse could lead to low loss in either the discriminator or generator, which can make the loss converge, even though the model hasn’t.

First of all, we can look at reconstructed images. We want the reconstructed images to look very similar to the original image.

Secondly, we can inspect generated images to make sure they are consistent and look like they are from the training data distribution.

Another important check to make sure the model has converged is to check the discriminator predictions. If the model as a unit has converged, the generator should make really realistic generated images that the discriminator can’t distinguish from real images. In practice, this means that the discriminator predicts values close to 0.5 for both real images and generated images.

**Loss**

* **Reconstruction loss**

The reconstruction loss in a GAN is used to train the encoder and the generator. As mentioned, the encoders job is to take a real image and encode it into a latent representation. Then the generator decodes the latent representation of the real and then reconstruct the original image. The reconstruction loss is therefore a representation of the difference between the original real image and the reconstruction of the same image.

To calculate the reconstruction loss, we calculate the mean squared error (MSE) of each channel between the original image and the reconstruction image separately. That means we get the MSE of the distance, mass and subhalo of the original image and reconstructed image. Then these three MSE’s are added together to a total reconstruction loss.

An issue with using MSE in the reconstruction loss is that it creates some blurriness in the reconstructed images. Zhao et al. (2018) [75] This is because the reconstructed images tend to be an average of potential solutions when we use MSE (L2 loss) as the reconstruction loss[76].

FID

* **Regular GAN loss**

The classical GAN loss is a classification loss. Remember that the discriminator wants to classify the real and generated separately, while the generator wants to generate images that the discriminator can’t distinguish from real images. The classical GAN loss reflects this tradeoff.

The discriminator wants to classify real images as real images and generated images as generated images. Goodfellow et al. proposed the following loss for the discriminator: (Goodfellow et al., 2014) [7]**:**

**A black text on a white background

Description automatically generated**

It is the mean sum of the log of the classification of the real images plus the log of one minus the classification of the generated images.

* The first term D(x^i) should be as close to 1 as possible, since the discriminator want to assign a high probability to real images being real. Remember the discriminators output is [0, 1], where the number describes the probability of the image being a real image. So 1 = real, 0 = generated image.
* The second term D(G(z^i)) should be as close to 0 as possible, since the discriminator want to assign a low probability to the generated image being real. Taking log(1 - D(G(z^i))) will give a lower value the lower D(G(z^i)) is.

The generator wants to generate images that are as similar to the real images as possible, so that the discriminator can distinguish between real and generated images. In other words, the generator wants to fool the discriminator with the generated images. In the same fashion as the discriminator loss, Goodfellow et al. proposed the following loss for the generator: (Goodfellow et al., 2014) [7]**:**

**A black text on a white background

Description automatically generated**

it is the mean sum of the log of one minus the classification of the generated images. The generator want to fool the discriminator, so it want the discriminator to predict real with high probability (close to 1) for generated images.

**With encoder**

Additionally, when training a GAN with an encoder, we add the reconstruction loss proposed to the classical GAN loss for the generator. This is because the encoder-generator process are reconstructing the images and to make sure that is done well, the reconstruction loss is a valuable asset to help with reconstruction convergence.

The encoder is also trained using the reconstruction loss.

**My approach with training**

To create a decent GAN model that works on the data, I took a lot of small steps to explore and develop the GAN structure and architecture, and to work with the difficulties of training a GAN. Following are my approach to reproducing the model architecture and results from Robles et al. (2019) in “A Halo Merger Tree Generation and Evaluation Framework“.

**AutoEncoder**

To check if the encoder-decoder architecture of any GAN is capable of reconstructing the images, it is a useful test to see if the encoder-decoder architecture manages to reconstruct images when trained as a classic Autoencoder (AE). The process is fairly simple, using the encoder-decoder architecture as a standard AE with MSE loss as reconstruction loss. Training this AE over 20 epochs and evaluate loss and some images.

If the encoder-decoder architecture of the GAN manages to reconstruct the input images as an AE, it should be capable of doing the same in the GAN.

* A graph with a line

  Description automatically generated**Reconstruction loss:**
  + When training the AE, make sure the loss converges. Here we can see the loss (y axis) depending on the number of iterations (x axis).
* **Reconstructed images:**
  + A screenshot of a computer game

    Description automatically generatedTo ensure reconstructed images are good, it is good practice to inspect them. Here we can see the original image (randomly chosen) on the left and the reconstructed image on the right. We see that the images are extremely similar, especially in terms of shape/structure. The shape of the two galaxies is 100% similar **(show to test)**, which is extremely good. However, in terms of coloring, the regenerated has some “blurry” spots or different shades of colors.
* **Checking regenerated color shades:**
  + Why bother analyzing the spots with different color shades in the regenerated images? Well, we are dealing with three variables, and if there is one specific variable that is the reason behind this discontinuity between the original and regenerated image, it would be good to know that, so that we know what to expect with A purple and yellow rectangular object with green lines

    Description automatically generated with medium confidenceour results or find a potential solution for this discontinuity.

Figure 1. Mass, real image on left

* + **Mass:** mass reconstruction does well. The structure is great, and the color grading is decent. There are some discontinuities, where the reconstructed image has less green (high mass) spots than the real image.
  + A screenshot of a computer game

    Description automatically generated**Dist:** The reconstructed distance to the main branch variable has very big discontinuities from the original values. We see that there are gaps in the reconstructed distance variable, which is not desirable. This can be a big factor in terms of the differences in color grading in the reconstructed image.
  + A purple and yellow logo

    Description automatically generatedA purple and yellow logo

    Description automatically generated**Subhalo type:** subhalo type is an interesting variable since it only has values of 0.0, 0.5 and 1.0. It is a categorical variable represented with float values. This could be hard to work with for a neural network if it is not treated with care and handled. As we can see, the reconstructed subhalo type values have the same structure and is very well predicted in terms of values (dark green where original image is dark green, yellow where original image is yellow). However, there are some discontinuities in the reconstructed image; there are multiple shades of dark green and yellow, which comes from the fact that the model gives it floating values. If these floating values are close to 0.5, but not exactly 0.5, they will have a different shade when represented as an image. Additionally, there are only three potential values for this variable, 0.0, 0.5 and 1.0. so this needs to be delt with.

Figure 2. Subhalo, real image on left

Figure 3. Dist, real image on left

* + One simple solution is to create a mapping function that maps values below a threshold to 0.0, values above a threshold to 1.0, and values between the thresholds to exactly 0.5. This would give only three (categorical) values like the real image. Applying a simple mapping function as described above, using 0.01 as the lower bound and 0.75 as the upper bound (not fine tuned, just intuitively set) gives the results on the right. As we can see, the images are identical apart from one subhalo pixel on the bottom left.
    - Checking what values for upper and lower bound gives the best results and using the best combination might enhance this result

Figure 4. subhalo, real image on left, mapped reconstructed image on the right

* + - Another option is to implement different solutions that train upper and lower bound, split the subhalo variable into a categorical variable and train the network using that.
  + **Overall:**
    - Overall the reconstructed image manages to gather the structure of the image perfectly, with some color gradient discontinuities. Mass and subhalo reconstruction are fairly good both in terms of structure and value. The categorical variable issue with the subhalo variable can be solved easily using a mapping function, but other more robust solutions might be good to inspect as well. In terms of the discontinuities between the reconstructed image and the original image, it seems like the main issue is the reconstruction of the distance variable.
    - The reconstructed distance variable doesn’t manage to reconstruct the whole structure of the original image and have gaps within the reconstructed values. This seems like a good explanation of the color shading differences between the reconstructed and the original image.
    - This is good to keep in mind to try to find a solution to the issue.
      * One “quickfix” solution would be to not use the distance variable at all and only use mass and subhalo in a 2-channel image. This is done **IN PAPER 2, 3 ???? with better or similar results than using all 3 variables ????**

**Simple GAN**

First, I created a very simple GAN consisting of a simple discriminator and a simple generating. The discriminator takes a flattened image as input and consists of two linear layers with a LeakyRelu between the layers, and a Sigmoid function at the end, to keep the value between 0 and 1. The generator takes noise of latent space size as input and generates a flattened image. I trained the generator and discriminator simultaneously as for the same amount of time. The loss function for both the discriminator and the generator was a Binary Cross Entropy Loss, where the generator was only trained on maximizing the discriminator output of the fake sample generated by noise, while the discriminator was trained to minimize its output of the real and fake image.

The images generated was fairly noisy and inconsistent and the discriminator managed to separate real and fake images fairly easily (0.75, 0.32)

**Simple GAN with encoder and reconstruction loss**

The second step on the journey to create a functional GAN model was to add an easy encoder to the simple GAN and reconstruction loss. The encoder is trained using reconstruction loss. The generator is also trained using the reconstruction loss as well as the Binary Cross Entropy Loss explained above.

Now the discriminator is trained by trying to classify real images as real, generated images from noise as fake, and reconstructed images as fake. The generator is trained on trying to make the discriminator classify both generated images from noise and reconstructed images as real images as well as trying to reconstruct the original image with the reconstruction loss.

**Results..**

**Simple GAN training discriminator less than generator, scaling, latens space size.**

Once the simple GAN with encoder and reconstruction loss is training fairly well, it is time to attempt to tailor the training to perfections. There are several techniques to tailor the training mentioned above, such as scaling the different losses, training the discriminator less frequently, and testing different latent space sizes. Exploring different combination of these methods makes the convergence of the model better.

**Results…**

**Simple convolutional GAN**

Once we have a simple GAN doing fairly well, we can slowly implement more advanced architectures and structures. The paper by Robles et al. (2019) proposes a convolutional GAN, so a natural step is to implement a simple convolutional GAN.

The discriminator in the simple convolutional GAN has one convolutional layer, an elu activation function, then a linear layer before applying a sigmoid function. The encoder has one convolutional layer, an elu activation function, then a linear layer which transform the image into a latent space representation. The generator takes a latent space representation as input, applies a linear layer, and an elu activation function before rearranging the data to the correct shape, then it applies a deconvolutional layer and a sigmoid function.

A table of text with numbers

Description automatically generated with medium confidenceThe training and loss are similar to the simple GAN with encoder and reconstruction loss. I also attempt to apply the training tricks to tailor the training process to create a model that converges nicely. The discriminator is trained less frequently and loss is scaled, experimenting with how frequently the discriminator is trained and the loss scalar.

**Results …**

* + - **Good reconstruction**
    - **Good discriminator balance (around 0.5 on real and fake images)**
    - **Bad consistency in generated images**

**Big conv GAN**

When the simple convolutional GAN performs fairly well, I extended the convolutional GAN to mimic the one of Robles et al. (2019).

**Results …**

**Trying to solve consistency issue with generated images**

The model manages to reconstruct images fairly well and the discriminator predicts around 0.5 for both fake and real images. However, there is an inconsistency in the reconstructed images and the generated images. In terms of the reconstructed images, the model manages to capture the structure of the galaxy formation, but there is some blurriness in the generated images. The first approach to inspect this would be to look at the encoder-generator architecture as an autoencoder (AE) to make sure the capacity of the model is able to fully reconstruct the images perfectly.

Another likely reason for the blurriness in the generated images might be that the subhalo variable is a categorical variable that is treated as the other two variables (mass and distance), that is, as a continuous variable. If the model don’t generate the same categorical values (0.0, 0.5, and 1.0), it might affect the coloring of each halo (pixel), which would lead to blurriness.

A screenshot of a computer generated image

Description automatically generated

There are multiple approaches to solve this:

* Create a mapping function that converts subhalo variables into the categorical values (0.0, 0.5, and 1.0) depending on a threshold. An example would be to map values above 0.8 to 1.0, and values lower than 0.05 to 0.0, and the rest to 0.5.
* Another approach would be to treat the subhalo variable as a categorical variable from the start. One hot encode the categories, so that each “image” has 5 channels. Then softmax the three subhalo categorical channels during training to assign subhalo category.

**Test this?????**

generated images have both **branch-gap-inconsistency** and **skip-step-inconsistency.** Additionally, the generated images are blurry with **single-point-inconsistencies.** To solve the different inconsistencies within the generated images it would be beneficial to test the subhalo categorical variables approaches, then test the Encoder-Decoder architecture as an Autoencoder to make sure it is able to reproduce the images fully, and if both give good results but the generated images still maintain the inconsistency, the approach would be to fine-tune the training process.

**Final convolutional GAN**

**Results**