**GAN reproduction**

On the road to implementing diffusion methods to generate merger trees, a first and reasonable step is to try to reproduce the results from the original merger tree generation using GAN papers, Robles et al. (2019) and Robles et al. (2022). Obviously, the following implementation will differ in various way due to lack of documentation of method in the papers, and analysis conducted in data section. The architecture of the model is xxx, and is very similar to the one in the papers. The main difference is the application of the relu activation in the last layer of the generator, instead of the suggested sigmoid activation function. Additionally, the data is normalized according to the work that was completed in the data section, that means log scaling and minmax normalizing distance and minmax normalizing mass. No preprocessing is performed on the subhalo variable.

In this section there will be experimented with different datasets and with two activation functions. The reason for that is because even though there is a lack of documentation in the papers, it is reasonable to attempt the methods described. The two activations functions that will be experimented with is the relu and the sigmoid (from the paper), and the datasets will be a normalized dataset with merger trees with six branches only (paper 2, but here it is normalized), a normalized full dataset with 5-10 branches and the original dataset without any preprocessing (from paper 3). Here, “normalized” means the normalization methods mentioned above and concluded with in the data section.

**Merger trees with six branches**

The first successful GAN approach was with merger trees containing six branches only. That means the merger trees with six branches are filtered out, then the merger tree representation is reduced down to only representing six branches, meaning the merger representation is transformed from shape (3, 29, 10) 🡪 (3, 39, 6). But only containing merger trees that originally have 6 branches. This transformation basically removes four empty branches/columns.

Code for results and model in this section can be found in the 11 - successful GAN six branches only notebook. The methodology of explaining results and evaluation of results in this section will be the “blueprint” for the following sections on GAN results.

Hyperparameters and specifications:

* Loss function = Binary Cross Entropy with logits
* Reconstruction loss = reconstruction loss penalizing if

target is 0, 0.5 or 1.0 as described in data preprocessing section.

* Optimizer = Adam
* Noise = uniform(-1, 1)
* Number of training epochs = 300
* Learning rate = 3e-4
* Batch size = 128
* Latent space = 300
* Discriminator/generator training ratio = 1
* Reconstruction loss scaling = 10

To test the consistency of the model, it generates new samples of the same size as the training data (8161) ten individual times. The average constancy is 10.12%, with a minimum consistency of 9.29% and a maximum consistency of 10.61%. Compared to the original training data which has 98.17% consistency, this is fairly low.

Here, consistency refer to the structural consistency discussed in data section and evaluation section. For a generated image to be consistent, it must satisfy all requirements, which quickly summarized is:

* All three variables should have zero and nonzero values in the same pixel index location. This is for all pixels except distance in the main branch
* Distance in the main branch should be zero only
* There should be no empty branch between branches. That means if there is an empty branch/column, the rest of the columns should be empty
* There should be no gaps within a branch, that means if there is a branch with progenitors (nonzero values) and there is a pixel location which is zero (absence of progenitor), the branch should stop, and the rest of the branch should be empty (zero)
* The last row (snapshot) should only contain a single last descendant in the main branch (first column). This means the last row should only have nonzero values in the first column.

Many of these points go hand in hand, and an inconsistency in one of the points will lead to an inconsistency in another point. If all these structural consistencies are satisfied, then and only then, will a generated merger tree be classified as consistent.

The size of the trees generated is as follows:

|  |  |  |
| --- | --- | --- |
| **Number of branches** | **All generated trees** | **Consistent trees** |
| **1** | 0 (0.000%) | 0 (0.000%) |
| **2** | 3 (0.036%) | 0 (0.000%) |
| **3** | 50 (0.612%) | 6 (0.69%) |
| **4** | 343 (4.202%) | 23 (2.67%) |
| **5** | 1872 (22.94%) | 81 (9.43%) |
| **6** | 5893 (72.2%) | 749 (87.2%) |
| **7+** | 0 (0.000%) | 0 (0.000%) |
| **total** | 8161 | 859 |

This is a great and particularly desired result, that the majority the generated trees have the same number of branches as the training merger trees. This represents the complexity of the generated tree, which we want to reflect the complexity of the training trees. For the consistent trees, 87.2% of the generated trees have 6 branches, and 96.63% have 5 or more branches. Another method to evaluate if the complexity of the generated trees reflects the complexity of the training data is to compare the number of nonzero entries:

**Nonzero entries**

|  |  |  |
| --- | --- | --- |
| **variable** | **Original data** | **All generated trees** |
| **Distance** | 266 699 | 290 383 (+ 8.88%) |
| **mass** | 468 178 | 482 987 (+ 3.16%) |
| **Subhalo type** | 468 178 | 475 546 (+ 1.57%) |

Overall, the generated merger trees have more nonzero entries than the original data, which have the potential to indicate higher complexity. However, this is not an accurate measure, since the consistency is far from the original data, it can also indicate a lot of inconsistencies with the generated images. Looking at the average number of nonzero entries in a consistently generated merger tree compared to the average number of nonzero entries in the original data indicate that the complexity is kept to a reasonable extent.

**Average nonzero entries**

|  |  |  |
| --- | --- | --- |
| **variable** | **Original data** | **All generated trees** |
| **Distance** | 32.68 | 31.08 (- 4.88%) |
| **mass** | 57.36 | 55.43 (- 3.37%) |
| **Subhalo type** | 57.36 | 55.43 (- 3.37%) |

A group of rectangular objects

Description automatically generated with medium confidence

In terms of the consistent generated merger trees, they have a slight reduction in the number of nonzero entries than the original trees. However, a reduction of around 4% is not extremely influential in terms of complexity because it only indicates that there are on average two progenitors less in the generated trees than in the original trees. That suggests that there are no sparsity issues in the generated trees, which is desirable.

On the right-hand side, there are three examples of consistent generated trees, and as the analysis shows, the generated trees have great complexity in terms of size of branches and placement of branches.

In terms of the KS test, the generated images with six branches only performs very well. It is relatively hard to compare the results to results from the papers [2], [3], since they don’t represent their results numerically, only graphically. However, reading the graphs from paper 3 shows a best KS statistic on the mass distribution of the generated merger trees around ≈ 0.05.

|  |  |  |
| --- | --- | --- |
| **variable** | **KS statistic** | **Nonzero values KS statistic** |
| **Mass** | 0.03775 | 0.10776 |
| **distance** | 0.01667 | 0.14147 |
| **Subhalo type** | 0.00518 | 0.02730 |

One issue regarding these KS test results is the p-value is 0.0 and sample size adapted significance level is less than the KS statistic, which indicates that the null hypothesis is rejected, which means the two distributions are significantly different. See KS section for more details regarding the implications of the p-value, adapted significance level, KS test sensitivity and weaknesses. This is one of the weaknesses of the KS test, it is extremely sensitive. However, the original papers have not commented on the significance level or p-value, which construct an awkward situation regarding the evaluation of the generated distributions since the KS statistic signify a great result, but the p-value indicates a significant difference between the original and the generated distribution.

A graph of value and value

Description automatically generated

The graphical representation of the two distributions further supports the great KS statistic. On the left is the distributions for the distance variable, which shows two comparable distributions with similar values. The only problematic feature of the generated distribution is the peak and quantity of values close to zero. This is an issue which were tried to work around and deal with by modifying the model architecture and loss functions.

Figure 1. Distance distributions

A graph of value and value

Description automatically generatedThe mass distributions convey a similar outcome: two comparable distributions with similar values and peaks at the same points. We see that both distributions have peaks at around 0.8 and 0.95, however, the generated mass value distributions have long tails on both sides of the center of frequency. In fact, the generated mass distribution has values from 0.0001 to 1.25, which is a contrast to the normalized original data which only contains values within a small range from 0.716 to 1.0. When that is said, 93.4% of the generated mass values is within the original range.

Figure 2. mass distributions

The results from the merger trees with six branches only are promising as a first step towards further progress within the GAN architecture. The generated values contain the traits that is desired, the model manages to create some consistent images with the wanted complexity level. Depending on the results on applying the full dataset as training data, the results from this section display evidence that conditioning on the number of branches when training the model could generate great results. Additionally, filtering the generated merger trees in terms of structure consistency could be a possible suggestion when extracting the final generated merger trees, since there are undesired inconsistencies in many generated images. Even though this would be ineffective, it would produce better generated merger trees. However, this depend on which features and structures are the cause for the inconsistencies. Remember that the generated merger trees from paper 3 had 0.0% consistent merger trees, but the inconsistencies were “small” or not significant in the grand scheme.

**How does inconsistent trees look**

Analyzing the inconsistencies in the around 90% of the generated merger trees with six branches shows that many inconsistencies are minor or insignificant. Remember that the generated merger trees from paper 3 had zero consistent image because of minor details.

**A group of images of different colors

Description automatically generated**The inconsistencies in the generated merger trees with six branches confirm that all inconsistencies are present as illustrated with the randomly selected merger trees from the inconsistent generated merger trees. Visually noticeable inconsistencies in images 1 and 4 are gaps in branches, for image 8 it is gap in a branch and gap between branches, for image 5 it is just a gap between branches. But for image 6, there are no visually obvious inconsistencies.

The inconsistency in image 6 is that in 0.9% of the pixel locations don’t have zero across all three channels, which translate to one pixel where the generated subhalo type variable is zero, while the other two generated variables are nonzero. That is the complete analysis of the inconsistency of image 6, which is very insignificant in the grand scheme. An image like image 6 could easily be qualified as consistent or could be fixed with some quick-fix modification and postprocessing.

What about a visually obvious inconsistent image like image 4? Image four has an obvious gap in branch in column 4. Additionally, the analysis shows that image 4 has inconsistent zero/nonzero in 0.87% of the pixel locations and only 93.10% of the distance variable in the main branch is zero. That translates to two generated distance values in the main branch which is nonzero (0.0679 and 0.0619), one generated subhalo value which is zero within a branch where the other two variables are nonzero, and of course the obvious lonely presence of a progenitor in the top of column 4. The two first inconsistencies are insignificant, and easy fixable: since the distance variable in the main branch should be zero, all main branch distance values could technically just be mapped to zero. This would fix some inconsistencies. The generated subhalo value that is zero where the other two variables are nonzero, which is an inconsistency in image 6 as well, could be fixed with a mapping to the above or below subhalo value to create further consistency. In terms of the gap within the branch, the “lonely island progenitor” in column 4, it is a more significant and obvious inconsistency. However, it could be fixed by some postprocessing like deleting the “lonely island”, but that would require further caution regarding which part of the branch to “delete”. Regarding gaps between branches, which is seen in image 5 and 8, a solution would be to relocate the branches that appear after a gap next to the closest branch. An illustration of this would be to “delete” the empty branches between the populated branches.

Even though postprocessing like the one mentioned would increase the consistency rate of generated images, the approach has some shortcomings. One caveat regarding postprocessing to minimize inconsistencies is that it is not a methodologically complete procedure. The merger trees would become almost surgically modified, which would reduce and discredit the “generative” part of the merger trees. But as mentioned, one approach would be to only utilize consistently generated merger trees, even though that approach would be ineffective since only around 10% of the generated merger trees with six branches are consistent.

**Full dataset**

For the full dataset, there are four normalization-activation function combinations that have been applied to train four different models. The combinations are normalized data with relu activation, normalized data with sigmoid activation, original unnormalized data with relu activation and original unnormalized data with sigmoid activation (from paper 2 and 3), where the activation function refer to the last layer of the generator. The analysis and results can be found in notebooks 12 and 13.

The first thing to observe is the original unnormalized data with sigmoid activation function combination. This combination, which is assumingly what Robles et al. (2019) and Robles et al. (2022) applied, does not make any sense. The original data is not normalized, which implies values outside the 0 - 1 range, that is especially true for the mass variable, which only contains zeros or nonzero values in the 7.64 – 10.98 range. Mass is arguably the most important variable in merger trees due to the mass assembly history, and when the original mass values are above 1, it is impossible to reconstruct the original values when using a sigmoid activation function which are limited to the range 0 to 1. This mismatch leads to a model that obviously isn’t capable to reproduce or generate images that resemble the training data. The results and analysis (notebook 12) also reflect this. Therefore, this combination is erased from the experiment list from this point on.

Here is an overview of the results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **dataset** | **Activation** | **Consistency rate range (10 generations)** | **Consistency rate mean (10 generations)** | **KS statistic mass full (nonzero)** | **Avg number of branches (full)** | **Avg number of branches (consistent)** |
| **normalized** | **relu** | 35.18 – 35.87 % | 35.43 % | 0.0955 (0.1464) | 4.65 | 4.33 |
| **normalized** | **sigmoid** | 0.23 – 0.34% | 0.29 % | 0.0674 (0.3369) | 7.43 | N/A |
| **original** | **relu** | 34.94 – 35.56 % | 35.24 % | 0.1151 (0.5993) | 3.52 | 3.08 |

The results have numerous of interesting outcomes. Choice of activation function have a huge impact on the generated trees and their features in various aspects. The sigmoid function creates merger trees that are inconsistent, only around 0.3% of the generated merger trees when applying the sigmoid activation are consistent. But applying the relu activation to the generator create consistent trees, up to 35.87% consistency. This explains why Robles et al. (2022) got 0.0% consistency in their generated merger trees. Contrary, the sigmoid activation function creates more complex trees with more branches that reflects the original data branch distribution with many generated trees with only 264 out of 38348 generated trees having less than five branches and 94.75% of all generated trees have six or more branches. On the other side, the relu activation creates sparse merger trees that does not resemble the complexity in the original data. Using the relu activation on the original data yield the worst complexity in terms of average number of branches. Of the generated merger trees that the normalized data with the relu activation produce, 22.78% have six or more branches. And for the consistent generated merger trees, only 13.02% have six or more branches, and these only have six or seven branches (no consistent merger trees with eight or more branches). The relu activation result can be explained by the fact that the relu activation function maps all negative values to zero, which leads to more zeroes than using the sigmoid, which technically only approaches zero asymptotically.

One takeaway from these results is that the normalized data outperforms the original data, therefore, the original data will be discarded from further analysis

A graph with blue lines

Description automatically generated

Figure 3. normalized data - sigmoid function nonzero mass distribution

In terms of the KS test, the sigmoid function has the best performance on the full generated data, however, when only looking at the nonzero generated values, the relu function outperforms the sigmoid function. This is because the sigmoid function generates a lot of small values close to zero, which is undesirable. These small mass values generated using the sigmoid function has the potential to be extremely influential regarding the consistency rate of the generated images. In fact, this is true for the other two variables as well. For the distance variable, the sigmoid function possesses the same issue: generating a lot of values close to zero. And for the subhalo variable, the sigmoid function with the mapping function creates an overload of zeros.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Nonzero distance (ratio)** | **Nonzero mass (ratio)** | **Nonzero subhalo (ratio)** |
| **Original (baseline)** | 1 518 348 (0.234) | 2 475 374 (0.383) | 2 475 374 (0.383) |
| **Sigmoid generated** | 2 857 111 (0.396) | 2 602 230 (0.360) | 1 751 466 (0.244) |

For the consistency to be high, the mass and rubhalo ratio should be the similar, since that can indicate that they have the same amount of zero and nonzero, and hopefully in the same pixel locations. The distance variable should have less nonzero values since the distance variable should only have zero values in the main branch, which is a huge fraction of all nonzero values in the other two variables. This is illustrated with the baseline above.

The ratio of nonzero values for the sigmoid generated images are way off the desired behavior. A quick estimation indicated that there are excessively many nonzero distance values and insufficiently many nonzero subhalo values. This ratio imbalance affects the consistency rate drastically, which will be investigated further.

**How does inconsistent trees look**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Zero index** | **Dist zero main branch** | **Gap between branches** | **Gap within branch** | **Last descendant** |
| **Normalized relu** | 45.86 % | 0.00 % | 81.97 % | 34.24 % | 0.27 % |
| **Normalized sigmoid** | 100.00 % | 100.00 % | 92.32 % | 99.93 % | 0.00 % |

Figure 4. Ratio of inconsistencies. This table is based on only inconsistent merger trees.

The merger trees generated using relu function only have three occurring inconsistencies, namely zero index, gap between branches and gap within branch inconsistency. Gap between branches is the most common inconsistency where almost 82% of all inconsistent images possess the gap between branches inconsistency. Around 50% of the inconsistent merger trees have two or more inconsistencies. Overall, 1/3 of generated images using relu activation function is consistent, another 1/3 is inconsistent with only one inconsistency, and the last 1/3 are inconsistent merger trees with two or more inconsistencies.

The merger trees generated using the sigmoid function contain the same inconsistencies as the merger trees generated by Robles et al. (2022). Basically, all images have zero index, distance in main branch is not zero, gap between branches and gap within branch inconsistency. Hence, all generated merger trees have two or more inconsistencies.

**Complexity vs consistency**

As seen in the results above, there is a clear tradeoff between complexity and consistency in the generated merger trees. The complexity/continuity features of the generated trees given a model depending on the activation function used in the last layer of the generator and its attributes and qualities.

Using the sigmoid function, like Robles et al. (2019) and Robles et al. (2022) produce bigger and complex trees with more branches. However, even though these generated merger trees look fairly good, they are highly structural inconsistent with multiple inconsistencies. The inconsistencies occurring in the generated merger trees can be a multi-inconsistency due to a single small mistake, like for example a very small nonzero distance or mass value in a spot that where the other two variables are zero. This could lead to multiple discontinuities because of one tiny “off-value”. This is the issue with the sigmoid activation function, that it technically only approach 0.0 asymptotically, so all values who should have been zero, are not necessarily equal to zero.

Using the relu activation function as the last layer in the generator produce more zero values due to a sharp cutoff. This leads to more zeroes and sparser generated merger trees. This means more complex and more consistent merger trees.

**Compare generated using 6 branches only vs full dataset**

One thing to note is that the model trained on merger trees with only six branches manages to capture more of the complexity of the training data while achieving a decent continuity. The generated merger trees are relatively complex and contains a lot of progenitors and branches. This opens up for the idea and possibility of introducing a solution where a model is trained with only merger trees with a given number of branches or condition the model on the number of branches in a merger tree. This approach could potentially lead to a good balance between complexity and consistency.