**Data preprocessing**

**Note:**

* **Make sure to explain the importance of predicting 0.0 and not 0.01**
  + **That the difference between 0.88 and 0.89 is the same in terms of loss as 0.01 and 0.0, but in terms of structure and what it means, it is completely different**

From the data analysis conducted in the previous part, there are two main questions regarding data processing in this problem. The first is normalization, and the second is how to handle the categorical values of the subhalo variable. There are three different variables, which needs different treatment and handling. All work regarding decisions made in this section is in notebooks:

* 1 – data analysis
* 2 – data processing
* 3 - Autoencoder - data processing test distance
* 4 - Autoencoder - data processing test subhalo
* 5 - lower and upper bound for subhalo mapping

**Mass**

The mass variable is the easiest one to deal with. It has fairly consistent nonzero values between 7.6377 and 10.9763, with a mean of 9.1345. The nonzero values have some sort of bi-distribution with peaks at around 8.5 and 10.0. And according to IQR outlier detection theorem, there are no outliers in the mass variable. The only thing needed to preprocess mass, is to normalize it. Since all values for all variables in the dataset is 0 or positive and describe either the absence of a progenitor (value = 0.0), or the presence of a progenitor if the value of any matrix is positive. Since the value is strictly zero or positive, it makes sense to normalize the values so that they are between 0.0 and 1.0, in other words, using the min-max scaling normalization process. This is also in line with image representation, where many image libraries requires pixel values between 0 and 1 or 0 and 255. Mass will therefore be normalized using the min-max normalization process which will yield mass values between 0 – 1. After normalizing, the nonzero mass values are between 0.6958 and 1.0 with a mean of 0.8322, and the distribution is similar to the original data as expected.

**Distance**

For the distance variable, we have a big issue where the majority of nonzero values are extremely close to 0.0, in fact, the mean of the nonzero values is 0.0657. This can make it hard to separate 0.0 and positive values in the generated images. In fact, when running the autoencoder test on the encoder and decoder of the CGAN, the model is struggling to reconstruct the distance variable. There are multiple aspects where it struggles:

* Reconstructed distance values have nonzero values in the main branch. Remember, the distance variable is distance from the main branch, so the main branch should have 0.0 values.
* Reconstructed distance values have nonzero values in random spots. These nonzero values are very low, but they are nonzero nonetheless, which means they predict the presence of a progenitor when there are none. This is a big discontinuity for the galaxy merger trees, because the presence of a progenitor means it should be nonzero in all 3 variables. And if the distance variable has nonzero values in random spots, it is a huge problem in terms of continuity with the other two variables.
* A screenshot of a video game

  Description automatically generatedReconstructed distance values have a discontinuity in terms of actual value when the reconstruction is good. When the reconstructed distance values assign a nonzero value to the correct pixel, it is usually not the correct value, or close to the correct value. Distance values should have a continuity in the same branch since progenitors in the same branch usually have the same distance to the main branch.

Figure 1. distance reconstruction using autoencoder. real distance values to the left, reconstructed to the right

On the right we can see all three aspects where the model fails to reconstruct the distance values. The reconstructed image on the left has nonzero values in the main branch (the leftmost column). It is fairly hard to see, since the reconstructed values are very close to 0, which is the issue I suspected. The reconstruction of the distance value also has spots on the right of the image, where there are a lot of spots with very low nonzero values. Lastly, we see that the original distance value has some sort of continuity in terms of distance within a branch, green follow green, yellow follow yellow, without any big jumps or mix between colors. This is because the distance to the main branch in a branch is very similar. But the reconstructed distance fails to capture this relationship, and even though it manages to reconstruct the structure fairly well, the actual values are off and doesn’t follow the same relationship.

There are multiple ways to deal with these low values of the distance variable. Standard normalization methods will keep the same structure of the values, keeping many values close to 0.0, therefore other methods need to be evaluated. The goal of the distance processing is to bump the low values and squeeze the high ones, so that the low values are separable from 0.0. To manage to distinguish and separate zero/nonzero values is extremely important since it is a key feature of the merger trees to be able to separate between absence/presence of progenitors, which is essentially what zero/nonzero is. This is one of the most important features of the merger tree, and if it is wrong during reproduction or generation, it will lead to inconsistencies in the merger tree.

**Log normalization**

One method that can give the desired results is a log normalization. Using the normalization function: x’ = log(c \* x + b), we can scale and normalize the values between 0 and 1. First of all, b needs to be equal to 1, so that normalization of 0 is 0. Secondly, to squeeze the end in to 1, c needs to be equal to (10 – 1) / max(dist). However, this normalization only increase the average nonzero value from 0.0657 to 0.1207, which is not a lot, and the majority of distance values are still close to 0.

**Log normalization with high scale**

Another approach using the log normalization would be to set b = 1, and c = large, then normalize with a regular normalization method. Setting c = large would push the values close to 0 up fast, then the log function would settle slowly. For example, setting c = 1000 would mean the log normalization of 0.0 would be 0, the log normalization of 0.01 would be 1.04 and the log normalization of 1.775 (max distance value) would be 3.25. this would indeed bump the low values and squeeze the high values. This transformation would do the following transformation to the nonzero distance distribution

**A graph of values with blue lines

Description automatically generated**

**A blue graph with white text

Description automatically generated**

🡪

Doing a min-max normalization on the this log-scaled distance variable will give values between 0 and 1 where the mean nonzero value is 0.5318. The minimum nonzero value is 0.0323, which is still close to 0.0. However, this transformation and normalization only as 0.14% of all nonzero values between 0.0 and 0.2, whereas 99.17% of the original nonzero distance values where between 0.0 and 0.2.

**Clipping and minmax normalization**

Another approach is to clip the high values, then normalize. According to normal IQR outlier detection theorem, the 1.83% highest values are outliers. That corresponds to all values over 0.1745 being outliers. One idea would be to clip all values above Q3 + 2 \* IQR down to Q3 + 2 \* IQR = 0.2027, then normalize the data. To make it easy, I clip values above 0.2 to 0.2. Minmax normalizing the clipped data gives a nonzero average of 0.3263, which is a massive increase. The number of values between 0.0 and 0.2 have decreased from 99.17% to 32.36%, which also is a desired result.

**Clipping and log normalization**

Clipping in the same way as described above then applying a log transformation with b = (10 – 1) / max(clipped\_values) puts all values between 0 and 1 and shift the values more to the right.The new nonzero mean is 0.5475 and only 5.14% of the nonzero values are between 0.0 and 0.2.

**Adding 0.5 to all nonzero values**

Last alternative is to simply add 0.5 (or another positive value) to all nonzero values, so that it creates a cutoff between zero values (absence of progenitor) and positive values (progenitor present). With this transformation and normalization, the mean nonzero value is 0.2509, and no values is between 0.0 and 0.2. This will hopefully make it easier to separate zero valued pixels and nonzero valued pixels when reconstructing and generating new images.

**Testing distance with reconstructed values with autoencoder**

The code and evidence for this section is in 3 - Autoencoder - data processing test notebook.

To reconstruct the images, the setup is a small encoder-decoder architecture inspired by paper 2 and 3 as an autoencoder with MSE loss. The autoencoder didn’t manage to reconstruct 0.0 values at all, no matter which normalization method is used for the distance variable. In fact, the autoencoder didn’t manage to reconstruct a 0.0 value for any of the variables, which is a big problem. The issue lies within the model, which has a Sigmoid activation function as the last layer of the generator. This makes it impossible to predict 0.0 since the sigmoid function only asymptotically approach 0.0. The original papers 2 and 3 uses a sigmoid function as the last layer in the generator/decoder, which would not manage to reconstruct or generate 0.0, which means it wouldn’t be able to generate absence of progenitor, which basically mean the generation would only be progenitors all over the merger tree. However, I will discuss more about the papers and their choices later.

* **Relu activation function**

Changing the activation function to relu makes it possible to predict 0.0, and the model should be able to learn how to restrict the positive values. The distance normalization techniques I tried was log scaling with minmax normalization, clipping values above 0.2 with minmax normalization, and 0.5 to each value with minmax normalizing. When using the relu activation function, all results managed to predict 0.0 in the absence of a progenitor. Following are the results compared to the original images which contains 86.35% 0.0 values. Note that the autoencoders are only trained for 20 epochs.

Here are the distance variable data results from the former relu activation function normalization experiments.

|  |  |  |  |
| --- | --- | --- | --- |
| **Normalization method** | **Zero predictions** | **Difference from original images** | **Absence/presence similarity to OI** |
| Normal minmax normalization (reference) | 85.01% | - 1.34 % | 98.28% |
| Log scaling with minmax normalization | 85.94% | - 0.41% | 99.56% |
| Clipping values over 0.2 to 0.2 with minmax normalization | 86.14% | - 0.20% | 99.70% |
| Pushing nonzero values by 0.5 then minmax normalization | 84.83 % | - 1.51% | 98.46% |

We see that the log scaling and clipping perform much better than the pushing normalization. However, since capturing the absence/presence of progenitors are an extremely important part of this generation task, further experiments are necessary to optimize this.

* **Different loss function**

We see that none of the normalization methods overperform in terms of predicting 0.0, they predict less zeros than the original images. Using a MSE loss function that penalize if the generator predicts a nonzero value when the original value is zero is an option in optimizing the reconstruction.

The loss function look like this if we scale with 10:

|  |  |  |  |
| --- | --- | --- | --- |
| **prediction** | **target** | **Loss MSE** | **Scaled loss** |
| 0.3 | 0.0 | 0.09 | 0.90 |
| 0.3 | 0.5 | 0.04 | 0.04 |
| 0.2 | 0.1 | 0.01 | 0.01 |
| 0.1 | 0.0 | 0.01 | 0.10 |
| 0.2 | 0.2 | 0.00 | 0.00 |

Total loss without scaling = sum(loss MSE) = 0.15

Total loss with scaling = sum(scaled loss) = 1.05

Scaling nonzero reconstructions on zero targeted pixels by 10 makes the normalizations reconstruct more zeros than the original images, they reconstruct around 0.20% more zeroes than what the original images have.

Scaling (penalizing) by a factor of 5 gives the following results:

Remember the original images contains 86.35% 0.0 values in the distance variable, and 80.61% of zeroes overall.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Normalization method** | **Zero predictions (distance)** | **Difference from original images (distance)** | **Absence/presence similarity to OI (distance)** | **Zero predictions (overall)** | **Difference from original images (overall)** |
| Log scaling with minmax normalization | 86.29% | - 0.06% | 99.83% | 80.50% | -0.11% |
| Clipping values over 0.2 to 0.2 with minmax normalization | 86.45% | + 0.10% | 99.78% | 80.59% | - 0.02% |
| Pushing nonzero values by 1.0 then minmax normalization | 84.54% | - 1.81% | 98.18% | 79.48% | - 1.13% |

Clearly, pushing nonzero values by 1.0 or 0.5 is the worst of the three normalization methods. Interestingly, clipping overshoots on distance, but is only 0.02% off on all three channels. While log scaling undershoots slightly by 0.06% on distance and undershoots with 0.11% on all three channels.

Next, I will train the autoencoder on the log scaling and clipping normalization over more epochs, to see if we can optimize the reconstruction further. Here are the results trained over 100 epochs:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Normalization method** | **Zero predictions (distance)** | **Difference from original images (distance)** | **Absence/presence similarity to OI (distance)** | **Zero predictions (overall)** | **Difference from original images (overall)** | **Absence/presence similarity to OI (overall)** |
| **Original data reference** | **86.35%** |  |  | **80.61%** |  |  |
| **Referemce, normal minmax normalized** | 86.48% | + 0. 13% | 99.77% | 80.65% | + 0.04% | 99.86% |
| Log scaling with minmax normalization | 86.33% | - 0.02% | 99.97% | 80.56% | -0.05% | 99.94% |
| Clipping values over 0.2 to 0.2 with minmax normalization | 86.38% | + 0.03% | 99.92% | 80.60% | - 0.01% | 99.92% |

Using the big convolutional model proposed in paper 2 with relu instead of sigmoid give the following results over 20 epochs:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Normalization method** | **Zero predictions (distance)** | **Difference from original images (distance)** | **Absence/presence similarity to OI (distance)** | **Zero predictions (overall)** | **Difference from original images (overall)** | **Absence/presence similarity to OI (overall)** |
| Log scaling with minmax normalization | 86.29% | - 0.06% | 99.93% | 80.58% | - 0.03% | 99.95% |
| Clipping values over 0.2 to 0.2 with minmax normalization | 86.38% | +0.04 | 99.88% | 80.6% | -0.01% | 99.93% |

Using the normal minmax normalized data didn’t manage to recreate the distance variable at all.

Moth log scaling and clipping seem more stable than regular minmax normalization which is struggling with capturing the distance structure, have some discontinuities, and doesn’t manage to reconstruct the distance variable in some cases. Both normalization methods have similar output, but clipping high values before normalizing destroys some of the original information, since the high outliers are clipped and bounded to 0.2. The benefits of log scaling are that only 0.14% of distance values are between 0 and 0.2, compared to 32.36% with clipping. However, with the relu activation, it seems like the values close to zero shouldn’t be that big of an issue for neither normalization methods. Additionally, log scaling doesn’t destroy any original information, which might be important. The log scaling normalization is also very easy to work backwards with. Therefore, the log scaling normalization will be the standard distance normalization method for this paper.

**Subhalo**

Normalizing the subhalo category doesn’t really make sense since it is a categorical value described with floats. Therefore, it will not be done. However, since the subhalo categories are represented as floats with values 0.0, 0.5 and 1.0, there are some issues that can occur. Representing a categorical value with floats is not ideal since it assigns closeness to the categories. However, in this specific case, there is some closeness to the categories, since being a subhalo is closer to a main halo than no halo present. Secondly, representing these categories with floats makes the generated and predicted values floats, which will not be accurate unless the model manages to learn the representation extremely well. What I mean by that is that if the model generates a float value other than 0.0, 0.5 or 1.0 for the subhalo category, it is not an acceptable value for that category. 0.495 is not technically anything in terms of subhalo since the only true values are 0.0, 0.5 and 1.0. Unless the model manages to learn this, it is an issue that needs to be solved.

There are multiple other options to process the subhalo variable that could turn out to be useful later one. One example would be to one-hot encode the variable and extend the subhalo channel into three channels to represent progenitor not present, progenitor is a subhalo, and progenitor is a main halo. This would solve both issues with the float representation of the categorical subhalo variable. The downside with this approach is that it requires preprocessing, and processing within the model or after the model have generated images.

Another less mathematically sound approach would be to use the original subhalo values of 0.0, 0.5 and 1.0 in the reconstruction and generation, then map the generated values back to 0.0, 0.5, and 1.0 depending on thresholds. The thresholds could either be learned or assigned given an analytical approach.

I will explore both approaches presented here.

From a test in the Autoencoder notebook, I analyzed which values for upper and lower bound was the best in terms of mapping back the generated subhalo. Using a lower bound of 0.19 and an upper bound of 0.77 got a total pixel-wise similarity between the subhalo variable of the original pictures and reconstructed pictures using the map function of 99.58%, and a total of 17594 / 38348 = 45.88% of the reconstructed images were 100% identical to the equivalent original image.

This result of this approach is decent, but it could create some issues, for example if the lower bound doesn’t catch all “suppose-to-be-0.0”. Then the mapping function would assign a subhalo to a spot were there shouldn’t have been a progenitor. This could lead to some multiple discontinuities described in the consistency & properties section.

Here are the experiments and results from testing different preprocessing method for the subhalo category. All results are documented in the 4 - Autoencoder - data processing test subhalo notebook.

* **No preprocessing, using different loss functions and mapping**

The first approach is straightforward: No preprocessing of the subhalo variable, which means the output will be floats that may not exactly be 0.0, 0.5 or 1.0. However, the generated values will be mapped after passing it through the autoencoder to see how well it performs. Additionally, different loss functions that penalize if the target is 0.0, 0.5 or 1.0 will be utilized.

**A graph with blue lines and numbers

Description automatically generated**The first attempt used the same loss function from the distance preprocessing which penalize if the target is 0.0, with a penalization scale of 5. After training this autoencoder for 100 epochs, we see that the reconstructed subhalo values centers around 0.0, 0.5 and 1.0. The minimum predicted nonzero value is 0.00002 and the maximum predicted nonzero values is 1.26355, which means there are multiple values that are not exactly 0.0, 0.5 or 1.0.

Looking at these values (rounded to 2 decimals), there are 118 unique values ranging from 0.0 to 1.19. There is a soft gap, meaning the values in the gap only have single digit counts, between 0.05 - 0.30, and 0.69 – 0.81, which looks promising for the lower and upper bound proposed of 0.19 and 0.77. The hard gap, meaning there is no values in that range, is between 0.12 – 0.18.

However, no reconstructed subhalo variable is 100% equal to the original image and only 77.73% of the reconstructed and original image in terms of pixels have the same value in terms of zero/nonzero values or absence/presence of progenitor. The positive thing is that all of these equal values are 0.0’s, which means the reconstructed and original image have 0.0’s in 99.90% of the same spots. This implies that even though the values are not the same, the absence/presence of progenitors’ structure is extremely well captured.

Now to the fun part: mapping the reconstructed values to 0.0, 0.5 and 1.0 using a lower bound of 0.19 and an upper bound of 0.77. Meaning if the reconstructed subhalo value is below or equal to 0.19, it will be mapped to exactly 0.0, if the reconstructed subhalo value is between 0.19 and 0.77, it will be mapped to exactly 0.5 and if the reconstructed subhalo value is above or equal to 0.77, it will be mapped to exactly 1.0. Now how does this change the result?

First of all, now the reconstructed subhalo variable only contains 0.0, 0.5 and 1.0, which is extremely good, since those are the only valid values. Average similarity, that is having the exact same value, pixel-wise, between the reconstructed and the original image is 99.94%, which is extremely good. And the number of reconstructed images that are 100% equal to the original image is surprisingly high at 85.32%. That means 85.32% of all images have been perfectly reconstructed in terms of the subhalo values, and the reconstructed subhalo values are identical to the original image.

This result is extremely good, but can it be better? What if instead of penalizing if the target is 0.0, we also penalize the loss if the target is 0.5 and 1.0? That is what the next three experiments do. The penalizing factor will be 5, 10, and 100, and the results are summarized in the table below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Penalize factor**  **(no mapping)** | **Min** | **Max** | **Num unique values** | **Soft gap lower** | **soft gap upper** | **Hard gap lower** | **Hard gap upper** |
| **reference** | 2.00e-5 | 1.26355 | 118 | 0.05 – 0.30 | 0.69 – 0.81 | 0.12 – 0.18 | - |
| **5** | 1.27e-07 | 1.19018 | 99 | 0.10 – 0.36 | 0.64 – 0.84 | 0.18 – 0.30 |  |
| **10** | 4.02e-0.7 | 1.20349 | 81 | 0.07 – 0.38 | 0.61 – 0.88 | 0.12 – 0.32 | 0.63 – 0.81 |
| **100** | 1.49e-07 | 1.17348 | 69 | 0.06 – 0.41 | 0.59 – 0.90 | 0.10 – 0.39 | 0.63 – 0.84 |

Here we see a lot of interesting results when changing the loss function to penalize when the target is 0.0, 0.5 and 1.0 and changing the scale. We see that the number of unique reconstructed values (rounded to 2 decimals) decrease, the soft gap in the lower end gets wider, the same is true for the soft gap in the upper end. Additionally, the hard gaps in the lower end gets very big, and 0.19 is within all the hard lower gaps, and when penalizing with 10 and 100, it is almost in the middle of the hard gap as well. There is no hard upper gap when penalizing with 5, but when penalizing with 10 and 100, there is hard upper gap which contains 0.77.

These results are promising, and all of the penalizing factors manages to capture the absence/presence of progenitor structure well, but the values are the main issue because they don’t reflect the true values of 0.0, 0.5 and 1.0. However, these results look extremely good in terms of the mapping function since the gaps increase and contain the lower and upper bound. Following are the results after applying the mapping to each of the penalization experiments:

|  |  |  |
| --- | --- | --- |
| **Penalize factor**  **(mapped)** | **Average similarity** | **100% equal reconstruction** |
| **5** | 99.88% | 97.13% |
| **10** | 100.00% | 99.33% |
| **100** | 99.98% | 95.88% |

The results without any special care were surprisingly good with 85.32% of all reconstructed images being identical to the original images in terms of the subhalo variable. But with the additional penalization, the performance improves, and by a lot! The best results are when penalizing with 10, which gets a 100.00% similarity in terms of pixel-wise subhalo values to the original images (there might be some small error with rounding), which is perfect. Additionally, an astonishing 99.33% of all reconstructed images are identical to the original images in terms of subhalo values.

These results show that there might not be need for extra processing of the subhalo variable, since the result with penalizing loss and the mapping of the reconstructed subhalo variables performs extremely well, and it will be hard to beat these results. However, it is worth taking a shot at other approaches.

* A close-up of a logo

  Description automatically generated**One hot encoded subhalo variable, argmax and one-hot encode**

The next approach is to one-hot encode the subhalo variable. In the forward method of the model, the argmax function will be used on the subhalo variable to determine which of the three categories is the most probable for reconstruction. Since the argmax reduces dimensionality, the subhalo variable will be one hot encoded again after the argmax operator to obtain the original structure of the data which is one-hot encoded.

One problem with this method is that when calculating the loss, it become extremely hard since either it is correct on all three subhalo channels, or it is wrong, which means two of the channels get a very hard loss (since it is completely wrong). This makes the model guess the next time if it is wrong, since it has not idea of knowing “how wrong” it is, and not a lot of real learning is happening. The result of this is that the model doesn’t mange to reproduce the variable at all. On the right we can see the original image on the top, with the reconstructed image on the bottom. To some extent, it looks like the reconstructed image just randomly guesses where to but the different values.

There is no surprise given these images that the result is poor. There are 0 reconstructed images that are identical to the original image, and only 31.07 of the pixel-wise values are the same, which seems like random guessing.

* **One hot encoded subhalo variable, softmax**

**A purple and yellow rectangles

Description automatically generated**Another solution to the one-hot encoded subhalo variable is to use the softmax operator in the forward function. The softmax operator converts the reconstructed one-hot encoded values into probabilities. This makes it easier for the model to learn since it has a sense of “how wrong” it is, since all one-hot encoded channels gets a probability and can be corrected.

This method works much better than the argmax method. However, the model struggles to predict 0.5 and 1.0, especially in the beginning. And at the end, the reconstructed images have 0.0 in 90.72% of all pixels, while the original images only have 0.0 in 77.74% of al pixels. Furthermore, the reconstructed and original image have zero/nonzero values in 87.03% of the same spots, so parts of the structure is captured. We can also see that on the images to the left, where the reconstructed image (bottom) seems to reconstruct correctly when it predicts 0.5 and 1.0, the issue is that this occurs infrequently with inconsistencies. One strange pattern is that the reconstructed images fail to reconstruct the main branch, which seems like the easiest part.

These results were the best results after trying with different loss scaling. Here, the loss only imposes a penalty if the target is 0.5 and 1.0, and the penalization factor is only 2. Other combinations of loss resulted in even more 0.0 in the reconstructed image.

The result here is promising, but not useful, especially not compared to the mapping results.

**FINAL data processing based on early results**

After some careful and comprehensive data analysis and data processing experiments, the final data processing methods for the rest of the paper have been assigned. The biggest impact in terms of reconstructing the absence/presence of progenitors was using the relu activation in the generator instead of the sigmoid, which paper 2 and 3 used.

Next up, the mass variable has a nice original distribution, and will only be minmax normalized so that the values are between 0 and 1.

The distance variable has an extremely low minimum and mean, with some outliers on the high end. To bump the lows and squeeze the highs, a log scaling was applied to the distance variable, before minmax normalizing it to the range 0 and 1. After the log scaling, the distance variable had similar distribution to the normal distribution with mean 0.5.

In terms of the subhalo variable, no normalization will be applied. This is mainly because the subhalo variable is categorical. One option is to one hot encode the categories, but experiments show that the best option is to keep the variable as is. However, to resolve the issue of predicting floats with no real categorical meaning, applying a mapping function to map the predicted values to exactly 0.0, 0.5 and 1.0 performed extremely well and will be utilized further.

The last takeaway from the data analysis and preprocessing experiments is that using a loss function that penalize if the target is 0.0 bumped the results. This is to emphasize the importance of predicting and reconstructing the structure of the data. The most important part of the structure is the absence/presence of progenitors. Penalizing small values close to 0.0 extra will hopefully force these values to 0.0, which has a lot of impact since 0.0 equals absence of progenitor, while 0.000001 equals presence of progenitor.

Developing this penalization loss further by penalizing if the target is 0.5 and 1.0 as well increased the subhalo variable performance drastically. The idea is the same as penalizing if the target is 0.0, and by also penalizing if the target is 0.5 and 1.0, the model will hopefully manage to separate and distinguish reconstructed values that are in between 0.0 and 0.5 or 0.5 and 1.0, which we saw proof of in terms of the soft and hard gaps when experimenting with this loss. Applying the mapping function on top of this boosted the results further to an extremely good performance.

Penalizing if the target is 0.0 will affect all three variables positively, since it is a question of absence/presence of progenitor, which is a factor in all variables. Penalizing if the target is 0.5 or 1.0 will only affect the subhalo category, since there is no or very few values in the distance and mass variable which is exactly 0.5 or 1.0, and if there is, these values will be enforced to be reconstructed.