Generative Temporal Models for Cosmology

Master Thesis
Jonathan Rosenthal
Saturday 1st September, 2018

Advisers: Dr. T. Kacprzak, Dr. A. Lucchi, Dr. N. Perraud
Supervisor: Prof. Dr. T. Hofmann
Department of Computer Science, ETH Zürich
Abstract

In order to better understand the history of the universe and dark matter distributions, cosmologists run n-body simulations. Such simulations involve large amounts of points which can all effect each other. The computations are very expensive and evaluated at discrete points in time. In this project we come up with a novel model architecture within the generative adversarial network [1] framework. The model is able to generate data for any arbitrary continuous point in time as well as more data at any other arbitrary point in time which may plausibly stem from the same sequence. While this project focused on time, the model is general enough that it could be conditioned with any continuous variable. The performance of our model matched the performance of the specialized non-continuous models we trained for the project. The generated data was analyzed from both a qualitative as well as a quantitative point of view. We also developed the mass difference histogram as a metric to analyze change over time within the n-body data.
Contents

1 Introduction and Motivation 1

2 Generative Adversarial Networks (GAN) 5
   2.1 Wasserstein GAN ................................. 6
   2.2 Relativistic GAN ................................. 7
   2.3 Conditional GAN ................................. 7

3 N-Body Simulations 9
   3.1 Cosmic Time: Redshift .......................... 9
   3.2 Quantitative Metrics for N-Body Systems ........ 10
      3.2.1 Mass Statistics ............................ 11
      3.2.2 Power Spectral Density ..................... 11
      3.2.3 Peak Statistics ............................ 11
      3.2.4 Mass Difference Histogram .................. 11
      3.2.5 Comparing Curves ......................... 12

4 The Data 13
   4.1 Points to Images ................................ 13
   4.2 Transformation .................................. 14
   4.3 Distribution ..................................... 15

5 Continuous Conditional Generative Adversarial Networks 19
   5.1 Conditional GANs and Prior Work ................ 19
   5.2 Proposed Architecture ........................... 20
      5.2.1 Generator .................................... 20
      5.2.2 Discriminator ............................... 22
      5.2.3 Specifics for the n-Body Problem .......... 23
      5.2.4 Optional Features ......................... 23
   5.3 Encoding a Continuous Variable in the Latent Space . . 24
5.3.1 Direct Encoding ................................................. 24
5.3.2 Full Vector Scaling ............................................ 24
5.3.3 Partial Vector Scaling ........................................... 24
5.3.4 Multiple Continuous Variables ................................. 24
5.3.5 The Nonlinearity of Time ..................................... 25

6 Results ............................................................. 27
   6.1 Experiment Setups ................................................ 27
   6.2 Specialized Single Redshift Models ............................ 28
      6.2.1 Spectral Normalization .................................... 28
      6.2.2 Scaled Exponential Linear Units ........................... 29
   6.3 Continuous Conditional GAN ................................... 37
      6.3.1 Quantitative Statistics Compared to Specialized Models 37
      6.3.2 Quantitative Comparison of Continuous Variable Encodings .................. 40
      6.3.3 Qualitative Analysis ......................................... 46

7 Conclusion and Future .............................................. 49

A Appendix .......................................................... 51
   A.1 Visualizations of Transform Parameters ..................... 51
      A.1.1 Shift and c by Redshift with Fixed Scaling .............. 51
      A.1.2 Shift and Scaling by Redshift with Fixed c ............. 54
      A.1.3 Scaling and c by Redshift with Fixed Shift ............. 57
   A.2 Quantitative Statistics for Models With and Without Spectral Norm and SELU ........................................... 60
   A.3 Images Generated by Continuous Conditional GANs ......... 61

Bibliography .......................................................... 65
Chapter 1

Introduction and Motivation

The history of the universe is a topic which people have been fascinated by for centuries. It was not until very recently that we have had the tools to objectively research this problem.

In the 1920’s there were major breakthroughs which helped answer that question. First, George Lemaître was able to show that the universe seemed to all come from one single point\(^1\). Two years later Edwin Hubble was able to prove that galaxies are drifting apart [3]. These two observations form the foundation for the Big Bang theory.

Over time our understanding of the universe and its evolution has not stopped improving. Arguably our tools for understanding the problem have improved even more. With the help of computers, researchers have been creating simulations to visualize how the universe might have looked like at different points in time. It is the nature of these simulations that they are extraordinarily expensive from a computational point of view. To make matters worse, the computational complexity of such simulations tend to be superlinear.

Neural Networks Machine learning has allowed problems to be solved which many people considered near impossible just a couple of decades ago. At the forefront of these developments are neural networks. The modern implementations of these special algorithms often contain many millions of parameters and require specialized processors in order to train with any reasonable efficiency.

The prize for being able to train such a neural network is great. Indeed, Hornik et al. [4] were able to prove that multilayer feedforward networks with even just one hidden layer are capable of approximating any arbitrary

\(^1\)The paper was translated to English in 1931 [2]
1. Introduction and Motivation

continuous function up to any arbitrary precision, given enough hidden units.

Unfortunately the previous paragraph comes with some large caveats. First, while a neural network may be capable of approximating any continuous function, it will only actually do that if the weights have the right values and the architecture has a suitable size and form.

Thanks to the huge amounts of computational resources available today both of these issues have become more or less solvable. Simply trying out different network architectures and keeping the one which performed best has become a viable method. As to the question of setting the weights correctly, the gradient backpropagation method has proven itself to be very effective. Armed with these tools researchers have been able to develop things such as self driving cars and methods to recognize cancer cells in images from under a microscope.

A differentiable loss function which can compare a networks output with the correct value for a training sample is all that is needed. For most problems this is the case, but there are exceptions, as we will see in a moment.

From Classification to Generation When people think of function approximation, they usually think of things like predicting the stock market or the weather. It is very easy to define a ground truth value for an output; the value that happened in the reality.

Now consider the problem of filling in missing pixels of an image. Here again we could consider the exact pixels from the original image as the ground truth, however in many cases different values could fill up the missing pixels and we would consider them reasonable. For example, if we have images of the Eiffel tower and the pixels showing the tourists below are missing, then we would not require an algorithm to replace the missing pixels with one specific set of tourists as long as there is some arbitrary set of tourists being shown. Since there is no longer a single ground truth, it becomes harder to define a loss function that can define how “correct” something is that the network creates.

We can go even further and assume the entire image is missing. In this case we would want the network to create anything from the same distribution as our dataset. In this case we call our network a generative model, the goal is to create new data point for a given set of data from the same distribution.

Only very few approaches have proved successful in creating a proxy for a potential loss function. The most popular of these approaches at the moment is the generative adversarial network approach [1]. This trains a second model to define how “correct” and image is in order to define differentiable loss functions over both networks.
Generating a History of the Universe  Being aware of generative models as well as ongoing research into the history of the universe, it becomes tempting to create images of the universe over time. Unfortunately, this involves a more difficult problem than simply generating random images. If we look at multiple images over time from specific points in time there are two constraints we need to deal with.

The first is class consistency. Images close to the big bang need to look like a Gaussian random field whereas images from distributions closer to today should contain more structure. In order to deal with this our generative network needs to be made aware of the additional constraint we are putting on it.

The other constraint is continuous (temporal) consistency. It is unlikely that a large cosmic structure is built over a longer period of time in part of an image and then suddenly completely vanishes. Changes need to be gradual and as time is continuous our generative model needs to be able to generate images for any continuous point in time. This last point makes this problem more interesting as our data is not continuous and contains only samples from a small, finite number of points in time.

Motivation  While this problem stems from cosmology, it is actually just a special case of a much more general problem. It is a generalization of generating videos with GANs. Instead of just being able to generate frames the problem we would like to solve is to generate any arbitrary frame of a video or even points in time between videos.

Furthermore if we can generate a sequence of data on a variable which is not time, then there should be no reason our model could not learn such a difference as well.

Though the focus in this project will be working with images in principle the underlying problem is more general than that. If we can modify arbitrary continuous variables one could see a world where a model is able to generate medical outcomes dependent on age or weight, so a doctor could more precisely decided which procedure is right for which patient.

Continuous Conditional GAN  The primary contribution of this thesis is the Continuous Conditional GAN\textsuperscript{2}. Which is a model architecture that can be trained within the generative adversarial network architecture to be conditioned with an arbitrary continuous variable. The model is able to generate data for any value of the conditional variable, including for values not in the training data. The architecture is completely generic and can deal with any arbitrary kind of data and any kind of continuous conditional variable.

\textsuperscript{2}See chapter 5.
1. **Introduction and Motivation**

The model has two primary downsides.

Training samples may not be sparse in the sense that if training data contains \( n \) different values for the conditional variable, then all samples used in training must be available for each of these \( n \). That is to say the model has not been extended to deal with missing values.

Furthermore, the model has a significantly larger batch size for the generator, dependent on \( n \), than in the discriminator. This can slow down training significantly and make training between the generator and discriminator more asymmetrical and difficult.

Finally we came up with a metric to measure the degree of change between two data samples. It is a rather obvious metric, which has likely been done before (even for \( n \)-body simulations) but which I haven’t seen described in literature. The reader is referred to subsection 3.2.4 for more information.
Chapter 2

Generative Adversarial Networks (GAN)

Since 2014 the Generative Adversarial Network [1] (GAN) framework has rapidly become one of the most popular tools for training generative neural network models.

Similar to an autoencoder, the generative part of a GAN (henceforth referred to as the generator $G$) consists of a neural network that takes a variable $z \in \mathbb{R}^d$ from some source distribution as input and transforms this variable into a sample from another distribution $p_g$. The framework is successful if this output distribution matches some target data distribution $p_{\text{data}}$.

Contrary to an autoencoder, a GAN does not necessarily have an encoder which may transform a data sample into the latent space. Instead a GAN has a discriminator $D$ which is trained to differentiate between samples from the generator $x_{\text{fake}} \sim p_g$ and training samples $x_{\text{real}}$ from our data-set.

The term adversarial comes from the fact that the discriminator $D$ is trained to minimize output values for input values from the generator (and conversely maximize output values for real samples) whereas the generator is trained to try to maximize the discriminators outputs to generators own outputs. Goodfellow [1] formulates the mini-max game the two networks play as in Equation 2.1. For any set of data there exists an optimal solution to this game where $G$ outputs exclusively samples found in the training data-set and $D$ outputs $\frac{1}{2}$ for any real samples or samples $G$ generates (which are identical) and 0 for any other input. In general however, the goal is not for the generator to be able to reproduce the entire data-set perfectly, instead, we would like for the generator to be able to create new samples which are near impossible to recognize as fake, even for a well trained discriminator.

$$\min_D \max_G \mathbb{E}_{x \sim p_{\text{data}}} \left[ \log D(x) \right] + \mathbb{E}_{z \sim p_z(z)} \left[ \log (1 - D(G(z))) \right] \quad (2.1)$$
Intuitively one can imagine the game being played during training by the GAN networks is similar to a cat and mouse game between an art forger and a journalist researching art forgeries. The art forger attempts to reproduce art and sell it on the black market. The journalist analyzes different pieces of art and writes about what he finds. At first the art forger (who never actually knew anything about art) just throws paint randomly at a canvas and hopes nobody will notice the difference. The journalist also does not know much about art. However, after spending some time looking at real pieces of art, he is eventually able to recognize that something is off with the random blobs of paint the forger is producing. The journalist then writes about why he thinks some of the paintings were fakes. The forger uses the journalist’s feedback to make his forgeries a bit more realistic. This goes back and forth as the forger keeps getting better at making forgeries and the journalist gets better at recognizing more minor details. If everything goes well, eventually the forger can reproduce the Mona Lisa up until the finest brush strokes.

**Mode Collapse** The previous example also makes it easy to understand a problem known as mode collapse. One can easily imagine the journalist having more trouble recognizing a painting as fake when the forger tries to make something resembling modern art, compared to when the forger tries to make something resembling more classical art. As a result the forger may try to get even better at modern art, at the cost of his skills at reproducing classical pieces. While the journalist may catch on and become better at recognizing fakes, the forger may completely give up on classical art in the mean time and only try to reproduce his specialty. In the worst case the forger (our generator) may end up becoming a specialist for only a single painting, and completely forget how to reproduce anything else. This problem is known as mode collapse.

### 2.1 Wasserstein GAN

The goal in generative modeling is to minimize the difference between the distribution of generated data $P_{G(x|\theta)}$ and the distribution of real data $P_r$. Asymptotically the Kullback-Leibler divergence [5] of the distributions is minimized in the optimum.

If our model for the distribution is very bad however, it is possible for the support of the two distributions to be very small or even 0. In this case the KL divergence tends towards infinity [6]. The standard approach to this problem is to add Gaussian noise to the data. For images this in turn degrades the quality of data and makes them blurry.

More recently a new approach to deal with this problem has arisen. With
help of the Wasserstein metric distributions may be compared even if they share no support at all. This metric is also known as the Earth Mover distance and is defined in Equation 2.2 for distributions $P_1$ and $P_2$ [6].

$$Wasserstein (P, Q) = \inf_{\gamma \in \Pi(P,Q)} E_{(x,y) \sim \gamma} [\|x - y\|] \quad (2.2)$$

The primary strength is of Wasserstein GANs is that they do not require carefully balancing training of the generator and discriminator. Instead the discriminator may be trained to optimality without the Wasserstein metric tending towards infinity or becoming undefined. Empirically Wasserstein GANs also suffer significantly less from mode collapse [6].

### 2.2 Relativistic GAN

Generators and discriminators are handled fundamentally differently in the original GAN formulation, as the discriminator explicitly tries to maximize the generators whole loss function, whereas the generator only cares about what the discriminator says about the data it generates. This formulation makes sense in that we are trying to get the discriminator to figure out what looks “real” and the generator to make things which match that description.

Instead Jolicoeur-Martinou [7] proposes to train the discriminator to differentiate between real and fake data while the generator is trained to explicitly minimize the difference between real and fake data.

### 2.3 Conditional GAN

The standard GAN formulation does not make any assumption about given data. For many problems however we would like to train our networks to be able to deal with additional class constraints. In this thesis the primary interest is in the specific case of temporal and continuous constraints, which will be described in chapter 5. Plenty of research has gone into conditional GANs, the reader is referred to section 5.1 where some of it is discussed.

To understand what is meant by this we could consider the MNIST [8] dataset containing a large number of handwritten digits. We could train a regular GAN to generate random digits [9], but perhaps we would like to generate large, specific numbers. In this case we could train a conditional GAN on MNIST where each class represents a specific digit. In order to generate a handwritten number it is enough to query the GANs generator with the correct digit classes in the right order.
Chapter 3

N-Body Simulations

The large scale distribution of matter in the universe is called the cosmic web [10]. It is a very complex structure which contains information which is of cosmological interest, such as the nature of dark matter and energy as well as gravity [11]. Many of the cosmological parameters are not known and cosmologists hope that they can be deduced by simulating particle distributions which may resemble the cosmic web.

The n-body problem refers to the problem in Physics and Cosmology of predicting the movement of \( n \) different celestial objects (bodies) such as planets or stars. The solution of the problem is dependent on a number of factors, such as each body’s mass, current velocity and coordinates within the system.

N-body simulations are solutions (approximate or exact) to n-body problems. The primary force that needs to be calculated in an n-body simulation is the effect of gravity. Our simulations only contain dark matter, so the only force in them is gravity.

Depending on the resource constraints and requirements for a simulation, n-body simulations are run with different discreet time steps. The computation begins with a structure which can be seen as a Gaussian random field and simulates the entire history of the universe up until the point of interest.

3.1 Cosmic Time: Redshift

In cosmology, time is measured in terms of what the redshift of light emitted is as seen in the present day. The further back in time an event is, the larger the redshift value. At the Big Bang the redshift tends towards infinite [12]. Light emitted in the present day has redshift 0.

There is a bijective relation between the time since the Big Bang and the positive redshift values. The precise time scale is dependent on the parameters
3. N-Body Simulations

3.1 Body Simulations

Figure 3.1: A Timeline of the Universe. Source: NASA / WMAP Science Team, retrieved September 1 from https://map.gsfc.nasa.gov/media/060915/index.html

of whichever cosmological model it is derived from. According to Carmeli et al. [12] under certain assumptions the relation is roughly given by Equation 3.1. Here $t$ denotes time since the Big Bang, $z$ the redshift and Gyr refers to one gigayear or $10^9$ years.

$$t \approx \frac{28}{1 + (1 + z)^2} \text{Gyr}$$

(3.1)

3.2 Quantitative Metrics for N-Body Systems

A large part of the reason why generative models are hard to train is that there is no cut and dry way to define how “good” an image is. If there was a clean metric for this, then this would be useful to design a loss function. Autoencoders circumvent this problem by defining a good output as being similar to the input and GANs circumvent this problem by letting a second neural network define whether an image is realistic.

Still, even if we don’t optimize for them directly, it makes sense to come up with statistics to describe the data. If the distribution of generated data $P_g$
matches the distribution of real data $p_{\text{data}}$ then any statistical metrics should be the same up to a margin of error as well.

### 3.2.1 Mass Statistics

The distribution of particle densities in $n$-body simulations is long tailed. As we will see in section 4.3, our data is no exception. Such a distribution can be hard to learn as towards the end of the distribution, just a few points can translate into an order of magnitude difference in relative terms. For this reason it makes sense to make histograms of the pixel intensities of the data.

### 3.2.2 Power Spectral Density

A popular metric for $n$-body simulations are power spectral density curves. The galaxy correlation function measures the amount of clustering occurring in either the spatial or angular distributions of galaxies. The power spectrum is the Fourier transform of the correlation function\cite{13}.

We calculate the power spectrum with the help of Equation 3.2. Here $\delta_D$ refers to the Dirac delta function\footnote{The Dirac delta function is a function which is 0 for any value $x \leq -\epsilon \vee x \geq \epsilon$ for any arbitrary $\epsilon > 0$, but is so large around 0 that the function’s integral is 1\cite{14} }, $\tilde{\kappa}$ refers to the Fourier transform of the deviation of the particle density from the mean at each Fourier bin $\ell$. The power spectrum is dependent only on the modulus of $\ell$ as a result of the statistical homogeneity and isotropy\footnote{uniformity in all directions} of our data\cite{15}.

$$\langle \tilde{\kappa} (\ell) \tilde{\kappa}^* (\ell') \rangle = (2\pi)^2 \delta_D (\ell - \ell') P_k (\ell)$$ \hspace{1cm} (3.2)

### 3.2.3 Peak Statistics

More recently a new metric was developed to analyze $n$-body simulations. Dietrich et al. \cite{16} suggest the shear peak statistic as an alternative to the power spectral density metric. Shear peak statistics are sensitive to non-Gaussian features and have been used to analyze large datasets of real, non-simulated data such as the Dark Energy Survey\cite{17}.

### 3.2.4 Mass Difference Histogram

The metrics described in the previous three subsections each describe metrics which can be used to quantitatively describe data from sets of independent images. In this project we are interested in creating a model which can not only generate independent images for any arbitrary redshift, but also generate sets of images which are consistent over time. In order to do this
we came up with a metric that can help quantitatively describe change over time.

There are two kinds of mass difference histograms, positive mass difference histograms and negative mass difference histograms. We assume a set of data samples \( x_1 \) and \( x_2 \) with \( \varrho(x_1) < \varrho(x_2) \) according to some function \( \varrho(x) \) used for ordering. In the case of our data \( \varrho(x_i) \) returns the time scalar of the image \( x_i \). We define positive and negative mass difference histograms by first applying the functions defined in Equation 3.3 or Equation 3.4 respectively and then calculating the mass histograms on the output. In the equations \( \text{ReLU}(x) := \max(0, x) \).

\[
\text{PosMass}(x_1, x_2) := \text{ReLU}(x_2 - x_1) \tag{3.3}
\]

\[
\text{NegMass}(x_1, x_2) := \text{ReLU}(x_1 - x_2) \tag{3.4}
\]

The main advantage of the mass difference histogram compared to simply plotting the difference between two data samples is that we can use log spaced histogram bins and visualize the change in a log-log plot.

### 3.2.5 Comparing Curves

The metrics of the previous three sections are all in the forms of curves. This is excellent as it can give us information about what is wrong with generated data. Unfortunately, it is also more complex to compare curves with one another than to compare scalars. In order to have an overview of how a model may be performing it thus makes sense to define good difference metrics of curves.

**Norms** The most popular way to compare two curves is undoubtedly to take the norm (typically \( \ell_1 \) or \( \ell_2 \)) of the vector of the curves differences. Probably the biggest problem with this approach is that if one part of the curve gets stuck it can dominate statistics very easily and mask when other parts of the curves begin to match better. To help deal with this issue we also kept track of the norm of the log of the difference vector.

**Wasserstein** The norm of the difference of two curves treats each point of measurement as independent. However, this is not normally the case. Instead the distance between curves can be defined by the Wasserstein metric, more descriptively known as the Earth mover distance. The Wasserstein metric is defined as the amount of mass which must be moved multiplied by how far it must be moved in order for two curves to match.
Chapter 4

The Data

Before getting into the details of the proposed architectural solution to the problem in chapter 5 we will take a look at the data format and data-set.

The raw data for this project is the output of an \( n \)-body simulation stored at ten different points in time. The simulation consists of a large number of points. For each point the \( x \), \( y \) and \( z \) coordinates in space as well as the velocity are stored for each point in time. Over the course of a simulation the positions of these points change based on formulas from physics. For more information the reader may refer to the chapter on \( n \)-body simulations.

The simulation was generated by the fast open source L-PICOLA [18] library. The resolution of our data-set is 500 MPc, which is equivalent to roughly 1.9 billion light-years.

4.1 Points to Images

Due to a number of reasons it is impractical to work with the locations of the bodies in our data directly. The raw data is hard to visualize and it is hard to train a model. Furthermore, it is improbable that such precise information is currently achievable with a generative neural network.

Since for the purposes of this project it suffices to know the rough distribution of points in space, our first preprocessing step is to add the points of each time step to a 3d histogram. From this histogram each bin of each axis may be interpreted as an individual image with each combination of bins of the remaining axis representing a pixel of an image.

In this project we relied on 512 \( \times \) 512 \( \times \) 512 histograms. Since we were working with 64 \( \times \) 64 images for the most part each image could be split up into 8 \( \times \) 8 individual images. Splitting an image at least once gives us the additional benefit that we can remove an artifact from the simulation data.
4. The Data

Specifically since the universe is infinite, but our data is not, the L-PICOLA simulations wrap around the edges. This leads to structures which reach the end of one side of a simulation box and come back out on the other. If we work with only parts of the original 512 x 512 images these artifacts no longer exist, resulting in more realistic images.

4.2 Transformation

Feedforward neural networks are known to work best when data is within a certain range, so input is typically normalized. Values in the range of [0, 1] or [−1, 1] have typically been preferred as these match the output ranges of the sigmoid and tanh activation functions respectively.

As discussed in section 4.3, the dynamic range of the pixel intensities of our data is very large. The data distribution is also very uneven, with the occasional exceptionally large value.

Previously in Rodriguez et al. [19] the transformation in Equation 4.1 and Figure 4.1 was used to normalize the data into the range [−1, 1]. During this project we found that the models were having trouble dealing with the high frequencies of the data. Since this transform saturates at 1, most of the outliers tend to be very close to this value. While outliers tend to be fairly rare, they can have a large effect on the power density spectrum\(^1\) of the data.

For single time steps, tuning \(k\) may be a viable solution. Unfortunately, a large portion of the pixel intensities in lower redshift images are outliers relative to the pixel intensities in higher redshift images.

\[
s_{Rod}(x) = \frac{2x}{(x+k)} - 1 \tag{4.1}
\]

To help alleviate this issue we came up with another transform which is given in Equation 4.2 and visualized with differing parameters in ???. The effect of different parameters can be found in the appendix in section A.1. This transform does not saturate at 1, instead it turns into a linear function above a certain, large threshold.

\[
f_{Per}(x,s,c) := g(x+s,c) - g(s,c) \tag{4.2}
\]

\[
g(x,c) := \begin{cases} 
3 \log(x+1) \over \log(c+1) & \\
3 + \frac{3x}{(c-1) \log(c+1)} & 
\end{cases} \tag{4.3}
\]

\(^1\)See subsection 3.2.2
4.3. Distribution

The distribution of our dataset at several different redshifts can be seen in Figure 4.4.

On closer inspection it is clear that the dynamic range of the data gets smaller over time and is the largest at the current point in time, redshift 0. The number of pixels with very few or no points increases dramatically as do the number of pixels with very many points. Note that the figure is in log log scale. Between redshift $z = 4$ and $z = 1.5$ the number of pixels with between 0 and 2 particles increases by a factor of almost $10^3$.

**Initial Mass Function** The initial mass function is the empirical distribution which describes the relative number of stars created per unit mass interval. Historically, this has been described by a sequence of power law distributions [20]. More recent work, such as that from Chabrier [21], describes the
4. The Data

Figure 4.3: The effect of rescaling input before transforming data.

![Effect of Rescaling Input Data on t](image)

Figure 4.4: Distribution of pixel intensities before transform for different redshifts. The histogram was created with the complete, unscaled, 500MPc data set with bins of size 2.

![Histogram of Pixel Intensities for Different Redshifts](image)

initial mass function of individual or groups of stars as broken power low distributions. This means the distributions become power law distributions above a certain mass threshold, but not below.

Our data resembles broken power law distributions as well. When a power law distribution is plotted in log-log scale, one can draw a vector perfectly through the data. In the case of a broken power law distribution this is true up to a point, where the distribution changes. Figure 4.5 shows different plots of the raw data at different redshifts, with attempts to plot such
4.3. Distribution

Broken Power Law Distributions Visualized

Figure 4.5: Visualization of raw pixel distributions at different redshifts. If the true distributions were to match the orange lines then the distributions would be power law distributions. Around the pixel intensity marked by the green line the (possible) power law distribution breaks.

aforementioned vectors. It should be noted that such a vector existing is a necessary, but not sufficient factor for a distribution to be a power law distribution [22].

---

See page 675 in the referenced source
Chapter 5

Continuous Conditional Generative Adversarial Networks

A model that can generate images of an $n$-body simulation at different points in time needs to be able to fulfill two different attributes.

1. The model must be able to generate images from an arbitrary class defined by a point in time.

2. The model must be able to generate images with consistency over time.

With respect to the first point, since time is continuous we would like our model to be able to generate images for points in time it has never seen before, which it must interpret based on its understanding of data it has seen before.

The latter point makes our problem fundamentally different from what a regular conditional GAN could solve.

5.1 Conditional GANs and Prior Work

While this is the first body of work focusing on GANs, $n$-body simulations and continuous time to the authors knowledge, there is a lot of prior work with conditional GANs.

**Conditional Generative Adversarial Nets** Mirza and Osindero [9] proposed simply adding a class variable $y$ to the latent variable $z$. In this case the discriminator needs to also receive $y$.

**InfoGAN** Chen et al. [23] managed to come up with an unsupervised form of conditional GAN. It was possible for the generator to learn many different features of the MNIST dataset which could then be varied during generation.
5. **Continuous Conditional Generative Adversarial Networks**

This included continuous features such as the angle of the digits. Since it is an unsupervised form of learning, their approach is not suited to learn a single specific variable to condition on.

**MoCoGAN** Tulyakov et al. [24] split the latent vector into two parts. One part was designed to represent static elements of a scene while the remaining portion of the latent vector was designed to represent the motion elements. Video sequences were generated with help of a recurrent neural network which determined the values of subsequent motion subvectors.

**TGAN** Saito et al. [25] rely on a dual generator setup in order to generate videos of fixed length. The first generator learns to output latent vectors for the second generator.

**Generating Videos with Scene Dynamics** Vondrick et al. [26] relied on spatio-temporal convolutions to generate short video sequences up to a second in length. They did so by splitting up images into background and foreground and relying on a fixed camera, so the background would remain stationary.

**StarGAN** Choi et al. [27] came up with a way to train a conditional GAN such that a data sample could be mapped from an arbitrary image domain to another. In order to do this a one-hot encoded binary vector is given to the generator along with the latent vector.

**GANimation** Pumarola et al. [28] build on StarGAN and came up with a way to do image to image facial expression translation on a continuous space. They did this by developing a concept called action units which describe smaller changes in facial expressions.

### 5.2 Proposed Architecture

#### 5.2.1 Generator

Our generator network needs to be able to generate images for any specific point in time given a latent vector and the specified time scalar. While our problem deals with time, our solution could deal with any continuous conditional variable.

**RNNs and Their Issues** A natural idea is to rely on a recurrent neural network architecture for the generator, which has been done in several papers
combining GANs and videos. There are two problems with a recurrent network. The first is that we would either have to condition on an input image for the initial time step, or we would have to have a separate network which could generate an image for the initial time step. The second, potentially larger issue with using a recurrent neural network for the generator is that there is a trade off between the computational overhead and the continuity of the images the network can generate.

Let us assume we have \( n_t \) time steps in our data and would like to be able to generate \( n_{pt} \) images at partial time steps between time steps in our data. Given an image to condition on at \( t_0 \) we would then have to call the generator \((n_t - 1) \times (n_{pt} + 1)\) times in a sequential fashion in order to generate an image for time \( t_{max} \). This fairly large computational overhead is necessary during training. Having only a small number of partial time steps between trained time steps would significantly reduce this burden, however the primary goal of this thesis is to come up with an architecture that can learn a continuous conditioning parameter.

One of the benefits of an RNN based architecture is that it would be easy to compare real output images with outputs of the RNN given a source image. It is also not out of the question that image quality generated from an RNN may be better than the approach generated by the architecture in this thesis. However, for the aforementioned reasons a different architecture was chosen for this project.

**Proposed Architecture** The architecture proposed for the GAN’s generator here is to use a straightforward fully deconvolutional architecture which takes as input a latent variable which is preprocessed in some way to encode time. There are numerous ways to encode time in the latent variable. For more on how this encoding is done I refer the reader to section 5.3.

The goal is that the generator is fed the same base latent variable multiple times, simply encoded with different points in time, then the images should all be generated from the same simulation. As long as our encoding scheme can encode continuous variables, our network will hopefully be able to generate frames of a simulation which are arbitrarily close together. It is clear that if such a generator can successfully be trained then we have a true continuous conditional generator.

**Training Phase for the Proposed Architecture** During training, given data with \( n_t \) time steps and a discriminator batch size of \( bs_d \) we feed our generator \( bs_g \) different latent variables \( z_{base} \in \mathbb{R}^L \) where \( L \) is the dimension of the latent space. Each \( z_{base} \) is then preprocessed with each of the \( n_t \) time variables.

---

\(^1\)See the previous section
independently to create new latent variables \(z_i\). Each \(z_i\) is then fed as a completely separate input to the generator. This means the generator has a larger batch size \((bs_g = n_t \times bs_d)\) than the discriminator for \(n_t > 1\). From these inputs, the generator then creates \(bs_g\) separate images.

### 5.2.2 Discriminator

Our architectures discriminator needs to be able to differentiate between two key issues. The first issue is that the discriminator needs to be able to recognize that an image belonging to a certain time step \(t\) lacks or contains features which it should not. This is the standard problem of a conditional GAN and means that our discriminator architecture must be fully aware of which image should belong to which (continuous) class. The second problem the discriminator is required to deal with is temporal consistency. What this means is that given two data samples from the same simulation (real or fake) the discriminator should be able to recognize if changes occur between them which are larger than reasonably plausible.

As an example consider a dataset which consists of balloons filling with air until they pop. If an image of the class of the final time step has a balloon which is half filled with air and not popped, then the image is not consistent with its temporal class. If on the other hand the image is that of a popped balloon, but one which has a different color from the previous images of same sequence, then the image is not temporally consistent.

There are different options that are reasonable here. In early stages of the project a two-discriminator approach was preferred, with one discriminator used to check continual consistency and one discriminator used to check class consistency. While similar approaches have been done in related works [24, 26], it was abandoned early in the project in favor of a more unified approach. The reasoning for the discriminator’s architecture is described in the following paragraphs.

**Reasoning Behind a Unified Discriminator** In the following two paragraphs we assume a two-discriminator architecture with one specialized on temporal consistency and one on class consistency.

Consider a dataset that consists of various spheres which each individually exclusively expand or exclusively contract over time. If our temporal discriminator only sees two time steps at a time, then it is possible for the generator to create a sample where a sphere oscillates between expanding and contracting, without the discriminator having the capacity to recognize this. In theory such an inconsistency could occur over any time frame, so it makes sense to train the temporal discriminator with as many time steps as are available.
If the temporal discriminator is fed with all the available time steps and is aware of their order, then the temporal discriminator is also aware of precisely which data sample refers to which time step. This also means our temporal discriminator can recognize class inconsistencies. Intuitively, it makes sense that class and continuity inconsistencies may belong to similar high level features of the data, so it makes sense for the architectures to be part of the same network, similar to the policy and evaluation networks in DeepMind’s AlphaGo [29].

During training the generator generates images as in the previous subsection. Images from the same base latent variable (differing only in the time encoding) are concatenated as different channels of a single image. These multi-channel images are then used to train the two networks via backpropagation as in a regular GAN.

5.2.3 Specifics for the $n$-Body Problem

As described in section 4.3, a particular challenge when dealing with densities from $n$-body simulations (especially over time) is the dynamic range of the data. In order to deal with this, Nathanaël Perraud came up with an excellent block that helps the networks learn about the particle distributions.

It is worth mentioning that a more theoretically founded learnable histogram has been developed [30]. Unfortunately, I was not able to stabilize a network with the cited network block in tests. Whether or not this is simply a matter of finding better hyper-parameters or better initialization of parameters is not out of the question and deserves further research, which was not produced here due to time constraints.

5.2.4 Optional Features

Over the course of this project we tried many different things to try to get good and stable results. Two are mentioned here, which are not in general mandatory for a continuous conditional GAN. However, they did prove to be interesting for our specific problem.

Spectral Normalization A recently discovered regularization technique is that of spectral normalization [31]. Spectral normalization differs from other techniques such as dropout or batch normalization as it both is independent of data and does not effect the network architecture in any way. It is a regularization technique which only effects the weight matrix of the layers it is used. We tested spectral normalization in the discriminator.

Self Normalizing Neural Networks A couple years ago the exponential linear unit (ELU) activation function [32] was developed as an alternative to
5. Continuous Conditional Generative Adversarial Networks

the popular (leaky) rectified linear units. The scaled exponential linear unit (SELU) activation function [33] in turn is an ELU activation with a specific set of hyper-parameters which helps avoid the vanishing gradient and exploding gradient problems respectively. We tested SELU and RELU activations in the generator.

5.3 Encoding a Continuous Variable in the Latent Space

Over the course of this thesis three different approaches to encode a continuous variable in the latent space were developed and tested.

5.3.1 Direct Encoding

The most direct way of encoding such a variable into the latent space is to simply and explicitly add it to the latent variable $z$. If the first layer of the generator is a fully connected layer then a single input for time suffices. Should the first layer instead be a convolutional layer, then the time variable needs to first be broadcast to the size of a full channel. This channel can then be concatenated to the reshaped $z$.

5.3.2 Full Vector Scaling

Another way to encode a continuous variable is to interpret a point in the latent space as a vector with an angle $\alpha \in \mathbb{R}^{(d-1)}$ and a magnitude $m \in \mathbb{R}$. In this case we interpret $m$ our continuous time variable.

5.3.3 Partial Vector Scaling

Inspired by recent work [34, 24] we also tested splitting the latent vector into two parts. The idea being that our data has components which are fairly invariant to time (eg: general positions and orientations of halos) and other components which are more variant. Splitting the vector apart could allow part of the vector be used to encode the invariant components and part of the vector to be used for variant components.

This variant of the previous idea is arguably more interesting than the previous one, since it naturally generalizes to multiple continuous variables.

5.3.4 Multiple Continuous Variables

Continuous variables can be found all over the place. If possible one would like to vary multiple different continuous variables at once. In $n$-body simulations it may be interesting to vary both the cosmological model as well as time for example.
Generalizing our proposed solution to multiple continuous variables could be done in several ways. In the following we will assume we have \( k \) different continuous variables that we would like to condition a continuous conditional GAN on. For each \( k_i \) our data contains \( d_i \) different values. Our data contains samples for every combination of continuous conditional values.

The most direct of the generalizations is to have a single discriminator \( D \) which we feed samples with \( \prod d_i \) channels. Samples can stem from real data or from outputs of the generator. The advantage of this approach is that there is only a single discriminator which must be trained and it should theoretically be able to recognize any continuous discontinuity which may arise as a result of non-linear combinations of continuous variables. Unfortunately the term \( \prod d_i \) may quickly become to big very quickly in practical terms when dealing with large \( k \).

### 5.3.5 The Nonlinearity of Time

An issue with any encoding of time is that it may not be clear if a scalar value makes sense for it. While time moves at a constant rate [?] for us in the physical world, our model is not aware of this and only sees time in relation to changes in data. Changes in data must not necessarily be constant.

As an example imagine a dataset consisting of video sequences of fireworks. Most of the frames consist of rockets flying up into the sky, with each succeeding frame changing in the minor way of the rocket being slightly higher. This continues until there is a huge change in one of the last frames as the rocket explodes.

While it makes sense to trust the network to learn the transformation based on some fixed time scalars it may make sense to learn the time scalar via a separate 1x1 convolution first. Due to time constraints this idea was not tested in depth in this thesis.
6.1 Experiment Setups

We would like to take a look at the effect of different parameters from both a quantitative as well as a qualitative point of view. Over the course of the thesis models with many different combinations of time steps, data sizes and image resolutions were trained and tested.

Due to time and space constraints we will focus on $64 \times 64$ pixel images with side lengths around $62Mpc$. The main time steps we will focus on are redshifts $z \in [0.000, 0.250, 0.666, 1.500, 4.000]$ for the single time step models and for the continuous conditional GAN models we will train on $z \in [0.000, 0.250, 0.666, 1.500]$. For the more complicated models we will also take a look at images produced at redshifts $z \in [0.111, 0.428, 1.000]$. While the models were not trained on these time steps we have data available for them and would like to compare what the model infers for these time steps with the ground truth.

Single time step models were trained for 24 hours though images usually were close to their best after around half that from both a quantitative as well as qualitative point of view. Continuous conditional GAN models were for the most part also trained for around 24 hours, but some were trained for around 72 hours. In order to do this the models had to be reloaded and training had to be continued from the point of the last save. Since statistics were done more often than the models were saved, this can be seen in statistics showing the training over time. Due to the nature of stochastic gradient descent it is normal that the statistic models differ slightly when training is continued from a previous point in time.
6.2 Specialized Single Redshift Models

6.2.1 Spectral Normalization

A large part of the difficulty of working with generative adversarial models is dealing with instability. Models can improve steadily before completely falling apart without any clear reason as to why. One of the ways to help deal with instability which proved effective at some point during the project was the use of Spectral Normalization [31].

A red flag pointing to issues with stability is when the gradients of the generator or discriminator start to grow late into training. Figure 6.1 visualizes the effect spectral normalization had on the training of two different single time step models. The two models on the right hand side are especially interesting as the orange model (with spectral normalization) briefly destabilized after around 100'000 batches but was able to completely re-stabilize. I did not see this happen in any of the models without spectral normalization (a) Blue is without spectral normalization while grey is with. \( z = 1.500 \) (b) Pink is without spectral normalization while orange is with. \( z = 4.000 \)

Figure 6.1: Quantitative comparison of gradient norms in discriminators of models with and without spectral normalization over the course of training. The \( x \) axis is represents the number of training iterations and the \( y \) axis is proportional to the gradient norm.

Intuitively, one would expect such model instability to be clearly visible in the different statistical curves. A quick look at the \( l2 \) norm of the log transforms of the curves over the duration of training shows a different picture however. Figure 6.2 shows a comparison of the statistics of the models on the left of Figure 6.1. Over the course of training it is hard to say that either model has better quantitative statistics, though the curves of the model with spectral norm do seem more stable. Spectral norm did help the quantitative statistics indirectly as it allowed the model to become stable enough to use the SELU activation function which is described in subsection 6.2.2.

A challenge when working with GANs and \( n \)-body simulations is that very
6.2. Specialized Single Redshift Models

Figure 6.2: Quantitative comparison of different statistics over the course of training with and without spectral normalization. The x axis is represents the number of training iterations and the y axis is the $l_2$ norm of the log curves of each respective statistic.

often the quantitative statistics and qualitative statistics do not necessarily match up. For this reason it always makes sense to take a look at both.

Figure 6.3 shows generated images of a model trained with and without spectral normalization as well as real images for redshift $z = 1.500$. The images from the model without spectral normalization have small isolated bits of mass not found in the real images. Aside from that they are fairly comparable to the images generated from the model with spectral normalization. Upon closer inspection filaments in the lower frequencies of the real images are visible, which are not visible in the generated images of either model.

6.2.2 Scaled Exponential Linear Units

As briefly mentioned in the previous section, thanks to spectral normalization the model became stable enough to train with scaled exponential linear units [33]. Before trying spectral normalization, SELU activations were tried but models tended to destabilize heavily and produce nothing but gibberish. SELU was much more sensitive to weight initialization as well. When a model with the newer activation function did remain stable it consistently resulted in the best images from a quantitative point of view.

We begin our look at the quantitative results of scaled exponential linear units by adding a model with the activation function to the models in Figure 6.2. It is immediately visible in Figure 6.4 that the SELU activation function has lead to significantly improved results. For further comparison we have also added the log $l_2$ over the course of training for models trained with redshift $z = 0.000$. Time steps not shown tell a similar story and can be found in the appendix in section A.2.

We would like to get a more precise impression of what this looks like for the different statistics. Where were our models failing that they are now succeeding at? We will take a look at each of our three primary quantitative
6. Results

(a) Model with spectral normalization

(b) Model without spectral normalization

(c) Real images from the simulation

Figure 6.3: Qualitative comparison of images from models with and without spectral normalization as well as real 62 Mpc images for redshift $z = 1.500$.

statistics individually, but taking a look at models trained from different time steps simultaneously. The only exception we will make is for redshift $z = 4.000$. The reason for this will become clear when we discuss that time step.

In Figure 6.5 the mass histograms of images from three different sources at different redshifts are plotted. The red curve shows the mass histogram of real simulated data at the respective redshift. The blue curves are from models trained without spectral normalization and with LRELU activations in the generator. The green curves (barely visible in the upper left behind the red curve) are from models trained with spectral normalization as well as with SELU activations in the generators.

It is impressive just how perfect the first green curve is, however it must be noted that there is a random factor as to what point in time the training was stopped. Higher redshift values did tend to be more challenging for most of the models that were trained.
6.2. Specialized Single Redshift Models

(a) SELU model in pink. Redshift $z = 1.500$

(b) SELU model in blue. Redshift $z = 0.000$

Figure 6.4: Quantitative comparison of different statistics over the course of training with and without scale exponential linear units. The $x$ axis represents the number of training iterations and the $y$ axis is the $l_2$ norm of the log curves of each respective statistic.

The models trained with the LRELU activation function had significantly more problems with creating too much mass in the high frequencies. This is something we will see more of as a challenge for the continuous conditional GAN. The plot is a log-log plot, so the huge frequencies on the right hand side of the plots are extremely infrequent. This is a tough problem to deal with and has a large effect on the power spectral density curves.

While subtle, a hint of problems to come is also visible at very low frequencies at higher redshifts. The higher the redshift becomes the greater this gap becomes. The first upwards part of the curves goes from an increase in factor 10 at redshift 0 to an increase of more than factor $10^2$ by redshift 4. Here the models trained with scaled linear units still deal with the problem perfectly, however the problem becomes exponentially more difficult with even higher redshift values and when dealing with multiple time steps.

The peak histograms of images from three different sources at different redshifts are visualized in Figure 6.6. The red curve shows the mass histogram of real simulated data at the respective redshift. The blue curves are from models trained without spectral normalization and with LRELU activations in the generator. The green curves (barely visible in the upper left behind the red curve) are from models trained with spectral normalization as well as with SELU activations in the generators.

The peak histograms are very similar to the mass histograms of the higher frequencies. This makes sense, since the very low frequencies are less fre-
6. Results

Figure 6.5: Mass histogram curves for different redshift values. The three curves are from models trained with scaled linear units in the generator (green), rectified linear units (blue) or from real simulated data (red).

quent than frequencies which are higher they rarely lead to forming local peaks. The first part of the peak histograms for redshift \(z = 1.500\) looks very smooth across all three curves. The models with rectified linear units seem to struggle a bit more in the higher frequencies compared to in the mass histogram, however it is within expected margins.

In Figure 6.7 the power spectral density curves of images from three different sources at different redshifts are visualized. The red curve shows the mass histogram of real simulated data at the respective redshift. The blue curves are from models trained without spectral normalization and with LRELU activations in the generator. The green curves (barely visible in the upper left behind the red curve) are from models trained with spectral normalization as well as with SELU activations in the generators.

The shift of the spectral density curves is heavily influenced by the highest frequencies of the generated images. When looking at these curves over time they had a tendency to jump up and down as the high frequencies got overproduced more or less relative to how frequent they are in the real images from simulations.

Before taking a look at the quantitative results for models trained on just redshift 4.000 we will take a look at the qualitative differences in images from one of the redshifts from the already discussed plots. Figure 6.8 fea-
6.2. Specialized Single Redshift Models

Figure 6.6: Peak histogram curves for different redshift values. The three curves are from models trained with scaled linear units in the generator (green), rectified linear units (blue) or from real simulated data (red).

The isolated points that were visible in the previous qualitative figure are not to be found in the new model with spectral norm and scaled linear units. Furthermore some subtle filaments in the low frequencies can be found in the images, such as on the bottom left of the third image of (b). From a qualitative perspective the images seem to be a bit better, though the differences are minor.

Finally we will take a look at the quantitative statistics for redshift 4.000 in Figure 6.9. The power spectral density curves look very good for this time step look excellent. Indeed taking a look at the same curve for the model at different points in training the model fairly consistently does well with this statistic for redshift 4. There is a minor but consistent difference between the models dependent on the activation function, but it is less than for the smaller redshifts. The smaller dynamic range of the the higher redshift data presumably makes it easier for the model to get this right.

The peak histogram looks very good and this too seems consistent over the course of training. The difficulties of dealing with the higher frequencies
6. Results

Figure 6.7: Power spectral density curves for different redshift values. The three curves are from models trained with scaled linear units in the generator (green), rectified linear units (blue) or from real simulated data (red).

persists from smaller redshifts, however the difference is significantly less pronounced.

The mass histogram is certainly the most interesting. As with the peak histogram, the higher frequencies are noticeably easier for the network to learn compared to at lower redshifts. On the other hand the lower intensities are very hard for the network to learn and contrary to the previous images results where both models with LRELU and SELU were fairly close to perfect there is suddenly a huge gap to the real data. Models trained with leaky rectified linear units again struggle more than the counterparts trained with scale exponential linear units, however both are very bad at low intensities.

Since how frequent the low intensity pixels are differs so greatly between the low redshifts and the high redshifts this ends up being even tougher for the continuous conditional GAN which we will take a look at in a moment.
6.2. Specialized Single Redshift Models

(a) Model with LRELU activation

(b) Model with SELU activation

(c) Real images from the simulation

Figure 6.8: Qualitative comparison of images from models with (a) leaky RELU and (b) SELU activations in the generators respectively. For reference (c) real 62 Mpc images for redshift $z = 1.500$. The first and third rows are the same as in Figure 6.3.
6. Results

Figure 6.9: Power spectral density curves for different redshift values. The three curves are from models trained with scaled linear units in the generator (green), rectified linear units (blue) or from real simulated data (red).
6.3 Continuous Conditional GAN

We would now like to take a look at the main contribution of this thesis, which is the continuous conditional GAN architecture. We tested three different ways of conditioning and encoding a continuous variable into the latent vector, as discussed in section 5.3.

6.3.1 Quantitative Statistics Compared to Specialized Models

The next research question is whether or not the continuous conditional GAN model can compete with specialized models trained on just one redshift, such as the models we took a look at in the previous section.

In order to answer this question, we will compare the best single time step models with one of the best continuous conditional GAN models from the last month of the thesis. The continuous conditional GAN is larger than the specialized models (9 layers vs 6 for the generator and discriminators respectively) so it is entirely possible the continuous conditional GAN may in fact outperform the specialized models, however it is facing a significantly more difficult task which is a superset of the problems that the specialized models had to solve.

We will begin our analysis of the quantitative statistics for continuous conditional generative adversarial networks along the same lines as in the previous section. We will look at the mass histogram, peak histogram and power spectral density curves a continuous conditional GAN, curves from separate specialized models and curves from real data. We will compare the quantitative statistics qualitatively.

There are a few things to take note on in the mass histograms of Figure 6.10. In the high intensities the continuous conditional GAN actually does better (!) at lower redshifts than the specialized model does. This could be explained by the fact that at higher redshifts the data has less high intensity pixels, so it becomes easier for the model to produce less mass for lower redshifts as well. At higher redshift values the model overproduces high intensity pixels at least as much as the specialized models.

We see a similar effect at low intensities. Looking closely the curve for the continuous conditional GAN at redshift $z = 0.000$ is actually slightly below the other two curves. This imprecision again makes sense, since the data the model has to learn to produce at higher redshifts have fewer pixels at lower intensities. By redshift $z = 0.666$ the trend has already inverted and by redshift $z = 1.500$ the continuous conditional gan is clearly overproducing.

---

1See chapter 5 for more information
2These are the same models as in the previous subsection with both spectral normalization and scaled exponential linear units
low intensity pixels. During most of the project this gap was considerably larger and was only reduced to what we see here with considerable amounts of optimization.

For the medium intensity values both the specialized models and the CC-GAN perform very well, with the only exception being the specialized model for redshift $z = 0.666$ which seems to produce too few pixels at that intensity. Looking at the curves over time it seems that this specific model was an exception in that respect and at other points in training this underperformance was not visible.

The next set of curves we would like to take a look at are the peak histogram curves.

From a qualitative perspective the peak histogram curves look similar to the mass histogram curves at high pixel intensities. Noteworthy is that the continuous conditional GAN struggles so much with the low intensities that the low intensity peaks for are too frequent for redshift $z = 1.500$.

The curves for the continuous conditional GAN and the specialized model for redshift $z = 0.000$ are almost identical. Both have minor issues at the highest pixel intensities, but aside from that the curves are both perfect.

The power spectral density curves in Figure 6.12 are more split between different redshifts than the previous metrics.
6.3. Continuous Conditional GAN

Figure 6.11: Peak histogram curves of a continuous conditional GAN model compared to the curves of specialized models and real data.

Figure 6.12: Power spectral density curves of a continuous conditional GAN model compared to the curves of specialized models and real data.
6. Results

The continuous conditional GAN has very similar curves over time, mostly differing only in a shift along the y-axis. The model’s curve is most accurate for redshift \( z = 1.500 \) as the curves of the simulated data flatten the earlier in the simulation the data stems from.

Qualitatively the power spectral density curve for the specialized models seem to outperform the curves at redshifts \( z = 0.000 \) and \( z = 0.250 \), while at least matching the conditional GAN at redshift \( z = 1.500 \). The exception is the specialized model for redshift \( z = 0.666 \), which is clearly shifted downwards. Presumably the weights were simply temporarily in a suboptimal point at the end of training.

Overall the larger model of the continuous conditional GAN seems to suffice to allow it to learn to generate conditional data which is on par with the data generated by specialized models from a quantitative point of view.

6.3.2 Quantitative Comparison of Continuous Variable Encodings

The next research question we would like to answer is which way of encoding a continuous variable is optimal for the model from a quantitative point of view.

We will compare the curves of the quantitative metrics qualitatively for different redshifts and three different ways of encoding continuous variables in the latent space, as well as the curves from the real simulated data.

The only difference for the models compared in this subsection is the way that time was encoded into the latent space. The models’ generators are comprised of exclusively deconvolutional layers. The models rely on scaled exponential linear units in the generators and leaky rectified linear units in the discriminator.

The model represented by the blue curve used the encoding scheme described in subsection 5.3.1. The latent variable was encoded by setting the last channel to the time variable to the size of the time variable.

The model represented by the yellow curves was encoded as in subsection 5.3.2. The components of the latent vector were scaled so the \( l_2 \) norm of the vector was proportional with time.

The model represented by the green curves used the approach described in subsection 5.3.3. The latent vector was split into two parts. The hope was that the model might be able to represent elements which remain fairly static over time, such as the positions of larger structures in the data, in the first unmodified portion of the latent vector. The second portion of the vector would then be rescaled similarly to the yellow model. The two parts of the vector are selected in such a way that they are evenly distributed in the reshaped input of the generator.
6.3. Continuous Conditional GAN

Figure 6.13: Mass histogram curves of three different continuous conditional GAN models as well as of real data.

The first set of curves for this experiment are visible in Figure 6.13.

The curves for each of the three continuous conditional GAN models match the curve of the real data near perfectly at medium pixel intensities. At low pixel intensities the three CCGAN models don’t match the distribution of the real data perfectly, however it is hard to say which model is best as the models seem to match each other very closely, even in their deviation from the real data.

At high pixel intensities things are a bit more interesting. The model scaling the full latent vector proportional to time struggles with the high pixel intensities at low redshifts, producing clearly the most mass.

The model scaling only half of the latent vector (in green) does better at low redshifts, but performs worst at redshift $z = 1.500$. Overall it seems to be outperforming the yellow model slightly at this metric.

The model which encodes time directly as a channel has a curve which is on par with the best result at each redshift, so it is the best model in terms of the curves of the mass histograms.

The next set of curves we would like to take a look at are the peak histogram curves.

The peak histograms tell a similar story to the mass histograms. The differ-
6. Results

Figure 6.14: Peak histogram curves of three different continuous conditional GAN models as well as of real data.

ence to the real data is perhaps a bit more pronounced than it was in that case, however the relative order of the models seems to be the same. The difference between the model scaling half the latent vector and the model encoding time directly is smaller however as the results at redshift $z = 1.500$ are much closer.

Next we take a look at the peak histograms in Figure 6.15. It is harder to decide which model performs best over the course of all the redshifts.

The curve which most closely matches the real curve in terms of its form at low redshift values is probably the yellow one. While the other two curves end up having a slight upwards trend at the end the yellow one gets continually steeper towards the end of the curve. In terms of distance between the curves it overall seems to be the winner at low redshifts, with the blue model and green models being quite similar.

There is a striking similarity in the form of the green and blue curves at low redshifts. When seeing such a similarity one would expect them to be from the models which scale the full and half of the latent variables respectively. The matching curves however come from the model encoding time in half the latent vector and the model encoding time directly by value as a channel. What these models have in common is that they have static elements over time in the latent variable, so they may be encoding similar things in there which the model encoding time in the full vector cannot.
6.3. Continuous Conditional GAN

At the two higher redshift values the green model edges out the competition by a slim but clear margin.

The last (two) metrics we will take a look at are specific to the temporal data we are working with. We will take a look at the positive and negative mass difference histograms. The former of which can be found in Figure 6.16 and the latter in Figure 6.17. As a quick reminder, the positive mass difference histogram shows positive changes of pixel intensity over time and the negative mass difference histogram shows negative changes over time.

A quick look over the positive mass difference histograms reveals the images from the generated models change similarly to how the real simulated data changes in the low pixel intensity differences. Unfortunately some pixels in the model are becoming brighter faster than in the real image series.

It would be interesting to know where these differences are coming from in terms of pixel intensity. Are the high intensity pixels becoming more intense or are the low intensity pixels becoming high intensity ones?

It is tempting to try to answer this last question with the data we have. Since the highest positive pixel intensity changes look like they are close to the maximum intensity of pixels in Figure 6.13 it seems logical to assume that it must be low intensity pixels becoming high intensity pixels. Unfortunately our plots are in log-log scale, so if a pixel becomes twice as intense, it

Figure 6.15: Power spectral density curves of three different continuous conditional GAN models as well as of real data.
6. Results

Figure 6.16: Positive mass difference histogram curves of three different continuous conditional GAN models as well as of real data.

would only shift very little within our histogram. This means that it would also be possible for a medium or even high intensity pixel to increased too much in intensity from one image to another. As such we cannot say where the models exactly are going wrong, only that the change in positive pixel intensity is sometimes too great.

The negative mass histogram curves in Figure 6.17 at first sight seem to tell a similar story. Again the pixel intensity changes from the models are similar to the real data, with the exception that larger changes sometimes occur than in the real simulated data.

This time however we can deduce to a certain degree which pixels sometimes lose too much intensity directly from our plots. Similar to before, the highest negative pixel mass differences are in a similar order of magnitude as the highest pixel intensities in general. Since we know however that pixel intensities cannot become negative, we also know that the pixels cannot have a higher negative pixel intensity difference than their own mass. For this reason it must be high intensity pixels generated by the models which sometimes lose too much mass.

Notably large drops in intensity sometimes also occur in the real images. It seems a bit unusual that large structures of mass seem to disappear from one point in time to another, especially if gravity is the only force in our simulations. There are two different way in which this can occur however.
6.3. Continuous Conditional GAN

![Negative mass difference histogram curves](image)

Figure 6.17: Negative mass difference histogram curves of three different continuous conditional GAN models as well as of real data.

The first option has to do with the fact that while our simulations are three dimensional, our images are not. Because of this, particles can move out of our images into the next image. If large enough structures do this, than such large drops in intensity can occur.

The alternative is similar. As it turns out particles do not need to leave the image for such changes to occur. If a concentrated set of particles moves to the side, then the pixel where the particles were previously located will also drop heavily in intensity. This second option could be dealt with by performing some kind of smoothing operation before calculating the mass difference histograms.
6.3.3 Qualitative Analysis

The last bit of results we would like to take a look at are the qualitative change of images over time. Continuous consistency can be measured to a certain degree quantitatively with the mass difference histograms that were just looked at, however what happens between time steps is harder to analyze, as changes could be happening at any nonlinear rate between time steps.

The main question we are interested in here is whether the change we see over time is gradual, or whether the models overfit on the specific time steps they were trained on? It is possible that change between two discrete points behave like a step function; after a certain threshold is reached the image spontaneously changes into an image of the next time step. In the worst case our model might not be generating anything resembling the real data at all in between redshifts it was trained on. Should these things be the case we are still interested in whether the model has discrete continuous consistency, ie whether the model has continuous consistency with for just the redshifts it has been trained on.

If a model is able to generate images where peaks strengthen and then weaken between redshifts, then we can conclusively say that the model cannot be purely step based. If the model would be step based, every element of an image would have to match either the preceding or proceeding redshift that it had been trained with. We will show with counter examples that the model cannot be purely step based.

Most probably our model will generate unreasonable images if we force specific latent codes onto it, such as the origin vector. We are more interested on the quality of our images in expectancy however. To this end the reader may find the images found in the appendix interesting.

The first model we will show to have continuous consistency as well is the model which encoded time by scaling the full latent vector. Figure 6.18 is what we will use to show this. As described in the figure caption there are three rows. The top row is an image sequence generated by the model. From left to right redshift decreases and time increases. The middle row shows which redshifts the training data for the model contained. A green bar means the redshift was contained in the training data whereas a black bar means the model has never encountered data from that redshift. The bottom row contains images from real simulated data as reference.

Looking at the far left of the first three images in the top row we there is a vertical structure. In that structure there is a single pixel peak in the second image which is smaller in both the first and third images. The model thus is not generating images stepwise, but continuously.
6.3. Continuous Conditional GAN

Figure 6.18: Top row contains an image sequence generated by a model encoding time via scaling of the full latent vector. The bar in the middle shows which redshifts the model was trained with. Green means the model was trained with data from that redshift. The bottom row shows an image sequence from real data. Redshift decreases and time increases from left to right.

There is a similar thing happening at the bottom of the first three images generated by the model scaling half of the latent vector in Figure 6.19. Finally we have the something similar in the top of the first three images generated by the model with channel encoding in ??.

Overall looking at the images it seems clear that there is a large degree of consistency over time in a sequence as well as compared to real images of each redshift.

Figure 6.19: Top row contains an image sequence generated by a model encoding time by scaling half of the latent vector.
Figure 6.20: Top row contains an image sequence generated by a model encoding time directly as a channel of the latent vector.
Chapter 7

Conclusion and Future

In this project we analyzed data from dark matter simulations. The data, representing very large amounts of particles which all interact with one another, proved a challenge to work with.

Our primary objective was to be able to build a model that could generate sequences of images which are consistent with different points in time, as well as consistent as a set amongst themselves. A constraint we hoped to be able to fulfill was that the model could generate data for any arbitrary continuous point in time, not just points in time for which training data is available.

The first step in order to train such a model was to build one that could generate data for any single arbitrary redshift, if it was trained on only that. We were able to come build models that could successfully produce distributions resembling the distributions from real simulations. High redshifts proved more difficult than low redshifts for these specialized models. With the help of spectral normalization and scaled exponential linear units, we were able to reach nice results from both a qualitative and a quantitative point of view.

For the single time step models we were able to rely on previously used metrics such as the power spectral densities and the shear peak statistics. These metrics are excellent in order to evaluate quantitatively whether sets of generated data match a certain distribution when viewed individually. Unfortunately, our problem required being able to evaluate whether data samples were consistent with other data samples from the same sequence. This required coming up with a new metric.

The mass difference histogram describes the distribution of change over two data samples. The positive and negative mass difference histograms are defined over an order between data samples. In terms of our data the positive mass difference histogram describes how much more intense pixels get
over time while the negative mass difference histogram describes how much
darker individual pixels get.

The mass difference histogram does not give a clear indication of where
change happens, only how much change happens. More quantitative met-
rics for change are needed to better understand how $n$-body simulations
behave over time.

We developed the continuous conditional GAN model which is able to gen-
erate arbitrary length sequences of data based on some continuous variable.
If one sample is generated in $O(1)$ then a sample with an arbitrary con-
tinuous distance in terms of the continuous variable may be generated in
constant time. The CCGAN is able to generate data from distributions it
does not have training data available for, such as from redshifts between the
redshifts in our data.

We did not test the continuous conditional GAN on other real world data. In
principle the model should be able to condition on any arbitrary continuous
variable, it is possible that the model may not perform on other data as well
as it did on our data.

Further research is needed with regards to dealing with multiple continuous
variables at once. While we briefly discussed how the architecture could be
expanded to deal with such a problem, we did not actually test it.

There are two main issues with continuous conditional generative adversar-
ial networks which further research could help alleviate.

Training samples may not be sparse in the sense that if training data con-
tains $n$ different values for the conditional variable, then all samples used in
training must be available for each of these $n$. That is to say the model has
not been extended to deal with missing values.

Furthermore, the model has a significantly larger batch size for the generator,
dependent on $n$, than in the discriminator. This can slow down training
significantly and make training between the generator and discriminator
more asymmetrical and difficult.

A solution to these two previous problems could go hand in hand, as if it is
possible to omit missing values then it is likely also possible to reduce the
batch size of the generator.
Appendix A

Appendix

A.1 Visualizations of Transform Parameters

A.1.1 Shift and c by Redshift with Fixed Scaling

Effect of the Shift and c Transform Parameters on Data

(a) Redshift $z = 0.000$

(b) Redshift $z = 0.111$
A. Appendix

(c) Redshift $z = 0.250$

(d) Redshift $z = 0.428$

(e) Redshift $z = 0.666$

(f) Redshift $z = 1.000$

(g) Redshift $z = 1.500$

(h) Redshift $z = 2.333$
A.1. Visualizations of Transform Parameters

Figure A.1: Shift increases from top to bottom and $c$ increases from left to right.

(i) Redshift $z = 4.000$

(j) Redshift $z = 9.000$
A. Appendix

A.1.2 Shift and Scaling by Redshift with Fixed c

Effect of the Shift and c Transform Parameters on Data

(a) Redshift $z = 0.000$  
(b) Redshift $z = 0.111$

(c) Redshift $z = 0.250$  
(d) Redshift $z = 0.428$
A.1. Visualizations of Transform Parameters

(e) Redshift $z = 0.666$

(f) Redshift $z = 1.000$

(g) Redshift $z = 1.500$

(h) Redshift $z = 2.333$
Figure A.2: Shift increases from top to bottom and pre-transform data is downscaled more from left to right.
A.1. Visualizations of Transform Parameters

A.1.3 Scaling and c by Redshift with Fixed Shift

Effect of the Scaling and c Transform Parameters on Data

(a) Redshift $z = 0.000$  
(b) Redshift $z = 0.111$

(c) Redshift $z = 0.250$  
(d) Redshift $z = 0.428$
A. Appendix

(e) Redshift $z = 0.666$

(f) Redshift $z = 1.000$

(g) Redshift $z = 1.500$

(h) Redshift $z = 2.333$
A.1. Visualizations of Transform Parameters

Figure A.3: Pre-transform data is downscaled more from top to bottom and $c$ increases from left to right.
A.2 Quantitative Statistics for Models With and Without Spectral Norm and SELU

Figure A.4: Quantitative comparison of different statistics over the course of training with and without scale exponential linear units and spectral normalization. The $x$ axis represents the number of training iterations and the $y$ axis is the $l_2$ norm of the log curves of each respective statistic.
A.3 Images Generated by Continuous Conditional GANs

Figure A.5: Set of images sequences generated by a model encoding time by scaling the full latent vector. In each subfigure the top row contains an image sequence generated by the model. The bar in the middle shows which redshifts the model was trained with. Green means the model was trained with data from that redshift. The bottom row of each subfigure shows an image sequence from real data. Redshift decreases and time increases from left to right.
Figure A.6: Set of images sequences generated by a model encoding time by scaling half of the latent vector. In each subfigure the top row contains an image sequence generated by the model. The bar in the middle shows which redshifts the model was trained with. Green means the model was trained with data from that redshift. The bottom row of each subfigure shows an image sequence from real data. Redshift decreases and time increases from left to right.
Figure A.7: Set of images sequences generated by a model encoding time directly as a channel of the latent vector. In each subfigure the top row contains an image sequence generated by the model. The bar in the middle shows which redshifts the model was trained with. Green means the model was trained with data from that redshift. The bottom row of each subfigure shows an image sequence from real data. Redshift decreases and time increases from left to right.
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Generative Temporal Models for Cosmology

Author(s) (in block letters):

For papers written by groups the names of all authors are required.

Name(s):
Rosenthal

First name(s):
Jonathan Daniel

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date
Zollikon, 1. September 2018

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.