Universiteit Utrecht

Faculty of Science

Machine Learning the Boltzmann Distribution:

Exploring how different training strategies affect the performance of Boltzmann Generators

Bachelor Thesis

Eline Kirsten Kempkes

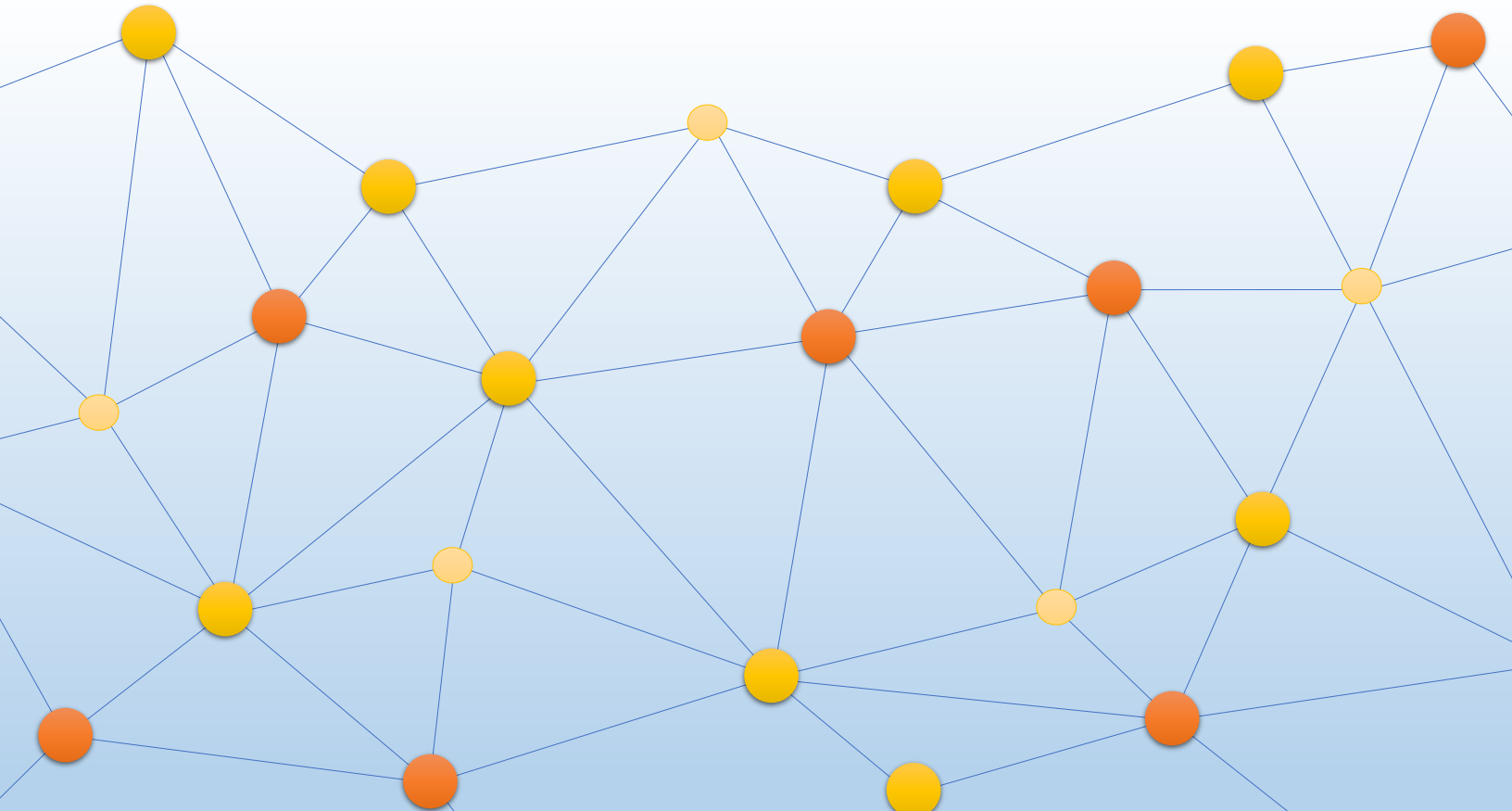
Supervisors:

Dr. Laura Filion
Debye Institute for Nanomaterials Science

BSc. Rinske Alkemade

Physics and Astronomy

16-06-2021



Cover page design: Eline Kempkes

Abstract

A common problem when studying many-body systems is accurately sampling the distribution function. In 2019, Noé et al. (Science, **365**, 6457 (2019)) introduced a novel sampling method: Boltzmann Generators (BGs). Motivated by this, we study how well these generators perform in different settings. BGs are essentially an invertible neural network, which learns a transformation from a simple distribution (Gaussian) to a complex one (Boltzmann) in order to sample from the Boltzmann distribution. To construct this neural network, a "flow of transformations" is used, i.e. the invertible transformation is broken into smaller pieces. Because the BG is invertible, we can train forwards and backwards. We have studied the generators in two different applications. First, we have used the generators to sample an artificial distribution: a smiley face that consisted of three separate Gaussian distributions. We have found that the results generally improve when we use more backwards training, especially in the early stages of training. Second, we have tested whether the BGs can predict the effective colloid-colloid potential in a colloid-polymer mixture. We have found that BGs are a viable way to extract this potential. The conclusion is that BGs are a strong approach for sampling the distribution function of many-body systems and therefore, they have possible applications in many branches of physics.

Contents

1 Introduction

Recently, machine learning techniques are becoming more and more important in different branches of physics, such as quantum matter, biophysics and soft matter. On the soft matter side, they have been extensively used to study, among other things, colloids, liquid crystals, polymers and nanoparticle assemblies [1, 2]. For example, neural networks have been used to classify various types of crystalline order in systems of Yukawa particles and binary hard spheres [3, 4]. As another example, machine learning methods have been used to make predictions on the long-term dynamics of glassy systems [5, 6]. Moreover, unsupervised learning has even been used to detect phase transitions in off-lattice systems [7]. In addition to characterizing such soft matter systems, machine learning methods can also play an important role in speeding up simulations [2]. For example, machine learning can be used to fit complex interaction potentials helping to speed up the energy and force calculations in systems of many particles [8, 9].

Because of these promising results, it is very worthwhile to explore additional applications of machine learning in physics. In this thesis we use machine learning in an attempt to solve a recurring problem: finding likely configurations of a many-body system. For instance: the challenge of finding the most likely folding of a protein, given a certain temperature. To answer these types of questions we need to calculate the probability of a certain configuration, which is given by the well-known Boltzmann distribution: $p \propto \exp(u(x))$, where $u(x)$ is the energy given a configuration x. When we know the Boltzmann distribution, we can easily sample likely configurations. However, one of the major problems when answering these questions is finding the Boltzmann distribution. Methods to find this distribution do exist, namely Monte Carlo simulations or molecular dynamics simulations. However, depending on the system in question, these methods can be computationally expensive and require significant computation power [10]. This is due to the fact that in general an energy surface has several local minima (wells) which are separated by high barriers. Therefore, these trajectory simulations start in one minimum, and usually take a long time before they move to another minimum. Depending on the system, this could take 10^9 to 10^{15} simulation steps [11–13]. Moreover, these simulations can struggle to predict rare events, such as phase transitions or nucleation [14]. Because of these shortcomings, it is clear that we need more efficient methods to sample likely configurations. In this thesis we explore a new sampling algorithm, introduced in 2019 by Noé et al. [11], which combines physics and machine learning and tries to circumvent some of these challenges: the so-called the Boltzmann Generator (BG).

A BG is a sampling method that combines statistical physics and machine learning. The goal of the algorithm is to exploit machine learning in order to learn an invertible transformation using neural networks. Note that a neural network is essentially a non-linear function: it maps input to the output. In a BG, these networks transform a simple distribution (a Gaussian) to a complex one: the Boltzmann Distribution. This means that a BG does not learn the Boltzmann distribution explicitly, it rather learns how to transform a configuration from a simple distribution into a configuration from the Boltzmann distribution. In Ref. [11] it is stated that BGs are a powerful approach for sampling configurations and overcome the limitations of Monte Carlo and molecular dynamics simulations. However, more research needs to be done, since it is not widely researched since the introduction by Ref. [11] in 2019. Therefore, in this project, we investigate how well a BG performs in different applications.

1.1 Aim of this thesis

The aim of this thesis is to explore and test the possibilities of BGs. Therefore, we test two different applications of the BGs. Specifically, we first explore how well a BG performs on a simple distribution function: a smiley face. In this case, the BG transforms a Gaussian distribution to a smiley distribution in order to sample configurations from the latter. For this study, we use several different methods to train the neural networks in order to investigate the performance of the generators. Here, we find that the BGs are a strong approach for sampling likely configurations, providing that the invertible networks are trained forwards and backwards. Secondly, we use a BG to predict the colloid-colloid effective interaction potential in a colloid-polymer mixture. Our findings are that the generators perform well on this task.

The remainder of this thesis is organized as follows: in Chapter 2, we explain the underlying structure of BGs and the different training methods. Then, in Chapter 3, we discuss the problem setup of the smiley distribution and we present its results. Afterwards, we present the setup and results of our second application of BGs, which is predicting the colloid-colloid effective interaction potential of the colloid-polymer mixture. Finally, in Chapter 4, we reflect on our findings and provide an overview of future research directions.

2 Theory & Method

The main goal of this project is to explore a recently introduced machine learning algorithm referred to as Boltzmann Generator (BG) [11]. As described in the introduction, a BG is a machine learning method that is trained to produce configurations drawn from a desired distribution function. In many cases, the desired function is the Boltzmann distribution. Hence, in the following discussion we focus on the Boltzmann distribution for notational simplicity.

The Boltzmann distribution gives the probability of a configuration of a system as a function of the energy and temperature. This distribution is proportional to $\exp\left(\frac{-u(x)}{k_B T}\right)$, where $u(x)$ is the energy of the system in a configuration x, T is the temperature of the system and k_B is the Boltzmann constant. BGs are a relatively new method and were first mentioned in a paper published by Noé et al. in 2019 [11]. In this paper, deep machine learning and statistical mechanics are combined into a powerful method to obtain low-energy configurations of complex systems. For instance, they used BGs to generate equilibrium all-atom structures of macro-molecules. The BGs are used to sample equilibrium states, i.e. states with a low energy and thus a high probability. The main problem is that we cannot directly generate equilibrium samples from the Boltzmann distribution. When we have the energy function and know a certain configuration, we are able to calculate its relative probability according to the Boltzmann distribution. However, we cannot directly sample likely configurations from the Boltzmann distribution since this exact distribution is unknown.

BGs try to generate the low-energy configurations associated with a specific distribution (the Boltzmann distribution) by learning a non-linear transformation. This is a transformation from a Gaussian distribution ($P(z) \equiv P_z$) to the more complex Boltzmann distribution ($P(x) \equiv P_x$). The Gaussian is used for easy sampling, because in this distribution the low-energy configurations are close to each other. Fig. 1 shows an overview of this method. The left figure shows 1000 samples drawn from a simple Gaussian distribution. This simple distribution is the starting point. The goal is to obtain statistically independent samples from a distribution close to the Boltzmann distribution associated with a chosen energy landscape, P_x on the right. The coordinate transformation F_{zx}, from z to x, is done using neural networks. Note that neural networks are essentially non-linear functions that map the input to the output. For a deeper understanding of neural networks we refer the reader to Ref. [15]. These neural networks are trained to generate a configuration drawn from P_x, by using a configuration drawn from the distribution P_z. Moreover, an important criterion is that this transformation is invertible; this means that $F_{zx}^{-1} \equiv F_{xz}$ is well defined. This is necessary for the training of the neural network, which will be explained in Sec. 2.2. In the following sections, the underlying structure of the neural networks of the BGs will be discussed. Moreover, we will explain a method to construct and train an invertible neural network.

2.1 Invertible Neural Networks

The goal of BGs is to learn the transformation F_{zx} and consequently F_{xz}. In this thesis, we refer to F_{zx} as the forwards transformation and F_{xz} as the backwards transformation. As we will see in Sec. 2.2, the forwards transformation is mostly used for sampling low energy configurations and the backwards transformation is necessary for jumping between the various low energy minima. In this case, we want an invertible transformation from a sample drawn from the simple P_z distribution to a sample drawn from the complex Boltzmann distribution P_x (see Fig. 1). To achieve a transformation between the two distributions, invertible neural networks are used. There exist many different approaches for invertible neural networks, such as NICE [16], planar flows [17] and Glow [18]. In this thesis, the real-valued non-volume preserving (NVP) method is used. This method was also successfully used in the BGs developed in Ref. [11]. Volume preservation means that P_z and P_x have the same "volume", i.e. the determinant of the transformation is 1. In contrast, the NVP-method allows for the probability distribution to be scaled differently at different parts of the configuration space.

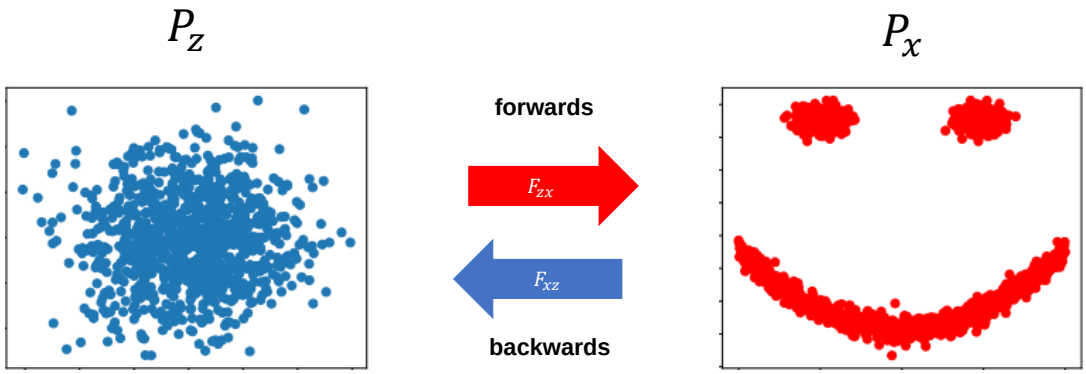


Figure 1: This figure depicts a schematic of the BG algorithm. The P_z distribution is a simple Gaussian distribution, and the P_x is the complex energy distribution. The forwards transformation F_{zx} is the transformation from z to x, and the backwards transformation F_{xz} is from x to z. The goal is to learn this transformation, in order to generate samples drawn from the P_x distribution.

In the following sections, we explain the complete structure of the invertible neural networks, using the NVP-approach. Firstly, we explain how we can define a transformation using a flow of smaller transformations in Sec. 2.1.1. Secondly, we discuss the criteria for a invertible transformation in Sec. 2.1.2. Finally, we present a method that matches these criteria (Secs. 2.1.3 and 2.1.4).

2.1.1 Flow of transformation

We start by breaking up the transformation F_{zx}, from P_z to P_x, into k separate, smaller transformations. These smaller functions are then stitched together afterwards. This way, we obtain a non-linear and easily computable transformation. This method is called a "flow of transformations". In the smaller transformations, neural networks are used. The goal of the neural networks is then to learn what this transformation should look like, i.e. to find out which points in P_z space correspond to which points in P_x space. Note that each point in the P_z space corresponds to a point in the P_x space and vice versa, i.e. it is bijective.

We will now discuss how the flow of transformation is constructed. The principle is as follows: we start with the data points drawn from the distribution P_z in a simple Gaussian distribution. The goal is to obtain the data points x of the Boltzmann distribution, via a flow of transformations. This method is shown in Fig. 2. Firstly, we define $h_k \equiv z$. The next step is to define the first smaller function $f_k(h_k) = h_{k-1}$, which is the first transformation of z. For now, we will not specify this function $f_k(h_k)$, in Sec. 2.1.3 we will elaborate on what this function should look like. This transformation step is repeated k times, in order to construct a flow of transformations. For each step n we define a function $f_n(h_n) = h_{n-1}$, until the last step where we define $f_1(h_1) = h_0 \equiv x$. Here, this x is a configuration whose probability should follow the distribution P_x. The functions f_n are the smaller steps in the bigger transformation. Now, using this sequential flow, we have defined a combined transformation F_{zx}, which brings us from z to x.

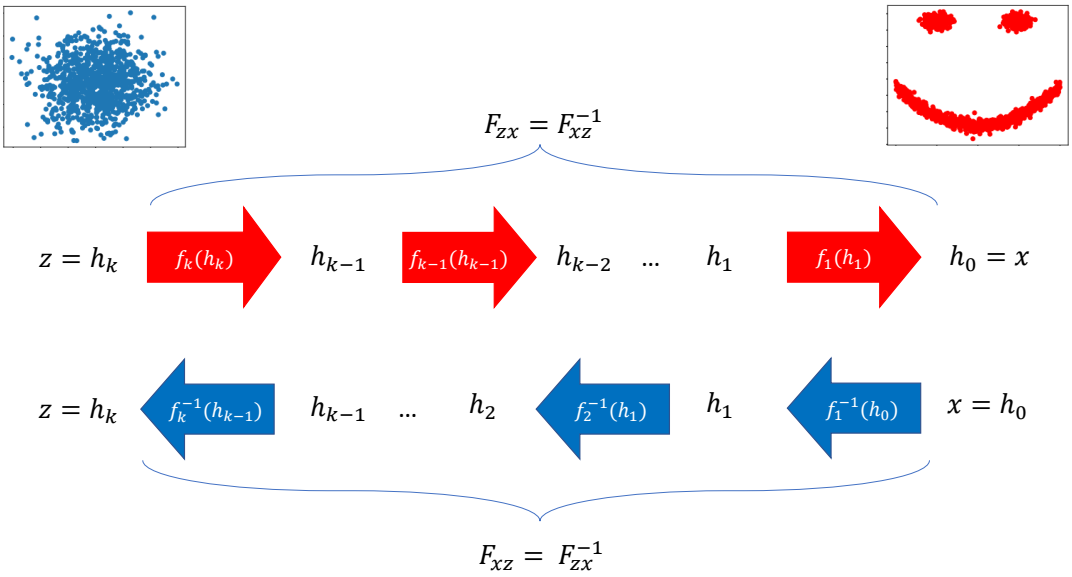


Figure 2: This figure depicts the principle of a flow of transformations. Here, it can be seen how the combined transformation F_{zx} is broken up into smaller transformations, the red arrows. The blue arrows show the architecture of the backwards function F_{xz}, which is also broken up into smaller transformations.

One of the criteria of this transformation is invertibility. Because of this approach, all the smaller transformations have to be invertible as well. This means, for example, that $f_1^{-1}(h_0) = h_1$, and more generally that $f_n^{-1}(h_{n-1}) = h_n$ for every step $n \leq k$. If all these functions are defined, they can be stitched together and form the backwards transformation $F_{xz} = F_{zx}^{-1}$, as shown in Fig. 2. The advantage of breaking up this transformation into smaller functions, is that we can easily compute the Jacobian determinant, which we need for training the neural networks. This will be shown in the next section.

2.1.2 Change of variable formula

In the following two sections, we argue what the transformation functions f_n should look like. These functions f_n are constructed with neural networks. A neural network is a non-linear function that learns how to map the input data to the output data. The networks are trained in different steps, and each step tries to optimize the performance of the networks. To this end, we need to define a loss function. This loss function is an error prediction of the neural network, and the network needs this during this training. Note that in Sec. 2.2 we will discuss how the loss function is defined for our training. During the training, the goal is to minimize this loss function, in other words: to minimize the error of the network. In this loss function, the determinant of the Jacobian matrix plays an important role, as we will see in Sec. 2.2. However, computing the determinant of k smaller functions f_n takes time $\mathcal{O}(k^3)$ in general, which is expensive for high-dimensional data [19]. Because of this, we need to be able to easily compute the determinant of the Jacobian matrix. This criterion helps to define what the functions f_n should look like. Therefore, we will first look into how to calculate this determinant, and after that we will show how we can fulfill this criterion.

Firstly, we consider the change of variable formula. This formula states the relation between the two distributions P_x and P_z and is given by [20]:

$$P_x(x) = P_z(z) \left| \det \left(\frac{\partial F_{zx}(z)}{\partial z} \right) \right|^{-1}. \tag{2.1}$$

Where, $P_x(x)$ is the Boltzmann distribution, and $P_x(z)$ is the Gaussian distribution. A criterion for this formula to be well defined is that the function F_{zx} is bijective [20]. Since F_{zx} is an invertible and thus bijective function, this formula holds. If we take the logarithm we can write it as a summation over all separate layers, and we obtain:

$$\log(P_x(x)) = \log(P_z(z)) + \log \left| \det \left(\frac{\partial F_{zx}(z)}{\partial z} \right) \right|^{-1} \tag{2.2}$$

$$= \log(P_z(F_{xz}(x))) + \log \left| \det \left(\frac{\partial F_{xz}(x)}{\partial x} \right) \right| \tag{2.3}$$

$$= \log(P_z(F_{xz}(x))) + \sum_{n=1}^{k} \log \left| \det \left(\frac{\partial h_n}{\partial h_{n-1}} \right) \right|. \tag{2.4}$$

From Eq. 2.2 to Eq. 2.3 we used the invertibility of the functions $F_{zx} = F_{xz}^{-1}$ which means that $\left| \det \left(\frac{\partial F_{zx}(z)}{\partial z} \right) \right|^{-1} = \left| \det \left(\frac{\partial F_{xz}(x)}{\partial x} \right) \right|$. Then, we used $F_{xz}(x) = z$ to obtain Eq. 2.3. To obtain Eq. 2.4 we used the flow of transformations to split up F_{xz}:

$$\frac{\partial F_{xz}(x)}{\partial x^T} = \prod_{n=1}^{k} \frac{\partial h_n}{\partial h_{n-1}}.$$

This is substituted in Eq. 2.3, and then Eq. 2.4 is obtained.

Eventually, we use the Jacobian determinant in the loss function, which we will discuss in Sec. 2.2. Because of this, the problem that now arises is that computing determinants is very time consuming. Therefore, we need to construct our layers in such a way that the determinant is easy to compute. One of the ways to do this is to use affine coupling layers, introduced by Ref. [20]. This approach will be discussed in the following sections.

2.1.3 Affine Coupling Layers

Affine coupling layers are a way to structure non-linear, invertible transformations and they make sure the Jacobian is easily computable. These transformations are used in the NVP approach (see Ref. [20]). The key is to split the data of dimensionality D, into two sets. The first set containing dimensions 1 to $\frac{D}{2}$ and the second set containing dimensions $\frac{D}{2}$ to D. For example, if $D = 3$, we split the data into the two sets as follows: the first set contains dimensions x and y, the second set contains dimension z. Now, the key is to transform each set in a different way, to get an easily computable determinant of the Jacobian matrix. A small step in the backwards transformation (F_{xz}), so from h_{n-1} to h_n, is defined as follows:

$$h_{n,x} = h_{(n-1),x} \tag{2.5}$$

$$h_{n,y} = h_{(n-1),y} \cdot \exp\left(s(h_{n-1,x})\right) + t(h_{n-1,x}). \tag{2.6}$$

In Eqs. 2.5 and 2.6, $h_{n,x}$ represents the data from the first set of dimensions, and $h_{n,y}$ represents the data from the second set. Further, s and t are neural networks. This transformation is invertible, even if s and t are not invertible themselves. This is due to the fact that in the end this transformation has the same input and output dimensions, and can be written as $y = ax + b$, which is an invertible expression. Hence, we can easily calculate the Jacobian of this transformation. These properties will be proven in the next paragraphs.

Invertibility
The invertibility property is easy to check: if we rewrite Eqs. 2.5 and 2.6 we obtain:

$$h_{(n-1),x} = h_{n,x} \tag{2.7}$$
$$h_{(n-1),y} = (h_{n,y} - t(h_{n,x})) \cdot \exp(-s(h_{n,x})). \tag{2.8}$$

The first equation stays the same, but we switched the right-hand side and the left-hand side. For Eq. 2.8, we use Eq. 2.7 to rewrite $h_{(n-1),x}$ into $h_{n,x}$. If we substitute this into Eq. 2.6, we obtain Eq. 2.8. This proves that this transformation is indeed invertible, which means it can be used in a flow of transformations.

Efficiently Computable Jacobian Determinant
Another criterion is that the determinant of the Jacobian matrix is efficiently computed. The Jacobian matrix of the transformation in Eqs. 2.5 and 2.6 is as follows:

$$\begin{pmatrix} \frac{\partial h_{n,x}}{\partial h_{(n-1),x}} & \frac{\partial h_{n,x}}{\partial h_{(n-1),y}} \\ \frac{\partial h_{n,y}}{\partial h_{(n-1),x}} & \frac{\partial h_{n,y}}{\partial h_{(n-1),y}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{\partial h_{n,y}}{\partial h_{(n-1),x}} & \exp(s(h_{n-1,x})) \end{pmatrix} \tag{2.9}$$

The resulting determinant is indeed easy to compute, since it is a triangular matrix. Because of this, the determinant is given by the easily calculated term: $\exp(s(h_{n-1,x}))$.

2.1.4 Combining Coupling layers

In the last section, we showed what one transformation step (f_n) looks like in a flow of transformations. Now, we will explain how to combine these transformation into a flow, to obtain the larger invertible transformation F_{xz}. This means that the only thing left is to combine different affine coupling layers. If every transformation looks like Eqs. 2.5 and 2.6, the x-coordinate will not change at all: it stays the same in every transformation. A solution for this is to combine these layers in an alternating pattern. This means that the components that do not change in one coupling layer are transformed in the next layer. The Jacobian determinant still remains easily computable, because it is computable for every step. Fig. 3 shows this approach. For example, in the first transformation from h_0 to h_1, the x-coordinate ($h_{0,x}$) remains unchanged. The y-coordinate ($h_{0,y}$) transforms as Eq. 2.6. This is shown in the picture by using the neural networks t and s and a plus and multiplication sign. For the alternating pattern, we require that in the next transformation the x-coordinate ($h_{1,x}$) transforms following Eq. 2.6, and the y-coordinate ($h_{1,y}$) remains the same (Eq. 2.5). This pattern continues in the next steps, indicated with the dots. Because of this pattern, both coordinates change and we obtain a non-linear invertible transformation from x to z.

In conclusion, this section showed an approach to construct an invertible coordinate transformation from a simple distribution P_z to a complex Boltzmann distribution P_x, as shown in Fig. 1. This is done using a flow of transformations, where each transformation is invertible, and because of that, the whole transformation is invertible. Affine coupling layers were needed to make sure the Jacobian of the transformation is easily computable. In this section, we have discussed the underlying structure of BGs, in the next section we discuss how the training of the networks is completed.

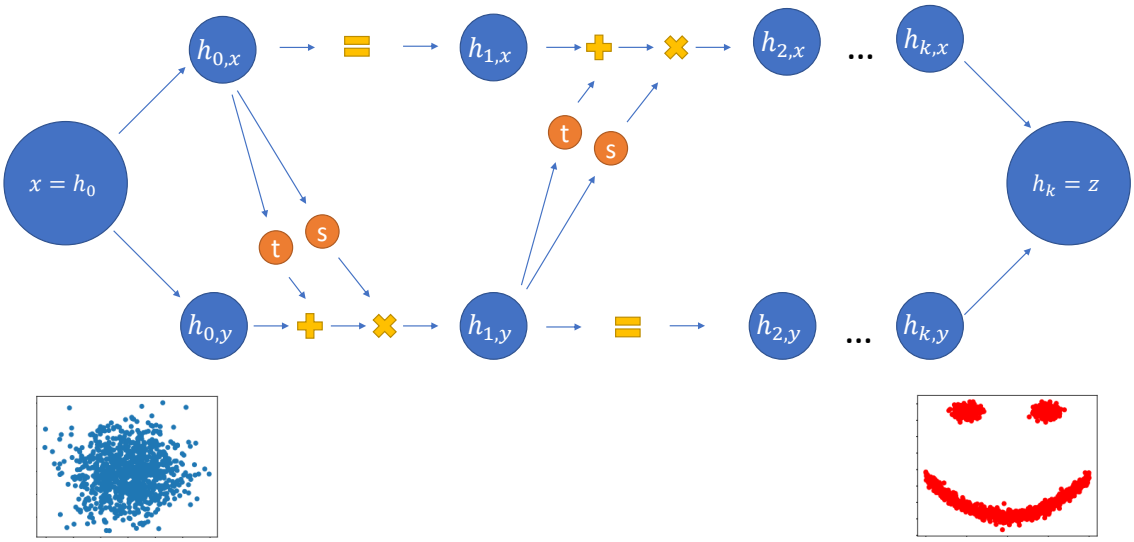


Figure 3: This figure depicts the principle of a flow of transformations via affine coupling layers. We start with x, a configuration from P_x, on the left side. We split this into two parts by its dimensions: $h_{0,x}$ and $h_{0,y}$. Then we use the transformation defined in Eq. 2.5 on $h_{0,x}$ and the transformation from Eq. 2.6 on $h_{0,y}$. Next, we obtain $h_{1,x}$ and $h_{1,y}$. Now we do it the other way around: we use Eq. 2.6 on the x-part and Eq. 2.5 on the y-part. We continue this alternating pattern until we find $h_k = z$.

2.2 Training

When using neural networks, we optimize the algorithm by minimizing the loss function. In this section, we define this loss function and also describe the associated training. As mentioned earlier, the determinant of the Jacobian is used for training. Moreover, because we defined a forwards function, F_{zx} and backwards function, F_{xz}, we can train both directions. In Ref. [11] this is called "training by energy" and "training by example" respectively. Training by energy is the same as training the forwards function i.e. we have a configuration from the known Gaussian distribution and we use this to train the networks to reproduce P_x. Training by example is training the backwards function, i.e. we have configurations drawn from the P_x distribution and we use those to train the networks to reproduce P_z.

2.2.1 Training forwards

Training forwards plays an important role in training BGs. This is because we do not know our distribution P_x exactly. However, when we have a configuration x, we can calculate the energy $u(x)$ exactly. The goal is thus to train the networks such that our P_x matches $u(x)$. Therefore, we start with the Gaussian distribution P_z. The procedure is as follows: we sample random configurations z from P_z. These configurations are transformed using our invertible neural networks, such that $x_{generated} = F_{zx}(z)$. Now we need to find a way of measuring the difference of the generated distribution from the real Boltzmann distribution which is proportional to $\exp(-u(x))$. This can be done by measuring the Kullback-Leibner (KL) divergence [11, 21]. This is also called the relative entropy and can be computed as follows:

$$L_{fw} = \mathbb{E}_z[u(F_{zx}(z)) - \log R_{zx}(z)]. \tag{2.10}$$

Here, L_{fw} is the forwards loss, $\mathbb{E}_z$ stands for the mean value of the configuration z, $u(F_{zx}(z))$ is the energy of the generated configuration x and $R_{zx}(z)$ is the determinant of the Jacobian matrix for the z to x transformation. The energy term is the mean potential energy of the Boltzmann distribution and tries to minimize the energy, in order to make low energy configurations more likely. The other term, $\log R_{zx}(z)$ roughly tries

to maximize the entropy. It is not exactly the entropy, but it is proportional to the scaling between the configurational space volumes. Thus, the KL divergence is roughly equal to the free-energy difference of transforming the Gaussian distribution to the Boltzmann distribution.

2.2.2 Training backwards

Training forwards is very powerful, but in most cases it is not sufficient. Despite the entropy term, the system tends to get stuck in a potential well, as we will see in Chapter 3. One potential solution is training backwards as well, i.e. training the function from complex distribution P_x to the Gaussian P_z. In this case, we need samples from x. These can be obtained from Monte Carlo simulations, molecular dynamics simulations or using experimental observations such as confocal images. In this way, we obtain samples from P_x, which we need for training backwards. For training backwards, the following loss function (L_{bw}) is used [11]:

$$L_{bw} = \mathbb{E}_x \left[u(F_{xz}(x)) - \log R_{xz}(x) \right] \tag{2.11}$$

$$= \mathbb{E}_x \left[\frac{1}{2} ||F_{xz}(x)||^2 - \log R_{xz}(x) \right]. \tag{2.12}$$

Here, $\mathbb{E}_x$ stands for the mean value, the first term $u(F_{xz}(x))$ is the energy of the configuration z, and the second term $R_{xz}(x)$ is the determinant of the Jacobian matrix from F_{xz}. For the energy we substitute the energy function of a harmonic oscillator, since this corresponds to the Gaussian distribution P_z. The second term maximizes the likelihood of the sample to be in the Gaussian density region. This is exactly the same equation as the other loss function (Eq. 2.10), with the only difference being that we use a different distribution that needs to be minimized.

2.2.3 Combining training forwards and backwards

We combine training forwards and training backwards by defining the loss function (L_{total}) as follows:

$$L_{total} = w_{bw} L_{bw} + w_{fw} L_{fw}. \tag{2.13}$$

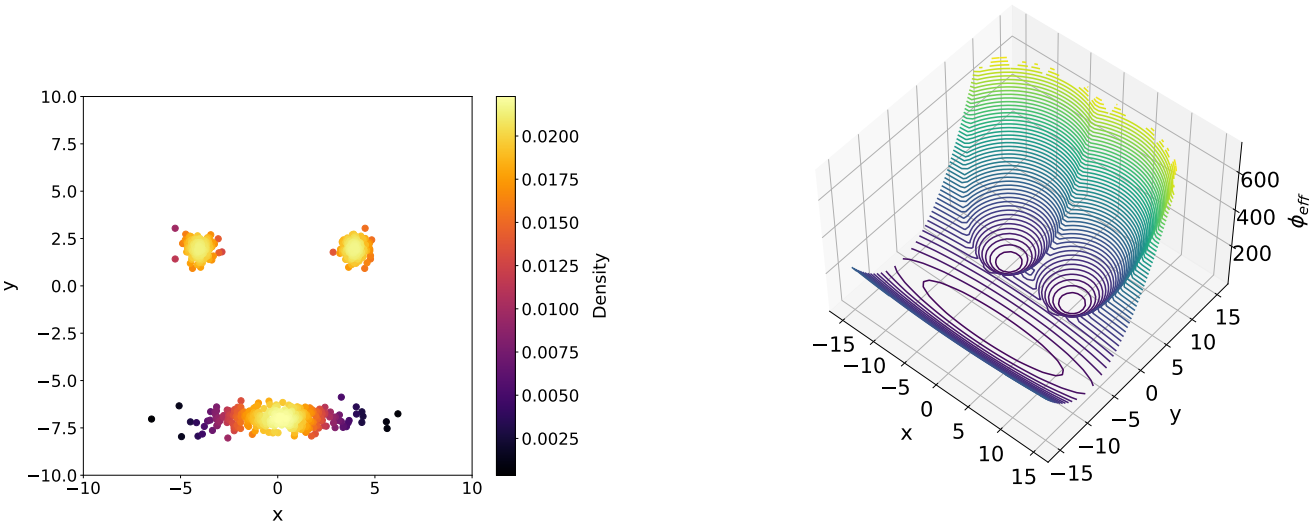
Here, w_{bw} is the weight we give to the backwards loss function, w_{fw} is the weight we give to the forward loss function L_{fw}. Using this total loss function, the networks can be trained forwards and backwards simultaneously.

3 Results & Discussion

In this chapter we present the results of our study of Boltzmann Generators. First, we consider an artificial "smiley face" distribution which we use to explore how the generators work and also for experimenting with the advantages and problems associated with training forwards and training backwards. Second, in Sec. 3.4, we consider a "real" colloidal system, namely a colloid-polymer mixture. Here, we use data from an event-driven molecular dynamics simulation to obtain the effective potential using BGs. We compare the result to the accurate theoretical prediction for the pairwise colloid-colloid effective interaction called the Asakura-Oosawa potential [22].

3.1 The Smiley Distribution

To develop and test BGs, we constructed an artificial distribution which is shown in Fig. 4a. The corresponding energy landscape is shown in Fig. 4b. This artificial distribution consists of three different, disconnected Gaussian distributions. This structure was chosen as i) it is a relatively simple 2D distribution which enabled us to do significant and efficient method testing, and ii) despite its simplicity, it includes three separate minima which allowed us to test how the algorithm performs in a complex, disconnected landscape.



(a) Smiley Distribution

(b) Energy landscape

Figure 4: This figure shows the smiley distribution (a) and its energy landscape (b). The smiley consists of three different Gaussian distributions. The eyes are two different Gaussian distributions, both with a standard deviation of 0.4 for both directions. For the left eye, the mean of the distribution is at $(-4, 2)$. For the right eye, the mean is $(4, 2)$. The mouth of the smiley has a mean of $(0, -7)$. The standard deviation in the x-direction is 2, to get the horizontal line. In the y-direction the standard deviation is 0.4.

3.2 Setting up the Machine Learning Transformation

To set up the transformation F_{zx}, we need to chose an architecture for the neural networks s and t, that appear Sec. 2.1.3. The exact composition of the networks s and t was determined by trial and error. We have tested a few architectures, e.g. with various numbers of hidden layers. Eventually, we chose the one that gave the best results during our tests. Note that we have not optimized the architecture for each situation we encountered, but rather kept the structure the same for all work shown in this section. Specifically, we use s and t neural networks that consist of six different layers: three Linear layers, two Rectified Linear Units and one Tanh layer. Also, we used six affine coupling layers in total, see Sec. 2.1.3. For more information on the layers involved in neural networks, see Ref. [23].

3.3 Training the Boltzmann Generator for the Smiley Distribution

The goal is using the smiley distribution to gain insight into training forwards (F_{zx}) and training backwards ($F_{xz} = F_{xz}^{-1}$), and how to combine these two types. Therefore, we have experimented with four different training methods:

- **Method I**: training forwards, i.e. using the loss function from Eq. 2.10
- **Method II**: training backwards, i.e. using the loss function from Eq. 2.11
- **Method III**: a combination of training forwards and backwards, meaning using the loss function from Eq 2.13. We use different values for the weights w_{fw} and w_{bw}.
- **Method IV**: we use two different training parts. In the first part we train the backwards function only. The variable p (%) indicates the size of this first part with respect to the total training. In the second part of the training we use forwards and backwards training simultaneously.

Note that the idea behind Method IV is that in this way the network learns all the relevant parts of the distribution in the beginning, instead of learning just one energy well. This method is inspired by Ref. [11].

In all four training methods, we used 1000 iterations. When training backwards, we used 500 samples drawn from the smiley distribution to compute the backwards loss (Eq. 2.11). We used 500 samples drawn from a Gaussian distribution for training forwards (Eq. 2.10). The duration of each training was approximately 3 minutes on a laptop. A sample of the code employed is found in Appendix A. In the following sections we will present the results of the four methods.

3.3.1 Method I: Training forwards

For the first training method we have used forwards training only, i.e. using the loss function from Eq. 2.10. This method has the advantage that it is possible even when there are no data samples drawn from the Boltzmann distribution available. The results of this training are shown in Fig. 5. As is clearly visible from the plot in the upper corner, this method fails to sample all the relevant parts of the distribution, since only the left eye of the smiley face is found. Thus, the forwards function fails to sample a configuration drawn from the smiley distribution. We can also use the same learned transformation in the reverse direction, i.e the backwards transformation F_{xz}. An example density plot is shown in the bottom row in Fig. 5. It might also be interesting to take a look which part of the smiley corresponds to which part of the configuration from the backwards function. This is shown in Fig. 6: every color on the left corresponds with the same color on the right. Note that the left eye is roughly transformed into a Gaussian distribution with a mean of $(0, 0)$. However, the backwards function clearly fails to transform the smiley distribution into a Gaussian. The reason for this is that the algorithm does not expect the right eye and the mouth to be there, and thus fails to transform them properly. After training, we again calculate both loss functions (Eqs. 2.10 and 2.11) to use as a measure for the performance of the BG. A good loss value is close to zero. For this training the forwards loss is 2.8 and the backwards loss has a high value of 109. This can be explained by the fact that only one part of the smiley was sampled. Note that the backwards loss is not used during training, but only computed afterwards to analyze the results.

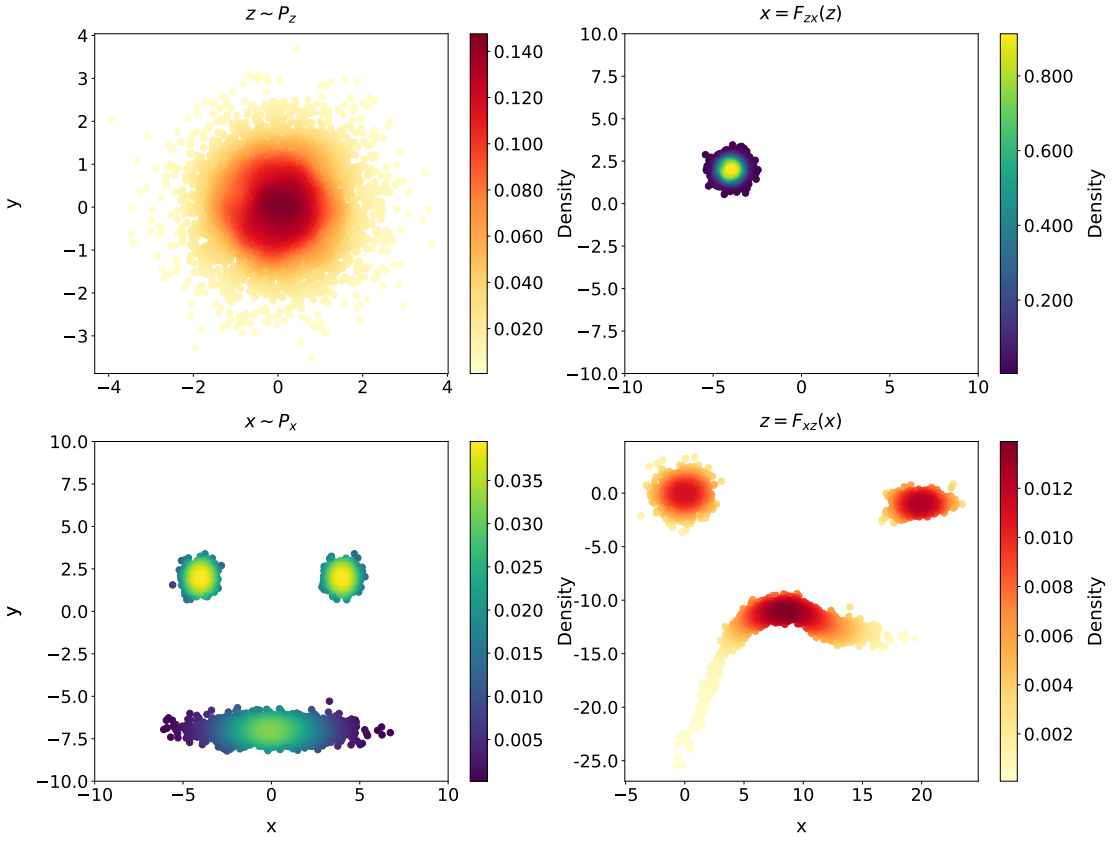


Figure 5: This figure shows the results when we use only the forwards training, i.e. using the loss function from Eq. 2.10. The colors indicate the density of the distribution in each subplot. In the upper left corner, a configuration drawn from a Gaussian distribution P_z is plotted. In the upper right corner the output of the trained function F_{zx} of this configuration is shown. In the left lower corner, a configuration drawn from the smiley face distribution is shown, i.e. $x \sim P_x$. Then we use this configuration as an input to our backwards function F_{xz} and we obtain a configuration from the Gaussian distribution, according to our network. This is shown in the lower right corner.

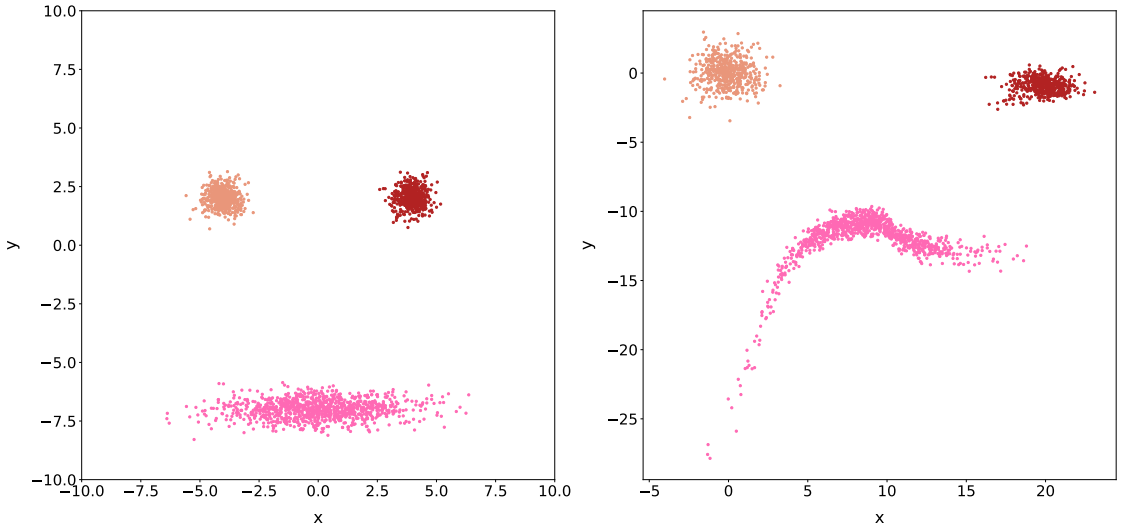


Figure 6: This figure consists of two subplots and shows the result when we train forwards only. The left subplot shows a configuration drawn from the smiley distribution. The outcome of F_{xz} can be seen on the right. This result is also shown in Fig. 5, this figure shows which part of the transformed distribution corresponds to which part of the original distribution: every color on the left corresponds with the same color on the right.

3.3.2 Method II: Training backwards

Now, for the second method, we used training backwards, i.e. using the backwards loss function (Eq. 2.11) to determine F_{zx}. This is possible because we know exactly what the desired distribution looks like (Fig. 4), so we can sample data from this distribution. The results are shown in Fig. 7. In the upper right corner from this figure we see that all the relevant parts of the smiley are sampled. Note that the three different parts are connected with a thin line of points: it fails to transform a Gaussian into a smiley with three perfectly disconnected regions. However, if we compare this with Fig. 5, we see that the results are significantly improved. This is also supported by the values of the losses: the forwards loss is 3.8 and the backwards loss is 3.1, which is significantly lower then the backwards loss from the first method. We have also used the backwards function F_{zx} to transform the smiley into a Gaussian. A density plot is shown in Fig. 7 and in Fig. 8 it can be seen how each part of the smiley transforms into the Gaussian. In these figures we see that F_{xz} resembles a Gaussian with a few gaps. These gaps represent the gaps between the different parts of the smiley. Note that each part of the smiley pertains to a distinct region in the Gaussian distribution. From these figures, we draw the conclusion that we can sample a configuration drawn from the smiley distribution, using a backwards trained BG. The same BG can be used for a backwards transformation, where we obtain a Gaussian with three different regions, that correspond with the three regions of the smiley.

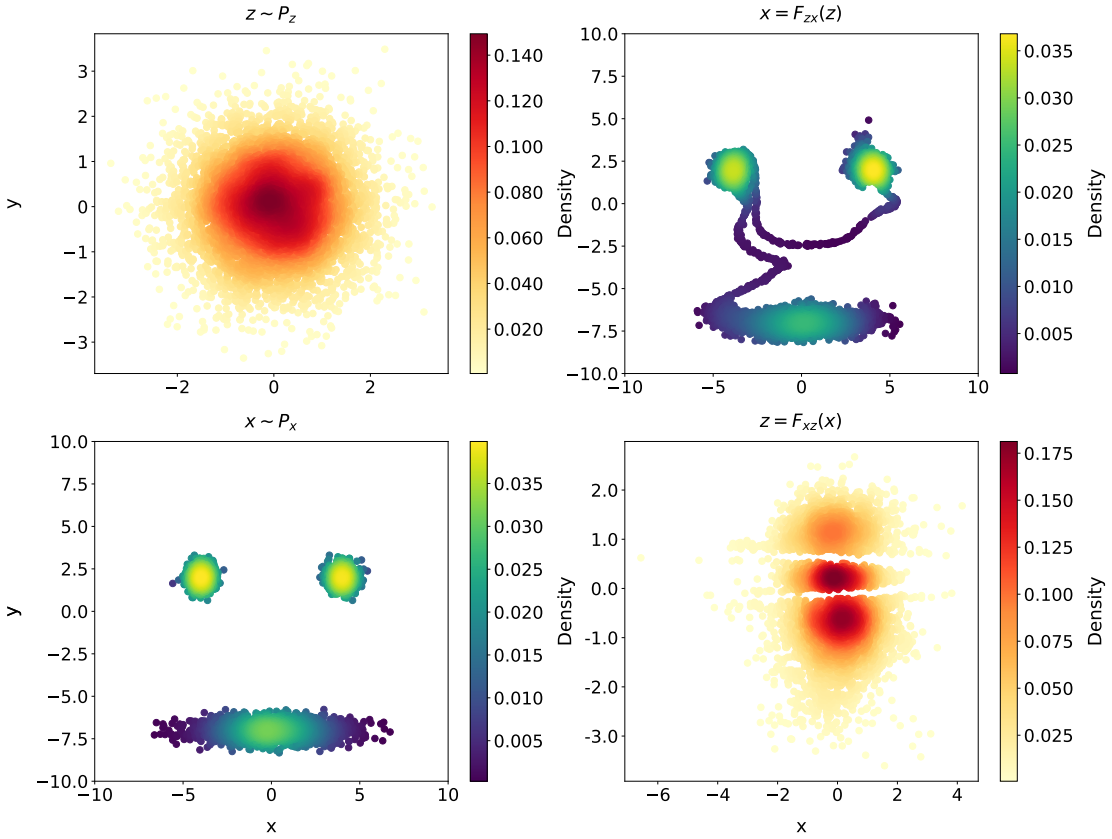


Figure 7: This figure shows the results of the second training method, so with only the backwards training, i.e. using the backwards loss from Eq. 2.11. All these figures are density plots, note the color bar on the right. In the upper left corner, this figure shows a configuration drawn from a Gaussian distribution. In the upper right corner, the result of the forwards function F_{zx} is shown. In the lower left a configuration drawn from the smiley face is shown. The backwards transformation F_{xz} of this configuration can be seen in the lower right corner.

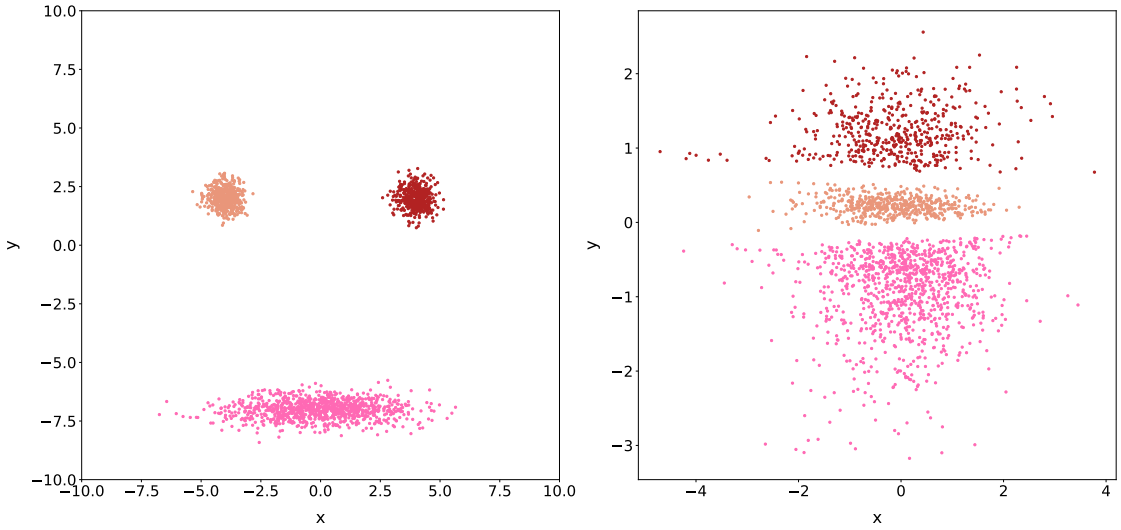


Figure 8: This figure shows the results when we train backwards only. The left figure shows a sample drawn from the smiley distribution. The right figure shows how the backwards function F_{zx} transforms this into a Gaussian. This is the same transformation as in Fig. 7, but now it can be seen how the Gaussian is transformed: each color on the left image corresponds to the same color to the right image.

3.3.3 Method III: Training forwards and backwards simultaneously

For the third training method we have combined training forwards and training backwards. Therefore, we train by using the loss function in Eq. 2.13. We have experimented with nine different settings for the values for the weights w_{fw} and w_{bw}. These values lie in the range $[0.1, 0.9]$, and are separated by 0.1.

We start by exploring a specific example which has the following values for the weights: $w_{fw} = 0.9$ and $w_{bw} = 0.1$. This is a more relevant setting then method I and method II, because when training forwards only, the algorithm has difficulty to find all the relevant parts of the configuration, as shown in Fig. 5. Additionally, in many of the situations where we would want to use a BGs, we will not have a lot of data for training backwards. Therefore, we explore what happens when we mostly train forwards (90%), and we also include a smaller amount of backwards training (10%). The results of this setting are shown in Fig. 9. If we compare this to Method I, we see that the algorithm samples both eyes, instead of one. However, since the mouth is not sampled, this function still fails to sample all the regions. Additionally, the right eye is a less dense region then the left eye. The backwards function F_{xz} associated with this training is shown in Fig. 9 and in Fig. 10. In the latter figure, it can be seen that the backwards function fails to transform the mouth. Comparing with Method I (Fig. 5) this plot is closer to a Gaussian. However, it is easy to see that the backwards function from Method II (Fig. 7) has a better performance. This is supported by the loss values associated with Method III: the forwards loss has a value of 3.0 and the backwards loss is equal to 6.8.

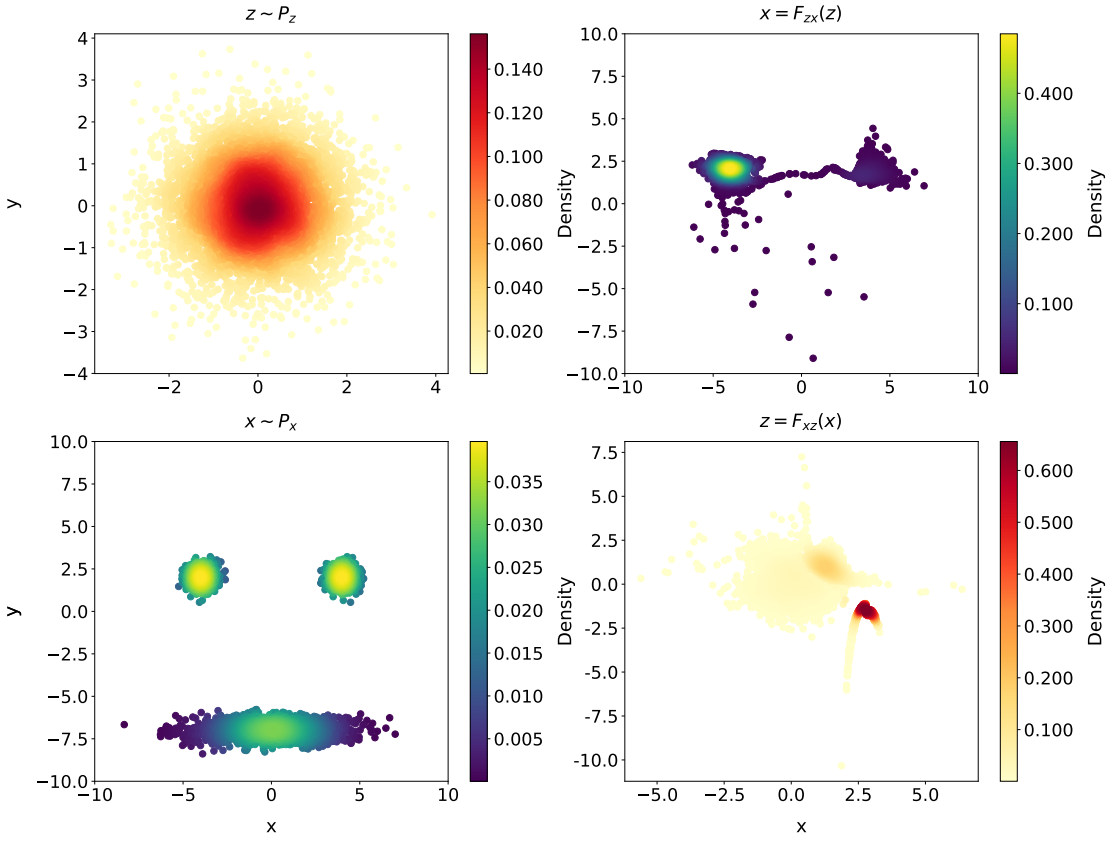


Figure 9: The results of Method III, using $w_{bw} = 0.1$ and $w_{fw} = 0.9$, are shown in four density plots. In the upper left corner, a Gaussian distribution is shown. The upper right corner shows the transformation from this distribution into the smiley distribution, i.e. the result of the forwards function. The figure in the lower left corner shows a configuration drawn from the smiley distribution. In the lower right corner we see the result of the backwards function, F_{xz}.

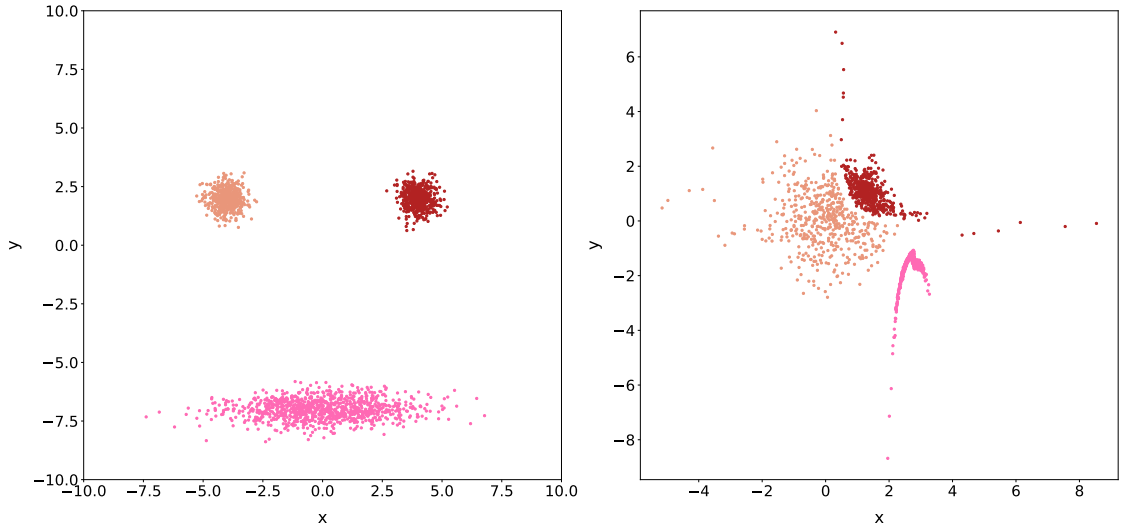


Figure 10: On the left, a configuration of the smiley distribution is shown. On the right, we see the result of the backwards transformation F_{xz}. The same transformation was used in Fig. 9. Here, we can see how the Gaussian is transformed: each part of the smiley corresponds with a part in the Gaussian. This is indicated by the different colors.

We have also trained the network on other settings for the values for these weights w_{fw} and w_{ww} from Eq. 2.13. Figures of these different settings can be found in Appendix (B). To compare the different settings we use the backwards and forwards loss function. The results are shown in Fig. 11. Note that we have also included the first training method and the second training method, since training forwards (Method I) is equivalent to $w_{fw} = 1$ and $w_{bw} = 0$, and training backwards (Method II) is equivalent $w_{fw} = 0$ and $w_{bw} = 0$. From Fig. 11, we see that when we decrease the weight of the backwards loss function, we increase the total loss. From this, we can conclude that training backwards has a positive influence on the results of BGs. Note that this is in agreement with our earlier results from Method I and Method II, training backwards improved the transformation. Interestingly, Fig. 11 shows that the forwards loss is roughly constant, except at $w_{fw} = 0$ and $w_{bw} = 1$. This means that the forwards loss does not depend heavily on w_{fw}.
In summary we have shown that training forwards is not sufficient when the distribution consists of multiple wells. For such cases, as shown in Fig. 11, the more backwards training included, the better the result.

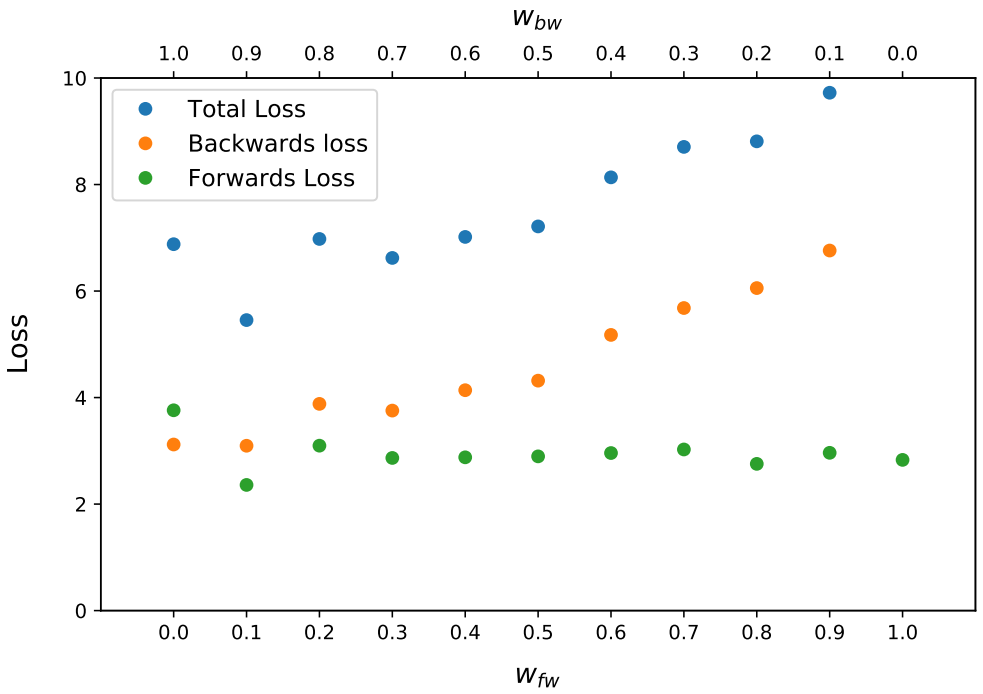


Figure 11: This figure shows the values of the backwards loss (orange), and the forwards loss (green) on the y-axis. The total loss, which is the sum of the forwards and backwards loss is shown in blue. On the lower x-axis the weight of the forward loss function (w_{fw}) is plotted. On the upper x-axis the weight of the backwards loss (w_{bw}) is plotted. Note that the value of the total and backwards loss at $w_{bw} = 0$ and $w_{fw} = 1$ is not shown. This is because the values (112 and 109 respectively) made the figure less readable. Also, the forwards loss at $w_{bw} = 1$ and $w_{fw} = 0$ are not used for training, only for analyzing the results afterwards.

3.3.4 Method IV: Training in parts

In the paper from Noé et al. (2019) [11] it is suggested that training backwards is especially interesting in the earlier stages of the training. The reasoning behind this is that with this strategy the algorithm can explore all the relevant parts of the distribution and thus has a solid basis to start from. In our particular case, we hope that it will allow the algorithm to explore both eyes and the mouth. To investigate this method we have chosen to split the training into two parts. During the first part of the training ($p\%$) we use the backwards loss function (Eq. 2.11). Here, p is the percentage of timesteps we use for backwards training only. For the remaining part of the training ($100 - p\%$) we use the total loss function (Eq. 2.13), i.e. we train forwards and backwards simultaneously. In this way, we can vary the weights of the total loss function in the same way as in Method III (Sec. 3.3.3). The expectation is that the total loss will be lower and the quality of our networks and distributions will improve. In this section, we first explore the optimal value for p and we use this result to compare the values of the forwards and backwards loss with the training methods I-III.

To explore the optimum value of p, we measured the total loss for five different settings for the weights w_{fw} and w_{bw}, for a range of p. The results are shown in Fig. 12. It can be seen that the loss function improves significantly when backwards training is used in the beginning. This holds for all values of the weights w_{fw} and w_{bw}. The minimum is reached at $p = 20\%$, after that the total loss does not decrease significantly. From this, we draw the conclusion that training backwards in the earlier stage of training can improve the results significantly. Moreover, this improvement is reached at $p = 20\%$, and for higher percentages of p, the total loss it does not improve significantly.

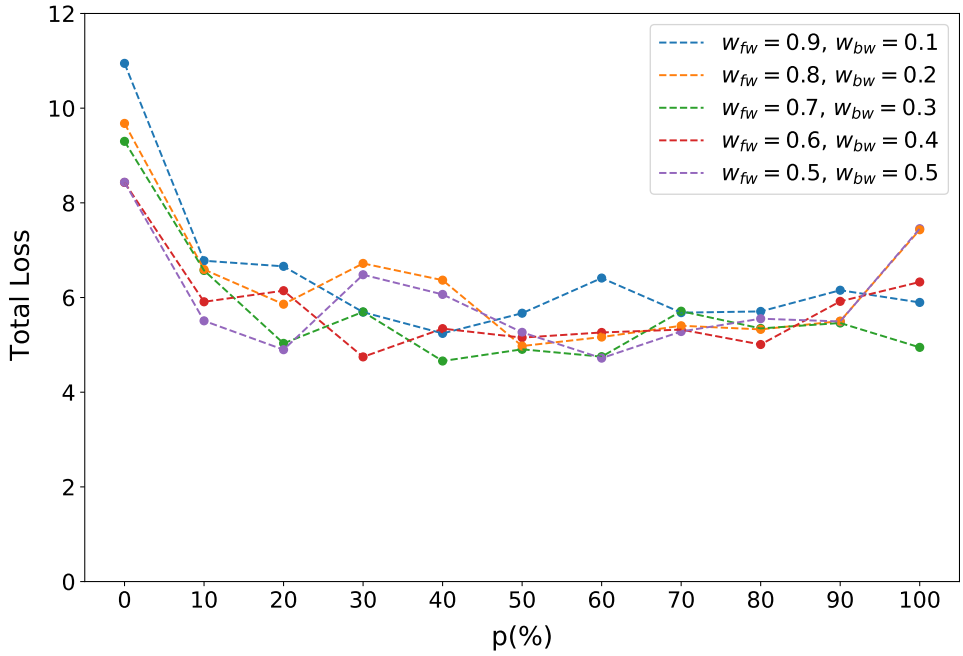


Figure 12: This figure shows the total loss function for different values of p. Also, five different values for w_{bw} and w_{fw} are used, which are indicated with different colors. Note that the minimum is reached at $p = 20\%$, for higher values the loss does not decrease significantly.

3.3.5 Method IV with $p = 20\%$

First, having established that the optimum value for p is 20%, we now examine in more detail what this training looks like for our smiley.In this case, we use $w_{fw} = 0.9$ and $w_{bw} = 0.1$ in order to be able to compare it with Method III, where we presented the results with the same values for the weights. The results are shown in Fig. 13. As shown in this figure, this network succeeds in sampling all the three regions of the smiley, although the left eye is not very dense. This is an improvement if we compare this to Fig. 9, because now all three different parts of the smiley appear. However, there is a thin line between the three parts so it fails to sample three fully disconnected regions. The value of the forwards loss is 1.5 and the backwards loss has the value of 4.4. This also indicates that the results are improved in comparison to those shown in Fig.9 associated with Method III.

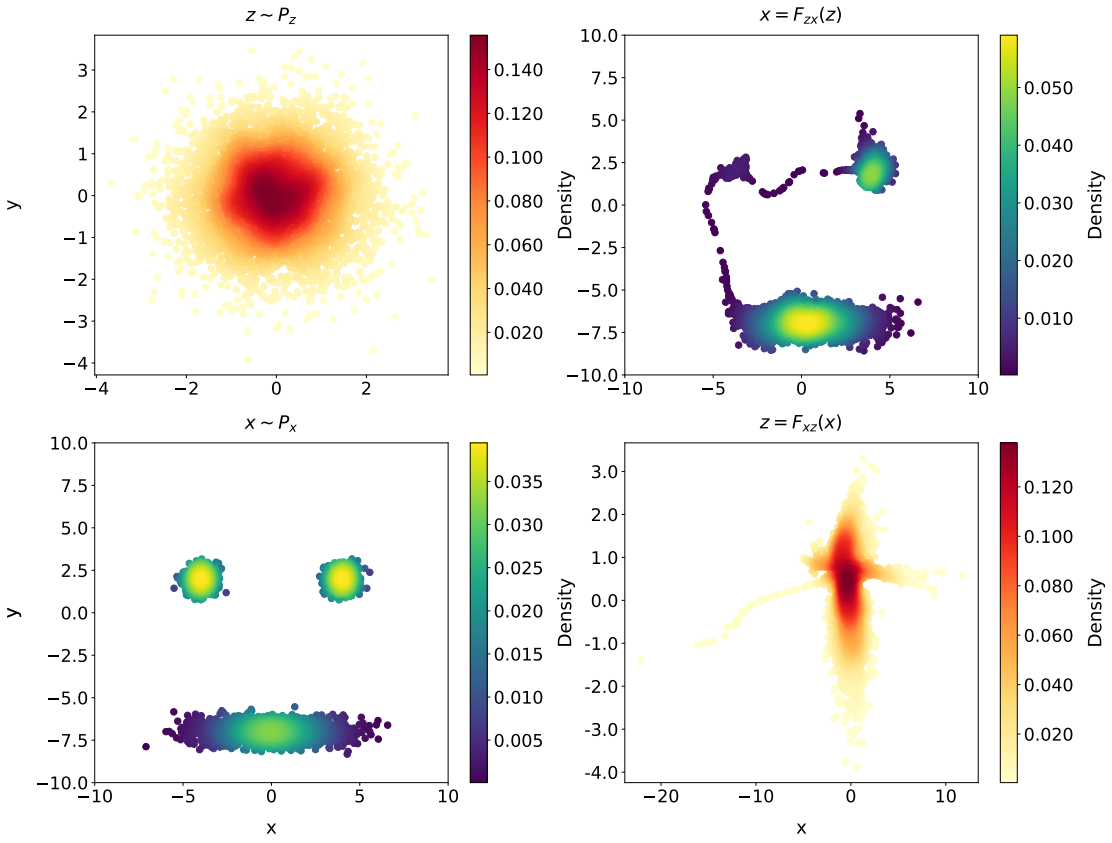


Figure 13: This figure shows the result of Method IV, with the setting $w_{fw} = 0.9$ and $w_{bw} = 0.1$. In the upper left corner we see a Gaussian distribution. In the upper right corner we see the forwards transformation (F_{zx}) of this distribution into the smiley. The lower left corner shows a configuration drawn from the smiley distribution, and the lower right corner shows the backwards transformation (F_{xz}) of this configuration.

Second, we examine how the backwards, forwards, and total loss change as we change the weights w_{bw} and w_{fw} in Method IV with $p = 20\%$. The results are shown in Fig. 14. Note that the distributions associated with these results can be found in Appendix C. In Fig. 14 we see that the backwards and forwards loss are both roughly constant. There is an exception at $w_{fw} = 1$ and $w_{bw} = 0$ where the backwards loss is very high (37). Note that this value is not plotted for readability reasons. Because the total loss stays roughly constant, we can draw the following conclusion: when we train backwards in the early stage, the setting of the weights in the later stage is not important, since this does not affect the total loss.

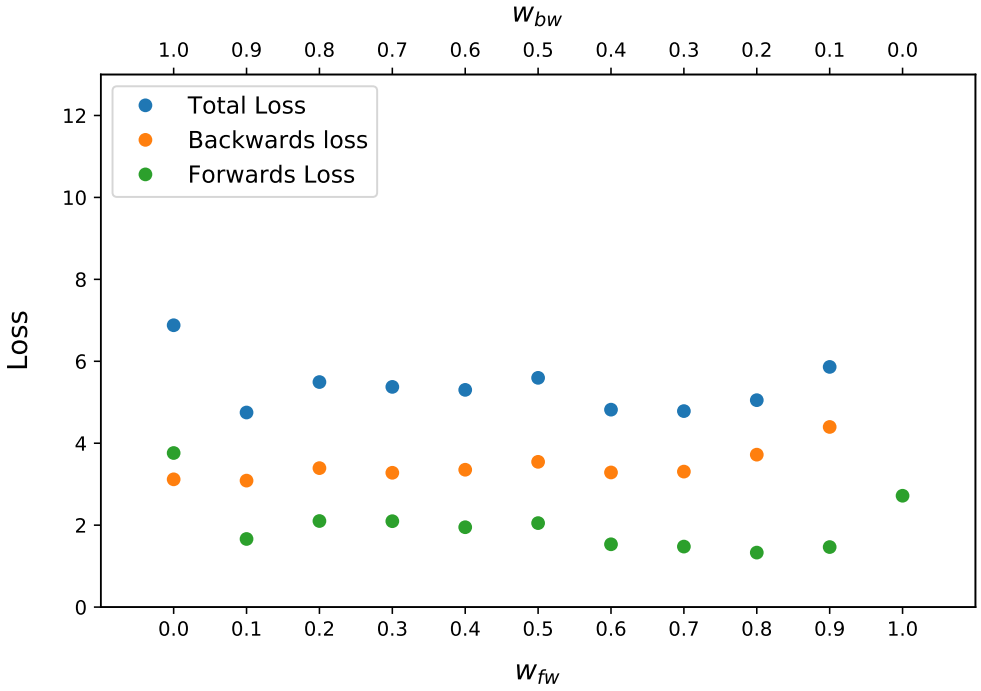


Figure 14: This figure shows the values of the backwards loss (orange), the forwards loss (green) and the sum of the two (blue) of Method IV with $p = 20\%$. On the lower x-axis the value of weight of the forwards loss during training is plotted. On the upper x-axis, we see the weight of the backwards loss. Note that for readability the value of the backwards loss (37) for $w_{fw} = 1$ and $w_{bw} = 0$ is not plotted.

3.3.6 Comparing Method I-IV

Finally, we compare the losses of Method I-III with Method IV (with $p = 20\%$). In Fig. 15 the values of the total loss are plotted against the values for the weights (w_{fw} and w_{bw}). It can be seen that the total loss function of Method IV is in general lower then the loss function of Method I-III. Thus, in this case training backwards only in the early stage of training improved the results. Furthermore, if we train backwards only in the early stage, then the setting of the weights in the later stage is not important, since the total loss function is approximately constant (except for $w_{fw} = 1$ and $w_{bw} = 0$, which do not appear in the figure for readability but are recorded in the caption.).

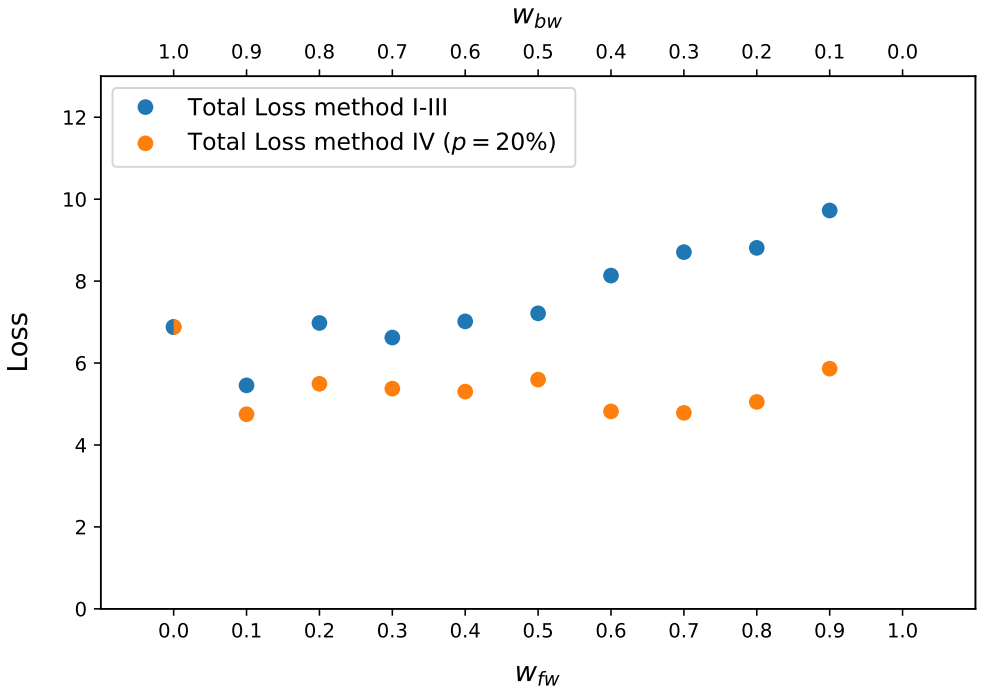


Figure 15: This figure shows a comparison between the total loss of Method I-III (blue) and Method IV, with $p = 20\%$, (orange). On the lower x-axis, we see the different values for the weight of the forwards loss. On the upper x-axis, we see the different values for the weight of the backwards loss. Note that for readability the values of the backward loss at $w_{fw} = 1$ and $w_{bw} = 0$ are not shown (112, 39, method II and method IV respectively). Additionally, note that at $w_{fw} = 0$ and $w_{bw} = 1$ the training methods are equivalent, and therefore they have the same loss.

In conclusion, we have used an artificial distribution, the smiley distribution, to experiment with four different training methods. Moreover, we have used different settings for the weight of the forwards function and the backwards function. We have found that training backwards is important to find all the relevant regions of the distribution. Furthermore, we found that training backwards is most important in the early stage of training.

3.4 Colloid-Polymer Mixtures

In this section we consider a "real" physical system, namely a colloid-polymer mixture. The goal of this section is to use a BG to approximate the colloid-colloid interaction in this system, when the polymers are integrated out. Here, the BG is only trained backwards, i.e. we train the function F_{xz} and use the loss function from Eq. 2.11. Once we have trained the networks, we can use them to perform a forwards transformation. In this way, we generate configurations drawn from the Boltzmann distribution. Then, we can calculate the effective potential of this system, according to the BG. In this section we first describe the colloid-polymer mixture we will study. Then, we describe how we apply BGs to the system.

3.4.1 Colloid-Polymer Model

The system exists of two colloids in a sea of ideal polymers. A figure of this system is shown in Fig. 16. The colloids are treated as hard spheres with a diameter σ_c, i.e. they cannot overlap each other. The polymers (diameter σ_p) are treated as ideal gas particles in that they can overlap each other without a any energy cost. However, a polymer cannot overlap with a colloidal particle, and the distance of closest approach is $\sigma_{cp} = \frac{\sigma_c+\sigma_p}{2}$. We chose this system because i) there exist an accurate prediction of the effective potential and ii) we are able to use data from a readily available event-driven molecular dynamics (EDMD) simulation.

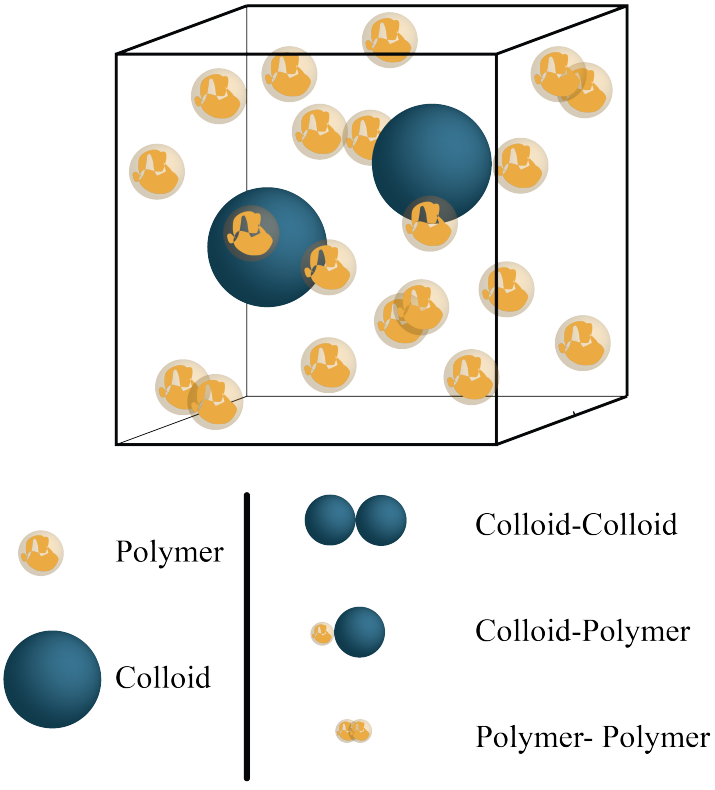


Figure 16: This figure shows a colloid-polymer mixture. The mixture consists of two colloids, which are treated as hard spheres, and a sea of polymers. The polymer-polymer interaction is ideal while the colloid-polymer interaction is hard-sphere like, i.e. the colloids and polymers cannot overlap.

This system has been studied extensively in the past. One of the most important results is that the effective pair-wise interaction ($\phi(r)$), between the colloids at a distance r, due to the polymers can be accurately described by the Asakura-Oosawa potential which is given by [22]:

$$\beta\phi(r) = \begin{cases} \infty & r < \sigma_c \\ -z\frac{4\pi\sigma_{cp}^3}{3}\left(1 - \frac{3r}{\sigma_{cp}} + \frac{1}{16}\left(\frac{r}{\sigma_{cp}}\right)^3\right) & \sigma_c \leq r \leq \sigma_c + \sigma_p \\ 0 & r > \sigma_c + \sigma_p \end{cases} \tag{3.1}$$

Here, $\beta = \frac{1}{k_B T}$ where T is the temperature and k_B the Boltzmann constant, r is the distance between the pair of colloids, and z is the fugacity of the polymers. Since the polymers are ideal gas particles, the fugacity is given by: $z = \frac{6}{\pi}\frac{\eta}{\sigma_p^3}$, where η is the packing fraction of the polymers. This equation describes an attractive potential between a pair of colloids when their separation r is in the range of $\sigma_c \leq r \leq \sigma_c + \sigma_p$. The strength of this attraction is is proportional to the fugacity (z) of the polymer. Fig. 17 shows a plot of the Asakura-Oosawa potential.

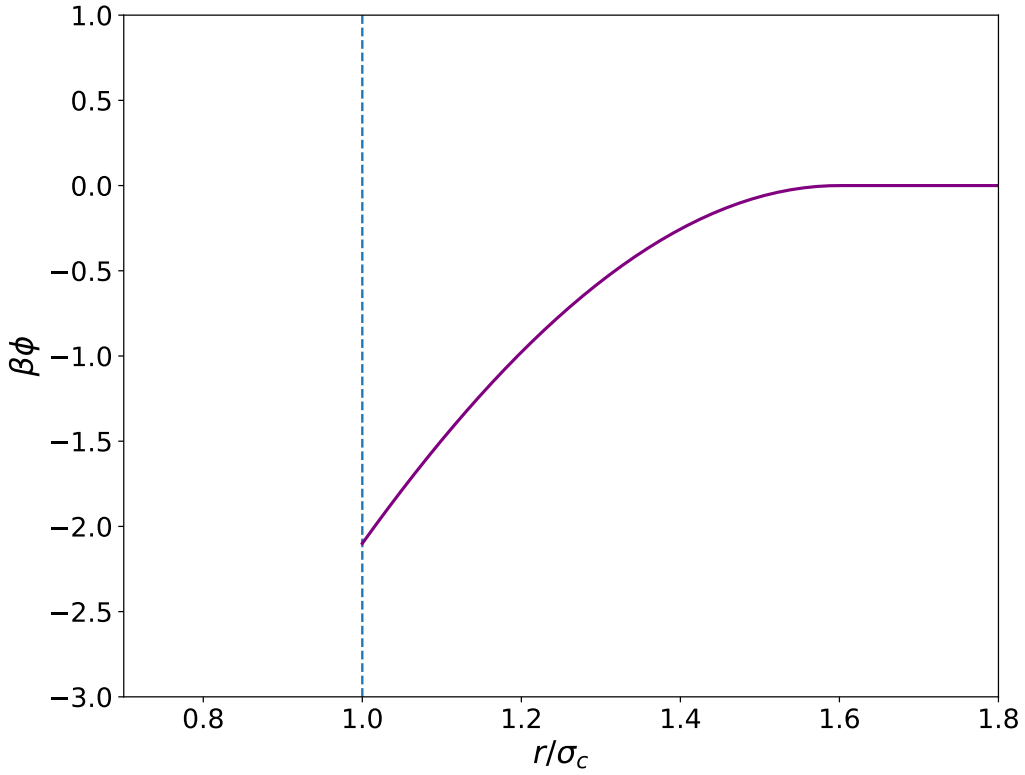


Figure 17: This figure shows a plot of the effective pair potential of two colloidal particles in a sea of polymers: the Asakura-Oosawa potential (Eq. 3.1). In this figure, the packing fraction of the polymer is $\eta = 0.6$, the colloid diameter is $\sigma_c = 1$ and the polymer diameter is $\sigma_p = 0.6$.

3.4.2 Obtaining training data for the colloid-polymer mixture

As mentioned in the introduction to this section, we will only be training backwards for this case. Hence, for the training we require a number of configurations of the colloid-polymer mixture. To obtain these data sets, we used an event-driven molecular dynamics simulation (EDMD) code that simulated the colloidal polymer mixture in the $N_c \mu_p VT$ ensemble, where N_c denotes the number of colloids, μ_p the chemical potential of the polymers, V is the volume of the system and T is the temperature. The EDMD code was written by Frank Smallenburg and Laura Filion. In this simulation, the packing fraction of the colloids was 0.01, the packing fraction of the polymers, $\eta = 0.6$. Additionally, $\sigma_c = 1$ and $\sigma_p = 0.6$.

3.4.3 Preparing data for the Boltzmann Generators

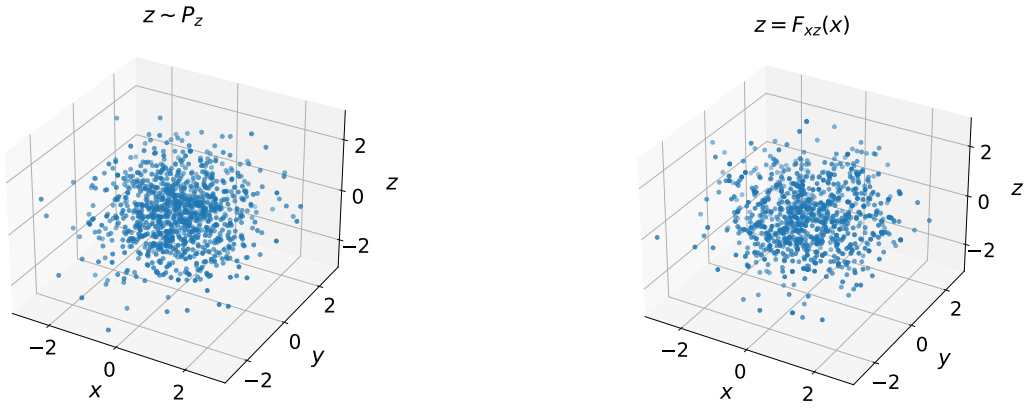
From the EDMD simulation we obtain samples from dx, dy and dz, which constitute the distance vector between colloids. Before we can use this for the BG, we take the absolute values: $|dx|$, $|dy|$ and $|dz|$. In other words, we map everything to the positive octant. The reasoning for this is that now we need less data for training the neural networks, since we only need to consider one octant.

3.4.4 Setting up the Machine Learning Transformation

In this section, we will mention some technical details associated with the neural networks and their training. First of all, we need to discuss the architecture of the neural networks s and t. The network s consist of five layers: three Linear layers and two Rectified Linear units. The t network has six layers: three Linear layers, two Rectified Linear Units and one Tanh layer. For more information about these layers, we again refer the reader to Ref. [23]. We used six affine coupling layers in total, see Sec. 2.1.3. For training backwards, we use a sample of 3000 points from the EDMD simulation. The training consisted of 1500 iterations. Each iteration, we select 300 random points from the sample to calculate the backwards loss (Eq. 2.11). The duration of the training is approximately 2 minutes on a laptop.

3.4.5 Comparing Configurations

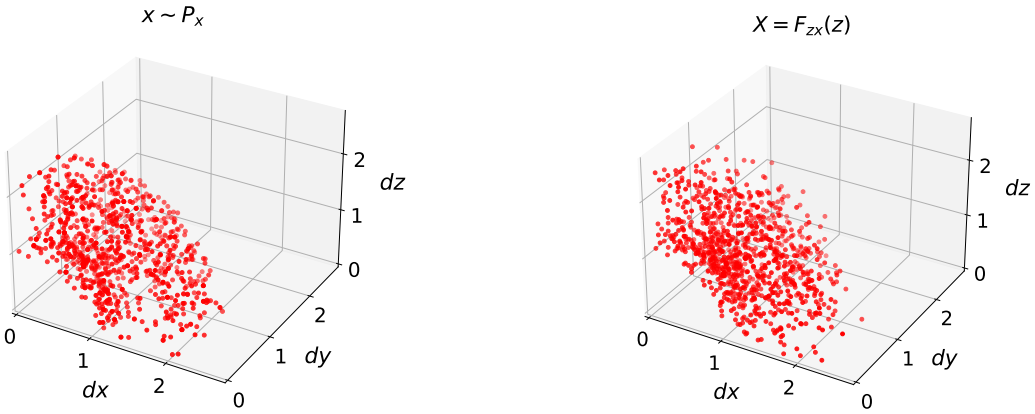
In order the evaluate the Boltzmann generator we present the results of the forwards and backwards transformations. The comparison between the real Gaussian distribution and the backwards transformation is shown in Fig. 18. We see that the backwards transformation roughly resembles a Gaussian Distribution, and thus the backwards function succeeds in sampling this.



(a) Gaussian Distribution

(b) Result from F_{zx}

Figure 18: The left figure shows a Gaussian distribution. The figure on the right shows the result of a backwards transformation, i.e. $z = F_{zx}(x)$



(a) Real Distribution

(b) Result from F_{zx}

Figure 19: The left figure shows a sample drawn from the EDMD simulation. The figure on the right shows the result of a forwards transformation, i.e. $x = F_{zx}(z)$

In Fig. 19 we show a comparison between a sample drawn from the EDMD simulation, and a forwards transformation (F_{zx}) from the BG. It can be seen that the forwards transformation resembles the real distribution. However, in the configuration of the BG there exists more points with a higher value for dx, dy or dz. The backward loss of this transformation is 1.7.

3.4.6 Effective potential

Finally, we predict the effective potential using the BGs. To this end, we sample 5000 values from a normal distribution. Then we use the forwards transformation, i.e. $x = F_{zx}(z)$. Next, we make a histogram of the separation $r = \sqrt{dx^2 + dy^2 + dz^2}$. We do exactly the same for a sample from the EDMD simulation, in order to compare them. After renormalizing the histograms to correct for the change in coordinates from Cartesian to spherical coordinates, we obtain Fig. 20. We see that the histogram of data from the BG resembles the histogram of the EDMD data. However, at $r = 1$ the peak of the BG data is lower then the data from the simulation. Also, at $r = 2.3$, the generated data has less visits than the simulation data. Thus, the BG failed to sample correctly around the edges, i.e. at $r = \sigma_c$ and at the end of the box.

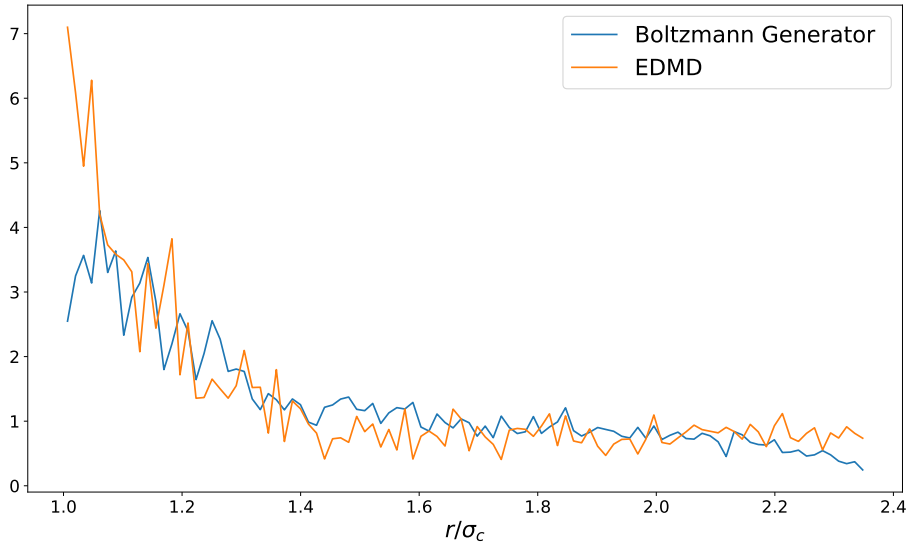


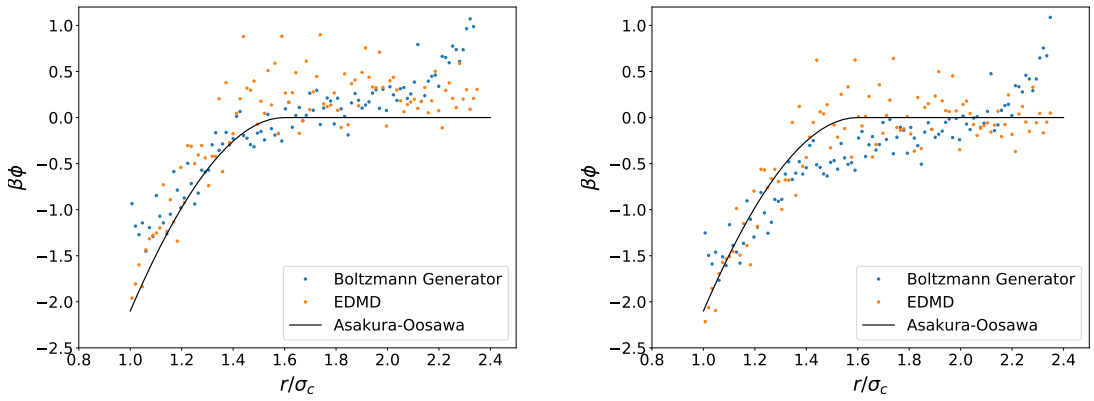
Figure 20: This figure shows two different histograms of the separation r. The orange histogram is a histogram from the data from the EDMD-simulation. The blue histogram uses the generated data from the forwards transformation. Both histograms are renormalized to correct for the change in coordinates.

With a histogram of the distances, we can predict the colloid-colloid effective interaction via:

$$\phi(r) = -\log\left(p(r)\right), \tag{3.2}$$

where p is the probability distribution. Note that this is the effective potential up to a constant. Therefore, we make sure that the mean of the points with $r > \sigma_c + \sigma_p$ is equal to zero, as the polymers cannot effect the colloid-colloid interaction outside of this range (see Eq. 3.1). This is shown in Fig. 21.

To compare our results to the Asakura-Oosawa prediction (Eq. 3.1 and Fig. 17), we fit the Boltzmann Generated interaction data and the EDMD data to the Asakura-Oosawa potential given in Eq.(3.1), where we allowed the fit to find the best choices of the fugacity z, colloid diameter σ_c and polymer diameter σ_p. The model fits are shown in Fig. 22. It can be seen that the models are close to the theoretical value. However, note that at $r = 2.3$ the points from the BG predict a higher potential. This is in agreement with our earlier conclusion that the BG fails to accurately sample the edges of the system. The real values of the fitting parameters are $z = 5.3$, $\sigma_c = 1$ and $\sigma_p = 0.6$. The respective parameter values of the points from the BG are 5.2, 1.0 and 0.6. The respective parameter values of the EDMD points are 6.0, 1.0 and 0.59. Note that these values differ slightly for every generated configuration. From these parameter values and from Fig. 22 we conclude that the BGs can be used to accurately find effective colloid-colloid interactions in colloid-polymer mixtures.



(a) Potential before shift

(b) Potential after shift

Figure 21: The left figure shows the potential obtained with the generated data (blue) from the forwards transformation. In orange it shows the potential obtained with the data from the EDMD simulation. Note that this is the effective potential up to a constant. The black line is the theoretical value of the effective potential (Eq. 3.1) The figure on the right shows the effective potentials after the shift.

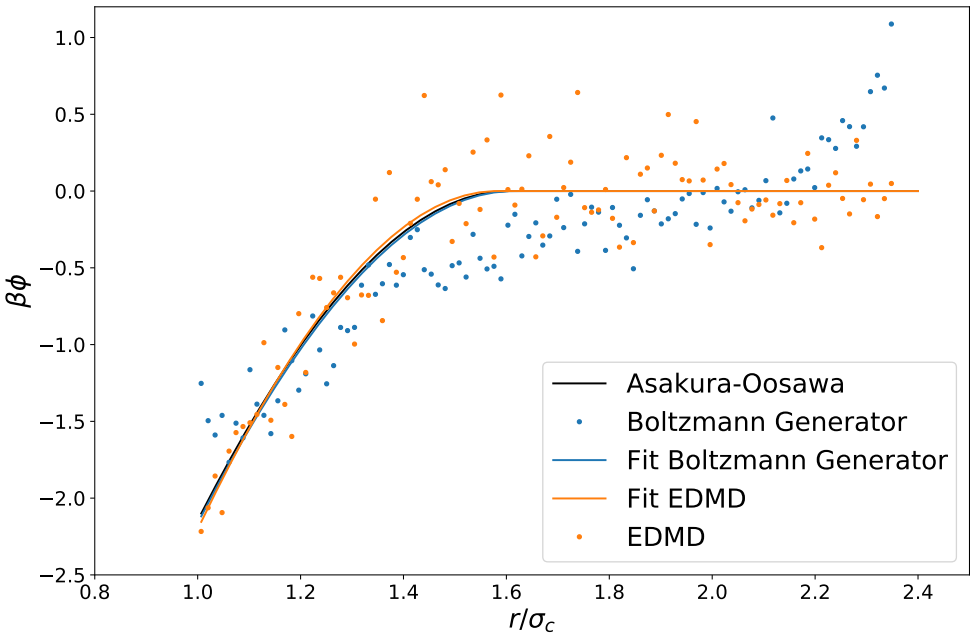


Figure 22: This figure shows a model fit of the effective potential $\beta\phi$ through the simulated points (orange) and to the generated points (blue). On the x-axis the separation r is plotted. The black line is the theoretical interaction potential.

4 Conclusions & Outlook

In this thesis we studied Boltzmann Generators. First, we studied how well a BG performs on an artificial distribution, the smiley distribution. We have used four different training methods. In Method I, we trained forwards (F_{zx}) only. In this case, the network failed to sample all the relevant parts of the smiley. Specifically, the network sampled only one minimum and the other ones were not visible. In Method II we trained only the backwards function (F_{xz}) . Here, the BG succeeded in sampling all minima. The third method involved training forwards and backwards simultaneously. In this setting, we have used different values for the weights for the forwards loss and backwards loss functions. In this way, we could adjust how much we train forwards and how much we train backwards. We have found that generally the results improve when we use more backwards training.

Method IV consisted of two different stages of training. During the first stage, which varied in size, we trained the backwards function only. During the second stage (the remaining iterations) we trained forwards and backwards simultaneously. The most important outcome is that the results improved compared to the training Methods I-III. We found that when the first training stage was 20% of the total training, the loss function was optimal. Specifically, we found that when we increased this percentage, the loss function did not improve. Also, for $p = 20\%$, the total loss function did not depend on the weights of the forwards loss and the backwards loss in the second training stage. Therefore, we can conclude that training backwards is most important in the early stage of training.
In the future, we could explore what happens when we change the composition of the neural networks s and t. For example, we could vary the number of layers or we could include other types of layers, such as the Exponential linear unit. Also, it would be interesting to vary the number of data points given to the network during training. We used 500 samples for each iteration during the training, and it would be interesting to see how well the BGs perform using less training data.

Secondly we have used the BG to predict the effective colloid-colloid interaction potential in a colloid-polymer mixture. For this, we trained only the backwards function F_{xz}. Then we have used the forwards function F_{zx} to produce the colloid configurations of the colloid-polymer mixture. From these configurations we have predicted the effective potential of the colloid-colloid interaction, which agreed well with the theoretical prediction. This shows that BGs are one potential way to extract effective interactions when only colloid configurations are known, for instance, from confocal images of colloidal systems. Moreover, it shows that the BG can be used for sampling 3D-systems. However, note that in this case it was not needed to use the BG to generate configurations: we could have used the configurations from the EDMD simulations to approximate the effective potential.

In summary, Boltzmann Generations can be used to generate configurations drawn from Boltzmann distributions. However, it is important to include training backwards, since training forwards only is not sufficient to reach all configurations. Therefore, the Boltzmann Generations requires data from the Boltzmann distribution in order to train backwards as well. This data can be obtained from an experiment or a simulation, e.g. a Monte Carlo simulation or a confocal image.

The BGs have many potential applications. In this thesis we have predicted the interaction potential in a colloid-polymer mixture using backwards training, but there are many more possibilities. With a known potential and a few data samples, one could use a combination of forwards and backwards training to sample the distribution function of a many-body system. Therefore, they can be used to speed up Monte Carlo simulations. This would be interesting to explore in future research. In conclusion, when trained forwards and backwards the BGs are a strong machine learning method to sample Boltzmann distributions and they have many applications in various branches of physics.

5 Layman's Summary in Dutch

Deze samenvatting is bedoeld voor iedereen die graag iets meer wil weten over dit onderwerp, maar geen achtergrond in de natuurkunde heeft. Het is geschreven voor de familieleden en vrienden die altijd voorzichtig vroegen wat ik *precies* deed met mijn scriptie en met het antwoord "Ik doe iets met *machine learning* en natuurkunde" geen genoegen namen. Deze samenvatting is geschreven in het Nederlands om het zo toegankelijk mogelijk te maken voor deze mensen. We zullen in deze samenvatting eerst mogelijke toepassingen van dit onderzoek schetsen. Daarna wordt het algemene natuurkundige probleem toegelicht, om vervolgens het algemene idee van de "Boltzmann Generators" uit te leggen.

5.1 Toepassing

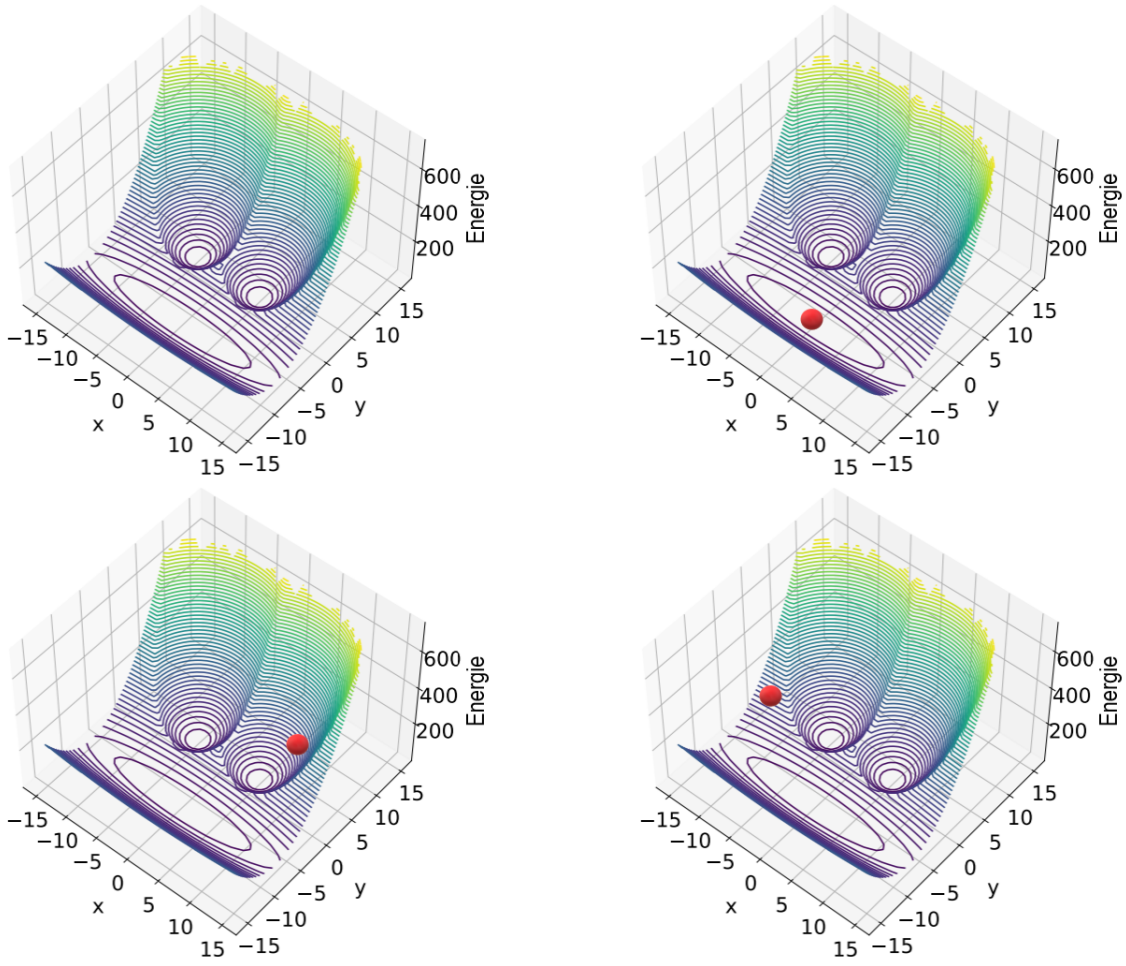
Voordat we echt de diepte induiken, zullen we eerst naar mogelijke toepassingen van dit onderzoek kijken. In deze scriptie wordt vooral gekeken wanneer deeltjes een lage energie hebben. Zo kan een eiwit (dat bestaat uit veel deeltjes) op talloze manieren worden opgevouwen. Echter, veel van deze manieren kosten veel energie, en zullen daarom niet of nauwelijks voorkomen in de natuur. Om te weten hoe een eiwit opgevouwen is en hoe het er dus waarschijnlijk uitziet, willen we weten welke vouwen de laagste energie hebben. Een concrete toepassing die erbij hoort is bijvoorbeeld de ontwikkeling van medicijnen: daarbij is het erg belangrijk welke vorm het eiwit precies heeft. Om achter deze vormen te komen, gebruiken natuurkundigen energielandschappen en Boltzmann verdelingen, die in de volgende paragraaf worden besproken.

5.2 Energielandschap

Bij de uitleg over energielandschappen wordt gebruik gemaakt van de afbeeldingen die in figuur 23 te zien zijn. We zullen het natuurkundige concept *energielandschap* uitleggen aan de hand van een voorbeeld. Linksboven zie je in figuur 23 - met een beetje fantasie - een smiley. Stel dat er een plakkerig balletje wordt gegooid in het berglandschap, dat gevormd wordt door deze smiley. Het balletje kan goed rollen, maar blijft een beetje kleven aan de randen. We gooien er nu honderd van deze balletjes in: waar zouden ze dan het vaakst terecht komen? De verwachting is dat het balletje het vaakst in een dal terecht komt: de ogen of de mond van de smiley (rechtsboven in de figuur). Echter, een paar balletjes zijn ook terecht gekomen op andere plekken, zo bleven er een paar kleven op de helling (links- en rechtsonder in de figuur). Echter, deze kans is veel kleiner. Oftwel, het balletje heeft de grootste kans om in een put te komen, de ogen of de mond dus. Hoe lager het landschap, des te groter de kans dat de bal zich hier bevindt. Nu kunnen we de connectie maken naar energieën: deze smiley is namelijk een voorbeeld van een energielandschap. Bij een energielandschap geeft de hoogte de hoeveelheid energie aan. Hoe lager de energie (dus bij een dal), des te groter de kans dat het balletje zich daar bevindt. Hoe hoger de energie (dus bij een top) hoe lager de kans is dat het balletje zich daar bevindt. Kortom: de laagste energieën hebben de grootste kans. Bij zo'n energielandschap hoort een zekere kansverdeling, de Boltzmann verdeling. Deze verdeling verklaart waarom hoge energieën een lage kans hebben en lage energieën een grote kans. Om die reden is het een erg belangrijke verdeling binnen de natuurkunde.

5.3 Configuratie

Nu zullen we het begrip *configuratie* bespreken. Een configuratie is in het voorbeeld de locatie van het balletje. In figuur 23 zie je dus verschillende configuraties van de bal. Merk hierbij op dat de configuraties niet allemaal even waarschijnlijk zijn. Zo zul je configuratie rechtsboven heel vaak tegen komen, maar moet je veel geduld en geluk hebben om rechtsonder te zien. In figuur 24 zien we rechts het resultaat als we 1000 balletjes in het landschap gooien: de rode puntjes zijn de plekken waar het balletje is terechtgekomen. Merk op dat de resultaten in figuur 24 anders zijn weergegeven dan in 23, in figuur 24 is de smiley van bovenaf weergegeven. De kleur geeft aan hoe vaak het balletje er is geweest: hoe donkerder, hoe vaker. Je ziet hier dat het balletje vrijwel alleen maar in de ogen of de mond komt, zoals we hadden voorspeld bij het landschap.



Figuur 23: Hier zie je vier keer hetzelfde energielandschap. In de afbeelding rechtsboven, linksonder en rechtsonder zijn ook mogelijke locaties van het kleverige balletje getekend. De kleuren geven de hoogte van het landschap aan.

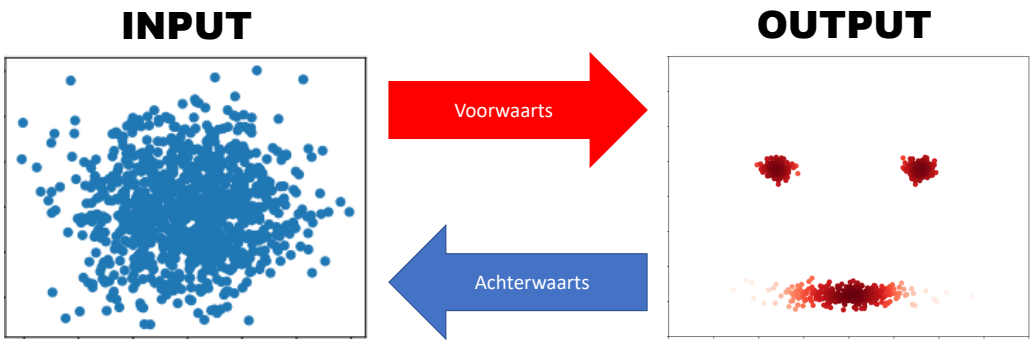
Het uiteindelijke doel is om te weten welke configuraties waarschijnlijk zijn, en dus waar het balletje zich het vaakst bevindt als we het experiment heel vaak herhalen. Het doel is dus de rechterkant van Fig. 24. Het grote probleem is dat de huidige methoden veel tijd en rekenkracht kosten: zo kan een berekening voor een eiwit wel een jaar duren op een supercomputer. In deze scriptie is dan ook een mogelijk efficiëntere methode onderzocht: de Boltzmann Generator. Deze methode is vrij recent, in 2019, geïntroduceerd door Noé et al. (2019) [11].

5.4 Boltzmann Generator

Het doel van de Boltzmann Generator is om te weten welke configuraties een hoge kans hebben, en dus vaak voorkomen. Dit wordt gedaan met behulp van neurale netwerken. Wanneer je aan neurale netwerken een input geeft, geven ze een output terug. Dit wordt weergegeven in Fig. 24. Aan de linkerkant van de figuur, stoppen we iets in het neurale netwerk (de input) Vervolgens gaat het neurale netwerken heel veel berekeningen doen en uiteindelijk geeft het een output: de smiley. Nu is ons doel om als output Fig. 24 te krijgen: dan weten we waar het balletje zich hoogstwaarschijnlijk bevindt. We gebruiken daarbij een speciale input, namelijk een normale kansverdeling. Deze wordt veel gebruikt in de wiskunde. Het neurale netwerk moet dan van deze input naar de output, het plaatje aan de rechterkant. De Boltzmann generator, bestaande uit neurale netwerken, vertaalt dus het linkerplaatje naar het rechter plaatje: hij weet van elk punt links, waar het terecht moet komen in het rechterplaatje. Een Boltzmann Generator is dus soort wiskundige vertaler.

Een handige eigenschap van deze netwerken is dat deze ook van de output naar de input kunnen gaan: dus andersom (achterwaarts op figuur 24). De meeste neurale netwerken kunnen dit niet, maar voor dit project hebben we een speciale structuur gebruikt waardoor dit wel mogelijk was. De reden hiervoor is dat de prestatie van de netwerken zo kon worden verbeterd.

Samengevat, een Boltzmann Generator kan dus te weten komen welke configuraties waarschijnlijk zijn, en dus vaak voorkomen. Hierbij kan je denken aan plakkerige balletjes, maar uiteindelijk ook aan ingewikkelde eiwitten. De Boltzmann Generator bestaat uit neurale netwerken gebruikt, die een bepaalde input vertalen naar een output. In deze scriptie hebben we deze Boltzmann Generator gemaakt en getest in verschillende contexten, waaronder deze smiley. De conclusie is dat ze tot goede resultaten leiden, ze slagen er aardig in om te laten zien waar de balletjes terechtkomen. Hopelijk zal er in de toekomst meer onderzoek hiernaar gedaan worden, om zo zelfs heel complexe systemen te onderzoeken.



Figuur 24: Dit is het doel van de Boltzman Generator: van de afbeelding links, naar de afbeelding rechts

6 Acknowledgements

During the work on this thesis a lot of people have helped me, and therefore, I would like to thank them. I would first like to thank my supervisor Laura Filion. Even though we have not met in person during this project: she gave me the guidance and the knowledge I needed to write this thesis. Moreover, she somehow manged to motivate me whilst being a little square on my a computer screen. Also, she took a lot of time to show me how to write a thesis in a structural, scientific way. It is up to the reader if I succeeded, but I definitely learned a lot from this and I will use these skils in the future.
I also would like to thank Rinske Alkemade, who always took the time for me to discuss my code and to help me with debugging. Also, she has a talent of making clear and beautiful figures (evidence: Fig. 16).
Special thanks to Frank Smallenburg, who provided his EDMD code and helped to make it work.
Furthermore, I must express my gratitude to everybody who joined the group meetings on Wednesdays: I learned a lot from your presentations. I also learned a lot about my drawing skills, at the occasional games night.

Finally, I would like to thank my friends and family. Even though most of them did not have the faintest idea what I was doing, they supported me nonetheless. Special thanks to my big brother Sander for proofreading my thesis, it has improved a lot from your comments. Last, but definitely not least, I would like to thank my boyfriend Sjoerd. Not only did he calm me down when I was feeling stressed, but he also helped me write this thesis. As of writing this, he is procrastinating his own deadline and proofreading my thesis. He helped me tremendously with checking on grammatical errors and rewriting weird sentences.

References

[1] M. Dijkstra and E. Luijten, Nature Materials **20**, 762 (2021).

[2] P. S. Clegg, Soft Matter **17**, 3991 (2021).

[3] C. Dietz, T. Kretz, and M. H. Thoma, Phys. Rev. E **96**, 011301 (2017).

[4] E. Boattini, M. Ram, F. Smallenburg, and L. Filion, Molecular Physics **116**, 3066 (2018).

[5] S. S. Schoenholz, E. D. Cubuk, D. M. Sussman, E. Kaxiras, and A. J. Liu, Nature Physics **12**, 469 (2016).

[6] V. Bapst, T. Keck, A. Grabska-Barwińska, C. Donner, E. D. Cubuk, S. S. Schoenholz, A. Obika, A. W. Nelson, T. Back, D. Hassabis, et al., Nature Physics **16**, 448 (2020).

[7] R. Jadrich, B. Lindquist, and T. Truskett, The Journal of chemical physics **149**, 194109 (2018).

[8] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007).

[9] E. Boattini, N. Bezem, S. N. Punnathanam, F. Smallenburg, and L. Filion, The Journal of Chemical Physics **153**, 064902 (2020).

[10] D. Frenkel and B. Smith, *Understanding molecular simulation* (Academic Press, Inc., 2001).

[11] F. Noé, S. Olsson, J. Köhler, and H. Wu, Science **365**, 6457 (2019).

[12] S. A. Hollingsworth and R. O. Dror, Neuron **99**, 1129 (2018).

[13] A. Laio and M. Parrinello, Proceedings of the National Academy of Sciences **99**, 12562 (2002).

[14] H. Grubmüller, Phys. Rev. E **52**, 2893 (1995).

[15] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H. Lin, *Learning from data*, vol. 4 (AMLBook, 2012).

[16] L. Dinh, D. Krueger, and Y. Bengio, *Nice: Non-linear independent components estimation* (2015), `arXiv:1410.8516`.

[17] D. Rezende and S. Mohamed, **37**, 1530 (2015).

[18] D. P. Kingma and P. Dhariwal, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), pp. 10236–10245.

[19] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, arXiv preprint arXiv:1912.02762 (2019).

[20] L. Dinh, J. Sohl-Dickstein, and S. Bengio, *Density estimation using real nvp* (2017), `arXiv:1605.08803`.

[21] S. Kullback and R. A. Leibler, The Annals of Mathematical Statistics **22**, 79 (1951).

[22] S. Asakura and F. Oosawa, Journal of Polymer Science **33**, 183 (1958).

[23] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation functions: Comparison of trends in practice and research for deep learning* (2018), `arXiv:1811.03378`.

[24] R. van Roij, L. Filion, and J. de Graaf, *Lecture notes: Advanced statistical physics* (2019).

Appendices

A Python Code

In this appendix we show the python code we used to set up the Boltzmann Generators. To make sure that the code is readable, we did not include the functions to sample from the Boltzmann distribution (*real_distribution*) and the function to calculate the energy (*potential*), since this differs per system. Also, we did not include the analysis (plotting the loss function and figures). Lastly, for writing this code we have used parts of the following code from GitHub: `https://github.com/senya-ashukha/real-nvp-pytorch/blob/master/real-nvp-pytorch.ipynb`

```
import numpy as np
import torch
from torch import nn
from torch import distributions
from torch.nn.parameter import Parameter
torch.set_default_dtype(torch.float64)

#set parameters
timesteps = 1000 #total timesteps of training
w_fw = 0.5 #weight forwards loss
w_bw = 0.5 #weight backwards loss
sample_size = 300 #size of the trainingdata per iteration

#here, we set up a flow of neural networks (NVP method)
class RealNVP(nn.Module):
    def __init__(self, nets, nett, mask, prior):
        super(RealNVP, self).__init__()

        self.prior = prior
        self.mask = nn.Parameter(mask, requires_grad=False)
        self.t = torch.nn.ModuleList([nett() for _ in range(len(masks))])
        self.s = torch.nn.ModuleList([nets() for _ in range(len(masks))])

    def g(self, z): #function from z to x
        log_det_zx, x = z.new_zeros(z.shape[0]), z

        for i in range(len(self.t)):
            x_ = x*self.mask[i] #we split it by the dimensions
            s = self.s[i](x_)*(1 - self.mask[i])
            t = self.t[i](x_)*(1 - self.mask[i])
            x = x_ + (1 - self.mask[i]) * (x * torch.exp(s) + t) #transformation
            log_det_zx += s.sum(dim=1) #determinant
        return x, log_det_zx

    def f(self, x):
        log_det_xz, z = x.new_zeros(x.shape[0]), x
        for i in reversed(range(len(self.t))):

            z_ = self.mask[i] * z #split z by its dimensions
            s = self.s[i](z_) * (1-self.mask[i])
            t = self.t[i](z_) * (1-self.mask[i])
```

```
            z = (1 - self.mask[i]) * (z - t) * torch.exp(-s) + z_
            log_det_xz-= s.sum(dim=1) #determinant
        return z, log_det_xz

    def log_prob_xz(self,x): #the log determinant
        z, logp_xz = self.f(x)
        return self.prior.log_prob(z) + logp_xz

    def log_prob_zx(self,z): #the log determinant
        x, logp_zx = self.g(z)
        energy = potential(x) #calculating the energy
        return -energy + logp_zx



#we combine the affine coupling layers in an alternating pattern:
masks = torch.from_numpy(np.array([[0, 1], [1, 0]] * 3).astype(np.float32))
#networks t and s:
nets = lambda: nn.Sequential(nn.Tanh(),nn.Linear(2, 256), nn.LeakyReLU(),\
                             nn.Linear(256,256), nn.LeakyReLU(), nn.Linear(256, 2))
nett = lambda: nn.Sequential(nn.Linear(2, 256), nn.LeakyReLU(), nn.Linear(256,256), \
                             nn.LeakyReLU(),nn.Linear(256, 2), nn.Tanh())


#the gaussian distribution
prior = distributions.MultivariateNormal(torch.zeros(2), torch.eye(2))
flow = RealNVP(nets, nett, masks, prior)
optimizer = torch.optim.Adam([p for p in flow.parameters() if p.requires_grad==True], lr=1e-4)


#training:
for t in range(timesteps):

    z = np.random.multivariate_normal(np.array([0,0]), np.eye(2), sample_size)
    loss_trainingforwards =  -flow.log_prob_zx(torch.from_numpy(z)).mean()
    x = real_distribution(sample_size)
    loss_trainingbackwards = -flow.log_prob_xz(torch.from_numpy(x)).mean()
    loss = w_fw * loss_trainingforwards + w_bw * loss_trainingbackwards
    optimizer.zero_grad()
    loss.backward(retain_graph=True)
    optimizer.step()
    if t % 100 == 0:
        print('iter %s:' % t, 'loss = %.3f' % loss, "loss_forwards = %.3f"\
              %loss_trainingforwards , "loss_backwards = %.3f" % loss_trainingbackwards)


#results:
x_real = real_distribution(1000)
z_BG = flow.f(torch.from_numpy(real_distribution(x_real)))[0].detach().numpy()
z_real = np.random.multivariate_normal(np.zeros(2), np.eye(2), 1000)
x_BG = flow.g(torch.from_numpy(z_real))[0].detach().numpy()
```

B Results training Method III

In this appendix we present the results of training Method III, except the results that are in included in Chapter 3. We show two different figures for every setting. The first figure has the same setup for each setting: there are four different density plots. In the upper left corner, a Gaussian $z \sim P_z$ is plotted. The plot in the upper right corner shows the result of $F_{zx}(z)$. In the lower left corner, a configuration drawn from the smiley face distribution (P_x) is shown. In the lower right corner we show the outcome of $F_{xz}(x)$. The second figure shows the results of the same Boltzmann Generator, but in this figure one can see how the smiley is transformed. Note that the colors of the left figure correspond with the same colors of the right figure. In the caption of every figure the values of the forwards loss and backwards loss are mentioned. L_{fw} refers to the loss from Eq. 2.10 and L_{bw} to Eq. 2.11.

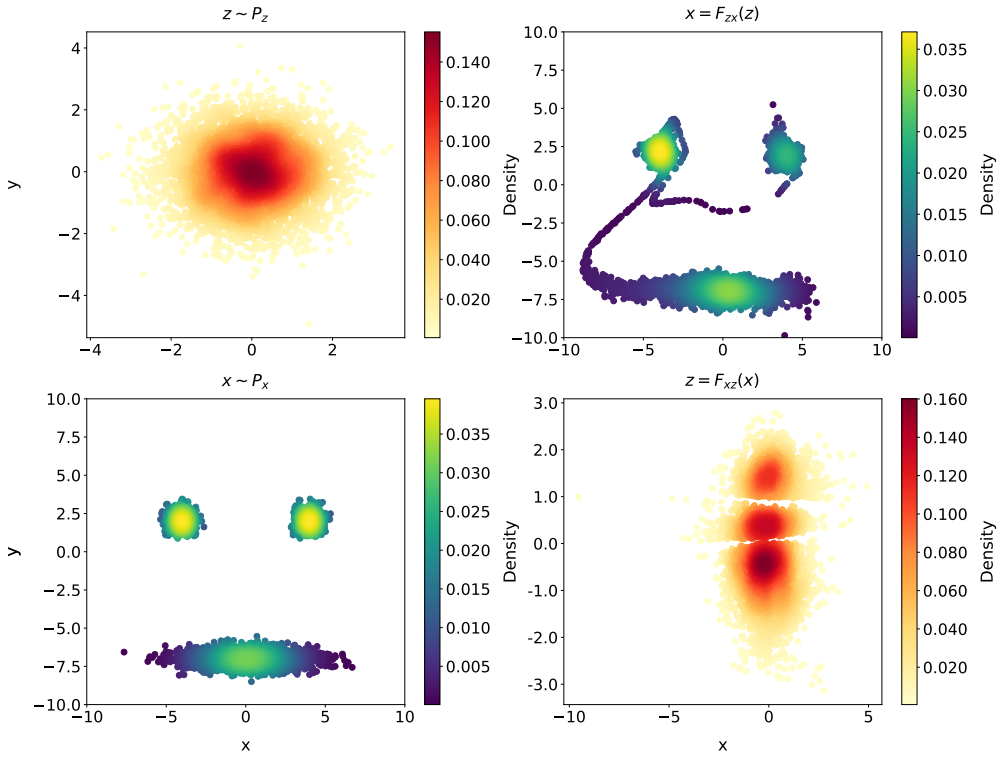


Figure 25: This figure shows the results of Method III, $w_{fw} = 0.1$, $w_{bw} = 0.9$. $L_{fw} = 2.4$, $L_{bw} = 3.1$.

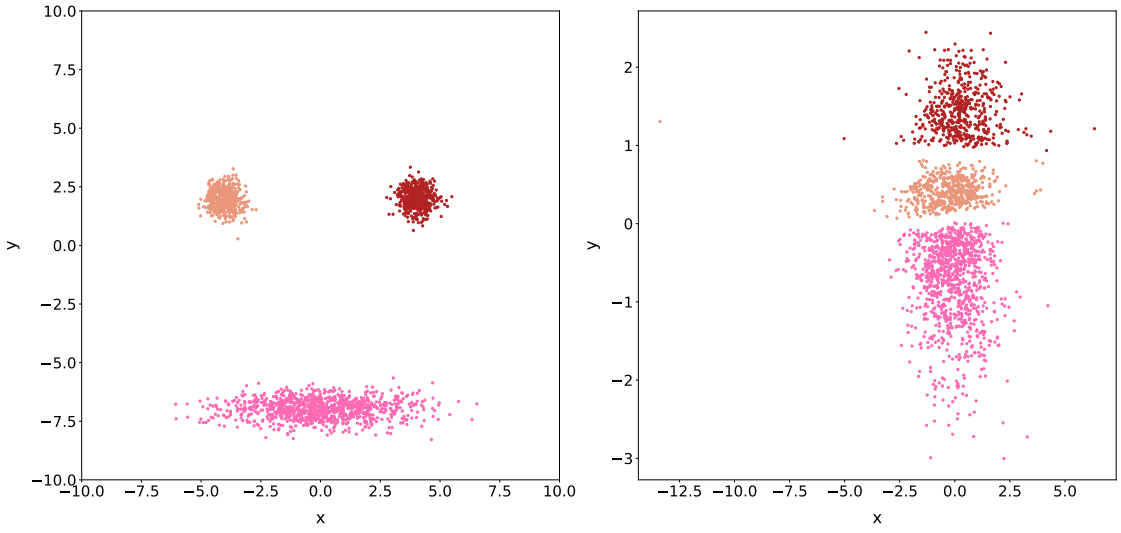


Figure 26: This figure shows the results of Method III, $w_{fw} = 0.1$, $w_{bw} = 0.9$. $L_{fw} = 2.4$, $L_{bw} = 3.1$.

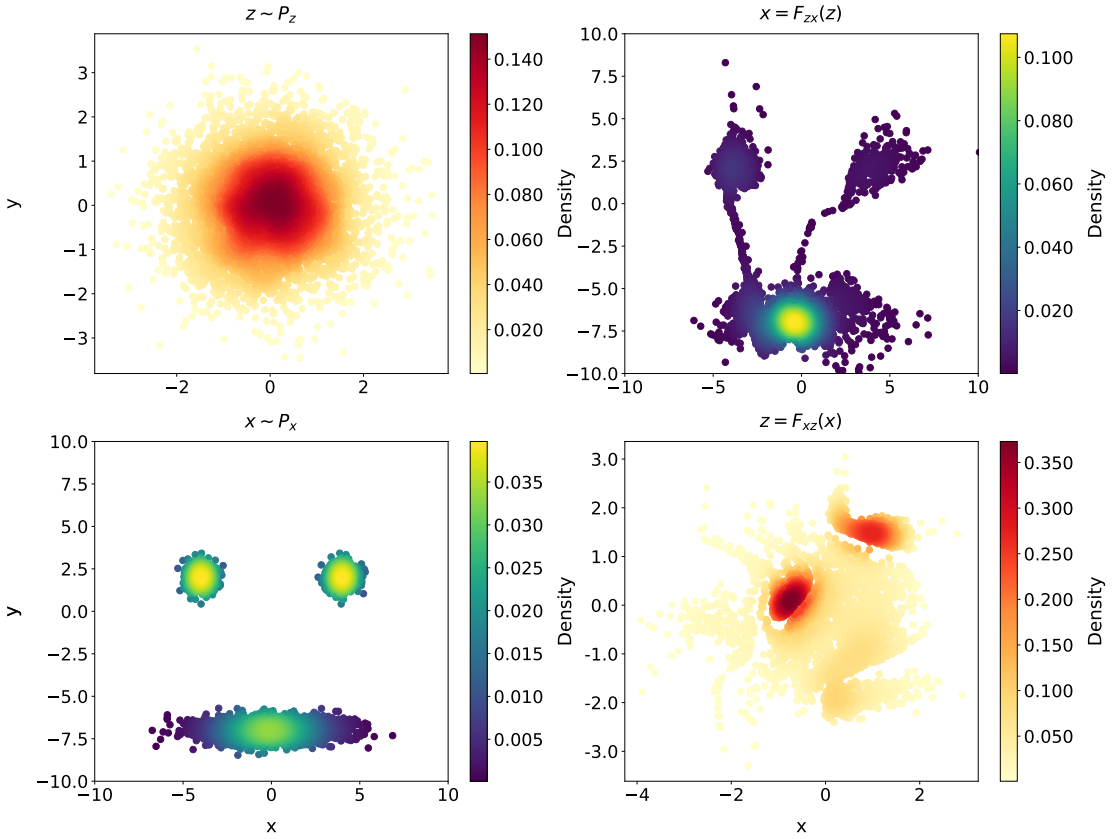


Figure 27: This figure shows the results of Method III, $w_{fw} = 0.2$, $w_{bw} = 0.8$. $L_{fw} = 3.1$, $L_{bw} = 3.9$.

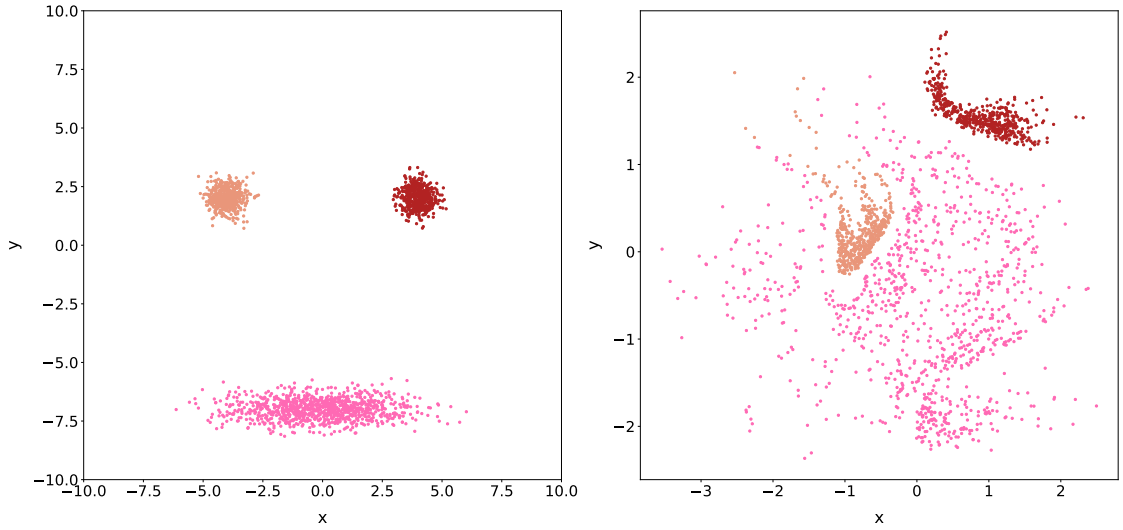


Figure 28: This figure shows the results of Method III, $w_{fw} = 0.2$, $w_{bw} = 0.8$. $L_{fw} = 3.1$, $L_{bw} = 3.9$.

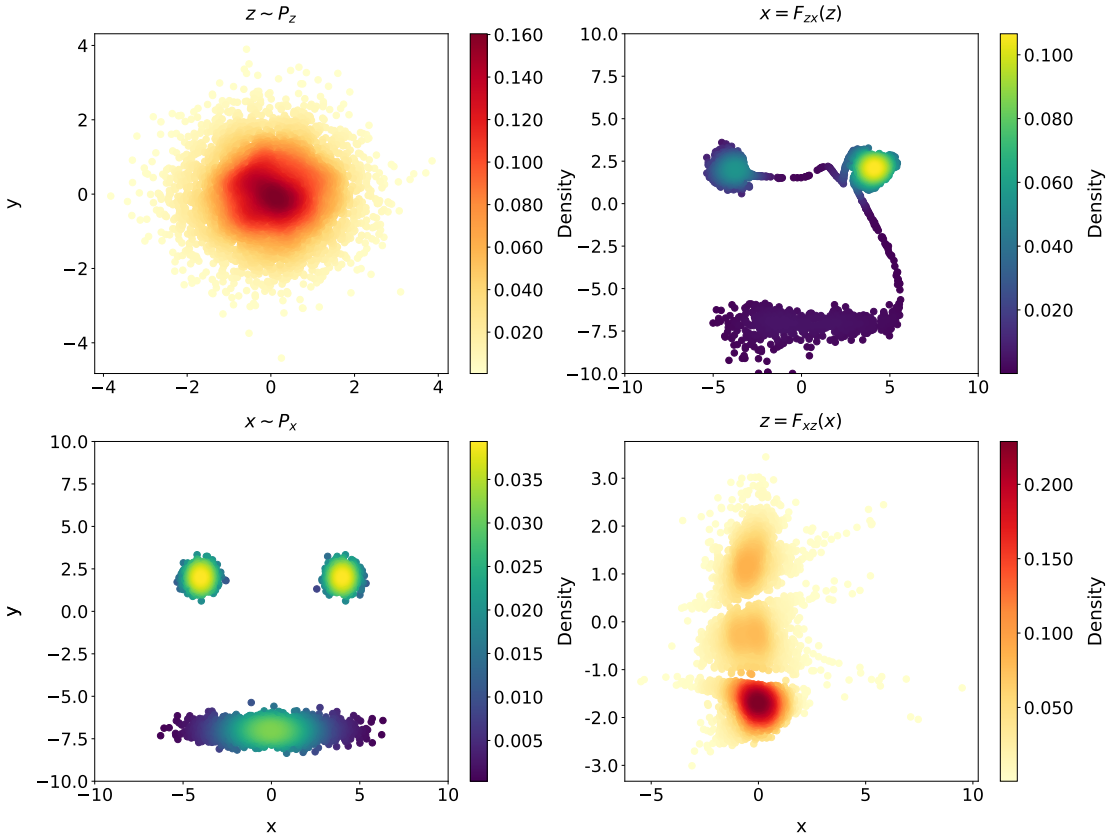


Figure 29: This figure shows the results of Method III, $w_{fw} = 0.3$, $w_{bw} = 0.7$. $L_{fw} = 2.9$, $L_{bw} = 3.8$.

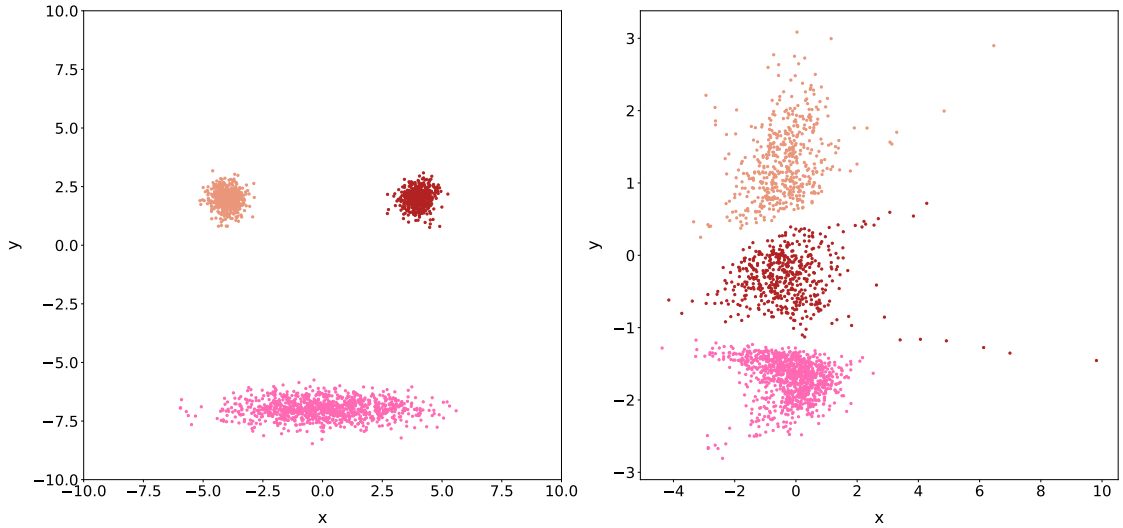


Figure 30: This figure shows the results of Method III, $w_{fw} = 0.3$, $w_{bw} = 0.7$. $L_{fw} = 2.9$, $L_{bw} = 3.8$.

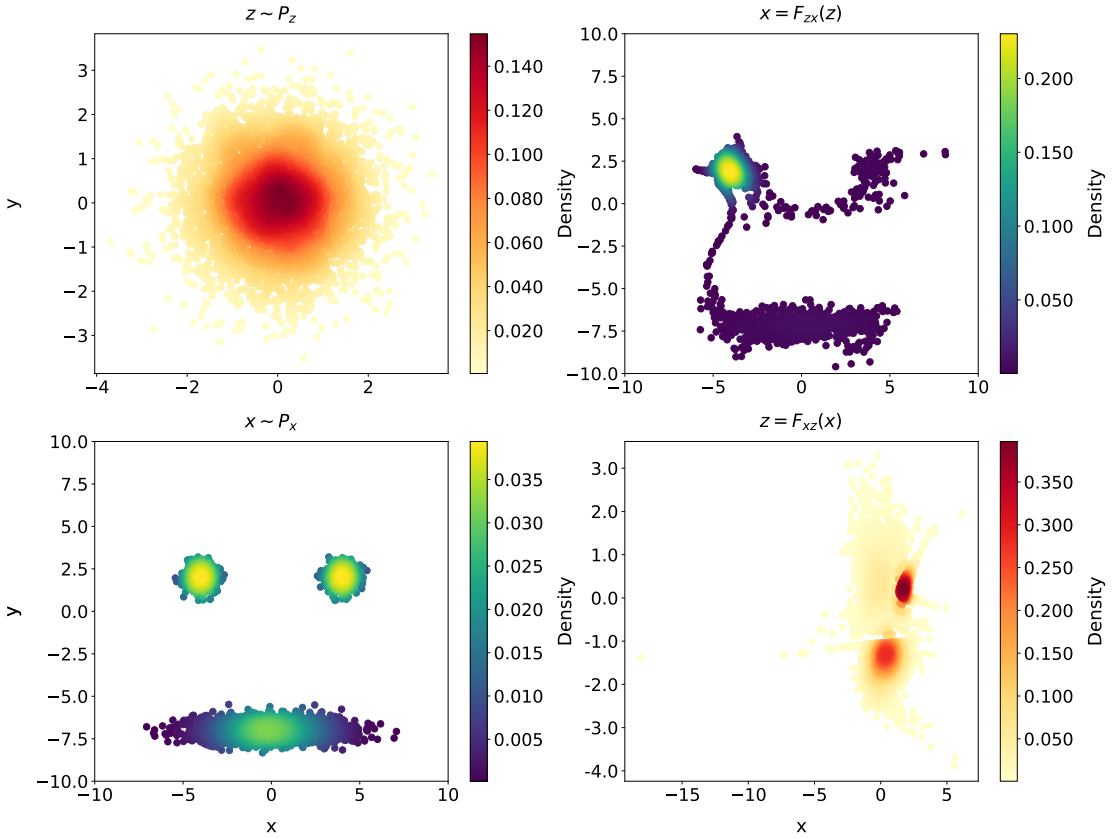


Figure 31: This figure shows the results of Method III, $w_{fw} = 0.4$, $w_{bw} = 0.6$. $L_{fw} = 2.9$, $L_{bw} = 4.1$.

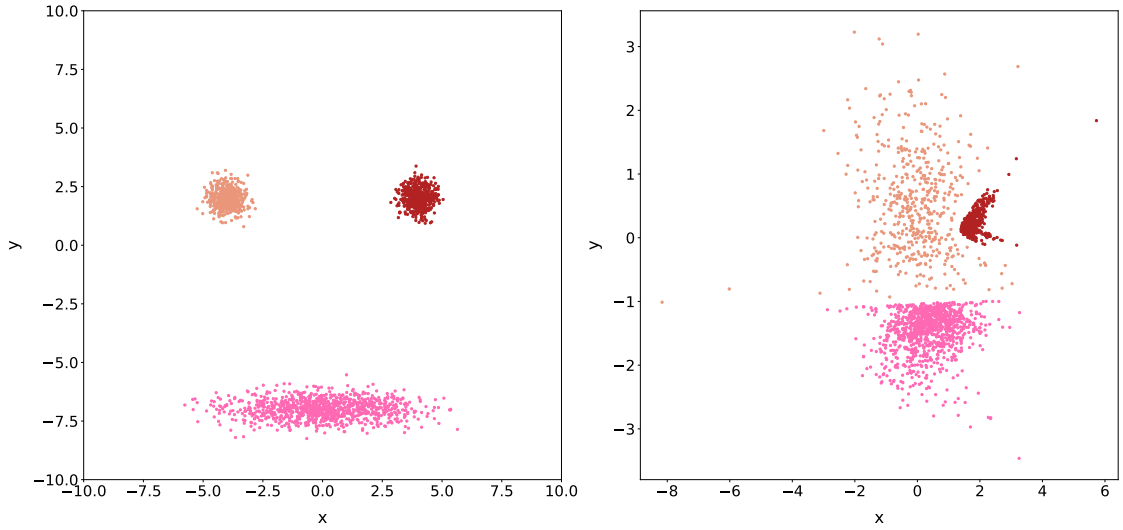


Figure 32: This figure shows the results of Method III, $w_{fw} = 0.4$, $w_{bw} = 0.6$. $L_{fw} = 2.9$, $L_{bw} = 4.1$.

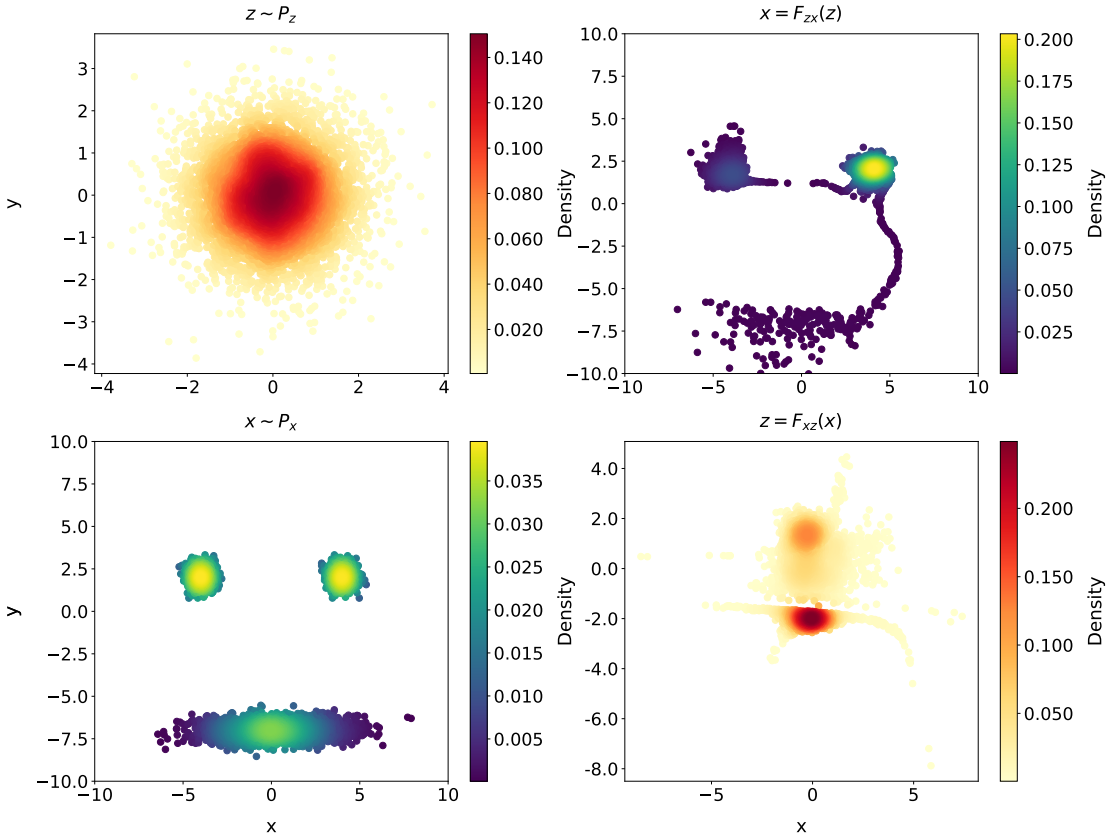


Figure 33: This figure shows the results of Method III, $w_{fw} = 0.5$, $w_{bw} = 0.5$. $L_{fw} = 2.9$, $L_{bw} = 4.3$.

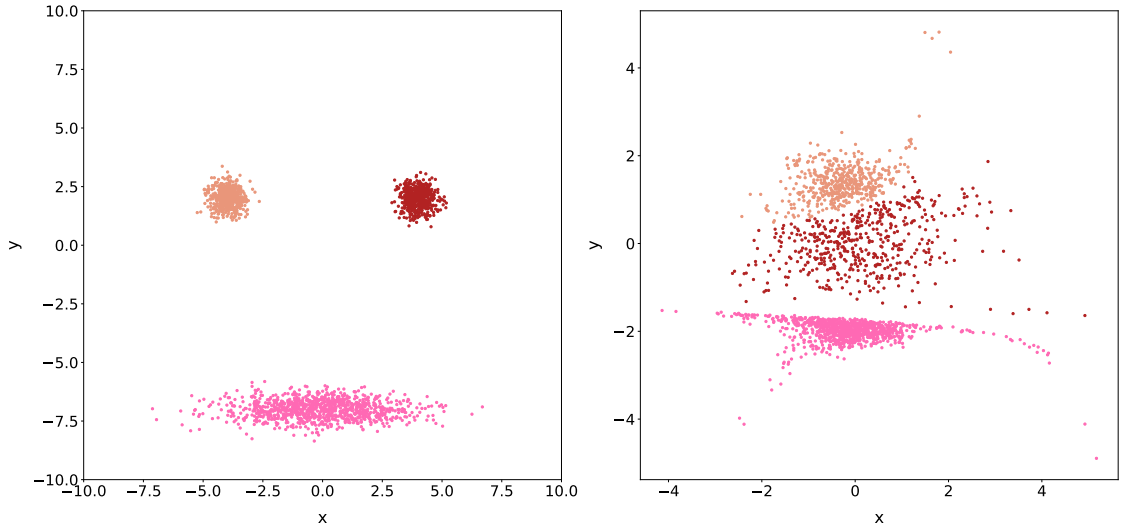


Figure 34: This figure shows the results of Method III, $w_{fw} = 0.5$, $w_{bw} = 0.5$. $L_{fw} = 2.9$, $L_{bw} = 4.3$.

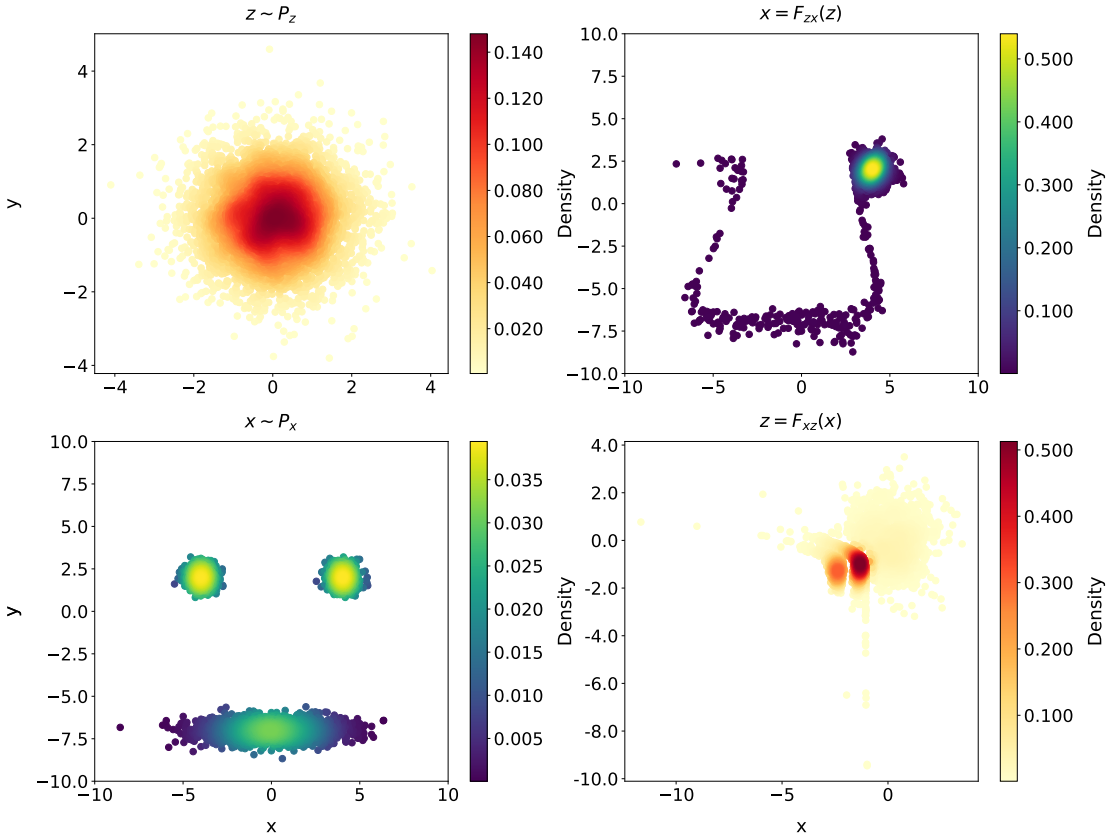


Figure 35: This figure shows the results of Method III, $w_{fw} = 0.6$, $w_{bw} = 0.4$. $L_{fw} = 3.0$, $L_{bw} = 5.2$.

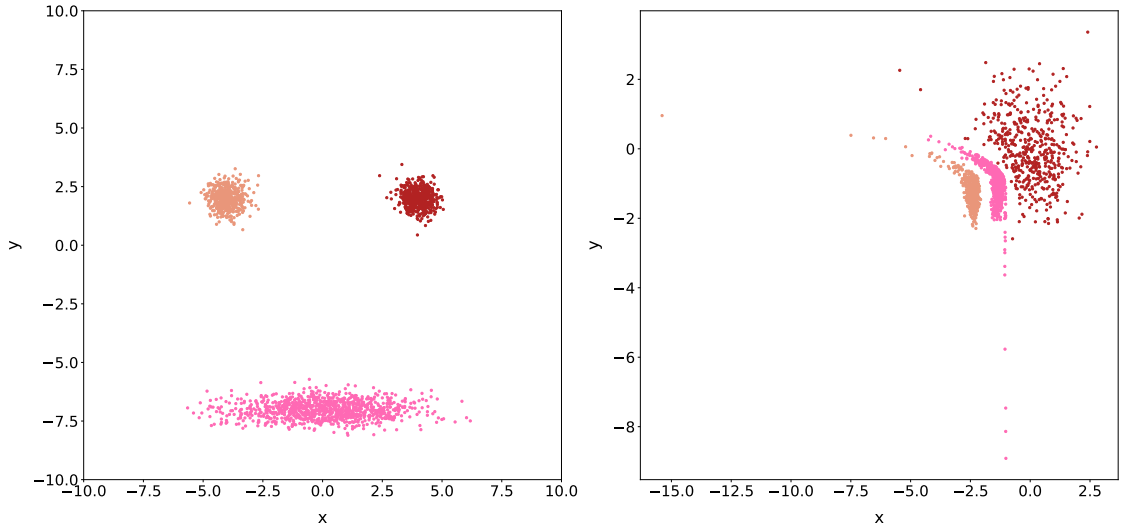


Figure 36: This figure shows the results of Method III, $w_{fw} = 0.6$, $w_{bw} = 0.4$. $L_{fw} = 3.0$, $L_{bw} = 5.2$

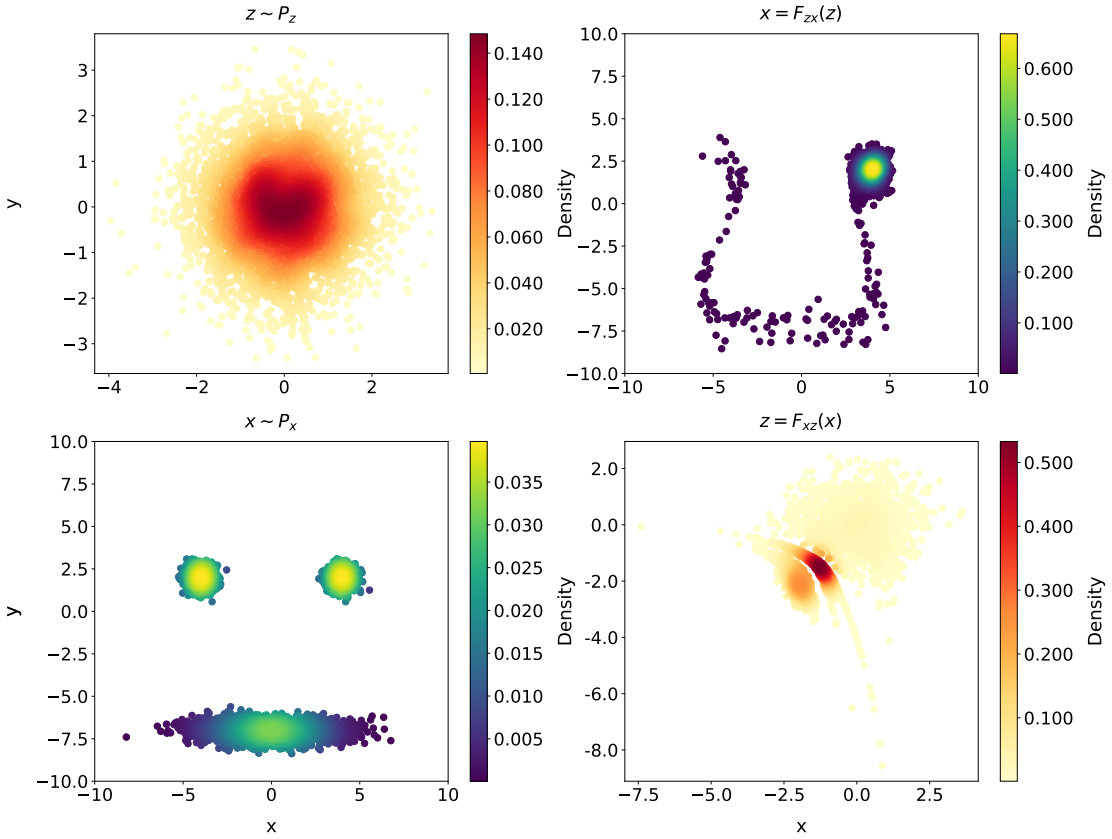


Figure 37: This figure shows the results of Method III, $w_{fw} = 0.7$, $w_{bw} = 0.3$. $L_{fw} = 3.0$, $L_{bw} = 5.7$

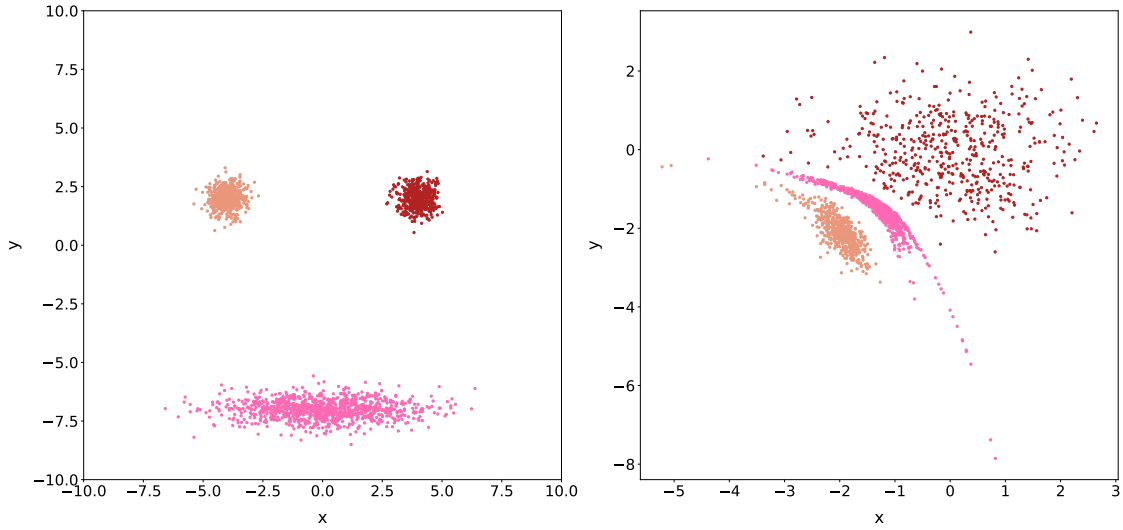


Figure 38: This figure shows the results of Method III, $w_{fw} = 0.7$, $w_{bw} = 0.3$. $L_{fw} = 3.0$, $L_{bw} = 5.7$

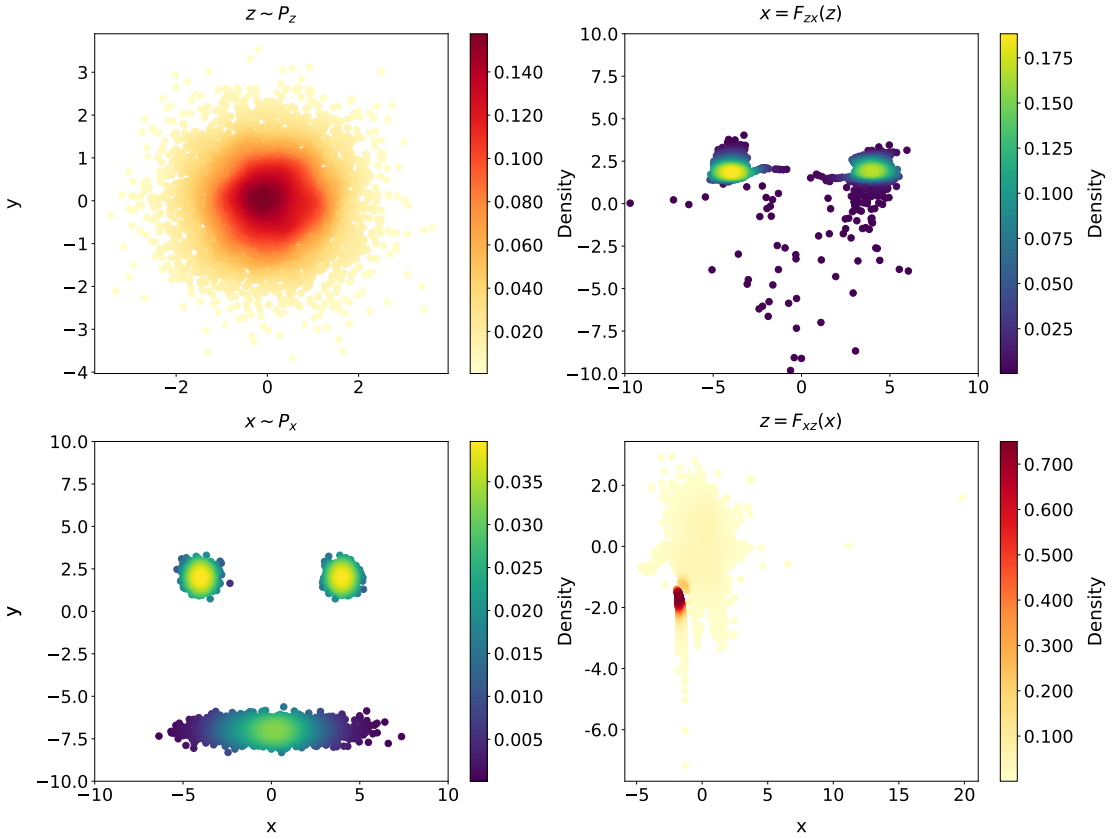


Figure 39: This figure shows the results of Method III, $w_{fw} = 0.8$, $w_{bw} = 0.2$. $L_{fw} = 2.8$, $L_{bw} = 6.1$

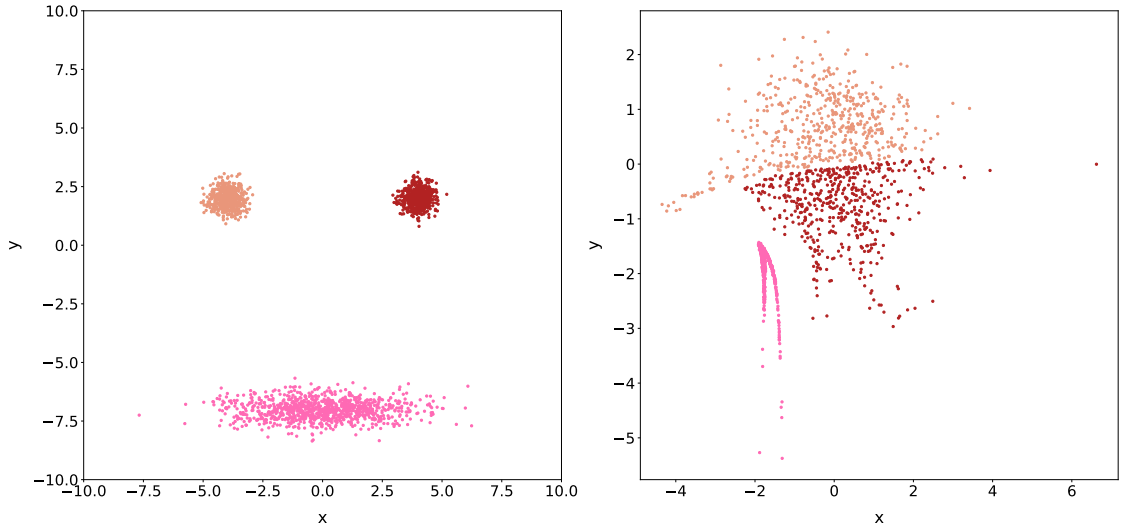


Figure 40: This figure shows the results of Method III, $w_{fw} = 0.8$, $w_{bw} = 0.2$. $L_{fw} = 2.8$, $L_{bw} = 6.1$

C Results training Method IV

In this appendix we present the results of training Method IV, except the results that are in included in Chapter 3. Again, we show two figures for every different training setting. The first figure we show consists of four density plots: in the upper left corner, a configuration from a Gaussian distribution is plotted. If we use this as an input for $F_{zx}(z)$, we obtain the figure on the upper right corner. In the lower left corner, a configuration from the smiley distribution is plotted. When we use the backwards transformation ($F_{xz}(x)$) on this configuration, we obtain the figure on the lower right corner. The second figure shows how the Boltzmann generator transforms the smiley distribution: note that each color in the left figure corresponds with the same color in the right figure. In every caption, we mention the loss functions, where L_{fw} refers to the loss from Eq. 2.10 and L_{bw} to Eq. 2.11.

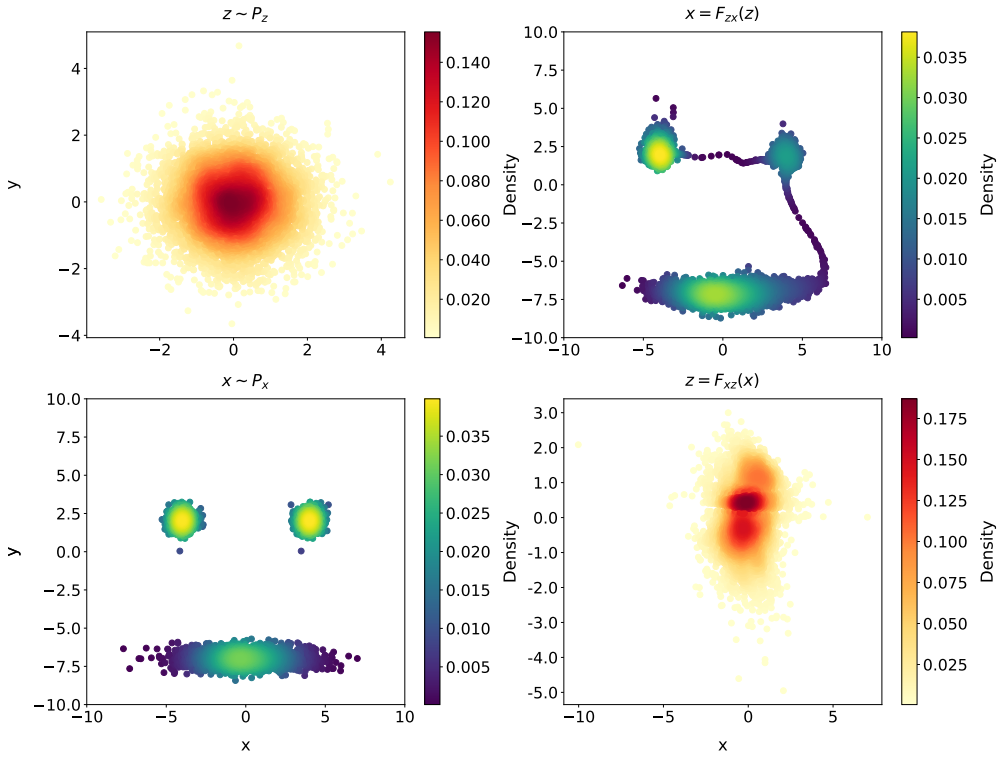


Figure 41: This figure shows the results of Method IV, $w_{fw} = 0.1$, $w_{bw} = 0.9$. $L_{fw} = 3.8$, $L_{bw} = 3.1$.

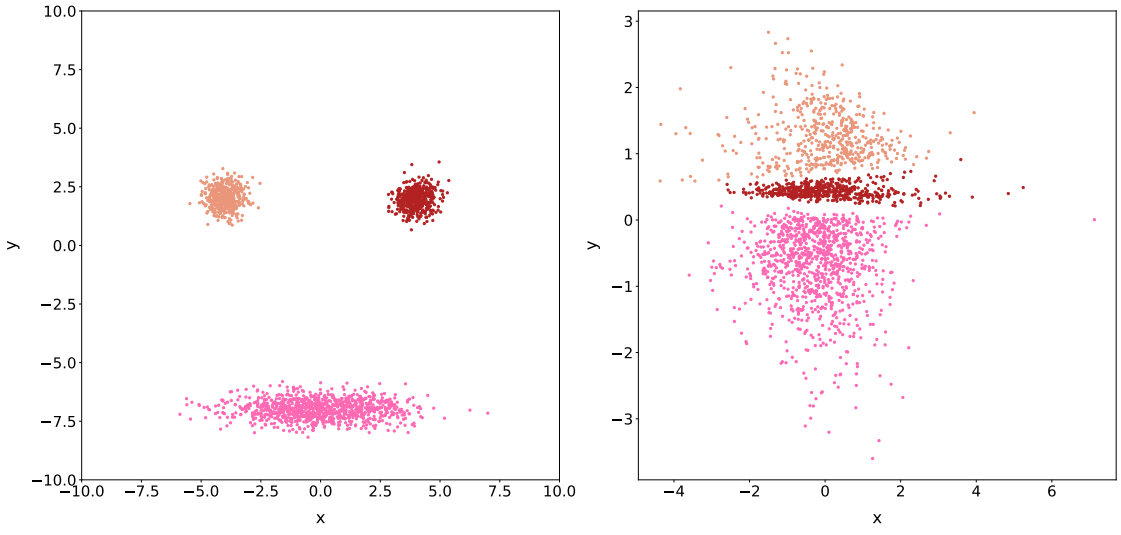


Figure 42: This figure shows the results of Method IV, $w_{fw} = 0.1$, $w_{bw} = 0.9$. $L_{fw} = 3.8$, $L_{bw} = 3.1$.

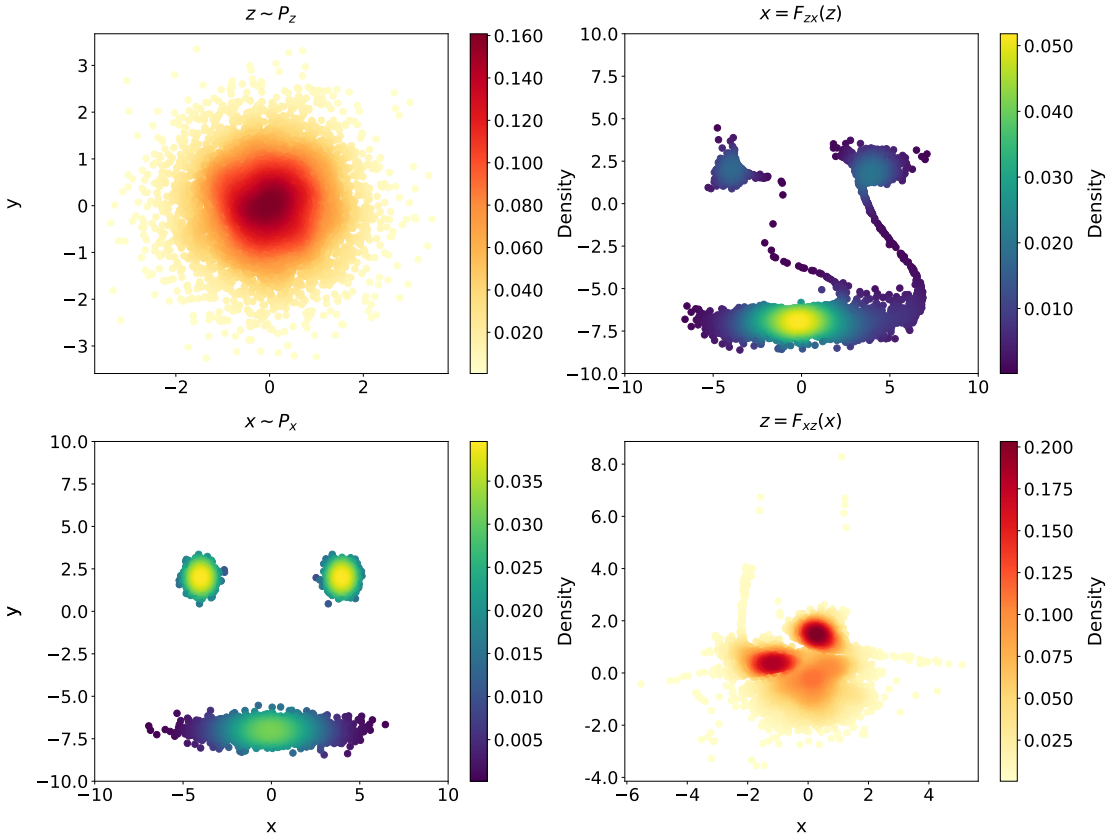


Figure 43: This figure shows the results of Method IV, $w_{fw} = 0.2$, $w_{bw} = 0.8$. $L_{fw} = 2.1$, $L_{bw} = 3.4$.

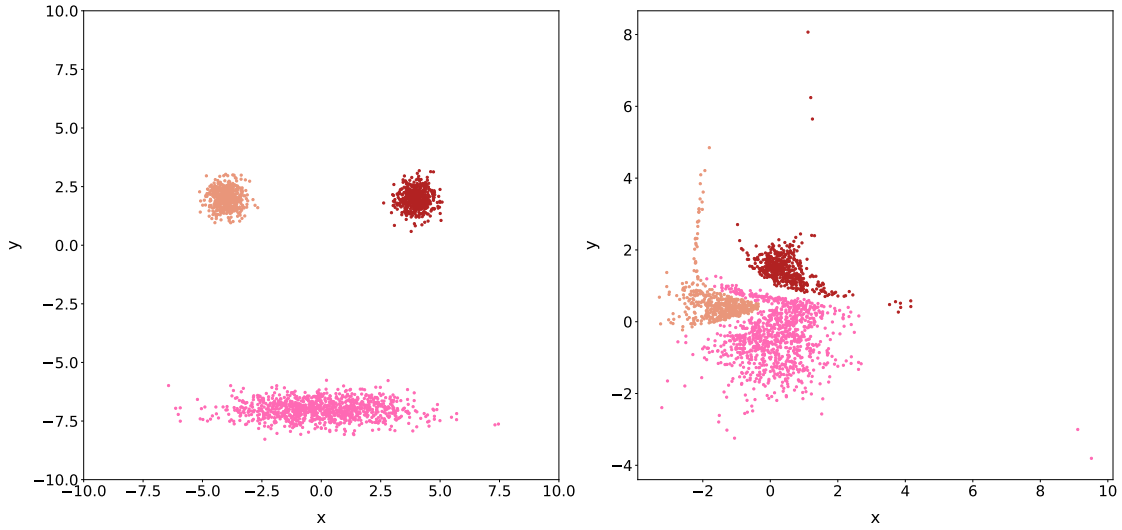


Figure 44: This figure shows the results of Method IV, $w_{fw} = 0.2$, $w_{bw} = 0.8$. $L_{fw} = 2.1$, $L_{bw} = 3.4$.

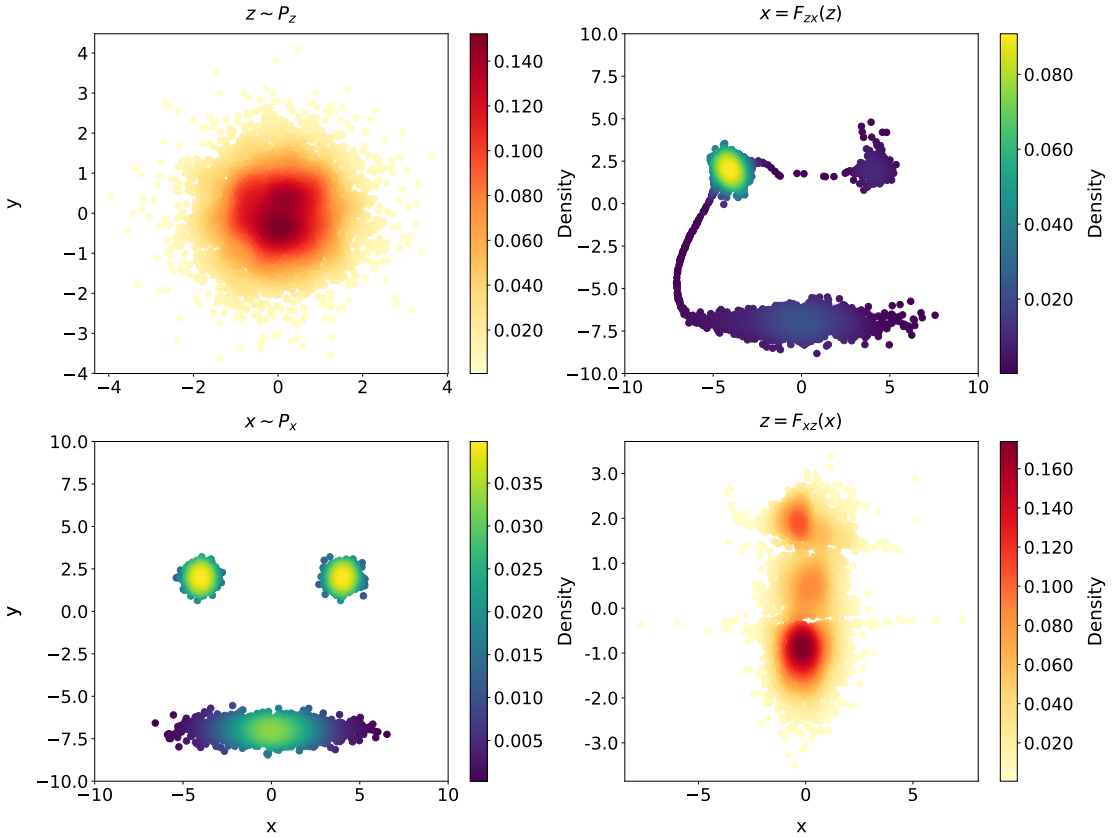


Figure 45: This figure shows the results of Method IV, $w_{fw} = 0.3$, $w_{bw} = 0.7$. $L_{fw} = 2.1$, $L_{bw} = 3.3$.

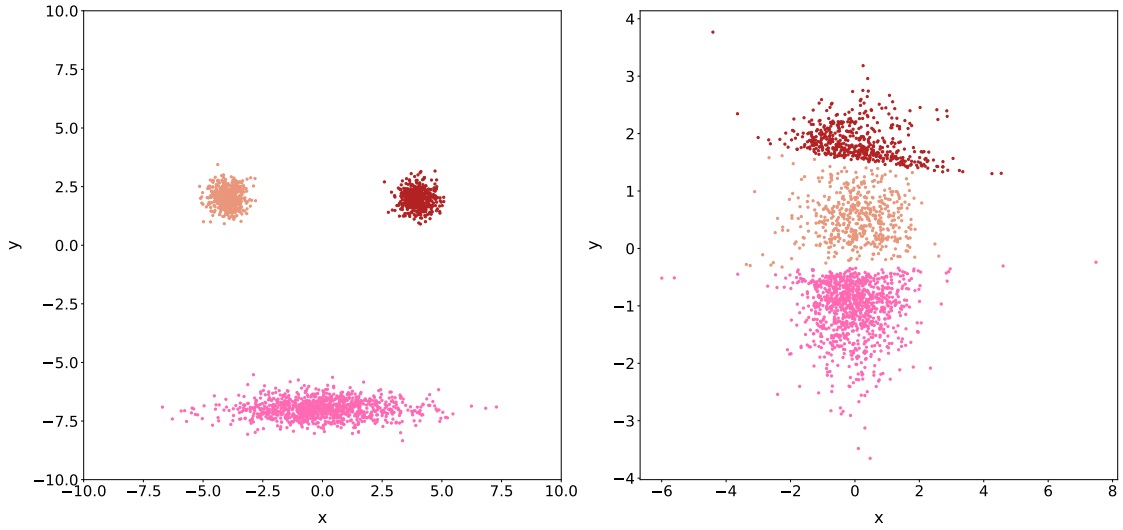


Figure 46: This figure shows the results of Method IV, $w_{fw} = 0.3$, $w_{bw} = 0.7$. $L_{fw} = 2.1$, $L_{bw} = 3.3$.

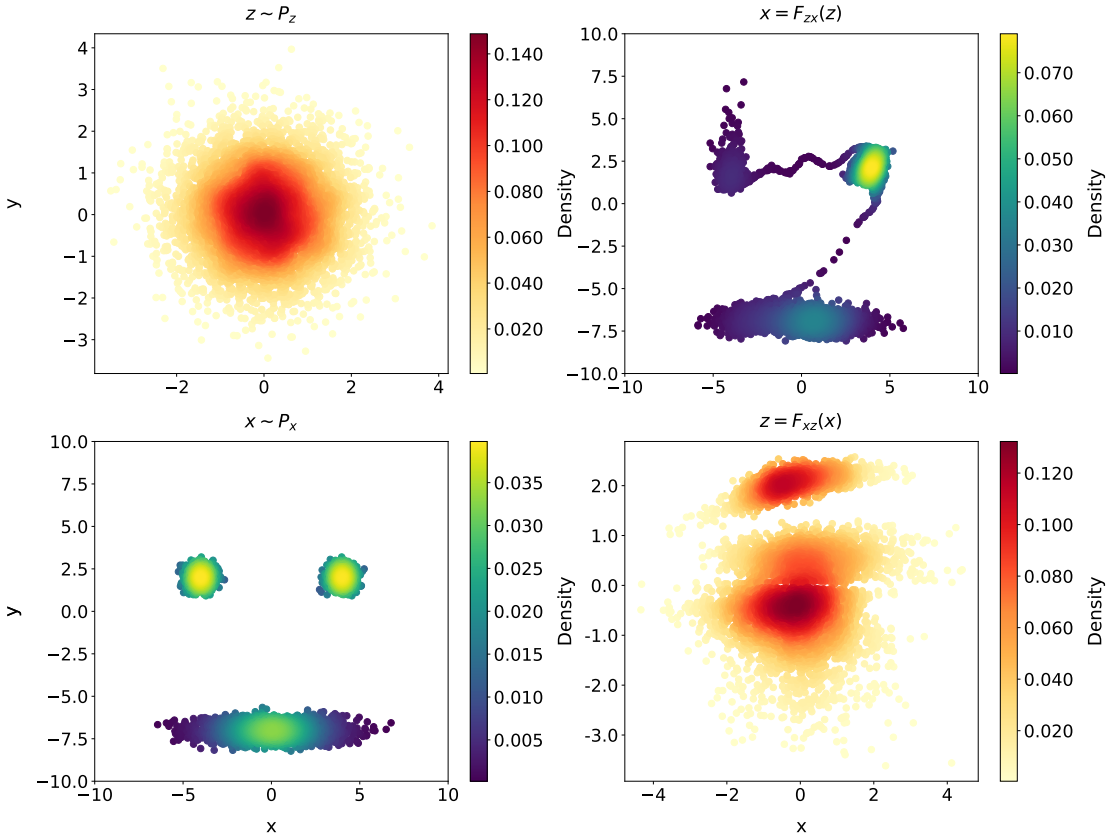


Figure 47: This figure shows the results of Method IV, $w_{fw} = 0.4$, $w_{bw} = 0.6$. $L_{fw} = 1.9$, $L_{bw} = 3.4$.

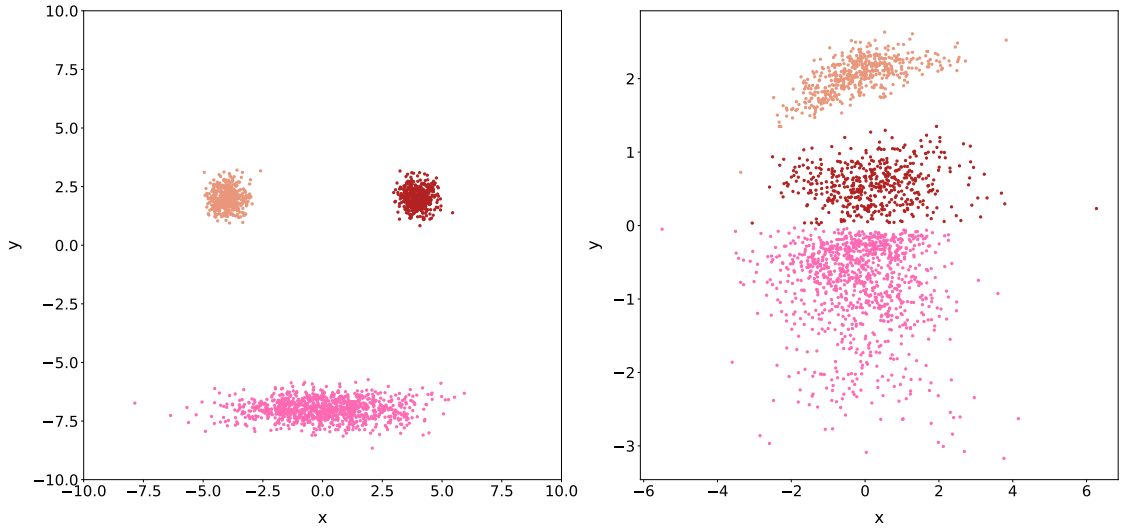


Figure 48: This figure shows the results of Method IV, $w_{fw} = 0.4$, $w_{bw} = 0.6$. $L_{fw} = 1.9$, $L_{bw} = 3.4$.

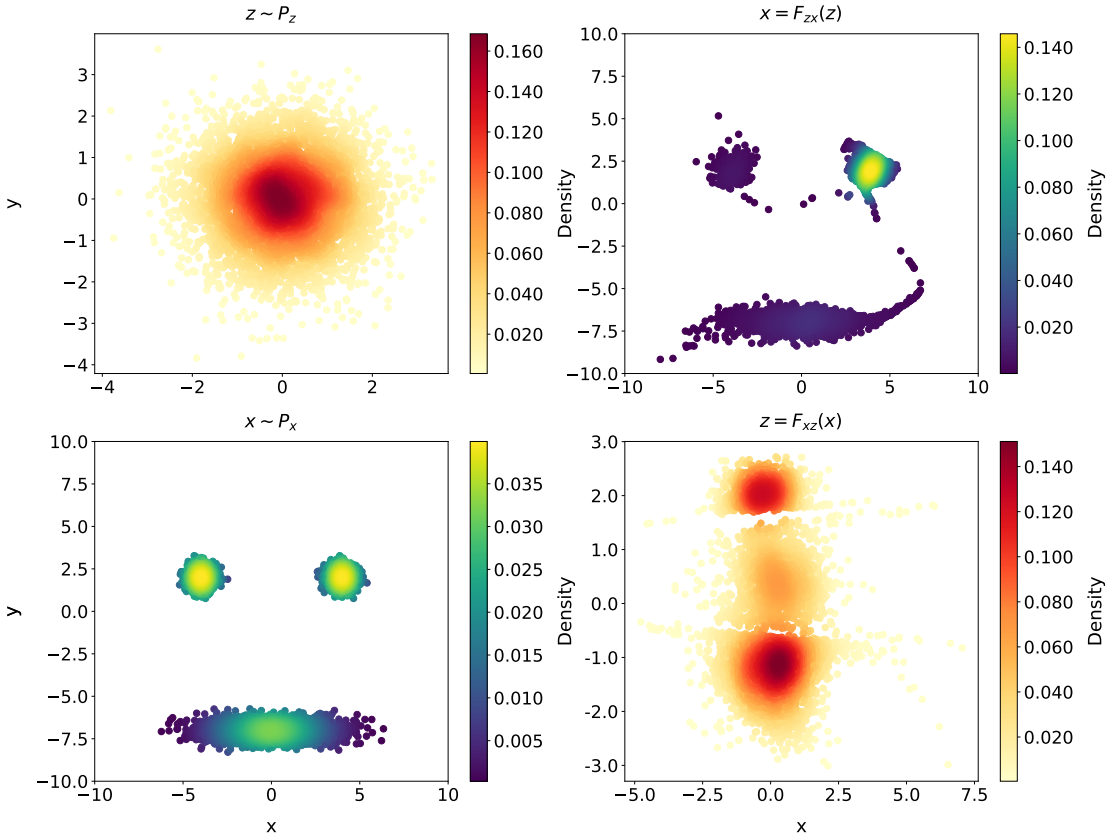


Figure 49: This figure shows the results of Method IV, $w_{fw} = 0.5$, $w_{bw} = 0.5$. $L_{fw} = 2.1$, $L_{bw} = 3.5$.

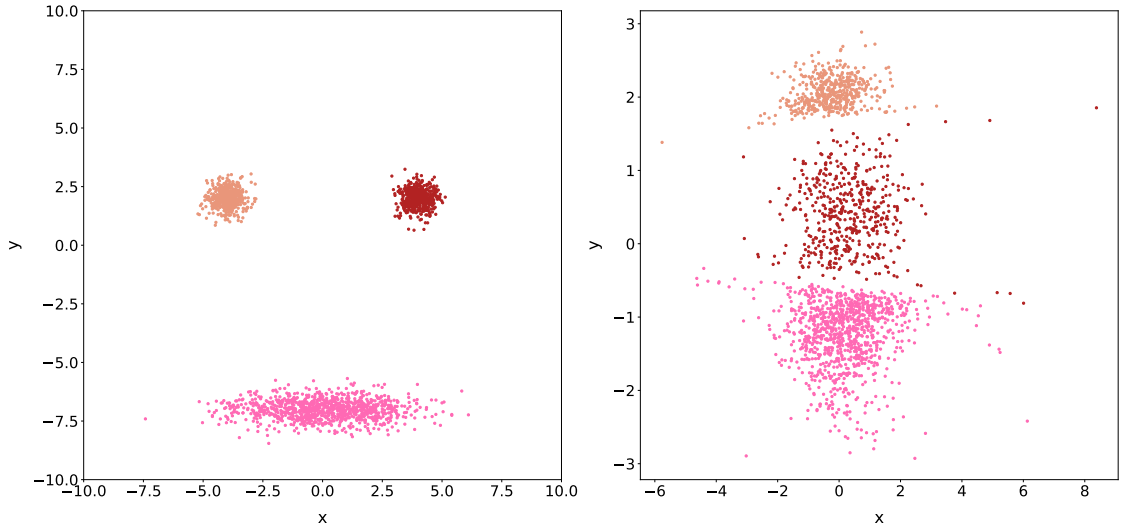


Figure 50: This figure shows the results of Method IV, $w_{fw} = 0.5$, $w_{bw} = 0.5$. $L_{fw} = 2.1$, $L_{bw} = 3.5$.

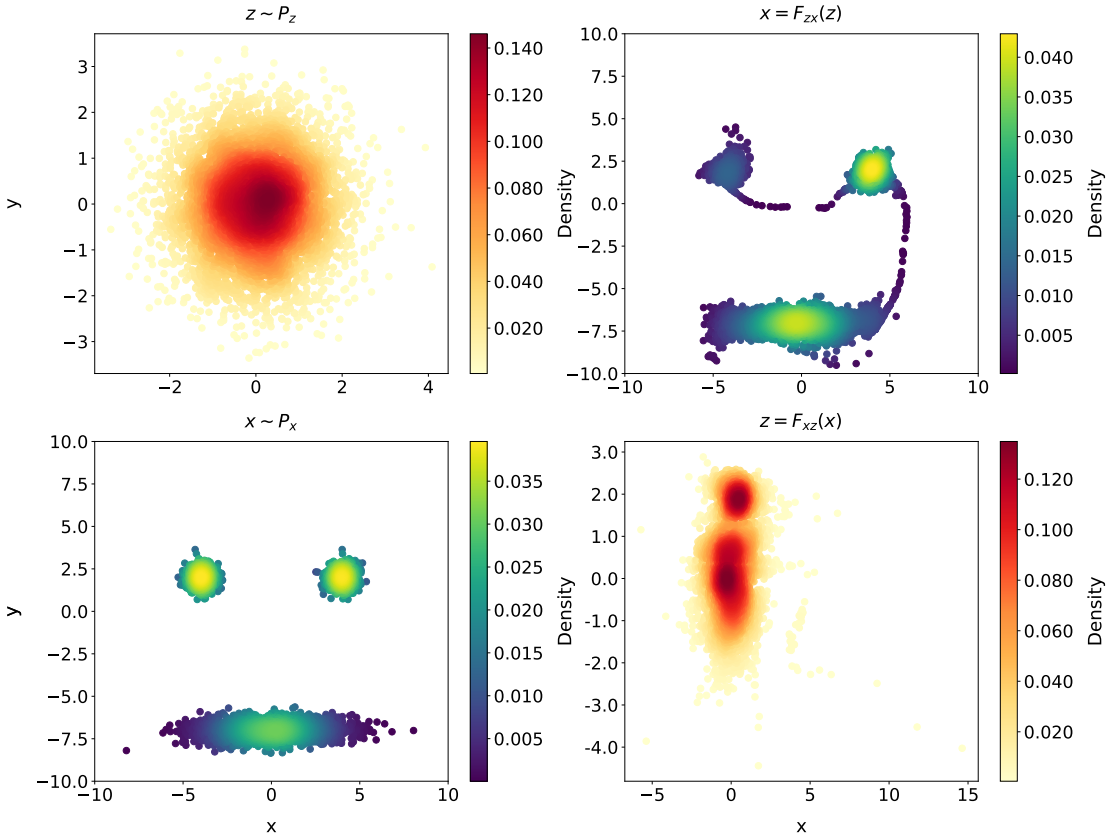


Figure 51: This figure shows the results of Method IV, $w_{fw} = 0.6$, $w_{bw} = 0.4$. $L_{fw} = 1.5$, $L_{bw} = 3.3$.

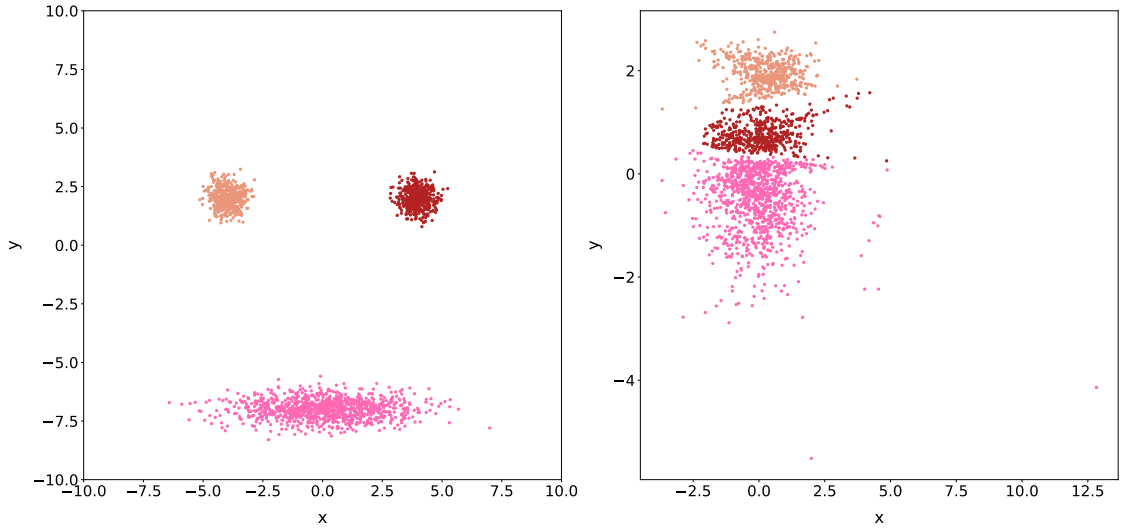


Figure 52: This figure shows the results of Method IV, $w_{fw} = 0.6$, $w_{bw} = 0.4$. $L_{fw} = 1.5$, $L_{bw} = 3.3$.

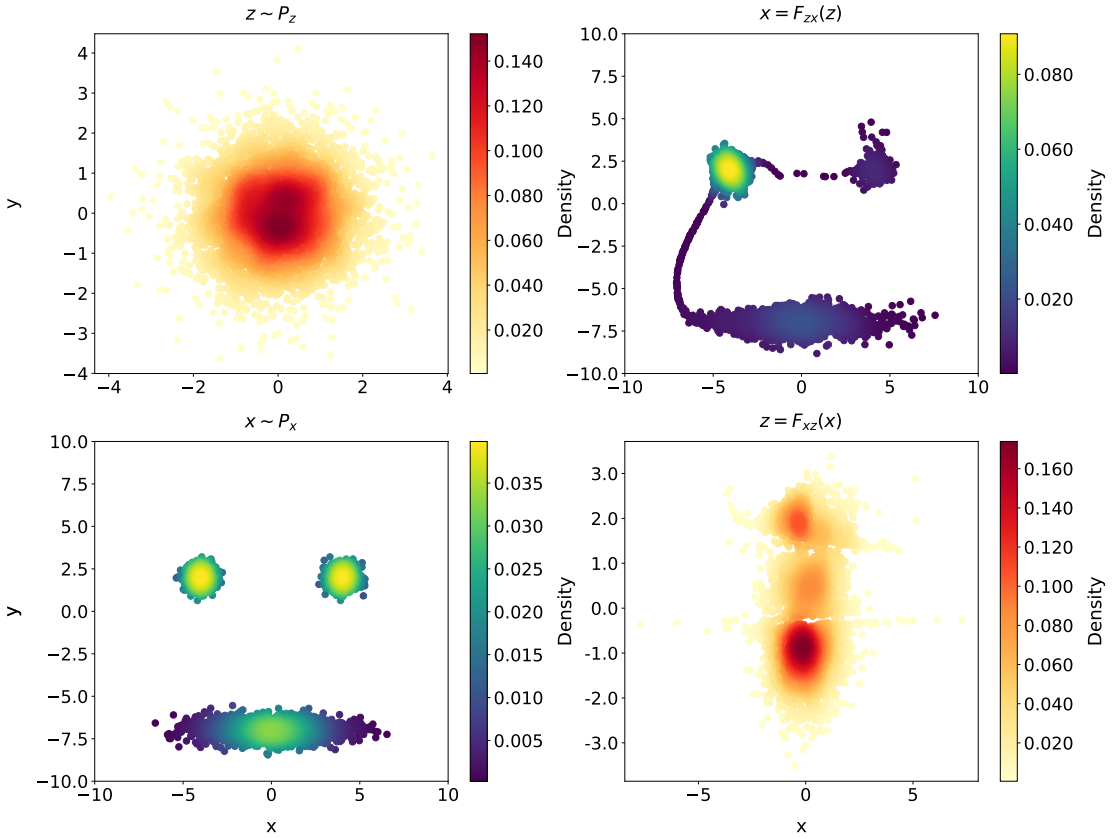


Figure 53: This figure shows the results of Method IV, $w_{fw} = 0.7$, $w_{bw} = 0.3$. $L_{fw} = 1.5$, $L_{bw} = 3.3$.

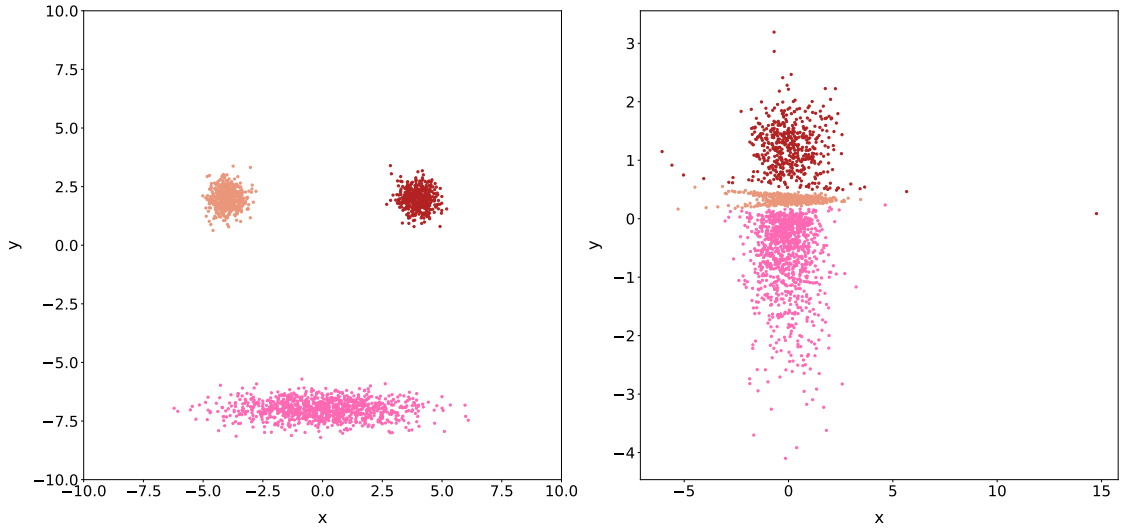


Figure 54: This figure shows the results of Method IV, $w_{fw} = 0.7$, $w_{bw} = 0.3$. $L_{fw} = 1.5$, $L_{bw} = 3.3$

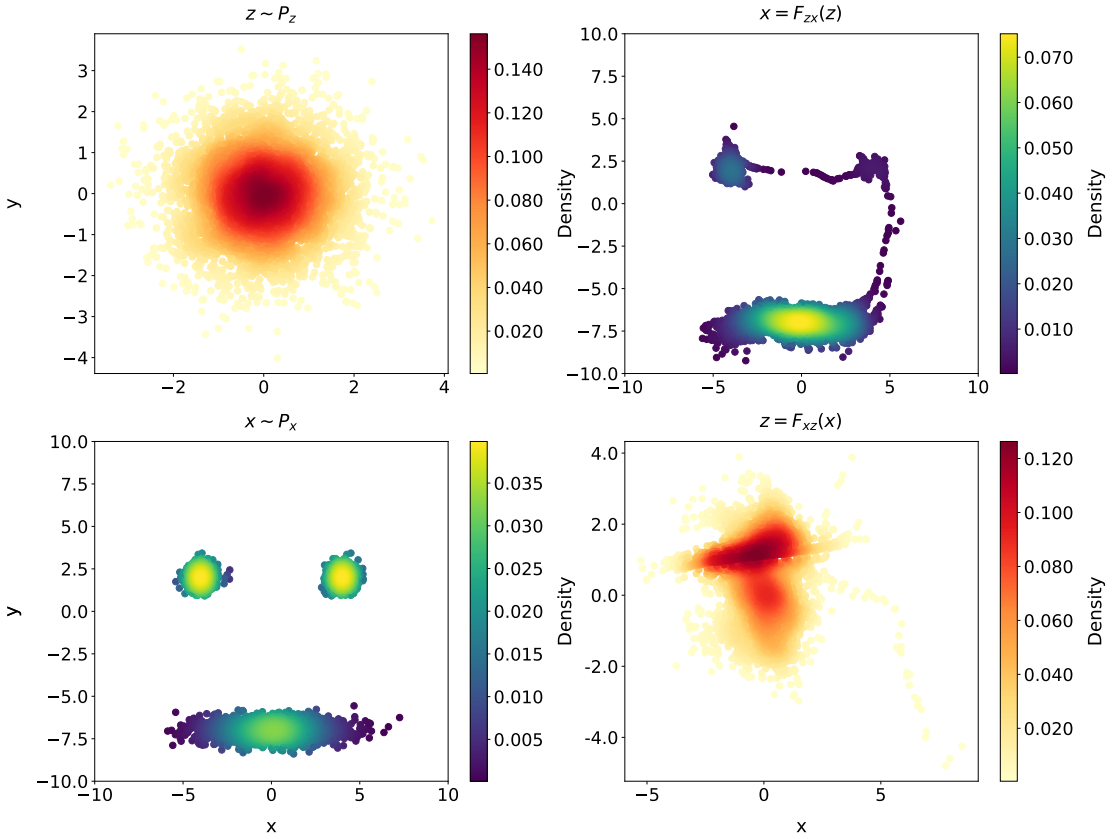


Figure 55: This figure shows the results of Method IV, $w_{fw} = 0.8$, $w_{bw} = 0.2$. $L_{fw} = 1.3$, $L_{bw} = 3.8$.

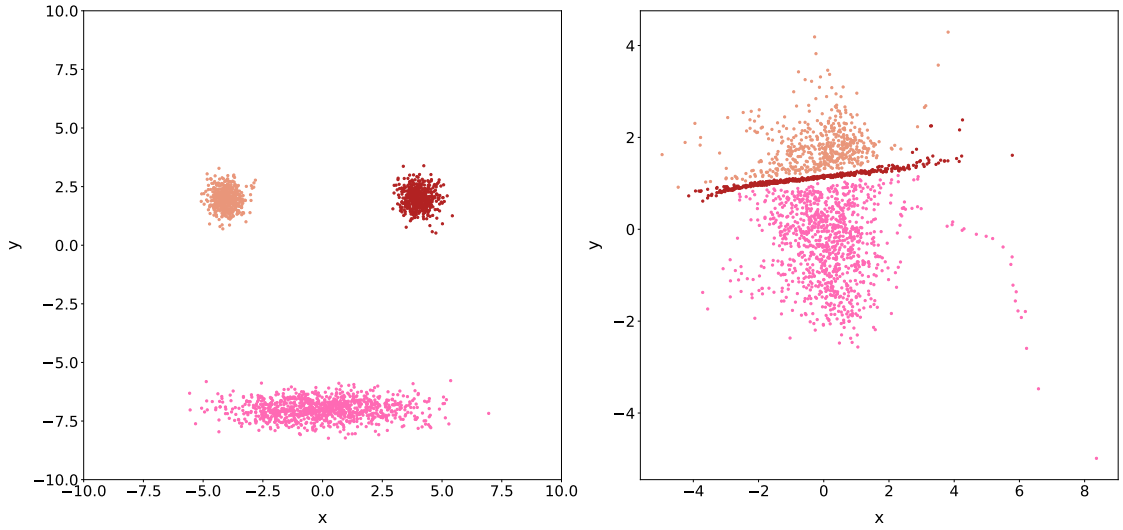


Figure 56: This figure shows the results of Method IV, $w_{fw} = 0.8$, $w_{bw} = 0.2$. $L_{fw} = 1.3$, $L_{bw} = 3.8$.

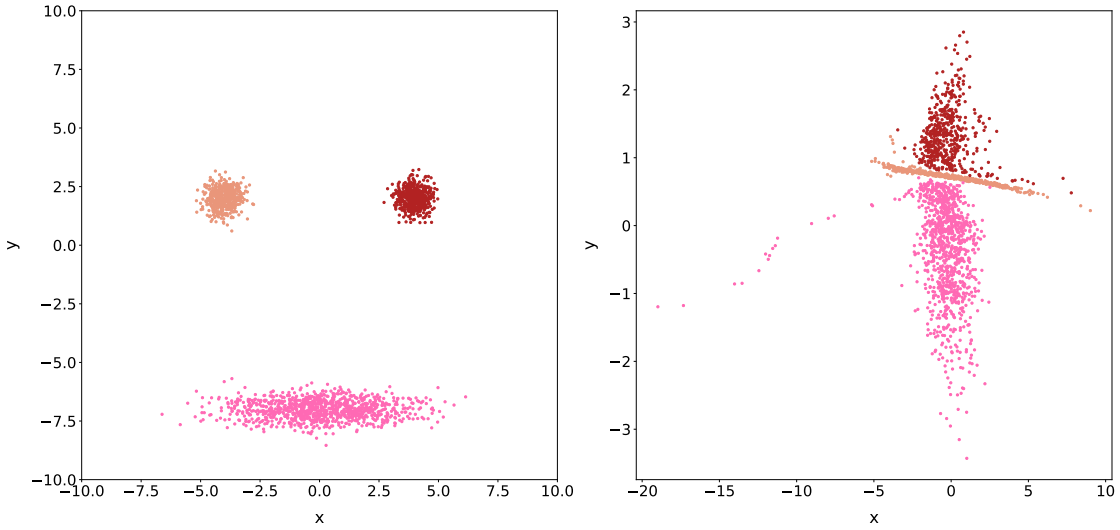


Figure 57: This figure shows the results of Method IV, $w_{fw} = 0.9$, $w_{bw} = 0.1$. $L_{fw} = 1.5$, $L_{bw} = 4.4$.

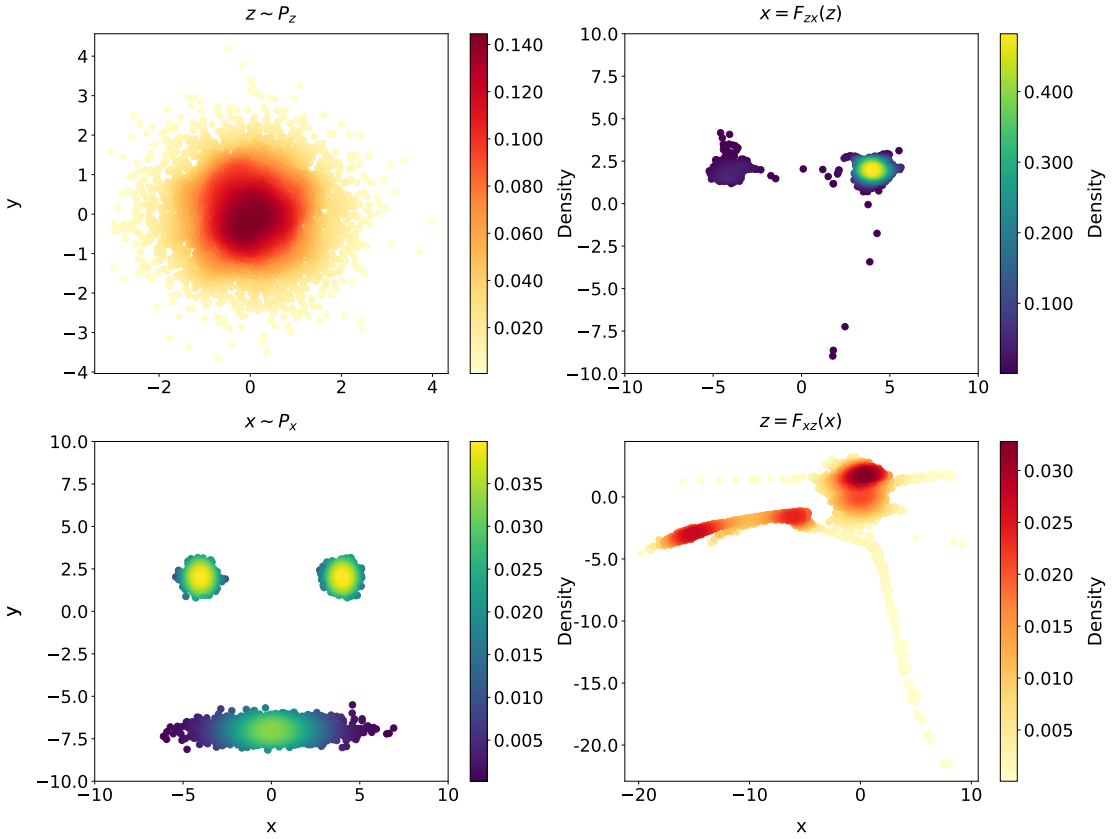


Figure 58: This figure shows the results of Method IV, $w_{fw} = 1.0$, $w_{bw} = 0.0$. $L_{fw} = 2.7$, $L_{bw} = 37$.

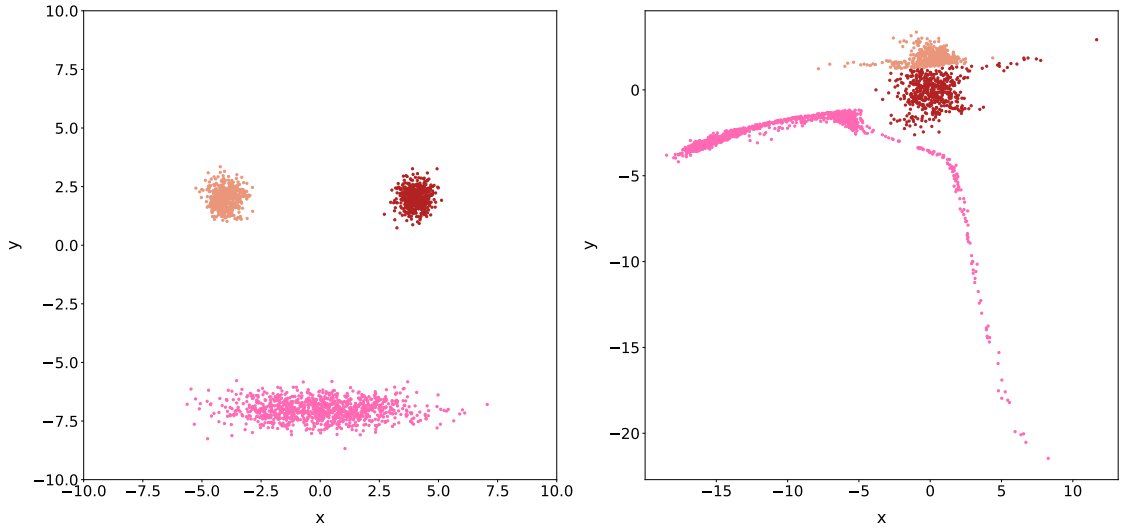


Figure 59: This figure shows the results of Method IV, $w_{fw} = 1.0$, $w_{bw} = 0.0$.$L_{fw} = 2.7$, $L_{bw} = 37$.