THE UNIVERSITY *of* EDINBURGH
School of Physics
and Astronomy

Machine Learning of Nonequilibrium States

MPhys Project Report

Matthew Stewart

Submitted for the 40pt MPhys Project course PHYS11016
March 28, 2022

Abstract

While the distribution of microstates is always known for equilibrium systems, the same cannot be said for nonequilibrium systems so novel methods must be used to investigate such distributions. Machine learning is a modern approach to learning underlying distributions of data. A neural network has been trained on snapshots of a paradigm nonequilibrium system called the totally asymmetric simple exclusion process and demonstrated understanding of the phase diagram and the physically relevant variables. Investigating other combinations of nonequilibrium system and network architecture are possible targets for future work.

Supervisor: Professor Richard Blythe

Personal statement

Initially, the machine learning (ML) literature was quite challenging for me so I had to spend about the few two to three weeks just reading about the theory behind the variational autoencoder (VAE) and some introductory material on information theory to understand KL divergence which was also new to me. Once I had a base level of understanding I could follow guides on the official Keras documentation to build my first VAE on the MNIST dataset of handwritten digits.

At this point I found some example code online applying the VAE to the Ising model so I started reading into simulating the Ising model to test out that code and verify some previously obtained results for myself. However, after some discussions about the direction of the project I abandoned the Ising model in favour of purely focusing on the totally asymmetric simple exclusion process (TASEP).

While continuing to read up on the VAE and in particular the 'reparameterization trick', in about week 5, I refamiliarised myself with and edited some code I had previously written for my Senior Honours Project to generate TASEP configuration snapshots but was unsure about how to generate uncorrelated snapshots. From discussion with my supervisor and some reading on TASEP correlations, I learned how to estimate correlation times but made a mistake in my code on my first attempt. Using Python for this was already slow enough so it annoyingly took a couple of weeks to get a rough estimate of the correlation times.

Now I could start building VAEs on TASEP snapshots which as far as I am aware has not been studied extensively before - previous work was primarily centred around equilibrium systems such as Ising and XY models. By the end of semester one I was confident with my code that was based on the previously mentioned Ising model VAE I had found online as well as code I had written in a workshop as part of the Data Analysis and Machine Learning course here. I was able to start analysing the output and get some pilot results here. Unfortunately, example VAEs online can be presented as a bit of a 'black-box' model and can be more interested in generating fictional pictures of cats instead of probing the latent distribution so I struggled to correctly visualise the latent distribution. I was glad to finally resolve this as it had been my main worry since the beginning of building VAEs.

After a week discussing the practice presentation and report section draft, at the start of semester two my supervisor had devised two new 'complexity' measures based on mean-field theory and I suggested a third based on KL divergence. I took me two weeks to code up the complexity measures. About the next three weeks were just collecting data on a tour of the TASEP phase diagram. Some results appeared illogical but once able to estimate the error on them, made more sense but were still unexpected.

The remainder of the semester involved generating snapshots and training VAEs on extensions of the TASEP with inhomogeneous hopping rates and elongated particles before writing up the report.

Acknowledgments

In addition to the support from my supervisor Prof. R. Blythe, I would like to acknowledge Dr. J. Szavits Nossan who introduced me to nonequilibrium statistical mechanics during my Senior Honours Project last year and I used some of my SH project code that simulated the TASEP as a base for this project.

Contents

1 Introduction

1.1 Equilibrium versus Nonequilibrium States

First, to understand what is meant by an equilibrium state we consider a system in contact with its surroundings. Let the system be air in some container recently taken out of a refrigerator and left out on a counter. This means the environment is the air in the room surrounding it and we could call this an infinite heat reservoir. Our intuition follows the first law of thermodynamics, that the system will reach an *equilibrium* state; the temperature of system will rise until it reaches the temperature of the surroundings [1], which in this example we would call room temperature. The surroundings being infinite means we can assume its temperature is fixed. Our intuition may also tell us that the temperature of the system may rise more quickly at first and from then increases more and more slowly before finally plateauing at room temperature. This is explained when we look at the *currents* of heat in and out of the system. Initially, the large temperature difference between the system and surroundings drives a large net current of heat into the system. After some relaxation time, the system reaches a state where net currents have decayed to zero and we call this state an equilibrium steady state - there is still a passive, back-and-forth exchange of heat between system and surroundings but zero *net* current.

So is the system nonequilibrium in that initial, 'transient', period where the net current is still decaying? Yes, but to properly understand what is meant by a nonequilibrium state in general, let us consider a second scenario. Now our container of air is connected to two new infinite heat reservoirs at fixed temperatures, one at each end. Let the left reservoir be at a temperature above room temperature and the right reservoir be at a temperature below it. This could be a radiator at one end and an open window in December at the other for example. From the moment the radiator is turned on and the window is opened, the net current will once again be changing in time and will stop changing after some relaxation time. The difference here is that the net current will become constant in space and time but not zero as the system cannot have the same temperature as both reservoirs at once. We call this a nonequilibrium steady state [2] - still 'steady' because the net current is constant but nonequilibrium because it is non-zero.

This means that nonequilibrium states are not just states of systems that we are still waiting to reach equilibrium, but can be states of **inherently** nonequilibrium systems. While the passive exchange of heat is characteristic of equilibrium steady states, for nonequilibrium steady states, the surroundings drive a current through the system and give a direction to the trajectory of the system; if shown a video of a nonequilibrium system one could identify the direction of the trajectory it should follow, by looking at reservoir temperatures in this example, and would be able to tell if the video was actually being played in reverse - something impossible if the video was instead of an equilibrium system.

Undergraduate physics focuses on equilibrium and shields us from the more complex nonequilibrium but in fact equilibrium systems are the exceptions and really nonequilibrium statistical mechanics can govern anything from weather, financial markets and epidemic spread [3]

to biological motors on the microscopic scale [4, 5] - the machinery of life constantly piecing together each and every one of our bodies' proteins. Therefore, being able to investigate nonequilibrium systems and evaluating the best methods to do so is a multidisciplinary, scientific goal.

In equilibrium statistical mechanics the probability of finding a system in a microstate, C at energy $E(C)$ and temperature T follows the Boltzmann distribution given by equation 1.

$$P(C) = \frac{e^{-\beta E(C)}}{Z} \tag{1}$$

Where $\beta = \frac{1}{k_b T}$ and the partition function Z is given by eqn. 2.

$$Z = \sum_C e^{-\beta E(C)} \tag{2}$$

In contrast, for nonequilibrium systems, there is no such one-size-fits-all distribution for the microstates so instead modellers are forced to turn to novel, tailored methods to explore the underlying distributions. Effective, tried and tested methods include the application of Markov processes, the Fokker-Planck equation or the Langevin equation [6] just to name a few examples.

1.2 Application of Machine Learning

Machine learning (ML) has exploded in popularity in recent years as evidenced by the dramatically increasing number of submissions made to popular machine learning journal, the Journal of Machine Learning Research, see figure 1. But how can ML be applied to physics? Classically, physics is about making predictions and we are able to make predictions by conducting an experiment and then use statistics to *learn* as much as we can from the data we collect from said experiment. ML is a modern method of doing exactly this - learning from data. The nature of ML is the produce empirical results just like physics and the fundamental approaches behind physics and ML greatly overlap. Therefore, new ML methods should have a place in a physicist's toolbox.

One of the most powerful tools in machine learning is the neural network (NN) and the branch of ML that employs these is called 'deep learning'. NNs are highly versatile and have famously been applied to tasks such as image classification [8], speech recognition [9], predictive advertising [10], chess engines [11] and the paradigm equilibrium statistical physics system, the Ising Model in 2D [12] as well as the XY Model in 3D [13] where NNs identified phases purely based on Monte Carlo snapshots of the systems and without any prior knowledge of order parameters or Hamiltonians. However, for these equilibrium systems the underlying distributions are known from the outset (by the physicist, not the NN), see Boltzmann distribution 1 so the goal of this project is to understand whether a

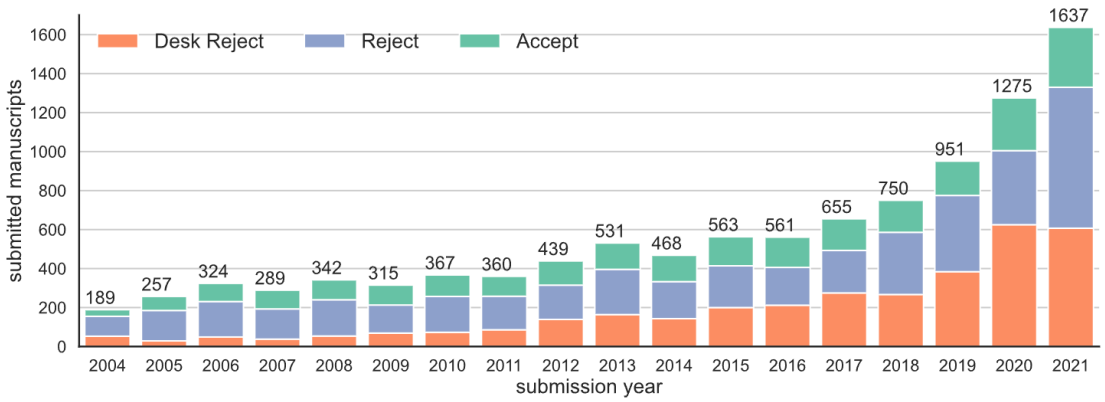


Figure 1. Number of submissions made to the Journal of Machine Learning by year with colours also show the the fractions accepted, rejected and 'desk rejected' (rejected without peer review) [7].

NN could learn the distribution of a system where the distribution is not known. Hence, nonequilibrium systems are excellent candidates.

This all sounds very promising and the success of NNs is undeniable but is there a theoretical understanding of how and why these techniques are so successful or do we have to accept them as 'black-box' models? One opinion is that our understanding is limited. Although, for decades now it has been shown that NNs can be capable of approximating any continuous function [14] and more recently it has been shown that deep learning is related to a well-tested and highly effective statistical physics technique - renormalization group [15].

The aim of this project is to generate a dataset of nonequilibrium states by simulating a nonequilibrium system and understand how effective a neural network can be at learning the underlying distribution.

2 Background

2.1 Markov Processes and the Master Equation

Consider a system that stochastically "jumps" from one configuration to another. Let the probability that the system is in some configuration C at time t be $P(C,t)$. Therefore we write the probability that the system is in a new configuration C' after a short time Δt, given the system was in exactly configuration C at time t, as $P(C', t+\Delta t \mid C, t)$. Let the overall probability of being in C' at time $t+\Delta t$, $P(C', t+\Delta t)$, be given by

$$P(C', t+\Delta t) = \sum_C P(C', t+\Delta t \mid C, t) P(C, t) \tag{3}$$

where the right hand side is the probability of there being a jump from a specific configuration C scaled by the probability of being in that configuration in the first place, summed over all possible C. This includes the definition of Markov Processes - that the probability of a jump to the next configuration C' only depends on the current configuration C and is independent of the history of the system ie. any configurations prior to C [16].

Now we can define $W(C \to C')$ as the rate at which the jump from C to C' occurs so that

$$P(C', t + \Delta t \mid C, t) = \begin{cases} W(C \to C')\Delta t & C' \neq C \\ 1 - \sum_{C'' \neq C} W(C \to C'') & C' = C \end{cases} \tag{4}$$

where in the case that $C' = C$, the right hand side is the probability of not having jumped to any other configuration C''. Combining 3 and 4 and taking the limit $\Delta t \to 0$ gives a differential equation for $P(C, t)$ called the master equation

$$\frac{\partial}{\partial t} P(C, t) = \sum_{C' \neq C} W(C' \to C) P(C') - \sum_{C' \neq C} W(C \to C') P(C) \tag{5}$$

where the first sum on the right hand side considers all jump rates *into* C while the second sum considers all jump rates *out* of C hence that term is negative.

The system is in a steady state if $\frac{\partial P(C,t)}{\partial t} = 0$. For an equilibrium system we know $P(C)$ is the Boltzmann distribution 1 and that the trajectory (the series of jumps between configurations) is reversible so for equilibrium steady states there is no net flow of probability between C and C' meaning

$$W(C' \to C) P(C') = W(C \to C') P(C) \tag{6}$$

This condition is called *detailed balance*. For a nonequilibrium system on the other hand, $P(C)$ is some other distribution and we classify steady states as nonequilibrium when detailed balance 6 is not satisfied in general. This means there *is* a net flow of probability between configurations hence why we see the trajectories of nonequilibrium systems have a 'direction' to them.

2.2 The Totally Asymmetric Simple Exclusion Process (TASEP)

Consider a one-dimensional lattice of length L, with sites indexed by $i = 1, ..., L$. Each site on the lattice can either be unoccupied or occupied by a single particle at any given time so we define occupancy $\tau_i = 1$ if site i is occupied and $\tau_i = 0$ if not. Similarly to the container of air from before 1.1, there is an infinite reservoir of particles (instead of heat) connected to the left most site on the lattice. Particles can hop one site to the right provided that site is unoccupied. See figure 2.

- Particles entering the lattice from the reservoir hop on to the left most site at a rate α.
- Particles anywhere on the lattice apart from the right most site hop at a rate ω, which is set to 1 without loss of generality. [2].
- Particles on the right most site exit the lattice at a rate β.

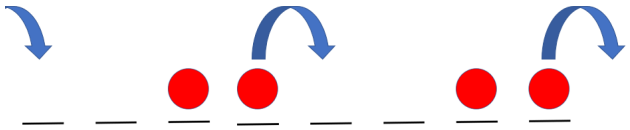


Figure 2. An example configuration for a lattice with $L = 8$. The red particles can hop to the right provided the site ahead is unnocupied or the particle is at the right most site. The first site is unnocupied so a particle can also hop onto the lattice from the reservoir.

Now the name of the process makes sense; 'totally asymmetric' because the particles only ever hop to the right and there is a 'simple exclusion' interaction between particles meaning they can never occupy the same site or overtake each other [17].

The local density at each site is defined as $\rho_i = \langle \tau_i \rangle$ where the average is over the distribution of configurations. The current at each site is defined as $J_i = \langle \tau_i(1 - \tau_{i+1}) \rangle$ [18] meaning it measures one average whether there is a particle at site i that can hop and therefore, the definition of currents follow out intuition - high current means that there is rate of hops along the lattice happening.

Just as the Ising model is a paradigm model for equilibrium statistical physics, the totally asymmetric simple exclusion process (TASEP) is a paradigm model for nonequilibrium statistical physics and one which can be modelled as a Markov process. An extension of the TASEP was first introduced by MacDonald et al. in 1968 as a mathematical model for mRNA translation in protein synthesis [19].

2.2.1 Phase Diagram

The different *phases* of a system are characterised by order parameters. For the liquid-gas transition, we have the density; for the Ising model, we have the magnetization. Order parameters need not just be one scalar but could be a a tensor like the $\mathbf{Q}$ tensor used to characterise phases liquid crystals [20].

For the TASEP, different pairs of α and β give rise to three different phases and these phases are characterised in the steady state by the bulk density ρ defined as the local density ρ_i of any site far enough away from either boundary and the current J where $J_i = J$ for all i as the current is constant across the lattice in the steady state. Bulk density and current are related by a current-density relation $J = \rho(1 - \rho)$.

The three phases are:

- Low density (LD) when $\alpha < \frac{1}{2}$ and $\alpha < \beta$
  - Current is limited by the low entry rate, α. By the time a new particle enters the lattice, it is likely that the previous one is already quite far ahead. The bulk density and current are $\rho = \alpha$ and $J = \alpha(1-\alpha)$ respectively [21] - both only depend on α.
- High density (HD) when $\beta < \frac{1}{2}$ and $\beta < \alpha$
  - Current is limited by the low exit rate, β. By the time a particle at the end of the lattice is able to exit, the particle behind likely already caught up creating a traffic jam. The bulk density and current are $\rho = (1-\beta)$ and $J = \beta(1-\beta)$ respectively - both only depend on β.
- Maximal current (MC) when $\alpha > \frac{1}{2}$ and $\beta > \frac{1}{2}$
  - α and β no longer limit the current, giving $\rho = \frac{1}{2}$ and $J = \frac{1}{4}$

These phases are summarised in table 1.

Phase	α	β	ρ	J
Low density (LD)	$< \frac{1}{2}$	$> \alpha$	α	$\alpha(1-\alpha)$
High density (HD)	$> \beta$	$< \frac{1}{2}$	$1-\beta$	$\beta(1-\beta)$
Maximal current (MC)	$> \frac{1}{2}$	$> \frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$

Table 1. Summary of the three phases for the TASEP in terms of entry rate α, exit rate β, bulk density ρ and current J.

They can also be depicted by a phase diagram in the α-β plane as shown in figure 3.

On inspection of the phase diagram it is clear that the LD and HD phases are symmetrical about the line $\alpha = \beta$. Consider one point in the LD phase, $(\alpha_{LD}, \beta_{LD})$ and the reflected point in the HD phase, $(\alpha_{HD} = \beta_{LD}, \beta_{HD} = \alpha_{LD})$. Particles in the LD system follow equivalent dynamics to 'holes' (unoccupied sites) in the HD system and vice versa. This is clear if we look at an extreme case - an HD system that is fully traffic jammed but eventually a particle exits with rate β_{HD} and the rest of the particles can sequentially hop to the right and get jammed again. This scenario is dynamically equivalent to a hole entering the lattice from the right at rate $\beta_{HD}\alpha_{LD}$ and progressing back along the lattice by hopping to the left exactly like a particle hopping to the right would in the corresponding LD system.

There are two special lines on the phase diagram that are of interest to this investigation. The first is the line $\alpha + \beta = 1$ called the 'mean-field line'. Along this line, the mean-field theory is exact [2] and give totally flat density profiles (plotting ρ_i against i) even at the boundaries. The second line of interest is the line $\alpha < \frac{1}{2}$ and $\beta < \frac{1}{2}$ called the 'coexistence line'. Along this line, taking a snapshot of the system at some time would observe some fraction of the lattice in the LD phase and remainder of the lattice in the HD phase. Hence, there is the point between them where the density is discontinuous and this is called the 'shock' and the position of the shock moves along the lattice with time.

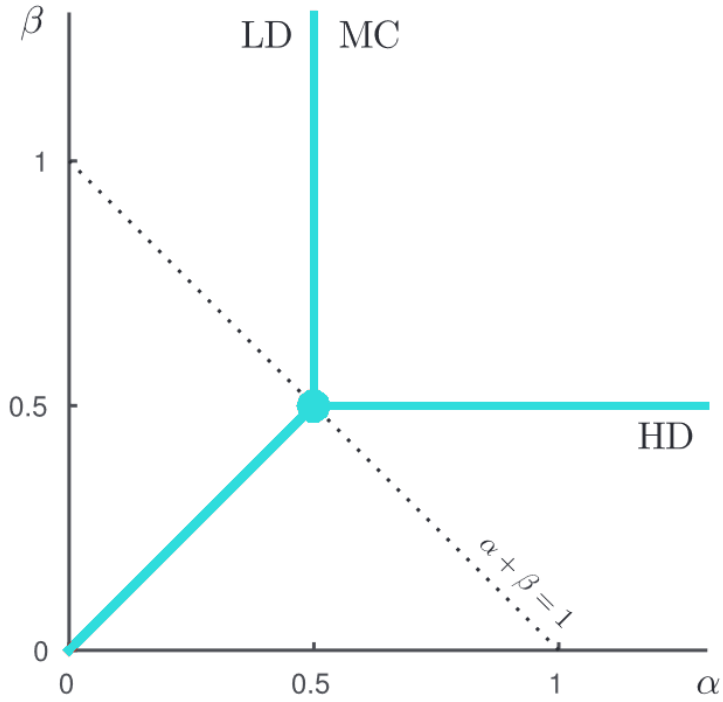


Figure 3. The TASEP phase diagram showing the low density (LD), high density (HD) and maximal current (MC) phases. The $\alpha + \beta = 1$ line called the mean-field line is where mean-field theory matches the exact solution. The phase boundary between LD and HD phases is called the coexistence line. Figure adapted from [22].

2.2.2 Extensions of the TASEP Model

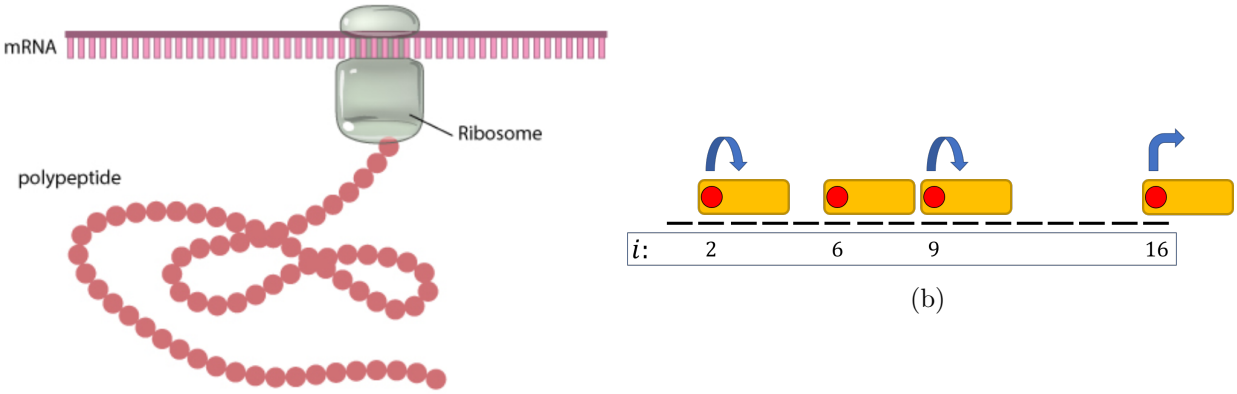
Instead of setting all $\omega = 1$ for all sites, we can define a vector $\omega = (\omega_1, \omega_2, ..., \omega_(L-1)$ (exit rate is still β), of site-dependent hopping rates to introduce disorder into the system. All ω_i can be generated by taking the inverse of a 'waiting time' $u_i = \frac{1}{\omega_i}$ where each u_i is sampled from some distribution of choice.

Particle size l can be increased from $l = 1$. This means particles require $l - 1$ unoccupied sites ahead of them to be allowed to hop (at a rate given by ω_i where i is the site occupied by the left most site of the new extended particle).

The two extra models studied in this project were:

1. $l = 1$ with u_i each drawn from an exponential distribution $p(u) = e_{-u}$, $\alpha = 0.2$ and $\beta = 0.2$
2. $l = 9$ with α, u_i and β were the actual hopping rates corresponding to a real gene sequence

The latter is a model for mRNA translation, see figure 4, where the hopping rates each depend on how accessible the animo acid pertaining to that particular codon is [23].



(a) (b)

Figure 4. (a) A ribosome 'hops' along codons on the mRNA and attaches an amino acid to the polypeptide chain which, once complete, forms a protein [24]. (b) mRNA translation can be modelled as an extension of the TASEP with inhomogeneous hopping rates and elongated particles - here $l = 3$ and arrows indicate the allowed hops.

2.3 Entropy and Kullback–Leibler Divergence

From information theory, the definition of information I of an event E drawn from a distribution $P(E)$ is given by $I = -log(P(E))$ [25] - if E with a very small probability occurs, I is large. For example, if we play hangman and guess the letter 'E' there is a high chance

it is in the word and if so, we do not learn much information because there are still many words it could be; meanwhile, if we had guessed the letter 'Z' there is a low chance it is in the word but on the occasion that it is, it massively reduces the number of possible words so gives us a lot of information. The *entropy* H of $P(E)$ is the expected value of information, for a discrete random variable E

$$H(E) = -\sum_{E} P(E)\log(E) \tag{7}$$

where the sum is over all possible E and the similarity to the Gibbs entropy from thermodynamics is clear. Now we can define relative-entropy also known as Kullback–Leibler (KL) divergence of a probability mass distribution $P(x)$ with respect to another distribution $Q(x)$ as

$$D_{KL}(P(x)\,||\,Q(x)) = \sum_{x} P(x)\log(P(x)) - \sum_{x} P(x)\log(Q(x)) \tag{8}$$

$$= \sum_{x} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \tag{9}$$

for a discrete random variable x or

$$D_{KL}(P(x)\,||\,Q(x)) = \int p(x)\log\left(\frac{p(x)}{q(x)}\right)dx \tag{10}$$

for a continuous random variable x where $p(x)$ and $q(x)$ are probability density functions corresponding to P and Q for any choice of base for the log. Equation 8 is very to close to a difference in entropy but the second term is still 'with respect to' P so KL divergence is a measure of dissimilarity between two distributions but is asymmetric - $D_{KL}(P(x)\,||\,Q(x)) \neq D_{KL}(Q(x)\,||\,P(x))$ in general. $D_{KL}(P(x)\,||\,Q(x))$ can be interpreted as a measure of how surprised we would be if we were using Q as some model and discovered that the actual distribution was P. If $D_{KL}(P(x)\,||\,Q(x)) = 0$, P and Q carry identical information. Also, KL divergence is always greater than zero which is known as the Gibbs' inequality [25].

2.4 Machine Learning and Neural Network Models

Consider fitting some experimental data with a best-fit curve. To accomplish this we require:

1. A dataset $D = (\boldsymbol{X}, \boldsymbol{y})$ where $\boldsymbol{X}$ is our independent variables and $\boldsymbol{y}$ is our dependent variables expressed as vectors. Say in our experiment we took 100 measurements - $\boldsymbol{X}$ and $\boldsymbol{y}$ both would be 100 component vectors.

2. A choice of function (model) for our curve $f(x; \boldsymbol{b})$ (x could be any point in some x-space not just limited to our data $\boldsymbol{X}$) - could be any functional form we like, take a linear function for example - where $\boldsymbol{\theta}$ is the set of our fit parameters ie. the gradient and y-intercept of our line. The function f is defined by $\boldsymbol{\theta}$ and maps some input x to some output y.

3. A choice of loss function $K(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{\theta})$ that scores how close our model got to the true values of $\boldsymbol{y}$ so we want to minimise this. We commonly choose K to be least squares error.

These three ingredients are required for all ML problems and the field of ML is all about exploring different types of models and loss functions. A neural network (NN) is just one such type of model. So the fundamentals of ML are familiar to physicists and nothing to be intimidated by.

Let us properly define an experiment; let us say we are aiming to recreate the phase diagram for water. In the lab we could choose a particular pressure, volume and temperature (P, V and T) and measure the density of water under these three conditions. This makes up one datapoint and in our experiment we collect 100 datapoints so now our $\boldsymbol{X}$ is 3x100 matrix of Ps, Vs and Ts. Our $\boldsymbol{y}$ is still a 100 component vector of densities.

As the name suggests, NNs are built out of units called *neurons*. A neuron can take a datapoint $\boldsymbol{x}$ from our experiment as an input where in this case $\boldsymbol{x}$ has three components (x_1, x_2, x_3). The fit parameters for a neuron are *weights* which is a vector $\boldsymbol{w}$ and a *bias* which is a scalar. The length of $\boldsymbol{w}$ is the same as $\boldsymbol{x}$, ie. each input to the neuron is assigned a weight.

The output z of the neuron is calculated in the following two steps [26] depicted in figure 5a

1. Each input is scaled by its own weight and the sum of these is shifted by the bias to give the ouput of this step u ie. $u = \boldsymbol{x} \cdot \boldsymbol{w} + b$

2. Then a nonlinear *activation function* A is applied ie. $z = A(u)$

There are a variety of different choices for activation functions. One of the most popular is *rectified linear unit* (ReLU)

$$A(u) = \begin{cases} 0 & u < 0 \\ u & u \geq 0 \end{cases} \tag{11}$$

Another is the *sigmoid* function

$$A(u) = \frac{1}{1 + e^{-u}} \tag{12}$$

We have just just built the simplest neural network model and once we fit this model we would have a way of estimating the density of water some set of (P, V and T). However, with just a single neuron we likely have given our model the flexibility to capture the underlying relationship between (P, V and T) and density, analogous to not including high enough order terms when fitting a polynomial. Therefore, our model, instead of being just a single neuron, is a 'network' of neurons, see figure 5b. Now the neurons are arranged in layers - the first layer is called the input layer, the last is the output layer any in between are called *hidden* layers and the presence of two or more of these mean we are building a *deep* NN model. Neurons in the input layer are just like before where each input is one of the *features* $(x_1, x_2, x_3...)$ based on our data. The inputs to each neuron in any hidden layers and the output layer are the outputs of all the neurons in the previous layer. Each neuron is typically referred to as a *node* and the number of nodes per layer as well as the number of hidden layers can be any number we choose depending on the task. Generally all nodes within a layer have the same activation function but they may differ between layers.

Now the role and interpretations of the nonlinear activation functions are clear. Without applying them, our output layer outputs a composition of linear functions which is itself a linear function of our input features which renders the use of hidden layers unnecessary. Hence, activation functions allow the NN to learn more complex, nonlinear distributions. Also, from the perspective of the machine, 'bigger numbers mean more important' so the ReLU function 11 effectively switches off any nodes outputting something deemed unimportant. If we wanted out output layer to output a probability, perhaps for a classification problem, the sigmoid function 12 could be a good choice as its output is always between 0 and 1.

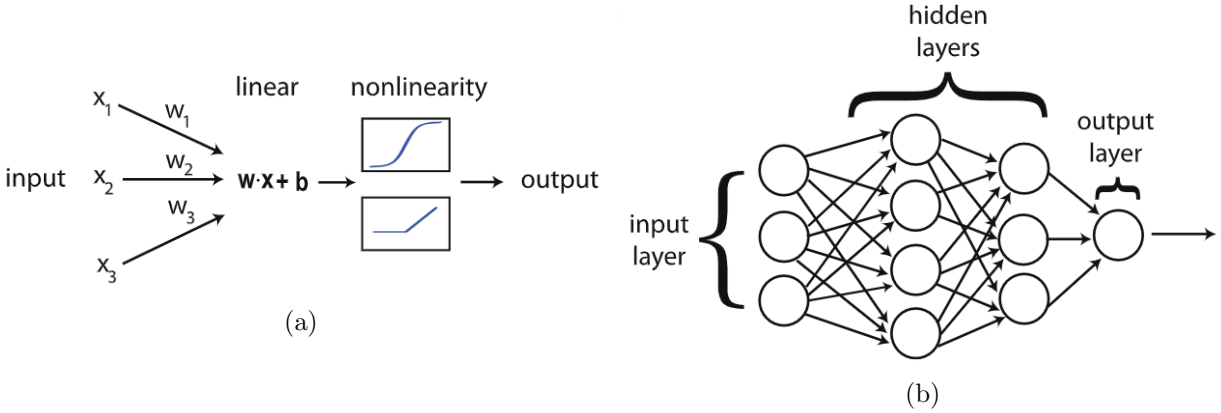


Figure 5. (a) How a single neuron maps and its inputs (x_1, x_2, x_3) to an output using a linear combination followed by applying a nonlinear activation function. The activation step would apply one of the two example functions which are sigmoid (above) and ReLU (below). (b) A general dense NN architecture where each circle is a neuron (node) which has its own bias and each arrow is an edge corresponding to a weight. Nodes are arranged into input, hidden and output layers. Both subfigures from [26]

We can imagine how NNs are such versatile and flexible models as counting up all the weights and biases often gives thousands if not millions of fit parameters. There are also other types

of layers other than the *dense* layers (each node in a layer is connected to every node in the previous layer) described here such as convolutional layers commonly used when working with image data.

2.4.1 Training NNs

The process of fitting the model is called *training* and as usual we are adjusting the trainable parameters to minimise the loss function. Our trainable parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_n)$, containing all the weights and biases for our chosen architecture, will be randomly initialized. To update $\boldsymbol{\theta}$ the two-step *backpropagation* algorithm is utilised. First is the *feedforward* step where our data is fed forward through the network from input layer to output layer and calculate our total loss according to our loss function K of choice. The second step is to go backwards through the network and calculate

$$\nabla L = \left(\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, ..., \frac{\partial L}{\partial \theta_n} \right) \tag{13}$$

Then update the weights and biases from $\boldsymbol{\theta}$ to $\boldsymbol{\theta'}$ according to the gradient descent update rule

$$\boldsymbol{\theta'} = \boldsymbol{\theta} - \eta \nabla L \tag{14}$$

Where η is called the *learning rate* and tells us how big of a step are we willing to take each time we adjust $\boldsymbol{\theta}$. We would like η to be large enough that training does not take too long but not so large that we keep stepping back and forth over the minimum in K meaning it is never properly minimised. To avoid this one can decrease η as K starts to plateau [27].

2.4.2 Bias-variance Tradeoff

The common pitfalls in any data fitting problem are underfitting and overfitting. Underfitting means we have not given out model enough flexibility to capture the underlying distribution of the data and in ML terminology this is called high *bias*. Overfitting means our model it too flexible allowing it to become too familiar with the data - it has gone beyond learning how the input features relate to the output and is learning random fluctuations unique the that particular dataset and in ML terms this is called high *variance*. Our ideal model has both low bias and low variance.

Achieving low bias is not too difficult, we just have to give our NN enough layers and nodes equipped with suitable activation functions. Achieving low variance on the other hand is more difficult. The common methods utilised in this investigation are

- *Stochastic Gradient Descent* [28] - The data is randomly divided into *minibatches* much smaller than the size of the dataset. The backpropagation algorithm is applied to update $\boldsymbol{\theta}$ after every minibatch. Once all the batches are used up we call this one *epoch* of training. We repeat this for several epochs typically but each time the dataset is randomly reallocated into the minibatches. This discourages overfitting because although the training data as a whole is the same each epoch, the $\boldsymbol{\theta}$ updates are made on different minibatches each epoch meaning the network does not get too familiar with the training data.

- *Validation Loss* - The data is partitioned into a training set and a *validation* set. The model updates $\boldsymbol{\theta}$ based only on the training set but at the end of every epoch, we also calculate K of the validation set. As the model starts to learn important relations between features and output, the training and validation losses will decrease. However, if the training loss ever continues to decrease while the validation loss begins to rise, it indicates the model has latched onto some unique fluctuation in the training data and does not generalise well to the validation set. See figure 6.

- *Dropout* [29] - For each update during training, a fraction (typically about 20%) of the nodes in a particular layer are 'dropped' meaning they are temporarily switched off, see figure 7. This means each batch sees a slightly different architecture to the last.

2.4.3 Unsupervised learning and the Variational Autoencoder

Up to this point loose analogies have been made to traditional curve fitting where we carry out an experiment or perhaps a simulation and collect data $D = (\boldsymbol{X}, \boldsymbol{y})$ and train the NN by minimising some K, a measure of how close the output of the NN is to the true $\boldsymbol{y}$. This is called *supervised learning* because we use $\boldsymbol{y}$ to incentivise the learning process - we are like a teacher that has a marking scheme $\boldsymbol{y}$ that, at the end of each batch, tells the network how to improve.

Another main type of ML learning problem is *unsupervised learning*. For this case, $D = (\boldsymbol{X})$ - there is no $\boldsymbol{y}$ targets to aim for. The role of the network is now not to find the best mapping $\boldsymbol{x} \rightarrow \boldsymbol{y}$ but is to learn for itself patterns and groupings within the input data. This more modern ML approach is important to study if we with to apply ML to real world applications; the real world data thrown at us is not normally in the structured format like the data we obtain by experiment and will be vastly unlabelled. The result of an unsupervised approach can be to summarise the data, reduce noise [30] and even generate new 'fictional' data [26]. Tasked with learning the underlying distribution of the TASEP configurations, an unsupervised approach is therefore well suited to the task.

The type of unsupervised NN model used here is a *deep latent variable model*. A latent variable z is not directly observed in but in some way summarises important features found in the observed variables $\boldsymbol{x}$ [31]. The deep latent variable model used in this project is the *variational autoencoder* (VAE). This begins with a model that maps a given observed data-point $\boldsymbol{x}$ to a vector $\boldsymbol{z}$ of latent variables where the dimensionality of $\boldsymbol{z}$ is much less than $\boldsymbol{x}$.

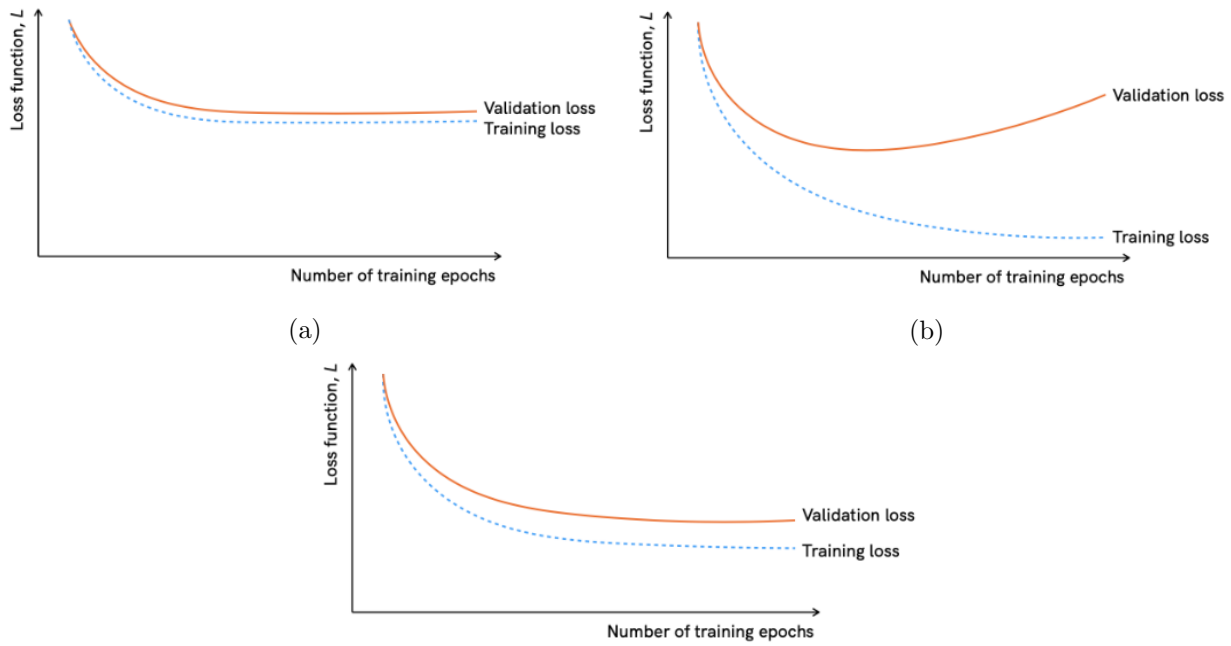


Figure 6. (a) An example of underfitting where the training and validation losses plateau but the loss overall is still high, indicating that the chosen architecture is not flexible enough to learn the relationship between input features and the output - high bias. (b) An example of overfitting where the validation loss begins to rise meaning the model has become overly specialised to the training data - high variance. (c) An ideal model with low bias and variance.

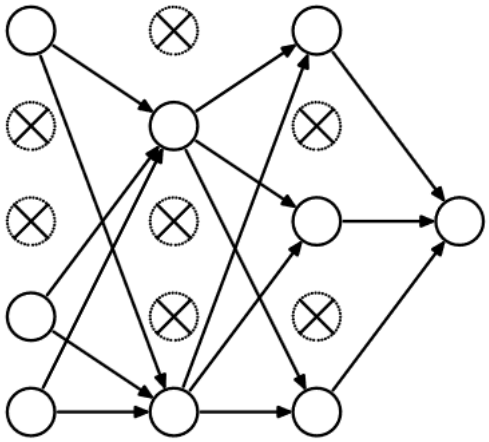


Figure 7. Hidden layers of an NN with dropout applied. A fraction of nodes on each layer are randomly dropped to discourage the NN from overfitting to the training data [29]

We use an NN model called the *encoder* to represent the conditional probability distribution $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ where φ are the trainable parameters (weights and biases just like before) - some input $\boldsymbol{x}$ is encoded in a condensed representation $\boldsymbol{z}$ which has its own underlying probability distribution $p(\boldsymbol{z})$. How do we know if the encoder is really extracting important features and encoding them in $\boldsymbol{z}$ and not just overcompressing and losing information? We introduce another NN model called the *decoder* which takes $\boldsymbol{z}$ output by the encoder and reconstructs the original datapoint $\boldsymbol{x}$ as best possible - represented by another conditional probability distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, see figure 8.

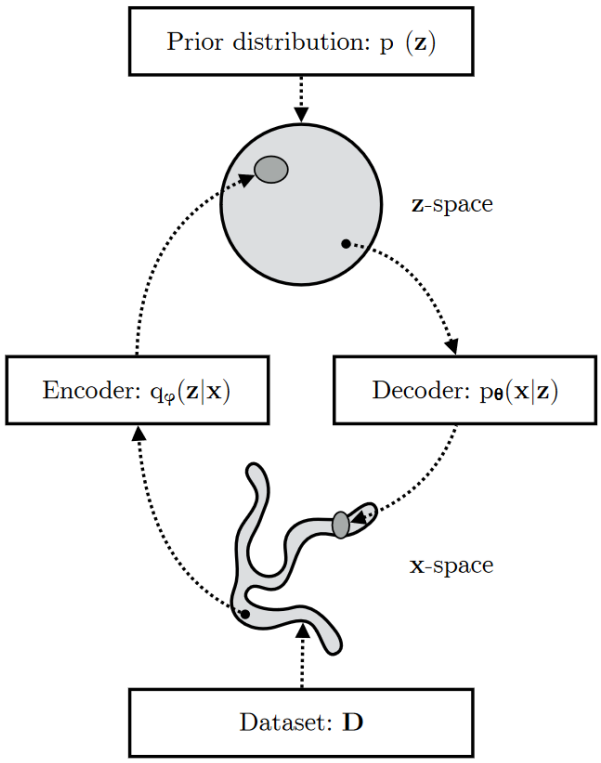


Figure 8. $q_\varphi(\boldsymbol{z}$ characterises the encoder NN and maps a point in x-space, the observed data, to a point in the latent z-space. $p_\theta(\boldsymbol{x}$ characterised the decoder NN and maps the point in z-space back to the point in x-space. The dataset is likely drawn from some unknown and complicated underlying distribution while the $p(\boldsymbol{z})$ that we aim for is chosen to be simple. Adapted from [31]

.

Variational Bayesian theory has derived a loss function K for the VAE called the *evidence lower bound* (ELBO) [26, 31]

$$K = -\mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] + D_{KL}(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})) \tag{15}$$

Where the $\mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[...]$ means taking the expected value over $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$. The two terms each have a neat interpretation when minimised:

1. For any choice of $\boldsymbol{x}$ mapped to a $\boldsymbol{z}$, we want the decoder to be able to best invert that mapping ie. maximise the likelihood of returning to $\boldsymbol{x}$. The first term is just the expected negative log-likelihood so minimising this is equivalent to maximising likelihood of the best possible reconstructions

2. The second term is the KL divergence of $p(\boldsymbol{z})$ with respect to $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$. The important part is we can choose a simple and tractable form for $p(\boldsymbol{z})$ *a priori* so minimising this term encourages $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ to be similar to $p(\boldsymbol{z})$.

This is a powerful method because I allows us to take data that is drawn from a unknown and complicated underlying distribution, such as a nonequilibrium distribution, condense the information down as into only the system's important underlying variables and encode these as well as possible in a simple and tractable. The distribution we aim for, $p(\boldsymbol{z})$, is chosen to be a Gaussian distribution with mean of 0 and a variance of 1. This distribution is also easy to sample from for any further analysis.

There is a deterministic version of the VAE simply called the *autoencoder* (AE) which also performs this mapping. What is attractive about the VAE is that it is not specifying the exact latent vector encodings for our data like an AE would, but is specifying a Gaussian probability density function in the latent space parameterised by a mean, μ and variance, σ^2. Therefore, after training we can sample from this continuous distribution and decode $\boldsymbol{z}$ that were not encodings in the original data.

To summarise the VAE training process:

1. Feed forward a batch of data through encoder which outputs a μ and σ^2 ($\log\sigma$ in practice) for the latent distribution

2. Calculate $D_{KL}(q_\varphi(\boldsymbol{z}|\boldsymbol{x})\,||\,p(\boldsymbol{z}))$

3. Sample the latent distribution and feed forward through decoder which ouputs reconstructions of the original data.

4. Calculate $-\mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})]$

See figure 9.

It must be noted that using backpropagation to update the weights and biases in φ and θ is not possible when the random sampling in step 3 occurs. To continue using backpropagation we train via the 'reparameterization trick' [31]. Now the lateent space is sampled like $z = \mu + \sigma\epsilon$ where ϵ is a sample from a Gaussian with mean of 0 and variance of 1. Hence, the random step is now generating ϵ which happens outside the NN ie. ϵ is independent of φ and θ so now there is no issue using backpropagation as normal.

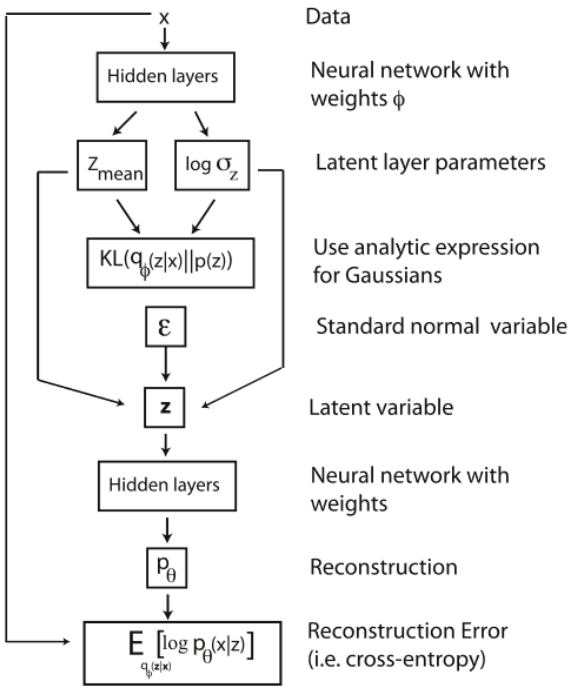


Figure 9. Summary of the VAE training process [26]

.

3 Methods

3.1 Simulating the TASEP

Simulations were carried out using an object oriented approach. Firstly, a simulation object is instantiated which allows the user a choice of initiation rate, α; termination rate, β; lattice size, L in number of sites; particle size, l in number of sites occupied per particle and hopping rates, $\{k_i\}$. Unless otherwise stated, default values of $L = 100$, $l = 1$, $k_i = 1\ \forall i$ were used. Secondly, an array of 'site' objects was created and stored as a 'lattice' object but it is important to note that, in the eyes of the computer, the lattice was two sites longer than the true size specified by the used because one fictional site was added to each end of the lattice to represent left and right-hand side particle reservoirs.

At time $t = 0$, the initial configuration has every site empty apart from the left particle reservoir. To choose a trajectory of the system (which particles hop and when) the Gillespie algorithm [32] was used. This is a stochastic modelling algorithm inspired by Monte Carlo methods, most commonly applied to biochemical systems [33]. It involved the following four steps:

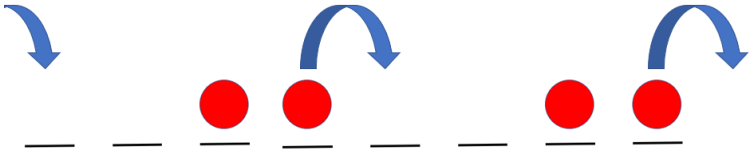
1. Based on the current configuration C, calculate a propensity vector $\boldsymbol{\omega}$. This was an $L + 1$ length vector where the i'th element corresponded to site i including the left reservoir ($i = 0$) but not the right. Each element $\boldsymbol{\omega}_i$ was k_i (or α for $i = 0$) if site i

was occupied and had at least its own length of unoccupied sites ahead of it and zero otherwise. Simply, $\boldsymbol{\omega}$ was a vector of the hopping rates where hops were possible.

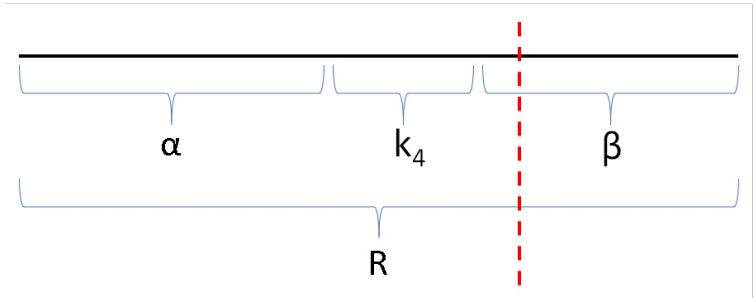
2. Calculate $R(C)$, the sum $\sum_{i=0}^{L} \boldsymbol{\omega}_i$ and sample a random number from a uniform distribution ranging from 0 to $R(C)$ using a PCG random number generator [34]. This selects which single particle hops next and hence chooses the next configuration as illustrated in Figure 10.

3. Sample a waiting time from an exponential distribution with average waiting time $\frac{1}{R(C)}$ ie. sample from $R(C)e^{-R(C)t}$, which is the characteristic form of waiting times in Markov processes [35].

4. Update the configuration to the chosen new configuration and return to step 1.



(a) The blue arrows show the possible hops. A particle can hop from the $i = 0$ (left reservoir), $i = 4$ and $i = 8$ sites.



(b) The propensity vector has $\boldsymbol{\omega}_0 = \alpha$, $\boldsymbol{\omega}_4 = k_4$, $\boldsymbol{\omega}_8 = \beta$ with all other elements being 0. Therefore, $R(C) = \alpha + k_4 + \beta$. In this case, the uniform random number from 0 to $R(C)$ landed between $\alpha + k_4$ and $\alpha + k_4 + \beta$ so the particle corresponding on site with hopping rate β, at $i = 8$, is chosen to hop.

Figure 10. (a) and (b) illustrate 'step two' of the Gillespie algorithm showing how a new configuration is chosen from an example lattice with $L = 8$, $l = 1$

A sweep is defined as L iterations of the algorithm. To determine when the system had reached the steady state, the average number number of particles on the lattice over 10 sweeps was compared to that of the next 10. Once these differed by less than 2%, at the end of the next 10 sweeps (just to be safe) the steady state was estimated to have been reached.

These simulations were carried out in the Python programming language using the NumPy library.

3.2 Generating Uncorrelated Snapshots

In any data analysis, it is important to understand whether data points are correlated or not as uncorrelated data makes the analysis more simple and would best represent the underlying distribution of possible configurations for a given point on the phase diagram. Therefore, the snapshots taken from the simulations to be used as input to the network must be uncorrelated. The input dataset consisted of 20000 snapshots per unique point on the phase diagram but using 20000 successive configurations would give a highly correlated dataset compared to 20000 totally random configurations sampled from the underlying distribution. Therefore, waiting a few correlation times between snapshots would generate uncorrelated data so the correlation time must be estimated.

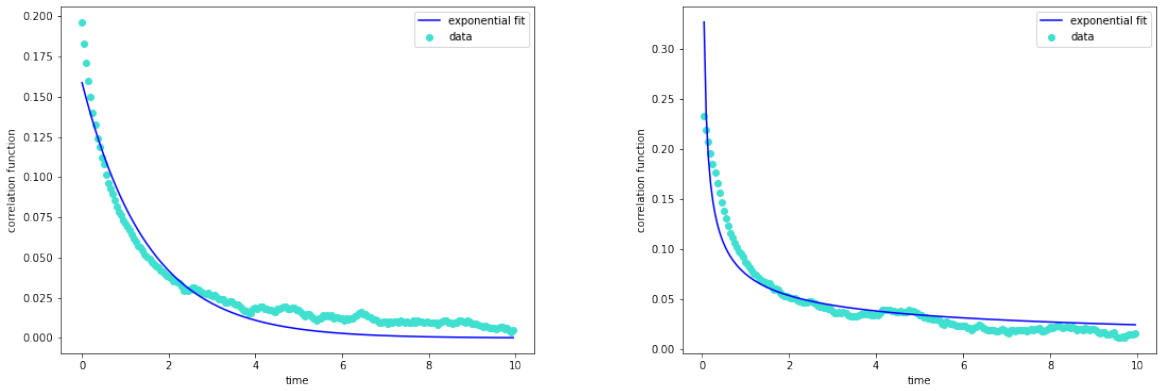
To obtain data for the main analysis, simulations were carried out for one point in the LD phase ($\alpha = 0.25$, $\beta = 0.75$) 3 and one point in the MC phase ($\alpha = 0.75$, $\beta = 0.75$). The correlations in the HD phase should be identical to the LD phase due to the symmetry between particles in one phase and holes in the other so there was no need to independently estimate correlation time for the HD phase. The occupancy of a site in the middle of the lattice σ in the default setup ($L = 100$, $l = 1$) was recorded as a function of time t for both LD and MC phases. From this, a correlation function (CF), given by equation 16, could be calculated where the averages are over 5000 repeat simulations. In the LD phase this is expected to behave like exponential decay $\sim e^{\frac{-t}{\theta}}$ while in the MC phase a power law $\sim t^{-\gamma}$ is expected [36] where θ and γ give measures of correlation times in LD and MC phases respectively.

$$CF = \langle \sigma(0)\sigma(t) \rangle - \langle \sigma(0) \rangle \langle \sigma(t) \rangle \sim \begin{cases} e^{\frac{-t}{\theta}} & \text{LD phase} \\ t^{-\gamma} & \text{MC phase} \end{cases} \tag{16}$$

The resulting curves were linearised and least-squares fits were carried out. These are shown in figure 11. The linear fits were more reliable in terms of best estimate parameter errors and gave $\theta = \frac{1}{k} = 3.7$ time units for the LD phase. Therefore, a time difference of 9 time units between snapshots would be nearly 3 correlation times so would be suitable to generate uncorrelated enough datasets. By inspecting figure 11a and 11b, 9 time units in long enough to see the correlation function decay to close to zero for both LD (and hence HD) and MC phases.

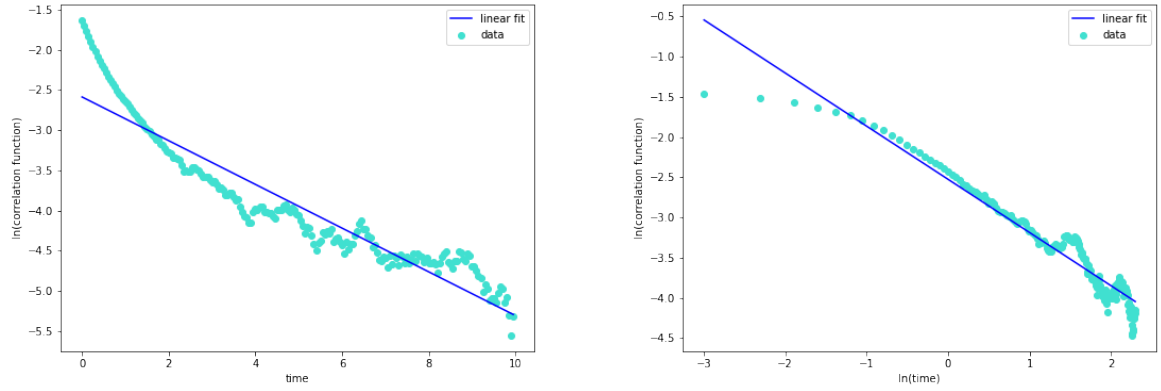
It is also noted that all curves in figure 11 are quite noisy and could be improved by averaging correlation functions over more simulations to give a better estimate of correlation times. However, the simulations in an interpreted language like Python were already quite slow so rewriting them in a compiled language like C++ or Fortran would likely give a better estimate of correlation times.

Simulations were then carried out for a point on the mean-field line with $\alpha = 0.2$, $\beta = 0.8$ and then for a point on the coexistence line with $\alpha = \beta = 0.2$. 20000 snapshots were recorder for each with $L = 100$, $l = 1$ waiting 9 time units between each snapshot. Only



(a)

(b)



(c)

(d)

Figure 11. Estimating correlation times with data from the simulations in light blue and least-squares best fit curves in dark blue. (a) and (b) show correlation functions against time for LD and MC phases with Ae^{-kt} and $Bt^{-\gamma}$ fits respectively where A and B are constants. (c) and (d) show the same data but linearised by taking logarithms on appropriate axes. Linear fits of the forms $ln(A)-kt$ and $ln(B)-kln(t)$ gave best estimate parameters of $A = 0.075(11)$, $k = 0.27(2)$, $B = 0.080(10)$ and $\gamma = 0.066(7)$.

the occupancies of the 100 real sites were recorded for each snapshot and not either of the fictional sites representing the reservoirs. The data was written to csv files.

The curve fitting was carried out using the iminuit library [37].

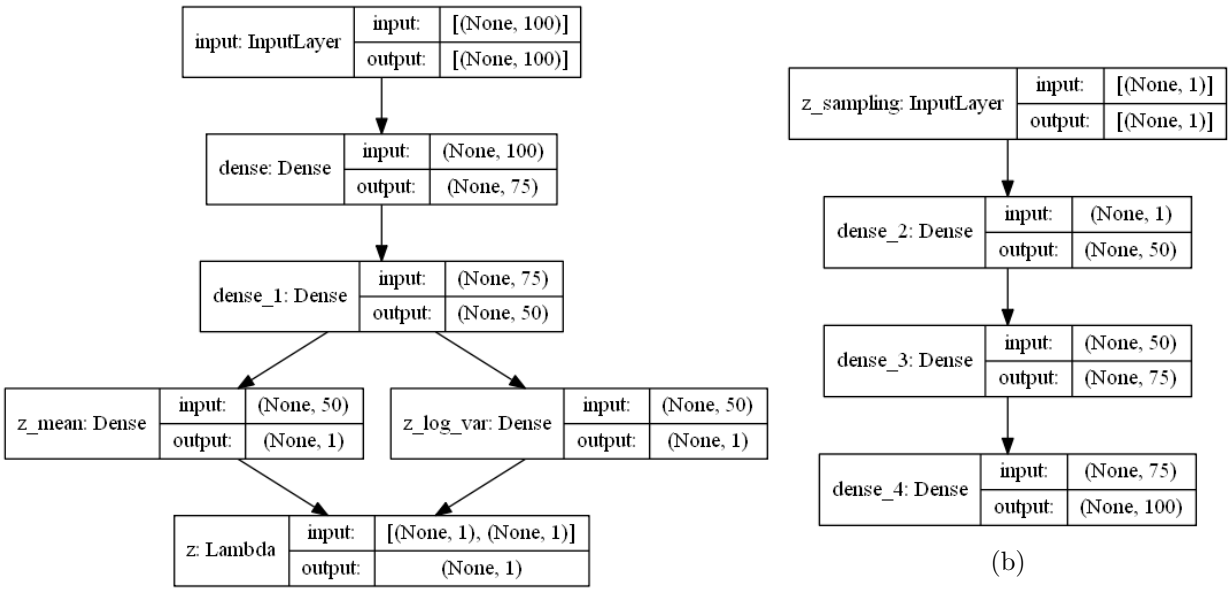
3.3 Programming Neural Networks

Once the two datasets, one for the mean-field point and one for the coexistance point, were read in from the csv files, they were partitioned into training and test sets and shuffled using the sklearn.model_selection module with 80% for training and 20% for testing. Two identical VAE objects, were instantiated, one for each dataset.

Various architectures were trialled using 2 to 3 dense layers each with somewhere between 25 and 100 nodes for the encoder and similar for the decoder. Many were successful and a simple, symmetrical architecture was chosen as illustrated is figure 12. The encoder inputs were the TASEP snapshots - 100 dimension vectors of 1's and 0's, for occupied and unoccupied sites respectively. Choosing the latent space to be one dimensional meant the encoder outputs were three one dimensional vectors; a mean and variance parameterising the latent Gaussian distribution and a sample from said distribution. These samples were taken via the 'reparameterisation trick' 2.4.3 using a lambda layer which essentially allows a mathematical function of choice to be a applied to the inputs of that layer. These samples were the decoder inputs and the decoder outputs were the 100 dimension vectors attempting to best reconstruct the encoder inputs which were the network's best guess at the density profile for that particular z encoding. The reconstructions were incentivised by maximising the likelihood of finding particles on sites where they were in the snapshots 15

It should be noted that 1D convolutional layers were also trialled instead of dense layers. Generally these would better preserve the local structure of the input data going forward through the network [26] which is why convolutional neural networks (CNNs) are often used when the inputs are images of handwritten digits or faces for example. However, unlike images, correlation in local structure between data points within a batch is not present in the TASEP data so the convolutional layers showed no improvement over the dense layers. Dropout layers were also trialled initially but also showed no improvement but would have been used if there was any evidence of overfitting to the trinaing data.

VAEs were compiled with the VAE loss function 15 and the Adam optimiser [38] which is one of the most popular stochastic gradient descent algorithms. Batches of 32 snapshots were used to train the networks over 10 epochs with 20% of the training data used for validation. The loss histories throughout training for the mean field point and the coexistance point are shown in figure 13. In both cases the training loss plateaus suggesting the network has converged on some solution and the validation losses are as low as the training losses so there is no evidence of overfitting to the training dataset. The network fed mean field inputs was fully trained after a single epoch due to the low input diversity. The network fed coexistance snapshots took several epochs to train.



(a) (b)

Figure 12. (a) shows the network architecture for encoder half of the VAE as generated by the Keras plot_model function. In brackets on the right of the layers are the shapes so the $L = 100$ inputs are encoded, via dense layers onto a one dimensional latent distribution and a final lambda layer is used to sample the latent distribution. Note, the expected batch size for each of the layers is 'None' which meant that it could be later specified at the start of training. (b) shows the decoder outputting an $L = 100$ reconstruction from the sample.

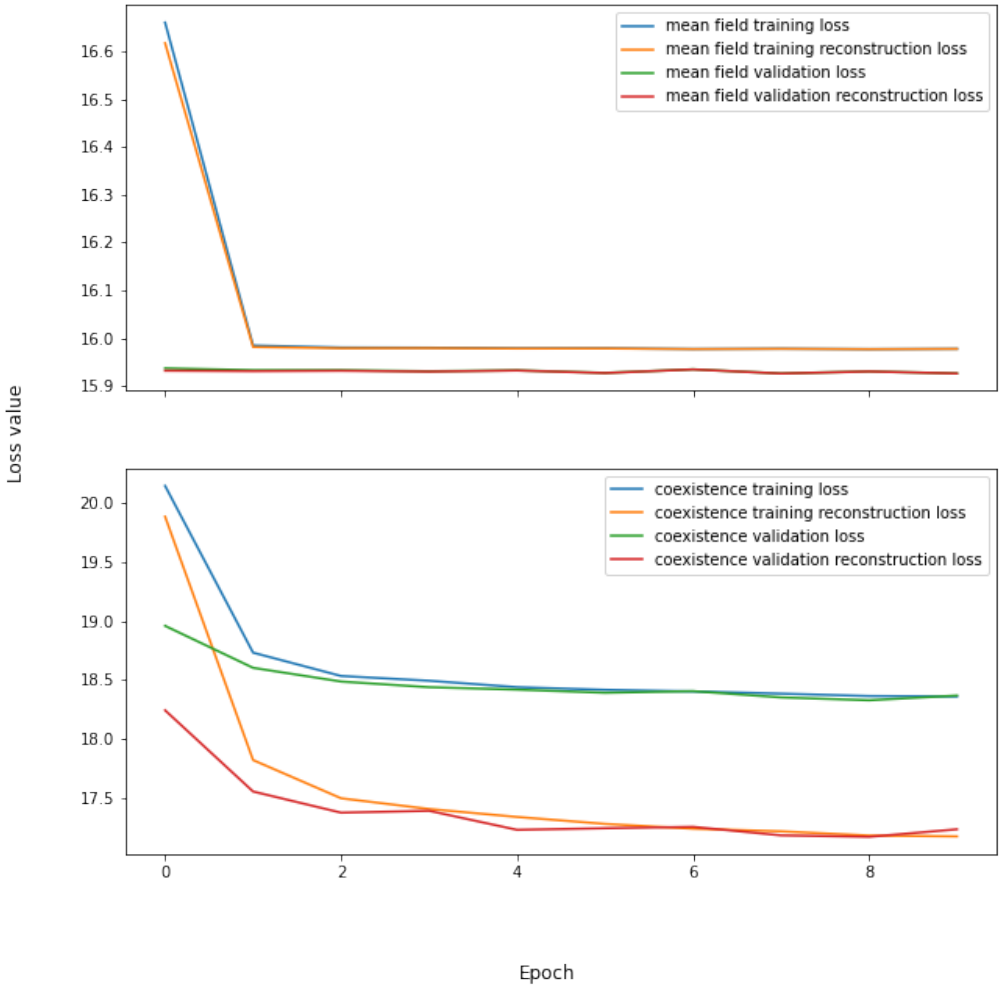


Figure 13. The losses against epoch number. Above is the curve for the mean field point which trains quickly. Below that is the curve for the coexistance point which looks more like a typical ML loss decay curve. Validation losses are similar to training losses so there is no evidence of overfitting.

VAEs were built and compiled in Python with the TensorFlow library using Keras, the high-level functional API, to make the process accessible and intuitive. The VAE objects inherited from the Keras 'model' object which gave access to built in methods which automated much of the training process.

3.4 Measuring Complexity

At this stage, there were latent space distributions $q(\boldsymbol{z}|\boldsymbol{x})$ for a point on the mean field line and a point on the coexistence line (training is finished so the θ and φ subscripts have been dropped from now on for simplicity). Values of z could be sampled from said distributions and decoded to generate density profiles $\rho = (\rho_1, \rho_2, ..., \rho_L)$. Actual TASEP configurations $\tau = (\tau_1, \tau_2, ..., \tau_L)$ were given by placing a particle on site i with probability ρ_i and leaving it unoccupied with probability $(1 - \rho_i)$. The probability of $P(\tau|\rho)$ was given by

$$P(\tau|\rho) = \prod_i \rho_i^{\tau_i}(1-\rho_i)^{1-\tau_i} \tag{17}$$

This approach interprets the output of the decoder as a 'mean-field-like' distribution for each site. This is because we are ignoring correlations by setting $\langle \tau_i \tau_j \rangle = \langle \tau_i \rangle \langle \tau_j \rangle$ ie. particles are placed on sites by their own Bernoulli distribution with probability ρ_i independent of which other sites are to be occupied or not. The overall distribution of configurations $P(\tau)$ was given by $\int dz\, q(z|x)P(\tau|\rho(z))$ - the expected value with respect to the latent distribution.

To investigate the latent space for different point on the phase diagram, two new measures of 'complexity' are proposed. The first defined as

$$\begin{aligned} Complexity_1 &= \int dz\, q(z|x) D_{KL}(P(\tau) \,||\, P(\tau|\rho(z))) \\ &= \int dz\, q(z|x) \sum_\tau P(\tau) \log\left(\frac{P(\tau)}{P(\tau|\rho(z))}\right) \\ &= \sum_\tau P(\tau)\log(P(\tau)) - \int dz\, q(z|x) \sum_\tau P(\tau)\log(P(\tau|\rho(z)) \end{aligned}$$

broadly asks on average over the latent distribution, how much more information is needed to specify the overall distribution $P(\tau)$ relative to the mean-field-like distribution of configurations from the decoder $P(\tau|\rho(z))$. The second new complexity measure defined as

$$
\begin{aligned}
Complexity_2 &= \int dz\, q(z|x) D_{KL}(P(\tau|\rho(z)) \,||\, P_{ref}(\tau)) \\
&= \int dz\, q(z|x) \sum_{\tau} [P(\tau|z) \log P(\tau|z) - P(\tau|z) \log P_{ref}(\tau)] \\
&= \int dz\, q(z|x) \sum_{i} [\rho_i \log \rho_i + (1-\rho_i) \log(1-\rho_i)] \\
&\quad - \sum_{i} [\bar{\rho}_i \log\ \bar{\rho}_i + (1-\bar{\rho}_i) \log(1-\bar{\rho}_i)]
\end{aligned}
$$

where $\log P(\tau|\rho) = \sum_i \log[\rho_i^{\tau_i}(1-\rho_i)^{1-\tau_i}]$, $\sum_\tau P(\tau|z) = 1$ and $\int dz\, q(z|x) = 1$ have been used. $Complexity_2$ broadly asks on average over the latent distribution, how much extra information about correlations is included in $P(\tau|\rho(z))$ relative to a reference mean-field distribution $P_{ref}(\tau)$ which by definition ignores correlations. Where $P_{ref}(\tau)$

$$
P(\tau) = \prod_i \bar{\rho}_i^{\tau_i} (1-\bar{\rho}_i)^{1-\tau_i} \tag{18}
$$

where $\bar{\rho}_i = \int dz\, q(z|x) \rho_i(z)$. In addition to these two complexity measures, $KLcomplexity = D_{KL}(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z}))$ post training measure how successful the training was at making the latent distribution close to a standard Gaussian. This measures how diverse looking the snapshots were because a set of snapshots where every datapoint looks similar like the mean-field line data, can be easily reconstructed meaning datapoints could be encoded anywhere on the standard Gaussian. In contrast a dataset of diverse data like the coexistence line data (a variety of shock positions) cannot all be reconstructed the same so should be more difficult to encode like a standard Gaussian. Therefore, it is expected that all three complexity measures $KLcomplexity$, $Complexity_1$ and $Complexity_2$ would be greater for the coexistence data than for the mean-field line data and serve as a test to see how the distribution learned by the VAE compares to what is known about the exact distribution.

These complexity measures were calculated using Monte Carlo to do any integrals and sums numerically by taking 100 Monte Carlo samples each time. This whole process from the start of NN training was repeated 10 times and average complexities were calculated with error given by the standard error of the mean.

This comparison of complexities for the mean-field line and the coexistence line phase diagram points was the central result of the project but the same method was used to calculate complexities for other points on the phase diagram as well as the two extensions of the TASEP model 2.2.2. VAEs were also fed data containing a balanced mixture of snapshots from four points along the mean-field line - ($\alpha = 0.2$, $\beta = 0.8$), ($\alpha = 0.4$, $\beta = 0.6$), ($\alpha = 0.6$, $\beta = 0.4$), ($\alpha = 0.8$, $\beta = 0.2$) and then four points along the coexistence line - ($\alpha = \beta = 0.1, 0.2, 0.3, 0.4$). A qualitative analysis of the latent space was carried out for these two datasets.

4 Results and Discussion

4.1 Determination of Complexities

The complexities for the point on the mean-field line ($\alpha = 0.2$, $\beta = 0.8$) and the point on the coexistence line ($\alpha = \beta = 0.2$) are shown in figure 14.

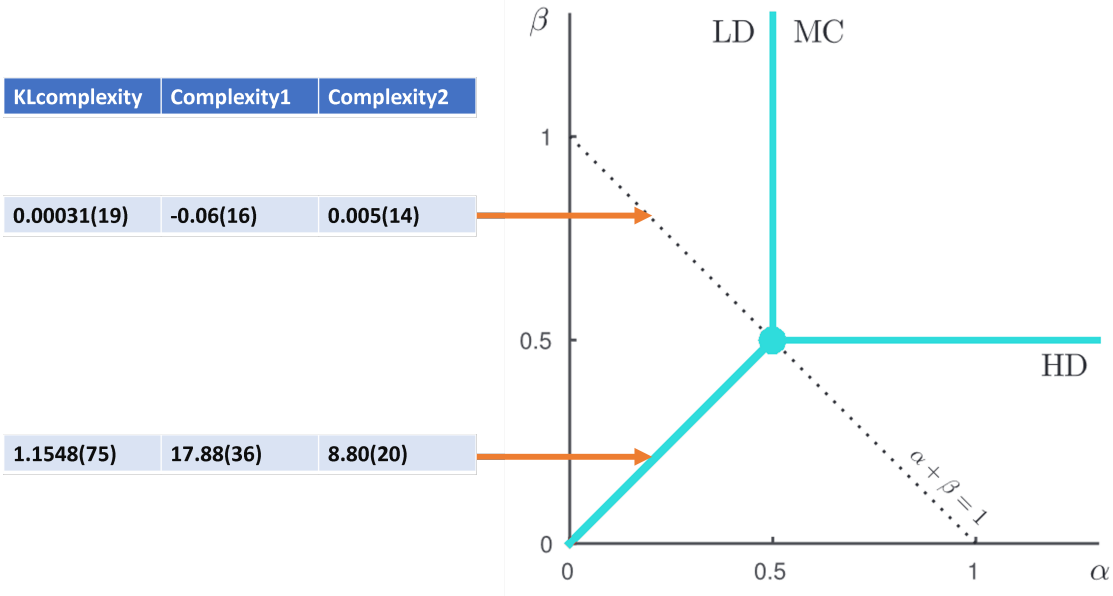


Figure 14. The complexities measured for the point on the mean-field line ($\alpha = 0.2$, $\beta = 0.8$) and the point on the coexistence line ($\alpha = \beta = 0.2$).

We see that all three are higher for the coexistence point as expected. The KLcomplexity result is evidence that the VAE has learned that configurations form the coexistence point have an extra degree of freedom, the shock position. $Complexity_1$ and $Complexity_2$ results are evidence that an attempt at a mean-field-like distribution for these configurations misses a lot of the information stored in the true distribution. All complexities are almost zero for the mean-field point as expected so confirms that a mean-field-like theory well describes the distribution for that region of the phase diagram.

However, complexities for the rest of the phase diagram which included the three bulk phases as well as both other phase boundaries (LD-MC and HD-MC) were all similar to those found for the mean-field point - although these second order phase transitions unlike the coexistence line which is first order so are not necessarily expected to show similar complexity. This is a somewhat unexpected result but it may be that complexities were not calculated to a high enough precision to pick up on these more subtle correlations. This is evidenced by that fact there were sometimes negative values estimated for $Complexity_1$ and $Complexity_2$ even after taking the mean of 10 measurements. These are KL-divergences so should always be positive which is a red flag but the error on these measurements was large enough that one error bar would shift the values above zero. Therefore, it could be argued that all complexities

off of the coexistence line are zero. This could be verified by calculating the integrals and sums in $Complexity_1$ and $Complexity_2$ with greater numbers of samples. However, it would be discouraged to attempt this in Python as the calculations were already very slow but a complied language C++ would likely speed up the process.

The VAE only considers the distribution to be complex if correlations present are significant enough to cause non-static density profiles like along the coexistence line. There are two possible implications from this. First that the correlations elsewhere on the phase diagram are not very significant and a mean-field-like distribution is a good approximation for the true distribution. Alternatively, the encoding process into the condensed representation in latent space dilutes correlations found in the original snapshots.

The complexities of the extension of the TASEP model for *S. cerevisiae*, also known as baker's yeast, genes YGR025W and YDR355C were all roughly zero like the majority of the phase diagram. This was expected given the above results because ribosomes are very unlikely to ever meet each other on the lattice [23] resulting in static density profiles. The model with exponentially distributed waiting times gave $KLcomplexity = 0.5$, $Complexity_1 = 5.9$ and $Complexity_2 = 2.8$ which was unexpectedly less than those for the coexistence point with the same ($\alpha = \beta = 0.2$). However, it is not clear whether the correlation time of 9 time units is enough to generate uncorrelated enough snapshots to properly randomly sample the underlying distribution of configurations. Also, no error is provided for these complexity measurements. With more time, the same method 3.2 could be used to verify the correlation time for these models and repeat complexity measurements could be taken to obtain the errors.

4.2 Exploring Latent Space

For a VAE trained on a single mean-field point and a single coexistence point, the latent space distributions look very close to the standard Gaussian as expected, see figure 15, so the encoder was working as intended. Only a one component $\boldsymbol{z}$ was required to recreate the proper density profiles.

For the dataset containing snapshots from four points along the mean field line, the latent distribution is shown in figure 16a where we see the VAE has learned to group the snapshots into the four different points on the phase diagram. The bulk density of the original data mapped to points in latent space was plotted against z in figure 16b. We see that z highly correlates with the bulk density. Figure 17 shows the decoded density profiles of the four peaks. The density profiles agree with what is expected for these (α, β) pairs [21].

For the dataset containing snapshots from four points along the coexistence line line, the latent space distribution and decoded density profiles are shown in figure 18. To achieve these density profiles, a component z was required. The latent space is best explained in polar coordinates. The distance from the origin encodes the densities at the boundaries while the angle from the positive x-axis encodes how far along the lattice the shock is positioned.

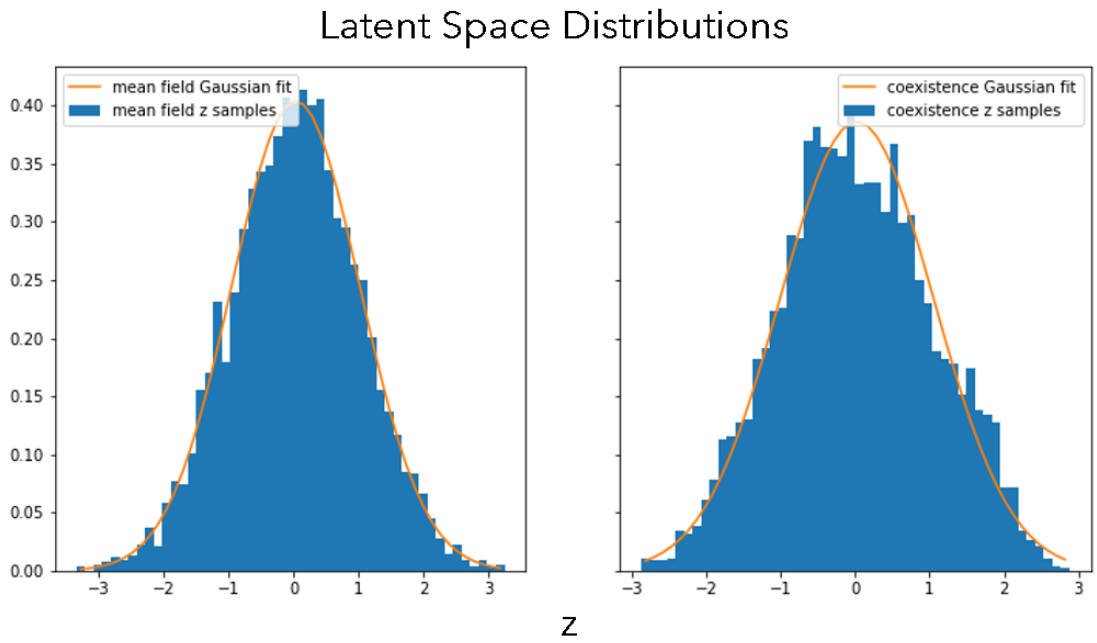
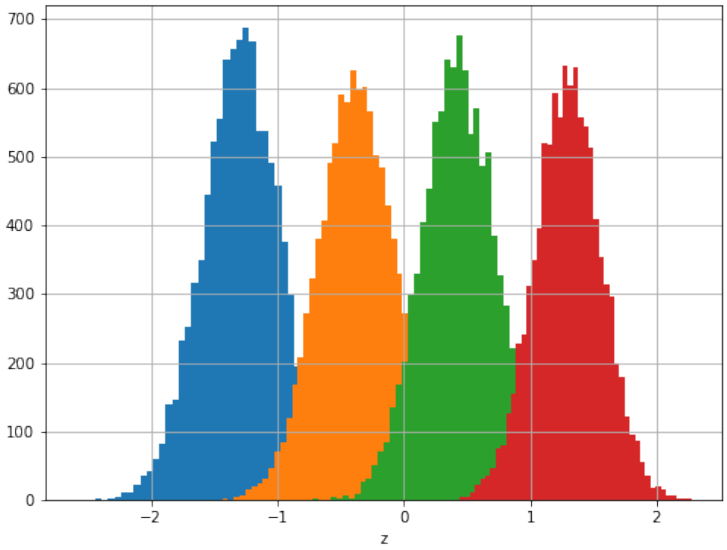


Figure 15. Left is the latent space distribution for the mean-field point and right is the latent space distribution for the coexistence point. Both are close to the standard Gaussian. A maximum likelihood Gaussian fit is overlayed on both.

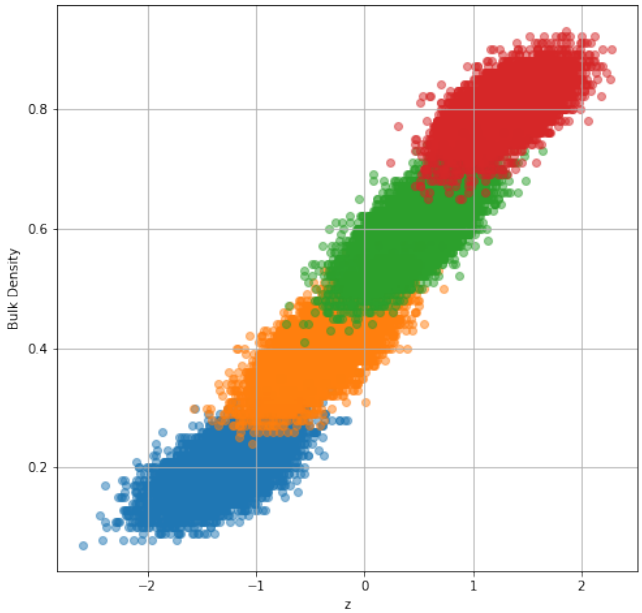
5 Conclusion

By training a NN on TASEP snapshots, the effectiveness of NNs for learning nonequilibrium distributions has been evaluated. By taking mean-field-like interpretation of the output, the NN understood that the coexistence line is more complex than the mean-field line and the rest of the phase diagram. By decoding the latent space, it is clear that the NN learns to summarise the TASEP snapshots by physical variables such as density and shock position. This is an analogous result to previous work on equilibrium systems - Ising and XY models - where the latent variable corresponds to magnetisation. With more time, these results could be confirmed by taking more Monte Carlo samples when calculating the complexities and taking more overall repeats. Although some pilot results for complexities of extensions to the TASEP model were obtained including one for mRNA translation, further work is needed to verify these results.

Overall, features of a nonequliibrium distribution have been learned using a modern ML approach which is important as a bridge between physics and data science. As ML is a rapidly growing field, future work could not only apply the VAE to nonequilibrium systems other than the TASEP but a different generative NN could be used all together such as the generative adversarial network (GAN) for example.



(a)



(b)

Figure 16. (a) The latent distribution for data containing snapshots from four points on the mean-field line shuffled together. Red points were mappings from the ($\alpha = 0.8$, $\beta = 0.2$), green - ($\alpha = 0.6$, $\beta = 0.4$), orange - ($\alpha = 0.4$, $\beta = 0.6$), blue - ($\alpha = 0.2$, $\beta = 0.8$). (b) Bulk density against z shows correlation.

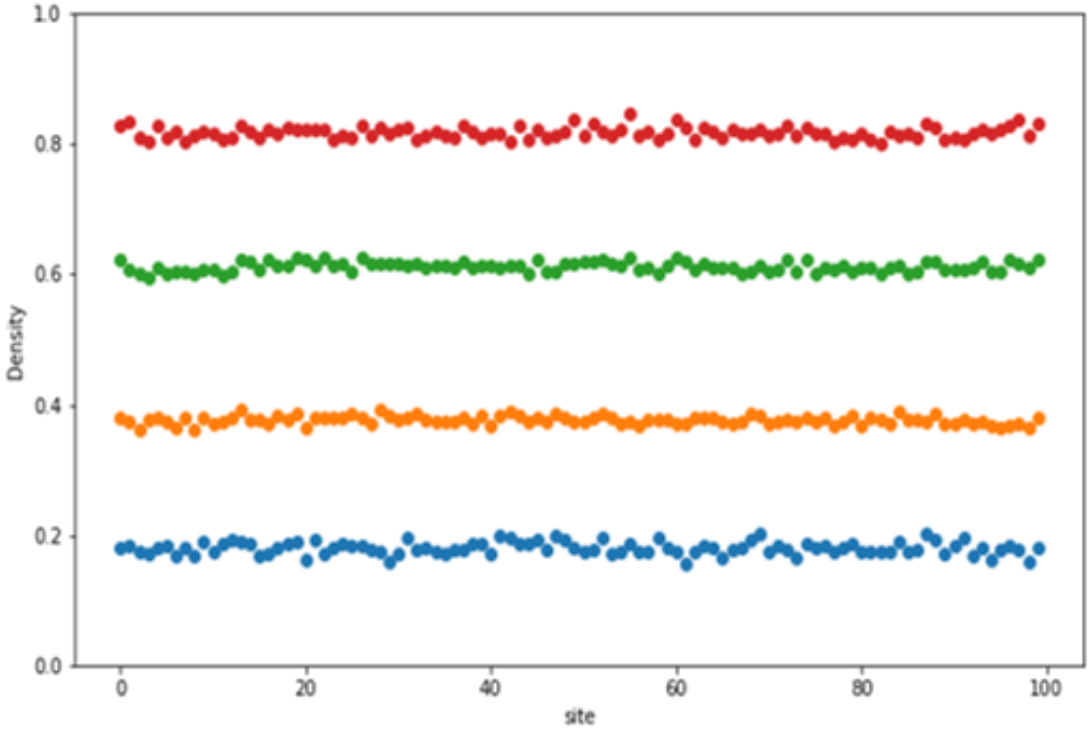


Figure 17. The decoded density profiles for peaks in the latent space red - ($\alpha = 0.8$, $\beta = 0.2$), green - ($\alpha = 0.6$, $\beta = 0.4$), orange - ($\alpha = 0.4$, $\beta = 0.6$), blue - ($\alpha = 0.2$, $\beta = 0.8$). The densities profiles look as expected.

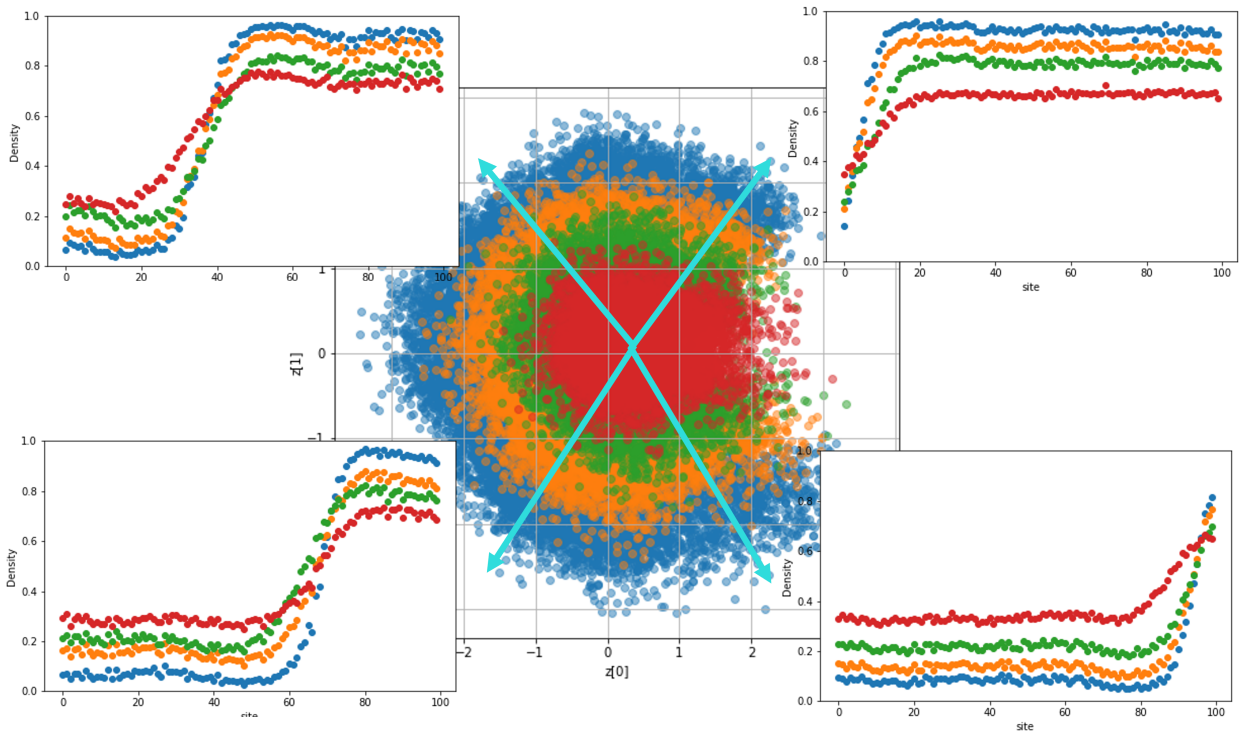


Figure 18. The latent space for the dataset with four coexistence line points. Red - ($\alpha = \beta = 0.4$), green - ($\alpha = \beta = 0.3$), orange - ($\alpha = \beta = 0.2$), blue - ($\alpha = \beta = 0.1$). The subgraphs show one decoded density profile for each colour along the cyan arrow and it is observed that different arrows give different shock positions.

References

[1] F. (Franz) Mandl. *Statistical physics / F. Mandl.* The Manchester physics series. Wiley, Chichester, second edition. edition, 1988.

[2] R A Blythe and M R Evans. Nonequilibrium steady states of matrix-product form: a solver's guide. *Journal of Physics A: Mathematical and Theoretical*, 40(46):R333–R441, oct 2007.

[3] Murray Gell-Mann and Constantino Tsallis. *Nonextensive Entropy: Interdisciplinary Applications*. Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, Incorporated, Cary, 2004.

[4] Carolyn T. MacDonald, Julian H. Gibbs, and Allen C. Pipkin. Kinetics of biopolymerization on nucleic acid templates. *Biopolymers*, 6(1):1–25, 1968.

[5] Leah B. Shaw, R. K. P. Zia, and Kelvin H. Lee. Totally asymmetric exclusion process with extended objects: A model for protein synthesis. *Phys. Rev. E*, 68:021910, Aug 2003.

[6] Roberto Livi and Paolo Politi. *Nonequilibrium Statistical Physics: A Modern Perspective*. Cambridge University Press, 2017.

[7] Number of papers submitted each year, Journal of Machine Learning Research. `https://www.jmlr.org/stats.html`. Accessed 20th March 2022.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[10] Indranil Bose and Radha K. Mahapatra. Business data mining — a machine learning perspective. *Information & Management*, 39(3):211–225, 2001.

[11] Lc0 training, Leela Chess Zero. `https://lczero.org/blog/2018/10/lc0-training/`. Accessed 25th March 2022.

[12] Wenjian Hu, Rajiv R. P. Singh, and Richard T. Scalettar. Discovering phases, phase transitions, and crossovers through unsupervised machine learning: A critical examination. *Phys. Rev. E*, 95:062122, Jun 2017.

[13] Sebastian J. Wetzel. Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders. *Phys. Rev. E*, 96:022140, Aug 2017.

[14] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[15] Pankaj Mehta and David J Schwab. An exact mapping between the variational renormalization group and deep learning. *arXiv preprint arXiv:1410.3831*, 2014.

[16] Paul A. Gagniuc. *Markov chains : from theory to implementation and experimentation / Paul A. Gagniuc.* John Wiley & Sons, Hoboken, NJ, 2017.

[17] Frank Spitzer. Interaction of markov processes. *Advances in Mathematics*, 5(2):246–290, 1970.

[18] Juraj Szavits-Nossan. *Phase transitions in driven diffusive systems far from equilibrium.* PhD thesis, University of Zagreb, 10 2011.

[19] Carolyn T. MacDonald, Julian H. Gibbs, and Allen C. Pipkin. Kinetics of biopolymerization on nucleic acid templates. *Biopolymers*, 6(1):1–25, 1968.

[20] Michael E. Cates and Elsen Tjhung. Theories of binary fluid mixtures: from phase-separation kinetics to active emulsions. *Journal of Fluid Mechanics*, 836:P1, 2018.

[21] Bernard Derrida, Martin Evans, V Hakim, and Vincent Pasquier. Exact solution of a 1d asymmetric exclusion model using a matrix formulation. *Journal of Physics A: Mathematical and General*, 26:1493, 01 1999.

[22] Anthony J Wood, Richard A Blythe, and Martin R Evans. Rényi entropy of the totally asymmetric exclusion process. *Journal of Physics A: Mathematical and Theoretical*, 50(47):475005, nov 2017.

[23] Luca Ciandrini, Ian Stansfield, and M. Carmen Romano. Ribosome traffic on mrnas maps to gene ontology: Genome-wide quantification of translation initiation rates and polysome size regulation. *PLOS Computational Biology*, 9(1):1–10, 01 2013.

[24] Translation: Dna to mrna to protein, Scitable by Nature Education. `https://www.nature.com/scitable/topicpage/translation-dna-to-mrna-to-protein-393/`. Accessed 27th March 2022.

[25] T. M. Cover. *Elements of information theory / Thomas M. Cover, Joy A. Thomas.* Wiley-Interscience, Hoboken, N.J, second edition. edition, 2006.

[26] Mehta P., Bukov M., Wang C. H., Day A. G., Richardson C., Fisher C. K., and Schwab J. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, 2019.

[27] Reducelronplateau, Keras Documentation. `https://keras.io/api/callbacks/reduce_lr_on_plateau/`. Accessed 25th March 2022.

[28] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[30] Xugang lu, Yu Tsao, Shigeki Matsuda, and C. Hori. Speech enhancement based on deep denoising auto-encoder. *Proc. Interspeech*, pages 436–440, 01 2013.

[31] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[32] Gillespie D. T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[33] Kierzek A. M. Stocks: Stochastic kinetic simulations of biochemical systems with gillespie algorithm. *Bioinformatics*, 18(3):470–481, 2002.

[34] Melissa E O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Transactions on Mathematical Software*, 2014.

[35] Binder K. and Heermann D. *Monte Carlo Simulation in Statistical Physics : An Introduction*. Springer, Heidelberg, 5th edition, 2010.

[36] Gier J. and Essler F. H. L. Exact spectral gaps of the asymmetric exclusion process with open boundaries. *Journal of Statistical Mechanics: Theory and Experiment*, 2006(12):P12011–P12011, dec 2006.

[37] iminuit documentation, iminuit. `https://iminuit.readthedocs.io/en/stable/`. Accessed 25th March 2022.

[38] Kingma D. and Ba J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.