
Approximate sampling and estimation of partition functions using neural networks

George T. Cantwell

Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM, 87501
gcant@umich.edu

Abstract

We consider the closely related problems of sampling from a distribution known up to a normalizing constant, and estimating said normalizing constant. We show how variational autoencoders (VAEs) can be applied to this task. In their standard applications, VAEs are trained to fit data drawn from an unknown and intractable distribution. We invert the logic and train the VAE to fit a simple and tractable distribution, on the assumption of a complex and intractable latent distribution, specified up to normalization. This procedure constructs approximations without the use of training data or Markov chain Monte Carlo sampling. We illustrate our method on three examples: the Ising model, graph clustering, and ranking.

1 Background

Many problems that arise in statistics, combinatorics, and physics can be understood through the lens of an appropriate probability distribution, together with its normalizing constant. To study objects \mathbf{x} , we consider the distribution $P(\mathbf{x}) = f(\mathbf{x})/\mathcal{Z}$, where $\mathcal{Z} = \sum_{\mathbf{x}} f(\mathbf{x})$.

For example, when considering a combinatorial problem we can set $f(\mathbf{x}) = 1$ if a set of constraints is satisfied and $f(\mathbf{x}) = 0$ otherwise. Then, \mathcal{Z} simply counts the total number of solutions to the constraints. In a physics setting, we could have $f(\mathbf{x}) = e^{-\beta H(\mathbf{x})}$, where $H(\mathbf{x})$ is an energy function and β the inverse temperature. The resulting distribution is known as the canonical distribution, and arises when a system is in thermal equilibrium with its surroundings. The quantity \mathcal{Z} is called the *partition function*, and encodes the thermodynamic properties of the system [1]. In a statistical inference problem we can set $f(\mathbf{x}) = P(\mathbf{x}, \mathcal{D})$, where \mathcal{D} is the observed data. The resulting distribution $f(\mathbf{x})/\mathcal{Z}$ is the posterior distribution for \mathbf{x} . The quantity $\mathcal{Z} = P(\mathcal{D})$ is known as the *model evidence* or *marginal likelihood* and is important for rigorous model selection [2]. Understanding such distributions, and gaining a handle on \mathcal{Z} , is a perennial problem across computer science, statistics, and physics.

Computing \mathcal{Z} exactly is generally difficult – even for many of the simplest examples the problem is #P-hard [3]. The naive approach is to enumerate all exponentially many \mathbf{x} , and directly sum over this set; #P-hardness means there is unlikely to be any substantially faster method than this. But while we usually cannot compute \mathcal{Z} exactly, it may still be possible to make useful approximations.

A standard approach for approximating $\mathcal{Z} = \sum_{\mathbf{x}} f(\mathbf{x})$ is to use Markov chain Monte Carlo (MCMC) [4, 5, 6]. This requires constructing a Markov chain that has $P(\mathbf{x}) = f(\mathbf{x})/\mathcal{Z}$ as its equilibrium distribution, and then simulating the process. While finding such a Markov chain can be straightforward, it often turns out that the process takes exponential time to approach its equilibrium distribution. If the chain takes exponential time to get close to the equilibrium distribution, it is ultimately little improvement over the naive but exact method. Even when Markov chains do mix in polynomial time, they may still scale with a large power of the problem size, and thus large problems are out of reach in practice.

The contribution of this paper is a new method for approximating \mathcal{Z} and sampling $\mathbf{x} \sim f(\mathbf{x})/\mathcal{Z}$, using neural networks. This provides an alternative to MCMC for practical problems in Bayesian statistics and physics.

The neural network architecture we employ is equivalent to a variational autoencoder (VAE) [7, 8]. An autoencoder is a neural network that is trained to map objects to themselves, i.e., to represent the identity function. To make this non-trivial, constraints are added to the intermediate representations. An autoencoder is constructed from two parts, known as the encoder and the decoder. In a *variational* autoencoder, the encoder maps complicated data, such as photographs of human faces, to a simple latent distribution, such as a (multivariate) normal or Bernoulli distribution. The decoder takes points from the latent space and attempts to reconstruct a posterior distribution for the data (e.g. a distribution over faces). The basic setup takes data \rightarrow latent distribution \rightarrow reconstructed data distribution, where the input data should match the output distribution.

We will essentially reverse the logic of VAEs. As discussed, in the standard application of VAE we train the network to map samples from an intractable data distribution to a simple latent distribution, and then back again. Here, in contrast, we will train the VAE to map a simple and tractable distribution to a more complex one. The only input to the training procedure will be a routine to compute $f(\mathbf{x})$. From this function alone (i.e., without any samples to be used as training data), the VAE will attempt to find a good approximation for $\mathcal{Z} = \sum_{\mathbf{x}} f(\mathbf{x})$, and will construct a sampler for $\mathbf{x} \sim f(\mathbf{x})/\mathcal{Z}$.

At a conceptual level, this procedure is similar to normalizing flows: we take a simple distribution and learn a mapping to a more interesting one [9]. Because normalizing flows use the Jacobian change-of-variables transformation, they require every step in the neural network to be a bijection (invertible) and are not readily applied to discrete distributions. To circumvent this issue, Ref. [10] advocates training two networks with normalizing flows. The approach is conceptually close to ours although we do not use normalizing flows, and so our networks are considerably less constrained. In particular, the dimensions of the inputs and outputs in our framework do not need to match, which allows us to fit low entropy but high dimensional distributions by using high entropy but low dimensional inputs – indeed, this is the key insight that makes VAEs useful. Further, by adapting the framework of autoencoders, all of the flexibility and advances in this field can be incorporated into the approximations. Finally, there is also a conceptual similarity between our goals and those in [11], which uses autoregressive networks. Both methods attempt to fit distributions, although again, ours is both more simple and more flexible, since it does not require autoregressive constraints, and the networks can thus be much smaller. In fact, as we shall see, our method can perform well in certain regimes with only a single binary input/output neuron (in which case, optimization can actually be done with a pen and paper).

In the remaining sections of this paper we first derive the required machinery, and then test it against three example problems.

2 Variational approximations and field theories

Variational methods turn the problem of estimating \mathcal{Z} into an optimization problem. We note that

$$\ln \mathcal{Z} = \ln \sum_{\mathbf{x}} f(\mathbf{x}) \geq \sum_{\mathbf{x}} R(\mathbf{x}) \ln \frac{f(\mathbf{x})}{R(\mathbf{x})} \tag{1}$$

where $R(\mathbf{x})$ is any arbitrary distribution. This inequality is a special case of Jensen’s inequality.

While the inequality in Eq. (1) is true for all $R(\mathbf{x})$, maximizing the right hand side with respect to R provides the best lower bound for $\ln \mathcal{Z}$. In fact, the unconstrained optimum is simply $R(\mathbf{x}) = P(\mathbf{x}) = f(\mathbf{x})/\mathcal{Z}$, in which case the inequality in Eq. (1) is saturated and $\ln \mathcal{Z}$ is computed exactly. Of course, this optimum is not practically useful, since by assumption $P(\mathbf{x})$ is difficult to work with. Instead we must restrict $R(\mathbf{x})$ to some sufficiently simple family of distributions, such as product distributions $R(\mathbf{x}) = \prod_i r_i(x_i)$.

In physics, this approach is known as a mean-field approximation [2]. If the $x_i \in \{\pm 1\}$ then we can write

$$R(\mathbf{x}) = \prod_i \frac{e^{x_i \phi_i}}{e^{\phi_i} + e^{-\phi_i}} \tag{2}$$

and ϕ_i is conceptualized as a field felt by x_i , characterizing the average (mean) effect of the rest of the system. Mean-field approximations are often effective but can be greatly inaccurate. Unfortunately, improving the approximation is often quite involved.

Moving beyond mean-field theory, we can allow the fields to fluctuate and develop what is known as a statistical field theory [12]. Equivalently, R is a mixture of product distributions, rather than a simple product distribution. Constructing such approximations is somewhat of an art, and often involves highly problem-specific insights (e.g. the Hubbard-Stratonovich transformation for the Ising model [13]).

Due to the complex and technical nature of developing statistical field theories, the approach is not nearly as widely applied as mean-field theory. For statistics problems in particular, where the precise details of the model usually change with each application, developing a statistical field theory is usually unreasonably difficult. Here we discuss a partial remedy to this.

2.1 Auxiliary variables, statistical fields

To make progress in our goal of constructing an improved approximation for $P(\mathbf{x}) = f(\mathbf{x})/\mathcal{Z}$, we introduce auxiliary variables \mathbf{y} . We couple the distributions of the variables \mathbf{x} and \mathbf{y} as

$$P(\mathbf{x}, \mathbf{y}) = P(\mathbf{x})Q(\mathbf{y}|\mathbf{x}) \quad (3)$$

where $Q(\mathbf{y}|\mathbf{x})$ is an arbitrary normalized distribution, and hence $\sum_{\mathbf{y}} Q(\mathbf{y}|\mathbf{x}) = 1$. Using the fact that Q is normalized we now invoke Jensen's inequality to find

$$\ln \mathcal{Z} = \ln \sum_{\mathbf{x}, \mathbf{y}} f(\mathbf{x})Q(\mathbf{y}|\mathbf{x}) \geq \sum_{\mathbf{x}, \mathbf{y}} P(\mathbf{y})R(\mathbf{x}|\mathbf{y}) \ln \frac{f(\mathbf{x})Q(\mathbf{y}|\mathbf{x})}{P(\mathbf{y})R(\mathbf{x}|\mathbf{y})} \quad (4)$$

for all distributions $P(\mathbf{y})$, $Q(\mathbf{x}|\mathbf{y})$, and $R(\mathbf{x}|\mathbf{y})$. Now, a double maximization of Eq. (4) with respect to both $Q(\mathbf{y}|\mathbf{x})$ and $R(\mathbf{x}|\mathbf{y})$ provides a lower bound for $\ln \mathcal{Z}$.

Equation (4) provides a framework to estimate $\ln \mathcal{Z}$. In addition, it also suggests a method to sample $\mathbf{x} \sim P(\mathbf{x})$. The Q and R that maximize Eq. (4) satisfy

$$P(\mathbf{x})Q(\mathbf{y}|\mathbf{x}) = P(\mathbf{y})R(\mathbf{x}|\mathbf{y}). \quad (5)$$

This means that if we were to sample \mathbf{y} from $P(\mathbf{y})$, and then \mathbf{x} from $R(\mathbf{x}|\mathbf{y})$, the \mathbf{x} we end up with is distributed according to $P(\mathbf{x})$. The problem, of course, is to find good Q and R .

As was the case for Eq. (1), the true optimum that saturates the inequality in Eq. (4) is probably useless in practice. Rather, we must restrict ourselves to cases where Q and R are tractable, such as a product of conditional distributions

$$R(\mathbf{x}|\mathbf{y}) = \prod_i r_i(x_i|\mathbf{y}), \quad (6)$$

and we let $P(\mathbf{y})$ be a simple and tractable distribution, for example a normal or Bernoulli distribution.

Returning to the case where $x_i \in \{\pm 1\}$ we can, without loss of generality, write the product as

$$R(\mathbf{x}|\mathbf{y}) = \prod_i \frac{e^{x_i \phi_i(\mathbf{y})}}{e^{\phi_i(\mathbf{y})} + e^{-\phi_i(\mathbf{y})}}. \quad (7)$$

The functions $\phi_i(\mathbf{y})$ now act as fields, although they fluctuate because \mathbf{y} is a random variable; this is a statistical field theory. We shall represent $\phi(\mathbf{y})$ using a neural network. The expectation of Eq. (7) over the auxiliary variables \mathbf{y} provides an approximation for the true $P(\mathbf{x})$. How we proceed now depends on our choice of distribution $P(\mathbf{y})$.

A convenient choice for $P(\mathbf{y})$ is that each y_i is an independent coin flip: +1 or -1 with probability 1/2. In this case we can use the same ansatz for Q as we did for R , Eq. (7),

$$Q(\mathbf{y}|\mathbf{x}) = \prod_i \frac{e^{y_i \theta_i(\mathbf{x})}}{e^{\theta_i(\mathbf{x})} + e^{-\theta_i(\mathbf{x})}}. \quad (8)$$

The function $\theta(\mathbf{x})$ can be represented in a neural network.

Alternatively, we could choose $P(\mathbf{y})$ to be a product of independent normal distributions, in which case the posterior $Q(\mathbf{y}|\mathbf{x})$ will (hopefully) be approximately normal about some mean $\boldsymbol{\mu}(\mathbf{x})$ with covariance matrix $\boldsymbol{\Sigma}(\mathbf{x})$. Specifically, we assume $Q(\mathbf{y}|\mathbf{x}) = \mathcal{N}_{\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x})}(\mathbf{y})$, with \mathcal{N} being the density function for the normal distribution. Optimal choices for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are

$$\boldsymbol{\mu}(\mathbf{x}) = \arg \max_{\mathbf{y}} R(\mathbf{x}|\mathbf{y}) \quad \text{and} \quad \boldsymbol{\Sigma}^{-1}(\mathbf{x}) = - \left. \frac{\partial^2 \ln R(\mathbf{x}|\mathbf{y})}{\partial y_i \partial y_j} \right|_{\mathbf{y}=\boldsymbol{\mu}(\mathbf{x})}, \quad (9)$$

which is equivalent to the Laplace approximation [2, 14] of Bayesian statistics. Two problems arise: (i) the $\arg \max_{\mathbf{y}}$ operation is expensive and (ii) we do not have easy access to gradient information. To fix these problems we can abandon Eqs. (9) and instead train a second neural network to represent $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\Sigma}(\mathbf{x})$.

Putting everything together, we maximize Eq. (4) using stochastic optimization. To do this we draw \mathbf{y} from $P(\mathbf{y})$ and then \mathbf{x} from $R(\mathbf{x}|\mathbf{y})$. When \mathbf{x} is discrete, as it often will be, we can apply the Gumbel softmax trick [15, 16] to ensure we can take derivatives through the sampling process. Each term in Eq. (4) is now simple to compute, and so are its derivatives, and we have a complete system.

We simultaneously train two networks. The first network represents $R(\mathbf{x}|\mathbf{y})$ and transforms each specific value of the auxiliary variables \mathbf{y} into a distribution over the variables of interest, \mathbf{x} . The second network represents $Q(\mathbf{y}|\mathbf{x})$ and does the reverse: it takes specific values of \mathbf{x} and maps them to a distribution over auxiliary variables \mathbf{y} . As discussed this is equivalent to a VAE, except the role of the encoder and decoder have been conceptually reversed; unlike the standard application, we do not train the neural networks on samples \mathbf{x} from the intractable distribution. In fact, the networks are trained to match the distribution of \mathbf{x} without ever seeing a sample of \mathbf{x} .

3 Examples

3.1 Square lattice Ising model

The Ising model is one of the most widely studied models from physics, and has been applied well beyond physics [1, 17]. Originally formulated as a model for magnetism, each site (i.e. atom) has a small magnetic dipole caused by “spin” that either points up or down. Due to an exchange interaction (a quantum mechanical effect) it is energetically favorable for neighboring spins to align. The spin at site i is denoted $x_i \in \{\pm 1\}$ and the energy function is

$$H(\mathbf{x}) = -J \sum_{\langle i,j \rangle} x_i x_j - h \sum_i x_i \quad (10)$$

where J is the coupling strength, $\langle i, j \rangle$ represent neighboring pairs, and h is an external magnetic field. When a majority of the spins are aligned, we observe a magnetic moment at macroscopic scales.

The model has been studied on a myriad of topologies, but the archetypal case is the two-dimensional square lattice. When $h = 0$ the square lattice Ising model has been solved exactly, both at finite size and in the limit of an infinite system [1, 3]. For $h \neq 0$ it remains an open problem.

So that we can compare our results to exact calculations we will set $h = 0$ and $J = 1$, in which case we have

$$f(\mathbf{x}) = e^{\beta \sum_{\langle i,j \rangle} x_i x_j}. \quad (11)$$

Where $\langle i, j \rangle$ represent neighboring pairs on the square lattice, and we wrap around at the boundaries so the the left-most sites are connected to the right-most, and likewise for the top and bottom.

We consider a system of $n = 64$ spins, and use simple architectures for the neural networks. Specifically, both neural networks have only a single hidden layer with 1024 units using the SELU activation function, and for the Gumbel softmax function, $\frac{e^{w_i/\tau}}{\sum_j e^{w_j/\tau}}$, we set $\tau = 1/16$. For all experiments we used the Adam optimizer with the default parameters in TensorFlow 2.8, and trained on a desktop computer with an Intel i5-7600K CPU, and no dedicated GPU.

For the auxiliary variables \mathbf{y} we used independent Bernoulli variables. It is not immediately obvious how many auxiliary variables are needed, and so we experimented with differing numbers. By Eq. (4), any number of auxiliary variables provides a bound; we experimented with different numbers (between 1 and 64) and took whichever estimate of $\ln \mathcal{Z}$ was largest.

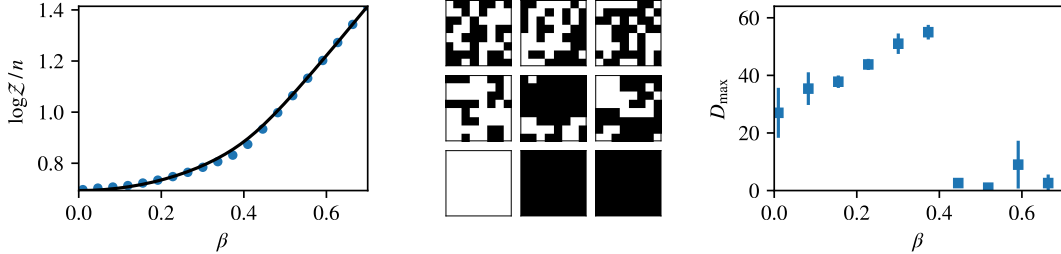


Figure 1: Experiments for the square lattice Ising model on n nodes. Left: estimates of $\frac{1}{n} \ln \mathcal{Z}$ for different values of β . The solid line shows the analytic result. Center: Random samples from the trained network, at three different temperatures, from top to bottom: $\beta = 0.01, 0.35, 0.7$. Qualitatively, these samples match expectations. At $\beta = 0.01$ we see a mostly random pattern, at $\beta = 0.35$ we see larger contiguous blocks, and at $\beta = 0.7$ we see a homogeneous system. Right: Networks were trained with differing sized input distributions, i.e. different numbers of auxiliary variables. Since the resulting approximation of $\ln \mathcal{Z}$ is a rigorous lower bound, regardless of the number of auxiliary variables, we ask which value of D – the number of auxiliary variables – gives the largest lower bound. For β larger than the critical value, the best approximations have a small number of auxiliary variables.

The results of our experiments are presented in Fig. 1. Once training is finished, we have estimates of the partition function along with an (approximate) sampler for the model. To generate states we first draw \mathbf{y} from $P(\mathbf{y})$ and then \mathbf{x} from $R(\mathbf{x}|\mathbf{y})$. Each new sample is independent and generated in time proportional to the size of the neural network. We can use these samples to probe arbitrary properties of the model.

The standard method for sampling from the Ising model is to use Markov chain Monte Carlo methods [5, 6]. Considerable effort has been spent to develop efficient Markov chains, although mixing times are still generally worse than linear in the system size.

We should also contrast the approach presented here to some previous studies of the Ising model using autoencoders, which also have the ability to approximately sample [18, 19, 20]. The key difference is that in these works, the Ising model is first simulated using Markov chain Monte Carlo methods, and then the results of the simulations fit using an autoencoder. The resulting fit is, of course, limited by the quality of the initial samples. In other words, this approach requires that one already has a high-quality Monte Carlo sampler, whereas in our approach no input data are required – we fit the distribution directly.

3.2 Graph clustering and the Stochastic Block Model

The Stochastic Block Model (SBM) is a generative statistical model for the graph clustering problem, also known community detection [21, 22, 23].

The model generates networks on n nodes as follows. First, each node is randomly assigned to a community. Let x_i be the community assignment for node i . Once each node is assigned to a community, edges are created independently at random. An edge between node i and j exists with probability $\omega(x_i, x_j)$ – a symmetric function of the community membership of nodes i and j only.

The probability to generate graph G is

$$P(G|\mathbf{x}, \omega) = \prod_{i < j} \omega(x_i, x_j)^{G_{ij}} (1 - \omega(x_i, x_j))^{1 - G_{ij}} \quad (12)$$

where G_{ij} is an indicator variable for the existence of edge (i, j) .

To invert the generative model, and hence to find a good community assignment \mathbf{x} for a given graph G , we could simply maximize Eq. (12) with respect to both \mathbf{x} and ω . Each value x_i can be thought of as a parameter, and we are then finding the maximum likelihood community assignment. However, because there are n community parameters, i.e. the number of parameters is growing linearly with the size of the network, this approach is liable to over-fitting. A principled approach to inference is to instead maximize $P(G|\omega) = \sum_{\mathbf{x}} P(G, \mathbf{x}|\omega)$.

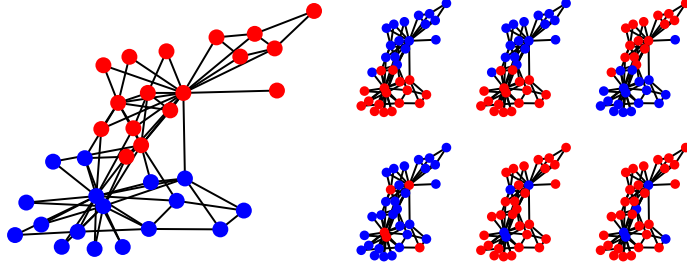


Figure 2: Karate club network. Left: network colored to match the “ground truth” communities, i.e. the factions. Right, top row: Samples from neural network at the factions local maximum. Right, bottom row: Samples from the neural network at the leaders-followers local maximum.

Assuming a uniform prior on \mathbf{x} over c different groups we have the posterior

$$P(\mathbf{x}|G, \omega) = \frac{c^{-n} \prod_{i < j} \omega(x_i, x_j)^{G_{ij}} (1 - \omega(x_i, x_j))^{1 - G_{ij}}}{\mathcal{Z}(\omega)} \quad (13)$$

and where $\mathcal{Z}(\omega) = P(G|\omega)$.

For the purpose of an example we will consider the case of two groups – a and b – with connection probabilities $\omega(a, a) = \omega(b, b) = \omega_{\text{in}}$ and $\omega(a, b) = \omega_{\text{out}}$. We take $f(\mathbf{x})$ to be the numerator of Eq. (13), and we consider the standard benchmark graph – the Karate Club graph [24].

The neural network architectures are unchanged from the previous example – 1024 units in a single hidden layer, with SELU activations, and again we set $\tau = 1/16$. For any specific ω we can then maximize Eq. (4) with respect to Q and R to estimate $\ln \mathcal{Z}(\omega)$. Better, we can allow ω to be another parameter in the model and thus optimize everything simultaneously.

The Karate Club graph [24] represents measured friendships among members of a university karate club. At a later date, the club fractured into two different clubs. The subsequent groups are often considered to represent a ground-truth on how the original graph should be divided.

Our experiments find a local maximum for $\mathcal{Z}(\omega) = P(G|\omega)$ in which $\omega_{\text{in}} > \omega_{\text{out}}$. We can sample \mathbf{x} by first sampling \mathbf{y} and then \mathbf{x} from $R(\mathbf{x}|\mathbf{y})$. This gives us independent samples of \mathbf{x} that are (approximately) distributed according to the posterior distribution $P(\mathbf{x}|G, \omega)$. We indeed find a distribution that is centered around the alleged ground-truth factions – see Fig. 2.

However, our experiments also find a second local maximum, in which $\omega_{\text{in}} < \omega_{\text{out}}$. That is to say, a situation in which the network is split into two groups but where connections within each group are sparse and between are frequent. This group also has an intuitive explanation: it is a leaders-followers dichotomy. In fact, the maximum likelihood group assignment finds precisely this case [25, 26].

Thus we find two local optima: factions and leaders-followers. While the leaders-followers finds the single best maximum likelihood community division, it is not immediately obvious which optima is actually the preferred clustering of the network, according to the SBM. In the factions posterior, both groups are roughly equally sized, whereas in the leaders-followers posterior, the groups are highly uneven. Because there are exponentially more ways to split n objects into roughly equally sized groups and highly uneven ones, it is possible that the factions grouping is preferred, i.e. that $P(G|\omega_{\text{factions}}) > P(G|\omega_{\text{leaders-followers}})$. In actuality, we find that this is not the case – the SBM truly favors the leaders-followers groupings. Depending on what you are hoping to find by graph clustering, this provides a compelling reason to use the degree-corrected SBM [26].

Again, we should contrast our method with existing approaches for analyzing the SBM. One approach is to use heuristics, such as the eigenvectors of an appropriately defined matrix or belief propagation [27, 28, 29]. These methods are generally fast but potentially highly biased.

An alternative approach for the SBM is again to use Markov chain Monte Carlo algorithms [30]. For example, one can use an Expectation Maximization (EM) algorithm in which the E-step is estimated by simulating an appropriate Markov chain. Of course, this relies on constructing a high quality

sampler, and it is also generally non-trivial to estimate the model evidence using this approach – deciding which local maximum is preferred remains challenging.

3.3 Ranking and noisy sorting

For a final example, we consider the problem of ranking from noisy comparisons. For a finite set of n objects, we are given noisy comparisons that indicate which of two objects is ranked higher. We want to infer the underlying ranks. Sports or chess matches provide prototypical examples of this scenario.

In the context of ranking, a state is a permutation of the n objects. We can represent this by letting $x_i \in \{\frac{1}{n}, \frac{2}{n}, \dots, \frac{n}{n}\}$, where $x_i = k/n$ means object i is ranked k th out of n . We will assume each comparison obeys the true ordering with probability w , or points in the wrong direction with probability $1 - w$.

If i and j were compared and i beat j , we write $i < j$. Any putative ranking either agrees or disagrees with the result of each comparison. Let $V(\mathbf{x})$ be the total number of disagreements between ranking \mathbf{x} and the comparisons,

$$V(\mathbf{x}) = \sum_{i < j} \mathbb{I}(x_i > x_j) \quad (14)$$

with \mathbb{I} being the indicator function. If there are m total comparisons then the posterior distribution over rankings is proportional to

$$f(\mathbf{x}) = w^m \left(\frac{1-w}{w} \right)^{V(\mathbf{x})}. \quad (15)$$

We will apply the auxiliary variable machinery to estimate $\mathcal{Z} = \sum_{\mathbf{x}} f(\mathbf{x})$ and the distribution $P(\mathbf{x}) = f(\mathbf{x})/\mathcal{Z}$. In general, exact calculation of $\ln \mathcal{Z}$ is #P-complete; for $w = 1$ it corresponds to counting linear extensions [31]. Even finding a single best ranking is NP-complete; it is equivalent to the minimum feedback arc set problem.

In this example x_i is discrete but can take any of n different values and we have the constraint that $x_i \neq x_j$ for all i, j (we don't allow for ties). Equation (7) and the Gumbel softmax trick no longer apply and so we need a new functional form for $R(\mathbf{x}|\mathbf{y})$ and a continuous relaxation to allow differentiation through the sampling procedure during training.

Thankfully, such a representation is available. The n dimensional unit hypercube can be split into $n!$ regular simplices, each defined as the set of points with a given sort-order. For example, one such simplex is $\{\mathbf{z} : 0 < z_1 < z_2 < \dots < z_n < 1\}$. Each vector of rankings \mathbf{x} is at the center of one of these simplices and by definition Eq. (14) is identical at each point within a simplex. The regions where $x_i = x_j$ in the have zero volume, so by relaxing from $x_i \in \{\frac{1}{n}, \dots, \frac{n}{n}\}$ to $x_i \in [0, 1]$, we will obey $x_i \neq x_j$ but leave $\mathcal{Z} = \sum_{\mathbf{x}} f(\mathbf{x}) = n! \int_{[0,1]^n} f(\mathbf{x}) d\mathbf{x}$ invariant.¹

We then assume R is a product of Beta distributions

$$R(\mathbf{x}|\mathbf{y}) = \prod_i \frac{x_i^{\alpha_i(\mathbf{y})-1} (1-x_i)^{\beta_i(\mathbf{y})-1}}{B(\alpha_i(\mathbf{y}), \beta_i(\mathbf{y}))} \quad (16)$$

and $\alpha(\mathbf{y})$ and $\beta(\mathbf{y})$ take the place of $\phi(\mathbf{y})$ and are represented by a neural network. Following the same logic of the Gumbel softmax trick, and to ensure pipeline is differentiable, we replace the non-differentiable indicator function in $V(\mathbf{x})$ with a steep sigmoid function, and reparametrize uniform noise using inverse transform sampling to generate \mathbf{x} .

For the ranking experiments, we used synthetic data. We fixed an arbitrary permutation of 64 objects and randomly made 512 noisy comparisons with noise parameter $w = 0.75$. We used this to define a distribution over permutations $P(\mathbf{x}|w)$.

We used the two neural networks, each with a single hidden layer with 1024 units, using SELU activation functions. The noise parameter w could be inferred by maximizing our estimate of $\ln \mathcal{Z}(w)$, however, we set it to the ground-truth value, $w = 0.75$ so that we could compare against MCMC, which does not infer the parameters (although in principle an EM style algorithm could be used to achieve this). Sample rankings were produced by first generating \mathbf{y} from $P(\mathbf{y})$ and then \mathbf{x} from

¹The factor of $n!$ is picked up because each simplex has volume $1/n!$.

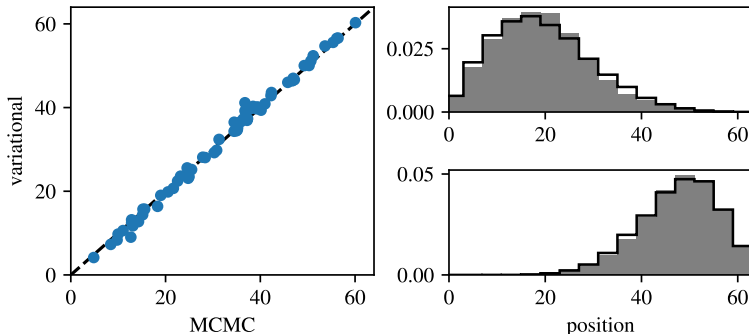


Figure 3: Distribution over rankings of 64 objects from 512 noisy comparison. Left: The average location of objects, as estimated using Markov chain Monte Carlo and the variational approximations of this paper. Right: The full distributions of position for two different objects. In gray, we show a histogram of the distributions as estimated from MCMC. The black line shows the inferred distribution from the variational method of this paper.

$R(\mathbf{x}|\mathbf{y})$. Ranks were computed by finding which simplex the relaxed \mathbf{x} fell inside, i.e. by the sort-order of its components.

We lack analytic values for $\ln \mathcal{Z}$ to compare against. Instead, in Fig. 3 we show comparisons to Markov chain Monte Carlo sampling for instances of \mathbf{x} ; we find excellent agreement.

4 Discussion

In this paper we have considered variational methods for solving physics and statistical inference problems. To this end we have repurposed the machinery of variational autoencoders but we have inverted the logic. Instead of encoding data samples to random noise (such as a normal distribution), we attempt to “encode” random binary noise into the distributions of interest.

For the three tasks we consider – Ising model, graph clustering, and ranking – we find good performance using small and simple networks: a single hidden layer with 1024 units. As a result training takes seconds, even without dedicated hardware². Presumably larger networks, and particularly those designed to match the structure of the problems, would improve performance. For example convolutional neural networks would match the locality and translational invariance of the Ising model, or graph neural networks would match the structure of the graph clustering problem.

One benefit of our methods are their contiguity with traditional mean-field methods, due to the fact that we can vary the number of auxiliary variables. For the Ising model, for example, at large β (low temperature) a highly accurate approximation is found with only a single binary auxiliary variable. The value of this binary variable switches between two solutions (majority of spins up vs. majority of spins down). Likewise, for very small β we are also able to find accurate approximations using only a few auxiliary variables. Conversely, near the phase transition ($\beta = -\ln(\sqrt{2} - 1)/2 \approx 0.4407$) accurate approximation requires using a larger number of auxiliary variables.

The examples we studied were chosen because they had known solutions – either exactly or approximately by using MCMC. This allowed us to *test* the VAE architecture, but in general it would be difficult to know whether the VAE has been successful. Nevertheless, we can always guarantee that the approximation for \mathcal{Z} is a lower bound. For some problems this may be useful even if we cannot be sure how close the bound is to the true answer.

Future work should systematically explore the results of differing neural network architectures and differing numbers of auxiliary variables. It should also address our lack of understanding of which distributions can and cannot be accurately represented by these feedforward neural networks. For example, these methods may be inappropriate for so-called *glassy* systems, where both variational

²Our code is available at <https://github.com/gcant/DistVAE>.

methods and MCMC can fail [32]. For extremely rich and complex distributions, i.e. those containing a very large number of roughly equal modes that are widely separated, the VAE may fail to find a good approximation. If such a failure occurs, is it due to the training scheme, or a failure of the architecture itself? At least in the case where feed-forward networks are used with a single hidden layer, as we considered in our experiments, it seems hopeful that analytic possibility/impossibility results may be achievable.

Acknowledgments and Disclosure of Funding

This work was supported by NSF Grant BIGDATA-1838251. I thank Cristopher Moore and Artemy Kolchinsky for helpful conversations.

References

- [1] R. J. Baxter. *Exactly Solved Models in Statistical Mechanics*. Academic Press, London, 1982.
- [2] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.
- [3] C. Moore and S. Mertens. *The Nature of Computation*. Oxford University Press, 2011.
- [4] D. A. Levin and Y. Peres. *Markov chains and mixing times*. American Mathematical Society, 2017.
- [5] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics*. Clarendon Press, 1999.
- [6] D. P. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 2021.
- [7] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv:1312.6114*, 2013.
- [8] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.
- [9] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021.
- [10] J. Altaosaar. *Probabilistic modeling of structure in science: Statistical physics to recommender systems*. PhD thesis, Princeton University, 2020.
- [11] D. Wu, L. Wang, and P. Zhang. Solving statistical mechanics using variational autoregressive networks. *Physical Review Letters*, 122:080602, 2019.
- [12] E. Brézin. *Introduction to Statistical Field Theory*. Cambridge University Press, 2010.
- [13] J. Hubbard. Calculation of partition functions. *Physical Review Letters*, 3:77–78, 1959.
- [14] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2013.
- [15] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-softmax. *arXiv:1611.01144*, 2016.
- [16] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv:1611.00712*, 2016.
- [17] D. Stauffer. Social applications of two-dimensional Ising models. *American Journal of Physics*, 76(4):470–473, 2008.
- [18] D. Yevick. Variational autoencoder analysis of Ising model statistical distributions and phase transitions. *The European Physical Journal B*, 95(3):1–15, 2022.
- [19] N. Walker, K.-M. Tam, and M. Jarrell. Deep learning on the 2-dimensional Ising model to extract the crossover region with a variational autoencoder. *Scientific Reports*, 10(1):1–12, 2020.
- [20] F. D’Angelo and L. Böttcher. Learning the Ising model with generative neural networks. *Physical Review Research*, 2(2):023266, 2020.

- [21] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983.
- [22] S. Fortunato and D. Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.
- [23] C. Moore. The computer science and physics of community detection: Landscapes, phase transitions, and hardness. *arXiv preprint arXiv:1702.00467*, 2017.
- [24] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.
- [25] P. J. Bickel and A. Chen. A nonparametric view of network models and Newman-Girvan and other modularities. *Proceedings of the National Academy of Sciences*, 106(50):21068–21073, 2009.
- [26] B. Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.
- [27] A. A. Amini, A. Chen, P. J. Bickel, and E. Levina. Pseudo-likelihood methods for community detection in large sparse networks. *The Annals of Statistics*, 41(4):2097–2122, 2013.
- [28] F. Krzakala, C. Moore, E. Mossel, J. Neeman, A. Sly, L. Zdeborová, and P. Zhang. Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences*, 110(52):20935–20940, 2013.
- [29] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová. Inference and phase transitions in the detection of modules in sparse networks. *Physical Review Letters*, 107(6):065701, 2011.
- [30] K. Nowicki and T. A. B. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455):1077–1087, 2001.
- [31] G. Brightwell and P. Winkler. Counting linear extensions is #P-complete. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing - STOC '91*, pages 175–181. ACM Press, 1991.
- [32] M. Mézard, G. Parisi, and M. A. Virasoro. *Spin glass theory and beyond: An Introduction to the Replica Method and Its Applications*, volume 9. World Scientific Publishing Company, 1987.