On Bayesian Neural Networks at Finite Temperature

Robert J.N. Baldock[1, *]

[1] *Theory and Simulation of Materials (THEOS), and National Centre for Computational Design and Discovery of Novel Materials (MARVEL), École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland*

(Dated: January 2019)

ABSTRACT

In this blog post I examine the Bayesian formulation of neural networks due to MacKay [1]. I consider how the noise level (or "temperature") T in that formulation affects the posterior distribution. I use this lens to show that, while sampling from the posterior might indeed lead to better generalisation than is obtained by standard optimisation of the cost function, even better performance can in general be obtained by sampling at finite temperature distributions derived from the posterior. Taking the example of deep (3 hidden layers) feed forward classifiers for MNIST data, and with small data sets, we see that in this case low temperature distributions derived from the posterior yield lower test loss than the posterior. I show that as the temperature is increased, neural networks transition from accurate classifiers to classifiers that are worse than random. Finally, I highlight an approximate method for performing model selection on deep NNs.

I. STRUCTURE OF THIS BLOG POST

1. Outline of the Bayesian formulation of feed forward neural networks.
2. An analogy with physics, introducing an "energy" function for the network, and a "temperature", which controls the noise level.
3. Results section
4. Sampling methods (code in this repository). I built an HMC version of an accelerated sampling method from materials simulation, called Replica Exchange Molecular Dynamics which simultaneously explores the behaviour of a model across a wide range of temperatures.

II. BAYESIAN FORMULATION OF FEED FORWARD NEURAL NETWORKS AND TRAINING

In training a classifier one typically uses a minimisation algorithm to find a local minimum of a cost function defined over the parameters of the neural network model. If we use a "softmax" then the activation of the output units sum to one, and can be interpreted as the Bayesian probability, assigned by the network, to the assertion that the input belongs to each individual class. We can write this as $\text{prob}\,(q|\mathbf{x},\mathbf{w})$ where q is the index of the class, $\mathbf{x}$ is the single data example to be classified and $\mathbf{w}$ represents the parameters of the neural network. For data in the training set $D=\{\mathbf{x}_i,t_i\}$ where $\mathbf{x}_i$ is an input data example and t_i the corresponding class label, then the probability that the network classifies $\mathbf{x}_i$ correctly is $\text{prob}\,(q=t_i|\mathbf{x}_i,\mathbf{w})$. This probability is a function of the weights $\mathbf{w}$. The probability that the network classifies the complete data set correctly is called the *likelihood* of the data, $\text{prob}\,(D|\mathbf{w})$, and is given by $\prod_i \text{prob}\,(q=t_i|\mathbf{x}_i,\mathbf{w})$.

In general, the *posterior* probability for the weights, which expresses what we have learned about the weights from the data, is given by

$$\text{prob}\,(\mathbf{w}|D) = \frac{\text{prob}\,(D|\mathbf{w})\,\text{prob}\,(\mathbf{w})}{\text{prob}\,(D)} \tag{1}$$

$$\propto \prod_i \text{prob}\,(q=t_i|\mathbf{x}_i,\mathbf{w})\text{prob}\,(\mathbf{w}) \tag{2}$$

where $\text{prob}\,(\mathbf{w})$ is the prior probability distribution for the weights, and represents our initial state of knowledge before the collection of any data.

If we assign a Gaussian prior with standard deviation σ to the parameters $\mathbf{w}$ then the posterior (2) is given by

$$\text{prob}\,(\mathbf{w}|D) \propto \prod_i \text{prob}\,(q=t_i|\mathbf{x}_i,\mathbf{w}) e^{-\frac{\mathbf{w}^2}{2\sigma^2}}. \tag{3}$$

The posterior probability of the weights (3) are therefore maximised when we minimise

$$J(\mathbf{w}|D) = -\sum_i \log\left[\text{prob}\,(q=t_i|\mathbf{x}_i,\mathbf{w})\right] + \frac{\mathbf{w}^2}{2\sigma^2} \tag{4}$$

which can immediately be recognised as the cross entropy loss function with L2 regularisation. The minimum of (4) is called the maximum a posteriori solution.

In this blog post I will be using a uniform prior for $\mathbf{w}$

$$\text{prob}\,(\mathbf{w}) = \begin{cases} \frac{1}{\prod_{i=1}^d \sigma_i}, & |w_i| < \sigma_i/2 \,\forall\, i, \\ 0, & \text{Elsewhere.} \end{cases} \tag{5}$$

Within the bounds of this prior the cost function is given by

$$J(\mathbf{w}|D) = -\sum_i \left(\log\left[\text{prob}\,(q=t_i|\mathbf{x}_i,\mathbf{w})\right] - \log\sigma_i\right). \tag{6}$$

* rjnbaldock@gmail.com

Outside the bounds of the prior the cost function is infinite. Minimising the cost function will simply mean minimising the likelihood within these bounds.

III. INTRODUCING ENERGY AND TEMPERATURE

One may define the "potential energy" over the parameters of the network [1]

$$E(\mathbf{w}|D) = -\sum_i \log\left[\text{prob}\left(q = t_i|\mathbf{x}_i, \mathbf{w}\right)\right] \tag{7}$$

and a "thermal" posterior distribution

$$\text{prob}\left(\mathbf{w}|T, D\right) \propto e^{-\frac{1}{T}E(\mathbf{w}|D)}\,\text{prob}(\mathbf{w}). \tag{8}$$

where we assign T controls the noise in our posterior. We name T the "temperature" in analogy with thermodynamic temperature.

For this blog post, it is important to imagine how the thermal posterior behaves as we vary T. The distribution (8) tends towards the prior distribution for $T \to \infty$, and is equal to the true posterior distribution (2) for $T = 1$. At $T < 1$ (low temperature) the thermal posterior (8) is concentrated around the maximum a posteriori solution (the minimum of (4)).

A. Temperature vs batch size

It is known that mini-batch learning improves generalisation. There has been a lot of interest in how the batch size in mini-batch learning controls the noise level during learning, and under what conditions the optimiser can be said to be approximately sampling from the posterior. Here though I wanted to explore instead using the temperature to control the noise in full batch training and as an alternative to early stopping.

IV. RESULTS

A. Sampling the posterior for small data sets

There is broad agreement that sampling from the posterior leads to better generalisation. Here we see that this may be true if you restrict the sampler to the region surrounding a minimum of the cost function, but need not be true in general.

Figure 1 shows the behaviour of a network formed of 3 hidden layers of 40 logistic neurons, and an output layer of 10 logistic neurons, corresponding to the 10 classes of the MNIST data set. A softmax was applied to the output layer. For computational speed the MNIST data images have been rescaled down from 28 x 28 to 16 x 16 (256 input dimensions), and normalisation has been applied. A random selection of rescaled data is shown in Figure 2.

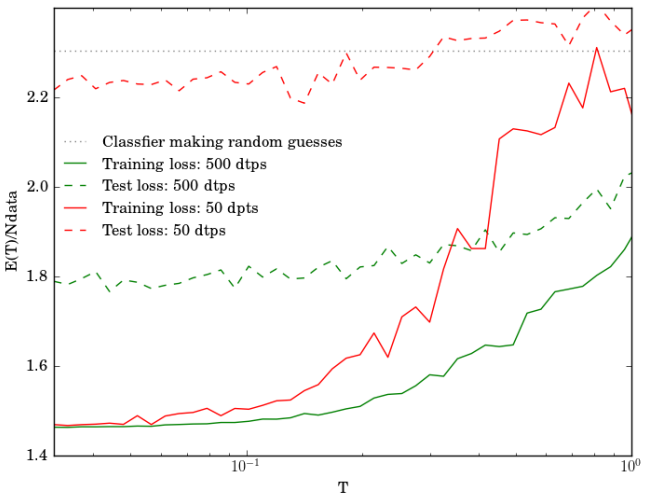

FIG. 1. Average (sampled) values of $E(T)$ for a network with three hidden layers and 40 nodes in each hidden layer. The loss values reported are the average energy E: the negative log-likelihood. Red: network trained with stratified data sets of 5 items per class (50 data points in total). Blue: network trained with stratified data sets of 500 items per class (5000 data points in total). The network exhibits substantially worse generalisation for both data sets at $T = 1$ (sampling from the posterior) than at $T = 10^{-\frac{3}{2}}$. At $T = 1$ the small data set is so uninformative that the network is apparently worse than a classifier making random guesses.

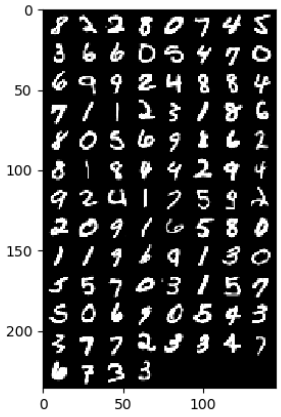

FIG. 2. 100 random data samples from the MNIST training set. These images have each been normalised and rescaled down from 28x28 to 16x16.

One remark we can draw from Figure 1 is that *in general it is better to sample a rather low T thermal posterior distribution than to sample from the posterior itself.* This observation can be rationalised as follows. Consider the thermal posterior (8) at $T = 1$ (which is just the standard

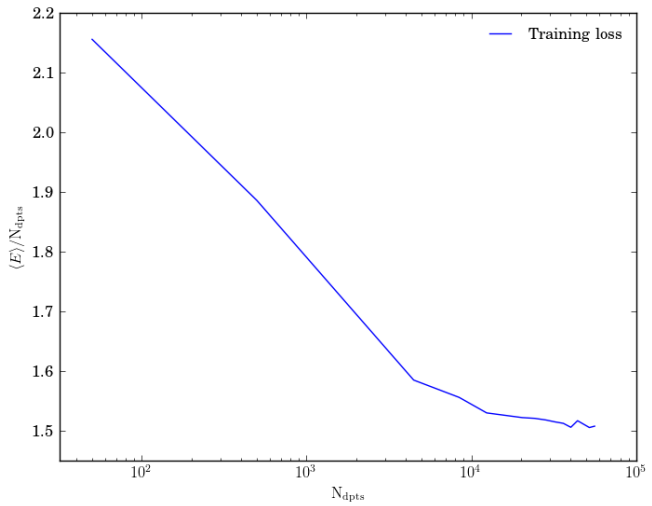

FIG. 3. Training loss for typical network parameters, sampled from the posterior (2) with a uniform prior (5), for different sized training sets, N_{dpts}. Training sets comprised stratified samples from the MNIST training set, composed of $N_{dpts}/10$ images for each of the 10 classes. The network used is described in the text.

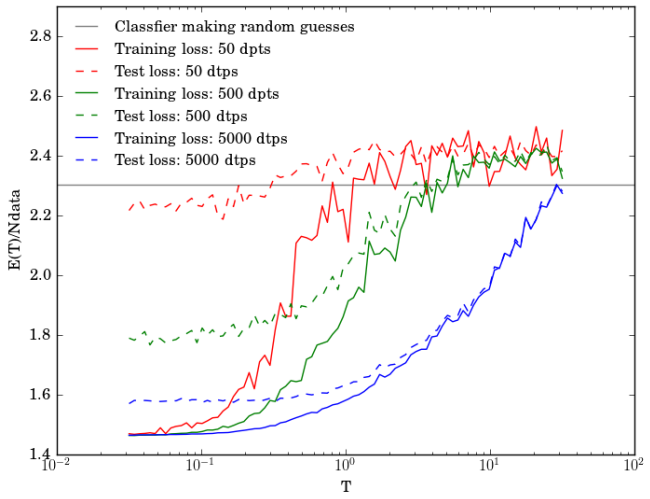

FIG. 4. Training and test errors of the network described in Section IV A at 3 different data sizes, over a wide temperature range. The loss values reported are the average energy E: the negative log-likelihood. Each of the 10 classes has an equal number of data points (5, 50 or 500 per class). In the high temperature limit the network approaches, or becomes worse than, a random classifier.

posterior). Let's imagine that the prior is wide, diffuse distribution, while the likelihood function $e^{-E(\mathbf{w}|D)}$ is sharply localised. Although the prior prob $(\mathbf{w})$ is a proper probability distribution and is normalised, the likelihood function is *not* a probability distribution and is not normalised. We can see from (7) that $E(\mathbf{w}|D)$ is roughly proportional to the number of data points, so the height of the likelihood scales approximately exponentially in the number of data points. When the number of data points is small, the height of the likelihood is exponentially small and the majority of the posterior weight corresponds to the prior. Unless the choice of prior was extraordinarily informed, models drawn from the posterior with a small data set will perform poorly as classifiers. Conversely, for large data sets, the main peak of the likelihood is exponentially taller, and the majority of the posterior weight is over models with high likelihood. Networks with parameters sampled from the posterior with large amounts of data are therefore effective classifiers. This behaviour is quantified in Fig. 3 which shows how the training error of our network depends on the size of the training set.

B. Temperature induced transition from an accurate to inaccurate classifier

Figure 4 shows the behaviour of the same network with three hidden layers just described in Section IV A over a much wider range of temperatures and for three different sized data sets.

As the temperature is increased the main body of the thermal posterior distribution transitions from parameters corresponding to accurate classifiers to worse than random classifiers that make guesses that are wrong more often than they are right.

For small data sets the training error shifts suddenly and rapidly over a narrow range of temperatures, even on a log scale. It is tempting to draw an analogy between the transition from an accurate to inaccurate classifier in small data sets, and a phase transition in statistical physics.

C. Hessian free Bayesian model selection for deep (or shallow) neural networks

I'll start with a quick overview of Bayesian model selection. Then I'll introduce the Laplace approximation (approximating a distribution by a Gaussian fitted to a given maximum of the true distribution). Finally, I'll show how you can compute the Bayesian evidence for that Gaussian distribution, by finding the determinant of the Hessian without calculating the Hessian directly.

1. *Bayesian Model Selection*

Imagine we have two different machine learning models M_1 and M_2, and a data set $D = \{\mathbf{x}, t_i\}$, or in vector notation $D = (\mathbf{x}, \mathbf{t})$ where $\mathbf{x}$ is a matrix of all the input data and $\mathbf{t}$ a vector of the corresponding labels.

In Bayesian statistics we are able to calculate the probability of each model given the data, prob $(M|\mathbf{x}, \mathbf{t})$. In Bayesian model selection you choose the model with the highest probability. Actually you never choose, you be-

lieve them all do different extents, according to their probabilities.

The ratio of the probabilities for M_1 and M_2 can be found as follows

$$\text{prob}\,(M,\mathbf{t}|\mathbf{x}) = \text{prob}\,(M|\mathbf{x},\mathbf{t})\,\text{prob}\,(\mathbf{t}|\mathbf{x}) \tag{9}$$

$$= \text{prob}\,(\mathbf{t}|\mathbf{x},M)\,\text{prob}\,(M|\mathbf{x}) \tag{10}$$

$$\Rightarrow \frac{\text{prob}\,(M_1|\mathbf{x},\mathbf{t})}{\text{prob}\,(M_2|\mathbf{x},\mathbf{t})}\frac{\text{prob}\,(\mathbf{t}|\mathbf{x})}{\text{prob}\,(\mathbf{t}|\mathbf{x})} = \frac{\text{prob}\,(\mathbf{t}|\mathbf{x},M_1)}{\text{prob}\,(\mathbf{t}|\mathbf{x},M_2)}\frac{\text{prob}\,(M_1)}{\text{prob}\,(M_2)} \tag{11}$$

$$\Rightarrow \frac{\text{prob}\,(M_1|\mathbf{x},\mathbf{t})}{\text{prob}\,(M_2|\mathbf{x},\mathbf{t})} = \frac{\text{prob}\,(\mathbf{t}|\mathbf{x},M_1)}{\text{prob}\,(\mathbf{t}|\mathbf{x},M_2)}\frac{\text{prob}\,(M_1)}{\text{prob}\,(M_2)} \tag{12}$$

The last term on the right in (12) is the ratio of our priors for the models. Typically we might initially have no preference between M_1 and M_2, in which case this last term is equal to 1.

In theory one can calculate the "marginal likelihood" of the model, $\text{prob}\,(\mathbf{t}|\mathbf{x},M)$, as follows

$$\text{prob}\,(\mathbf{t}|\mathbf{x},M) = \int d\mathbf{w}\text{prob}\,(\mathbf{t},\mathbf{w}|\mathbf{x},M) \tag{13}$$

$$= \int d\mathbf{w}\text{prob}\,(\mathbf{t}|\mathbf{w},\mathbf{x},M)\,\text{prob}\,(\mathbf{w}) \tag{14}$$

$$= \int d\mathbf{w} e^{-J(\mathbf{w}|D)} \tag{15}$$

In (15) we have made the connection with (6). There are lots of clever ways to calculate this integral. For deep neural networks it is almost always highly multimodal.

2. The Laplace approximation

One way to compute integral (14) is to make the Laplace approximation. We move to a local (ideally global) maximum of the posterior $\text{prob}\,(\mathbf{t}|\mathbf{w},\mathbf{x},M)\,\text{prob}\,(\mathbf{w}|M)$, which is at $\mathbf{w}_{\text{MP}}$. We approximate the integral (14) as the height of the peak of the integrand posterior, times its parameter space volume $\prod_{i=1}^{d}\sigma_{\text{MP},i}$ where $\{\sigma_{\text{MP},i}^{-2}\}$ are the eigenvalues of the Hessian of $J\,(\mathbf{w}_{\text{MP}}|D)$ calculated at $\mathbf{w}_{\text{MP}}$. In this case we can approximate the integral (14) as

$$\text{prob}\,(\mathbf{t}|\mathbf{x},M) \simeq e^{-J(\mathbf{w}_{\text{MP}}|D)}\prod_{i=1}^{d}\sigma_{\text{MP},i} \tag{16}$$

The value of $J\,(\mathbf{w}_{\text{MP}}|D)$ is directly obtained at the conclusion of minimisation. All we need is the $\prod_{i=1}^{d}\sigma_{\text{MP},i}$ but we don't want to calculate or diagonalise the Hessian. The cost of doing so is cubic in the number of parameters, which could easily reach the millions for a production model.

3. Hessian free estimation of the determinant of the Hessian

We can use thermodynamic integration to calculate the free energy of such a quadratic form. I'm just finishing this calculation and I'll write it up *very* soon.

D. Discussion

We have seen that, for MNIST, sampling a low temperature analogue of the posterior (8) leads to parameters for deep neural networks with lower bias than sampling the posterior itself. This behaviour is especially pronounced for small data sizes. We observed that, for small data sizes, typical values of the cross entropy loss in the posterior distribution exhibit an approximately exponential dependence on the size of the data set.

We saw that for network parameters sampled from the "thermal" posterior across a range of temperatures, deep neural networks exhibit a transition between accurate and inaccurate classifiers, reminiscent of a phase transition in statistical physics.

(In progress) I am working to show how thermodynamic integration can be used to perform model selection on deep and shallow neural networks, avoiding the need to ever calculate or invert the Hessian of the cost function.

V. SAMPLING METHODS

A. Replica Exchange Hamiltonian Monte Carlo

This work was partly inspired by Radford Neal's PhD work [2] on Hamiltonian (also called Hybrid) Monte Carlo (HMC) [3, 4] in neural networks.

I developed a HMC formulation of a technique called Replica Exchange Molecular Dynamics (REMD) [5] for use with NN models. REMD is a method for exploring the behaviour of a sampler simultaneously across a range of temperatures. In REMD the user specifies a temperature range to be studied. Normally this range goes from the lowest temperature of interest, up to an artificially high temperature where the sampler moves very quickly. Independent samplers are created at particular samplers throughout that temperature range. These samplers periodically attempt to exchange their current points in parameter space with samplers at neighbouring temperatures. This accelerates parameter space sampling at low temperatures because each set of coordinates spends some of the time at high temperatures where they move around much more rapidly.

Parameter swaps between neighbouring temperatures are proper Markov chain Monte Carlo (MCMC) swaps [6,

7], constructed to preserve the joint distribution

$$\text{prob}\left(\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_N | T_1, T_2, \ldots T_N, D\right) \propto \text{prob}\left(\mathbf{w}_1 | T_1, D\right) \times \text{prob}\left(\mathbf{w}_2 | T_2, D\right) \times \cdots \times \text{prob}\left(\mathbf{w}_N | T_N, D\right). \tag{17}$$

Importantly, this joint distribution is separable, so samplers at each temperature are always sampling from the respective "thermal" posterior (8) even immediately after a parameter swap.

The standard REMD algorithm samples parameter space using "molecular dynamics" (MD) [8], which is based on motion according to Newton's Laws of Motion on the energy surface $E(\mathbf{w}|D)$. Another algorithm called Parallel Tempering (PT) [9] is closely related to REMD and replaces MD with MCMC [6, 7]. My implementation is based around HMC and in the code I chose to use the name Parallel Tempering as a generic term to cover all variants. I could just have well used Replica Exchange. I will first describe the general REMD (PT) algorithm and then justify my use of HMC.

1. The Replica Exchange algorithm

The general REMD (PT) algorithm is as follows. We choose a set of temperatures for the samplers: $\{T\}$. The current parameters values for the sampler at temperature T_i are $\mathbf{w}_i$.

Algorithm 1 General REMD / PT.

procedure ReplicaExchange($\{(T_i, \mathbf{w}_i)\}$)
  loop
    for T_i in $\{T\}$ **do**
      Update sample $\mathbf{w}_i$ from (8) using MCMC/MD.
    end for
    for $i \leftarrow 1, N_T$ **do**
      Choose random adjacent T pair (T_j, T_{j+1}).
      Exchange $(\mathbf{w}_j, \mathbf{w}_{j+1})$ with probability
      $\min\left[1, e^{\left(\frac{1}{T_j} - \frac{1}{T_{j+1}}\right)\left(E_j - E_{j+1}\right)}\right]$.
    end for
  end loop
end procedure

In my implementation I replaced MCMC/MD with HMC (see Sec. V A 2).

Additional details include: how, and how often, to update the step size (time step) of the MCMC/MD algorithm; and for MD, how to treat the momenta after a swap. In my implementation step sizes are updated by running additional trajectories which are not included in the main simulation, and measuring the acceptance rate of the trajectories. The time step is updated to obtain an acceptance rate inside a given range. The frequency of doing this is set by the user. For simple HMC the momenta are discarded at the start of each trajectory, so I did not transfer them.

At the start of the simulation I initialised independent parameters of the neural network for each temperature by the following approach:

1. Draw a sample from the uniform prior (5).
2. Minimise the cost function (6) using Alg. 2. I found this to be very efficient for these simple networks. Minimisation avoids starting the dynamics from parameter values where the gradient is extremely large.
3. Perform burn in trajectories at the appropriate temperature.

Algorithm 2 Fast minimisation of the cost function (6).

procedure RoughMinimise($(\mathbf{w}_i)$)
  $\mathbf{p} = 0$ ▷ Set initial momenta to zero.
  for $i = 1, N_{\text{steps}}$ **do**
    $\mathbf{w}_{\text{save}}$, $J_{\text{save}} = \mathbf{w}$, $J(\mathbf{w}|D)$ ▷ Save lowest cost state.
    Propagate $\mathbf{w}$ through 1 MD time step, time step dt.
    if $J(\mathbf{w}|D) < J_{\text{save}}$ **then** ▷ Downwards step
      $dt = dt + 0.05$. ▷ Slowly increase dt.
    else ▷ Upwards step
      $\mathbf{p} = 0$ ▷ Zero momenta
      $\mathbf{w} = \mathbf{w}_{\text{save}}$ ▷ Return to lowest cost $\mathbf{w}$
      $dt = dt * 0.95$ ▷ Rapidly decrease dt.
    end if
  end for
end procedure

In my experiments, algorithm 2 the maximum gradient of the cost function in any dimension was typically reduced to 10^{-4} within 400 steps for the networks described in this report.

2. Hamiltonian Monte Carlo

B. Thermodynamic Integration

[1] D. J. MacKay, Neural computation **4**, 448 (1992).
[2] R. M. Neal, *Bayesian learning for neural networks*, Vol. 118 (Springer Science & Business Media, 2012).
[3] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, Physics letters B **195**, 216 (1987).
[4] R. M. Neal *et al.*, Handbook of markov chain monte carlo **2**, 2 (2011).
[5] R. H. Swendsen and J.-S. Wang, Phys. Rev. Lett. **57**, 2607

(1986).
[6] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, The journal of chemical physics **21**, 1087 (1953).
[7] D. Frenkel and B. Smit, *Understanding molecular simulation: from algorithms to applications*, Vol. 1 (Elsevier, 2001).
[8] B. J. Alder and T. E. Wainwright, The Journal of Chemical Physics **31**, 459 (1959).
[9] R. H. Swendsen and J.-S. Wang, Physical review letters **57**, 2607 (1986).