Scaling MCMC methods for Bayesian neural networks

Yiu Sing Lau

Contents

I Theory **5**

1 Model **7**

1.1 Markov Chain Monte Carlo 7

1.2 Hamiltonian Monte Carlo 9

1.3 MCMC methods applied to hierarchical models 22

1.4 approximate inference for fast training and prediction 25

1.5 Scaling MCMC methods to large datasets 27

1.6 Bayesian Model Selection and Prediction 33

II Experiments **37**

2 Experimeints **39**

2.1 neural networks 44

Part I

Theory

Chapter 1

Model

1.1 Markov Chain Monte Carlo

Starting from the beginning of research using MCMC techniques in statistical applications, there has been work [35, 5] in the area of image analysis. Because of the high-dimensional nature of image datasets, initially only the Gibbs sampler were used, circumventing the well known slow-mixing/low-acceptance behaviour of the random-walk Metropolis-Hastings sampler in high dimension. The introduction of Hamiltonian Monte Carlo into the field of statistics by Neal [73, 72], who built on the work of physicists using HMC to simulate from lattice field models [25], brings an exciting new tool that allows statisicians to simulate much more efficiently from high-dimenisonal distributions. Unfortunately, because of the relative difficulty in implementing such samplers its use in statistics has been limited. In the last few years there has been a rebirth of the HMC sampler, with new developments thattry to utilize information about the local curvature of the density function during sampling [38, 7], as well as to automate the selection of tuning parameters [44, 8].

In this work, we seek to understand the behaviour of the HMC sampler when applied to multilayer neural network models grouped under the umbrella term of "deep learning"[88]. Previous work exists [21, 72] but they date from before the "deep learning revolution" of the mid 2000s, when training neural networks of more than one hidden layer proved to be feasible and yielded superior performance to one-hidden-layered models. Also, those works rely heavily on manual tuning, assume small datasets and do not have access to parallel computing, all of which make the methodology lacking for dealing with the type of tasks present in deep learning presently.

[40] gives a good summary of dynamical MCMC methods (langevin diffusion,MALA,HMC) and related convergence assessment methodologies.

In the late 80s and throughout the 90s, Bayesian statistical modeling became feasible for a much larger class of problems than previously thought possible, because of the availability of abundant computing power. Much work was produced in MCMC methodologies[82], theory[94, 87] as well as in applications thereof.

Bayesian Inference can be simply summarized by the specification of a likelihood and a prior function. Suppose $\theta \in \mathbb{R}^d$ is the parameter of the likelihood function for observed data $X \sim p(x|\theta)$, we can model our uncertainty about it with a prior function $p(\theta)$. Upon observing the data, we can perform Bayesian inference by using information from the poserior

distribution

$$p(\theta|x) \propto p(x|\theta)p(\theta)$$

Examples: MAP, marginal posterior, credible interval. The exact calculation of the normalizing constant $Z = \int p(x|\theta)p(\theta)d\theta$ is possible for certain convenient functions only, and we must rely on Markov Chain Monte Carlo techniques to approximate the constant.

See more about Bayesian data analysis in [32].

The Metropolis-Hastings sampler.

Suppose we have a an unnormalized density $p(x)$, and $q(x, y)$ is a proposal function such that conditional on the current state x, the probability of moving from x to a measurable set A is denoted by $q(x, A)$, we can calculate the Hastings ratio

$$r(x, y) = p(y)q(y, x)p(x)q(x, y).$$

Then we accept the move to state y with probability $\min(1, r(x, y))$, and stay in the current state otherwise.

The acceptance probability can be optimized, as shown in [83, 33, 86]. In the case of a target distribution with zero correlation between components and using a symmetric gaussian proposal, the optimal acceptance rate is 0.234. Optimal scaling can also be derived for the langevin sampler [84].

Very much like optimization [102], for differentiable density functions, information about the gradient can be useful for proposing the next state. It inspires the Langevin diffusion [], which is a stochastic process defined by a stochastic differential equation with the posterior distribution as its stationary distribution. It can only be implemented by discretization, which introduces some bias into the samples drawn. To offset this, a Metropolis acceptance step is introduced and this gives the Metropolis Adjusted Langevin Algorithm.

Algorithm 1 Metropolis-Adjusted Langevin Algorithm

▷ *iter* is the maximum number of iterations, $p(\cdot)$ is the unnormalized target density, τ is the variance parameter

function MALA($iter, p(\cdot), \tau$)

x_0 is initialized randomly.

for i in 1:*iter* **do**

$Z \sim \mathcal{N}(0, I_d)$

$\tilde{x} = x_{i-1} + \tau \nabla log p(x_{i-1}) + \sqrt{2\tau} Z$

$\alpha = \min\{1, \frac{p(\tilde{x}q(x_{i-1}|\tilde{x})}{p(x_{i-1})q(\tilde{x}|x_{i-1})}\}$

$U \sim U(0, 1)$

if $U \leq \alpha$ **then**

Set $x_i = \tilde{x}$

else

Set $x_i = x_{i-1}$.

end if

end for

1. General Metropolis-Hastings framework. Theoretical ideas like ergodicity, central limit theorem, and convergence

diagnostics. Problem of correlation in the multivariate case. Need for reparametrization sometimes.

The general idea of detailed balance. What does this property bring us? Fact: The Metropolis-Hastings algorithm satisfies the detailed balance condition and if the proposal conditional density function satisifies the positivity condition

$$q(y|x) > 0 \text{ for every } (x, y) \in \Omega \times \Omega$$

Gibbs sampling as a special case of Metropolis-Hastings. Suppose our distribution is p dimensional, then the Gibbs sampler consists of drawing samples from each of the p conditional distributions in turn, visiting each component deterministically or randomly. (Insert reference to random vs fixed scan Gibbs sampler)

Evaluate sample quality Effective sample size is defined as

$$ESS = N\{1 + 2\sum_{k} \gamma(k)\}^{-1},$$

estimated by the initial monotone sequence estimator [36]

It is a useful metric for measuring quality of the samples obtained from a Markov Chain. We can take the mean or minimum ESS across all covariates.

Convergence diagnostics for high-dimensional target distributions. Theoretically, need convergence of the Markov Chain to the joint distribution, in practice only convergence to marginal distributions are checked. The literature has only looked at applying univariate diagnostics to each parameter individually and then calculate some summary statistic (mean,min) or carry out multiple testing if the diagnostic is based on a hypothesis test.

KL-divergence can be used to compare samples drawn from two different samplers. The method advanced in [15, 14] uses a kernel estimator to estimate the KL divergence between two empiricial distributions. The curse of dimensionality makes it difficult for application to high-dimensional posterior distributions $p(\theta|x)$, however, we can apply it to marginal predictive distributions $p(y|D)$ which usually are low-dimensional, when the covariates x are fixed.

Tune MCMC algorithms. MCMC algorithms usually have a small number of tuning parameters, such as a stepsize ϵ, the number of leapfrog steps, or more basic quantities like the length of the chain. Classic Bayesian methodology [82] uses a mix of visual inspection of trace plots and numerical convergence diagnostics like ESS as discussed earlier. The process requires human intervention each time an algorithm is run and tuning parameters readjusted after the diagnostics are computed.

1.2 Hamiltonian Monte Carlo

.

Originally developed in the physics community [25], Hamilotnian Monte Carlo was introduced to the statistics community by Neal [72], by way of Computer Science, through his work on inference for Bayesian neural networks. It was

shown to be very efficient in sampling from high-dimensional distributions and was used in bayesian inference to achieve superior predictive performance in machine learning tasks[41]. However, its sensitivity to tuning parameters, relative difficulty of implementation, and a lack of accessible exposition to its theory, itself a subject of open research, precludes widespread adoption. While some of these challenges remain to this day, the development of STAN [18] has made it significantly easier to carry out Bayesian inference with HMC, and is the main reason for increasing adoption of HMC in applications (cite statistics). Meanwhile, the invention Riemmanian Manifold Hamiltonian Monte Carlo (RMHMC) [38], which exploits differential geometry to aid in local adaptation of the HMC sampler, and subsequent exploration the ideas introducd in the their paper, lead to better understanding of the mathematics behind HMC [61, 10] and improvements to the sampler [9, 7]. Many of these new developments have been and will be included in the STAN language. STAN also eliminates the need for tuning HMC. Users of STAN are shielded from the implementation details and tuning parameters settings, required only to specifiy the distributions involved in the models.

Since STAN is so central to the present HMC literature, we would explain how it functions in the following exposition. Before that, however, we would explain the basic version of HMC on which STAN is built and to which many extensions are added.

Suppose we have a density $p(x), x \in \mathbb{R}^d$ from which we would like to sample. If $p(x)$ is simply the marginal density of some distribution $p(x, y), (x, y) \in \mathbb{R}^{d+p}$ in a larger space containing the original domain, then sampling from $p(x, y)$ and keeping only the x's is equivalent to sampling from $p(x)$ directly. These extra variables introduced are called auxillary variables. The auxillary variables methods are known to speed up sampling by introducing extra degrees of freedom in the state space and allows the chain to move more easily across different parts of it. The Swendsen-Wang sampler [97], the slice sampler[97] are well known examples of this class of methods. See [59, 60] for more details.

The Metropolis-Hastings sampler is a general algorithm that allows us to sample from any distribution, discrete, continuous or neither, as long as we know its density up to normality. The Hamiltonian Monte Carlo (HMC) sampler is less general in that it requires the unnormalized density be continuous and differentiable.

We denote the unnormalized density of the target distribution by $P(x)$, and the normalizing constant by $Z = \int P(x)dx$, and we define the potential energy function $U(x)$ as

$$U(x) = \exp(-\log(P(x)))$$

If we now introduce a kinetic energy function $K(y)$, and define the Hamiltonian as the total energy, i.e., sum of the kinetic and potential energy functions,

$$H(x, y) = U(x) + K(y)$$

we get that

$$P(x, y) \propto \exp(-H(x, y))$$

has $P(x)$ as the marginal density, and thus defines an auxilary variable method. In statistical applications,we usually have $P(x)$ as the unnormalized posterior density,

$$U(q) = -\log(prior(x) \cdot likelihood(x|data)),$$

and

$$K(y) = \sum_{i=1}^{d} \frac{y_i^2}{m_i}$$

,which is equivalent to introducing an independent multivariate Gaussian random variable of the same dimension as the original distribution as an auxilary variable.

Note that there is physical interpretation of the system. The Hamiltonian uniquely determines the motion of a particle, whose position in space at any time is described by the x coordinates, and whose momentum is described by the y coordinates. Its motion can be described by solving a system of ordinary differnetial equations in Hamiltonain dynamics, known as Hamilton's equations:

To simulate the trajectory of a particle given its initial state (x_0, y_0) one would have to solve Hamilton's equations. Unfortunately, there are in general no explicit solutions for Hamilton's equations except for trivial models. One would then have to resort to numerical methods that discretize the Hamiltonian dynamics [55]. The discretization should possess some nice properties , including reversibility and volume-preservation, to ensure convergence to the target distribution, as well as maintain stability and accuracy of the approximation over long trajectories, the lack of which significantly reduces the efficiency of the sampler, irrespective of model-related factors which might affect sampling efficiency, like parametrization or posterior correlation.

For the simple version of HMC where the momentum variable is independent from the position variable, i.e. $p(y|x) = p(y)$, an easy-to-implement integrator which is both volume-preserving and reversible, with lower approximation error than conventional alternatives for numerically solving like Euler's method, known as the leapforg integrator, exists and works as follows. At time t of the trajectory, given its current position and momentum, $q(t)$ and $p(t)$,and stepsize ϵ, we would update the position and momentum at time $t + \epsilon$ by setting

$$p(t + \frac{\epsilon}{2}) = p(t) - \frac{\epsilon}{2} \cdot \frac{\partial U}{\partial q} q(t) \tag{1.1}$$

$$q(t + \epsilon) = q(t) + \epsilon \frac{\partial K}{\partial p}(p(t + \epsilon/2)) \tag{1.2}$$

$$p(t + \epsilon) = p(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2} \cdot \frac{\partial U}{\partial q}(q(t + \epsilon)) \tag{1.3}$$

One leapfrog step consists of first a half-step update for momentum, a full-step for position, then another half-step for momentum. Since the update for momentum at time $t + \epsilon$ and time $t + \epsilon + \frac{\epsilon}{2}$ both involve $q(t + \epsilon)$, when L leapfrog steps are used performed sequentially with stepsize ϵ, as done during the simulation of a trajectory when proposing a new state, we can combine the updates in the implementation, so that instead of perform two half steps for momentum

$p(t+\frac{\epsilon}{2}) \to p(t+\epsilon)$, then $p(t+\epsilon) \to p(t+\epsilon+\frac{\epsilon}{2})$, we can perform a full step for momentum $p(t+\frac{\epsilon}{2}) \to p(t+\epsilon+\frac{\epsilon}{2})$ as

$$p(t+\epsilon+\frac{\epsilon}{2}) = p(t+\epsilon) - \frac{\epsilon}{2}\frac{\partial U}{\partial q}(q(t+\epsilon)) \tag{1.4}$$

$$= p(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial U}{\partial q}(q(t+\epsilon)) - \frac{\epsilon}{2}\frac{\partial U}{\partial q}(q(t+\epsilon)) \tag{1.5}$$

$$= p(t+\frac{\epsilon}{2}) - \epsilon\frac{\partial U}{\partial q}(q(t+\epsilon)) \tag{1.6}$$

Note, however, we still need $p(t+\frac{\epsilon}{2})$ to perform the full steps, so that at least one half step has to be made for the momentum. Also, in this alternative implementation of the leapfrog integrator, momentum p is always one half step ahead of position, hence the last update for momentum has to be a half step so we end up with $p(t+\epsilon L)$ and $q(t+\epsilon L)$, both synchronized. The two implementations yield exactly the same updates, bar loss of accuracy from calculating the sums and extra computer time for updating the momentum variables twice in the naive implementation.

Starting at some initial time t_0, with initial coordinates $(q(t_0), p(t_0))$, the position and momentum at time $t+s$ can be determined by repeating the steps above roughly $\frac{s}{\epsilon} = L$ times, taking care to round up or down.

One advantage of the leapfrog integrator over other more well-known methods for solving ODE numerically such as Euler's method is that it is symplectic, which means it preserves volume in the phase space, just like the Hamilonian. This contributes to its relative stability along the trajectory compared to Euler's method and its modifications. An unstable trajectory as simulated by Euler's method would propose a state that has diverged from the energy level set to which the intitial state belong, which makes proposals much more likely to be rejected, especially in high dimensional distributions. However, the leapfrog method does not preserve the Hamiltonian, so the energy of the particle would oscillate as it is being simulated. It has important implications for the sampling algorithm. Intuitively, this breaks the law of conservation of energy and suggests the position of the particle at the end of its trajectory may deviate from the correct trajectory, on which the energy should be preserved. Reducing the stepsize would mitigate the problem but it comes with increased computational costs. This necessitates the introduction of a Metropolis acceptance step in order to preserve the invariant distribution. To explain this, let's put HMC in the context of an auxillary variable method.

First, the joint distribution of the position (original) and momentum(auxillary) variables is factored into a product of a conditional distribution $p(x, y|E)$ and a marginal distribution $p(E)$

$$p(x, y) = p(x, y|E)p(E)$$

Sampling from the joint distribution is done by first sampling from the auxillary (energy) distribution $p(E)$. In this case randomness comes in only from the resampling of the momentum variable, that is, change in the kinetic energy, because the potential energy is not dependent on the momentum. Then conditional on the value of the sampled energy, sample from the conditional distribution for the joint conditional distribution of the position and momentum variable. Leapfrog steps are involved only in the sampling from the conditional distribution $p(x, y|E)$. This "sampling" is actually deterministic as it consists of simulating a trajectory under Hamiltonian dynamics with given initial states. Because

Hamiltonian flow is reversible and volume-preserving independent of numerical discretization, consistency theorems apply without the need for Metropolis correction if we can simulate Hamiltonian dynamics exactly. Indeed, there has been experiments with HMC where the Metropolis acceptance step is dispensed with and samples drawn from the resulting algorithm, when used for inference, still yields good predictive performance on regression tasks [69], although it is offset by decreased robustness as the uncorrected trajectory could become unstable when the stepsize becomes too large.

Now, we describe a simple version of the HMC sampler. The sampler starts by drawing from the conditional (marginal because of independence) distribution of the momentum/auxillary variables y, then it simulates the movement of a particle having initial position and momentum (x, y) according to Hamiltonian dynamics. More precisely, we do L leapfrog steps of stepsize ϵ, both of which are important tuning parameters to the algorithm. At the end of the trajectory we get $(q(t + L\epsilon), p(t + L\epsilon)) = (\tilde{q}, \tilde{p})$, which becomes the proposed state for the chain and is used in the calculation of the Hastings ratio. The proposed state is then accepted with probability $\min(1, \exp(H(t+L\epsilon 0) - H(t)))$.

Here we state some facts about Hamiltonian dynamics that would help us understand why proposals generated from the simulation of Hamiltonian dynamics would form a Markov Chain sampler that has the right invariant properties.

First, the Hamiltonian flow, i.e. the mapping induced by the Hamiltonian dynamics, is reversible. That is, any mapping $T_s(x(t), y(t)) = (x(t + s), y(t + s))$ is bijective and has an inverse T_{-s}. Intuitively, it says that any particle whose trajectory follows Hamiltonian dynamics can be returned to its initial state (x_0, y_0) from current state (x_t, y_t) by reversing the sign of y_t and having the particle follow its tracjetory for time t. Reversibilty is one important as one of the necessary conditons for Markov Chains to have the correct invariant and convergence properties [82].

Second, the Hamiltonian flow T_s is a volume-preserving mapping. That is,

$$Vol(T_s(A)) = Vol(A)$$

where A is a measurable set in the augmented parameter space (e.g. $\mathbb{R}^{2d}$) which contains the support of joint probability density $p(x, y)$, and $Vol(\cdot)$ a volume measure. It makes it possible to calculate the Hastings ratio without involving the determinant of the Jacobian matrix of the mapping T_s , which might be tricky for most mappings. For Hamiltonian flows the determinant is just one. Third, the Hamiltonian is constant with respect to time, i.e., $\frac{dH}{dt} = 0$. From the physics point of view, this is simply the conservation of energy in a closed system. It keeps acceptance probability high if the discrete simulation of the Hamiltonian dynamics is accurate enough, which is equivalent to staying close to the true trajectory. This has been mentioned earlier and we will look at it from another angle here. The Metropolis acceptance step accepts the proposed state (x_t, y_t) as the next state in the Markov chain with probability $\min(1, \exp(-H(x_t, y_t) + H(x_0, y_0)))$, where (x_0, y_0) is the previous state in the chain, also the initial state in the Hamiltonian trajectory. In theory, because we are sampling from an energy level set

$$S = \{(x, y)|H(x, y) = E\}$$

for some fixed energy value E, the entire trajectory $(x_0, y_0), (x_\epsilon, y_\epsilon), \ldots (x_{\epsilon L}, y_{\epsilon L})$ where $\epsilon L = t$ has the same Hamiltonian

value $H(x_s, y_s)$. Then

$$-H(x_t, y_t) + H(x_0, y_0) = 0$$

and the proposed state is accepted with probability 1. When a numerical approximation to the correct Hamiltonian dynamics is used to find the proposed state, it inevitably deviates from the exact trajectory, and the order of that error determines how large the difference $-H(x_t, y_t) + H(x_0, y_0)$ will be. Although it is possible that (x_t, y_t) gives a smaller energy that the initial state and hence resulting in an acceptance probability of 1 as well, empirical evidence suggests that the Hamiltonian tends to oscillate along the trajectory traced out by leapfrog, and hence it is equally likely for $\exp(\Delta H)$to be larger or less than 1.

Having discussed the Hamiltonian function, we now look at one of its two components, the kinetic energy function $K(y)$.

Because the Hamiltonian function remains constant along its trajectory after the initial position and momentum is fixed, the only change in the value of the joint probabiltiy density function $p(x, y)$ comes from the initial resampling of the momentum variables y from its marginal distribution.

From the theory of auxillary variable sampling we see that there are infinitely many marginal distributions for the momentum variable y that would give the same target marginal distribution for the original parameter x. For example, if we restrict

$$y \sim \mathcal{N}(0, M)$$

where M is positive definite then any choice of M on the support gives a different kinetic energy and, by extension,a different Hamiltonian function. In the HMC literature, the focus has been on Gaussian kinetic energies, justified by empirical performance and intuition appealing to the central limit theorem. In Girolmani and Calderhead's paper [38] they experimented with the more heavy-tailed Student-T distribution and found it to perform not as well as Gaussian momentum distributions. For the rest of the discussion of HMC we assume a Gaussian distribution is used to model the marginal variables.

This is one of the two important tuning parameters of the sampler. Its selection mainly has to do with decorrelating the irregular geometry of the target distribution. For example, if we would like to sample from a multivariate Gaussian distribution with a highly skewed covariance, one could adopt an appropriately adjusted mass matrix M for the kinetic energy, and the induced sampling can be shown to be equivalent to sampling from a standard Gaussian with identity covariance in a linearly transformed parameter space. Sampling from a standard Gaussian distribution is easy for HMC and for MCMC in general, because all the variables are independent and on the same scale. Dimensionality has no impact on sampling in this case because each dimension is sampled independently and separately, and since the proposal distribution is Gaussian it is very easy explore the support of the target distribution in the 1-dimensional as it overlaps with the target distribution exactly.

To see how to transform the problem of sampling from a skewed multivariate Gaussian distribution to sampling from a standard Gaussian, let $x' = Ax$, where A is a invertible matrix. If we also transform the momentum variables

$y' = (A^T)^{-1}y$,then the Hamilton's equations for the transformed variables (x', y') will be the same as the original variables (x, y). As an aside, note this also implies that HMC is rotationally invariant, i.e. if A is orthgonal, $A^{-1} = A^T$, then reparametrizing the variables by multiplying (x, y) by A would result in exactly the same simulated trajectories if we keep all tuning parameters the same.

Suppose we want to sample from $x \sim \mathcal{N}(0, \Sigma)$. Let $y \sim \mathcal{N}(0, \Sigma^{-1})$, then the updates would be equivalent to using an identity covariance for y for drawing from a standard multivariate Gaussian distribution. This setting is ideal for sampling efficiency because the covariates are indepenent and the standard deviations for the x variables and that for the y variables are on the scale. Dimensionality has no impact on sampling in this case because each dimension is sampled independently and separately, and since the proposal distribution is Gaussian it is very easy explore the support of the target distribution in the 1-dimensional as it overlaps with the target distribution exactly. This is exactly the reparametrization trick for improving mixing in general MCMC methodology. This fact about the HMC suggests it would improve sampling efficiency if one has access to the covariance of the target distribution and its inverse, although this is true for MCMC sampling in general as well. (Pseudo Code for basic HMC)

Algorithm 2 HMC update

Input: potential energy function $U(x)$, its gradient $\nabla U(x)$, initial position x_0, ϵ, L,Σ

2. $x = x_0$
3. $y = \mathcal{N}(0, \Sigma)$
4. $y_c = y$
5. **for** $i = 1 : L$ **do**
6. $\quad y = y - \frac{\epsilon}{2} * \nabla U(x)$
7. $\quad x = x + \epsilon * \Sigma^{-1} y$
8. $\quad y = y - \frac{\epsilon}{2} * \nabla U(x)$
9. **end for**
10. $y = -y$
11. $H_c = U(x_0) + \frac{y_c^T \Sigma^{-1} y_c}{2}$
12. $H_p = U(x) + \frac{y^T \Sigma^{-1} y}{2}$
13. $u = Unif(0, 1)$
14. **if** $u < \exp(H_c - H_p)$ **then**
15. $\quad out = x$
16. **else**
17. $\quad out = x_0$

There are several reasons why HMC is better than random-walk Metropolis. First, it avoids random walks. Random walks are inefficient because successive proposals might double back to previous positions, thus limiting the range of the exploration of the support, whereas Hamiltonian trajectories would be in the same direction for many steps. Assuming independence, n steps of random walk would result in a random displacement of R units. Its variance

$$Var(R) = Var(R_1 + \cdots + R_n) = Var(R_1) + \ldots Var(R_n) = O(n)$$

is a linear function of n. Its standard deviation is $O(\sqrt{n})$, and can be interpreted as the typical distance a random

walk moves. As an analogy, if we have a 1-dimensional random walk $e_i \sim \mathcal{N}(0, \sigma^2)$ starting from the origin, then, with high probability the particle is likely to end up in the interval $(-\sqrt{\sigma}, \sqrt{\sigma})$. On the otherhand, since leapfrog steps are deterministic given initial position and momentum, the distance moved will be proportional to n. Second, HMC is much more efficient in high-dimensional settings. Because random walk Metropolis randomly proposes changes in the 2^p possible directions, it becomes exponentially more unlikely to move to an area of the support where there is significant mass. The acceptance probability would be extremely low and the chain becomes stuck. On the other hand, because HMC uses information of the gradient and an auxillary variable tuned to the curvature of the target distribution, the exploration stays close to the mode and the high density neighbourhood around it. There is a delicate balance between staying close to the mode and in the neighbourhood containing the mode. From almost any standard introduction to statistical learning [28] we learn about the curse of dimensionality, which says that the probability mass in the neighbourhood containing the mode of the density function decreases exponentially as the dimension increases. This means that simply sampling from around the mode will miss a lot of the support which contributes to the integral, but it remains an important starting point because the interesting part of the support lies just outside the immediate neighbourhood of the mode. The gradient draws the trajectory to the mode while the momentum variable keeps it from getting stuck there and moves it around with each resampling.

We have described some of the advantages of HMC, but unfortunately it has one great disadvantage, which is its sensitivity to tuning parameters. In the description of the basic HMC sampler above, we assume the stepsize ϵ, trajectory length L, and the momentum covariance M are known. These cannot be set arbitrarily as the performance of the algorithm can change completely when they are not set appropriately. HMC can perform no better than random walk Metropolis –itself an highly inefficient sampler– for high-dimensional distributions. The manual tuning of these parameters is the next part of our discussion.

The first parameter we will discuss is the stepsize ϵ. A small stepsize would cost more computation time but gives better approximation to the correct trajcetory. Worse still, when a too large stepsize is used, there is no constraints on how far the simulated trajectory would deviate from teh correct one. The simulated trajectory could quickly diverges to a region of the support where the Hamiltonian function blows up. There needs to be a mechanism in the algorithm to detect divergence due to the stepsize and signals us to possibly tune it down, otherwise the unstable trajectory would diverge to infinity in terms of its Hamiltonian function value and causes numerical instability. It is suggested by Neal that introducing a random jitter around the stepsize fat the start of each trajectory, for example, use $\epsilon + \epsilon * Unif(-0.1, 0.1)$ instead of ϵ, would help alleviate the problem, as sometimes the trajectory would be simulated using a stepsize under the stability limit for ϵ, which can differ between different parts of the support of $p(x, y)$. This also deals with the problem of accidentally have ϵL match the period of some variable or equivalently simulate a trajectory which returns to the original position at the end of its trajectory. This destroys ergodicity and makes the sampler biased.

The second parameter of interest is the trajectory length L. Once ϵ is fixed, it is essentially a proxy for integration time $T = \epsilon L$. It controls how much of the state space is explored at each iteration of the Markov chain. The longer we let the trajectory run the more of the state space would be explored, but letting it run too long we would eventually come back to the initial position and the exploration is wasted. Similar to the discussion about tuning ϵ, the tuning of

L could differ on different parts of the state space.

In the basic HMC sampler, we use the same ϵ and L throguhout the entire sampling process.

Neal suggests doing several preliminary runs with a combination of ϵ and L values using the acceptance rate as guidance. There is no guarantee of optimality and in even moderately complex distributions we never know whether we have sufficiently integrated the trajectory long enough. The optimal acceptance rate for HMC has been derived as 0.65. Optimality here is defined with respect to the cost of obtaining an independent point. See [73] for an intuitive derivation and see [6] for a theoretical derivation. This sort of handtuning is the same as the tried-and-true manual tuning strategies from the MCMC literature, and shares with them the disadvantage of requiring human intervention after each chain is sampled.

A heuristic for handtuning was given by Gelman et al [32]: first fix $\epsilon L = T$, then adjust ϵ and L by setting $\epsilon = \frac{\epsilon}{10}$ and $L = 10L$ and see the changes reflectd in diagnostics such as ESS and traceplots. Of course, one problem with this heuristic is that it fixes the integration time at 1, which might be too long or too short, depending on the contour of the target. The handtuning process can be made somewhat more efficient by listing all feasible values for ϵ and L and select the best combination by performing a grid-search. This only works if we accept the use of a one-dimensional diagnostic to quantify the quality of samples generated from a pair of $\{\epsilon, L\}$ values. This can be done by Bayesian optimization as well. Whether selected by handtuning or Bayesian optimization, the tuning process has no impact on the ergodicity of the sampler as we only use the samples generated once we have decided on a pair of tuning parameters (ϵ, L). This approach does not utilize information about the target distribution, relying only on inappropriate choices of parameters to be weeded out when the diagnostics indicate poor results.

In Neal's thesis he suggested a way to tune ϵ together with the momentum distribution under the assumption that the latter has a diagonal covariance matrix. We would set the stepsizes ϵ_i individually for each covaraite as

$$\epsilon_i = \eta(\frac{\partial^2 H}{\partial x^2})^{-\frac{1}{7}2}$$

where η is an extra tuning parameter to be set as $0 < \eta < 1$. In this modified version of HMC, we still have to tune for the integration time T via the trajectory length L.

In general the momentum distribution can be tuned by first exploring the density with a simple HMC sampler with standard Gaussian momentum distribution, then use the samples to estimate the covariance matrix of the target distribution, the inverese of which could be set as the covariance. The disadvantage of allowing arbitrary covaraince matrices for the momentum distribution is the $O(p^3)$ time complexity for inverting matrices, which quickly becomes problematic as the problem becomes high dimensional.

Another problem with the above approach is that it is only unsuitable when the target distribution can be approximately linearly transformed into a distribution which can be easily explored by a basis HMC sampler.If the target distribution has more complicated geometry that might differ in curvature in various parts of its support, a constant M is not enough to ensure effective decorrelation during sampling. This motivates the use of a location-dependent mass matrix $M(x)$ and is called the Riemannian Hamiltonian Monte Carlo. More on this later.

So far we have already touched upon the integration time in discussion of its connection with stepsizes and trajectory lengths. It is a tuning parameter that exists in both the theoretical and numerical version of HMC. In the theoretical version, we assume Hamiltonian dynamics can be simulated exactly, even then we would still have to decide the duration for which the imaginary particle is integrated. This is the amount of time we decide to simulate the trajectory in the augmented space (x, y) before projecting back to the sample space (x). In any numerical implementation this is subsumed in the product of the stepsize and the number of leapfrog steps ϵL. It influences mainly the exploration of the augmented space, where a trajectory is simulated along the level set $\{(x, y) : H(x, y) = E\}$.

Now begin discussion of RMHMC.

Another important trick in speeding up Hamiltonian Monte Carlo is Neal's window method [70].

First we select a window size $W < L$, then sample randomly $s \in \{0, 1, 2, \ldots, W-1\}$. Take the current (q, p) as (q_s, p_s), we geneerate the sequence $[(q_0, p_0), (q_1, p_1), \ldots (q_L, p_L)]$ deterministically by applying forward leapfrog steps (original leapfrog step with stepsize $\epsilon) for (\mathrm{q}_i, p_i), i > s$, and backward leapfrog steps (stepsize equal to $-\epsilon$.). Then the acceptance window is a sequence of W (q_i, p_i) pairs at the end of the trajectory, more specifically,

$$\{(q_{L-W+1}, p_{L-W+1}), \ldots, (q_L, p_L)\}$$

And the rejection window the W pairs

$$\{(q_0, p_0), \ldots, (q_{W-1}, p_{W-1})\}$$

Set the probability of accepting the acceptance window as

$$\min(1, \frac{\sum_{i \in A} P(q_i, p_i)}{\sum_{j \in R} P(q_j, p_j)}) = \min(1, \frac{\sum_{i=L-W+1}^{L} P(q_i, p_i)}{\sum_{i=0}^{W-1} P(q_i, p_i)})$$

where A, R are the set of indices for pairs belonging to the acceptance window and rejection windows respectively. We are essentially choosing the acceptance window with probability equal to the ratio of the sum of probabilities of each pair in the acceptance window to the sum of probability of pairs in the rejection sequence.

Once a window has been chosen, we select a pair among the W pairs in the window with probability weighted by $P(q_i, p_i)$. That is, suppose the rejection window is chosen, we chose the new state to be (q_i, p_i), $i \in 0, \ldots, W-1$ with probability

$$\frac{P(q_i, p_i)}{\sum_{j=0}^{W-1} P(q_j, p_j)}$$

It shows that even when we don't accept the proposal window near the end of the trajectory (acceptance window), there is still a probability of moving the chain away from its current state. This method generates a Markov Chain that leaves the target distribution invariant, and has been applied by Neal in neural network models to increase acceptance probability of the HMC sampler[70].

Another trick that helps to speed up the HMC sampler is data subsampling, also denoted partial gradient in Neal's

thesis [72]. Assuming there are n data points, the potential energy function $U(q)$ can be written as

$$U(q) = -\log(p(q)p(q|data) = -\log(p(q)) - \log p(q|data) = -\log(p(q)) - \sum_{i=1}^{n} \log p(q|data_i)$$

where $p(q)$ is the prior density function and $p(q|data)$ is the likelihood function. Note that the finite series in the last expression above can be approximated as

$$\sum_{i=1}^{n} \log p(q|data_i) \approx \frac{n}{k} \sum_{i \in I} \log p(q|data_i)$$

where I is a subset of $\{1, \ldots, n$ of size $\frac{k}{n}$. More pratically, if we divide the datapoints into M equally sized subsets. Then for any such subset S, the potential energy function can be approximated by

$$U(q) \approx -\log(p(q)) + M \sum_{i \in S} \log(p(q|data_i))$$

Note to preserve the correct invariant distribution, the calculation of the potential energy function in the Hastings ratio must be done exactly. randomly varying the trajectory length is a recommended part of standard HMC methodology

HMC is invariant to rotation. Randomly choose stepsize. Shortcut method. Mapping Multivariate Gaussian distribution. width of the distribution in the most constrained direction. Square root of the smallest eigenvalue of the covariance matrix for q. Quote: HMC is valid as long as the dynamics is simulated using a method thatis reversible and volume-preserving.

Tuning the HMC is tricky: for optimal stepsize see [6]

Understanding of optimal HMC stepsize, number of leapfrogs steps, number of leapfrog steps to reach nearly independent points. Ex: Grows as $O(d^{1/4})$.

Setting the stepsize that makes the leapfrog mapping stable.

Analysis: For one dimensional potential energy function $U(q) = \frac{q^2}{2\sigma^2}$, that is, normal distributed with variance σ^2. Writing out the matrix that encodes the linear mapping from $(q(t), p(t))$ to $(q(t+\epsilon), p(t+\epsilon))$ gives that the mapping is stable if $\epsilon < 2\sigma$, and diverges otherwise.

For the general multivariate q with arbitrary potential energy function, equivalent to a general density fucntion, we approximate the potential energy function with a second order taylor expansion, so that the q^2 term in the taylor expansion of $U(q)$ has coefficient $\frac{1}{2}\frac{\partial^2 U}{\partial q^2}$ By matching the two expressions we get

$$\sigma \approx (\frac{\partial^2 U}{\partial q^2})^{-1/2}$$

So by setting ϵ to that value, we are exactly in the middle of the domain of allowable stepsizes, equivalent to half of the boundary limit. Care has to be taken to note evalue the second derivative using values of the current parameters, because then the leapfrog steps are no longer reversible (i.e. get different stepsizes at two ends of a trajectory. So when

you reverse direction at the other end you don't return to the starting point).

This is only useful if you are doing a sort of blocks gibbs sampling, where you use HMC to sample from the low-level paramters while keeping the hyperparameters fixed, then sample the hyperparameters while keep the parameters fixed.

Might be much more difficult to select stepsize if we don't do this. Try to use Riemmanian Monte Carlo to overcome skewness of joint distribution of hyperparameters and parameters.

Partial momentum:

Suppose the kinetic energy has the form $K(p) = p^T M^{-1} p/2$,, then p has the same marginal distribution as

$$p' = \alpha p + (1 - \alpha^2)^{1/2} n$$

where $\alpha \in [-1, 1]$, and $n \sim \mathcal{N}(0, M)$. It is claimed that only a small improvement is obtained with this generalization.

Leapfrog does not preserve volume in the case where the potential energy and kinetic energy functions interact.

Symplectic = volume-preserving Tempered trajectory.

During a trajectory (full L leapfrog steps), for the first half of it, muliply each momentum variable before updating it by $\sqrt{\alpha}$ where α is slightly larger than 1, then follow by an exact number of division by $\sqrt{\alpha}$ in the second half of the directory. [71]

Neural networks

Deep learning, a field in machine learning studying multilayered neural network models, has become quite successful in artificial intelligence tasks and a lot of research has been, and is currently underwayto improve our understanding of these models and apply them better and to more fields. See [46, 54] a general introduction into the methodology and application of deep learning, and [88] for an in-depth literature review. It is futile to attempt a complete overview of the field because it is too vast but we will provide some pointers that reflect what the author finds important and some motivations. Before there was deep learning, a term which only came about in the mid 2000s, there were Neural Networks [11, 80], which were also hugely popular in the 90s, but ultimately could not scale up. The main reason for this was the difficulty of training neural networks of more than one or two hidden layers. Traditional optimization techniques like stochastic gradient descent with backpropogation did not improve empirical test error for networks with more layers,not until ideas like pre-training to initialize optimization were experimented could we fit "deep" neural networks with more than 2 hidden layers. This restarted the field, now rebranded as Deep learning, and innovations like using the rectified linear units as activation functions [68] to allow training deep networks without pre-training, and using GPUs to speed up practical training process [51], as well as the creation of numerical libraries like Theano [4], Caffe [47], Tensorflow[1], to name a few, that make building and training neural network models much easier by automating the backpropogation step with symbolic differentiation [3] and handles the transition between CPU and GPU computation modes. Suppose we have n observed datapoints $\{y_i, x_i\}$ where $X_i \in \mathcal{R}^p$ is a p-dimensional vector,

then a neural network models the data as follows:

$$y_i = \prod_{j=1}^{L} g_j(V_j f_j(W_j))x_i$$

where L is the number of layers of the network, W_j a $n_j \times m_{j-1}$ weight matrix, V_j a $m_j \times m_{j-1}$ weight matrix, and each g_j,f_j a pair of activation functions that are usually non-linear functions. There are usually no restrictions on what these activation functions must be other than on g_L which must match the data type of the y_i's. For example, if y_i are binary observations then we would like the last output activate function to be a logistic function so that it maps output to strictly $[0, 1]$, just like in logistic regression. Popular activation functions in the classical neural network literature includes the logistic function and the hyperbolic tangent function, but in modern deep learning the Rectified linear unit (RELU) has come to dominate the field. It has a simple form of $g(x) = \max 0, x$, which is straightforward to do backpropogation with despite a single point of nondifferentiability, and it was found to improve optimization enormously. It's discovery has allowed training neural network models with many more layers than 2 with random initialization of weights, getting rid of the need for pre-training. While neural networks can have different number of layers and number of hidden units within each layer, theoretically only one layer is sufficient to approximate any function as we increase the number of hidden units [45]. For Bayesian neural networks, where one put a Gaussian prior distribution on the weights of model, it has been shown that for a one-hidden-layer model, the prior converges to a Gaussian process as the number of hidden units converge to infinity [72].This gives a neat intrepretation of bayesian neural networks. Further work by [29], shows that heuristic training techniques such as dropout are actually are actually doing variational inference with Deep Gaussian Process [22], a process where several GPs are stacked one after another. The most exact way to perform bayesian inference for BNNs is still MCMC sampling as pioneered in Neal's thesis. In his work he put a hiearchical prior on the weights of the network and sampled the weights with HMC and the hyperparameter with block-Gibbs updates. One of his graduate students attempted tosample both of the weights and hyperparameters with HMC but did not get better performance, mostly because of the highly skewed distribution common in hierarchical models. With the invention of RMHMC and its successful application to sample more efficiently from hierarchical models, it would be interesting to try to see if this method can be adapted to sample from the posterior distribution of a BNN. For some theory on why, for the same number of hidden layers, it would be better to have multiple layers in a NN rather than a wide single-layer network, see [67]. enumerate

1.3 MCMC methods applied to hierarchical models

1. Centered vs uncentered parametrization in hierarchical models

2. Deal with skewness through Riemanian Manifold Hamiltonian Monte Carlo In the original Hamiltonian Monte Carlo, the kinetic energy is defined by the weight matrix M with momentum p as $p^T M^{-1} p$. In the Riemann Manifold Hamiltonian Monte Carlo (RMHMC), the mass matrix is generalized to a position specific metric tensor $G(\theta)$, when the parameter space is treated as a Manifold.

   The Hamiltonian is defined as

$$H(\theta, p) = -\mathcal{L}(\theta) + \frac{1}{2}\log(2\pi)^D |G(\theta)|) + \frac{1}{2}p^T$$

   In the original HMC, the momentum variables are normal auxillary variables independent of the position variables, which are usually the parameters of interest. In the RMHMC, the momentum variables are conditionally distributed as $\mathcal{N}(\imath, \mathcal{G}(\theta)$ given the current position.

   To simulate the Hamiltonian dynamics, we use

$$\frac{\partial H}{\partial p_i} = (G(\theta)^{-1} p)_i \tag{1.7}$$

$$\frac{\partial H}{\partial \theta_i} = \frac{\mathcal{L}(\theta)}{\partial \theta_i} - \frac{1}{2} Tr(G(\theta)^{-1} \frac{\partial G(\theta)}{\partial \theta_i}) + \frac{1}{2} p^T G(\theta)^{-1} \frac{\partial G(\theta)}{\partial \theta_i} G(\theta)^{-1} p \tag{1.8}$$

   Any implementation needs to discretize the simulation. The algorithm chosen is referred to as the Generalized Leapfrog.

$$p^{n+1/2} = p^n - \frac{\epsilon}{2} \nabla_\theta H(\theta^n, p^{n+1/2}) \tag{1.9}$$

$$\theta^{n+1} = \theta^n + \frac{\epsilon}{2}[\nabla_p H(\theta^n, p^{n+1/2}) + \nabla_p H(\theta^{n+1}, p^{n+1/2}) \tag{1.10}$$

$$p^{n+1} = p^{n+1/2} - \frac{\epsilon}{2} \nabla_\theta H(\theta^{n+1}, p^{n+1/2}) \tag{1.11}$$

   Note in the last three steps above we see the term to be generated on both sides of the equation. The authors recommend solving the equation by fixed point iteration. If

$$f_n = g(f_n)$$

   then to solve for f_n, we start with some initial guess, usually $f^* = f_{n-1}$, then iterate

$$f_n^1 = g(f^*), f_n^2 = g(g(f^*)), \ldots, f_n^k = g^k(f^*)$$

   k is set to a small number like 5 or 6.

   The choice of $G(\theta)$ can be very general. Techincally it gives the correct invariant distribution as long as it remains positive definite.

   In the examples given in the JRSS-B paper, the metric tensor chosen was the expected fisher information matrix, which has theoretical justification form information theory. It is recommeneded that the observed Fisher information be used when the expected value can't be calculated analytically, which is the case for practically all neural networks, except trivial cases. In those cases, we can modify the Empirical FIM to make it PD, by adding a constant multiple of the identity matrix, for example.

In Betancourt's paper, he demonstrated the effectiveness of the softabs metric in sampling from hierarchical models, which motivates us to test it with neural network models.

The softabs metric consists of the following. Let the SoftAbs map be

$$SA(X) = [\exp(\alpha X) + \exp(-\alpha X)] \cdot X \cdot [\exp(\alpha X) - \exp(-\alpha X)]^{-1}$$

Then the proposed metric is $SA(H)$,where H is the Hessian matrix taken with respect to the potential energy function, equivalent to the negative log posterior density in this case.

There are lots of implementation details that we can find in [7]. Notably, the 3rd order partial derivatives of the posterior density need to be calculated.

This sampler is reminescent of second order methods in optimization. The move from using only information about the gradient to the Hessian is like going from steepest descent to Newton's method. There is already in the literature of neural networks a body of work which looks at using a Riemmanian metric to define the descent direction. See [76, 65, 78] We will be experimenting with some of the more memory efficient metrics.

The same metrics can be applied to the Riemann langevin dynamics.

[38] To quote MALA, [**?**]

3. Softabs metric There is an easily implemented extension to the original HMC which does some tempering to help sampling from multi-modal distribution.

   Semi-separable HMC has limited applicability because it still requires inverting a Fisher information matrix, which may be singular for neural network models. Likely to depend heavily on the choice of mass matrices.

   Let θ be the model parameter and ϕ the hyperparameter such that the distribution of θ is parameterized by ϕ. Following the HMC methodology, we introduce corresponding auxillary variables r_θ and r_ϕ. In the HMC methodology following Neal's introduction. The auxillary variables are modeled to have Gaussian distribution marginally, with a diagonal precision matrix to be tuned manually. In RMHMC this is assumed to be the expected Fisher information matrix. Importantly, the precision matrices $G_\theta(\theta)$, $G_\phi(\phi)$ are dependent on the varaibles they are modeling. In semi-separable Hamiltonian Monte Carlo (SSHMC), the precision matrix has the form

$$G(\theta, \phi) = \begin{matrix} G_\theta(\phi, x) & 0 \\ 0 & G_\phi(\theta) \end{matrix}$$

   While giving up information about the local curvature of θ, this modeling assumption can deal with the correlation between the parameters and the hyperparameters, which gives enough improvement in mixing speed to yield better performance for hierarchical models, as shown in the paper by Zhang and Sutton.

   The Hamiltonian

$$H(\theta, \phi, r_\theta, r_\phi) = U(\theta, \phi) + K(r_\theta, r_\phi | \theta, \phi)$$

can then be rewritten as the sum of two Hamiltonians

$$H = H_1(\theta, r_\theta) + H_2(\phi, r_\phi)$$

Even though the original Hamiltonian H is not separable, the two Hamiltonians that sum to H are separable when looked at separately. This enables the simple leapfrog to be used instead of the more time consuming generalized leapfrog.

Algorithm 3 SSHMC

Input:$L, \epsilon, \theta_0, \phi_0$

4. Initialize $r \sim \mathcal{N}(0, G_\theta(\phi, x))$ and $r_\phi \sim \mathcal{N}(0, G_\phi(\theta))$
5. **for** ℓ in $1, 2, \ldots, L$ **do**
6. $(\theta^{\ell+\epsilon/2}, r^{\ell+\epsilon/2}) = leapfrog(\theta^\ell, r_\theta^\ell, H_1, \epsilon/2)$
7. $(\phi^{\ell+\epsilon/2}, r^{\ell+\epsilon/2}) = leapfrog(\phi^\ell, r_\theta^\ell, H_2, \epsilon)$
8. $(\theta^{\ell+\epsilon}, r^{\ell+\epsilon}) = leapfrog(\theta^{\ell+\epsilon/2}, r_\theta^{\ell+\epsilon/2}, H_1, \epsilon/2)$
9. Acceptance-Rejection Step
10. **end for**

1.4 approximate inference for fast training and prediction

Because of the challenges of MCMC sampling from the posterior distribution of bayesian neural networks described above, much work has been done to bypass MCMC sampling altogether when performing bayesian inference on these models.

Variational inference [**?**, 27, 12] is a popular class of methods designed to do just that. Its basic idea is to approximate the complicated posterior distribution by a simpler parametric distribution avaialble in closed form. The approximating distribution is chosen by an optimization procedure minimizing the Kullback-Leibler (KL) divergence between the approximating distribution and the posterior distribution. Numerical integration such as MCMC is intuitively more difficult than optimization because it requires knowing global properties of the target function, whereas optimization can be done using local information (gradient descent) only. Immediately, one observes a disadvantage of variational inference compared to MCMC sampling – inference done using the varitional distribution is biased,in the sense of not targeting the posterior distribution of interest, whereas that with MCMC samples is unbiased and asymptotically consistent as the length of the chain goes to infinity [82]. While this is might seem highly undesirable, given a finite computational budget, variational inference allows fast training and prediction, which might win out over MCMC sampling, especially if the model size severely limits the number and quality of the MCMC samples. The bias-varaiance trade-off suggests we should not disregard the variational approach immediately. An interesting experiment would be to test the above hypothesis, that variational inference yields better predictive performance than MCMC sampling in the early training phase, and that it would take a disproportionate amount of time and memory storage for it to match the performance of the former.

More specifically, in variational inference, we take a parametric family of distributions $q(w|\theta)$, called the variational distributions, and try to minimize its KL divergence with the posterior distribution calculated under the original model

$$KL(q(w|\theta)||p(w|D)) = -\int q(w|\theta)\ln(\frac{p(w|D)}{q(w|\theta)})dw = KL(q(w|\theta)||p(w)) - E_{q(w|\theta)}[\ln(p(D|w))]$$

The variational distribution is usually chosen to be a member of the exponential family. A popular choice is the diagonal Gaussian distribution, that is

$$q(w|\theta) = \prod_{i=1}^{p} \mathcal{N}(w_i|\mu_i, \sigma_i^2)$$

where $\mathcal{N}(w|\mu, \sigma^2)$ is the density function of normal distribution with mean μ and variance σ^2.

The minimization can be done by usual optimization techniques, with extension to Because the KL divergence is asymmetric, one is tempted to consider minimizing it but with $p(w|D)$ and $q(w|\theta)$ swapped in the order of evaluation. There is some subtlety with regards to the usual multimodality of the posterior distribution that calls for a different optimization technique. This inspires as class of approximation inference algorithms called expectation propogation [66].

Finally, there is the simple idea of approximating the the posterior distribution by fitting a gaussian density around the posterior mode, this is known as Laplace's method. It is first applied to BNNs by Mackay in [63]. In [96], the author compared the predictive performance of Laplces's method against HMC and found the latter to perform slightly better, with more marked improvement on smaller datasets. An interesting point to note about this method is that by using a Gaussian approximation we have to calculate its covariance matrix by using the Hessian. Since neural network models are actually singular the Hessian is not positive definite everywhere and adjustments have to be made to make it work. In [42] the authors compared Laplace's method with several approximate inference methods as well as HMC and found it to perform less well than the others. However, they constrained the covariance matrix of the Gaussian approximation to be diagonal to save computational time. Understandibly this leads to worse approximation. On the other hand, one is not forced to choose between a diagonal covariance or the full-rank one. Quasi-newton approximations are possible, and fits neatly within the optimization pipeline.

Variationa inference for bayesian neural networks started with the work of [43], who developed a variational inference algorithm using the language of information theory. He used the concept of minimizing the description length of the weight, which is equivalent to minimizing the KL divergence as described above.

Graves' work [39] built on the the Hinton's paper and introduced a stochastic gradient variational method. However, the stochastic gradient estimates are biased and the prior is limited to be Gaussian. In [13] this is further improved upon using the re-parametrization trick [77, 49, 79]. This yields the Bayes by Backprops (BBB) algorithm.

The Expectation propogation (EP) approach to approximate inference in BNNs was experimented in [48] but saw little follow-up because they method the authors developed was batch-only and could not scale to large datasets.A stochastic version was designed by [42], called the Probabilistic Backpropogation (PBP), and was shown to perform well on a

wide range of test datasets. However, it has the limitation of being applicable only to regression problems. In [37] it is extended to multiclass classification problems through a monte carlo approximation.

In Myshkov et al's submission to a NIPS workshop (cite), the authors analyzed the performance of a few approximate inference algorithms for BNNs against the "gold-standard" that is the HMC, among those tested were PBP and EP. And they did not perform too poorly against MCMC methods. More analysis of this work to come later.

In Bayesian dark knowlege, we also try to minimize the KL divergence, but do so in a way that compresses model knowledge.

Why do we assume the prior distribution of weight parameters to be normal centered at 0?

Study of prior distributions in neural networks.[52, 95]

1.5 Scaling MCMC methods to large datasets

In modern deep learning, fitting of neural network models usually is done by stochastic gradient descent (SGD) [75]. This is exactly the same as regular gradient descent [102], except at each weight update and random subset(batch) of datapoints are sampled to be the full observations, and the sequence of stepsizes over the optimiztion process must decrease while satisfying the property

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \sum_{t=1}^{\infty} \epsilon_t^2 < \infty$$

Theory from the stochastic optimization literature [81] guarantees convergence to a local minimum. However, as per the norm in the optimization literature, the convergence property is usually only provable for easier classes of problems (.e.g. convex, quadratic functions) to which the likelihood of neural network models does not belong. Nonetheless, SGD has the advantage of easy implementation and low memory requirement (O(p)), and hence remains the dominant optimization method in deep learning.

The success of SGD lies in being able to avoid evaluating the full likelihood, which has contribution from each data point, at each update. The problems that neural network models tend to be fitted are usually in the high-sample-size, high-dimensional regime, hence a full evaluation would slow down the algorithm significantly. A stochastic gradient extension to MCMC was first experimented by [101], where the author proposed to use the following updates :

$$\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2}(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i \in D} \nabla \log p(x_i|\theta_t)) + \eta_t, \eta \sim \mathcal{N}(0, \epsilon_t)$$

where D is a random subset of $\{1, 2, \ldots, N\}$ of size n sampled at each update of θ. We assume there are N observations in total. The stepsizes are assumed to decay following the constraint stated above for SGD. The authors recommended setting $\epsilon_t = a(b + t)^{-\gamma}$ with $\gamma \in (0.5, 1]$.

Marginal predictive distributions or any expectation taken with respect to the posterior distribution can be calculated by

$$E[f(\theta)] \approx \sum_{t=1}^{T} \epsilon_t f(\theta_t) / \sum_{t=1}^{T} \epsilon_t$$

This estimator has lower variance than the naive alternative $\frac{1}{T}\sum_{t=1}^{T} f(\theta_t)$ and is unbiased as well.

The author proved that as the stepsizes ϵ_t goes to 0 the samples $\{\theta_t\}$ converge tothe Langevin dynamics targeting the posterior distribution. Of course, in practice we set lower bound on ϵ_t and stop decreasing the stepsize once it is small enough. Also, the fact the we are not doing the acceptance-rejection step means there will be a bias in our posterior samples, which unlike the samples drawn from a Metropolis-Hastings sampler, does not decrese to 0 as we sample from the chain longer.

Algorithm 4 Stochastic Gradient Langevin Dynamics

Input: N,n,$\theta_0, \{\epsilon_t\}_{t=1}^{T} Initialize \theta_0$

for t in 0:T **do**

  Sample random subset $\{x_{ti}\}_{i=1}^{n}$

  Sample $\eta_t \sim \mathcal{N}(0, \epsilon_t)$

  $\theta_{t+1} = \theta_t + \frac{\epsilon}{2}(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta_t)) + \eta_t$

end for

Simulating from the discrete approximation to a Langevin dynamics is the basis for Langevin Monte Carlo or Metropolis adjusted Langevin Algorithm, which includes an acceptance-rejection step at each update. This can also be seen as a special case of HMC where only one leapfrog step is used ($L = 1$). It has been demonstrated that HMC is much more efficient than LMC because it avoids random walk. (Show $O(d^{4/3})$ vs $O(d^{5/4})$ kind of results). This inspires Stochastic Gradient Hamilotian Monte Carlo (SGHMC) [20], which extends HMC the way SGLD extends LMC.

Algorithm 5 Stochastic Gradient HMC

Input: $\eta, \alpha, n, T, m, \theta_0$

Initialize θ_0, v_0

for t = 1:T **do**

  $\theta^0 = \theta_{t-1}$

  $v^0 = v_{t-1}$

  for i = 1:m **do**

    $\theta^i = \theta^{i-1} + v^{i-1}$

    Sample $z \sim \mathcal{N}(0, 2(\alpha - \hat{\beta}\eta)$

    Sample minibatch and calculate $\tilde{U}(x)$

    $v^i = -\eta\tilde{U}(x) - \alpha v^{i-1} + z$

  end for

  $\theta_t = \theta^m$

  end for

Important points to retain: connection to sgd with momentum. dont know how to set C matrix. noise of gradient B not actually estimated in any of the examples in the paper claiming as stepsizes goes to zero the noise goes to zero as well. The estimation can be done using along diagonal approximation, or low-rank approximations.

Preconditioned SGLD Sampling via Stochastic Gradient Fisher-Scoring (SGFS) was proposed in [2].

Algorithm 6 pSGLD

Input $\{\eta_t\}_{t=1}^{T}, \lambda, \beta, \theta_0$

30. Initialize $v_0 = 0$
31. **for** t in 1:T **do**
32. $\tilde{f}_t = \nabla \tilde{U}_t(\theta)$
33. $v_t = \beta v_{t-1} + (1-\beta)\tilde{f}_t \odot \tilde{f}_t$
34. $G_t^{-1} = diag(1 \oslash (\lambda 1 + v_t^{1/2}))$
35. $\psi_t \sim \mathcal{N}(0, \eta_t G_t^{-1})$
36. $\theta_t = \theta_{t-1} + \frac{\eta_t}{2} G_t^{-1} \tilde{f}_t + \psi_t$
37. **end for**

Sampling via Stochastic Gradient thermostat. Sampling via pre-conditioned Stochastic Gradient langevin dynamics.

Algorithm 7 SG Thermostat

Input: η, a, θ_0

38. Initialize $u_0 \sim \mathcal{N}(0, \eta I), \alpha_0 = a$
39. **for** t in 1:T **do**
40. Evaluate $\nabla \tilde{U}(\theta_{t-1})$
41. $u_t = u_{t-1} - \alpha_{t-1} u_{t-1} - \nabla \tilde{U}(\theta_{t-1})\eta + \mathcal{N}(0, 2a\eta)$
42. $\theta_t = \theta_{t-1} + u_t$
43. $\alpha_t = \alpha_{t-1} + \frac{1}{n} u_t^T u_t - \eta$
44. **end for**

Algorithm 8 Preconditioned SGLD

Inputs:$\{\epsilon_t\}_{t=1}^{T}, \lambda, \alpha, T$

45. Initialize
46. **for** t in 1:T **do**

Applying MCMC methods to large datasets often involve 2 problems. First, evaluating the unnormalized posterior density

$$p(D|\theta) = \prod_{i=1}^{n} p(x_i|\theta)$$

is equivalent to passing through all datapoints, which alone can make the sampling algorithm unacceptably slow since all versions of the Metropolis-Hastings sampler requires such evaluations for proposing new samples. Second, models that require MCMC samples when a large number of datapoints are available are necessarily singular models. Using the terminology of [99], regular models are models for which the bayesian central limit theorem [53] holds, that is, when the number of observations goes to infinity the posterior distribution of θ converges to a Gaussian distribution. Models that are non-regular are called singular models. A lot of methods designed to scale MCMC [74, 89, **?**] ultimately rely on this asymptotic normality for validity. Bayesian logistic regression is a classical model for which the CLT holds and is used in many papers to demonstrate the effectiveness of newly proposed large-scale MCMC methods. However, the very fact that asympotitic normality holds means there is little added utility to drawing MCMC samples from the

posterior distribution, since a normal approximation would suffice. On the other hand, for singular models there is no guarantee that the estimates converge to the target posterior distribution.

Asympotic normality is also assumed for the application of stochastic gradient mcmc methods to neural network models [101, 20, 2, 24, 62]. Even though neural networks are singular models where the conditions for the central limit theorem do not hold, these methods have shown good predictive performance on a variety of datasets. This is opposite of what one would expect and merits further investigation.

For now we focus on the class of datasubsetting or parallelisable MCMC sampler. This consists of splitting the data points into subsets and sample from the respective posterior distributions corresponding to each of these subsets. The problem with applying this to neural network models, notwithstanding the assumption of bayesian central limit theorem, is the difficulty of tuning the MCMC samplers. If many partitions/parallel servers are required to speed up the sampling prcoess, it makes monitoring and diagnosing convergence an even harder task. Essentially, one would have to treat these samplers as a black-box [16], which is fine when there is abundant labeled data for sanity checks with classification and/or regression tasks on hold-out sets. This makes it unsuitable for unsupervised learning. Even in the case where the scaled MCMC samples yield better predicitve performance, it is difficult ot interpret the results, seeing that we could not claim the algorithm is sampling exactly from the target distributions, not even asymptotically.

First, we review the data subsampling approach. (Talk about Consensus MCMC). What happens when some subset servers mix poorly?

Let $\mathbf{x}$ be the full data of size N and θ be the model parameters. Assuming independence of the data, for $S \leq N$, the posterior distribution of θ can be written as

$$p(\theta|\mathbf{x}) \propto \prod_{s=1}^{S} p(\mathbf{x}_s|\theta)p(\theta)^{1/S}$$

where $\{\mathbf{x}_s\}_{s=1}^{S}$ is a disjoint partition of the original data. Technically, in this formulation, each subset can be of variable size. But for the purpose of computational efficiency we assume $|\mathbf{x}_s| = \frac{N}{S} \forall s$.

Now, for each subset $\mathbf{x}_s$, we draw G samples $\theta_{s1}, \ldots, \theta_{sG}$ from $p(\theta|\mathbf{x}_s) \propto p(\mathbf{y}_s|\theta)p(\theta)^{1/S}$. To combine these $S \times G$ samples we do

$$\theta_g = (\sum_s W_s)^{-1} \sum_s W_s \theta_{sg}$$

where W_s is a weighting matrix. For models where the posterior distributions $p(\theta|\mathbf{x}_s)$ are Gaussian. $\sum_s = Var(\theta|\mathbf{x}_s)^{-1}$ is exact. By the bayesian central limit theorem, for regular models the normal approximation for the posterior distribution is arbitrarily good, so we can use the inverse of the empirical variance as W_s. There is no theoretical guarantee that normal approximation to the posterior distribution for neural network models is accurate.

For high dimensional θ, we can ignore the correlations and use the marginal variances for W_s.

This is a general methodology that can be coupled with any sampling methods, be it MCMC or exact sampling.

Scaling based on splitting techniques from Neal. Random data subset used to approximate the Hamiltonianduring leapfrog step but uses the full dataset for calculating the Hastings ratio. Retains convergence to invariant distribution.

Algorithm 9 Consensus Monte Carlo

Input: $S, G, \mathbf{x}$

47. Divide $\mathbf{x}$ into S disjoint parts $\{\mathbf{x}_s\}_{s=1}^S$
48. Draw samples $\theta_{sg} \sim p(\theta|\mathbf{x}_s) \propto p(\mathbf{x}_s|\theta)p(\theta)^{1/S}$ for $g = 1, \ldots, G, s = 1, \ldots, S$
49. Calculate W_s for $s = 1, \ldots, S$.
50. Combine the draws for $g = 1, \ldots, G$:$\theta_g = (\sum_{s=1}^S W_s)^{-1}(\sum_s W_s\theta_{sg})$

The idea of simulating Hamilotnian dynamics without the metropolis acceptance step was first experimented in the statistics/machine learning literature by Neal [69] in a batch MCMC sampler for the posterior distribution of a BNN. He found that with careful selection of the leapfrog stepsize, the biased samples achieve similar predictive performance as the full MCMC samples. Therefore, it seems to suggest that for predictive tasks .e.g. regression or classificaiton where observations are abundant, we could use a validation set (predictive performance) to select tuning parameters, hoping to achieve trade off a bit of bias for better predictive performance.

Now we review some important samplers in the stochastic gradient MCMC literature. Stochastic Gradient Langevin Dynamics (SGLD) was first introduced by [101]. Stochastic gradient Hamiltonian Monte Carlo Got rid of the metropolis acceptance step. No longer sampling from the exact distribution. Second order method, needs to estimate fisher information matrix.

[20]

Problem is you are not saving that much computation time if you now end up with a second-order method, and if you want to sample exactly from the target distribution you still have to evaluate the Hastings ratio using the entire dataset.

There is a tendency in the stochastic gradient MCMC literature to omit the Metropolis correction step.

See [24] Quote: "If the stationary distribution is not the target distribution, a Metropolis-Hastings (MH) correction can often be applied. Unfortunately, such correction steps require a costly computation on the entire 2 dataset. Even if one can compute the MH correction, if the dynamics do not nearly lead to the correct stationary distribution, then the rejection rate can be high even for short simulation periods h. Furthermore, for many stochastic gradient MCMC samplers, computing the probability of the reverse path is infeasible, obviating the use of MH. As such, a focus in the literature is on defining dynamics with the right target distribution, especially in large-data scenarios where MH corrections are computationally burdensome or infeasible. " [62] This is total crazy talk (not sampling from the target distribution exactly!)

Perhaps if we can quantify the bias somehow we can make it work.

Note the data subsampling approach does not work for Gaussian Process models because of the calculation of the covariance inducing dependence among observations [26]. [93] Proves consistency of SGLD despite throwing away Hastings ratio step. Tradeoff is slower exploration.

Stochastic gradient Hamiltonian Monte Carlo [20] relies on the assumption that the gradient of the log target density is normally distributed, which in turn comes from assuming independence of the observations x and appealing to the

central limit theorem.

Same CLT assumption about the gradient is made in the SGLD paper [101].

Some more theory behind SGLD and SGHMC [62] Stochastic gradient thermostat : an improvement on SGHMC by getting rid of the need to estimate a covariance matrix.

First introduced here[24], exteneded[31] to have multiple thermostat parameters instead of just 1 in the original version, some analysis as well as introduction of a higher-order numerical integrator here[57, 19].

Preconditioned SGLD [56] uses information of the curvature. Can be thought of as extension of SGLD same way RHMC extends HMC. Does not use full Fisher information matrix. Uses diagonal approximation to reduce complexity. Stochastic expectation propogation.

Stochastic gradient fisher scoring [2]

Using stochastic gradient is only one way of scaling and automating MCMC. Also can try adaptive MCMC methods[85]. Relevant dynamical versions include [98],where Bayesian optimization [64, **?**, **?**] is used to tune the HMC sampler.

stochastic gradient quasi netwon langevin dynamics requires extra tuning paramters with little guidance on how to tune them [90]

preconditioned sgld introduces bias[56].

Expectation propogation. Original paper[66], Gelman and collegues wrote a paper on it [34]

Stochastic version discussed here [58] See also [23] for another study of expectation propogation in the large sample size context.

A stochastic natural gradient extension has been devised and applied to bayesian neural network models [92] Probabilistic Backpropogation is a special case of SEP applied to Bayesian neural networks [42].

Laplace Approximation. It is based on the second order approximation of the log-likelihood of the posterior density around its mode.

Suppose the log posterior density $f(x)$ has a mode at x^*, then near x^* we have

$$f(x) = f(x^*) + \nabla f(x)|_{x=x^*}(x - x^*) + \frac{1}{2}(x - x^*)^T H(x - x^*)$$

where H is the Hessian matrix of $f(x)$ evaluated at $f(x)$, that is,

$$H_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}|_{x=x^*}$$

Automatic tuning of MCMC algorithms:

If datasubsetting / parallel MCMC is to be sucessful, the intensely manual task of tuning the samplers must be automated. The Stan software comes with an optimized U-turn sampler [44] that automates the selection of trajectory

length. There is also the adaptive MCMC literature which have created adaptive methods for Hamiltonian Monte Carlo. However, all of those methodologies mentioned above is batch only, which limits applicability.

Also, the theory of adaptive HMC relies on assumptions about the HMC which is still not proven on general state spaces (e.g. uniform or geometric ergodicity).

There is also an older version of adaptive HMC which uses predictive performance to adjust tuning parameters adaptively. The two need to be compared, along with immediate stochastic gradient extension. Propose a stochastic gradient extension to adaptive HMC. Replace the sampler by stochastic gradient version.

Adaptive MCMC means varying the tuning parameters (stepsize, trajectory length) while sampling along the chain. Since all samples collected need to converge to the target distribution, the requirement is much more strict on the sampler. On the other hand, we can simply consider the sampler as a black-box and select tuning parameters using bayesian optimization.[91].

Compare different acquisition functions for choosing tuning parameters in MCMC algorithms, in batch and stochastic gradient settings. Use acquisition functions in the Snoek paper, as well as expected squared jump distance.

Can it beat grid search? Compare ESS/L.

Algorithm 10 Bayesian optimization

51.

52. p is the dimension

1.6 Bayesian Model Selection and Prediction

In a neural network model there are multiple hyperparameters left to be tuned be the user. These include the number of hidden layers, the number of hidden variables in each layer, the choice of activation functions, the use of convolutional layers in the case of convolutional neural networks, the prior distribution on the weights of the network. Each different combination of these hyperparameters constitute a model choice. In the neural network literature, usually a train-test dataset split is used. The full dataset with observations $\{\mathbf{y}, \mathbf{x}\}$ is split randomly into a training set $\{\mathbf{y}_{train}, \mathbf{x}_{train}\}$ and a test set $\{\mathbf{y}_{test}, \mathbf{x}_{test}\}$. Each model is fit on the training set and its predictive performance evaluated on the test set.

The model selection procedure above is a special case of cross-validation (citation). Suppose there are n data points, in $k - fold$ cross validation, $k \leq n$, the dataset is split into k disjoint subsets, then repeatedly one subset is used as test set and the remaining sets used as training set. The extreme case is leave-one-out cross-validation (LOO-CV), where $k = n$. To calculate the log pointwise predictive density, $lppd_{loo-cv}$, we do as follows. For each split of the n split $\{d_i, d_{-i}\}$, where d_i is the ith datapoint $\{y_i, x_i\}$, and d_{-i} are the $n-1$ remaining data points. Then

$$lppd_{loo-cv} = \sum_{i=1}^{n} \log p_{post(-i)}(d_i)$$

where

$$p_{post(-i)(d_i)=\int p(y_i|x_i,\theta)p(\theta|d_{-i})d\theta}$$

which is the marginal predictive density of the ith data point, with θ integrated out with respect to the posterior density of θ having observed the remaining $n-1$ data points. In practice, the integration is performed approximately by drawing S samples from $p(\theta|d_{-i})$, denoted $\{\theta^1_{-i}, \ldots, \theta^S_{-i}\}$, and they form the estimate $\frac{1}{S}\sum_{s=1}^{S} p(y_i|x_i, \theta^s_{-i})$.

Then

$$lppd_{loo-cv} \approx \sum_{i=1}^{n} \log(\frac{1}{S}\sum_{s=1}^{S} p(y_i|x_i, \theta^s_{-i}))$$

The problem with applying this to model selection for neural networks is the simulation from n different posterior distribution required, each of these require tuning and monitoring to ensure low variance of the integral estiamte. For high dimensional distributions such as the posterior distribution of a neural network, simulation becomes more difficult because of potential correlation as well as the lack of tools for evaluating convergence of the joint distribution. Most convergence diagnostics require storing the entire Markov chain, or even simulating multiple chains. First, it is tedious to perform 10 or 5 convergence diagnostics(in the case of k-fold cross-validation for $k = 10$, or $k = 5$). Second, even with a single chain memory storage scales linearly with the number of dimensions p. For reasons discussed above this is infeasible for neural network models.

Holdout cross-validaiton as currently used in the neural network literature has higher variance than the k-fold and leave-one-out alternatives and is problematic especially if the sample size is small. It would be much more preferable to use an information criterion which requires only simulation from one posterior distribution, hence reducing the difficulty of convergence diagnostics and the memory storage. Traditional information criterai like the AIC, BIC or DIC rely on the asympotitic normality of the mle for its justification, which does not hold for non-identifiable (singular) models such as mixture models, hidden markov models, and more relevantly neural network models. WAIC is a recently introduced information criterion that works for singular models as well. The justification uses advanced mathematics beyond the scope of this thesis, notably tools from algebraic geometry and empirical processes. Hence, we do not attempt to give an intuitive explanation of the theory and instead take on faith its validity and its asymptotic equivalence to leave-one-out cross-validation [100].

The WAIC can be calculated as follows. Let $\{\theta_1, \ldots, \theta_S\}$ be S samples drawn from the posterior distribution $p(\theta|D)$ given all observed data points. Then

$$WAIC = lppd - p_{WAIC}$$

where

$$lppd = \sum_{i=1}^{n} \log(\frac{1}{S}\sum_{s=1}^{S} p(y_i|x_i, \theta^s))$$

is the computed log pointwise predictive density.And

$$p_{WAIC} = \sum_{i=1}^{n} V_{s=1}^{S}(\log p(y_i|x_i, \theta^s))$$

is the effective number of parameters, where $V_s = 1^S \log(p(y_i|x_i, \theta^s))$ is the empirical variance of the S $\log(p(y_i|x_i, \theta^s))$

for fixed i. Note that only nS points need tobe stored to calculate the WAIC. p_{WAIC} can be calculated differently, but here we follow the recommendation of Gelman et al and use the formula above.

WAIC can also be calculated by importance sampling, using the variational distribution as the proposal distribution [103].

Once we have selected a model, we can make predictions about new data using its marginal posterior predictive distribution. That is, given a new data point $\{x_{new}\}$, we can find its marginal posterior predictive density $p(y_{new}|D)$ as

$$p(y_{new}|D) = \int p(y_{new}|x_{new}, \theta)p(\theta|D)d\theta$$

If this is a classification problem of C classes we would evaluate $p(y_{new}|D)$ for all $y_{new} = 1, \ldots, C$, and if a regression problem a number of points on the domain of the output can be evaluated and a prediction made by drawing from the empirical distribution.

Multiple Modes

Neural network models are known to have multiple modes, although it is widely believed that most local modes have similar likelihood values to the global mode. In Neal's work on bayesian neural network, he also assumed that multimodality would not cause any problems in inference and in prediction.

Insert pseudocode.

Part II

Experiments

Chapter 2

Experimeints

Plan: Collect problems on which to fit neural networks of modest size.

Example 1: Mackay's robot arm's data.

This is a regression problem where the data is generated as follows:

$$y_1 = 2.0 * \cos(x_1) + 1.3\cos(x_1 + x_2) + z_1$$

$$y_2 = 2.0 * \sin(x_1) + 1.3\sin(x_1 + x_2) + z_2$$

where z_1, z_2 are independent Gaussian random variables of standard deviation 0.05 and x_1 is uniformly generated from $[-1.932, -0.453] \cup [0.453, 1.932]$, and x_2 is uniformly generated from the range $[0.534, 3.142]$. 200 training cases and 200 test cases were generated according to the equations above.

A neural network with a single hidden-layer of 16 hidden units was used to model the data.

Bayesian logistic regression with hierarchical prior. Following [104]

1. Sample weights with HMC, sample hyperparameter with Gibbs sampling.

2. Same as above with Windowed update.

3. Same as 1 but with tempering.

4. Sample weights and hyperparameter together with HMC. (STAN)

5. Softabs

6. Empirical fisher information matrix Compare ESS/L. Data: Same classification problem datasets from the RHMC paper [38], as well as simulated data. Funnel problem Compare ESS/L as well as marginal distribution of the hyperparameter, which is known in this case.

Bayesian neural network of more than 2 hidden units and of 1 hidden layer only. Test 1 layer network of 100 units with fixed normal prior and hierarchical prior. Data: Same regression problem datasets from the PBP paper [42].

For all the above problems it is straight forward to adapt code to compare manual hyperaparameter tuning with automatic tuning by STAN and by bayesian optimization. Compare ESS/L results for the final parameters selected.

Test robustness of consensus mcmc to poor mixing. Split data into a K subsets randomly, K being a manageable small number (5 or 8), then tune some posterior distribution samplers to mix well and some others don't. Test on both regular and singular models.

For neural networks being used to fit a small number of observations relative to its dimension, there is a problem of overfitting. In [30], it was shown that approximate bayesian inference via dropout helps to mitigate this problem. We argue that MCMC sampling would achieve even better performance.

Use only a small percentage (5 or 10 percent) of the MNIST dataset and perform batch inference with BNN. Try both fixed prior and hierarchical priors. Use the best sampler found from earlier experiments. Compare with gradient descent and dropout.

Done with MCMC Bayesian model selection. Bayesian neural networks can also be used to model data when the size of the training set is small. In this setting frequentist training of neural networks suffers from overfitting. Practioners who use neural network would like to make predictions for new data $\{x_{new}\}$. In principle, before making predictions the practioner should first decide on the model(s) to be used to fit the training data. While bayesian model averaging (BMA) is an interesting idea which promises to improve prediction accuracy, assignign prior to different models is difficult to justify and may appear arbritary. More importantly, for large models the time and memory complexity required to sample from the posterior distribution or the variational approximation thereof, as well as sample from the predictive distribution, would quickly become unmanageable as the number of models increases.

For this reason, model selection is usually carried out, where a model is selected and then trained to make predictions on new data. Because there are infinitely many combinations of different tuning parameters and model structure paramters, an exhaustive search is impossible and instead a heuristic combinations of tuning paramters are chosen to form the candidtate models. Usually a holdout set (test set) is used to estimate the predictive performance of the model on new data. The model with the lowest average predictive error is then chosen.

In bayesian statistics [32] cross-validation, especially leave-one-out cross-validatino is recommended for model selection. It comes with a high computational cost, but remains the most principally sound method for estimating the out-of-sample log predictive density. WAIC(Widely applicable information criterion) is derived using singularity learning theory and estimates consistently the loocv, even for singular models. Traditional information criteria like AIC, BIC and DIC relies on the the asymptotic normality of the mle, which only holds if the Fisher information matrix is non-singular at the mle. While the justification of WAIC relies on heavy algebraic machinery, to apply it only requires being able to draw samples from the posterior distribution under a model and evaluate the likelihood. In the machine learning/deep learning community, model selection is usually done by comparing the predictive accuracies of the different trained models on a holdout set. The usual justification of this practice is that if the dataset exhibits enough data redundancy then this is enough to estimate the test accuracy. However, this is certainly dependent on the particular dataset and the size of the model one would like to use to fit the data. The most statistically principled practice would be too use

all available data to train models and then select the best model using cross-validation. Efficeint approximation to the cross-validation log density like the WAIC is therefore deemed desirable for model selection.

An important statistical question is therefore as follows: for neural network models, is the use of a holdout set for model selection consistent with best practice? That is, do these two methods make the same model selection decision ? What happens in the small data regime? I suspect the utitilty of WAIC would be the greatest in small-to-medium data size settings, where a randomly chosen subset that is 25 percent the size of total dataset is unlikely to estimate the test error with a low variance (unbiased by highly variable).

Approximation inference. Compare PBP, BBB, LA, to HMC and SGD. Use fixed prior. Test effect of small training set. Compare time to reach test accuracy. Use importance sampling to calculate WAIC. Decrease memory until these approximation methods perform better than MCMC.

Scaling MCMC SGLD, SGHMC, pSGLD, SGFS, Thermostat. Compare test accuracy with batch HMC. Tuning by test accuracy, or tuning by ESJD.

Consensus MCMC See if singularity severely degrades quality of classification.

Effects of pruning In the BBB paper, performance does not degrade drastically even if 98 percent of the weights are removed. See same thing applies to MCMC samples.

But before all this can start we first need to overcome the difficulties of sampling from the posterior distribution of a bayesian neural network. The first source of difficulty comes from the correlation in the posterior distribution induced by the connection between neurons in consecutive layers. Each neuron in a layer is connected to all neurons from the previous layer and hence is influenced by values in the previous layer. This is the first source of correlation. The second source of correlation comes from the use of a hierarchical prior on the weights of the network. This is tackled by Riemmanina Manifold type MCMC methods, which make use of information of the second derivatives of the log-posterior. The trade-off is now a linear system of full rank must be solved to calculate each update in the markov chain. This has the time complexity of $O(p^3)$ which makes it difficult to apply to neural network models, who's expressiveness comes from its large number of parameters.

Posterior correlation due to hierarchical prior vs inter-layer correlation.

The second source of difficulty comes from the cost of evaluating the log-posterior density and its derivative with respect to the parameters. For exact calculation, both requires going through the entire dataset. In frequentist fitting of neural networks, this problem is bypassed by evaluating the gradient using a random subset of the original datapoints as input. The mcmc analogues are stochastic gradient mcmc methods. These methods, however, suffer from slow mixing in the case of SGLD variants, and the need to do matrix inversion in the case of SGHMC variants. Interesting compromise can be made by replacing the full fisher information matrix by a diagonal matrix or low-rank approximations.

And other obstacle prevents the widespread adoption of mcmc for inference in BNN is the dependance on GPUs for calculating the gradient of the weights by backpropogation. GPUs make fast calculation of gradient of networks with increasing number of layers feasible, however its downside is its limited fast-access memory. Transferring weights from the GPU to the main disk creates a bottleneck in the calculation that balances out the speed gain. Even if we disregard

the problem with speed and only focuses on the memory, we find that bayesian inference by mcmc as carried out traditionally by the statistics community, where a large number of samples from the chain is generated and saved, diagnosced for convergence and then quantities where expectation is taken with respect to the posterior distribution are approximated by taking expectation with respect to the empirical distribution instead. The first problem with this approach is the lack of tool for assessing convergence for high-dimensional distributions. Convergence of every marginal distribution does not imply convergence of the joint distribution. Also, it is impossible to visually inspect the trace plot of all covariates along the chain in order to assess convergence, depriving ourselves one of the more reliable tools in the classical mcmc methodology. Another problem with the traditional methodology is the necessity to compute a long chain (even after thinning) and use the samples for expectaion calculation. No matter how decorrelated the samples from the chain are, more than a handful of samples at a time is required to calculate an unbiased estimate of the expectation that doesn't suffer from high variance. This limits the size of the network that can be trained, often it precludes the use of a GPU altogether and limit us to shallow networks. A frequentist training of neural network only $O(p)$ memory is required to store the weights in memory, but in the bayesian framework described above $O(pT)$ memory is required, where T is the number of samples from the chain that we wish to retain. In many applications a large model of 1 million paramters is found to minimize the test error on a holdout set among many possible model structure. One might then wish to assign prior to weights in this network and carry out bayesian inference. Even assuming there is enough memory on the GPU, it is reasonable to assume that in most applications it might not be worth the 50 times increase in memory budget to improve predictive accuracy by 2 percent.

Since neural networks are mostly used for predictive purposes and there is little interpretability in the posterior distribution of the individual weights, which are always integrated out to find the predictive distribution anyway, one might find it worthwhile to look at approximate inference methods that aim to find an approximation to the posterior distribution with a tractable distribution and which allows for easy calculation of the posterior predictive distribution.

One note that to calculate the WAIC, the memory requirement is only $O(nT)$, where n is the number of observations and T the number of mcmc samples. This inspires the methdology of bayesian model selection and inference. First we choose the model by using the most efficient mcmc methods. Since evaluation of the WAIC is independent of the dimension of the model, this can be done on the GPU and enjoy the speed up that it provides. Once the model is chosen, approximation inference is carried out to fit an approximate model that allows fast training and prediction, with most variations using $O(p)$ memory.

While batch version of HMC is not practical because of size constraints, for research purposes or small data problem it might be necessary to simulate a markov chain that is as targeting as close the posterior distribution as possible, excluding stochastic gradient versions where bias is introduced by dropping the metropolis correction step. A comparative study of the effectivenss the new RMHMC type samplers applied to jointly sample the weights and hyperparameters is long due.

Make the connection between natural gradient learning and riemannian manifold hmc

What is the largest neural networks that can be trained by modern hardware using 1996 techniques (HMC tuned by

heuristics plus windowed state) from Neal's thesis?

How good are the samples generated using these techniques? Look at min effective sample size across all covariates. Trace plot of randomly selected components/ hyperparameters. Should see skewed distribution. Plot it.

Train models of moderate size first (can fit on harddisk, most likely 1 hidden layer), then apply Riemman Hamiltonian Monte Carlo to see if you get much better results. Also compare that to just normal approximation around max. Multiple mode seems to be a problem. But also have result(heuristics) from frequentist results saying that local min is not a serious problem cuz most local modes are similar cost value.

Sample using MALA, then Riemann MALA.

The langevin dynamics algorithm has fewer tuning parameters, but mixes slower. Hope is that Riemann MALA is a compromise between MALA and full HMC.

Compare different metrics used in RHMC.

How much does initialization matters. i.e. Initialize to random normal covariates vs variance-matching initialization. See Xavier initialization:

$$X \sim U[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}]$$

Compare predictive accuracy reached under fixed time budget.

Simulation studies:

Target distribution: multivariate student-t distribution. Zero mean and diagonal covariance with common variance. See Jake Baker's master's thesis.His simulations don't stress the samplers as much as I'd wish. For example the highest dimension tested was 20.

He used the KL divergence as performance metric, calculated as follows [14].

Since we care about prediction, we can follow Mykshov by comparing different MCMC samplers/approximate bayesian inference using the KL divergence of the predictive distribution with the x integrated out instead. First, this is usually a one dimensional distribution, which makes much more sense when we are using kernel methods to approximate the KL divergence anyway.

Funnel distribution: typical of bayesian models with hierarchical priors. Neal has used slice sampling and HMC to draw from it. Betancourt used Riemmanian HMC with his metric.

Hasn't been done: MALA, RMALA, HMC with partial momentum, windowed, semi-separable HMC (benchmark),RHMC with softabs(hasn't been a comparison of ESS) adaptive HMC (Gaussian process) each of the HMC variety above can also be

coupled with the trick in [17] (fix conditioning matrix between each sample). Experiment with efficient diagonal approximation to covariance matrix.

Hierarchical bayesian logistic regression: Methods above as well as stochastic gradient MCMC (sgld,pre-conditioned sgld, thermostat,sghmc) Try on simulated data as well as benchmark datasets.

Bayesian dark knowledge: how does the quality of the MCMC samples from the posterior distribution of the teacher model affect the approximation of the prediction distribution by the student neural network. How sensitive is it?

Memory efficient implementation of MCMC. Keep history of random seeds for mv normals, as well as history of acceptance decision. Given only θ_T, can reproduce chain $\{\theta_1, \ldots, \theta_T\}$ on command. Useful for GPU implementations of MCMC. Predictive performance(cross-validation) used for adaptive tuning[98]

Using predictive performance as metric to tune MCMC. i.e. select tuning parameters that give the highest average posterior predictive density for new data or highest accuracy on test set.

Compare variational posterior to true posterior.

Statistical question: compare the train/test holdout metric to WAIC. Do they give the same conclusions?[50] Try k-fold cross-validation as well.

Evaluate variational models in terms of WAIC.

Work using evidence aka marginal likelihood for neural network model comparison.

2.1 neural networks

Previous work:[13]

Suppose the dimension p of θ is so high that we can only afford to keep 20 copies or so of θ in memory (on a GPU, for example). What is the best practice (MCMC vs variational inference) for bayesian inference in this scenario.

Effect of number of layers on autocorrelation.(with or without hierarchical prior).

Model selection as follows: select certain quantities to monitor convergence (hyperparameters, acceptance ratio etc), store values needed to calculate WAIC, select model. Then use approximate inference (variational inference or expectation propogation) to compute approximation to predictive distribution for predictions.

Sgmcmc: how many minibatches of data should be used to update the main paramters before switching to sampling hyperparameters? In sghmc, they have examples where the entire dataset is passed through and the other just some data. Goal:

stochastic gradient quasi netwon langevin dynamics requires extra tuning paramters with little guidance on how to tune them [90]

preconditioned sgld introduces bias[56].

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Sungjin Ahn, Anoop Korattikara, and Max Welling. Bayesian posterior sampling via stochastic gradient fisher scoring. *arXiv preprint arXiv:1206.6380*, 2012.

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*, 2015.

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python.

[5] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.

[6] Alexandros Beskos, Natesh Pillai, Gareth Roberts, Jesus-Maria Sanz-Serna, Andrew Stuart, et al. Optimal tuning of the hybrid monte carlo algorithm. *Bernoulli*, 19(5A):1501–1534, 2013.

[7] Michael Betancourt. A general metric for riemannian manifold hamiltonian monte carlo. In *Geometric science of information*, pages 327–334. Springer, 2013.

[8] Michael Betancourt. Identifying the optimal integration time in hamiltonian monte carlo. *arXiv preprint arXiv:1601.00225*, 2016.

[9] MJ Betancourt. Generalizing the no-u-turn sampler to riemannian manifolds. *arXiv preprint arXiv:1304.1920*, 2013.

[10] MJ Betancourt, Simon Byrne, Samuel Livingstone, and Mark Girolami. The geometric foundations of hamiltonian monte carlo. *arXiv preprint arXiv:1410.5110*, 2014.

[11] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[12] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

[13] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[14] Sylvain Boltz, Eric Debreuve, and Michel Barlaud. High-dimensional statistical distance for region-of-interest tracking: Application to combining a soft geometric constraint with radiometry. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[15] Sylvain Boltz, Eric Debreuve, and Michel Barlaud. knn-based high-dimensional kullback-leibler distance for tracking. In *Image Analysis for Multimedia Interactive Services, 2007. WIAMIS'07. Eighth International Workshop on*, pages 16–16. IEEE, 2007.

[16] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[17] Martin Burda, John Maheu, et al. Bayesian adaptive hamiltonian monte carlo with an application to high-dimensional bekk garch models. Technical report, 2011.

[18] Bob Carpenter. Stan: A probabilistic programming language.

[19] Changyou Chen, Nan Ding, and Lawrence Carin. On the convergence of stochastic gradient mcmc algorithms with high-order integrators. In *Advances in Neural Information Processing Systems*, pages 2278–2286, 2015.

[20] Tianqi Chen, Emily B Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *ICML*, pages 1683–1691, 2014.

[21] Kiam Choo. *Learning hyperparameters for neural network models using Hamiltonian dynamics.* PhD thesis, Citeseer, 2000.

[22] Andreas C Damianou and Neil D Lawrence. Deep gaussian processes.

[23] Guillaume Dehaene and Simon Barthelmé. Expectation propagation in the large-data limit. *arXiv preprint arXiv:1503.08060*, 2015.

[24] Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D Skeel, and Hartmut Neven. Bayesian sampling using stochastic gradient thermostats. In *Advances in neural information processing systems*, pages 3203–3211, 2014.

[25] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[26] Maurizio Filippone and Raphael Engler. Enabling scalable stochastic gradient-based inference for gaussian processes by employing the unbiased linear system solver (ulisse).

[27] Charles W Fox and Stephen J Roberts. A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95, 2012.

[28] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[29] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning.

[30] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

[31] Zhe Gan, Changyou Chen, Ricardo Henao, and David Carlson. Scalable deep poisson factor analysis for topic modeling.

[32] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*, volume 2. Chapman & Hall/CRC Boca Raton, FL, USA, 2014.

[33] Andrew Gelman, G Roberts, and W Gilks. Efficient metropolis jumping hules. 1996.

[34] Andrew Gelman, Aki Vehtari, Pasi Jylänki, Christian Robert, Nicolas Chopin, and John P Cunningham. Expectation propagation as a way of life. *arXiv preprint arXiv:1412.4869*, 2014.

[35] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[36] Charles J Geyer. Practical markov chain monte carlo. *Statistical Science*, pages 473–483, 1992.

[37] Soumya Ghosh, Francesco Maria Delle Fave, and Jonathan S Yedidia. Assumed density filtering methods for learning bayesian neural networks. In *AAAI*, pages 1589–1595, 2016.

[38] Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[39] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[40] Peter J Green, Krzysztof Łatuszyński, Marcelo Pereyra, and Christian P Robert. Bayesian computation: a summary of the current state, and samples backwards and forwards. *Statistics and Computing*, 25(4):835–862, 2015.

[41] Isabelle Guyon, Steve R Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *NIPS*, volume 4, pages 545–552, 2004.

[42] José Miguel Hernández-Lobato and Ryan P Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. *arXiv preprint arXiv:1502.05336*, 2015.

[43] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

[44] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

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

[46] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[47] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[48] Pasi Jylänki, Aapo Nummenmaa, and Aki Vehtari. Expectation propagation for neural networks with sparsity-promoting priors. *Journal of Machine Learning Research*, 15(1):1849–1901, 2014.

[49] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[50] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[52] Jouko Lampinen and Aki Vehtari. Bayesian approach for neural networks—review and case studies. *Neural networks*, 14(3):257–274, 2001.

[53] Lucien Le Cam. *Asymptotic methods in statistical decision theory*. Springer Science & Business Media, 2012.

[54] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[55] Benedict Leimkuhler and Sebastian Reich. *Simulating hamiltonian dynamics*, volume 14. Cambridge University Press, 2004.

[56] Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. *arXiv preprint arXiv:1512.07666*, 2015.

[57] Chunyuan Li, Changyou Chen, Kai Fan, and Lawrence Carin. High-order stochastic gradient thermostats for bayesian learning of deep models. *arXiv preprint arXiv:1512.07662*, 2015.

[58] Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Stochastic expectation propagation. In *Advances in Neural Information Processing Systems*, pages 2323–2331, 2015.

[59] Faming Liang, Chuanhai Liu, and Raymond Carroll. *Advanced Markov chain Monte Carlo methods: learning from past samples*, volume 714. John Wiley & Sons, 2011.

[60] Jun S Liu. *Monte Carlo strategies in scientific computing*. Springer Science & Business Media, 2008.

[61] Samuel Livingstone, Michael Betancourt, Simon Byrne, and Mark Girolami. On the geometric ergodicity of hamiltonian monte carlo. *arXiv preprint arXiv:1601.08057*, 2016.

[62] Yi-An Ma, Tianqi Chen, and Emily Fox. A complete recipe for stochastic gradient mcmc. In *Advances in Neural Information Processing Systems*, pages 2917–2925, 2015.

[63] David JC MacKay. The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736, 1992.

[64] Nimalan Mahendran, Ziyu Wang, Firas Hamze, and Nando De Freitas. Adaptive mcmc with bayesian optimization.

[65] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[66] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[67] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[68] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[69] Radford M Neal. Bayesian learning via stochastic dynamics.

[70] Radford M Neal. An improved acceptance procedure for the hybrid monte carlo algorithm. *arXiv preprint hep-lat/9208011*, 1992.

[71] Radford M Neal. Sampling from multimodal distributions using tempered transitions. *Statistics and computing*, 6(4):353–366, 1996.

[72] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[73] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

[74] Willie Neiswanger, Chong Wang, and Eric Xing. Asymptotically exact, embarrassingly parallel mcmc. *arXiv preprint arXiv:1311.4780*, 2013.

[75] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[76] Yann Ollivier. Riemannian metrics for neural networks i: feedforward networks. *arXiv preprint arXiv:1303.0818*, 2013.

[77] Manfred Opper and Cédric Archambeau. The variational gaussian approximation revisited. *Neural computation*, 21(3):786–792, 2009.

[78] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[79] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[80] Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[81] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[82] Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[83] Gareth O Roberts, Andrew Gelman, Walter R Gilks, et al. Weak convergence and optimal scaling of random walk metropolis algorithms. *The annals of applied probability*, 7(1):110–120, 1997.

[84] Gareth O Roberts and Jeffrey S Rosenthal. Optimal scaling of discrete approximations to langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268, 1998.

[85] Gareth O Roberts and Jeffrey S Rosenthal. Examples of adaptive mcmc. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.

[86] Gareth O Roberts, Jeffrey S Rosenthal, et al. Optimal scaling for various metropolis-hastings algorithms. *Statistical science*, 16(4):351–367, 2001.

[87] Gareth O Roberts, Jeffrey S Rosenthal, et al. General state space markov chains and mcmc algorithms. *Probability Surveys*, 1:20–71, 2004.

[88] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[89] Steven L Scott, Alexander W Blocker, Fernando V Bonassi, Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayes and big data: The consensus monte carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88, 2016.

[90] Umut Şimşekli, Roland Badeau, A Taylan Cemgil, and Gaël Richard. Stochastic quasi-newton langevin monte carlo. *arXiv preprint arXiv:1602.03442*, 2016.

[91] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[92] Yee Whye Teh, Leonard Hasenclever, Thibaut Lienart, Sebastian Vollmer, Stefan Webb, Balaji Lakshminarayanan, and Charles Blundell. Distributed bayesian learning with stochastic natural-gradient expectation propagation and the posterior server. *arXiv preprint arXiv:1512.09327*, 2015.

[93] Yee Whye Teh, Alexandre H Thiery, and Sebastian J Vollmer. Consistency and fluctuations for stochastic gradient langevin dynamics.

[94] Luke Tierney. Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.

[95] DM Titterington. Bayesian methods for neural networks and related models. *Statistical science*, pages 128–139, 2004.

[96] Francesco Vivarelli and Christopher KI Williams. Comparing bayesian neural network algorithms for classifying segmented outdoor images. *Neural Networks*, 14(4):427–437, 2001.

[97] Jian-Sheng Wang and Robert H Swendsen. Cluster monte carlo algorithms. *Physica A: Statistical Mechanics and its Applications*, 167(3):565–579, 1990.

[98] Ziyu Wang, Shakir Mohamed, and Nando De Freitas. Adaptive hamiltonian and riemann manifold monte carlo samplers.

[99] Sumio Watanabe. *Algebraic geometry and statistical learning theory*, volume 25. Cambridge University Press, 2009.

[100] Sumio Watanabe. Asymptotic equivalence of bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, 11(Dec):3571–3594, 2010.

[101] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[102] Stephen Wright. Numerical optimization.

[103] Koshi Yamada and Sumio Watanabe. Information criterion for variational bayes learning in regular and singular cases. In *Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS), 2012 Joint 6th International Conference on*, pages 1551–1555. IEEE, 2012.

[104] Yichuan Zhang and Charles Sutton. Semi-separable hamiltonian monte carlo for inference in bayesian hierarchical models. In *Advances in neural information processing systems*, pages 10–18, 2014.