Bayesian Deep Learning – A Stochastic Dynamics Perspective

Niklas Bühler
Technical University of Munich
School of Computation, Information and Technology
niklasbuehler@mailbox.org

Abstract

Bayesian statistics provides an alternative approach for training and evaluating neural networks, which allows for the incorporation of uncertainty as well as prior knowledge.

Stochastic dynamics is a branch of mathematics that deals with the study of systems that evolve over time and are subject to random fluctuations. As such, it provides a framework for modeling stochastic training processes of Bayesian neural networks.

This report gives an overview of Bayesian deep learning from a stochastic dynamics perspective by first introducing Bayesian deep learning as well as two important methods for training Bayesian neural networks, and then building upon this fundament by presenting various approaches and variations inspired by stochastic dynamics.

1. Introduction

Bayesian deep learning differs from conventional frequentist deep learning by incorporating uncertainty into the model and its learning process. Training a Bayesian neural network is equivalent to performing Bayesian inference of a network's parameters, which involves finding the posterior probability distribution of the parameters given the training data. However, an exact evaluation of this probability distribution is typically intractable.

There are two approaches to solving this issue. One is to perform variational inference instead of exact inference. This method is part of the *Bayes by Backprop* algorithm. Another approach is to use *Markov Chain Monte Carlo (MCMC)* methods to sample from the posterior distribution. One specific MCMC method is the *Hamiltonian Monte Carlo (HMC)* algorithm, which is based on *Hamiltonian dynamics*. However, the exact application of the HMC algorithm requires gradient computations over the whole dataset and is thus oftentimes computationally infeasible in practice.

Stochastic dynamics provides a way of describing stochastic processes and thus can serve as a framework for formulating and analyzing stochastic versions of algorithms. For example, conventional learning algorithms can be transformed into Bayesian learning algorithms by incorporating stochasticity in different ways. Another application of stochastic dynamics in this report is the formulation of stochastic variants of the presented HMC algorithm, allowing its application to large datasets and thus enabling its use in practice.

The general concepts of Bayesian deep learning and Bayesian neural networks, as well as the *Bayes by Backprop* algorithm are introduced in Sec. 2. In Sec. 3, Markov Chain Monte Carlo methods as well as Hamiltonian dynamics are introduced. These two concepts are then combined in the Hamiltonian Monte Carlo algorithm. Stochastic dynamics is introduced in Sec. 4 and serves as a framework for defining multiple stochastic variations of the previously presented algorithms. Finally, in Sec. 5, this report is summarized and the main insights are briefly put into context.

2. Bayesian Deep Learning

The term *Bayesian deep learning* describes the application of methods from Bayesian statistics to deep learning. This approach allows for the integration of uncertainty into neural networks, which not only makes it possible to assess the degree of certainty of predictions, but also leads to richer representations through cheap model averaging and regularization. Furthermore, by incorporating a prior probability distribution, the Bayesian approach allows for the injection of expert knowledge into the learning process.

In Sec. 2.1, Bayesian deep learning is introduced and contrasted against conventional frequentist deep learning. In Sec. 2.2, the *Bayes by Backprop* algorithm is introduced, followed by a brief assessment of its performance.

The content of this section is based on [7] and [2].

2.1. Frequentist and Bayesian Deep Learning

Conventional frequentist learning of a neural network fits the networks parameters to a single "optimal" set of values w^* that corresponds to the maximum likelihood

estimate $\arg\max_w P(\mathcal{D} \mid w)$ given the training data $\mathcal{D}$. A conventional neural network f then uses these fixed parameters w^* to produce a prediction $\hat{y}$ about a previously unseen data point x, *i.e.* $\hat{y} = f(x, w^*)$.

In contrast, Bayesian deep learning uses the training data $\mathcal{D}$ to find the posterior probability distribution $P(w \mid \mathcal{D})$ over all possible weight vectors. The weights in a Bayesian neural network are thus represented by probability distributions over all possible values, rather than having a fixed value as in conventional neural networks (see Fig. 1). This posterior probability distribution can be obtained using Bayes' theorem

$$P(w \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid w)P(w)}{P(\mathcal{D})}, \tag{1}$$

where $P(\mathcal{D} \mid w)$ describes the likelihood of the data given the weights as used in frequentist learning, $P(w)$ denotes the prior distribution of the network weights and $P(\mathcal{D})$ is the probability of the data. A Bayesian neural network then forms predictions by taking expectations, *i.e.* the outputs obtained from all models with all possible weight vectors are averaged, each contributing in proportion to its posterior probability. The whole prediction process consists of the following steps:

1. Define the prior distribution for the weight vectors, *e.g.* give each weight a Gaussian prior with zero mean and standard deviation σ: $P(w) \propto \exp(-|w|^2/2\sigma^2)$.

2. Formulate the posterior distribution over the weight vectors given the training data $\mathcal{D}$ using Bayes' Theorem: $P(w \mid \mathcal{D}) \propto P(w)P(\mathcal{D} \mid w)$.

3. Given the input vector x, the model predicts

$$\begin{aligned} \hat{y} &= \mathbb{E}_{w \sim P(w|\mathcal{D})}[f(x, w)] \\ &= \int f(x, w)P(w \mid \mathcal{D})dw. \end{aligned} \tag{2}$$

However, the integral in Eq. (2) is typically intractable, as the dimensionality of w is very large and the neural network f isn't suitable for exact integration. Furthermore, an exact evaluation of the posterior distribution $P(w \mid \mathcal{D})$ is typically intractable as well.

One possible solution to this problem is to perform variational inference of $P(w \mid \mathcal{D})$, as explained in Sec. 2.2.

Another solution is to sample from the posterior using Markov Chain Monte Carlo (MCMC) methods and then evaluate the expectation of Eq. (2) using Monte Carlo approximations. MCMC methods are discussed in Sec. 3.1.

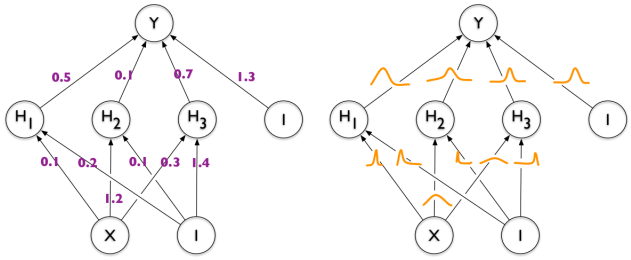

Figure 1. **Comparison between Frequentist and Bayesian Neural Networks:** Left: In a conventional frequentist neural network, each weight has a fixed value. Right: In a Bayesian neural network, each weight is modeled by a probability distribution. Figure taken from [2].

2.2. Bayes by Backprop

Bayes By Backprop is an approximate learning algorithm similar to backpropagation, proposed in [2]. It performs variational inference of the posterior distribution of a Bayesian neural network, *i.e.* a variational approximation is optimized instead of the full posterior distribution.

Note that although this method trains an infinite ensemble of neural networks using unbiased Monte Carlo estimates of the gradients, it typically only doubles the number of parameters, as is shown below.

Variational Bayesian Learning Variational learning can be used to find the parameters θ^* of a variational distribution of the weights $q(w \mid \theta)$ that minimize the Kullback-Leibler (KL) divergence from the true Bayesian posterior of the weights:

$$\begin{aligned} \theta^* &= \arg\min_\theta \mathrm{KL}[q(w \mid \theta) \mid\mid P(w \mid \mathcal{D})] \\ &= \arg\min_\theta \int q(w \mid \theta) \log \frac{q(w \mid \theta)}{P(w)P(\mathcal{D} \mid w)} dw \\ &= \arg\min_\theta \mathrm{KL}[q(w \mid \theta) \mid\mid P(w)] \\ &\qquad - \mathbb{E}_{q(w|\theta)}[\log P(\mathcal{D} \mid w)]. \end{aligned} \tag{3}$$

The resulting cost function is called *variational free energy* or *expected lower bound (ELBO)*. It is denoted as

$$\begin{aligned} \mathcal{F}(\mathcal{D}, \theta) = &\mathrm{KL}[q(w \mid \theta) \mid\mid P(w)] \\ &- \mathbb{E}_{q(w|\theta)}[\log P(\mathcal{D} \mid w)]. \end{aligned} \tag{4}$$

This term is the sum of a data-dependent part (*likelihood cost*) and a prior-dependent part (*complexity cost*). Thus, it embodies a trade-off between satisfying the complexity of the data $\mathcal{D}$ and the simplicity prior $P(w)$.

Unbiased Monte Carlo Gradients To evaluate $\mathcal{F}(\mathcal{D}, \theta)$, one can apply the Monte Carlo method together with a generalization of the Gaussian reparameterization trick.

First, approximate the exact cost $F(\mathcal{D}, \theta)$ by

$$\begin{aligned} &\int q(w \mid \theta) \log \frac{q(w \mid \theta)}{P(w)} dw - \mathbb{E}_{q(w|\theta)}[\log P(\mathcal{D} \mid w)] \\ &\approx \sum_{i=1}^{n} \log q(w^{(i)} \mid \theta) - \log P(w^{(i)}) - \log P(\mathcal{D} \mid w^{(i)}), \end{aligned} \tag{5}$$

with $w^{(i)}$ denoting the i-th Monte Carlo sample drawn from the variational posterior $q(w^{(i)} \mid \theta)$.

Next, let $\epsilon \sim q(\epsilon)$ and $w = t(\theta, \epsilon)$, where t is a deterministic function. Suppose $q(w \mid \theta)$ is such that $q(\epsilon)d\epsilon = q(w \mid \theta)dw$. For $f(w, \theta) = \log q(w \mid \theta) - \log P(w)P(\mathcal{D} \mid w)$, the derivative of its expectation can be expressed as the expectation of a derivative:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{q(w|\theta)}[f(w, \theta)] = \mathbb{E}_{q(\epsilon)}[\frac{\partial f(w, \theta)}{\partial w} \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta}], \tag{6}$$

as is shown in [2]. This result will be used to calculate the gradients in the next step.

Gaussian Variational Posterior The deterministic function $t(\theta, \epsilon)$ transforms a sample of parameter-free noise ϵ and the variational posterior parameters θ into a sample from the variational posterior.

For the Gaussian case, the posterior sample of the weights is

$$\begin{aligned} w &= t(\theta, \epsilon) \\ &= \mu + \sigma \circ \epsilon \\ &= \mu + \log(1 + \exp(\rho)) \circ \epsilon, \end{aligned} \tag{7}$$

where $\circ$ denotes pointwise multiplication and $\theta = (\mu, \rho)$ are the variational posterior parameters. The standard deviation is parameterized as $\sigma = \log(1 + \exp(\rho))$. In this case, each weight has two degrees of freedom (in μ and ρ), doubling the number of parameters in the network.

The whole optimization process in the Gaussian case can thus be summed up as

1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let $w = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3. Let $\theta = (\mu, \rho)$.
4. Let $f(w, \theta) = \log q(w \mid \theta) - \log P(w)P(\mathcal{D} \mid w)$.
5. Calculate the gradient w.r.t. the mean
$$\Delta_\mu = \frac{\partial f(w, \theta)}{\partial w} + \frac{\partial f(w, \theta)}{\partial \mu}. \tag{8}$$
6. Calculate the gradient w.r.t. the standard deviation parameter ρ
$$\Delta_\rho = \frac{\partial f(w, \theta)}{\partial w} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(w, \theta)}{\partial \rho}. \tag{9}$$
7. Update the variational parameters
$$\begin{aligned} \mu &\leftarrow \mu - \alpha \Delta_\mu \\ \rho &\leftarrow \rho - \alpha \Delta_\rho. \end{aligned} \tag{10}$$

Note that the term $\frac{\partial f(w, \theta)}{\partial w}$ which is shared by both gradients describes exactly the gradient found by the usual backpropagation algorithm. Thus, to learn both mean and standard deviation, one simply has to calculate the usual gradients and then scale and shift them as described above.

Scale Mixture Prior In the original paper [2], a scale mixture of two Gaussian densities is used as the prior. All prior parameters are shared by all the weights.

The priors don't necessarily have to be Gaussians. Nevertheless, Gaussians lend themselves easily to analytical handling and are therefore a practical choice.

2.2.1 Performance of Bayesian Neural Networks

Networks trained with the *Bayes by Backprop* algorithm achieve good results in several domains, as has been shown in [2] for a simple classification task, but also when employed in reinforcement learning tasks via Thompson sampling.

Classification on MNIST For a classification task on the MNIST dataset, it is shown in [2], how a Bayesian neural network trained using *Bayes by Backprop* achieved an improved performance compared to simple feedforward neural networks (test error of 1.32% instead of 1.6%). The achieved performance is similar to that achieved by using dropout (test error of 1.3%).

An interesting result stems from eliminating some of the weights during runtime. First, the weights were ordered by their signal-to-noise ratio ($|\mu_i|/\sigma_i$), then those with the lowest ratio were removed by replacing their variational posterior with a constant zero.

Even when 95% of weights are removed, the network still performs well, as can be seen in Tab. 1. Hence, even though introducing additional degrees of freedom for each weight multiplies the amount of parameters in the network, only a few of these parameters need to be stored at runtime.

3. Hamiltonian Monte Carlo

The posterior distributions utilized in Bayesian deep learning are oftentimes extremely difficult to sample from

Proportion removed	# Weights	Test Error
0%	2.4m	1.24%
50%	1.2m	1.24%
75%	600k	1.24%
95%	120k	1.29%
98%	48k	1.39%

Table 1. **Effects of Weight Pruning:** When eliminating a large percentage of weights with low signal-to-noise ratio from the MNIST classifier during runtime, test performance stays high. Results taken from [2].

and thus don't allow for direct sampling. Markov Chain Monte Carlo (MCMC) methods comprise an approach to sample indirectly from such distributions.

The Metropolis-Hastings (MH) algorithm is a typical MCMC method for obtaining a sequence of random samples from a target distribution.

The Hamiltonian Monte Carlo (HMC) algorithm is another MCMC method. It simulates Hamiltonian dynamics to define distant state proposals with high acceptance probabilities in a Metropolis-Hastings framework.

In this section, the general framework of Markov Chain Monte Carlo methods is briefly explained in Sec. 3.1. This includes their general workings, as well as the Metropolis-Hastings algorithm. Next, Hamiltonian dynamics and a way to discretize its simulation are introduced in Sec. 3.2. Finally, in Sec. 3.3, the two concepts are brought together to define the Hamiltonian Monte Carlo algorithm.

The content of this section is based on [8] and [7].

3.1. Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC) methods comprise a useful approach to sample from probability distributions that are otherwise hard to sample from. They operate by constructing a Markov chain that has the desired distribution as its equilibrium distribution.

Thus, they offer an alternative way to variational inference (see Sec. 2.2) for working with complicated posterior distributions.

Metropolis-Hastings Algorithm The *Metropolis-Hastings (MH) algorithm* is a typical MCMC method. It can draw samples from any probability distribution for which the probability density can be evaluated up to a normalizing constant. This is particularly useful in practice, as calculating the normalization factor is often difficult.

The MH algorithm works by generating a sequence of sample values which approximate the desired distribution more closely as more sample values are produced. The algorithm iteratively produces a candidate for the next sample value, based only on the current sample value, hence making the sequence of samples a Markov chain. This candidate value is then either accepted or rejected with an acceptance probability that is based on the ratio of the probability densities for the current and candidate sample values.

The algorithm works as follows: Let $f(x) \propto P(x)$ be a function that is proportional to the target distribution $P(x)$ and choose a *proposal density* $g(x \mid y)$. This proposal density suggests a candidate for the next sample value x, given the current sample value y. A typical choice is $g(x \mid y) = \mathcal{N}(y, \sigma)$, generating a random walk of samples in which points closer to y are more likely to be visited next.

First, choose an arbitrary point x_0 as the initial sample value. Then, for each iteration t, a candidate value x' is generated by sampling from the distribution $g(x' \mid x_t)$. The acceptance ratio of this candidate is given as

$$\alpha = \frac{f(x')}{f(x_t)} = \frac{P(x')}{P(x_t)}, \tag{11}$$

because $f(x) \propto P(x)$.

The acceptance decision can be handled by generating a uniform random number $u \in [0, 1]$. If $u \leq \alpha$, the candidate x' is accepted and $x_{t+1} = x'$. Otherwise, it is rejected and $x_{t+1} = x_t$.

3.2. Hamiltonian Dynamics

The random walk approach used in the Metropolis-Hastings algorithm is inefficient, as it blindly proposes candidate samples from the vicinity of the current sample.

The essential property of Hamiltonian dynamics is that the probability to arrive at a position during its simulation corresponds to the exponential of the potential energy at this position. If the potential energy is thus chosen to be the logarithm of some target distribution, a simulation of Hamiltonian dynamics will yield perfect samples from this target distribution. The Hamiltonian Monte Carlo algorithm simulates Hamiltonian dynamics in order to propose better-informed candidates, which leads to a more efficient exploration of the sample space.

In order to define the HMC algorithm afterwards, this section introduces *Hamiltonian dynamics*, as well as a way to discretize its simulation.

A Two-dimensional Analogy Hamiltonian dynamics can be visualized in two dimensions as a frictionless puck that slides over a surface of varying height, as was proposed in [8]. The state of this system is described by

1. the position $q \in \mathbb{R}^2$ of the puck,
2. the momentum $q \in \mathbb{R}^2$ of the puck (its mass times velocity).

The *potential energy* $U(q)$ of the puck is proportional to the height of the surface at position q. Its *kinetic energy* $K(p)$ is equal to $|p|^2/(2m)$, with m being its mass.

The movement of the puck across the surface depends on the slope at its current position as well as its momentum. On a level part of the surface, the puck moves at a constant velocity, equal to p/m. On a rising slope, its momentum allows it to continue, while its kinetic energy decreases and its potential energy increases. Once the kinetic energy (and thus also p) is zero, the puck will slide back down, this time with its kinetic energy increasing again, while its potential energy decreases.

When simulating Hamiltonian dynamics for obtaining a sequence of random samples, the position variables correspond to the variables of interest and the potential energy is the negative log likelihood of these variables. The momentum variables on the other hand, one for each position variable, will be introduced artificially.

3.2.1 Hamilton's Equations

Hamiltonian dynamics describes a system with a state consisting of a d-dimensional position vector q and a d-dimensional momentum vector p, so that the full state space consists of $2d$ dimensions. The *Hamiltonian* $H(q, p)$ is a function of q and p that descibes this system.

Equations of Motion The change of q and p over time t is determined by partial derivatives of the Hamiltonian which are given according to Hamilton's equations:

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} \tag{12}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} \tag{13}$$

These equations define a mapping T_s for any time duration s, which maps the state at time t to the state at time $t + s$.

Potential and Kinetic Energy The Hamiltonian can usually be written as

$$H(q, p) = U(q) + K(p). \tag{14}$$

In this form, $U(q)$ is called the *potential energy*. It is typically defined as the negative log probability density of the target distribution for q plus any constant that is convenient.

The term $K(p)$ is called the *kinetic energy*. It is practical to use a quadratic function for the kinetic energy $K(p) = p^T M^{-1} p/2$, because then p will have a zero-mean multivariate Gaussian distribution. The components of p are often chosen to be independent with variances m_i, simplifying the kinetic energy to $K(p) = \sum_{i=1}^{d} \frac{p_i^2}{2m_i}$. Setting the variances to $m_i = 1$ further simplifies the formula to

$$K(p) = \sum_{i=1}^{d} \frac{p_i^2}{2} = \frac{1}{2}|p|^2. \tag{15}$$

The term $K(p)$ thus corresponds to the negative log probability density (plus a constant) of the zero-mean Gaussian distribution with identity covariance matrix $M = I$.

Using these simplifications, Hamilton's equations can be written as

$$\frac{dq}{dt} = p, \tag{16}$$

$$\frac{dp}{dt} = -\nabla U(q). \tag{17}$$

3.2.2 Discretization of Hamiltonian Dynamics

When simulated in a computer, Hamilton's equations must be approximated by discretizing time, using some small step size ϵ. The simulation starts with the state at time 0 and iteratively computes the approximated state at times $\epsilon, 2\epsilon, \ldots$ In the following, it is assumed that the Hamiltonian has the form $H(q, p) = U(q) + K(p)$ with kinetic energy $K(p) = \frac{1}{2}|p|^2$, leading to the simplified forms of Eqs. (16) and (17).

The Leapfrog Method The leapfrog method is a fairly simple method for simulating Hamiltonian dynamics in a discrete way. Given a step size ϵ, an iteration of the leapfrog method is given by the update equations

$$p(t + \epsilon/2) = p(t) - \frac{\epsilon}{2}\nabla U(q(t)) \tag{18}$$

$$q(t + \epsilon) = q(t) + \epsilon p(t + \epsilon/2) \tag{19}$$

$$p(t + \epsilon) = p(t + \epsilon/2) - \frac{\epsilon}{2}\nabla U(q(t + \epsilon)). \tag{20}$$

Each iteration starts with a half step for the momentum variables (Eq. (18)), then does a full step for the position variables, using the new values of the momentum variables (Eq. (19)). Finally, another half step for the momentum variables is performed, this time using the new values of the position variables (Eq. (20)). This method preserves volume in (q, p) space exactly.

3.3. MCMC from Hamiltonian Dynamics

In order to use Hamiltonian dynamics to sample from a distribution, the density function of this distribution has to be translated to a potential energy function and auxiliary "momentum" variables have to be introduced. To obtain samples, a Markov chain is generated, in which each iteration resamples the momentum variables and performs a Metropolis update with a proposed new state obtained via Hamiltonian dynamics.

3.3.1 Canonical Distributions

The canonical distribution for an energy function $E(x)$ for state x is defined as

$$P(x) = \frac{1}{Z} \exp(-E(x)/T), \tag{21}$$

where T is the temperature of the system and Z is a normalizing constant needed for this function to integrate to one. Viewing this definition the opposite way, a target distribution $P(x)$ can be formulated as a canonical distribution with $T = 1$ by letting

$$E(x) = -\log P(x) - \log Z, \tag{22}$$

where Z is any convenient positive constant.

Because the Hamiltonian is an energy function for the joint state of position q and momentum p, it also defines a joint distribution:

$$P(q, p) = \frac{1}{Z} \exp(-H(q, p)/T). \tag{23}$$

As H is invariant under Hamiltonian dynamics, a Hamiltonian trajectory will move within a hyper-surface of constant probability density. The joint density of a Hamiltonian with the form $H(q, p) = U(q)+K(p)$ can be obtained by simply inserting:

$$P(q, p) = \frac{1}{Z} \exp(-U(q)/T) \exp(-K(p)/T). \tag{24}$$

Note that the variables q and p are independent in Eq. (24) and each have canonical distributions with energy functions $U(q)$ and $K(p)$.

3.3.2 Energy Formulation of Bayesian Learning

Recall that q was used to represent variables of interest, while p was introduced just to allow Hamiltonian dynamics to operate. For Bayesian deep learning, the goal is to sample weight vectors q_t from their posterior distribution $P(q \mid \mathcal{D})$. This target distribution can be expressed as a canonical distribution (with $T = Z = 1$) using a potential energy function defined as

$$U(q) = -\log P(q \mid \mathcal{D}), \tag{25}$$

such that

$$P(q \mid \mathcal{D}) = \exp(-U(q)). \tag{26}$$

The distribution for the momentum variables p can be chosen arbitrarily and is specified via the kinetic energy function $K(p)$, which is typically defined as $K(p) = \frac{1}{2}|p|^2$, corresponding to a multivariate Gaussian distribution.

The Hamiltonian is defined as the sum of the potential and kinetic energy:

$$H(q, p) = U(q) + \frac{1}{2}|p|^2. \tag{27}$$

According to Eqs. (23) and (24), this Hamiltonian defines a joint probability distribution over the phase space of (q, p):

$$\begin{aligned} P(p, q) &\propto \exp(-H(q, p)) \\ &= \exp(-U(q) - K(p)) \\ &= \exp(-U(q)) \exp(-K(p)) \\ &= P(q \mid \mathcal{D}) \exp(-K(p)). \end{aligned} \tag{28}$$

As mentioned before, q and p are independent in such an equation and the marginal distribution for q in Eq. (28) is again the original target distribution $P(q \mid \mathcal{D})$. To sample from the target distribution, one can therefore instead sample from this joint distribution for q and p, and just ignore the values obtained for p.

3.3.3 Simulating Hamiltonian Dynamics for Bayesian Learning

Simulating the Hamiltonian dynamics of the system through fictitious time t is achieved according to Eqs. (16) and (17). Note that this dynamics leaves H constant and preserves the volumes of regions in phase space. Therefore, it visits points on a surface of constant H with uniform probability.

As the leapfrog method (Eqs. (18) to (20)) also exactly maintains the preservation of phase space volume, it's a suitable discrete approximation for simulating the dynamics. However, the leapfrog method doesn't leave H constant, leading to slightly suboptimal acceptance probabilities in the MH framework.

3.3.4 The Hamiltonian Monte Carlo Algorithm

The Hamiltonian Monte Carlo (HMC) algorithm is a variant of the Metropolis-Hastings (MH) algorithm, which generates a Markov chain by considering random changes to its state. In contrast to the MH algorithm, candidate changes are produced by picking a random value for p via the joint distribution of (q, p) and then performing some predetermined number of leapfrog steps.

The algorithm consists of two steps. The first step only changes the momentum variables, while the seconds step may change both position and momentum variables. Both steps leave the canonical joint distribution of (q, p) invariant. The steps are the following:

1. Draw a new momentum vector p from its multivariate Gaussian distribution.

2. Perform a Metropolis update with a candidate state obtained via Hamiltonian dynamics.

In the first step, the momentum variables are reset by randomly drawing new values from their Gaussian distribution, independent of the current momentum or position variables. This step leaves the canonical joint distribution invariant, as q remains unchanged and p is drawn from its correct conditional distribution given q, which is the same as its marginal distribution because q and p are independent.

In the second step, a Metropolis update is performed for a candidate state obtained via Hamiltonian dynamics. Given the current state (q, p), Hamiltonian dynamics is simulated for L steps via the leapfrog method, as described in Sec. 3.3.3. After performing these L steps, the momentum variables are negated, giving a candidate state (q^*, p^*). This proposed state is then accepted as next state in the Markov chain with probability

$$\begin{aligned}\min[1, \frac{P(q^*, p^*)}{P(q, p)}] &= \min[1, \exp(-H(q^*, p^*) + H(q, p))] \\ &= \min[1, \exp(-U(q^*) + U(q) - K(p^*) + K(p))].\end{aligned} \tag{29}$$

A candidate change is thus always accepted if it lowers the energy H or leaves it unchanged. If the change would increase the energy, it is accepted with probability $\exp(-\Delta H) = \exp(-H(q^*, p^*) + H(q, p))$, otherwise rejected. For an exact simulation of Hamiltonian dynamics, these proposed changes would always be accepted, but since the leapfrog method is only approximate, H can sometimes increase and thus changes are sometimes rejected.

4. Stochastic Dynamics

Stochastic dynamics describes the dynamics of a system that is subject to random fluctuations. Thus, it can serve as a framework for performing Bayesian learning and also for describing stochastic variations of the presented MCMC algorithms. For algorithms like the presented HMC algorithm, such stochastic variations are a necessity for practical use, because the full gradient computation of $\nabla U(q)$ is computationally infeasible on large datasets.

In Sec. 4.1, alternative approaches to Bayesian deep learning are presented from a stochastic dynamics perspective. Stochastic variations of the HMC algorithm are introduced in Sec. 4.2.

The ideas described in this section are based on [6], [1], [5], [3], and [10].

Stochastic Gradient Methods *Stochastic gradient (SG)* methods offer an efficient way of calculating the approximate gradient for a loss function $\mathcal{L}$ defined on parameters θ for large datasets. The central idea of SG methods is to replace the exact gradient $\nabla_\theta \mathcal{L}(\theta)$ with a stochastic approximation $\nabla_\theta \tilde{\mathcal{L}}(\theta)$. This noisy estimate is based on a minibatch $\tilde{\mathcal{D}}$ sampled uniformly at random from the full dataset $\mathcal{D}$. For example, when minimizing the negative log-likelihood of a posterior probability distribution $P(\theta \mid \mathcal{D})$, the loss gradient is approximated as:

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta) &\propto -\nabla_\theta \log P(\mathcal{D} \mid \theta) - \nabla_\theta \log P(\theta) \\ &\approx -\frac{N}{n} \sum_{x \in \tilde{\mathcal{D}}} \nabla_\theta \log P(x \mid \theta) - \nabla_\theta \log P(\theta).\end{aligned} \tag{30}$$

Here, $N = |\mathcal{D}|$ denotes the size of the full dataset and $n = |\tilde{\mathcal{D}}|$ denotes the size of the minibatches.

By assuming that the observations $x \in \mathcal{D}$ are independent and appealing to the central limit theorem, the exact gradient can thus be approximated as

$$\nabla_\theta \tilde{\mathcal{L}}(\theta) \approx \nabla_\theta \mathcal{L}(\theta) + \mathcal{N}(0, \mathbb{V}[\theta]). \tag{31}$$

Empirically, a minibatch size in the order of hundreds of data points is sufficient for the central limit theorem approximation to be valid.

Stochastic gradient descent (SGD) describes the method of minimizing a given loss function $\mathcal{L}(\theta)$ by updating the parameters θ according to the stochastic gradient:

$$\theta_{t+1} = \theta_t - \epsilon \nabla_\theta \tilde{\mathcal{L}}(\theta), \tag{32}$$

where ϵ is called the step size.

4.1. Stochastic Dynamics in Bayesian Learning

Conventional frequentist learning can be made Bayesian by introducing only slight modifications to the optimization algorithm.

For example, by picking a specific learning rate and keeping it constant or by introducing the right kind of noise into stochastic gradient descent, this method can be used to perform Bayesian learning. These two approaches are briefly presented in this section.

4.1.1 Stochastic Gradient Descent as Approximate Bayesian Inference

Stochastic Gradient Descent (SGD) with a constant learning rate simulates a Markov chain with stationary distribution, as is shown in [6]. This so called *Constant Stochastic Gradient Descent (CSGD)* can be used as an approximate Bayesian posterior inference algorithm for large datasets.

Conventional SGD optimizes a function by following noisy gradients with a typically decreasing step size. This procedure provably attains the optimum (or a local optimum for non-convex functions), while at the same time allowing efficient optimization using massive amounts of data.

In contrast, constant SGD moves towards an optimum of the objective function as well, but then bounces around its vicinity, as the learning rate doesn't decrease to allow for smaller step sizes. The central idea of constant SGD is to use the resulting stationary distribution of the simulated Markov chain to approximate the posterior. This can be achieved with minimal implementation effort.

Assumptions First, however, four assumptions have to be made:

1. Invoking the central limit theorem, assume that the gradient noise is Gaussian with covariance $\frac{1}{n}\mathbb{V}[\theta]$, where n describes the minibatch size.

2. Assume that the covariance matrix of θ is approximately constant w.r.t. θ: $\mathbb{V}[\theta] \approx C$. As a symmetric positive-semidefinite matrix, C factorizes as $C = BB^T$.

3. Assume the finite-difference equation

$$\Delta\theta_t = \theta_{t+1} - \theta_t = -\epsilon\nabla_\theta\mathcal{L}(\theta_t) + \frac{\epsilon}{\sqrt{n}}B\Delta W \quad (33)$$

with $\Delta W \sim \mathcal{N}(0, I)$ can be approximated by the stochastic differential equation

$$d\theta_t = -\epsilon\nabla_\theta\mathcal{L}(\theta)dt + \frac{\epsilon}{\sqrt{n}}BdW(t). \quad (34)$$

4. Assume that the stationary distribution of the iterates θ_t is constrained to a region where the loss is well approximated by a quadratic function

$$\mathcal{L}(\theta) \approx \frac{1}{2}\theta^T A\theta. \quad (35)$$

These assumptions result in a specific kind of stochastic process, called multivariate *Ornstein-Uhlenbeck* process [9]. This process has an analytical stationary distribution that is Gaussian, which is used in [6] to analyze the properties of SGD and prove the following results.

Constant SGD The optimal constant learning rate for minimizing the KL divergence from the stationary distribution of CSGD to the posterior is

$$\epsilon^* = 2\frac{n}{N}\frac{D}{\mathrm{Tr}[BB^T]}, \quad (36)$$

where n is the minibatch size, N is the size of the full dataset, D is the dimension of θ and $\mathbb{V}(\theta) \approx C = BB^T$, as above.

Preconditioned Constant SGD Instead of a scalar learning rate, a preconditioning matrix can be used. The optimal full preconditioner for constant SGD is

$$H^* = 2\frac{n}{N}(BB^T)^{-1}. \quad (37)$$

Diagonally Preconditioned Constant SGD The optimal diagonal preconditioner for constant SGD is

$$H^*_{kk} = \frac{2n}{N(BB^T)_{kk}}. \quad (38)$$

4.1.2 Bayesian Learning via Stochastic Gradient Langevin Dynamics

Stochastic Gradient Langevin Dynamics (SGLD) describes another approach for Bayesian learning on large scale datasets using smaller minibatches of data. It was first proposed in [10]. The algorithm is again similar to the conventional SGD algorithm, but this time random noise is injected into the parameter updates of the optimization process. This way, the parameters will again converge to their posterior distribution rather than just their maximum likelihood estimate.

SGLD thus offers a surprisingly simple solution that combines the best of both worlds, only relying on stochastic instead of exact gradients, but nevertheless sampling from the posterior distribution of the weights. Its biggest advantage is that it transitions from stochastic optimization to posterior sampling seamlessly.

The SGLD Algorithm The central idea of the SGLD algorithm is to combine stochastic gradient descent with *Langevin dynamics*.

Langevin dynamics is an approach to modeling a dynamical system by simplifying the model and accounting for omitted degrees of freedom by using stochastic differential equations.

The combination of the two concepts is implemented in a straightforward way, by simply adding random Gaussian noise to the stochastic gradients. The added noise is balanced with the current step size, which asymptotically goes to zero.

The proposed update for the parameters to be optimized is then

$$\Delta\theta_t = -\frac{\epsilon_t}{2}\nabla\tilde{\mathcal{L}}(\theta) + \eta_t, \quad (39)$$

where $\eta_t \sim \mathcal{N}(0, \epsilon_t)$ is the random Gaussian noise balanced by the current step size ϵ_t. It is essential for the step size to decrease to zero at rates satisfying

$$\sum_{t=1}^{\infty}\epsilon_t = \infty \text{ and } \sum_{t=1}^{\infty}\epsilon_t^2 < \infty. \quad (40)$$

There are two sources of stochasticity in Eq. (39):

1. The noise in the stochastic gradient with variance $(\frac{\epsilon_t}{2})^2\mathbb{V}[\theta_t]$ and
2. the injected Gaussian noise η_t with variance ϵ_t.

In the initial phase of running this algorithm, the stochastic gradient noise dominates and the algorithm thus imitates the efficient SGD algorithm. However, in the later phase, as $\epsilon_t \to 0$, the injected noise dominates and the algorithm thus imitates Langevin dynamics. SGLD transitions smoothly between the two phases.

Sampling from the Posterior When using this algorithm to produce samples from the posterior distribution, it is important to only start collecting samples after it has entered the posterior sampling phase. This only happens when it imitates Langevin dynamics.

Performance The SGLD algorithm provides a simple way to perform Bayesian learning, even on large datasets. Nevertheless, it also outperforms the conventional SGD algorithm on the task of MNIST classification, as can be seen in Fig. 3.

4.2. Stochastic Dynamics Variants of HMC

This section presents stochastic variations of the previously defined HMC algorithm. Stochastic dynamics forms the underlying framework for describing and analyzing these variations.

In Sec. 4.2.1, another way of changing the value of the Hamiltonian inside the HMC algorithm is outlined. Sec. 4.2.2 presents the stochastic dynamics perspective on a special case of the HMC algorithm, called *Langevin Monte Carlo*.

As mentioned before, the HMC algorithm requires gradient computations over the whole dataset. This limitation makes its exact application infeasible for practical use on large datasets. To alleviate this problem, stochastic variants can be defined. One such variant, called *Stochastic Gradient Hamiltonian Monte Carlo* is presented in Sec. 4.2.3.

4.2.1 A Simple Stochastic Dynamics Variant of HMC

The leapfrog iterations of the HMC algorithm keep H approximately constant. In order to create a Markov chain that converges to the desired joint distribution from Eq. (28), it is thus necessary to interleave them with steps that can change H. It is convenient for these changes to only affect p, since it enters into H in a simple way.

The previously presented way of doing this is sampling a new momentum vector in each iteration of the MCMC algorithm. Another possible approach to changing the momentum variables p is inspired by stochastic dynamics and was originally proposed in [1]. The idea is to perform stochastic steps of the form

$$p' = \alpha p + (1-\alpha^2)^{\frac{1}{2}} r, \tag{41}$$

where $0 \leq \alpha < 1$ and r is a random vector with independent Gaussian components: $r \sim \mathcal{N}(0, I)$. These stochastic steps leave the joint distribution invariant. Using a value of α close to one is best, as this reduces the random walk aspect.

4.2.2 Langevin Monte Carlo

Langevin Monte Carlo (LMC) is a special case of the HMC algorithm, in which only a single leapfrog step is performed per iteration, *i.e.* $L = 1$.This variant is thus described by the steps

1. Sample momentum variables $p \sim \mathcal{N}(0, I)$.
2. Generate proposed values q^* and p^*:

$$q_i^* = q_i - \frac{\epsilon^2}{2}\frac{\partial U}{\partial q_i}(q) + \epsilon p_i \tag{42}$$

$$p_i^* = p_i - \frac{\epsilon}{2}\frac{\partial U}{\partial q_i}(q) - \frac{\epsilon}{2}\frac{\partial U}{\partial q_i}(q^*). \tag{43}$$

The proposed state (q^*, p^*) is then accepted as the new state with probability

$$\min[1, \exp(-(U(q^*) - U(q)) - \frac{1}{2}\sum_i((p_i^*)^2 - p_i^2))]. \tag{44}$$

The name *Langevin Monte Carlo* arises because Eq. (42) is a *Langevin equation*.

4.2.3 Stochastic Gradient Hamiltonian Monte Carlo

Hamiltonian Monte Carlo methods are limited by the required gradient computation for the simulation of the Hamiltonian dynamical system. These gradient computations are often infeasible since they utilize the entire dataset. Replacing the exact gradient with a stochastic gradient can alleviate this problem and hence enable the application of HMC to bigger datasets. This method marries the efficiencies in state space exploration of HMC with the computational efficiencies of stochastic gradients. It was first proposed in [3].

Naïve Stochastic Gradient HMC The naïve approach to obtaining a stochastic version of the HMC algorithm is to simply replace $\nabla U(q)$ in Eq. (17) with its stochastic approximation $\nabla \tilde{U}(q)$. This introduces noise in the momentum update, which according to Eq. (31) becomes

$$\frac{dp}{dt} = -\nabla \tilde{U}(q) \approx -\nabla U(q) + \mathcal{N}(0, \mathbb{V}[q]). \tag{45}$$

The corresponding hockey puck analogy consists of the same scenario, the puck on a frictionless surface of varying height, but this time with a random wind blowing as well.

However, as shown in [3], this naïve approach no longer leads to Hamiltonian dynamics encapsulating the desired target distribution. Thus, HMC with stochastic gradients requires a frequent Metropolis-Hastings correction step, or alternatively, long simulation runs with low acceptance probabilities. This finding can be seen in Fig. 2.

Stochastic Gradient HMC with Friction The solution to this problem consists of introducing an additional friction term to the momentum update of the naïve method. This modified method is called *Stochastic Gradient HMC (SGHMC)*. The resulting second-order Langevin dynamics maintains the desired target distribution as the stationary distribution.

The update equation with added *friction* term is given as:

$$\frac{dp}{dt} = -\nabla\tilde{U}(q) - \frac{1}{2}\mathbb{V}[q]p + \mathcal{N}(0, \mathbb{V}[q]). \qquad (46)$$

Here, $\frac{1}{2}\mathbb{V}[q]p$ is the friction term.

The corresponding hockey puck analogy this time is to imagine street hockey instead of ice hockey, which introduces friction from the asphalt. There's still a random wind blowing, but the friction of the surface prevents the puck from running far away.

This type of dynamical system is called second-order Langevin dynamics. In comparison, the Langevin dynamics found in SGLD (Sec. 4.1.2) and LMC (Sec. 4.2.2) are both first-order.

Note that when the additional stochastic noise is removed, SGHMC reduces to a stochastic gradient method with momentum.

Performance While the results obtained from naïve stochastic gradient HMC diverge significantly from the target distribution unless a Metropolis-Hastings correction is added (see Fig. 2), the final results of the sampling based methods improve upon optimization-based methods, showing an advantage of Bayesian inference on the task of MNIST classification, as can be seen in Fig. 3.

5. Summary

Bayesian deep learning was introduced and contrasted against conventional frequentist deep learning. The concept of a Bayesian neural network was introduced and the *Bayes by Backprop* algorithm for training such networks using a variational approach was explained. A short analysis of the performance achieved by Bayesian neural networks was presented, hinting that Bayesian neural networks offer a promising way of tackling overfitting, incorporating uncertainty into predictions and reducing model size at runtime.

Markov Chain Monte Carlo (MCMC) methods were introduced as a way to sample from complex probability distributions and thus enable Bayesian inference for neural networks. Furthermore, Hamiltonian dynamics was motivated and introduced. The concepts of MCMC methods and Hamiltonian dynamics were then combined, to introduce a special type of MCMC method, called the *Hamiltonian Monte Carlo (HMC)* algorithm. It was shown how to formulate and simulate Hamiltonian dynamics for the HMC algorithm and also how to utilize the HMC algorithm for Bayesian deep learning.

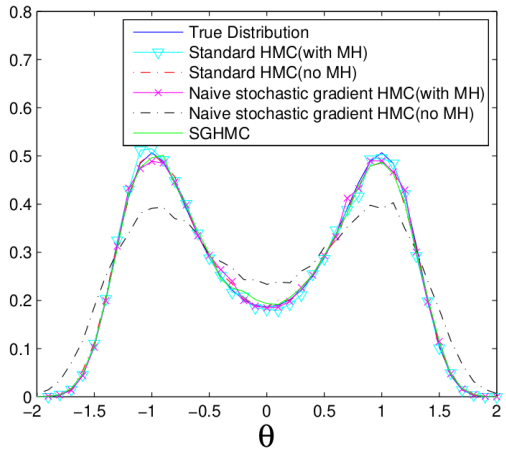

Figure 2. **Empirical Distributions of Different Sampling Algorithms:** It can be seen that the naïve stochastic gradient HMC algorithm without MH corrections no longer samples from the true target distribution. Figure taken from [3].

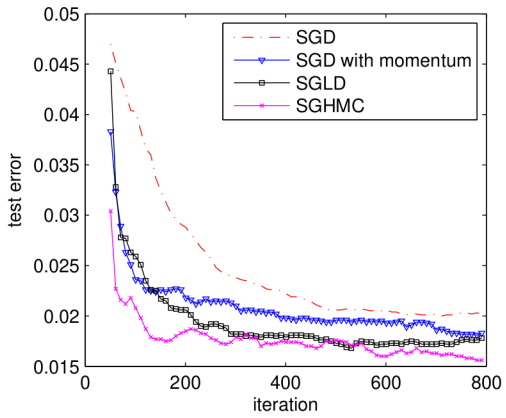

Figure 3. **Test Error on MNIST Classification:** Sampling based methods (SGLD, SGHMC) outperform optimization-based methods (SGD, SGD with momentum) on MNIST classification. The SGHMC algorithm also outperforms SGD, SGD with momentum and SGLD. Figure taken from [3].

Stochastic dynamics was introduced as a way to approach Bayesian deep learning by introducing stochasticity into the optimization process of conventional deep learning. Additionally, multiple stochastic variations of the HMC algorithm based on stochastic dynamics as an underlying framework were presented and their performance was briefly compared.

References

[1] Hans C Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of chemical physics*, 72(4):2384–2393, 1980. 7, 9

[2] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015. 1, 2, 3, 4

[3] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning*, pages 1683–1691. PMLR, 2014. 7, 9, 10

[4] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[5] AD Kennedy. The theory of hybrid stochastic algorithms. In *Probabilistic methods in quantum field theory and quantum gravity*, pages 209–223. Springer, 1990. 7

[6] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *arXiv preprint arXiv:1704.04289*, 2017. 7, 8

[7] Radford Neal. Bayesian learning via stochastic dynamics. *Advances in Neural Information Processing Systems*, 5, 1992. 1, 4

[8] Radford M Neal et al. MCMC using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011. 4

[9] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930. 8

[10] 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. 7, 8