The School of Mathematics

THE UNIVERSITY
of EDINBURGH

Bayesian Uncertainty Reduction Analysis by Langevin Dynamics Sampling

by

Xuexun Lu

Dissertation Presented for the Degree of
MSc in Computational Applied Mathematics

August 2022

Supervised by
Prof. Benedict Leimkuhler

Abstract

In this paper, we aim to use Langevin dynamics sampling, a specific Bayesian inference technique based on molecular dynamics, to reduce the uncertainty of the data. The discussion commences with a list of the deficiencies of conventional neural network optimizers, the advantages of Bayesian inference, and the most recent research in Bayesian neural networks. In the following section, the fundamental physics definitions and stochastic processes associated with Langevin dynamics are introduced and clarified. Combining the numerical methods for Langevin dynamics, we describe the simulation algorithms for the Langevin path. In the experimental section, the handwritten images of the digits 1 and 7 are chosen as the objective of our uncertainty analysis. The comparison between normal stochastic gradient descent and Langevin sampling methods comes first. The distinction between an overdamped and an underdamped Langevin sampling procedure is discussed. Then, we introduce a neural network with a similar structure but more neurons. Using it as a benchmark, Langevin sampling methods can be considered to reduce the complexity of neural networks. Also discussed is the mechanism by which the Langevin sampling method reduces uncertainty.

Acknowledgments

I'd like to thank everyone who contributed comments and discussions while I was researching the various papers that comprised this thesis. I would like to thank (in alphabetical order) Christian Jimenez Beltran, Pavel Yordanov Bozmarov, and many other reviewers for their helpful comments and discussions. I would further like to give special thanks to my supervisor Benedict Leimkuhler and his PhD student Katerina Karoni. Lastly, I would like to thank my family especially my mother for supporting my master study.

Own Work Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

Deep neural networks (DNNs) have made tremendous strides in the past decade, encouraging their application in numerous scientific domains where complex systems must be represented or comprehended, such as earth observation, medical image analysis, and robotics. Although DNNs have gained popular in high-risk domains like medical image analysis [33, 44, 46, 17, 43, 5] or autonomous vehicle control [7, 4, 1, 24], their deployment in mission- and safety-critical real world applications remains limited. The primary contributors to this constraint are

- the lack of expressiveness and transparency of a deep neural network's inference model, which makes it difficult to trust their results [44],
- the inability to distinguish between in-domain and out-of-domain samples [20, 31], as well as the sensitivity to domain alterations [37],
- the inability to give credible uncertainty estimates for a deep neural network's decision [2] and frequently occurring overconfident predictions [12, 61],
- the sensitivity to adversarial attacks that make deep neural networks vulnerable for sabotage [42, 48, 52].

These factors are primarily based on an inherent uncertainty in the data (data uncertainty) or a lack of neural network knowledge (model uncertainty). To overcome these restrictions, it is necessary to offer uncertainty estimates so that uncertain forecasts can be ignored or forwarded to human specialists [9]. Remote sensing [45] is an example of a field where the data sources are highly heterogeneous and labelled data is scarce in which it is essential to provide uncertainty estimates for safe decision-making. In addition to fields where uncertainties are essential to the learning procedures, such as active learning [10, 36] or reinforcement learning [9, 13, 14, 25], estimations of uncertainty are crucial.

Researchers have shown a rising interest in the estimation of DNNs in recent years [3, 9, 18, 28, 64, 62, 56, 41]. The most prevalent method for estimating the uncertainty of a prediction (the predictive uncertainty) is based on modelling the uncertainty generated by the model (epistemic or model uncertainty) and the uncertainty caused by the data separately (aleatoric or data uncertainty). While the former can be reduced by enhancing the model learned by the DNN, the latter cannot be reduced. The most important approaches for modeling this separation are Bayesian inference [3, 9, 32, 1, 15], ensemble approaches [18, 55, 59], test time data augmentation approaches [50, 58], or single deterministic networks containing explicit components to represent the model and the data uncertainty [53, 47, 29, 28, 40]. Estimating the forecast uncertainty is insufficient for making sound decisions. Moreover, it is essential to ensure the accuracy of uncertainty estimates. To this purpose, the calibration property (the degree of dependability) of DNNs has been explored, and methods for re-calibration have been developed [12, 60, 63] to get reliable (well-calibrated) uncertainty estimates.

This study will focus on the Langevin dynamics sampling method, which is part of Bayesian inference and is simulated by a molecular dynamical system. The Langevin dynamics, the Hamiltonian system, the canonical distribution, and their mathematical properties will be presented as the first step of a comprehensive physics foundation. Langevin dynamics explains the optimizer mechanism in the sampling process. This dynamic system has a strong relationship with the Hamiltonian system and the canonical distribution.

Then, we will couple the Bayesian neural network to Langevin dynamics and develop simulation algorithms for two distinct Langevin dynamics cases. In the experimental section, we will evaluate the sampling performance of the samplers on the MNIST dataset and examine the effect of momentum on the sampling process.

The article will conclude with an analysis of MNIST uncertainty in relation to the Langevin dynamics sampling method. We will compare the performance of Langevin dynamics samplers with a more complex neural network and investigate the sampling uncertainty reduction mechanism.

2 Preliminaries

In this section, we will first introduce and thoroughly explain some fundamental physics concepts relating to Langevin dynamics, as well as their numerical simulation techniques. Then, with the aid of these concepts, the Bayesian learning scheme will be illustrated. These prerequisites would be helpful for constructing our Bayesian model and the stochastic optimizer on BNN for uncertainty analysis.

2.1 Hamiltonian System

The *Hamiltonian* of the system, which can often be interpreted as describing the total energy of any given state in that system. Each physical phenomenon (e.g. electromagnetism, atomic bonding, particles in a potential well, etc.) may correspond to a single Hamiltonian H, while each type of mechanics (classical, quantum, statistical, etc.) corresponds to a different way of using that Hamiltonian to describe a physical system. In this paper, we only using its application on classical mechanics aspect.

In classical physics, the Hamiltonian is a function $(q, p) \mapsto H(q, p)$ of the positions q and momenta p of the system, which then evolve according to Hamilton's equations

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\partial H}{\partial p}; \frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial H}{\partial q}.$$

For simplicity, we define our standard Hamiltonian as

$$H(q, p) = p^T Mp/2 + U(q). \tag{2.1}$$

where $U(\cdot)$ is considered as *potential energy* function, $p^T Mp/2$ stands for the *kinetic energy* and M is the corresponding *mass matrix* thoughout the paper.

2.2 Canonical Distribution

When it comes to formulation of the probabilistic model in terms of energy, we should introduce the canonical distribution which builds a bridge between the energy and probability in molecular dynamics. The original physics background is energy exchange process between two weakly coupled molecular system [22]. For example, we could imagine it as pouring a bottle of cold water into a hot bath. Our interest is the particles' interaction of these two liquid with respect to energy transition. The energy in this case is assumed to be in the form of heat, with equilibrium naturally associated to the systems having matching temperature. According theoretical assumption, the sole mechanism of interaction is just in the form of energy exchange, they do not exchange particles (or momentum) [22]. By the computing thermal equilibrium from the energy transformation with the help of stationary point of Gibbs-Boltzmann entropy [11], we would obtain the phase space density associated to the canonical ensemble. It is also referred as *Gibbs-Boltzmann density*:

$$\rho_\beta = Z_{\text{can}}^{-1} e^{-\beta H(q,p)}. \tag{2.2}$$

The exponential parameter is defined as $\beta = 1/(\mathrm{k_B T})$ where $\mathrm{k_B}$ and T represent *Boltzmann constant* and *absolute temperature* respectively. Usually, we denote the phase state of system as $\omega = [q, p]$. The quantity Z_{can} is a normalizing constant so that this is a probability density. Therefore, it is necessary that the quantity Z_{can} defined by

$$Z_{\text{can}} = \int e^{-\beta H(q,p)} \mathrm{d}\omega$$

be finite. i.e, that the integral converge.

If we just focus on potential energy $U(q)$, the density is defined as:

$$\rho_\beta = \hat{Z}_{\text{can}}^{-1} e^{-\beta U(q)}. \tag{2.3}$$

Similarly, the quantity would be

$$\hat{Z}_{\text{can}} = \int e^{-\beta U(q)} \mathrm{d}q.$$

Therefore, $\hat{Z}_{\text{can}}$ and Z_{can} are mathematically defined as a constant that normalises the probability density.

2.3 Stochastic Processes

Prior to delving into Langevin dynamics, it is necessary to define a number of significant stochastic processes formally in order to facilitate future discussion.

2.3.1 Markov Processes

As [38][p.29] stated, "Roughly speaking a Markov process is a stochastic process that retains no memory of where it has been in the past: only the current state of a Markov process can influence where it will go next. A bit more precisely, a Markov process is a stochastic process whose past and future are statistically independent, conditioned on its present state."

We only need to consider the discrete-time case *Markov processes* in paper. In particular, it is known as the *Markov Chain*:

It is a series of random variables, $X^{(0)}, X^{(1)}, X^{(2)}, \ldots$ in which the influence of the values of $X^{(0)}, \ldots, X^{(n)}$ on the distribution of $X^{(n+1)}$ is mediated entirely by the value of $X^{(n)}$. Mathematically,

$$P(x^{(n+1)}|x^{(n)}, \{x^{(t)} : t \in \xi\}) = P(x^{(n+1)}|x^{(n)}) \tag{2.4}$$

where ξ is any subset of $\{0, \cdots, n-1\}$ and n should be an integer.

2.3.2 Orstein-Uhlenbeck Process

The Ornstein–Uhlenbeck process was introduced by Leonard Ornstein and George Uhlenbeck in 1930 as a model for the velocity of a Brownian particle under the influence of friction [54]. It is defined by the following stochastic differential equation:

$$\mathrm{d}X = -\gamma\, X\, \mathrm{d}t + \eta\, \mathrm{d}W \tag{2.5}$$

where γ is usually considered as *friction coefficient* and η is a parameter related with *temperature*. And Ornstein–Uhlenbeck process is also a Markov process [38]. The exact distributional solution is

$$X(t) = e^{-\gamma t} X(0) + \eta \sqrt{(2\gamma)^{-1}(1 - e^{-2\gamma t})} R(t) \tag{2.6}$$

where $R(t)$ is an independent Gaussian noise with respect to time. When $X \in \mathbb{R}$, $R(t) \sim \mathcal{N}(0,\, 1)$. In multidimensional case, $R(t) \sim \mathcal{N}(0, \mathbb{I})$.

2.4 Langevin Dynamics

The development of thermodynamics and the statistical interpretation of its concepts at the end of the nineteenth and early twentieth centuries enabled us to bridge the scale gap between the macroscopic world and the world of atoms. In 1905 and 1906, Einstein and Smoluchowski demonstrated that the Brownian motion of particles of measurable size is a manifestation of the motion of atoms in fluids [6, 57]. Langevin showed in 1908 how to put the subtle effect of the randomness of the atomic world, which is responsible for the fluctuating force driving the motion of the Brownian particle and the viscosity of the 'macroscopic' flow occurring around the same Brownian particle, into a coherent framework [39]. All of these legends made a significant contribution to modern *Langevin Dynamics*.

The *Langevin equation* describes the motion of a particle that is subject to friction and stochastic forcing [38]:

$$\ddot{q} = -\nabla U(q) - \gamma\dot{q} + \sqrt{2\gamma\beta^{-1}}\dot{W}. \tag{2.7}$$

It comes from Newton's equation of motion with two additional terms, a linear dissipation term $\gamma\dot{q}$ and a stochastic forcing $\sqrt{2\gamma\beta^{-1}}\dot{W}$. The noise $\xi(t) = \sqrt{2\gamma\beta^{-1}}\dot{W}$ and dissipation $-\gamma\dot{q}$ in Eq. (2.7) are controlled by two parameters, the *friction coefficient* γ (also noted as damping constant) and *temperature* parameter $\beta^{-1} = \mathrm{k_B T}$. And $U(q)$ is considered as the particle interaction *potential*. From Eq. (2.7), the next position of the particle purely depends on the current position. Therefore, the particle's dynamics above are Markov processes.

2.4.1 Underdamped Langevin Equation

By introducing the momentum $p_t = \dot{q}_t$, we can convert the Langevin equation Eq. (2.7) to a system of first-order stochastic differential equations in the phase space $(q, p) \in \mathbb{R}^{2d}$:

$$\mathrm{d}q_t = p_t \mathrm{d}t \tag{2.8a}$$

$$\mathrm{d}p_t = -\nabla U(q_t)\mathrm{d}t - \gamma p_t \mathrm{d}t + \sqrt{2\gamma\beta^{-1}}\mathrm{d}W_t \tag{2.8b}$$

where $\mathrm{d}W_t = \mathrm{d}W(t)$ is the infinitesimal increment of a vector of stochastic Wiener processes $W(t)$. Without loss of generality, we assume that q and p should have same dimensionality d.

Eq. (2.8) explicitly describes the particles' motion in underdamped Langevin case. When $t \to \infty$, the distribution of the particles would converge to a certain equilibrium ρ_∞ which is referred as *invariant distribution*.

The invariant distribution governs the long-time dynamics of the Markov process. In particular, when $X_0 \sim \rho_0$ initially, we have that

$$\lim_{t \to +\infty} P_t^* \rho_0 = \rho_\infty, \tag{2.9}$$

where P_t^* is an operator that advances the system in the time dimension.

2.4.2 Overdamped Langevin Equation

As γ grows in Eq. (2.7), particles lose their inertia and eventually enter the diffusive regime due to the intense friction they encounter [38]. In this case, the molecular dynamics would be referred to as *overdamped Langevin dynamics* or *Brownian dynamics*.

Replacing $\ddot{q}$ by 0 and setting $\gamma = 1$ in Eq. (2.7), we obtain the *overdamped Langevin equation*:

$$\mathrm{d}q_t = -\nabla U(q_t)\mathrm{d}t + \sqrt{2\beta^{-1}}\mathrm{d}W_t. \tag{2.10}$$

A notable change is that we eliminate the middleman momentum p from the origin system. In the other word, noise would directly influence the position of the particles. It reveals that the particles no longer possess inertia.

Similarly, overdamped Langevin dynamics as a special case of underdamped also has it own invariant distribution $\hat{\rho}_\infty$.

2.5 Numerical methods for Langevin dynamics

Over the past decade, my supervisor, Prof. Benedict Leimkuhler, and other mathematicians in the field of molecular science have refined the numerical simulation for Langevin dynamics. This section will discuss some stable and accurate numerical methods for solving Langevin dynamics.

2.5.1 Solving Overdamped Langevin Dynamics

A simple and popular method for numerical solution of Eq. (2.10) is the *Euler-Maruyama method*:

$$q_{n+1} = q_n - h\nabla U(q_n) + \sqrt{2\beta^{-1}h}R_n, \tag{2.11}$$

where R_n is a vector of random variables with standard normal distribution and h is the stepsize. This produces a sequence of points $q_0, q_1, q_2, \ldots$, which, following a certain relaxation period, are approximately distributed according to the canonical invariant distribution with first order accuracy [21]. According to the Euler-Maruyama method's property, time averages along discrete trajectories have inaccuracy proportional to h in the limit of large time (given the right conditions on the potential $U(q)$ and assuming no impacts from floating point rounding error).

A second order accuracy method could be obtained by making a simple modification. And that is

$$q_{n+1} = q_n - h\nabla U(q_n) + \sqrt{\frac{\beta^{-1}h}{2}}(R_n + R_{n+1}), \tag{2.12}$$

where R_n and R_{n+1} are independent for all n. This simulation is referred as *Leimkuhler-Matthews method* [21].

2.5.2 Splitting Underdamped Langevin Dynamics

In contrast to the overdamped Langevin dynamics, the underdamped case has stochastic differential equations in two dimensions. To solve the problem, we must first introduce the concept of *splitting method*.

In a splitting method for a deterministic system $\dot{z} = f(z)$, one divides the vector field f into exactly solvable parts, i.e. $f = f_1 + f_2$, which are treated sequentially within a timestep. Usually, we separate position and momentum into f_1 and f_2. For example, appling splitting method on Hamiltonian system with energy $H(x,p) = \sum_i p_i^2/(2m_i) + U(x_1, x_2, \ldots, x_N)$ is the "symplectic Euler" method defined by $f_1 = \sum_i m_i^{-1} p_i \partial_{x_i}$, $f_2 = -\sum_i \frac{\partial U}{\partial x_i}\partial_{p_i}$. The so-called position Verlet method is obtained by dividing the vector field as $f = \frac{1}{2}f_1 + f_2 + \frac{1}{2}f_1$, and by switching the roles of f_1 and f_2 we obtain the velocity Verlet method. Such explicit splitting methods are crucial for molecular dynamics, where the force calculation serves as the standard indicator of the difficulty of the computations per timestep [21].

Similarly, underdamped Langevin dynamics may be treated by splitting [51, 30]. So we divide the system Eq. 2.8 as follows:

$$\mathrm{d}\begin{bmatrix} q \\ p \end{bmatrix} = \underbrace{\begin{bmatrix} p \\ 0 \end{bmatrix}\mathrm{d}t}_{\mathrm{A}} + \underbrace{\begin{bmatrix} 0 \\ -\nabla U \end{bmatrix}\mathrm{d}t}_{\mathrm{B}} + \underbrace{\begin{bmatrix} 0 \\ -\gamma p \mathrm{d}t + \sqrt{2\gamma\beta^{-1}}\mathrm{d}W \end{bmatrix}}_{\mathrm{O}}, \tag{2.13}$$

and each of the three parts may be solved 'exactly'. In the case of the Orstein-Uhlenbeck part (labelled here simply as O) we mean by this that we realize the stochastic process by the equivalent formula Eq. 2.6

$$p(t) = e^{-\gamma t}p(0) + \sqrt{(1 - e^{-2\gamma t})\beta^{-1}}R(t), \tag{2.14}$$

where $R(t)$ is again a vector of independent standard normal random processes without correlation (Gaussian white noise).

2.5.3 BAOAB method

After splitting the system into three parts, we could discretize them and update our position q and momentum p by steps accordingly. The steps are given by a useful and easy-to-understand method, *BAOAB method*. The reason of the choice is it achieves 4th order accuracy on reaching the invariant distribution which is a superconvergence property [21]. This property will be helpful for our sampling in Bayesian neural network in later section.

From the discretization form Eq. (2.13) and explicit solution Eq. (2.6), we have the updating rules of three split steps as follows:

(A) dynamics step

$$\begin{aligned} q(t+h) &\leftarrow q(t) + hp(t) \\ p(t+h) &\leftarrow p(t) \end{aligned}$$

(B) dynamics step

$$\begin{aligned} q(t+h) &\leftarrow q(t) \\ p(t+h) &\leftarrow p(t) - h\nabla U(q(t)) \end{aligned}$$

(O) dynamics step

$$\begin{aligned} q(t+h) &\leftarrow q(t) \\ R(t+h) &\leftarrow \mathcal{N}(0, I) \\ p(t+h) &\leftarrow \exp(-\gamma h)p(t) + \sqrt{1/\beta}\sqrt{1-\exp(-2\gamma h)}R(t+h) \end{aligned}$$

As the name states, position q and momentum p will update by order B $\rightarrow$ A $\rightarrow$ O $\rightarrow$ A $\rightarrow$ B. Under this scheme, the stepsize we take in each A step and B step should be $h/2$. Because we must ensure that each type of dynamic step walks the same distance after one complete iteration.

After introducing the explicit steps for simulating BAOAB, we will use the it to solve a simple example of an underdamped Langevin dynamics system. Potential energy of the example one-dimensional Langevin system is defined as

$$U(q) = q^2(0.1 + \sin^2(q)). \tag{2.15}$$

Based on the previous discussion in Section 2.4 and canonical distribution, the invariant distribution of position and momentum should be

$$\rho_q \propto \exp(-U(q)), \tag{2.16}$$

$$\rho_p \propto \exp(-p^2/2). \tag{2.17}$$

The simulation of the example system by the BAOAB method is shown below:

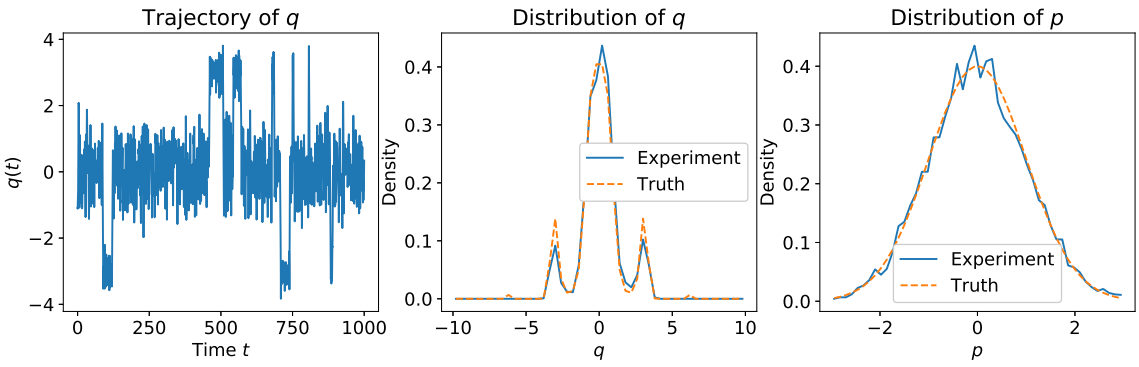

Figure 1: Position's trajectory, position distribution and momentum's distribution of the example underdamped Langevin dynamics system obtained by BAOAB method: It begins 10000 iterations of BAOAB method from randomly generated position $q_0 = -1.09441427$ and momentum $p_0 = -0.20038268$ with a stepsize of $h = 0.1$. $\beta = 1.0$ is the temperature coefficient of the Langevin dynamic system. And the friction coefficient $\gamma = 1.0$.

From Figure 1, 10000 iterations of the BAOAB method could produce an acceptable distribution in terms of the true position's distribution ρ_q that is the invariant distribution we mentioned previously.

This section introduced numerical stimulation methods for underdamped and overdamped Langevin dynamics. In particular, a BAOAB-stimulated dynamics system is presented. In the next section, we will discuss an important Langevin dynamics property.

2.6 Ergodicity for Langevin Dynamics

Ergodicity expresses the notion that a point of a moving system, be it a dynamical system or a stochastic process, will eventually visit all regions of the system's space in a uniform and random fashion. A simple example of an erogodic process is that a large number of people flipping a coin once is equivalent to a single person flipping a coin a large number of times. Due to the fact that the coin flipping would result in a uniform head or tail outcome across all probability state space regions if we have enough trials.

Mathematically, an observable is called ergodic if its time average equals its expectation in terms of the state space, that is, if it satisfies Birkhoff 's equation

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T f(\omega(t))\mathrm{d}t = \int_\Omega f(\omega)P(\omega)\mathrm{d}\omega \tag{2.18}$$

Here, f is determined by the system's state ω. In Langevin dynamics, our state would be position q and momentum p. On the left-hand side, the state in turn depends on time t. On the right-hand side, a timeless $P(\omega)$ assigns weights to ω. If Eq. (2.18) holds we can avoid integrating over time (up to the divergent averaging time, T, on the left), and instead integrate over the space of all states, Ω (on the right). In our case $P(\omega)$ is given as the distribution obtained by Langevin dynamics sampling.

Why does Langevin Dynamics has Ergodicity? Chapter 6.4 from [22] provides the specifics of demonstrating the ergodicity of the Langevin dynamics. According to the definition of ergodicity, the Langevin dynamics sampling process has the ability to eventually converge to a certain probability distribution and traverse its entirely if sufficient sample steps are taken [23]. It is easily explicable by a physical phenomenon. If we poured a bottle of ice-cold water into a large bowl of boiling water, the water in that bowl would eventually cool to a certain temperature and remain in that state forever, energy loss through the air notwithstanding. Thus, the mathematical term ergodicity essentially describes this deterministic destination to which the system would converge.

2.7 Bayesian Learning

In [34], Radford Neal stated a concise and precise introduction of Bayesian learning: "I view neural networks as probabilistic models, and learning as statistical inference. Conventional network learning finds a single 'optimal' set of parameter values, corresponding to maximum likelihood or maximum penalized likelihood inference. Bayesian inference instead integrates the predictions of the network over all possible values of the network parameters, weighting each parameter set by its posterior probability in light of the training data."

2.7.1 Probabilistic Neural Network

Consider a network taking a vector of real-valued inputs, $\mathbf{x}$, and producing a real-valued outputs, $\bar{\mathbf{y}}$, perhaps computed using hidden layers. This architecture corresponds to a mapping function, f, with $\bar{\mathbf{y}} = f(\mathbf{x}, \theta)$, where θ is a vector of connection weights $\mathbf{W}$ including biases $\mathbf{b}$. If we assume the observed outputs, $\mathbf{y}$, are equal to $\bar{\mathbf{y}}$ plus Gaussian noise of standard deviation σ, the network defines the conditional probability for an observed output vector given an input vector as follows:

$$P(\mathbf{y} \mid \mathbf{x}, \sigma) \propto \exp(-|\mathbf{y} - f(\mathbf{x}; \theta)|^2/2\sigma^2) \tag{2.19}$$

The assumption of predicted and observed output,

$$\mathbf{y} = \bar{\mathbf{y}} + \mathcal{N}(0, \sigma^2), \tag{2.20}$$

stems from the fact that our training set is limited and unable to provide complete and accurate information on our area of interest. In all honesty, our given data set is always biased. This biased data set would result in a discrepancy between the predicted and observed output. And the difference can be usually measured using Gaussian noise with a unknown standard deviation σ [35].

The probability of the outputs in a training set $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$ given this fixed noise level is therefore

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, \ldots, \mathbf{x}_n, \sigma) \propto \exp\left(-\sum_c |\mathbf{y}_c - f(\mathbf{x}_c; \theta)|^2 / 2\sigma^2\right) \tag{2.21}$$

where we assume that all data points $(\mathbf{x}_c, \mathbf{y}_c)$ are independent with each other.

2.7.2 Bayesian Learning and Prediction

Bayesian learning, unlike conventional training, does not seek a single 'optimal' set of network weights. Instead, training data are utilised to determine the *posterior* probability distribution over weight vectors. Predictions for future cases are made by averaging the outputs from all possible weight vectors, with each weight vector contributing proportionally to its posterior probability.

To obtain the posterior, a *prior* distribution for weight vectors must first be defined. We could, for instance, assign each weight a Gaussian prior [34] of standard deviation ω.

$$P(\theta) \propto \exp(-|\theta|^2 / 2\omega^2) \tag{2.22}$$

We can then obtain the posterior distribution over weight vectors given the training cases $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$ using Bayes' Theorem:

$$P(\theta \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)) \propto P(\theta) P(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid \mathbf{x}_1, \ldots, \mathbf{x}_n, \sigma) \tag{2.23}$$

According to the given training data, the best prediction for the output vector in an unseen test case with input vector $\mathbf{x}_*$ is

$$\hat{\mathbf{y}}_* = \int f(\mathbf{x}_*, \theta) P(\theta \mid (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))) \, \mathrm{d}\theta \tag{2.24}$$

It is also possible to generate a complete predictive distribution for the outcomes in the test case, characterising the uncertainty in the above prediction. Integrals such as that of Eq. (2.24) are difficult to calculate. David MacKay addressed this problem by approximating the posterior distribution using a Gaussian [27]. As Neal did, we would choose a more intuitive evaluation that is *Monte Carlo methods* for this paper.

If we randomly select weight vectors, $\theta_0, \cdots, \theta_{N-1}$, each distributed according to the posterior, we can approximate the integral of Eq. (2.24) using the mean output of networks with these weights to derive the prediction for a test case:

$$\hat{\mathbf{y}}_* \approx \frac{1}{N} \sum_c f(\mathbf{x}_*, \theta_c) \tag{2.25}$$

Even if the θ_c are dependent, this formula is still valid, however a higher sample size would be required to meet the specified error bound [35]. Such a sample can be found by simulation from a ergodic Markov chain that has the posterior as its stationary distribution [34]. The early part of the chain where the samples are not yet to reach in a stationary distribution should be discarded. And it this unstable part in the beginning of the chain usually is called as "burn-in".

2.8 Neural Network

After a detailed explanation of Bayesian learning, this section will describe some fundamental tools for training a neural network in addition to the Bayesian neural network.

2.8.1 Cross-entropy

In classification problem, *cross-entropy* is a popular and reasonable option for our neural network's loss function. For two probability distributions A and B in a same probability space $\mathcal{X}$, the cross-entropy of them is defined as

$$\ell_{CE}(A, B) = -\mathbb{E}_A [\log B] . \tag{2.26}$$

It is originally derived from the distance between the trained model's output distribution $P(\mathbf{y}|\mathbf{x}; \theta)$ and true output distribution $P^*(\mathbf{y}|\mathbf{x})$. The closer the obtained distribution is to the true distribution, according to the classical optimization, the more accurate the model's predictions will be. Then our model should be trained to minimize the distance of these two distributions. A common-used distance measurement of two probability distribution is *Kullback-Leibler(KL) divergence* which is developed from relative entropy in information theory [16].

For discrete probability distributions A and B defined on same probability space $\mathcal{X}$, the KL-divergence from A to B is defined to be

$$\mathrm{KL}(A||B) = \mathbb{E}_A \left[\log \frac{A(x)}{B(x)} \right] = \sum_{x \in \mathcal{X}} \left(A(x) \log \frac{A(x)}{B(x)} \right) . \tag{2.27}$$

Substituting our obtained output distribution and true output distribution into Eq. (2.27), we would have

$$\begin{aligned} \mathrm{KL}(P^*(\mathbf{y}|\mathbf{x})||P(\mathbf{y}|\mathbf{x}; \theta)) &= \sum_{\mathbf{y}} P^*(\mathbf{y}|\mathbf{x}) \log \frac{P^*(\mathbf{y}|\mathbf{x})}{P(\mathbf{y}|\mathbf{x}; \theta)} \\ &= \sum_{\mathbf{y}} P^*(\mathbf{y}|\mathbf{x}) \left[\log P^*(\mathbf{y}|\mathbf{x}) - \log P(\mathbf{y}|\mathbf{x}; \theta) \right] \\ &= \underbrace{\sum_{\mathbf{y}} P^*(\mathbf{y}|\mathbf{x}) \log P^*(\mathbf{y}|\mathbf{x})}_{\text{entropy}} \underbrace{- \sum_{\mathbf{y}} P^*(\mathbf{y}|\mathbf{x}) \log P(\mathbf{y}|\mathbf{x}; \theta)}_{cross-entropy} . \end{aligned}$$

Without loss of generality, the true output distribution $P^*(\mathbf{y}|\mathbf{x})$ is assumed to be a fixed distribution. Because our dataset should be given and fixed. It yields that the entropy of the true distribution should be a constant. Therefore, our KL-divergence completely depends on the last term $-\sum_{\mathbf{y}} P^*(\mathbf{y}|\mathbf{x}) \log P(\mathbf{y}|\mathbf{x}; \theta)$ which is our definition of cross-entropy loss. And minimizing the KL-divergence is equivalent to minimizing the corresponding cross-entropy loss:

2.8.2 Softmax

The *softmax* function, also referred to as softargmax or normalised exponential function, transforms a vector of K real numbers into a probability distribution for K potential outcomes. On the basis of Luce's choice axiom [26], it is frequently used as the final activation function of a neural network to normalise the output of the network to a probability distribution over the predicted output classes.

For a output $\mathbf{y} = [y_1, \ldots, y_K]^T \in \mathbb{R}^K$ from neural network, the softmax normalization procedure is defined as

$$\mathrm{Softmax}(\mathbf{y})_i = \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}} \qquad \text{for } i = 1, \ldots, K. \tag{2.28}$$

Prior to applying softmax, some vector components y_i could be negative or greater than one,

and they might not add up to 1. However, after applying softmax, each component will be in the interval $(0, 1)$, and the components will sum to 1, allowing them to be interpreted as probabilities. Moreover, the larger the input components, the greater the probabilities.

Statistical mechanics can also interpret the softmax function [22]. As mentioned in canonical distribution, each state phase z_i could be probabilistically formulated as follows:

$$\rho_i = Z_{\text{can}}^{-1} e^{-\beta z_i}$$

Intuitively, if we could add up all possible distributions ρ_i, we would obtain the proportion of each state phase z_i corresponding to its probability. Therefore, in statistical mechanics, the softmax function will be

$$\text{Softmax}_{\text{mechanics}}(\mathbf{z})_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}} \qquad \text{for } i = 1, \ldots, K. \tag{2.29}$$

2.9 MNIST

A sizable database of handwritten numbers called the MNIST database (Modified National Institute of Standards and Technology database) is frequently used to train different image processing systems [19]. The database is frequently used for machine learning training and testing. It was made by 're-mixing' samples from the initial datasets from NIST. The digits have been size-normalized and centered in a fixed-size image.

Furthermore, the black and white images from NIST were normalized to fit into a 28×28 pixel bounding box and anti-aliased, which introduced grayscale levels. It means that we do not have to do any feature engineering or data washing process before using it.

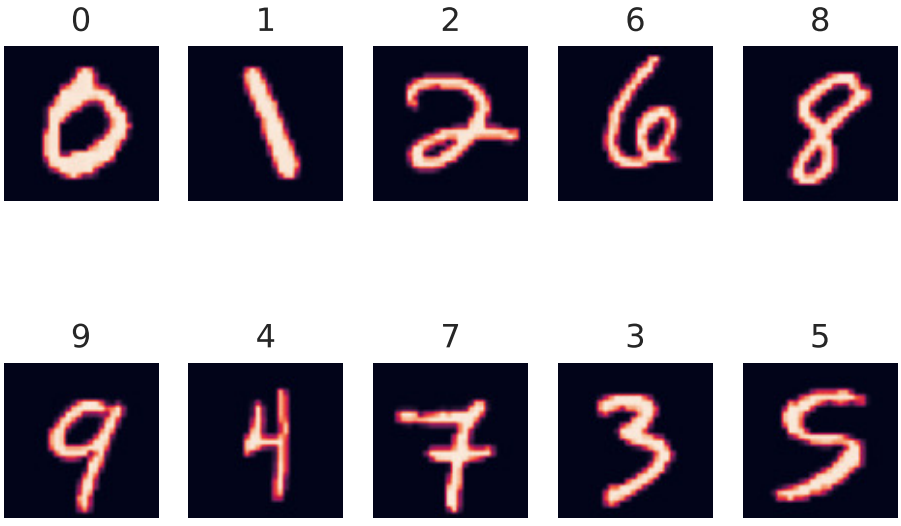

Figure 2: Example images in MNIST data set.

3 Model

In Preliminaries, we presented numerous foundational definitions, ranging from Langevin dynamics to Bayesian learning. We can now connect the dots for a more in-depth discussion of Bayesian neural networks.

3.1 Formulating the Model in term of Potential Energy

In this section, we will look at the formulation of potential energy in Bayesian neural network. First, we will go through the relationship between Empirical risk with posterior. Then a clear explanation and a neat proof on how regularization play a role as prior are given in the second subsection.

3.1.1 Empirical risk and Posterior

First, we consider the general problem of obtaining a sample of (dependent) vectors, q_c, with probabilities given by $P(q)$. Now for Bayesian network learning, position q will be the parameters θ of our network and the distribution of of interest will be the posterior $P(q)$.

From the definition of canonical distribution in Section 2.2, it will be convenient to express this posterior probability distribution in terms of a potential energy function, $U(q)$, chosen so that

$$P(q) \propto \exp(-\beta U(q)) \tag{3.1}$$

From the Bayesian viewpoint, we should try to select a position q (equivalently speaking parameters θ) to maximise the posterior in Eq. (3.1). And this optimization scheme is known as *maximum a-posteriori*(MAP). If we take the logarithm on posterior, we have a more intuitive relation

$$\log(P(q)) \propto -\beta U(q). \tag{3.2}$$

Knowing that the temperature coefficient β should be positive, it is obvious that maximizing posterior is equal to minimizing our potential energy function $U(q)$ from Eq. (3.2).

Therefore, the potential energy function $U(q)$ from the distribution would be the target that our neural network is trained to minimize. The target is usually referred as regularized *empirical risk* $\mathcal{L}$, which typically decomposes into a sum over empirical loss term $\ell(\mathbf{x}, \mathbf{y}; \theta)$ and a regularizer $r(\theta)$. So the empirical risk or potential energy function should be

$$\mathcal{L}(\mathcal{D}; \theta) = r(\theta) + \sum_c \ell(\mathbf{x}_c, \mathbf{y}_c; \theta) \tag{3.3}$$

where $\mathcal{D}$ stands for the dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{c=1}^N$. Hence, we have

$$\theta_{\text{MAP}} = \arg\min_q U(q) = \arg\min_\theta \left(r(\theta) + \sum_c \ell(\mathbf{x}_c, \mathbf{y}_c; \theta) \right). \tag{3.4}$$

3.1.2 Regularization and Prior

Recall the Bayes' theorem Eq. (2.23) in preliminaries section, our posterior $P(q)$ now becomes

$$P(\theta|\mathcal{D}) \propto P(\theta) \prod_{c=1}^N P((\mathbf{x}_c, \mathbf{y}_c)|\theta) \tag{3.5}$$

by assuming that each data points are independent. If we substitute Eq. (3.3) into the potential energy function $U(q)$ in Eq. (3.1) and then compare it with Eq. (3.5), it is not hard to see that prior $P(\theta)$ regularizes the parameters while the likelihood terms constitute the empirical loss $\ell(\mathbf{x}, \mathbf{y}; \theta)$ to be optimized. In particular, when we use normal L_2 regularization as our regularizer

$r(\theta) = ||\theta - 0||_2^2/2$, our prior would be consider as a Gaussian distribution $\mathcal{N}(0, \mathbb{I} \cdot \sqrt{\beta})$. More specifically, by substituting $||\theta||_2^2 + \sum_c \ell(\mathbf{x}_c, \mathbf{y}_c; \theta)$ into Eq. (3.1), we have

$$\begin{aligned} P(q) = P(\theta|\mathcal{D}) &\propto \exp\left(-\beta \left(\frac{||\theta||_2^2}{2} + \sum_c \ell(\mathbf{x}_c, \mathbf{y}_c; \theta)\right)\right) \\ &\propto \exp\left(-\beta \frac{||\theta||_2^2}{2}\right) \cdot \exp\left(-\beta \left(\sum_c \ell(\mathbf{x}_c, \mathbf{y}_c; \theta)\right)\right) \\ &\propto P(\theta) \prod_{c=1}^{N} P((\mathbf{x}_c, \mathbf{y}_c)|\theta). \end{aligned}$$

Therefore, giving a specific L_2 regularization coefficient σ_{re} into $r(\theta) = \sigma_{re}||\theta||_2^2/2$ will help us control the variance of given Gaussian prior $\mathcal{N}(0, \mathbb{I} \cdot \sqrt{\beta\sigma_{re}})$. The choice of σ_{re} would be discussed in experimental section.

3.2 Formulating the Model in term of Hamiltonian

If we consider a full Hamiltonian system in canonical distribution of posterior, it means that momentum p should be introduced. So remind that our energy function will be the sum of potential energy and kinetic energy. $H(q, p) = |p|^2/2 + U(q)$ where mass matrix M is set to be an identity matrix. From the new energy function $H(q, p)$, we define a joint probability distribution over q and p (*phase space*) as follow:

$$P(q, p) \propto \exp(-\beta H(q, p)) \tag{3.6}$$

The marginal distribution for q in Eq. (3.6) is that of Eq. (3.1), from which we wish to sample. Therefore, we can proceed by sampling from this joint distribution for q and p, and then just ignoring the values obtained for p [34]. Good news is introducing momentum p into our energy function would not affect all the results we got in Section 3.1. Because neither posterior $P(\theta|\mathcal{D})$ nor empirical risk $\mathcal{L}(\mathcal{D}; \theta)$ directly depends on it. Therefore, Eq. (3.4) and L_2 regularization representing as Gaussian prior still hold in Hamiltonian energy system. The only impact of momentum is that it would indirectly affect the parameter θ in Langevin dynamics system based on Eq. (2.8). So we could consider momentum p as an auxiliary variable that has no contribution on make predictions but enhances our sampling process for our model. And this enhancement qualities will be thoroughly covered in a subsequent section on interpretation.

Based on the properties we discussed above, introducing Hamiltonian would not destroy any results in Section 3.1.

4 Langevin Dynamics Sampling Optimizer

From Section 3, we differentiate two types of formulations on our objective - posterior with respect to the system's energy (Potential energy or Hamiltonian). Now, we will apply the numerical methods from Section 2.5 on sampling the Langevin paths to obtain the stationary posterior for our Bayesian neural network (BNN).

4.1 Overdamped Case

From discussion in Section 3.1, if we only considered potential energy in our system, it would be a nice example of overdamped Langevin dynamics. In overdamped case, the particles' position is purely based on their given potenital energy function $U(q)$. By simulating the Langevin paths, we would obtain a stationary distribution that is an approximation to and even mathematically converge to the true probability distribution of particles' potential energy [22]. As we discussed in

canonical distribution, the distribution of potential energy could be converted into the posterior which is the objective term we are trying to optimize.

Specifically, a Langevin path could be generated by using Euler-Maruyama method as follows:

$$\theta_{i+1} = \theta_i - h\nabla\mathcal{L}(\mathcal{D};\theta_i) + \sqrt{2\beta^{-1}h}R_n \tag{4.1}$$

where h is the stepsize equivalently learning rate and β stands for the temperature coefficient in Langevin system. This sampling scheme looks very similar with normal gradient descent except for adding a Gaussian noise R_n. A second similarity is that the gradient term $\nabla\mathcal{L}(\mathcal{D};\theta_i)$ can be computed by using backpropagation.

Algorithm 1 : Overdamped Langevin for BNN
Processing by Euler-Maruyama method

Input: learning rate h, temperature coefficient β, training dataset $\mathcal{D}$, empirical risk $\mathcal{L}$
Initialize: θ_i is initialized from $\mathcal{N}(0, \mathbb{I}\cdot\sqrt{\beta\sigma_{re}})$, $i = 0$

Repeat
- $R_g \leftarrow \sqrt{2\beta^{-1}h}\cdot\mathcal{N}(0,\mathbb{I})$ ▷ Gaussian noise
- $g \leftarrow \text{backpropa}(\mathcal{L}(\mathcal{D};\theta_i))$ ▷ Gradient of current θ
- $\theta_{i+1} \leftarrow \theta_i - hg + R_g$ ▷ Euler-Maruyama
- $i \leftarrow i + 1$

Until forever

Leimkuhler-Matthews method is an another generating Langevin path method with high accuracy. According to Eq. (2.12), the sampling procedure for Bayesian neural network is defined to be

$$\theta_{i+1} = \theta_i - h\nabla\mathcal{L}(\mathcal{D};\theta_i) + \sqrt{\frac{\beta^{-1}h}{2}}(R_n + R_{n+1}). \tag{4.2}$$

Instead of using a single Gaussian noise in Euler-Maruyamma method, double Gaussian noise is utilised for illustrating the stochasticity from the random motion here. However, doubling the noise would also increase the stochasticity's standard deviation 2 times. So we need adjust the rescale of the noise coefficient by dividing by 2 in order to maintain the same variance as Langevin dynamics requires.

Algorithm 2 : Overdamped Langevin for BNN
Processing by Leimkuhler-Matthews method

Input: learning rate h, temperature coefficient β, training dataset $\mathcal{D}$, empirical risk $\mathcal{L}$
Initialize: θ_i is initialized from $\mathcal{N}(0, \mathbb{I}\cdot\sqrt{\beta\sigma_{re}})$, $i = 0$

Repeat
- $R_{LM} \leftarrow \sqrt{\beta^{-1}h/2}\cdot\mathcal{N}(0,\mathbb{I})$
- $R_{LM} \leftarrow R_{LM} + \sqrt{\beta^{-1}h/2}\cdot\mathcal{N}(0,\mathbb{I})$ ▷ Leimkuher-Matthews noise
- $g \leftarrow \text{backpropa}(\mathcal{L}(\mathcal{D};\theta_i))$ ▷ Gradient of current θ
- $\theta_{i+1} \leftarrow \theta_i - hg + R_{LM}$ ▷ Leimkuhler-Matthews
- $i \leftarrow i + 1$

Until forever

4.2 Underdamped Case

Actually, we mentioned that overdamped Langevin dynamics is a extreme case of underdamped dynamics. As our discussion in beginning of Section 4.1, the underdamped Langevin path would also preserve the convergence to the true posterior distribution [21]. As we introduce underdamped system in Eq. (2.8), neural network's parameter θ would only be modified through it momentum p. And the noise term goes to momentum instead.

For sampling the Langevin path, "BAOAB" method is an efficient and accurate choice. The steps of it are showed as follows:

$$p_{n+1/2} = p_n - \frac{h}{2}\nabla\mathcal{L}(\mathcal{D};\theta_n) \tag{B}$$

$$\theta_{n+1/2} = \theta_n + \frac{h}{2}p_{n+1/2} \tag{A}$$

$$\hat{p}_{n+1/2} = \exp(-\gamma h)p_{n+1/2} + \sqrt{1/\beta}\sqrt{1-\exp(-2\gamma h)}R_{n+1} \tag{O}$$

$$\theta_{n+1} = \theta_{n+1/2} + \frac{h}{2}\hat{p}_{n+1/2} \tag{A}$$

$$p_{n+1} = \hat{p}_{n+1/2} - \frac{h}{2}\nabla\mathcal{L}(\mathcal{D};\theta_{n+1}) \tag{B}$$

Notice that we initialize the momentum $p_0 = \underline{0}$. Because we should not randomly assign a variable which has a chronic impact on updating our parameter θ. If we assign it a nonzero value but it is in the wrong direction, it would slow down the convergence process or sometimes even cost divergence. Similarly, the computation on the gradient of empirical risk could also be from the normal backpropagation. So the full sampling algorithm is

Algorithm 3 : BAOAB underdamped Langevin for BNN

Input: learning rate h, temperature coefficient β, training dataset $\mathcal{D}$, empirical risk $\mathcal{L}$
Initialize: θ_i is initialized from $\mathcal{N}(0, \mathbb{I}\cdot\sqrt{\beta\sigma_{re}})$, $i = 0, p_i = 0$

Repeat

$R \leftarrow \sqrt{1/\beta}\sqrt{1-\exp(-2\gamma h)}\cdot\mathcal{N}(0,\mathbb{I})$	▷ OU noise
$g_1 \leftarrow \text{backpropa}(\mathcal{L}(\mathcal{D};\theta_i))$	▷ Gradient of θ_i
$p'_i \leftarrow p_i - g_1 h/2$	▷ First B step
$\theta'_i \leftarrow \theta_i + p'_i h/2$	▷ First A step
$p''_i \leftarrow \exp(-\gamma h)p'_i + R$	▷ O step
$\theta_{i+1} \leftarrow \theta'_i + p''_i h/2$	▷ Second A step
$g_2 \leftarrow \text{backpropa}(\mathcal{L}(\mathcal{D};\theta_{i+1}))$	▷ Gradient of θ_{i+1}
$p_{i+1} \leftarrow p''_i - g_2 h/2$	▷ Second B step
$i \leftarrow i+1$	

Until forever

4.3 Interpretation

Before obtaining simulation results, we can still undertake a physics-based performance analysis and comparison on these algorithms. This section, we will discuss the philosophic interpretation on the Langevin dynamics sampler.

4.3.1 Overdamped vs Underdamped

As mentioned in previous sections, the primary distinction between the overdamped and underdamped cases in Langevin dynamics is whether momentum is explicitly used to update position. From Eq. (2.10), if we consider $-\nabla U(q_t)$ to be the gradient of momentum p_t and transfer the noise term to momentum, we get the following result:

$$\mathrm{d}q_t = p_t\mathrm{d}t \tag{4.3a}$$

$$\mathrm{d}p_t = -\nabla U(q_t)\mathrm{d}t + \sqrt{2\gamma\beta^{-1}}\mathrm{d}W_t. \tag{4.3b}$$

Now, overdamped and underdamped system looks really simliar. Based on Eq. (4.3a), next position q_{t+1} is entirely dependent based on the current position q_t and current momentum p_t.

And according to Eq. (4.3b), current momentum p_t is defined by noise and the gradient of current potential energy $\nabla U(q_t)$.

Eq. 2.8 contains an additional friction term for the underdamped case. This friction term is proportional to their momentum. In other words, the faster you drive, the harder brake you would take when you see a speeding camera. The primary effects of defining a friction force subject to momentum would be twofold. Initially, the increment of momentum $\mathrm{d}p_t$ would decrease in comparison to the overdamped case. It follows that the amount of positional change would also decrease. Second, $-\gamma p_t$ will impart inertia to the particle. Because the next momentum is no longer just depend on gradient term $\nabla U(q_t)$ but also on current momentum p_t. Consequently, particles now possess contemporary momentum 'memory'.

In beginning of Section 4, we discussed in detail the equivalence between particle motion and the procedure for obtaining posterior in BNN. Therefore, under our BNN scheme, the underdamped sampling method would have a relatively slow parameter updating scale. In addition, the optimizer would utilise its previous momentum to generate future momentum. Each coin has two facets. These attributes aid the optimizer in obtaining a better grid search of the loss function by directly reducing the increment of momentum. Adding a friction term to the optimizer also forces it to average the gradient of the current empirical risk with the previous momentum. In the meanwhile, underdamped optimizer would relatively slow down the training process.

4.3.2 Why Adding the Noise?

In molecular analysis, the random motion of particles could be mathematically formulated using Gaussian noise. According to Section 2.4, the stochasticity scale is determined by the temperature coefficient. The higher the temperature of the liquid in a real molecular system, the more aggressively the particles move. Therefore, the presence of the noise term in actual molecular dynamics is understandable. However, this raises the question of why we include noise in our optimization process.

As stated in Section 2.7, Bayesian learning does not seek a singular optimal solution. Under our BNN scheme, our optimizer is not only tasked with discovering the parameter that maximises the posterior probability given the training data. It should approximate the posterior distribution. If we do not include the noise term, our optimizer is determined to obtain the optimal solution rather presenting the full posterior distribution. In Section 2.6, we introduced the ergodicity of the Langevin dynamics sampling approach. It indicates that our Langevin optimizer in BNN can theoretically converge and traverse the posterior distribution if it has taken enough steps. Therefore, the stochasticity provided by the system's noise enables the optimizer to visit low-probability regions with a lower probability when compared to high posterior regions.

4.4 Make it More Stochastic

In the preceding sections, we outlined the explicit sampling steps for both underdamped and overdamped Langevin dynamics. As stated in the algorithms, random noises are the source of stochasticity. As a result of these noise setting in Langevin dynamics, we have ergodicity, which tells us that our sampler will converge to equilibrium with any initial condition if sufficient sampling steps are taken. From this perspective, ergodicity prevents these noises from giving our algorithm a truly 'random' appearance. So we can ask, how can we make our algorithm slightly more 'random'?

One intuitive and straightforward method is to introduce randomness into the gradient of $\mathcal{L}(\mathcal{D};\theta)$. Instead of using the entire dataset $\mathcal{D}$ for each sampling step, we could divide $\mathcal{D}$ into small batches $\mathcal{D}_i$ iterate over each of these small batches to calculate the gradient $\nabla\mathcal{L}(\mathcal{D}_i;\theta)$. It has two benefits:

- Using $\mathcal{D}_i$ could reduce the computational cost of gradient $\nabla\mathcal{L}$. This will accelerate our samplers.

- Using a portion of the data's information would compromise the process's determinism. Therefore, it could assist the samplers in escaping the local minimum trap.

5 Bayesian Neural Network for Classification

After such a lengthy and in-depth discussion on the Langevin dynamics sampling method for BNN, we can now construct a neural network to classify two handwritten digits from the MNIST dataset. In this section, we will proceed with the BNN's construction.

5.1 Framework of BNN

According to Section 2.9 introduction to MNIST, each image is stored as a 28-by-28 matrix. Prior to being input into the neural network, the data must be converted to vector format. Therefore, the input layer should have 784 nodes, or $28 \times 28 = 784$. We choose 1200 neurons for the two hidden layers because we need to guarantee that the neural network can at least identify 28-by-28 image. Additionally, it cannot be too complex. If we add more hidden nodes to the model, our neural network should become more certain and able to confidently make predictions. Consequently, it would not be possible to investigate the performance of our Langevin dynamics sampler for uncertainty analysis in later sections. As we will be working on a binary classification problem, the neural network has two outputs that measure the class's likelihood. The network's structure is depicted in Figure 3.

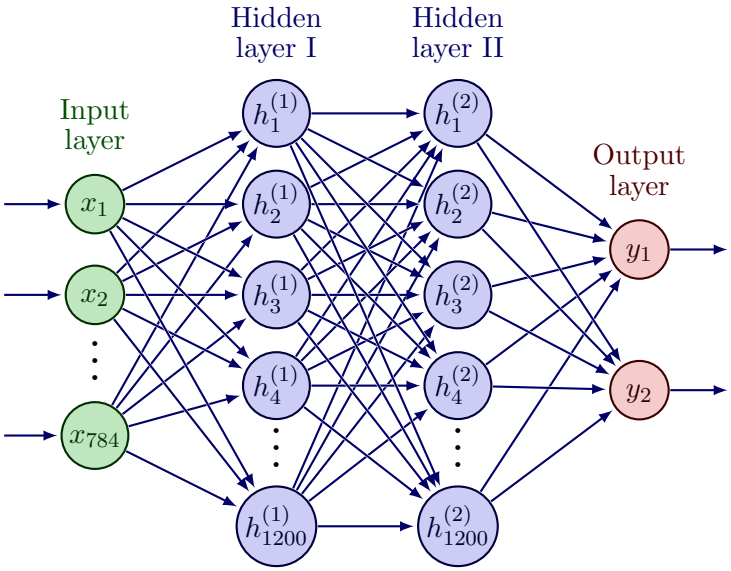

Figure 3: Fully-connected neural network f for binary classification: It has 784 input nodes and 2 hidden layers with 1200 neurons each. The output layer only has two nodes.

5.2 Forward Propagation

In the previous section, we discussed the general architecture of our neural network. We will now discuss the specific setting in it by illustrating forward propagation.

To perform a step in forward propagation, a linear combination must be performed on the current layer's value with respect to the weights $\mathbf{W}$ and the biases $\mathbf{b}$. After adding this linear combination term to the activation function Φ, we will obtain the value for the subsequent layer. We select the ReLu activation function because its gradient is constant. It will aid in improving the accuracy of the backward propagation. A example step is depicted in detail in Figure 4.

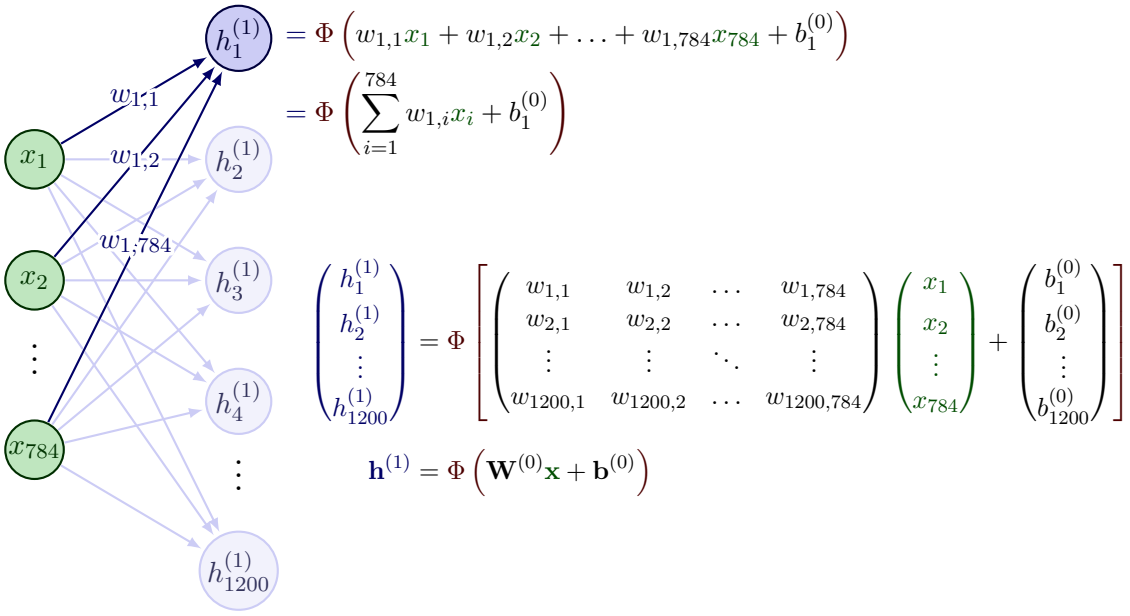

Figure 4: Example forward propagation from input layer to first hidden layer: $\mathbf{x}$ represents a input vector; $\mathbf{W}^{(0)}$ and $\mathbf{b}^{(0)}$ stand for the weight matrix and bias of first layer respectively; Φ corresponds the ReLu activation function; The value in first hidden layer is $\mathbf{h}^{(1)}$.

Each layers has it own weights matrix $\mathbf{W}$ and biases vector $\mathbf{b}$. All of these weights and biases consist of the model parameter θ we mentioned previously. A input $\mathbf{x}$ after a complete forward propagation would produce a 2×1 output vector $\mathbf{y}_{predicted}$ that indicates the distribution of the two classes. Mathematically, it would be

$$\mathbf{y}_{forward} = f_{forward}(\mathbf{x}; \theta), \tag{5.1}$$

where $f_{forward}(\cdot)$ stands for the forward propagation. If we want to convert it in terms of probability, we should apply Softmax function. Our predicted probability distribution $\mathbf{y}_{predicted}$ is defined as

$$\mathbf{y}_{predicted} = f(\mathbf{x}; \theta) = \text{Softmax}(f_{forward}(\mathbf{x}; \theta)). \tag{5.2}$$

5.3 Loss Function

We concluded in Section 3.1.1 that maximising Bayesian posterior $P(\theta|\mathcal{D})$ is equivalent to minimising empirical risk $\mathcal{L}(\mathcal{D}; \theta)$. In neural network terminology, the empirical risk is referred to as the *loss function*. In this section, we will define our loss function mathematically.

The model is expected to solve the classification issue. The predictive distribution output (y_1, y_2) should be close to the true distribution. Cross-entropy loss would be an ideal choice for the empirical loss $\ell(\mathcal{D}; \theta)$, as mentioned in the introduction. $\sigma_{re}||\theta||_2^2/2$ is selected as the regularizer. Due to the fact that, as stated, this L_2 regularisation would execute the 0 mean Gaussian prior with variance $\beta\sigma_{re}$. Therefore, the loss function is defined as follows:

$$\text{Loss} = \mathcal{L}(\mathcal{D}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \ell_{CE}(f(\mathbf{x}; \theta), \mathbf{y}_{true}) + \sigma_{re}||\theta||_2^2/2 \tag{5.3}$$

where σ_{re} should be specified manually. Now, we could put the loss as potential energy into Langevin dynamics sampling scheme.

6 Uncertainty Analysis

In this paper, we will investigate our neural network based on the MNIST dataset. So, what is the degree of uncertainty associated with this classification? According to the structure of our neural network, we will have two output values that correspond to the prediction potential. From Section 2.8.2, we can utilise the softmax function, which can convert these potentials to probabilities. Intuitively, our final step is to predict the scenario with the greater probability. But it raises the question, how confident are you in your prediction? If a scenario's probability is marginally greater than the other one, the neural network will predict it without hesitation. In this circumstance, we can say that this prediction is extremely uncertain.

6.1 Uncertainty Metrics

From the intuitive discussion above, we now have a basic idea on the predictive uncertainty. In this section, we will analyse three approaches to summarise uncertainty within classification: *variation ratios* [8], *predictive entropy* [49], and *mutual information* [49]. These measures capture different notions of uncertainty: *model uncertainty* and *predictive uncertainty*, and will be explained below.

6.1.1 Variation Ratios

The variation ratio is a measure of dispersion—how "spread" the distribution is around the mode. And it is the simplest measure of qualitative variation. In classification scheme, variation ratios is defined in terms of softmax probabilities as follow:

$$\mathcal{V}[\mathbf{x}] := 1 - p(y = c^*|\mathbf{x}, \mathcal{D}_{\text{train}}) \tag{6.1}$$

where $c^* = \underset{c=1,\ldots,C}{\text{argmax}}\, p(y = c|\mathbf{x}, \mathcal{D}_{train})$ and $p(y = c|\mathbf{x}, \mathcal{D}_{train})$ presents the softmax probabilities for a input $\mathbf{x}$. In the binary case, the variation ratio attains its maximum of 0.5 when the two classes are predicted equally likely, and its minimum of 0 when only a single class is predicted.

6.1.2 Predictive Entropy

Unlike variation ratios, predictive entropy has its foundations in information theory. This quantity captures the average amount of information contained in the predictive distribution:

$$\mathcal{H}[y|\mathbf{x}, \mathcal{D}_{\text{train}}] := -\sum_c p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \log p(y = c|\mathbf{x}, \mathcal{D}_{\text{train}}) \tag{6.2}$$

summing over all possible classes c that y can take. Given a test point $\mathbf{x}$, the predictive entropy attains its maximum value when all classes are predicted to have equal uniform probability, and its minimum value of zero when one class has probability 1 and all others probability 0 (i.e. the prediction is certain). Actually, the predictive entropy we define here is identical to the cross-entropy loss function we introduced in the preliminaries. Consequently, the predictive entropy is assumed to be obtained automatically under our neural network scheme.

6.1.3 Mutual Information

As an alternative to the predictive entropy, the mutual information between the prediction y and the posterior over the model parameters ω offers a different measure of uncertainty:

$$\mathcal{M}[y, \omega|\mathbf{x}, \mathcal{D}_{\text{train}}] := \mathcal{H}[y|\mathbf{x}, \mathcal{D}_{\text{train}}] - \mathbb{E}_{p(\omega|\mathcal{D}_{\text{train}})}\left[\mathcal{H}[y|\mathbf{x}, \omega]\right] \tag{6.3}$$

It consists of the predictive entropy and the expectation of predictive entropy with respect to posterior $p(\omega|\mathcal{D}_{\text{train}})$. As we discussed previously, predictive entropy of a given input $\mathbf{x}$ would not be a problem. However, there is no explicit way to compute the expectation part. As mentioned

in Section 2.7.2 regarding Bayesian prediction approximation, Monte Carlo approximation is an ideal option. Similarly, Similarly, we first sample sufficient model parameter $\omega_1, \cdots, \omega_N$ by Langevin dynamics scheme. The approximate expectation would then be

$$\hat{\mathbb{E}}_{p(\omega|\mathcal{D}_{\text{train}})}\left[\mathcal{H}[y|\mathbf{x},\omega]\right] := \frac{1}{N}\sum_{i=1,\cdots,N}\mathcal{H}[y,\omega_i|\mathbf{x},\mathcal{D}_{\text{train}}].$$

Mutual information, unlike the previous two measures, includes information from the posterior model. Consequently, it can be utilised to measure model uncertainty. In addition, the other two methods for measuring predictive uncertainty are considered.

6.2 Objective Selection

To illustrate the effectiveness of our sampling method in terms of gauging uncertainty, our model must handle data that are relatively vague. In particular, the data should share numerous patterns that make it difficult for our neural network to recognise or even confuse the classification model. Based on this principle, our MNIST training objectives should not be overly specific. For instance, the numbers 4 and 5 are undesirable options. Because the order in which we write them and their visual form are significantly different from a recognition perspective. Intuitively, the uncertainty in the prediction of 4 and 5 should not be so great that it requires analysis.

Using the numbers 1 and 7 is a reasonable option for this problem. According to how we write, both consist of a similar downward stroke. The small stroke at the top of the number 1 is easily confused with the rightward stroke from the number 7. Figure 5 depicts some terrible handwritten images of the numbers 1 and 7 that are difficult to distinguish even for humans.

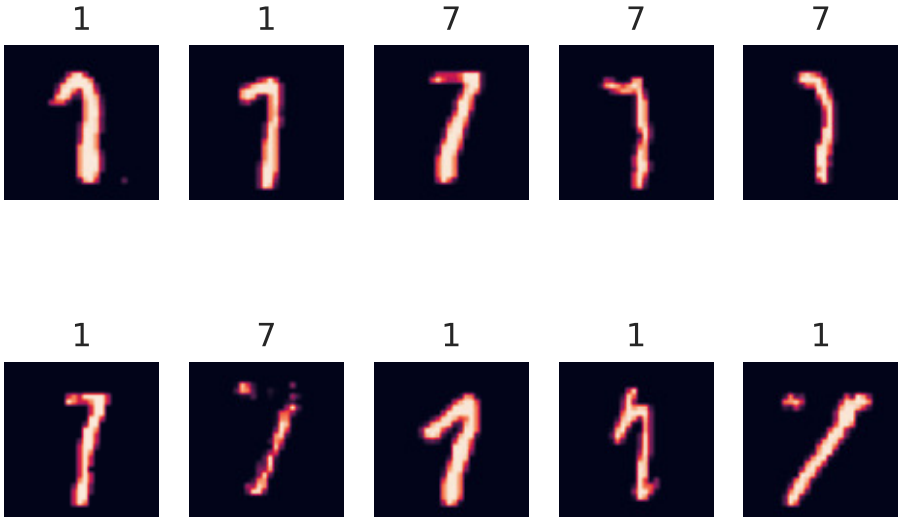

Figure 5: Confusing handwritten digits 1 and 7 in MNIST. The small titles are their true labels.

6.3 Uncertainty in 1 and 7

In the previous section, we explained why we should choose 1 and 7 as our dataset. In this section, we will select images with high uncertainty from the dataset and display their uncertainty metrics.

However, it raises a question. How can we locate the unclear handwritten image? Using a standard algorithm to generate a satisfactory neural network f_O is a practical way. By utilising this initial model, we can compute the output probabilities and uncertainty metrics for each image in the dataset. Based on these probabilities or metrics, we could then obtain the uncertain images.

In accordance with the loss function and neural network structure described in Section 5, we employ Stochastic Gradient Descent (SGD)[1] algorithm to obtain the initial model f_O, specifically the parameter the parameter θ_0. In fact, SGD could provide an excellent classification model for the digits 1 and 7. The excellent performance is provided:

Table 1: The performance of the initial model $f_O(\mathbf{x}; \theta_0)$ in in handwritten digit 1 and 7.

	Training set	Test set
#correct	13004(out of 13007)	2157(out of 2163)
Accuracy	0.999769389	0.9972260594
Loss	3.313247348293	19.767127038

According to Table 1, our initial model performs exceptionally well in classifying the images. However, if we examine the outputs of the neural network which are the probability of each scenario $[p_1, p_7]$, there are a few images that the initial model cannot confidently classify. And we say those with considerable uncertainty.

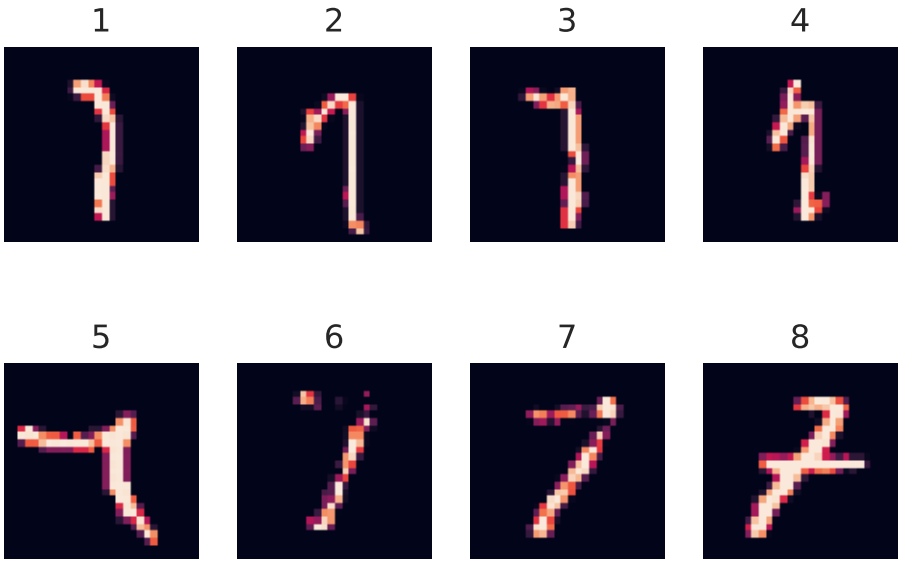

Figure 6: Images 1 and 7 in the MNIST with high levels of uncertainty: It is selected by $|p_1 - p_7| < 0.4$

[1]SGD is an extremely basic and effective machine learning optimization algorithm. Andrew Ng's course provides additional information at `https://www.youtube.com/watch?v=W9iWNJNFzQI&t=686s`

Table 2: Probabilities and uncertainty metrics of high uncertainty images in Figure 6.

	f_O Output				Uncertainty metrics	
Index	p_1	p_7	Label	Index	$\mathcal{V}$	$\mathcal{H}$
1	0.38845268	0.6115473	7	1	0.38845268	0.66805094
2	0.34836814	0.65163183	7	2	0.34836814	0.64643061
3	0.44078043	0.5592195	7	3	0.44078043	0.68611681
4	0.3088064	0.69119364	1	4	0.3088064	0.61814225
5	0.32821783	0.67178214	7	5	0.32821783	0.63290936
6	0.56526613	0.43473384	7	6	0.43473384	0.68460345
7	0.48985422	0.5101457	7	7	0.48985422	0.69294131
8	0.31457564	0.6854244	7	8	0.31457564	0.62271291

Despite the fact that the initial model $f_O(\mathbf{x};\theta_0)$ has remarkable accuracy in both training and test sets, there is significant room for improvement in its classification reliability, especially the handwritten images in Figure 6. In the subsequent sections, we will examine how the Langevin dynamics sampler addresses this issue.

6.4 Uncertainty in Langevin Dynamics Sampler

Remind that the greatest advantage of the Langevin dynamics sampler is its ability to sample the parameter θ from the optimal posterior distribution. Instead of using only one set of sampling parameters θ_0, as SGD does, it uses all sampling parameters $\theta_1, \ldots, \theta_M$, where M represents the number of samples.

As described in Section 2.7.2, the Monte Carlo method in Eq. (2.25) is a reliable way for BNN to utilise the Markov Chain $\theta_1, \ldots, \theta_M$ to make a prediction. However, if we desire a Monte Carlo prediction, we must feed an input image $\mathbf{x}$ into M neural networks. Considering the size of our model, it should incur a substantial amount of computational expense. Instead of averaging the output $f(\mathbf{x};\theta)$, is it possible to simply average the parameter θ so that only one network is required for classification? This paper refers to this method of averaging as *Lazy Monte Carlo* and defines its prediction as follows:

$$f(\mathbf{x}; E(\theta)) \tag{6.4}$$

And prediction from the Monte Carlo is given as:

$$E_\theta(f(\mathbf{x};\theta)) \tag{6.5}$$

In the following two sections, we will demonstrate the difference between these two methods using images with a high level of uncertainty in relation to an underdamped and an overdamped Langevin dynamics sampler.

6.4.1 Underdamped Case

As stated in the preceding section, SGD sets our initial parameter to θ_0 in order to be close to the convergence state. It means that we locate the position of our particle q_0 and that this initial position is comparable to the equilibrium position for a given temperature and friction in underdamped Langevin dynamics. By virtue of the ergodicity of Langevin dynamics, the momentum p_0 could be set to zero. The sampling trajectory is depicted below in terms of loss:

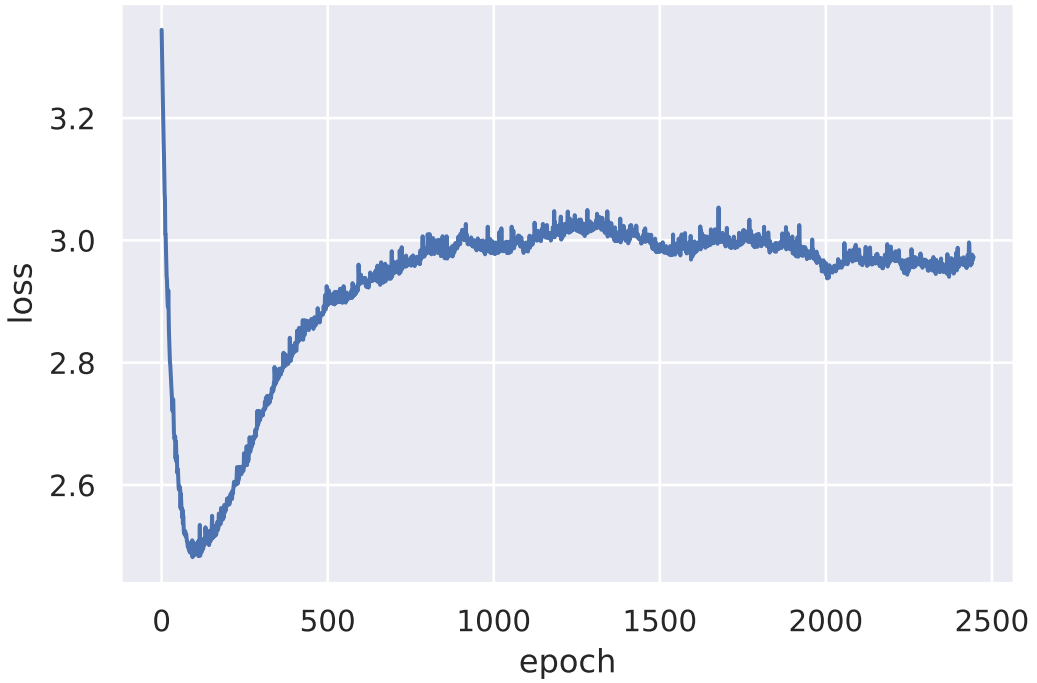

Figure 7: The loss trajectory of BAOAB underdamped Langevin dynamics sampler in traing set: It begins 2447 iterations of sampling from the trained model f_O with an initial momentum of zero and a learning rate of $h = 0.001$. $\beta = 10^5$ and $\gamma = 1$ are the temperature coefficient and friction of the Langevin dynamic system. In regularisation part, the variance of prior is given by $\sigma_{re} = 1/20$.

From the initial 200 sampling steps, the loss of the underdamped sampler decreases until it reaches its minimum Loss ≈ 2.5. This is due to the fact that samplers have zero initial momentum. Specifically, when the sampler's momentum p_0 has not accumulated sufficiently, the update of parameter θ in the BAOAB method relies almost exclusively on the gradient term $\nabla\mathcal{L}(\mathcal{D};\theta)$. Therefore, it functions almost identically to a standard gradient descent, which would always cause our sampler's loss to decrease. After gaining sufficient momentum, the parameter θ would begin to approach equilibrium. The first 500 steps of Figure 7 demonstrate how the equilibrium of momentum p and gradient term $\nabla\mathcal{L}$ affects the trajectory of the samples. Following 800 sampling steps, the trajectory begins to fluctuate around a loss value of 3.0. Therefore, we can assert that the sampling method has converged, and the preceding steps can be regarded as "burn-in".

Next, we would delete the "burn-in" samples and use the remaining samples for Lazy Monte Carlo and standard Monte Carlo evaluations. Mathematically, we have the Lazy Monte Carlo output probability:

$$\begin{bmatrix} p_1 \\ p_7 \end{bmatrix} = f_{under}(\mathbf{x}; E(\theta)) \approx f_{under}\left(\mathbf{x}; \frac{1}{1647}\sum_{i=800}^{2447}\theta_i\right). \tag{6.6}$$

And in regular Monte Carlo method, we have

$$\begin{bmatrix} p_1 \\ p_7 \end{bmatrix} = E_\theta(f_{under}(\mathbf{x};\theta)) \approx \frac{1}{1647}\sum_{i=800}^{2447} f_{under}\left(\mathbf{x};\theta_i\right). \tag{6.7}$$

Using the aforementioned methods, we obtain the following output probabilities and uncertainty metrics for images with high levels of uncertainty:

Table 3: Probabilities and uncertainty metrics for images in Figure 6 predicted by the Lazy and normal Monte Carlo methods in underdamped Langevin dynamics

Probability from Lazy Monte Carlo		
Index	p_1	p_7
1	0.29396936	0.7060306
2	0.14289916	0.85710084
3	0.5672529	0.43274713
4	0.16140981	0.8385902
5	0.1730322	0.82696784
6	0.7880168	0.21198325
7	0.69739366	0.30260634
8	0.23968259	0.76031744

Probability from Monte Carlo		
Index	p_1	p_7
1	0.277988	0.722012
2	0.15032138	0.84967862
3	0.53433503	0.465664971
4	0.16254579	0.83745421
5	0.17864212	0.82135788
6	0.77564071	0.224359291
7	0.69024116	0.309758841
8	0.24035304	0.759646961

Metrics of Lazy Monte Carlo		
Index	$\mathcal{V}$	$\mathcal{H}$
1	0.29396936	0.60566765
2	0.14289916	0.4101916
3	0.43274713	0.68407381
4	0.16140981	0.4420003
5	0.1730322	0.46066168
6	0.21198325	0.51657248
7	0.30260634	0.61305654
8	0.23968259	0.55071378

Metrics of Monte Carlo		
Index	$\mathcal{V}$	$\mathcal{H}$
1	0.277988	0.59104301
2	0.15032138	0.42326615
3	0.465664971	0.69078754
4	0.16254579	0.44386737
5	0.17864212	0.46932821
6	0.224359291	0.53237026
7	0.309758841	0.61890757
8	0.24035304	0.55148653

Comparing the output probabilities of Lazy Monte Carlo and Monte Carlo with those of Table 2 reveals that both methods generally produce forecasts with significantly less uncertainty. Observe that the initial model has an almost 50/50 correct prediction on the third image. Not only do these two methods retain the uncertainty, but they also fail to correctly classify the image. In other words, Langevin sampling does not enhance the cognitive ability of the neural network for this particular image. Additionally, the sixth image is incorrectly classified by the initial. In addition to failing to correct the misclassification made by f_O, both Lazy and regular Monte Carlo increase the probability of the incorrect scenario. In this instance, we can conclude that our underdamped Langevin sampling methods are only able to reduce uncertainty, not correct initial errors. Despite the fact that $f(\mathbf{x}; E(\theta)) \neq E_\theta(f(\mathbf{x}; \theta))$ because neural network f is a non-linear function, the output probabilities from these two Monte Carlo methods are very similar. However, if we consider the uncertainty metrics, the regular Monte Carlo performs marginally better at reducing uncertainty from Table 3.

Now we understand how the inference of Langevin dynamics operates with these ambiguous images. In the section that follows, we will examine an extreme case of Langevin dynamics known as the overdamped case.

6.4.2 Overdamped Case

In contrast to underdamped Langevin dynamics, in the overdamped case the position of particles, p, is only updated by the current position and the given noise. It indicates that we do not need to add additional momentum to our neural network. As we did in underdamped case, we can simulate the overdamped Langevin dynamics using the Euler-Maruyamma method based on the pre-trained parameter θ_0. The sampling trajectory is given as follows:

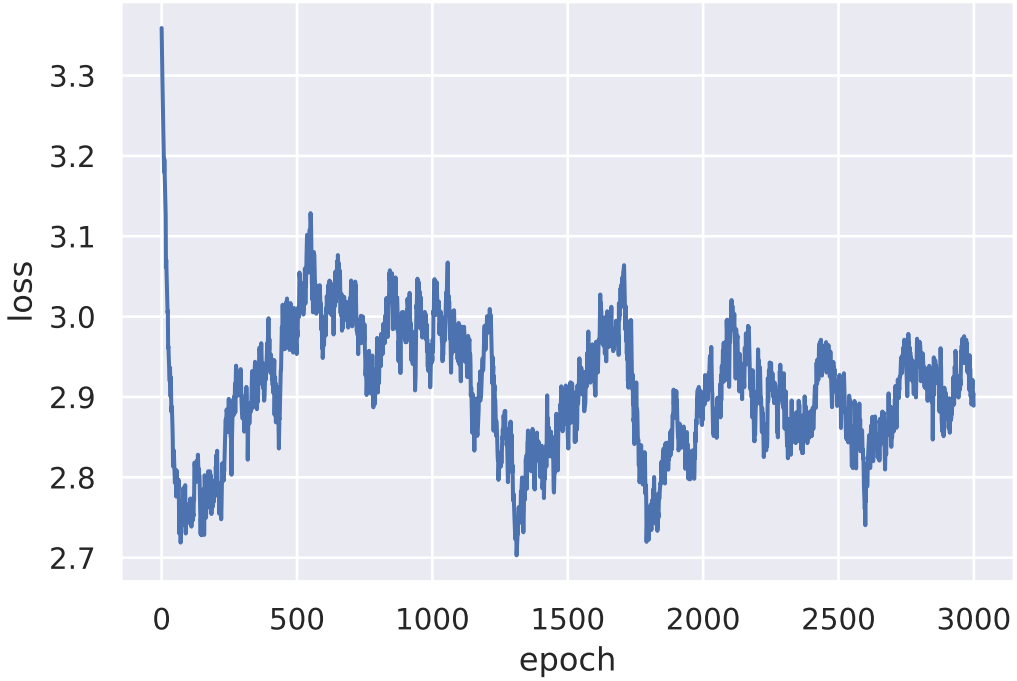

Figure 8: The loss trajectory of Euler-Maruyama overdamped Langevin dynamics sampler in traning set: It begins 3000 iterations of sampling from the trained model f_O with a learning rate of $h = 0.001$. $\beta = 10^5$ is the temperature coefficient of the Langevin dynamic system. In regularisation part, the variance of prior is given by $\sigma_{re} = 1/20$.

Similar to the underdamped case, the overdamped case reaches its minimum Loss ≈ 2.75 in the first 100 steps. After that, it appears to fluctuate between 3.0 and 2.8 for the remainder of the iteration. The reason for this is because the pre-trained parameter θ_0 does not occupy a relatively flat position on the loss function's landscape. The gradient term $\nabla\mathcal{L}(\mathcal{D};\theta_i)$ in Eq. 4.1 has a significant effect on the updating procedure and shields the function from noise at the very beginning of the sampling. Therefore, the trajectory of the first hundred steps resembles a gradient descent. The first hundred steps therefore resemble a gradient descent. Obviously, this deterministic descending path must be eliminated in a subsequent evaluation step. Significantly greater fluctuations are observed in the overdamped case compared to the underdamped case. A straightforward explanation for this is that the system lacks momentum. When the sampler reaches a minimum where it should be relatively flat, the gradient term in Eq. 4.1 has less of an effect on the coming parameter. Consequently, the overdamped path would only be dominated by noise. In the underdamped case, however, the system's inertia would significantly neuralize the stochasticity of the noise.

Next, we would look at the performance of overdamped Langevin dynamics method on high uncertainty images in Fig 6. Again, the evaluation is based on the Lazy Monte Carlo and Monte Carlo that we described for the underdamped case. In particular, the probabilities generated by Lazy Monte Carlo would be

$$\begin{bmatrix} p_1 \\ p_7 \end{bmatrix} = f_{over}(\mathbf{x}; E(\theta)) \approx f_{over}\left(\mathbf{x}; \frac{1}{2900}\sum_{i=100}^{3000}\theta_i\right). \tag{6.8}$$

And in regular Monte Carlo method, we have

$$\begin{bmatrix} p_1 \\ p_7 \end{bmatrix} = E_\theta(f_{over}(\mathbf{x};\theta)) \approx \frac{1}{2900}\sum_{i=100}^{3000} f_{over}\left(\mathbf{x};\theta_i\right). \tag{6.9}$$

Table 4: Probabilities and uncertainty metrics for images in Figure 6 predicted by the Lazy and normal Monte Carlo methods in overdamped Langevin dynamics

Probability from Lazy Monte Carlo		
Index	p_1	p_7
1	0.3046644	0.6953356
2	0.26294407	0.7370559
3	0.5316118	0.46838817
4	0.18740115	0.8125989
5	0.25523955	0.74476045
6	0.8317428	0.16825718
7	0.8417849	0.15821509
8	0.3372505	0.6627495

Probability from Monte Carlo		
Index	p_1	p_7
1	0.2463365	0.7536635
2	0.18081402	0.81918598
3	0.4583018	0.5416982
4	0.14144659	0.85855341
5	0.17791874	0.82208126
6	0.76047515	0.23952485
7	0.73441443	0.26558557
8	0.28266924	0.71733076

Metrics of Lazy Monte Carlo		
Index	$\mathcal{V}$	$\mathcal{H}$
1	0.3046644	0.61476475
2	0.26294407	0.57611382
3	0.46838817	0.69114727
4	0.18740115	0.48243257
5	0.25523955	0.56801856
6	0.16825718	0.45311195
7	0.15821509	0.4366982
8	0.3372505	0.63919485

Metrics of Monte Carlo		
Index	$\mathcal{V}$	$\mathcal{H}$
1	0.2463365	0.55827447
2	0.18081402	0.47262558
3	0.4583018	0.68966566
4	0.14144659	0.40758081
5	0.17791874	0.46822285
6	0.23952485	0.55053161
7	0.26558557	0.57881854
8	0.28266924	0.59545667

In general, both of these Monte Carlo methods for overdamped are capable of reducing uncertainty, whether by correcting or misrepresenting it. The sixth and seventh images reveal an intriguing distinction between Lazy and regular Monte Carlo. Lazy Monte Carlo seems to provide a more certain but incorrect prediction. However, if the prediction is accurate, regular Monte Carlo would provide a more certain prediction than Lazy Monte Carlo. On the other hand, when the prediction is incorrect, Monte Carlo has less faith in the incorrect option than Lazy Monte Carlo. Thus, the regular Monte Carlo method contains an auto-correction scheme. Regular Monte Carlo uses the approximated posterior distribution with respect to the data and neural network, whereas Lazy Monte Carlo does not. This is a simple explanation.

According to the probabilities and uncertain metrics from Table 4 and Table 3, we could conclude that there is little difference between overdamped and underdamped samplers when they are close to the equilibrium state in terms of reducing uncertainty. Both of them can increase the certainty of our predictions. In an underdamped case, however, the Lazy Monte Carlo is not significantly different from the regular Monte Carlo, as determined by a careful examination in Table 3. However, as we showed in Table 4 regular Monte Carlo has a significant advantage over Lazy Monte Carlo in overdamped case.

6.5 Uncertainty and Size of Neural Network

Using the samples from Langevin dynamics could reduce some of the uncertainty in the handwritten images shown in Figure 6, as deduced from the previous two sections. Building a larger

neural network is another common technique for reducing uncertainty. A neural network with more layers or neurons should have at least the same potential performance capability as a network with fewer layers or neurons. Imagine that we fix the extra neurons in the larger network to zero, so that both networks would have the same structure. And adding more neurons creates additional dimensions. Consequently, it is a given that a larger neural network could reduce the uncertainty of the data.

6.5.1 A Bigger Neural Network

In this section, we will first introduce a larger neural network and then demonstrate its performance on the MNIST dataset image digits 1 and 7.

Adding new neurons to the current hidden layer is one way to increase the dimensionality of a neural network. In our discussion, we avoid increasing the neural network's complexity by adding hidden layers because doing so could drastically alter the network's response to uncertainty. For instance, adding a pooling layer to a Convolutional neural network (CNN) could reduce the dimensions of the feature maps, thereby significantly improving its image processing performance. To make f_O comparable, we simply add 300 neurons to each hidden layer, resulting in a neural network F that is slightly more complex. Its framework is depicted in Figure 9.

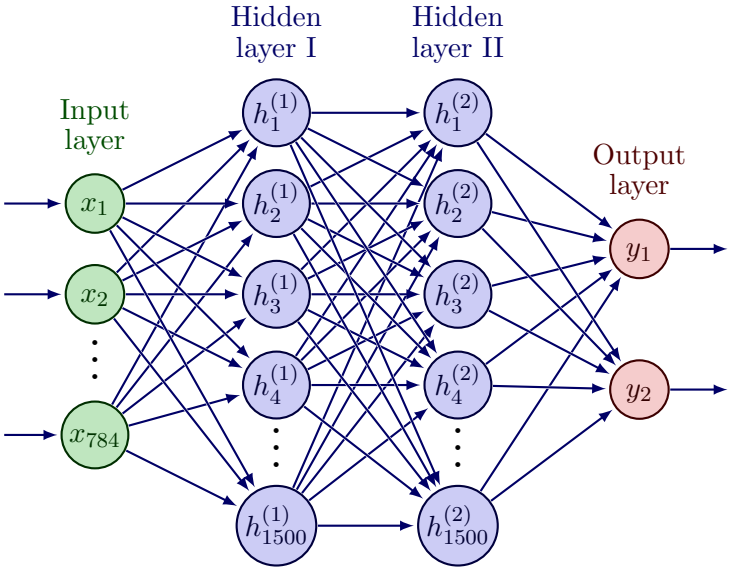

Figure 9: Fully-connected neural network F for binary classification: It has 784 input nodes and 2 hidden layers with 1500 neurons each. The output layer only has two nodes.

To achieve the general performance as f_O, the larger neural network F could be trained using SGD to reach a comparable loss with respect to the same training set. The details are given as follows:

Table 5: The performance of the bigger neural network F with parameter Θ in handwritten digit 1 and 7.

	Training set	Test set
#correct	13000(out of 13007)	2156(out of 2163)
Accuracy	0.9994618892	0.99676376
Loss	3.248132637370	19.653495525

In this section, a neural network $F(\mathbf{x};\Theta)$ with a loss comparable to $f(\mathbf{x};\theta)$ is presented. We will utilise it as a benchmark for the uncertainty discussion in following sections.

6.5.2 Uncertainty from F

In light of the preceding discussion, we can assume that our larger neural network $F(\Theta)$ will be better able to process images with a high degree of uncertainty. As a result, the high uncertainty images in Figure 6 will serve as our reference. First, the output probabilities and uncertainty metrics of the reference images will be evaluated. Then, we will compare it to the original neural network f_O. And the last step is to compare it to the underdamped and overdamped Langevin dynamics evaluations.

Table 6: Probabilities and uncertainty metrics of high uncertainty images in Figure 6 obtained by $F(\Theta)$.

	F Output				Uncertainty metrics	
Index	p_1	p_7	Label	Index	$\mathcal{V}$	$\mathcal{H}$
1	0.28229815	0.71770191	7	1	0.28229815	0.59511071
2	0.06203696	0.93796307	7	2	0.06203696	0.23253586
3	0.41137058	0.58862942	7	3	0.41137058	0.67735351
4	0.22937694	0.77062303	1	4	0.22937694	0.53852242
5	0.12339623	0.87660378	7	5	0.12339623	0.37363755
6	0.75041813	0.2495819	7	6	0.2495819	0.56187533
7	0.5768109	0.42318904	7	7	0.42318904	0.68130051
8	0.31306905	0.68693089	7	8	0.31306905	0.62153433

Table 6 demonstrates specifically how certain the larger neural network F is with respect to the reference handwritten images. Similar to the smaller neural network $f_O(\theta)$, $F(\Theta)$ could correctly classify the majority of images with the exception of the third, sixth and seventh images under maximum probability scheme. However, based on the output probabilities, $F(\Theta)$ appears to have a greater ability to handle uncertainty, as the difference between $p1$ and $p7$ is generally larger. Specifically for the second image, the probability that a network with fewer neurons will make an incorrect prediction is so high as to reach 34.8% in Table 2. Despite this, $F(\Theta)$ could confidently classify this image as 7 with 93% chance. It demonstrates that adding 300 neurons to each hidden layer reduces uncertainty significantly. This, however, does not help correct the incorrect classification of images 3, 6, and 7. Moreover, $F(\Theta)$ even increases the wrong probabilities so that it is more confident in making incorrect predictions than $f_O(\theta)$. For example, $F(\Theta)$ predicts the fourth image to be 7 with a probability of 0.77, whereas the initial model fO predicts a probability of 0.69. From this, we can conclude that adding more neurons does not always increase the prediction's accuracy in some cases.

Recall the discussions we had regarding the underdamped and overdamped sampling methods for Langevin dynamics. It is an intriguing coincidence that the probabilities predicted using Langevin dynamics schemes and probabilities derived from a more complicated neural network share the same characteristics. It could pose an intuitive question. *Can we use the Langevin dynamics sampling methods to reduce uncertainty rather than expanding the neural network model?*

6.5.3 Langevin Dynamics Sampling with Reducing the Size of Neural Network

Comparing the Langevin dynamics sampling method and increasing the size of the neural network, we discussed in detail in the previous section the performance of reducing the uncertainty of certain fuzzy handwritten images using the Langevin sampling method. How about the ordinary data? Does our sampling method increase the degree of certainty? In this section, we'll examine the general performance of the four models we've constructed and provide a philosophical answer with numerics to the question we posed.

Table 7: The quantity of MNIST test images with six distinct levels of uncertainty for the initial neural network f_O, the larger neural network F, the overdamped sampler f_{over} and the underdamped sampler f_{under}. The test dataset consists of 2163 images.

	The number of images with 6 levels of uncertainty		
Model	$\lvert p_1 - p_7\rvert < 0.7$	$\lvert p_1 - p_7\rvert < 0.8$	$\lvert p_1 - p_7\rvert < 0.9$
f_O	21	26	34
F	16	22	28
f_{over}(Lazy Monte Carlo)	17	20	25
f_{over}(Monte Carlo)	15	19	24
f_{under}(Lazy Monte Carlo)	15	21	24
f_{under}(Monte Carlo)	10	15	20
Model	$\lvert p_1 - p_7\rvert < 0.4$	$\lvert p_1 - p_7\rvert < 0.5$	$\lvert p_1 - p_7\rvert < 0.6$
f_O	8	11	17
F	9	11	15
f_{over}(Lazy Monte Carlo)	8	9	12
f_{over}(Monte Carlo)	8	8	12
f_{under}(Lazy Monte Carlo)	7	9	13
f_{under}(Monte Carlo)	8	8	11

From Table 7, three major conclusions can be drawn. First, F has less uncertain predictions than f_O across all three degrees of uncertainty, consistent with the theoretical assumption made in Section 6.5.1. Second, Lazy Monte Carlo appears weaker than regular Monte Carlo when attempting to push the difference between p_1 and p_7. The final result from the table above is that the obtained underdamped dynamics sampling method performs better on MNIST data than the overdamped case.

According to the analysis in this paper, my answer to the question posed at the end of the previous section is both yes and no. Langevin dynamics scheme applies all sample parameters θ_i to approximate the true posterior distribution $P(\theta|\mathcal{D})$ with respect to the dataset $\mathcal{D}$. Theoretically, if we use this approximate distribution to handle some data that are correlated to $\mathcal{D}$, the sampling method could significantly increase the predictability. Because each θ_i may be adept at handling distinct data subsets. If we just use so-call "best" θ^*, the model may be quite accurate for the majority of data. But when confronting a minority from $\mathcal{D}$ like the images in Figure 6, the 'best' parameter may lack confidence which means it could produce a vague or unclear prediction. Therefore, sampling multiple parameters for prediction would increase the chance that we may find some good parameters to handle the minority.

The aforementioned explanation can be verified by simulating the output probabilities from two Langevin sampling paths, f_{under} and f_{over}, with respect to the second and fourth images in Figure 6, $\mathbf{x}_2$ and $\mathbf{x}_4$.

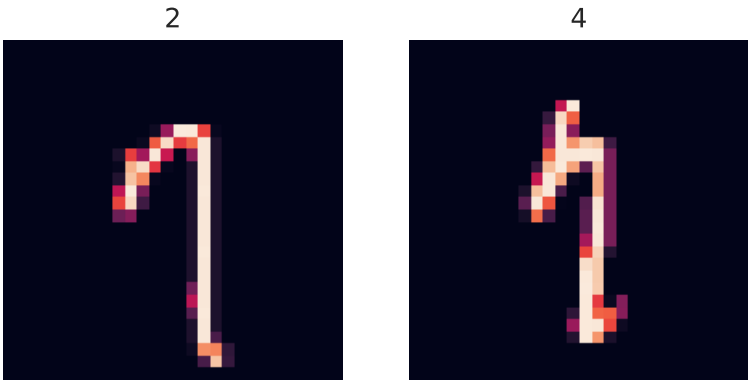

Figure 10: Figure 6's second and fourth images are shown on the left and right, respectively.

Table 8: Some probabilities from underdamped and overdamped Langevin dynamics sampling path with respect to the second and fourth images $\mathbf{x}_2$ and $\mathbf{x}_4$ in Figure 6.

Sampling path $f_{under}(\mathbf{x}_2, \theta_i)$			Sampling path $f_{over}(\mathbf{x}_2, \theta_i)$		
iteration i	p_1	p_7	Iteration i	p_1	p_7
352	0.2036244	0.7963756	511	0.14643753	0.85356247
379	0.1629487	0.8370513	694	0.18184628	0.81815372
630	0.1335918	0.8664082	1243	0.25284634	0.74715366
940	0.1810185	0.8189815	1834	0.09284651	0.90715349
1218	0.2095542	0.7904458	2058	0.17152314	0.82847686
1428	0.1389428	0.8610572	2341	0.20297194	0.79702806
1619	0.1057845	0.8942155	2639	0.10395623	0.89604377
2094	0.2371219	0.7628781	2810	0.16239857	0.83760143
Sampling path $f_{under}(\mathbf{x}_4, \theta_i)$			Sampling path $f_{over}(\mathbf{x}_4, \theta_i)$		
Iteration i	p_1	p_7	Iteration i	p_1	p_7
352	0.1239857	0.8760143	511	0.34863034	0.65136966
379	0.2619443	0.7380557	694	0.20394750	0.79605250
630	0.3801382	0.6198618	1243	0.12731832	0.87268168
940	0.1729374	0.8270626	1834	0.18349479	0.81650521
1218	0.1336181	0.8663819	2058	0.25937925	0.74062075
1428	0.1917238	0.8082762	2341	0.12482784	0.87517216
1619	0.3512971	0.6487029	2639	0.16395477	0.83604523
2094	0.1427842	0.8572158	2810	0.14294743	0.85705257

In particular, θ_{511} is not able to reduce the uncertainty of the fourth image $\mathbf{x}_4$ in overdamped case. However, $f_{over}(\mathbf{x}_2, \theta_{511})$ indicates with 85% certainty that 7 should be the correct classification. Therefore, θ_{511} has a inequivalent uncertainty performance for both $\mathbf{x}_2$ and $\mathbf{x}_4$. It corresponds to the fact that each sampled θ_i may be effective at various images. Using a particular one would only improve the prediction of a single instance, but not the overall performance.

Comparing with the results from Table 4 and Table 3, we can conclude that the neural network model could simultaneously reduce the uncertainty of many instances by sampling various parameters and evaluating through Monte Carlo.

However, Lazy Monte Carlo also employs multiple parameters; but if we compare Table 4 and Table 3 carefully, why is it inferior to standard Monte Carlo? Because averaging the parameters is identical to generating a new sample using the sampling technique. And if we only use this single sample parameter, it contains no information from the training posterior distribution approximation $P(\theta|\mathcal{D})$. Therefore, it is technically incapable of reducing the uncertainty of data. Therefore, the regular Monte Carlo would be generally better than the Lazy Monte Carlo method.

In a more complex neural network, the 'best' parameter Θ^* would have a good performance on these minority due to the flexibility of the model. And that is why we could make the first conclusion above. Hence, Langvein dynamics sampling method could play a role as the reducing the size of neural network.

No is also one of my answers, because the Langvein dynamics sampling method can only converge to the posterior distribution for the given dataset $\mathcal{D}$. If a instance $\bar{\mathbf{x}}$ is constructed very differently from $\mathcal{D}$, the posterior distribution obtained by the sampling method would not match the "universal" posterior that includes $\bar{\mathbf{x}}$ information. We are unlikely to sample a parameter capable of handling this situation. The obtained samples were just able to capture the specified characteristics from $\mathcal{D}$. Consequently, they apply the partial cognition from $\mathcal{D}$ on the prediction of $\bar{\mathbf{x}}$ as well. Typically, it would result in worse predictions.

To make everything easier, we could consider getting the good model as solving a problem. The Langevin dynamics sampling method could assemble a team of mathematics professors to solve a maths problem defined by the provided training dataset. Each professor has their own specialist or research area. Therefore, it is normal for a powerful group of professors to be equal to or even more powerful than a single expert professor in a corresponding field. This is how sampling method could reduce the neural network's dimensionality. However, when we assign these great mathematicians a political problem, they likely have no idea what is going on. Because of this, the sampling methods were unable to surpass the cognitive boundary, meaning they are incapable of processing completely new data.

7 Conclusions

7.1 What We Have Done

In this paper, we start with listing the common weakness of the deep neural network with respect to uncertainty. And the idea to overcome these problems by Bayesian inference is introduced. Then, we simulate the inference process under a special molecular dynamics system, Langevin dynamics, to make sure it would converge to a equilibrium posterior distribution. In the experimental section, we construct a binary classification based on digits 1 and 7 from MNIST dataset to investigate the predictive uncertainty from the images with respect to the sampling methods. The relationship between reducing the size of neural network and using Langevin dynamics sampling is well discussed at the end of the paper. In conclusion, this paper accomplishes the following:

- provided a comprehensive and intermediate introduction on both Bayesian neural network and Bayesian inference.
- illustrated a intuitive physics background on Langevin dynamics and its application on Bayesian inference,
- rigorously presented and analysed how does the Langevin dynamics sampling method work in Bayesian inference,

- investigated and explained the ability of reducing the uncertainty from three different methods on MNIST dataset.

7.2 What We Have Observed

In the experimental section, we initially simulated two sampling paths for two cases of Langevin dynamics and compared the numerical performance of samplers to that of a larger neural network. The difference in terms of evaluation methods was also discussed. Here are what we observed:

- Lazy Monte Carlo is inferior than regular Monte Carlo in terms of uncertainty,
- overdamped Langevin dynamics sampler seems to have more oscillation than underdamped sampler in the sampling process,
- Langevin dynamics samplers can reduce the neural network's complexity by traversing the posterior,
- Langevin dynamics samplers are ineffective when the input data is distant from the posterior.

7.3 Outlook

Even though numerous advancements in uncertainty quantification by the Langevin sampling method in neural networks have been illustrated in this paper, their implementation in mission- and safety-critical applications remains limited. Here are two potential future paths we could pursue.

- **Look for Ground Truth Uncertainties**
  Current Langevin sampling methods are evaluated empirically, and their performance is highlighted by reasonable and explicable uncertainty values. In general, there is no ground truth uncertainty that could be used for validation. In addition, even though existing methods are calibrated on specific data sets, one cannot simply transfer these results to any other data set due to shifts in data distribution and the fact that many fields can only cover a small fraction of the actual data environment. Therefore, we emphasised that the Langevin sampling method can only yield the posterior distribution of the given data. Our next investigation could involve attempting to overcome this limitation.
- **Vigorously Prove the Evaluation Methods**
  We evaluated the approximate posterior distribution of the MNIST dataset using both Lazy Monte Carlo and standard Monte Carlo. On the basis of numerical results, we provided a concise explanation of the distinction between these two approaches and their impact in terms of uncertainty. However, proving it mathematically will aid researchers in comprehending the Bayesian neural network's architecture.

References

[1] A. Amini, A. Soleimany, S. Karaman, and D. Rus. Spatial uncertainty sampling for end-to-end control. *arXiv preprint arXiv:1805.04829*, 2018.

[2] M. S. Ayhan and P. Berens. Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks. In *Medical Imaging with Deep Learning*, 2018.

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1613–1622, 2015.

[4] J. Choi, D. Chun, H. Kim, and H.-J. Lee. Gaussian yolov3: An accurate and fast object detector using localization uncertainty for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 502–511, 2019.

[5] S. Eggenreich, C. Payer, M. Urschler, and D. Štern. Variational inference and bayesian cnns for uncertainty estimation in multi-factorial bone age prediction. *arXiv preprint arXiv:2002.10819*, 2020.

[6] A. Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der Physik*, 322(8):549–560, 1905.

[7] D. Feng, L. Rosenbaum, and K. Dietmayer. Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3266–3273. IEEE, 2018.

[8] L. G. Freeman. *Elementary Applied Statistics for students in Behavioral Science*. Wiley, New York, 1965.

[9] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[10] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR, 2017.

[11] J. Gibbs. *Elementary principles in statistical mechanics.* C. Scribner, New York, 1902.

[12] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

[13] W. Huang, J. Zhang, and K. Huang. Bootstrap estimated uncertainty of the environment model for model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3870–3877, 2019.

[14] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[15] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

[16] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[17] T. LaBonte, C. Martinez, and S. A. Roberts. We know where we don't know: 3d bayesian cnns for credible geometric uncertainty. *arXiv preprint arXiv:1910.10793*, 2019.

[18] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[20] K. Lee, H. Lee, K. Lee, and J. Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*, 2018.

[21] B. Leimkuhler and C. Matthews. Rational Construction of Stochastic Numerical Methods for Molecular Sampling. *Applied Mathematics Research eXpress*, 2013(1):34–56, 06 2012.

[22] B. Leimkuhler and C. Matthews. *Molecular dynamics : with deterministic and stochastic numerical methods*. Interdisciplinary applied mathematics, 39. Springer, Cham, 2015.

[23] B. Leimkuhler, C. Matthews, and T. Vlaar. Partitioned integrators for thermodynamic parameterization of neural networks. *CoRR*, abs/1908.11843, 2019.

[24] A. Loquercio, M. Segu, and D. Scaramuzza. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, 5(2):3153–3160, 2020.

[25] B. Lötjens, M. Everett, and J. P. How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE, 2019.

[26] R. D. Luce. *Individual choice behavior: a theoretical analysis*. Wiley, New York, 1959.

[27] D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[28] A. Malinin and M. Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pages 7047–7058, 2018.

[29] A. Malinin, B. Mlodozeniec, and M. Gales. Ensemble distribution distillation. In *8th International Conference on Learning Representations*, 2020.

[30] S. Melchionna. Design of quasisymplectic propagators for langevin dynamics. *The Journal of chemical physics*, 127(4):044108, 2007.

[31] J. Mitros and B. Mac Namee. On the validity of bayesian neural networks for uncertainty estimation. *arXiv preprint arXiv:1912.01530*, 2019.

[32] A. Mobiny, H. V. Nguyen, S. Moulik, N. Garg, and C. C. Wu. Dropconnect is effective in modeling uncertainty of bayesian deep networks. *arXiv preprint arXiv:1906.04569*, 2019.

[33] T. Nair, D. Precup, D. L. Arnold, and T. Arbel. Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation. *Medical image analysis*, 59:101557, 2020.

[34] R. Neal. Bayesian learning via stochastic dynamics. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

[35] R. M. Neal. *Bayesian learning for neural networks*. Lecture notes in statistics; 118. Springer, New York, 1996.

[36] V.-L. Nguyen, S. Destercke, and E. Hüllermeier. Epistemic uncertainty sampling. In *International Conference on Discovery Science*, pages 72–86. Springer, 2019.

[37] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, pages 13991–14002, 2019.

[38] G. A. Pavliotis. *Stochastic processes and applications : diffusion processes, the Fokker-Planck and Langevin equations*. Texts in applied mathematics, volume 60. Springer, New York, 2014.

[39] Y. Pomeau and J. Piasecki. The langevin equation. *Comptes Rendus Physique*, 18(9):570–582, 2017. Science in the making: The Comptes rendus de l'Académie des sciences throughout history.

[40] M. Raghu, K. Blumer, R. Sayres, Z. Obermeyer, B. Kleinberg, S. Mullainathan, and J. Kleinberg. Direct uncertainty prediction for medical second opinions. In *International Conference on Machine Learning*, pages 5281–5290. PMLR, 2019.

[41] T. Ramalho and M. Miranda. Density estimation in representation space to predict model uncertainty. In *Engineering Dependable and Secure Machine Learning Systems: Third International Workshop, EDSMLS 2020, New York City, NY, USA, February 7, 2020, Revised Selected Papers*, volume 1272, page 84. Springer Nature, 2020.

[42] M. Rawat, M. Wistuba, and M.-I. Nicolae. Harnessing model uncertainty for detecting adversarial examples. In *NIPS Workshop on Bayesian Deep Learning*, 2017.

[43] J. C. Reinhold, Y. He, S. Han, Y. Chen, D. Gao, J. Lee, J. L. Prince, and A. Carass. Validating uncertainty in medical image translation. In *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, pages 95–98. IEEE, 2020.

[44] A. G. Roy, S. Conjeti, N. Navab, C. Wachinger, A. D. N. Initiative, et al. Bayesian quicknat: Model uncertainty in deep whole-brain segmentation for structure-wise quality control. *NeuroImage*, 195:11–22, 2019.

[45] M. Rußwurm, S. M. Ali, X. X. Zhu, Y. Gal, and M. Körner. Model and data uncertainty for satellite time series forecasting with deep recurrent models. In *2020 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 7025–7028, 2020.

[46] P. Seeböck, J. I. Orlando, T. Schlegl, S. M. Waldstein, H. Bogunović, S. Klimscha, G. Langs, and U. Schmidt-Erfurth. Exploiting epistemic uncertainty of anatomy segmentation for anomaly detection in retinal oct. *IEEE transactions on medical imaging*, 39(1):87–98, 2019.

[47] M. Sensoy, L. Kaplan, and M. Kandemir. Evidential deep learning to quantify classification uncertainty. In *Advances in Neural Information Processing Systems*, pages 3179–3189, 2018.

[48] A. C. Serban, E. Poll, and J. Visser. Adversarial examples-a complete characterisation of the phenomenon. *arXiv preprint arXiv:1810.01185*, 2018.

[49] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(4):623–656, 1948.

[50] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[51] R. D. Skeel and J. A. Izaguirre. An impulse integrator for langevin dynamics. *Molecular Physics*, 100(24):3885–3891, 2002.

[52] L. Smith and Y. Gal. Understanding measures of uncertainty for adversarial example detection. In *Proceesings of the Conference on Uncertainty in Artificial Intelligence*, pages 560–569, 2018.

[53] T. Tsiligkaridis. Information robust dirichlet networks for predictive uncertainty estimation. *arXiv preprint arXiv:1910.04819*, 2019.

[54] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823–841, Sep 1930.

[55] M. Valdenegro-Toro. Deep sub-ensembles for fast uncertainty estimation in image classification. In *Bayesian Deep Learning Workshop at Neural Information Processing Systems 2019*, 2019.

[56] J. Van Amersfoort, L. Smith, Y. W. Teh, and Y. Gal. Uncertainty estimation using a single deep deterministic neural network. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9690–9700. PMLR, 2020.

[57] M. von Smoluchowski. Zur kinetischen theorie der brownschen molekularbewegung und der suspensionen. *Annalen der Physik*, 326(14):756–780, 1906.

[58] Q. Wen, L. Sun, X. Song, J. Gao, X. Wang, and H. Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[59] Y. Wen, D. Tran, and J. Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *8th International Conference on Learning Representations*, 2020.

[60] J. Wenger, H. Kjellström, and R. Triebel. Non-parametric calibration for classification. In *International Conference on Artificial Intelligence and Statistics*, pages 178–190, 2020.

[61] A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4697–4708, 2020.

[62] Q. Wu, H. Li, W. Su, L. Li, and Z. Yu. Quantifying intrinsic uncertainty in classification via deep dirichlet mixture networks. *arXiv preprint arXiv:1906.04450*, 2019.

[63] J. Zhang, B. Kailkhura, and T. Y.-J. Han. Mix-n-match: Ensemble and compositional methods for uncertainty calibration in deep learning. In *International Conference on Machine Learning*, pages 11117–11128. PMLR, 2020.

[64] X. Zhao, Y. Ou, L. Kaplan, F. Chen, and J.-H. Cho. Quantifying classification uncertainty using regularized evidential neural networks. *arXiv preprint arXiv:1910.06864*, 2019.

Appendices

A Code

Listing 1: Neural network in PyTorch

```
class Linear_2L(nn.Module):


    def __init__(self, input_dim, output_dim):
        super(Linear_2L, self).__init__()

        # attribute for counting the times (start with 1)
        self.count = 1

        # number of hidden nodes
        n_hid = 1500

        # assign the attribute
        self.input_dim = input_dim
        self.output_dim = output_dim


        self.fc1 = nn.Linear(input_dim, n_hid)
        self.fc2 = nn.Linear(n_hid, n_hid)
        self.fc3 = nn.Linear(n_hid, output_dim)



        # pick an activation function

        #self.act = nn.Tanh()
        #self.act = nn.Sigmoid()
        self.act = nn.ReLU(inplace=True)
        #self.act = nn.ELU(inplace=True)
        #self.act = nn.SELU(inplace=True)

    def forward(self, x):

        x = x.view(-1, self.input_dim) # view(batch_size, input_dim)
        # -----------------
        x = self.fc1(x)
        # -----------------
        x = self.act(x)
        # -----------------
        x = self.fc2(x)
        # -----------------
        x = self.act(x)
        # -----------------
        x = self.fc3(x)
        # -----------------
        y = self.act(x)

        return y

    def get_parameter(self):
        new_params = {}                                 # create a new dictionary
        for name, params in self.named_parameters():  # loop over all parameters
            new_params[name] = params.clone()          #
        return new_params

    def accumulate(self,NN):
        NN_pars = NN.get_parameter()

        for name,params in self.named_parameters():
```

```
            params.data.copy_(params + NN_pars[name])

        self.count += 1

    def average(self):

        for params in self.parameters():
            params.data.copy_(params/self.count)
        self.count = 1
```

Listing 2: Underdamped Langevin dynamics sampler

```
def A_step( qp , h ):
    q,p = qp

    q = q + h*p

    return [q,p]

def B_step( qp , h, force ):
    q,p = qp

    F = force

    p = p + h*F

    return [q,p]

def O_step( qp , h,gamma, beta ):
    q,p = qp

    alpha = np.exp(-h*gamma)

    R = p.data.new(p.data.size()).normal_(mean=0, std=1)

    # Theoretical definition on O step
    #p = p * alpha + np.sqrt((1-alpha*alpha)/beta) * R

    # Hokseson's way
    # (directly control the amount of noise on momentum)
    p = p*0.999 + 0.00001*R


    return [q,p]

from torch.optim.optimizer import Optimizer, required

# Based on Underdamped Langevin Equation
class ULD(Optimizer):
    """
    ULD optimiser based on pytorch's SGD.
    Note that the weight decay is specified in terms of the gaussian prior sigma

    We use BAOAB method for solving the Langevin path
    """

    def __init__(self, params, lr=required, norm_sigma=0, addnoise=True,
    momentum_list = required, friction = required, beta = required ,fh_sh =
    required):

        # setup the weight_decay coefficient
        # equivalent to gaussian prior
        weight_decay = 1/(norm_sigma**2)


        # Some warning notification
```

```
        if weight_decay < 0.0:
            raise ValueError("Invalid weight_decay value: {}".format(
    weight_decay))
        if lr is not required and lr < 0.0:
            raise ValueError("Invalid learning rate: {}".format(lr))


        # Add self's group
        defaults = dict(lr=lr, weight_decay=weight_decay, addnoise=addnoise,
    friction=friction, beta = beta,momentum_list=momentum_list,fh_sh=fh_sh)

        super(ULD, self).__init__(params, defaults)


    def step(self):
        """
        Performs a single optimization step.
        """
        loss = None

        for group in self.param_groups:


            # Assign the constant variable
            weight_decay = group['weight_decay']

            lr = group["lr"]

            beta = group["beta"]

            friction = group["friction"]

            momentum_list = group["momentum_list"]

            fh_sh = group["fh_sh"]




            # Loop over the parameter from layers to layers work on parameter

            # For example layer1 weights -> layer1 bias -> layer2 weights ->
    ......
            for idx,p in enumerate(group['params']):
                if p.grad is None:
                    continue

                # the gradient: d_p
                d_p = p.grad.data

                """
                It comes from the [Loss = MSE(y_hat, y) + weight_decay * sum(w
    ^2)] regularisation

                If we take the derivative with respect to the parameter w on the
     right handside
                """
                # d_p         d_p + weight_decay * p.data * 2
                if weight_decay != 0:
                    d_p.add_(weight_decay, p.data)



                """
                Underdamped Langevin Dynamics

```

```
                Sampling method : BAOAB method

                ATTENTION : We need to compute the gradient of the new parameter in the second B step
                            So we need half the BAOAB process to compute the gradient by back propagation
                            First half we do BAO fh_sh=0
                            Second half we only do AB fh_sh=1
                """
                qp = [p.data, momentum_list[idx]]


                if fh_sh[idx] == 0:
                  #first half
                  #B
                  qp_B = B_step(qp=qp,h=lr/2,force = -d_p)

                  #A
                  qp_BA = A_step(qp=qp_B,h=lr/2)

                  #O
                  qp_end = O_step(qp=qp_BA,h=lr,beta=beta,gamma=friction)
                  fh_sh[idx] = 1
                else:
                  #second half
                  #A
                  qp_A = A_step(qp=qp,h=lr/2)

                  #B
                  qp_end = B_step(qp=qp,h=lr/2,force = -d_p)
                  fh_sh[idx] = 0

                # update parameter and momentum
                p.data = qp_end[0]
                momentum_list[idx] = qp_end[1]




        return loss
```

Listing 3: Overdamped Langevin dynamics sampler

```
from torch.optim.optimizer import Optimizer, required

# Just add the Langevin noise
class SGLD(Optimizer):
    """
    SGLD optimiser based on pytorch's SGD.
    Note that the weight decay is specified in terms of the gaussian prior sigma
    """

    def __init__(self, params, lr=required, norm_sigma=0, addnoise=True):

        weight_decay = 1/(norm_sigma**2)

        # Some warning notice
        if weight_decay < 0.0:
            raise ValueError("Invalid weight_decay value: {}".format(weight_decay))
        if lr is not required and lr < 0.0:
            raise ValueError("Invalid learning rate: {}".format(lr))

        # Add self's group
        defaults = dict(lr=lr, weight_decay=weight_decay, addnoise=addnoise)
```

```
22
23          super(SGLD, self).__init__(params, defaults)
24
25      def step(self):
26          """
27          Performs a single optimization step.
28          """
29          loss = None
30
31          for group in self.param_groups:
32
33              weight_decay = group['weight_decay']
34
35
36              # Loop over the parameter from layers to layers work on parameter
37              # For example layer1 weights -> layer1 bias -> layer2 weights ->
    ......
38              for p in group['params']:
39                  if p.grad is None:
40                      continue
41
42                  # the gradient: d_p
43                  d_p = p.grad.data
44
45                  """
46                  It comes from the [Loss = MSE(y_hat, y) + weight_decay * sum(w
    ^2)] regularisation
47
48                  If we take the derivative with respect to the parameter w on the
     right handside
49                  """
50                  # d_p         d_p + weight_decay * p.data * 2
51                  if weight_decay != 0:
52                      d_p.add_(weight_decay, p.data)
53
54                  # Add the noise
55                  if group['addnoise']:
56                      # construct the noise
57                      # Create a new tensor with the same size.
58                      # Turn it as a Gaussian noise
59                      langevin_noise = p.data.new(p.data.size()).normal_(mean=0,
    std=1)*np.sqrt(group['lr'])
60
61                      # (-group["lr"] times 0.5*d_p + langevin_noise) plus p
62                      p.data.add_(-group['lr'],0.5*d_p + langevin_noise)
63                  else:
64                      p.data.add_(-group['lr'], 0.5*d_p)
65
66          return loss
```