Likelihood Estimation Comparison for Energy Based Models: A Framework

Manuel Günther

March 27, 2024

Contents

1 Introduction

Energy based models are widely used in statistical learning and can also be found as elegant models of spatial interactions in point process models. However, maximum likelihood (ML) based inference usually remains challenging due to the difficulty in approximating the partition function using Markov Chain Monte Carlo (MCMC) for high- dimensional multimodal distributions.
Following the work of Gao and Song ([**?**]), the goal of this project is to explore the efficacy and efficiency of recovery likelihood estimation in learning parametrised models. The RL approach is based on applying Gaussian noise to the training data and subsequently using the conditional instead of the marginal distribution. We will shown that this approach yields an unbiased estimator of the underlying parameters and investigate the MCMC sampling properties.
As the inference process depends on many components and hyperparameters, comparing these estimators empirically is necessarily not comprehensive. Depending on the training data, the model used, the sampling approach and components like the optimiser, the results might be different. Thus a large part of this project was dedicated to develop a general code framework that allows for more systematic testing in a setting of interest.
Gao and Song have shown that RL produces good results and computational savings in the context of image generation. In that context and many others the models are typically some variety of neural networks. For neural network architectures the concrete values of the weights and biases are less significant, and two architectures with fundamentally different parameter values can still produce similar distributions. In this project the emphasis is on parametric models whose parameters carry a direct meaning or function as the basis for further analysis. For such models it is not just important that the resulting distribution coincides with the data distribution, but also that the parameters themselves are estimated accurately. The later comparisons will hence focus primarily on how well the estimated parameters match the distribution parameters.

1.1 Problem Formulation

The problem that is investigated in statistical learning looks like this: We have a dataset $\mathcal{D} = (\boldsymbol{x}_i)_{i=1}^n \subseteq \mathbb{R}^d$ from an unknown distribution of a random vector $\boldsymbol{X} : \Omega \to \mathbb{R}^d$, with some probability density function $p : \mathbb{R}^d \to \mathbb{R}$. We want to use the data to approximate this density function with a parametrised model density function $p_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}$, for a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^m$, by estimating an optimal $\hat{\boldsymbol{\theta}}$ such that $p_{\hat{\boldsymbol{\theta}}}$ is close to p.

1.2 Approaches

For high dimensional multimodal problems and complicated target densities p estimating $\hat{\boldsymbol{\theta}}$ is a highly non-trivial task, where the standard ML inference can be computationally infeasible. Complicating the problem further is that the model distributions need to be sufficiently complex to be able to resemble the target distribution and thus even the model density is often only determined by its probability kernel, as the normalisation constant is not available analytically and costly to calculate numerically. There are various approaches that attempt to make this inference problem tractable, such as

2 Energy Based Models

For a function $U_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}$, also called (potential) energy functional, an energy-based model (EBM) is defined as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{\theta}}} \exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

where $Z_{\boldsymbol{\theta}} = \int \exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x}$ is the partition function which is analytically intractable for high dimensional $\boldsymbol{x}$.

2.1 Transformation

Any standard distribution for a random variable or vector $\boldsymbol{X} : \Omega \to \mathbb{R}^d$ with a continuous density $f_{\boldsymbol{X}}$ can be transformed into an EBM. For this, an analytical representation of the probability kernel $K_{\boldsymbol{X}}$ of the distribution, i.e. the un-normalised version of the density, is already sufficient, making it an excellent tool for Bayesian models where the normalisation constant of the posterior is typically analytically unavailable or intractable. Given the kernel we can set

$$U(\boldsymbol{x}) = -\log K_{\boldsymbol{X}}(\boldsymbol{x}),$$

which reproduces the desired density when substituted in the EBMs density form:

$$\begin{aligned} p(\boldsymbol{x}) &= \frac{1}{Z} \exp(-U(\boldsymbol{x})) \\ &= \frac{1}{\int \exp(-U(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x}} \exp(-U(\boldsymbol{x})) \\ &= \frac{1}{\int \exp(\log K_{\boldsymbol{X}}(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x}} \exp(\log K_{\boldsymbol{X}}(\boldsymbol{x})) \\ &= \frac{1}{\int K_{\boldsymbol{X}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}} K_{\boldsymbol{X}}(\boldsymbol{x}) \\ &= f_{\boldsymbol{X}}(\boldsymbol{x}) \end{aligned}$$

2.2 Differentiating

For sampling and estimation we will have to differentiate the energy functional of a model by both the input $\boldsymbol{x}$ and by the parameter vector $\boldsymbol{\theta}$.
In general, for a function $f : \mathbb{R}^d \to \mathbb{R}$ we have by the chain rule that

$$\nabla_{\boldsymbol{x}}(-\log f(\boldsymbol{x})) = -\frac{1}{f(\boldsymbol{x})} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = -\frac{\nabla_{\boldsymbol{x}} f(\boldsymbol{x})}{f(\boldsymbol{x})}$$

so for a model were we have the kernel analytically the derivative with respect to $\boldsymbol{x}$ is

$$\begin{aligned} \nabla_{\boldsymbol{x}} U_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{x}}(-\log K_{\boldsymbol{\theta}}(\boldsymbol{x})) \\ &= -\frac{\nabla_{\boldsymbol{x}} K_{\boldsymbol{\theta}}(\boldsymbol{x})}{K_{\boldsymbol{\theta}}(\boldsymbol{x})} \end{aligned}$$

and analogously for $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} U_{\boldsymbol{\theta}}(\boldsymbol{x}) = -\frac{\nabla_{\boldsymbol{\theta}} K_{\boldsymbol{\theta}}(\boldsymbol{x})}{K_{\boldsymbol{\theta}}(\boldsymbol{x})}.$$

3 Test Distributions

The models corresponding to the distributions, that are introduced in the following, will be explored in the experiments in the results section. This is just a small selection, that will not cover many interesting questions or potential test cases, so the reader is invited to customise the framework, introduced later in this report, by adding a new model they consider interesting. In the following, let $X : \Omega \to \mathbb{R}$ or $\boldsymbol{X} : \Omega \to \mathbb{R}^d$ be a random variable or a random vector respectively, with the respective distribution.

3.1 Multivariate Normal Distribution

The multivariate normal distribution is a great place to start as it is ubiquitous, easy to handle and serves as a base component for many complex models and algorithms. For mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance $\Sigma \in \mathbb{R}^{d \times d}$ we say X follows a multivariate Gaussian or normal distribution $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, if its probability density function is given by:

$$f_X(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

From this density one can obtain several components used later in sampling and estimation: **Kernel:**

$$K_X(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

Energy Function:

$$\begin{aligned} U(\boldsymbol{x}) &= -\log(K_X(\boldsymbol{x})) \\ &= -\log(\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)) \\ &= \left(\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \end{aligned}$$

Gradient of Energy Function with respect to $\boldsymbol{x}$:

$$\begin{aligned} \nabla U(\boldsymbol{x}) &= \nabla\left(\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \\ &= \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) \end{aligned}$$

Gradient of Energy Function with respect to $\boldsymbol{\mu}$:

$$\begin{aligned} \nabla_{\boldsymbol{\mu}} U(\boldsymbol{x}) &= \nabla_{\mu}\left(\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \\ &= \Sigma^{-1}(\boldsymbol{\mu}-\boldsymbol{x}) \end{aligned}$$

The partial derivatives of the energy functional with respect to $\Sigma_{i,j}$ however are not as easily obtainable analytically, so the gradient with respect to Σ is obtained instead with automatic differentiation, that is leveraged by the employed framework.

3.2 Gaussian Mixture Distribution (GMD)

Gaussian mixture distributions make for simple but sufficiently complex distribution for our parameter estimation testing process. A GMD is a random vector $\boldsymbol{X} : \Omega \to \mathbb{R}^d$, whose density is the weighted sum of the densities $f_i : \mathbb{R}^d \to \mathbb{R}$, of m Gaussian random vectors $\boldsymbol{Z}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$ with corresponding weights $w_i \in (0, 1)$ such that $\sum_{i=1}^m w_i$. The density is given by

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) := \sum_{i=1}^{m} w_i f_i(\boldsymbol{x})$$

where the weights, means and covariances are aggregated in the parameter $\boldsymbol{\theta}$ for convenience. For each $i \in [m]$ the density f_i can be split in two parts, namely the kernel, which is denoted as K_i, and the normalisation constant Z_i, i.e.

$$f_i(\boldsymbol{x}) = \underbrace{\frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}}}_{=:Z_i} \underbrace{\exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\boldsymbol{x} - \boldsymbol{\mu_i})\right)}_{=:K_i(\boldsymbol{x})}$$

and with these one can rewrite the density as

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \sum_{i=1}^{m} w_i \frac{K_i(\boldsymbol{x})}{Z_i} \\ &= \sum_{i=1}^{m} \frac{w_i (\prod_{j \neq i} Z_j) K_i(\boldsymbol{x})}{\prod_{j=1}^{m} Z_j} \\ &= \frac{\sum_{i=1}^{m} w_i (\prod_{j \neq i} Z_j) K_i(\boldsymbol{x})}{\prod_{j=1}^{m} Z_j} \end{aligned}$$

By defining

$$K_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{m} w_i (\prod_{j \neq i} Z_j) K_i(\boldsymbol{x})$$

we obtain this expression for the energy $U_{\boldsymbol{\theta}}$:

$$\begin{aligned} U_{\boldsymbol{\theta}}(\boldsymbol{x}) &= -\log(K_{\boldsymbol{\theta}}(\boldsymbol{x})) \\ &= \log(\frac{1}{\sum_{i=1}^{m} w_i (\prod_{j \neq i} Z_j) K_i(\boldsymbol{x})}) \end{aligned}$$

Here neither the gradient of $U_{\boldsymbol{\theta}}$ w.r.t. the inputs $\boldsymbol{x}$ nor w.r.t. to the parameters $\boldsymbol{\theta}$ is simple to calculate, and hence the framework leverages the automatic differentiation capacity of `pytorch` for both of these.

3.3 Univariate Polynomial Energy

A polynomial can serve as an energy functional for an EBM and here, instead of deriving the energy from the density, the distribution is defined via the energy directly. This is also the more common approach in practice, where typically an energy is defined to create a distribution. Let $m \in \mathbb{N}$ and $\forall i \in [m] : w_i \in \mathbb{R}$ and for $x \in \mathbb{R}$ define the energy functional as

$$U_{\boldsymbol{w}}(x) := \sum_{i=1}^{m} w_i x^i.$$

Then the density is given by

$$p_{\boldsymbol{w}}(x) := \frac{\exp(-\sum_{i=1}^{m} w_i x^i)}{Z_{\boldsymbol{w}}}$$

where $Z_{\boldsymbol{w}}$ can, in general, only be approximated numerically. In this case it is easy to derive the analytical gradients of $U_{\boldsymbol{w}}$ directly.
Gradient of $U_{\boldsymbol{w}}$ with respect to x:

$$\frac{\mathrm{d}}{\mathrm{d}x} U_{\boldsymbol{w}}(x) := \sum_{i=1}^{m} i w_i x^{i-1}$$

Gradient of $U_{\boldsymbol{w}}$ with respect to $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}} U_{\boldsymbol{w}}(x) := \sum_{i=1}^{m} \boldsymbol{e}_i x^i$$

where $\boldsymbol{e}_i = (\delta_{i,j})_{j=1}^{m}$ is the ith unit vector in $\mathbb{R}^m$.

4 Sampling Algorithms

4.1 Metropolis-Hastings

Let $\tilde{p} : \mathbb{R}^d \to \mathbb{R}$ be the unnormalised probability density function, of the density

$$p(\boldsymbol{x}) := \frac{\tilde{p}(\boldsymbol{x})}{\int_\Theta \tilde{p}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}}$$

with a potentially intractable normalisation constant $\int_\Theta \tilde{p}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$.

Metropolis-Hastings

```
Initialise $\boldsymbol{x}_0$
for $k \leq n$ do
    sample $\hat{\boldsymbol{x}} \sim q(\hat{\boldsymbol{x}}|\boldsymbol{x}_k)$
    compute $\alpha(\boldsymbol{x}_k, \hat{\boldsymbol{x}}) = \min\{1, \frac{\tilde{p}(\hat{\boldsymbol{x}}) q(\boldsymbol{x}_k|\hat{\boldsymbol{x}})}{\tilde{p}(\boldsymbol{x}_k) q(\hat{\boldsymbol{x}}|\boldsymbol{x}_k)}\}$
    sample $u \sim \mathcal{U}(0, 1)$
    if $u \leq \alpha(\boldsymbol{x}_k, \hat{\boldsymbol{x}})$ then
        $\boldsymbol{x}_{k+1} := \hat{\boldsymbol{x}}$
    else
        $\boldsymbol{x}_{k+1} := \boldsymbol{x}_k$
```

We can use the unnormalised $\tilde{p}(\cdot)$ instead of $p(\cdot)$ as the normalisation constant cancels in the fraction. A typical proposal distribution driving the Markov chain is $q(\cdot|\boldsymbol{x}) \sim \mathcal{N}(\cdot|\boldsymbol{x}, \boldsymbol{\Sigma})$, mimicking a random walk in the parameter space. To achieve high acceptance rates one can choose smaller proposed transitions. But this approach increases the traversal time and can become inefficient for high dimensional parameter spaces. This can imply row rates of acceptance, resulting in a highly correlated Markov chain and/or high mixing times. The mixing time refers to the time it takes until the chain reaches the target distribution, while the correlation comes from the chain remaining in the same states when the transition is rejected.

4.2 Langevin Dynamics

4.2.1 Langevin Equation

The Langevin equation describes the evolution of a particle's position in a dissipative medium and is given by:

$$m\frac{d^2x}{dt^2} = -\gamma\frac{dx}{dt} + \sqrt{2kT\gamma}\eta(t)$$

where m is the mass, γ is the friction coefficient, x is the position of the particle, T is the temperature and $\eta(t)$ is a white noise term. The Langevin equation combines deterministic motion with a stochastic term to model the impact of random collisions with surrounding particles in a medium. It's widely used in statistical mechanics to describe the behaviour of particles under the influence of both deterministic and random forces.

4.2.2 Langevin Diffusion

The Langevin diffusion process is a stochastic differential equation (SDE) that represents the continuous-time evolution of a particle's position and is given by:

$$d\boldsymbol{X}_t = -\frac{1}{\gamma}\nabla U(\boldsymbol{X}_t)dt + \sqrt{\frac{2kT}{\gamma}}d\boldsymbol{B}_t$$

where $U(x)$ is the potential energy at position x and $\boldsymbol{B}$ is a Brownian motion. Langevin diffusion describes the random motion of a particle in a potential field, incorporating the effects of both deterministic drift and stochastic noise.

4.2.3 Invariant Measure via Fokker-Planck Equation

Consider a simpler form of the Langevin diffusion SDE:

$$\mathrm{d}\boldsymbol{X}_t = -\nabla U(\boldsymbol{X}_t)\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}\boldsymbol{B}_t$$

where $\gamma = 1$ and $T = k = 1$. The Fokker-Planck Equation for the density of the solution $p(\boldsymbol{x}, t)$ reads

$$\begin{aligned}
\partial_t p(\boldsymbol{x}, t) &= \sum_i \partial_{x_i}\left[\partial_{x_i} U(\boldsymbol{x}) p(\boldsymbol{x}, t)\right] + \sum_i \sum_j \partial_{x_i}\partial_{x_j}\delta_{i,j} p(\boldsymbol{x}, t) \\
&= \sum_i \partial_{x_i}\left[\partial_{x_i} U(\boldsymbol{x}) p(\boldsymbol{x}, t)\right] + \sum_i \partial_{x_i}^2 p(\boldsymbol{x}, t) \\
&= \nabla \cdot (\nabla U(\boldsymbol{x}) p(\boldsymbol{x}, t) + \nabla p(\boldsymbol{x}, t))
\end{aligned}$$

We can use this equation to show that the distribution with density

$$\pi(\boldsymbol{x}) = \frac{1}{\int \exp(-U(\boldsymbol{y}))d\boldsymbol{y}} \exp(-U(\boldsymbol{x})) =: \frac{1}{Z}\exp(-U(\boldsymbol{x}))$$

is an invariant measure of the process defined by this SDE. To see this, consider that the gradient of π with respect to $\boldsymbol{x}$ is given by:

$$\nabla\pi(\boldsymbol{x}) = \frac{1}{Z}\nabla\exp(-U(\boldsymbol{x})) = -\frac{1}{Z}\nabla U(\boldsymbol{x})\exp(-U(\boldsymbol{x})) = -\nabla U(\boldsymbol{x})\pi(\boldsymbol{x})$$

So when we replace $p(\boldsymbol{x}, t)$ on the r.h.s. of the Fokker-Planck equation with $\pi(\boldsymbol{x})$, we have that

$$\begin{aligned}
\partial_t p(\boldsymbol{x}, t) &= \nabla \cdot (\nabla U(\boldsymbol{x})\pi(\boldsymbol{x}) + \nabla\pi(\boldsymbol{x})) \\
&= \nabla \cdot (\nabla U(\boldsymbol{x})\pi(\boldsymbol{x}) - \nabla U(\boldsymbol{x})\pi(\boldsymbol{x})) \\
&= 0
\end{aligned}$$

Thus, if the Langevin diffusion process attains the measure given by $\pi(\boldsymbol{x})$ it will not change anymore and hence $\pi(\boldsymbol{x})$ is an invariant measure of the process. This means for one that the Langevin diffusion process can be made to converge to a desired density of the exponential family. On the other hand one can use it to sample from any unnormalised density $\tilde{p}(\boldsymbol{x})$ as taking $-U(\boldsymbol{x}) := \log \tilde{p}(\boldsymbol{x})$ means the stationary density of the Langevin diffusion will be

$$\begin{aligned}\pi(\boldsymbol{x}) &= \frac{\exp(-U(\boldsymbol{x}))}{\int \exp(-U(\boldsymbol{y}))d\boldsymbol{y}} \\ &= \frac{\tilde{p}(\boldsymbol{x})}{\int \tilde{p}(\boldsymbol{y})d\boldsymbol{y}} \\ &= p(\boldsymbol{x})\end{aligned}$$

i.e. the density of the stationary distribution of this Langevin diffusion process corresponds to the normalised version of $\tilde{p}$.

4.2.4 Unadjusted Langevin Algorithm (ULA)

The problem in general is that we do not have a closed form solution for the Langevin diffusion SDE, and cannot directly sample from the process it defines. The straightforward approach then is to simulate the process by discretising it and sampling from the discrete approximation instead. Discretising the SDE with the Euler-Maruyama scheme is the basis for the unadjusted Langevin Algorithm. For the SDE of the Langevin diffusion process this discretisation produces

$$\hat{\boldsymbol{X}}_{t_{i+1}} = \hat{\boldsymbol{X}}_{t_i} - \nabla U(\hat{\boldsymbol{X}}_{t_i})[t_{i+1} - t_i] + \sqrt{2(t_{i+1} - t_i)}\boldsymbol{Z}_i$$

where the $\boldsymbol{Z}_i$ are i.i.d. normal distributed, i.e. $\boldsymbol{Z}_i \sim \mathcal{N}(0, \boldsymbol{I})$. Let's modify this equation to reflect an equidistant grid of time points with step size ε and let $\boldsymbol{X}_n := \hat{\boldsymbol{X}}_{t_n}$ be the discretised approximation of $\boldsymbol{X}$ w.r.t. this grid at step n. We then have:

$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n - \varepsilon \nabla U(\boldsymbol{X}_n) + \sqrt{2\varepsilon}\boldsymbol{Z}_n$$

Which gives us a chain of connected random variables. So we can simulate realisations of this discrete process with this simple algorithm

```
Unadjusted Langevin Algorithm
  Initialise $\boldsymbol{x}_0$
  for $n \leq N$ do
      sample $\boldsymbol{Z}_n \sim \mathcal{N}(0, \boldsymbol{I})$
      $\boldsymbol{x}_{n+1} := \boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n) + \sqrt{2\varepsilon}\boldsymbol{Z}_n$
```

This scheme incurs a discretisation/integration error however that could change the stationary distribution. Guarantees for strong convergence are given only under some regularity conditions (see [?] chapter 11.2), namely that $\forall \boldsymbol{x}, \boldsymbol{y}$:

$$\begin{aligned}&\|\nabla U(\boldsymbol{x})\|_2 \leq M(1 + \|\boldsymbol{x}\|_2) \\ &\|\nabla U(\boldsymbol{y}) - \nabla U(\boldsymbol{x})\|_2 \leq L\,\|\boldsymbol{x} - \boldsymbol{y}\|_2\end{aligned}$$

for some constants M, L. This is a condition that for example the polynomial distribution introduced earlier does not fulfil. This problem gives rise to the Metropolis adjusted Langevin algorithm that introduces an acceptance probability to ensure convergence to the desired stationary distribution.

4.2.5 Metropolis Adjusted Langevin Algorithm (MALA)

The MALA method is a Metropolis-Hastings algorithm where the proposal density is based on the Langevin diffusion. We can now adapt our point of view and consider the next point $\boldsymbol{X}_{n+1}$ as the suggestion of our proposal density. For this we take this not as a discrete process rather think of a fixed vector $\boldsymbol{x}_n$ as a concrete realisation of $\boldsymbol{X}_n$. So for a fixed specific step the random vector $\hat{\boldsymbol{X}}$ is given by

$$\hat{\boldsymbol{X}} = \boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n) + \sqrt{2\varepsilon}\boldsymbol{Z}_n$$

Then the law of $\hat{\boldsymbol{X}}$ inherits the normal distribution of $\boldsymbol{Z}_n$ with mean

$$\begin{aligned}\mathbb{E}\left[\hat{\boldsymbol{X}}\right] &= \mathbb{E}\left[\boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n) + \sqrt{2\varepsilon}\boldsymbol{Z}_n\right] \\ &= \boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n) + \sqrt{2\varepsilon}\mathbb{E}\left[\boldsymbol{Z}_n\right] \\ &= \boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n).\end{aligned}$$

The difference between $\hat{\boldsymbol{X}}$ and its expectation is thus

$$\begin{aligned}\hat{\boldsymbol{X}} - \mathbb{E}\left[\hat{\boldsymbol{X}}\right] &= \left(\boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n) + \sqrt{2\varepsilon}\boldsymbol{Z}_n\right) - (\boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n)) \\ &= \sqrt{2\varepsilon}\boldsymbol{Z}_n\end{aligned}$$

and with this term we can directly calculate the variance

$$\begin{aligned}\mathrm{Var}(\hat{\boldsymbol{X}}) &= \mathbb{E}\left[(\hat{\boldsymbol{X}} - \mathbb{E}\left[\hat{\boldsymbol{X}}\right])^T(\hat{\boldsymbol{X}} - \mathbb{E}\left[\hat{\boldsymbol{X}}\right])\right] \\ &= \mathbb{E}\left[(\sqrt{2\varepsilon}\boldsymbol{Z}_n)^T(\sqrt{2\varepsilon}\boldsymbol{Z}_n)\right] \\ &= 2\varepsilon\mathbb{E}\left[\boldsymbol{Z}_n^T\boldsymbol{Z}_n\right] \\ &= 2\varepsilon\boldsymbol{I}.\end{aligned}$$

With this we can define a Metropolis-Hastings proposal density

$$q(\hat{\boldsymbol{x}}|\boldsymbol{x}_n) = \mathcal{N}(\hat{\boldsymbol{x}}|\boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n); 2\varepsilon\boldsymbol{I})$$

and state the complete algorithm:

Metropolis Adjusted Langevin Algorithm

```
Initialise $\boldsymbol{x}_0$
for $n \leq N$ do
    sample $\hat{\boldsymbol{x}} \sim \mathcal{N}(\boldsymbol{x}_n - \varepsilon \nabla U(\boldsymbol{x}_n), 2\varepsilon\boldsymbol{I})$
    compute $\alpha(\boldsymbol{x}_n, \hat{\boldsymbol{x}}) = \min\left\{1, \frac{p(\hat{\boldsymbol{x}})q(\boldsymbol{x}_n|\hat{\boldsymbol{x}})}{p(\boldsymbol{x}_n)q(\hat{\boldsymbol{x}}|\boldsymbol{x}_n)}\right\}$
    sample $u \sim \mathcal{U}(0,1)$
    if $u \leq \alpha(\boldsymbol{x}_n, \hat{\boldsymbol{x}})$ then
        $\boldsymbol{x}_{n+1} := \hat{\boldsymbol{x}}$
    else
        $\boldsymbol{x}_{n+1} := \boldsymbol{x}_n$
```

In order to implement the algorithm efficiently we would like to get a simplified expression for the acceptance probability

$$\alpha(\boldsymbol{x}_n, \hat{\boldsymbol{x}}) = \min\left\{1, \frac{p(\hat{\boldsymbol{x}})q(\boldsymbol{x}_n|\hat{\boldsymbol{x}})}{p(\boldsymbol{x}_n)q(\hat{\boldsymbol{x}}|\boldsymbol{x}_n)}\right\}$$

For the ratio of the target density we get

$$\frac{p(\hat{\boldsymbol{x}})}{p(\boldsymbol{x}_n)} = \frac{\exp(-U(\hat{\boldsymbol{x}}))}{\exp(-U(\boldsymbol{x}_n))} = \exp(U(\boldsymbol{x}_n) - U(\hat{\boldsymbol{x}}))$$

as the normalisation constant cancels in the ratio.
When substituting the density for the multivariate normal in the proposal density we get for general $\boldsymbol{x}, \boldsymbol{y}$,

$$q(\boldsymbol{x}|\boldsymbol{y}) = \frac{\exp\left(-\frac{1}{4\varepsilon}\left\|\boldsymbol{x} - \boldsymbol{y} + \varepsilon\nabla U(\boldsymbol{y})\right\|_2^2\right)}{\sqrt{(4\pi\varepsilon)^d}}$$

So with $\hat{\boldsymbol{x}}, \boldsymbol{x}_n$, we can write the ratio as

$$\begin{aligned}
\frac{q(\boldsymbol{x}_n|\hat{\boldsymbol{x}})}{q(\hat{\boldsymbol{x}}|\boldsymbol{x}_n)} &= \frac{\frac{\exp\left(-\frac{1}{4\varepsilon}\|\boldsymbol{x}_n - \hat{\boldsymbol{x}} + \varepsilon\nabla U(\hat{\boldsymbol{x}})\|_2^2\right)}{\sqrt{(4\pi\varepsilon)^d}}}{\frac{\exp\left(-\frac{1}{4\varepsilon}\|\hat{\boldsymbol{x}} - \boldsymbol{x}_n + \varepsilon\nabla U(\boldsymbol{x}_n)\|_2^2\right)}{\sqrt{(4\pi\varepsilon)^d}}} \\
&= \frac{\exp\left(-\frac{1}{4\varepsilon}\left\|\boldsymbol{x}_n - \hat{\boldsymbol{x}} + \varepsilon\nabla U(\hat{\boldsymbol{x}})\right\|_2^2\right)}{\exp\left(-\frac{1}{4\varepsilon}\left\|\hat{\boldsymbol{x}} - \boldsymbol{x}_n + \varepsilon\nabla U(\boldsymbol{x}_n)\right\|_2^2\right)} \\
&= \exp\left[\frac{1}{4\varepsilon}\left\|\hat{\boldsymbol{x}} - \boldsymbol{x}_n + \varepsilon\nabla U(\boldsymbol{x}_n)\right\|_2^2 - \frac{1}{4\varepsilon}\left\|\boldsymbol{x}_n - \hat{\boldsymbol{x}} + \varepsilon\nabla U(\hat{\boldsymbol{x}})\right\|_2^2\right] \\
&= \exp\left[-\frac{1}{4\varepsilon}\left(\left\|\boldsymbol{x}_n - \hat{\boldsymbol{x}} + \varepsilon\nabla U(\hat{\boldsymbol{x}})\right\|_2^2 - \left\|\hat{\boldsymbol{x}} - \boldsymbol{x}_n + \varepsilon\nabla U(\boldsymbol{x}_n)\right\|_2^2\right)\right]
\end{aligned}$$

Together we get the acceptance probability

$$\alpha(\boldsymbol{x}_n, \hat{\boldsymbol{x}}) = \min\left\{1, \exp\left(U(\boldsymbol{x}_n) - U(\hat{\boldsymbol{x}}) - \frac{1}{4\varepsilon}\left(\left\|\boldsymbol{x}_n - \hat{\boldsymbol{x}} + \varepsilon\nabla U(\hat{\boldsymbol{x}})\right\|_2^2 - \left\|\hat{\boldsymbol{x}} - \boldsymbol{x}_n + \varepsilon\nabla U(\boldsymbol{x}_n)\right\|_2^2\right)\right)\right\}$$

4.3 MCMC using Hamiltonian Dynamics

The idea to reconsider a distributions density as an energy landscape is the foundation of not only ULA and MALA but also of Hamiltonian Monte Carlo (HMC). HMC is a much more general concept however, that, in addition to the potential energy function U that characterises an EBM, introduces momentum variables to assign both potential as well as kinetic energy to a state via the Hamiltonian. The combination of these two energy functionals is then subjected to Hamiltonian dynamics to generate a trajectory to the next proposal state. MALA is in principle a special case of HMC where the trajectory to generate a new proposal consists of a single step.

4.3.1 Hamiltonian Dynamics

Hamiltonian dynamics is a framework used in classical mechanics to describe the evolution of physical systems. It introduces the concept of a Hamiltonian function, denoted as $H(\boldsymbol{x}, \boldsymbol{p})$,

$$H(\boldsymbol{x}, \boldsymbol{p}) = U(\boldsymbol{x}) + K(\boldsymbol{p})$$

where $\boldsymbol{x}$ represents the position vector and $\boldsymbol{p}$ the momentum vector and U and K are called the potential and kinetic energy respectively. Hamiltonian dynamics are governed by the equations:

$$\begin{aligned}
\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{dt}} &= \frac{\partial H}{\partial \boldsymbol{p}} \\
\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{dt}} &= -\frac{\partial H}{\partial \boldsymbol{x}}
\end{aligned}$$

For an EBM Model the potential energy function coincides with U, and for the kernel of any other density we can do the familiar transformation $U(\boldsymbol{x}) = -\log(\tilde{f}(x))$. The Hamiltonian in its turn can be used as the energy function in a new EBM to define a joint density

$$P(\boldsymbol{x},\boldsymbol{p}) = \frac{1}{Z}\exp(-H(\boldsymbol{x},\boldsymbol{p})) = \frac{1}{Z}\exp(-U(\boldsymbol{x}) - K(\boldsymbol{p}))$$

4.3.2 Hamiltonian Monte Carlo (HMC)

HMC is a Metropolis-Hastings sampling algorithm that employs Hamiltonian dynamics to generate new proposal states, and in this way guide the exploration of the state space efficiently. The simulated dynamics allow the algorithm to explore distant regions in a more coherent and directed manner compared to random-walk-based methods. Unlike traditional Metropolis Hastings algorithms, that often sample the proposed next state using a multivariate-normal step directly, HMC samples a momentum and subsequently uses the deterministic dynamics to simulate a trajectory from the current position to the next proposal state. Unlike in Langevin Dynamics this trajectory can consist of multiple steps which can boost convergence speed.
A key component is to negate the momentum variables at the end of one trajectory simulation. This guarantees that the proposal distribution q is symmetric which makes it vanish in the Metropolis-Hastings acceptance probability

$$\begin{aligned}\alpha(\boldsymbol{x},\boldsymbol{p};\hat{\boldsymbol{x}},\hat{\boldsymbol{p}}) &= \min\left(1, \frac{P(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}})q(\boldsymbol{x},\boldsymbol{p}|\hat{\boldsymbol{x}},\hat{\boldsymbol{p}})}{P(\boldsymbol{x},\boldsymbol{p})}q(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}}|\boldsymbol{x},\boldsymbol{p})\right)\\ &= \min\left(1, \frac{P(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}})}{P(\boldsymbol{x},\boldsymbol{p})}\right)\\ &= \min\left(1, \frac{P(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}})}{P(\boldsymbol{x},\boldsymbol{p})}\right)\end{aligned}$$

So that it can be expressed purely in terms of the respective Hamiltonians

$$\begin{aligned}\alpha(\boldsymbol{x},\boldsymbol{p};\hat{\boldsymbol{x}},\hat{\boldsymbol{p}}) &= \min\left(1, \frac{P(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}})}{P(\boldsymbol{x},\boldsymbol{p})}\right)\\ &= \min\left(1, \frac{\exp(-H(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}}))}{\exp(-H(\boldsymbol{x},\boldsymbol{p}))}\right)\\ &= \min\left(1, \exp(H(\boldsymbol{x},\boldsymbol{p}) - H(\hat{\boldsymbol{x}},\hat{\boldsymbol{p}}))\right)\end{aligned}$$

HMC's ability to use gradient information and simulate deterministic dynamics makes it a powerful tool for efficient exploration of complex, high-dimensional parameter spaces.
Set with the parameters

1. Desired number of samples: N
2. Number of leapfrog steps: L
3. Step size for leapfrog integration: ε
4. Mass matrix for kinetic energy: $\boldsymbol{M}$

and with concrete kinetic energy function $K(\boldsymbol{p}) = \frac{\boldsymbol{p}^T M \boldsymbol{p}}{2}$

```
Hamiltonian Monte Carlo Algorithm $(N, L, \varepsilon, \boldsymbol{M})$
Initialise: $\boldsymbol{x}_0$
$\boldsymbol{p}_0 \sim \mathcal{N}(0, \boldsymbol{M})$
for $i \leq N$ do
    $\hat{\boldsymbol{x}} = \boldsymbol{x}_i$
    sample Momentum: $\hat{\boldsymbol{p}} \sim \mathcal{N}(0, \boldsymbol{M})$
    $\hat{\boldsymbol{p}} = \hat{\boldsymbol{p}} - \varepsilon \frac{\nabla U(\hat{\boldsymbol{x}})}{2}$
    leapfrog Integration:
    for $k \leq L - 1$ do
        $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}} + \varepsilon \hat{\boldsymbol{p}}$
        $\hat{\boldsymbol{p}} = \hat{\boldsymbol{p}} - \varepsilon \nabla U(\hat{\boldsymbol{x}})$
    $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}} + \varepsilon \hat{\boldsymbol{p}}$
    $\hat{\boldsymbol{p}} = \hat{\boldsymbol{p}} - \varepsilon \frac{\nabla U(\hat{\boldsymbol{x}})}{2}$
    negate Momentum: $\hat{\boldsymbol{p}} = -\hat{\boldsymbol{p}}$
    evaluate Hamiltonian for the current point: $H(\boldsymbol{x}_i, \boldsymbol{p}_i) = U(\boldsymbol{x}_i) + K(\boldsymbol{p}_i)$
    evaluate Hamiltonian for the proposed point: $H(\hat{\boldsymbol{x}}, \hat{\boldsymbol{p}}) = U(\hat{\boldsymbol{x}}) + K(\hat{\boldsymbol{p}})$
    compute $\alpha((\boldsymbol{x}_i, \boldsymbol{p}_i), (\hat{\boldsymbol{x}}, \hat{\boldsymbol{p}})) = \min\{1, \exp(H(\boldsymbol{x}_i, \boldsymbol{p}_i) - H(\hat{\boldsymbol{x}}, \hat{\boldsymbol{p}}))\}$
    sample $u \sim \mathcal{U}(0, 1)$
    if $u \leq \alpha((\boldsymbol{x}_i, \boldsymbol{p}_i), (\hat{\boldsymbol{x}}, \hat{\boldsymbol{p}}))$ then
        $\boldsymbol{x}_{i+1} := \hat{\boldsymbol{x}}$
    else
        $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i$
```

4.4 Sampling Accuracy

It is important to note here that in practice we can not run the corresponding algorithm for an indefinite amount of time. A sampler based on MALA, ULA or HMC is used for generating training samples from the parametric model.
Typically one either just takes the first m samples in the chains, or lets the sampler run for some K iterations before taking the subsequent samples. That K is often called burn-in period and can improve the quality of the samples at the cost of more computation time.
When cutting a samplers chains off we incur two fundamental errors. The first is the error due to the discretisation of the process that is using $\hat{\boldsymbol{X}}_t$ instead of the true $\boldsymbol{X}_t$. It depends on the discretisation step ε and is meant to be counteracted by the Metropolis Hastings acceptance step in MALA. We are interested in strong convergence here and we know that the convergence order of the Euler-Maruyama discretisation is proportional to $\sqrt{\varepsilon}$. For ULA this is significantly more important as the quality of the samples suffers directly, while for MALA a poorly chosen ε means that less new samples will be accepted which can either impact quality or computational effort.
The second error is due to incomplete convergence, i.e. even if we could sample from $\boldsymbol{X}_t$ directly, we only have a guarantee that $\boldsymbol{X}$ converges to the target distribution in the limit, but might not resemble it closely enough at a specific t. One could run convergence diagnostics like the Gelman-Rubin test to assess the convergence status. That would introduce computational overhead however and complicate the code. Another option is to set a specific burn-in period and hope for sufficient convergence.

5 Maximum Likelihood Estimation (MLE)

In principle an EBM can be learned with MLE where for a dataset $(\boldsymbol{x}_i)_{i=1}^n$ the normalised log-likelihood function is given as

$$\frac{1}{n}\log \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\log \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \frac{1}{n}\sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \doteq \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]$$

where the gradient with respect to $\boldsymbol{\theta}$ is

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\frac{1}{n}\log\mathcal{L}(\boldsymbol{\theta}) &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[\frac{1}{n}\sum_{i=1}^{n}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right] \\
&= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[\frac{1}{n}\sum_{i=1}^{n}\log\left(\frac{1}{Z_{\boldsymbol{\theta}}}\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\right)\right] \\
&= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[-\frac{1}{n}\sum_{i=1}^{n}U_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \frac{1}{n}\sum_{i=1}^{n}\log\left(Z_{\boldsymbol{\theta}}\right)\right] \\
&= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[-\frac{1}{n}\sum_{i=1}^{n}U_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \log\left(Z_{\boldsymbol{\theta}}\right)\right] \\
&= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[-\frac{1}{n}\sum_{i=1}^{n}U_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \log\left(\int\exp(U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}\right)\right] \\
&= -\frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\log\left(\int\exp(U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}\right)
\end{aligned}$$

where by the chain rule we get that

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\log\left(\int\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}\right) = \frac{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\int\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}}{Z_{\boldsymbol{\theta}}}$$

and assuming $\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x})$ is differentiable in $\boldsymbol{\theta}$ for all $\boldsymbol{x}$ and is integrable in $\boldsymbol{x}$ for all $\boldsymbol{\theta}$, then

$$\begin{aligned}
\frac{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\int\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}}{Z_{\boldsymbol{\theta}}} &= \frac{\int\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}}{Z_{\boldsymbol{\theta}}} \\
&= -\frac{\int\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))d\boldsymbol{x}}{Z_{\boldsymbol{\theta}}} \\
&= -\int\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\frac{\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))}{Z_{\boldsymbol{\theta}}}d\boldsymbol{x} \\
&= -\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right]
\end{aligned}$$

The term $\frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ is a MC estimate for the expectation of the parameter gradient with respect to the data, i.e.

$$-\frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \doteq -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right]$$

so the final expression we get for $\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\log\mathcal{L}(\boldsymbol{\theta})$ is

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\frac{1}{n}\log\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right].$$

For training we use the negative log likelihood (NLL) as a loss so the sign reverses and hence the gradient for our loss is

$$-\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\frac{1}{n}\log\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right].$$

In order to approximate this gradient we can use a MC estimate with samples from the respective distributions, using our dataset for the first, and generated samples from our models for the second expectation. In order for this estimation strategy to work and to apply the samplers introduced earlier we need to have a way to calculate the energy functional U and have it be differentiable, both with respect to the inputs $\boldsymbol{x}$ and with respect to the model parameters $\boldsymbol{\theta}$.

6 Recovery Likelihood

The idea of recovery likelihood is to create perturbed samples $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \sigma\varepsilon$ with $\varepsilon \sim \mathcal{N}(0, I)$, from our original dataset and then use the conditional distribution

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) &= \frac{p(\tilde{\boldsymbol{x}}|\boldsymbol{x})p_{\boldsymbol{\theta}}(\boldsymbol{x})}{p(\tilde{\boldsymbol{x}})} \\ &= \frac{\exp(-\frac{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2^2}{2\sigma^2})\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}))}{Z_{\boldsymbol{\theta}}(2\pi\sigma^2)^{\frac{n}{2}}p(\tilde{\boldsymbol{x}})} \\ &= \frac{\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2^2}{2\sigma^2})}{Z_{\boldsymbol{\theta}}(2\pi\sigma^2)^{\frac{n}{2}}p(\tilde{\boldsymbol{x}})} \\ &= \frac{\exp(-U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2^2}{2\sigma^2})}{\tilde{Z}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})} \end{aligned}$$

Suppose now that for our samples $(\boldsymbol{x}_i)_{i=1}^n$ we have the perturbed samples $(\tilde{\boldsymbol{x}}_i)_{i=1}^n$ that we gained by adding noise with $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \sigma\varepsilon_i$, then the normalised recovery-log-likelihood (RLL) is defined as

$$\frac{1}{n}\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^n \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\tilde{\boldsymbol{x}})$$

With the expression above this is

$$\frac{1}{n}\sum_{i=1}^n \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) = \frac{1}{n}\sum_{i=1}^n \left(-U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2^2}{2\sigma^2}) - \log(\tilde{Z}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})\right)$$

where

$$\begin{aligned} \log(\tilde{Z}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}})) &= \log(Z_{\boldsymbol{\theta}}(2\pi\sigma^2)^{\frac{n}{2}}p(\tilde{\boldsymbol{x}})) \\ &= \log(Z_{\boldsymbol{\theta}}) + \frac{n}{2}\log(2\pi\sigma^2) + \log(p(\tilde{\boldsymbol{x}})) \end{aligned}$$

Thus upon differentiating by θ we have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\frac{1}{n}\mathcal{J}(\boldsymbol{\theta}) &= -\frac{1}{n}\sum_{i=1}^n \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\log(\tilde{Z}_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}) \\ &= -\frac{1}{n}\sum_{i=1}^n \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}(\log(Z_{\boldsymbol{\theta}}) + \frac{n}{2}\log(2\pi\sigma^2) + \log(p(\tilde{\boldsymbol{x}}))) \\ &= -\frac{1}{n}\sum_{i=1}^n \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\log(Z_{\boldsymbol{\theta}}) \\ &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\frac{1}{n}\mathcal{L}(\boldsymbol{\theta}) \end{aligned}$$

So the gradients of RLL and standard LL coincide. This means we can sample from the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\tilde{\boldsymbol{x}})$, which is less multimodal. Indeed Gao and Song have shown that for small σ the conditional can be approximated by a normal distribution,

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) \doteq \mathcal{N}\left(\boldsymbol{x}|\tilde{\boldsymbol{x}} - \sigma^2 \nabla_{\boldsymbol{x}} U_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}), \sigma^2 I\right)$$

so the smaller we choose σ, the more the conditional resembles a shifted normal distribution, which is much easier to sample from. When we use the previously introduced Markov chain samplers we can treat the perturbed distribution like another EBM model with conditional energy functional

$$\tilde{U}_{\boldsymbol{\theta}}(\boldsymbol{x}) = U_{\boldsymbol{\theta}}(\boldsymbol{x}) + \frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_2^2}{2\sigma^2}$$

and use its gradient to generate the samples. This gradient is given by

$$\begin{aligned}
\nabla_{\boldsymbol{x}} \tilde{U}_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{x}}(U_{\boldsymbol{\theta}}(\boldsymbol{x}) + \frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_2^2}{2\sigma^2}) \\
&= \nabla_{\boldsymbol{x}} U_{\boldsymbol{\theta}}(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} \frac{(\tilde{\boldsymbol{x}} - \boldsymbol{x})^T(\tilde{\boldsymbol{x}} - \boldsymbol{x})}{2\sigma^2} \\
&= \nabla_{\boldsymbol{x}} U_{\boldsymbol{\theta}}(\boldsymbol{x}) - \frac{\tilde{\boldsymbol{x}} - \boldsymbol{x}}{\sigma^2}
\end{aligned}$$

7 Metrics

Recovery Likelihood promises an improvement in convergence speed by accelerating the calculation of the likelihood gradient. Does this improvement come at the cost of quality, reliability or increased maintenance? What would we like to have:

1. Accurate parameter estimates
2. A sufficiently close parametric representation/model for the data distribution
3. Consistent quality in training
4. Independence of the model
5. Straightforward parameter choice and low dependency for perturbation

Comparing the likelihood approaches and the involved samplers in a rigorous way requires some measure of both the quality of the final estimates and the reliability of the approach.

7.1 Metrics for Estimated Parameters

Many metrics could be used to evaluate the quality of the estimates, but for simplicity, here only some simple metrics of the distance between the known parameters and the estimated parameters are employed. For vector valued parameters $\boldsymbol{\theta} \in \mathbb{R}^m$ the p-norm family, and specifically the euclidian norm is used i.e.

$$\text{error}_p(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \left\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right\|_p = \left(\sum_{i=1}^{m} (\theta_i - \hat{\theta}_i)^p\right)^{\frac{1}{p}}$$

while for matrix valued parameters, like e.g. the covariance matrices of the Gaussian model, the Frobenius norm of the difference is applied

$$\operatorname{error}(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) = \left\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\right\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n}(\theta_{ij} - \hat{\theta}_{ij})^2\right)^{\frac{1}{2}}$$

These give a measure of how good the respective approach is at estimating the parameters accurately and would be specifically important in contexts, where the parameters have a specific meaning and the estimates are meant to be interpretable.

8 Framework

The effectiveness of an inference procedure depends on many factors and components, and with so many variables in play general conclusions are difficult to obtain from empirical tests. Several models and settings were tested in this project, but the results obtained may not translate to other circumstances.
So while there is value in conducting these experiments and conclusions can be drawn from them, providing the infrastructure to conduct specialised experiments in the future might be a more fruitful contribution. Thus a central goal of this project was to create a code framework, that facilitates setting up and combining components in a way that can be tailored to circumstances of interest which are not discussed here.
Designing code for generality is a difficult task but the created modules should allow tests of the RL and ML estimation techniques for a variety of energy based models. In the design special care was taken to keep the components modular and extensible wherever possible, so hopefully the framework can not only be used to reproduce the results discussed here but serve as a basis for further investigation. Another key objective in the design was maximising computational efficiency in the major components. Thus all of the core components operate purely on `torch.Tensor`s and allow for leveraging the optimised `torch` GPU processing.

8.1 The Energy Model

The estimation process consists of calculating the respective likelihood gradients that a `torch` optimiser then uses to update the model parameters during training. These gradients are obtained as Monte Carlo estimates of the expectations show in chapter ( ).
For the model part samples from the model distribution need to be generated and their parameter gradients evaluated batch-wise. Here the model samples are produced via MCMC samplers whose iterations require evaluating the energy functional and the gradient with respect to the input of that functional. Thus the model class needed methods for calculating the energy functional $U_{\boldsymbol{\theta}}(\boldsymbol{x})$, the gradient with respect to the input $\nabla_{\boldsymbol{x}} U_{\boldsymbol{\theta}}(\boldsymbol{x})$ and the gradient with respect to the parameters evaluated at the input $\nabla_{\boldsymbol{\theta}} U_{\boldsymbol{\theta}}(\boldsymbol{x})$.
This is achieved by the `EnergyModel` base class, which inherits from `torch.nn.Module`, to integrate well with the machine learning architecture `torch` provides. It initialises with an empty `params` dictionary of type `torch.nn.ParameterDict` and has the methods

1. `forward(x)` which relays calls to `energy`
2. `energy(x)` corresponding to $U_{\boldsymbol{\theta}}(\boldsymbol{x})$
3. `energy_grad(x)` corresponding to $\nabla_{\boldsymbol{x}} U_{\boldsymbol{\theta}}(\boldsymbol{x})$
4. `avg_param_grad(x)` which directly calculates $\frac{1}{n}\sum_{j=1}^{n} \nabla_{\boldsymbol{\theta}} U_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$

where `x` can be either a single input or a batch of inputs. The method `avg_param_grad` stands out here as for a batch it does not produce a matrix of gradients, but already averages the gradients across samples, effectively calculating the MC estimate for $\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{\theta}}}\left[\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}U_{\boldsymbol{\theta}}(\boldsymbol{x})\right]$ directly.
Any desired energy model, subclassing `EnergyModel`, needs only inscribe its named parameters in the `params` dictionary and override the `energy` method. The `energy_grad` and `avg_param_grad` methods can be overridden, if an analytical representation of these gradients is available, or inherited from the base class, where they leverage `torch`s automatic differentiation capability. The sampler and ML classes can use any instance of such a model directly.

8.2 Recovery Adapter

For RL estimation it is more complicated as the gradients are taken with respect to the conditional energy that depends on the perturbed samples and requires extra terms, that are derived from the normal distribution of the noise. One approach could have been to create parallel class hierarchies and create a regular and a conditional class for every model, training and evaluating them separately. This would have made the pipeline and evaluation more complicated however, and since the conditional version is only relevant during training, there would have been a lot of boilerplate.
The chosen solution for this problem is utilising the Object Adapter design pattern with the `RecoveryAdapter` class. Essentially the adapter both inherits from `EnergyModel` and incorporates an instance of this type at the same time, overriding the methods by relaying the calls first to this instance and then adding the conditional terms. It has one extra method that allows to set a batch of perturbed samples which are utilised in these conditional terms. At every training iteration a perturbed version of the current batch of training data is created by adding Gaussian noise in the RecoveryLikelihood class and passing them to the adapted model before generating samples and calculating the parameter gradient.
The major benefits of this approach are that, for one any model can be adapted directly, without having to design an additional class, and that after training the actual model can simply be retrieved from within the adapter and the adapter can be discarded without the need for transformation.

9 Results / Assessment

As the MC estimates, used to approximate the gradient of the likelihood are based on random samples of the model distribution and the data distribution respectively, are themselves random variables, the entire estimation process is stochastic in nature and we can view it as a stochastic process.
Since the sequence of parameter estimates at every iteration is essentially a realisation of this stochastic process, the error metrics can be viewed as one as well. Suppose for an experiment a full training procedure is conducted with a total number of n iterations across epochs. Because of the stochastic nature of the training the training procedure is repeated m times, with the same hyper parameters, to get an impression of the parameter estimation process as a whole. For any error metric M we then obtain a set of m time series of length n and can write $M_i(j)$ for the value of the metric for training run i at iteration j. So the comparisons in the following are done by assessing and plotting the development of the error measures or likelihood values as stochastic processes, and by their final parameter estimate.

9.1 Remarks on Hyperparameters

Assessing performance of learning in this context is complicated a lot by the large number of hyperparameters and the fact that the best choice of these is typically model dependent. In order to simplify the task and level the playing field some are thus chosen as fixed. In all experiments the used optimiser is ADAM [**?**] and the learning rate is fixed to $\alpha = 10^{-3}$ as recommended in the paper. In ADAM α is used in form of a coefficient in front of a linearly transformed form of the gradient

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha f(\nabla_{\boldsymbol{\theta}}\mathrm{NLL}(\boldsymbol{\theta}_k))$$

and as the framework uses the MC estimates directly the gradient is $\frac{1}{b}\nabla_{\boldsymbol{\theta}}\text{NLL}(\boldsymbol{\theta}_k)$, and thus the de-facto learning rate changes with the batch size, from α to something like $\frac{\alpha}{b}$. The batch size is also set fixed to $b = 200$ samples per batch so the learning rate is set at $\alpha = \frac{200}{10^{-3}} = 0.2$ This learning rate may not be the optimal choice for any of the tested models, but makes comparisons easier and more direct.
The number of epochs is set to 10 and with a randomly loaded dataset of 10^4 samples. Thus the resulting total number of training iterations for a complete training run is $n = 500$. All samplers require a starting batch which is set to zeros. Excepted when claimed otherwise all experiments have been run repeating the training procedure $m = 50$ times. Thus when talking about a mean for an experiment setting this mean is taken across training procedures. The parameter process plots shown later represent the mean and standard deviation of the parameter process, where the estimates at each training iteration are averaged across training runs, i.e. for a metric M

$$M(j) = \frac{1}{m}\sum_{i=1}^{m} M_i(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_j)$$

9.2 The Choice of the Sampler Step Size

The choice of the sampling step size ε is a complex problem. In many experiments the estimation process broke down due to the choice of ε. This was due to two main problems:
For the polynomial model, as stated before, ULA sampler is inherently unstable and even during a single training step the sampling process can diverge easily. It is a distribution with very light tails and the probability mass is concentrated in a small region. If a sample in just one of the chains happens to be close to the boundary of the complement of that region, in which the energy increases very strongly, then a random step can take it to a location where the state has very large energy. This in turn produces a very large gradient in absolute value, and that gradient can push the chain in the opposite direction such that it again lands in a region of high energy. Essentially this chain then fluctuates itself to infinity, which breaks the parameter gradient, leading to a breakdown of the training.
This does not happen for ULA and MALA due to their acceptance step, but there is still a problem with the instability of the models. `pytorch` has no direct mechanism to constrain parameters to a specific domain, and so the models discussed here can have their parameters changed in such a way, that they are no longer probability distributions, which subsequently leads to a breakdown of the training procedure. For the polynomial model, the coefficient of the highest order monomial needs to be positive, otherwise the density simply turns into an exploding exponential function. Something similar happens in the Gaussian models. If any of the covariance matrices stop being positive semi-definite the energy can turn negative and the same problem of an unbounded density occurs. This problem is especially pronounced in the beginning of the training procedure and depends both on ε and the start batch. The start batch was set to zeros, both for convenience and to reflect the issue that a good start batch from the model distribution is unavailable for complex models. But of course a batch of zeros does not represent the model distribution and if the burnin or ε are too low, then the samples generated initially from the start batch poorly reflect the model distribution and effect large parameter gradients that can push the parameters to infeasible regions in the first couple training iterations.
One could counteract this problem by reducing the learning rate but here the approach was to adapt the step size ε instead for consistency. Both for MALA and HMC a larger step size like $\varepsilon = 10^{-1}$ turned out to be mostly stable. For ULA this was also true for the Gaussian models but not for the polynomial where the problem described first led to divergence.

9.3 Expectations

As the recovery adaptation makes the distribution to be sampled from more unimodal the smaller the perturbation variance is chosen, one can expect RL to do better the more multimodal the target and starting distribution are. There should be a tradeoff between burnin choice and quality of the estimates. Burnin has the largest impact on overall training time. This is intuitively easy to understand as every sampling process involves iterating as many chains as the batch dimension $\text{burnin} + 1$ times. With a high burnin both ML and RL can be expected to perform similarly well. The higher the burnin the closer the samples are to the model distribution in ML, so a higher burnin can be expected to improve quality overall. This improvement comes at the cost of training iteration time however, which is the essential conundrum that RL intends to resolve. Thus the main focus of testing for RL here is to compare the results when the burnin is set low, as even with relatively low burnin the resulting samples at each iteration should be more accurate when the original distribution is multimodal.
RL does have an additional price during every sampler iteration, which comes from calculating the conditional components at every call to the `energy` and `energy_grad` methods during sampling. Additionally the samples that the methods are conditioned on have to be reset after each training iteration. If the distribution is itself unimodal this should be a price without extra gain, but when the distribution is otherwise difficult to sample from, then RL could allow to use a lower burnin.

9.4 Univariate Polynomial

The polynomial model was tested with a bimodal target distribution with target parameters:

$$\boldsymbol{w} = (w_1, w_2, w_3, w_4) = (-1.2, -0.7, 2, 1)$$
$$\text{i.e.}$$
$$U(x) = x^4 + 2x^3 - 0.7x^2 - 1.2x$$

and an unimodal start distribution with the parameters:

$$\boldsymbol{w}_0 = (w_1^0, w_2^0, w_3^0, w_4^0) = (1, 1, 1, 1)$$
$$\text{i.e.}$$
$$U_{\boldsymbol{w}_0}(x) = x^4 + x^3 + x^2 + x$$

As the model is one dimensional in the input it is fairly fast to train and was tested most comprehensively. For MALA and HMC the step size was set to $\varepsilon = 10^{-1}$, but for ULA sampler consistently diverged for this setting so all tests for ULA were done with $\varepsilon = 10^{-2}$. This may seem like an unfair comparison as a higher step size allows for faster convergence, if convergence happens at all. But this disadvantage for ULA is somewhat mitigated by the fact that the sample distribution for both MALA and HMC tend to change much slower overall due to the acceptance step, which causes chains of both samplers to often remain in a state for some time in the onset of the training process.
A full training procedure was run for all combinations of the samplers and the following settings for burnin and perturbation variance

1. Burnin $\in [10, 20, 50, 100]$
2. Perturbation variance $\sigma \in [0.1, 0.5, 1]$

for RL and without the perturbation variance also for ML. This resulted in a total number of more than a million training iterations combined.
In most of the conducted experiments ML performed better across the board, but in some settings RL obtained better results on average. One important result was that the choice of perturbation variance is extremely important for RL. Choosing $\sigma = 0.1$ produced parameter divergence irrespective of the sampler used.

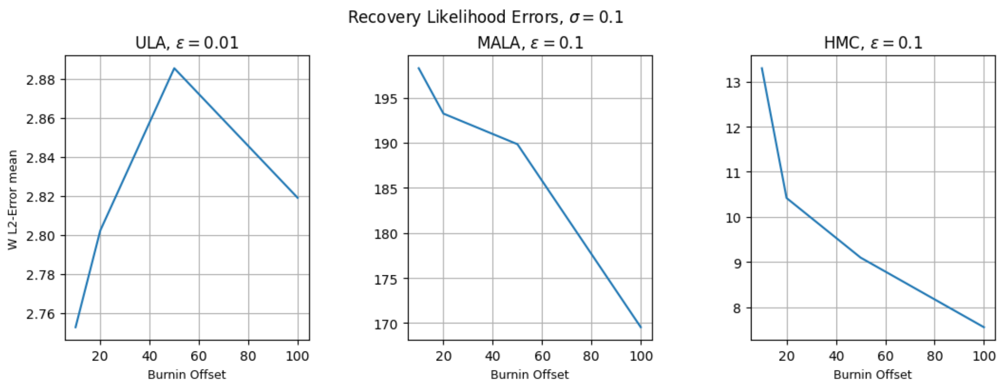

Figure 1: Recovery Likelihood error for $\sigma = 0.1$ against burnin

In that setting, while still performing very poorly, ULA surprisingly performed the best. A reason for this could be that the data samples upon which the recovery adapter conditions change at every training iteration. If σ is too small, and the conditional distribution has thus low variance, then changing the samples can potentially shift the mean and effectively reset the convergence of the chains. Especially for MALA and HMC this could lead to a large number of rejected samples and thus poor sample quality, where the chains might remain mostly in their previous modes.
As the parameters diverged for $\sigma = 0.1$ these runs are excluded in the following results.

9.4.1 Results for ULA

For larger burnin, recovery likelihood with $\sigma = 0.5$ actually outperformed ML, with both better final errors and a lower standard deviation.

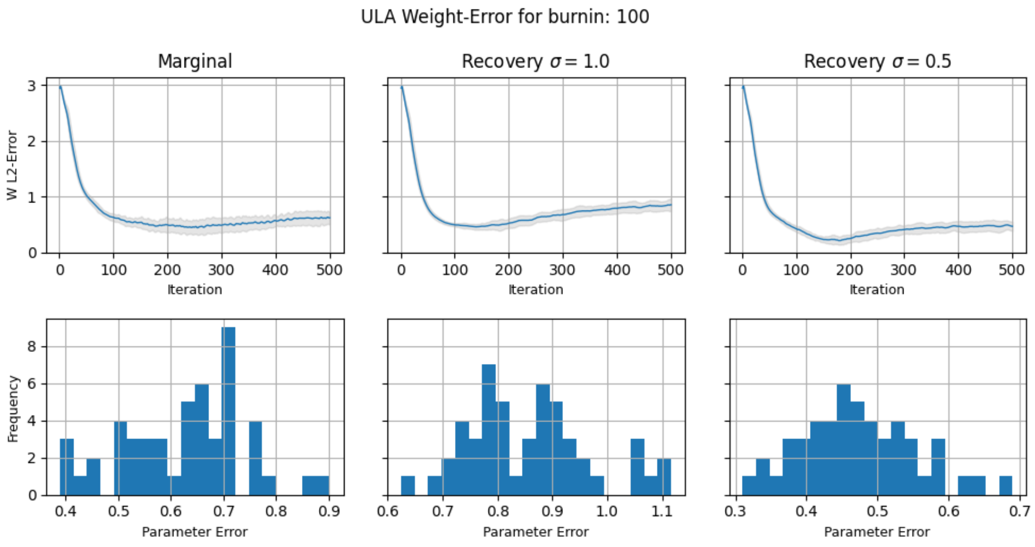

Figure 2: ULA performance across runs. The upper plots represent the entire parameter process, histograms show final error

The average error for ML at this setting was at 0.62 with a standard deviation of 0.11, while RL had an average error of 0.47 with a standard deviation of 0.08. This trend reversed rapidly when decreasing the burnin however, with noticeably worse results for both settings of RL at a burnin lower than 50.

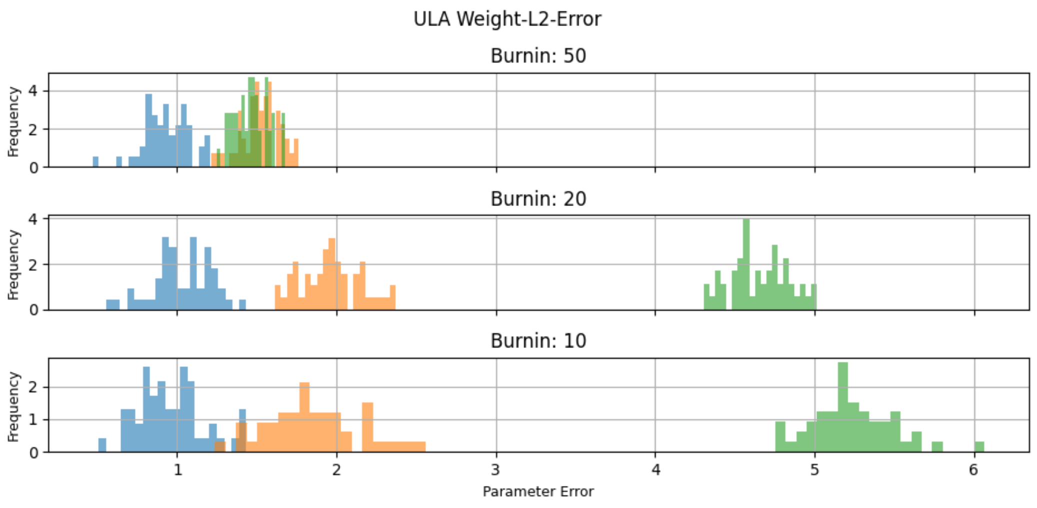

Figure 3: ULA performance across runs. Marginal Likelihood (blue), RL with $\sigma = 0.5$ (green), RL with $\sigma = 1.0$ (orange)

It is noteworthy that ML didn't suffer as significantly as expected for the lower burnins, although the overall performance was worse using ULA.

9.4.2 Results for MALA

The experiments with MALA show a trend for the variance of the RL estimator more clearly. For a burnin of 100 the overall estimates are fairly good, but we can see that decreasing σ increases the variance of the final estimator.

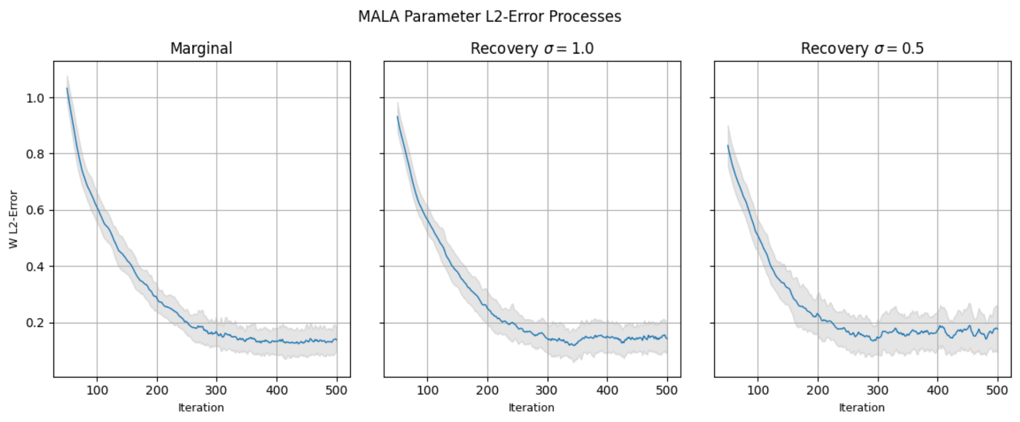

Figure 4: MALA processes for burnin 100. RL for $\sigma = 0.5$ shows significantly higher variance

9.5 Multivariate Gaussian Model

The multivariate Gaussian distribution is unimodal and easy to sample from. It serves as a baseline, for which one would expect both ML and RL do perform similarly well in terms of estimate quality. The test model is two dimensional with target parameters

$$\mu = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

and start parameters

$$\mu_0 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$
$$\Sigma_0 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}.$$

9.6 Gaussian Mixture Model

Target parameters:

$$\mu_1 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \qquad \mu_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$
$$\Sigma_1 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \qquad \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

with weights $(w_1, w_2) = (0.2, 0.8)$
Start parameters:

$$\mu_1^0 = \begin{pmatrix} 3 \\ 3 \end{pmatrix} \qquad \mu_2^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$\Sigma_1^0 = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \qquad \Sigma_2^0 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

with weights $(w_1, w_2) = (0.5, 0.5)$

10 Discussion

warm up period Samplers could implement ε schedule, potentially improving especially ULA Perturbation variance schedule

10.1 Framework

During the course of this project the framework grew organically with some major milestones along the way being the creation of the multi-chain samplers, the base class for the energy models and the implementation of the recovery adapter and the training observers.
Many of the created components turned out well and reusable but there is a lot of room for extension and potential improvement in the pipeline. As the issues and findings discussed in the results section show, the samplers could profit significantly from introducing an ε schedule. By setting the initial ε high and decreasing it in later epochs, one could avoid the problem of steep gradients in the beginning, and at the same time approximate the model distribution better in the later stages, when an overall burnin of the chains has taken place.
Another way to counteract the initially poor representation of the model distribution would be to introduce a warmup burnin, before the actual training procedure starts. This would allow using even a poorly chosen start batch with lower burnin periods during the training.
Similarly it could make sense to introduce a schedule for the perturbation variance as well.

11 References