Generative Models

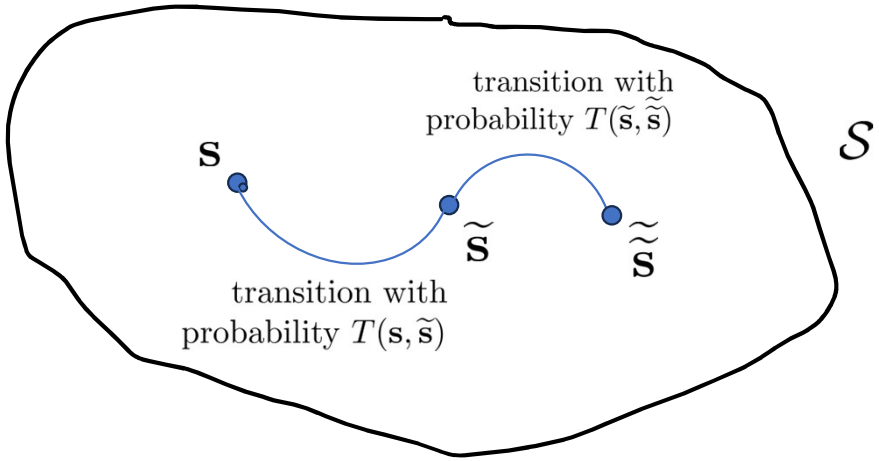
Lecture 3

Markov Chain Monte Carlo (MCMC)

Recall that a (homogeneous) Markov Process consists of a set $\mathcal{S}$ called the *state space* and for each $\mathbf{s} \in \mathcal{S}$ a *transition probability*

$$T(\mathbf{s}, \widetilde{\mathbf{s}}) \text{ over } \mathcal{S}$$

This is the probability of transitioning from state $\mathbf{s}$ to state $\widetilde{\mathbf{s}}$.



A *Markov Chain* is generated by starting at a state $\mathbf{s}_0$ and then sample $\mathbf{s}_1$ from the distribution $T(\mathbf{s}_0, \widetilde{\mathbf{s}})$.

Then sampling a $\mathbf{s}_2$ from $T(\mathbf{s}_1, \widetilde{\mathbf{s}})$ etc. and in general, sample $\mathbf{s}_{t+1}$ from $T(\mathbf{s}_t, \widetilde{\mathbf{s}})$.

Thus we get a sequence

$$\mathbf{s}_0 \to \mathbf{s}_1 \to \mathbf{s}_2 \to \cdots \to \mathbf{s}_t \to \mathbf{s}_{t+1} \to \ldots$$

Assume we are given a distribution p_0 on $\mathcal{S}$. Then we get another distribution p_1 by

$$p_1(\widetilde{\mathbf{s}}) = \int T(\mathbf{s}, \widetilde{\mathbf{s}}) p_0(\mathbf{s}) d\mathbf{s} = \mathbb{E}_{p_0(\mathbf{s})}(T(\mathbf{s}, \widetilde{\mathbf{s}}))$$

and in general

$$p_{t+1}(\widetilde{\mathbf{s}}) = \int T(\mathbf{s}, \widetilde{\mathbf{s}}) p_t(\mathbf{s}) d\mathbf{s}$$

In case $\mathcal{S}$ is a finite set the integral is replaced by a sum.

Thus given an initial distribution on $\mathcal{S}$ makes $T(\mathbf{s}, \widetilde{\mathbf{s}})$ equal to the conditional probability $p_t(\widetilde{\mathbf{s}} \,|\, \mathbf{s})$ for all t

Limiting Distribution

A distribution q on $\mathcal{S}$ is said to be a limiting or stable distribution for the Markov Process if

$$q(\widetilde{\mathbf{s}}) = \int T(\mathbf{s}, \widetilde{\mathbf{s}}) q(\mathbf{s}) d\mathbf{s}$$

This means that

$$q_t = q_{t+1} = q_{t+n}$$

for all t and n

Existence of a Limiting Distribution

Theorem Assume that for any $\mathbf{s}, \widetilde{\mathbf{s}} \in \mathcal{S}$ there exists a chain

$$\mathbf{s} = \mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_u = \widetilde{\mathbf{s}}$$

for some u depending on $\mathbf{s}, \widetilde{\mathbf{s}}$. So any two points in $\mathcal{S}$ can be joined by a Markov Chain (the process is said to be *irreducible*)

Assume further that the process is non-periodic.

Then starting from any distribution p_0 there exists a t such that $p_t = p_{t+1}$ so p_t is a stable distribution. Furthermore the stable distribution is unique and is independent of the starting distribution p_0

The idea of $MCMC$ is, for a given distribution, p, we want to sample from, construct a Markov Process with p as its stable distribution.

Generating a chain $\{\mathbf{s}_t\}_t$, $\mathbf{s}_t$ will be samples from the stable distribution for $t >> 1$

The samples $\mathbf{s}_t, \mathbf{s}_{t+1}, \mathbf{s}_{t+2}, \ldots$ are obviously not independent so one might want to take samples $\mathbf{s}_t, \mathbf{s}_{t+n}, \mathbf{s}_{t+2n}, \ldots$ for some, sufficiently large n, to make the samples be less correlated.

Metropolis-Hastings Algorithm

There are several ways to make Markov Processes with a specific stable distribution but they are all more or less based on the *Metropolis-Hastings* method.

Proposition

If π is a distribution such that

$$T(\mathbf{s}, \widetilde{\mathbf{s}})\pi(\mathbf{s}) = \pi(\widetilde{\mathbf{s}})T(\widetilde{\mathbf{s}}, \mathbf{s})$$

then π is the stable distribution.

Indeed

$$\int T(\mathbf{s}, \widetilde{\mathbf{s}})\pi(\mathbf{s})d\mathbf{s} = \int \pi(\widetilde{\mathbf{s}})T(\widetilde{\mathbf{s}}, \mathbf{s})d\mathbf{s} = \pi(\widetilde{\mathbf{s}}) \int T(\widetilde{\mathbf{s}}, \mathbf{s})d\mathbf{s} = \pi(\widetilde{\mathbf{s}})$$

We want to generate samples from a distribution p. Typically we can write the density in the form $p = \frac{g}{Z}$ where g is a positive function $g : \mathbb{R}^d \to \mathbb{R}$ and $Z = \int_{\mathbb{R}^d} g(\mathbf{x}) d\mathbf{x}$ is an intractable normalization constant (any density function can be written in this form)

The Metropolis-Hastings algorithm is a *proposal, acceptance/rejection* algorithm

Choose a *proposal distribution* $Q(\mathbf{s}, \widetilde{\mathbf{s}})$ which is symmetric in the sense

$$Q(\mathbf{s}, \widetilde{\mathbf{s}}) = Q(\widetilde{\mathbf{s}}, \mathbf{s})$$

For instance a Gaussian $\mathcal{N}(\mathbf{s} \,|\, \widetilde{\mathbf{s}}, \Sigma)$ is symmetric

Given a state $\mathbf{s}$ we sample another state $\widetilde{\mathbf{s}}$ from the distribution $Q(\mathbf{s}, \widetilde{\mathbf{s}})$

This is the proposed new point, we then have to decide whether to accept or reject $\widetilde{\mathbf{s}}$. If accepted this becomes the new point in the sequence.

If rejected we sample another point

The acceptance/rejection is controlled by an *acceptance function*

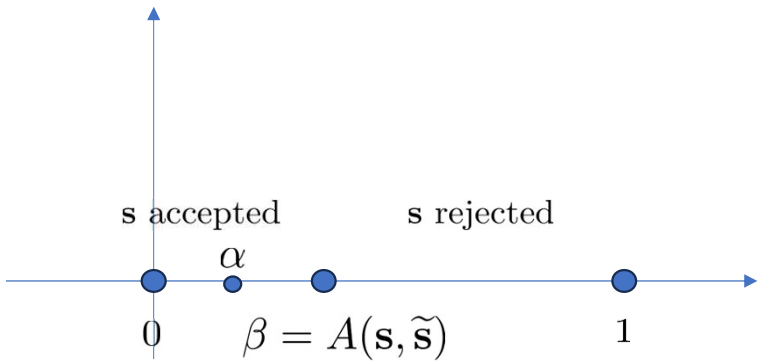
The acceptance function

$$A(\mathbf{s}, \widetilde{\mathbf{s}}) \in [0, 1]$$

determines the probability that the sample $\widetilde{\mathbf{s}}$ from the proposal distribution $Q(\mathbf{s}, \widetilde{\mathbf{s}})$ is accepted.

Let $A(\mathbf{s}, \widetilde{\mathbf{s}}) = \beta$

To determine whether $\widetilde{\mathbf{s}}$ is accepted, sample α from a uniform distribution over $[0, 1]$ then the proposed sample $\widetilde{\mathbf{s}}$ is accepted if $\beta \geq \alpha$



Now we construct a Markov Process with transition probabilities

$$T(\mathbf{s}, \widetilde{\mathbf{s}}) = \begin{cases} Q(\mathbf{s}, \widetilde{\mathbf{s}}) A(\mathbf{s}, \widetilde{\mathbf{s}}) \text{ if } \mathbf{s} \neq \widetilde{\mathbf{s}} \\ Q(\mathbf{s}, \mathbf{s}) + \int Q(\mathbf{s}', \mathbf{s})(1 - A(\mathbf{s}', \mathbf{s}))d\mathbf{s}' \end{cases}$$

We need to show that $T(\mathbf{s}, \widetilde{\mathbf{s}})$ is indeed a transition distribution.

We just show this in the finite case. So we need to show

$$\sum_{\widetilde{\mathbf{s}} \in \mathcal{S}} T(\mathbf{s}, \widetilde{\mathbf{s}}) = 1 \text{ for every } \mathbf{s}$$

Write the sum as

$$\begin{aligned}
&\sum_{\widetilde{\mathbf{s}} \in \mathcal{S}/\{\mathbf{s}\}} Q(\mathbf{s}, \widetilde{\mathbf{s}}) A(\mathbf{s}, \widetilde{\mathbf{s}}) + T(\mathbf{s}, \mathbf{s}) \\
&= \sum_{\widetilde{\mathbf{s}} \in \mathcal{S}/\{\mathbf{s}\}} Q(\mathbf{s}, \widetilde{\mathbf{s}}) A(\mathbf{s}, \widetilde{\mathbf{s}}) + Q(\mathbf{s}, \mathbf{s}) + \sum_{\mathbf{s}' \in \mathcal{S}/\{\mathbf{s}\}} Q(\mathbf{s}, \mathbf{s}')(1 - A(\mathbf{s}, \mathbf{s}')) \\
&= \sum_{\widetilde{\mathbf{s}} \in \mathcal{S}/\{\mathbf{s}\}} Q(\mathbf{s}, \widetilde{\mathbf{s}}) + Q(\mathbf{s}, \mathbf{s}) = 1
\end{aligned}$$

Now given a distribution $p = \frac{g}{Z}$ define an acceptance probability by

$$A(\mathbf{s}, \widetilde{\mathbf{s}}) = \min\left(1, \frac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})}\right) = \min\left(1, \frac{g(\widetilde{\mathbf{s}})}{g(\mathbf{s})}\right)$$

Remark how the troublesome normalization factor , Z, drops out in the fraction

We shall show that $p = \frac{g}{Z}$ is the stable distribution of the Markov process we just constructed (remark that this does not depend on the choice of the proposal distribution Q as long as it is symmetric)

Assume $\widetilde{\mathbf{s}} \neq \mathbf{s}$ (otherwise there is nothing to show)

Then

$$\frac{T(\mathbf{s}, \widetilde{\mathbf{s}})}{T(\widetilde{\mathbf{s}}, \mathbf{s})} = \frac{Q(\mathbf{s}, \widetilde{\mathbf{s}}) A(\mathbf{s}, \widetilde{\mathbf{s}})}{Q(\widetilde{\mathbf{s}}, \mathbf{s}) A(\widetilde{\mathbf{s}}, \mathbf{s})} = \frac{A(\mathbf{s}, \widetilde{\mathbf{s}})}{A(\widetilde{\mathbf{s}}, \mathbf{s})}$$

Assume $p(\widetilde{\mathbf{s}}) > p(\mathbf{s})$ so $\dfrac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})} > 1$ and $\dfrac{p(\mathbf{s})}{p(\widetilde{\mathbf{s}})} < 1$.

Then

$$A(\mathbf{s}, \widetilde{\mathbf{s}}) = \min\{1, \frac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})}\} = 1 \text{ so } \widetilde{\mathbf{s}} \text{ is accepted}$$

and

$$A(\widetilde{\mathbf{s}}, \mathbf{s}) = \min\{1, \frac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})}\} = \frac{p(\mathbf{s})}{p(\widetilde{\mathbf{s}})}$$

Thus

$$\frac{T(\mathbf{s},\widetilde{\mathbf{s}})}{T(\widetilde{\mathbf{s}},\mathbf{s})} = \frac{A(\mathbf{s},\widetilde{\mathbf{s}})}{A(\widetilde{\mathbf{s}},\mathbf{s})} = 1/\frac{p(\mathbf{s})}{p(\widetilde{\mathbf{s}})} = \frac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})}$$

so

$$T(\mathbf{s},\widetilde{\mathbf{s}})\, p(\mathbf{s}) = p(\widetilde{\mathbf{s}})\, T(\widetilde{\mathbf{s}},\mathbf{s})$$

If $p(\widetilde{\mathbf{s}}) < p(\mathbf{s})$ $A(\mathbf{s},\widetilde{\mathbf{s}}) = \dfrac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})}$ and $A(\widetilde{\mathbf{s}},\mathbf{s}) = 1$ so again

$$\frac{T(\mathbf{s},\widetilde{\mathbf{s}})}{T(\widetilde{\mathbf{s}},\mathbf{s})} = \frac{p(\widetilde{\mathbf{s}})}{p(\mathbf{s})}$$

By the **Proposition** this shows that p is the limiting distribution

Hamiltonian Markov Chain (HMC)

The general Metropolis-Hastings algorithm converges very slowly (i.e. it takes many steps to reach the limiting distribution) and a much faster method to generate samples is the *Hamiltonian Monte Carlo*.

This is based on Hamiltonian Physics:

If a particle with mass= m is moving in a potential field $V(q)$, where q is the position of the particle, with momentum $= p$, the total energy of the particle is the potential energy $V(q)$, plus the kinetic energy $T = ||p||^2/2m$. For simplicity we take $m = 1$.

The Hamiltonian is the total energy $H(p,q) = V(q)+T(p)$ and the equations of motion in the coordinates of the phase space (p,q), are Hamilton's equations

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \frac{\partial T}{\partial p} \text{ and } \frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial V}{\partial q}$$

If the particle has coordinates (p,q) at time t and Δt is a small time increment then at time $t+\Delta t$ the coordinates in phase space are approximately

$$(p + \Delta t \frac{dp}{dt}, q + \Delta t \, \frac{dq}{dt}) = (p - \Delta t \frac{\partial V}{\partial q}, q + \Delta t \, \frac{\partial T}{\partial p})$$

In general we cannot solve the Hamiltonian equations to find the paths of motion so we must resort to numerical integration methods

How can we use this to generate a Markov Process with a specified limiting distribution?

Assume we have a dataset $\mathcal{D}$ and we hypthesize that the data are samples from a distribution $p(\mathbf{x}|\theta)$ with an unknown parameter vector θ.

We want to create samples from a posterior distribution

$$p_{post}(\theta|\mathcal{D})$$

to allow us to approximately compute things like Maximum Apostiori Probability (MAP) which is $\theta_0 = \underset{\theta}{\text{argmax}}\ p(\theta|\mathcal{D})$, mean and variance.

Choosing an appropriate prior $p_{prior}(\theta)$, we can use Bayes' formula

$$p_{post}(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)\, p_{prior}(\theta)}{p(\mathcal{D})}$$

Where the likelihood

$$p(\mathcal{D}|\theta) = \prod_{\mathbf{x}\in\mathcal{D}} p(\mathbf{x}|\theta)$$

As usual the denominator

$$p(\mathcal{D}) = \int p(\mathcal{D}|\theta) p_{prior}(\theta)\, d\theta$$

is intractable so we can only compute the posterior up to the normalization factor

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)\, p_{prior}(\theta)$$

θ plays the role of the position q

The potential field is defined by

$$V(\theta) = -\log\left(p(\mathcal{D}|\theta)\, p_{prior}(\theta)\right) = -\sum_{\mathbf{x}\in\mathcal{D}} \log p(\mathbf{x}|\theta) - \log p_{prior}(\theta)$$

If $\dim \theta = n$ the phase space $\{(p, q)\}$ is $2n$ dimensional $=\{\mathbf{r}, \theta\}$

Momentum is a vector $\mathbf{r}$ of the same dimension as θ and the Hamiltonian is

$$H(\mathbf{r}, \theta) = -\log(p(\mathcal{D}|\theta)\, p_{prior}(\theta)) + ||\mathbf{r}||^2/2$$

We lift the (unknown) posterior distribution $p(\theta|\mathcal{D})$ to a distribution on the phase space $\{\mathbf{r}, \theta\}$ by

$$p(\mathbf{r}, \theta) = p(\mathbf{r}|\theta)p(\theta|\mathcal{D})$$

and

$$p(\mathbf{r}|\theta) = \mathcal{N}(0, I)$$

If we have samples from this distribution $\{(\mathbf{r}, \theta)\}$ then the θ coordinates are samples from the marginal distribution

$$p(\theta) = \int p(\mathbf{r}, \theta) d\mathbf{r} = \int \mathcal{N}(\mathbf{r}|0, I) p_{post}(\theta|\mathcal{D}) d\mathbf{r} = p_{post}(\theta|\mathcal{D}) \int \mathcal{N}(\mathbf{r}|0, I) d\mathbf{r} = p_{post}(\theta|\mathcal{D})$$

We shall construct a Markov sequence on the phase space with limit distribution $p(\mathbf{r}, \theta)$.

So according to the Metropolis-Hastings formalism, we need a proposal distribution and an acceptance distribution.

How do we construct a Markov Chain in the phase space?

Assume we have constructed $\mathbf{s}_{m-1} = (\mathbf{r}_{m-1}, \theta_{m-1})$, and put $\theta^0 = \theta_{m-1}$.

We want to construct the next step.

The idea is to view this as a particle at position θ^0 at time t. At time t we give the system a push by generating a momentum $\mathbf{r}^0$ as a random sample from $\mathcal{N}(0, I)$

Using the Hamiltonian equations of motion with starting state $(\mathbf{r}^0, \theta^0)$, we can compute the (deterministic) position of the particle in the phase space at some later time $t + L$.

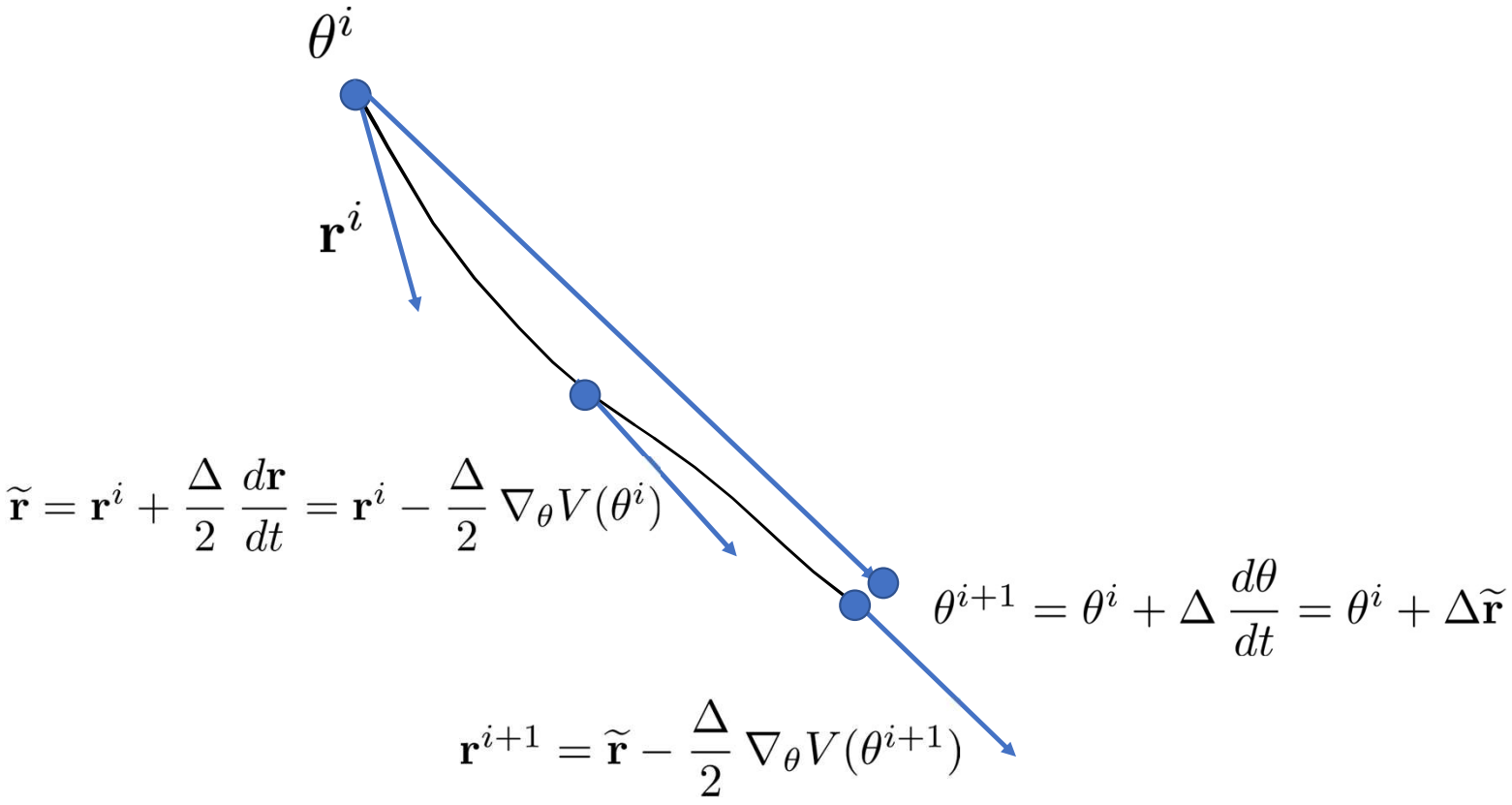
In general we cannot integrate the Hamiltonian equations analytically so we have to use numerical integration

We shall use the following *LeapFrog* algorithm.

We divide L into n small subintervals of length Δ, and let $n = \dfrac{L}{\Delta}$ be the number of these subintervals

1. Let $\mathbf{r}^0 \sim \mathcal{N}(0, I)$ and $\theta^0 = \theta_{m-1}$
2. For $i = 0, 1, \ldots, n-1$.
   If $(\mathbf{r}^i, \theta^i)$ has been computed, set $\widetilde{\mathbf{r}} = \mathbf{r}^i - \dfrac{\Delta}{2} \nabla_\theta V(\theta^i)$
3. Set $\theta^{i+1} = \theta^i + \Delta\, \widetilde{\mathbf{r}}$
4. Set $\mathbf{r}^{i+1} = \widetilde{\mathbf{r}} - \dfrac{\Delta}{2} \nabla_\theta V(\theta^{i+1})$
5. $(\mathbf{r}_m, \theta_m) = (-\mathbf{r}^n, \theta^n)$

One step in LeapFrog Integration in the interval $t + i\Delta, t + (i+1)\Delta$



The reason for negative sign is to make the proposal distribution symmetric.

Indeed the construction of $(\mathbf{r}^0, \theta^0) \to (\mathbf{r}_m, \theta_m))$ is deterministic so

$$p(\mathbf{r}', \theta' | \mathbf{r}^0, \theta^0) = \delta(\mathbf{r}' - \mathbf{r}_m)\, \delta(\theta' - \theta_m) = \begin{cases} 1 \text{ if } (\mathbf{r}', \theta') = (-\mathbf{r}^n, \theta^n) \\ 0 \text{ otherwise} \end{cases}$$

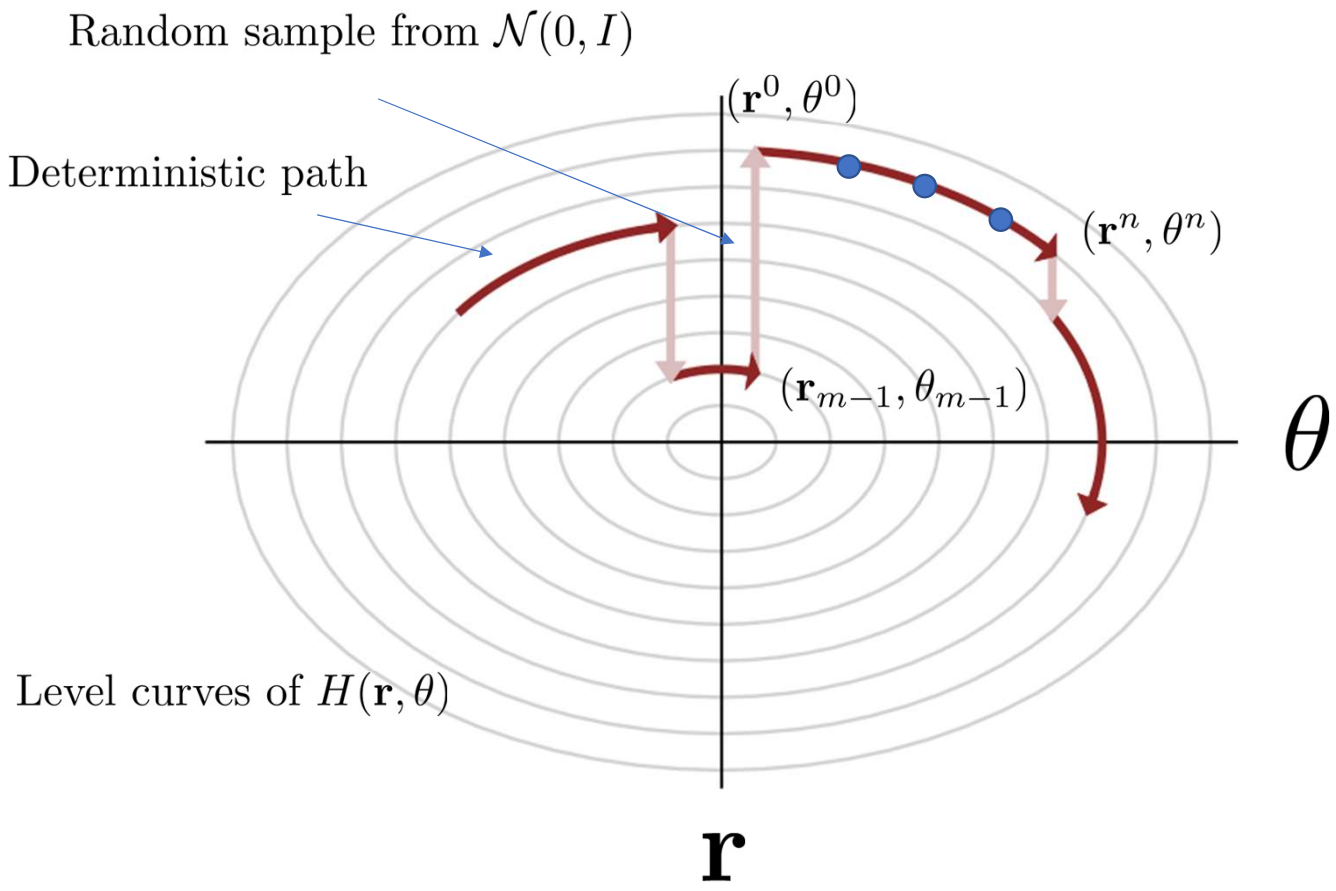
We have

$$p(\mathbf{r}_m, \theta_m | \mathbf{r}_{m-1}, \theta_{m-1})) = p(\mathbf{r}_m, \theta_m | \mathbf{r}^0, \theta^0) = 1$$

But it is clear that the LeapFrog is reversible, so if we start with $(-\mathbf{r}^n, \theta^n)$ we end up with $(-\mathbf{r}^0, \theta^0)$ and so going backwards in the process we get $(\mathbf{r}^0, \theta^0)$ (remark that we have to change sign on $\mathbf{r}^0$). Hence

$$p(\mathbf{r}_{m-1}, \theta_{m-1} | \mathbf{r}_m, \theta_m) = p(\mathbf{r}^0, \theta^0 | \mathbf{r}_m, \theta_m) = p(\mathbf{r}^0, \theta^0 | -\mathbf{r}^n, \theta^n) = \delta(\mathbf{r}^0 - \mathbf{r}^0)\delta(\theta^0 - \theta^0) = 1$$

.


Random sample from $\mathcal{N}(0, I)$
Deterministic path
Level curves of $H(\mathbf{r}, \theta)$
$(\mathbf{r}^0, \theta^0)$
$(\mathbf{r}^n, \theta^n)$
$(\mathbf{r}_{m-1}, \theta_{m-1})$
θ
$\mathbf{r}$

The acceptance probability is

$$
\begin{aligned}
A((\mathbf{r}_m, \theta_m), (\mathbf{r}^0, \theta^0)) &= \min\left(1, \frac{p(\mathbf{r}_m, \theta_m)}{p(\mathbf{r}^0, \theta^0)}\right) \\
&= \min\left(1, \frac{\mathcal{N}(\mathbf{r}_m; 0, I) p(\theta_m|\mathcal{D})}{\mathcal{N}(\mathbf{r}^0; 0, I) p(\theta^0|\mathcal{D})}\right) \\
&= \min\left(1, \frac{\mathcal{N}(\mathbf{r}_m; 0, I) \frac{p(\mathcal{D}|\theta_m) p_{prior})(\theta_m)}{p(\mathcal{D})}}{\mathcal{N}(\mathbf{r}^0; 0, I) \frac{p(\mathcal{D}|\theta^0) p_{prior}(\theta^0)}{p(\mathcal{D})}}\right) \\
&= \min\left(1, \frac{\mathcal{N}(\mathbf{r}_m; 0, I) p(\mathcal{D}|\theta_m) p_{prior})(\theta_m)}{\mathcal{N}(\mathbf{r}^0; 0, I) p(\mathcal{D}|\theta^0) p_{prior}(\theta^0)}\right)
\end{aligned}
$$

Now

$$V(\theta) = -\log p(\mathcal{D}|\theta)p_{prior}(\theta))$$

so

$$p(\mathcal{D}|\theta)p_{prior}(\theta)) = \exp(-V(\theta))$$

The Gaussian density is of the form

$$\mathcal{N}(\mathbf{r}|0, I) = const\ \exp(-\frac{\mathbf{r} \cdot \mathbf{r}^T}{2}) = const\ \exp(-\frac{||\mathbf{r}||^2}{2})$$

It follows that we can write

$$\begin{aligned}\mathcal{N}(\mathbf{r}|0, I)p(\mathcal{D}|\theta)p_{prior}(\theta) &= const\ \exp(-V(\theta) - \frac{||\mathbf{r}||^2}{2}) \\ &= const\ \exp(-H(\mathbf{r}, \theta))\end{aligned}$$

and finally

$$A(\mathbf{r}_m, \theta_m|\mathbf{r}_{m-1}, \theta_{m-1}) = A((\mathbf{r}_m, \theta_m), (\mathbf{r}_0, \theta_0)) = \min\left(1, \frac{\exp(-H(\mathbf{r}_m, \theta_m))}{\exp(-H(\mathbf{r}^0, \theta^0))}\right)$$

Sample an α from $Uniform([0,1])$.

If $A((\mathbf{r}_m, \theta_m), (\mathbf{r}^0, \theta_{m-1})) \geq \alpha$ we accept the new point $(\mathbf{r}_m, \theta_m)$ and if $A((\mathbf{r}_m, \theta_m), (\mathbf{r}^0, \theta_{m-1})) < \alpha$ we reject it and set $(\mathbf{r}_m, \theta_m) = (\mathbf{r}_{m-1}, \theta_{m-1})$ and sample another $\mathbf{r}^0$

From our discussion of the Metropolis-Hastings theory it follows that this defines a Markov Chain $\{(\mathbf{r}_m, \theta_m)\}$ with $(\mathbf{r}_i, \theta_i)$ a sample from $p(\mathbf{r}, \theta)$ for $i >> 0$.

Then θ_i for $i >> 0$ is a sample from the marginal distribution,

$$\int p(\mathbf{r}, \theta) d\mathbf{r} = p_{post}(\theta|\mathcal{D})$$

Generating HMC samples is very simple in `pyro`.

We can use our previous model for the GMM

```
# we import the appropriate pyro packages

from pyro.infer.mcmc.api import MCMC
from pyro.infer.mcmc import HMC,NUTS,MCMC
pyro.set_rng_seed(2)
kernel = NUTS(model)
mcmc = MCMC(kernel,num_samples=1000,warmup_steps=50)
mcmc.run(data)
posterior_samples = mcmc.get_samples()
```

This is literally all the code that is needed to generate 1000 samples of $\{\mu_0, \sigma_0^2, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2, \pi(0), \pi(1), \pi(2)\}$

It does take a longer time to run

```
Sample:  65%|                                                  | 681/1050 [06:45,  2.47it/s, step size=1.47e-02, acc. prob=0.888]
```

It first runs 50 warm-up iterations. Of course we do not know if the chain has *mixed* i.e. whether we have reached the limit distribution. The existence theorem unfortunately does not say anything about how many steps are needed to reach the limit distribution

The samples are stored in a `dict`

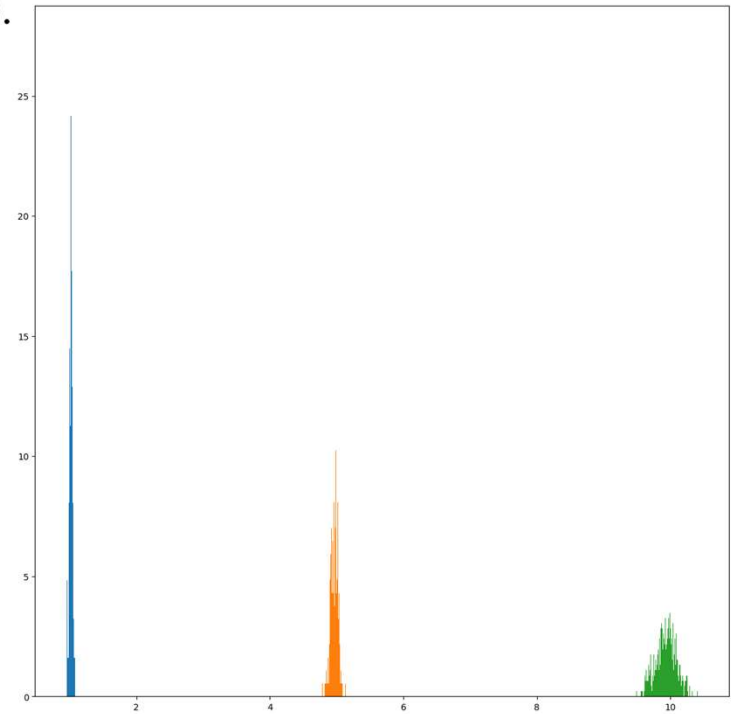
```
1 posterior_samples.keys()
```

```
dict_keys(['locs', 'scales', 'weights'])
```

We can plot the histograms for all the different parameters.

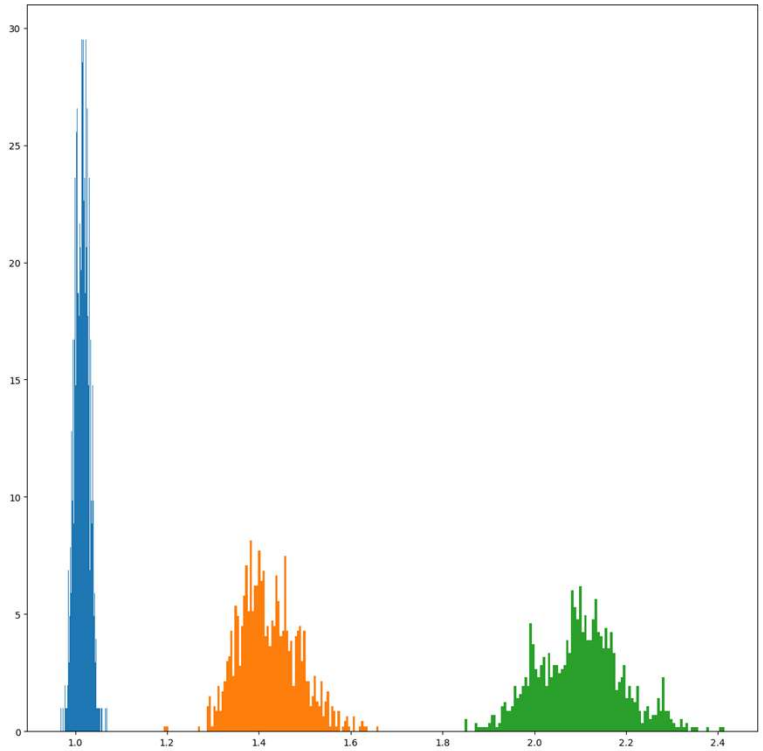
The distributions for the means have low variance

```
1 pyplot.hist(posterior_samples['locs'].numpy()[:,0],bins=100,density=True);
2 pyplot.hist(posterior_samples['locs'].numpy()[:,1],bins=100,density=True);
3 pyplot.hist(posterior_samples['locs'].numpy()[:,2],bins=100,density=True);
```

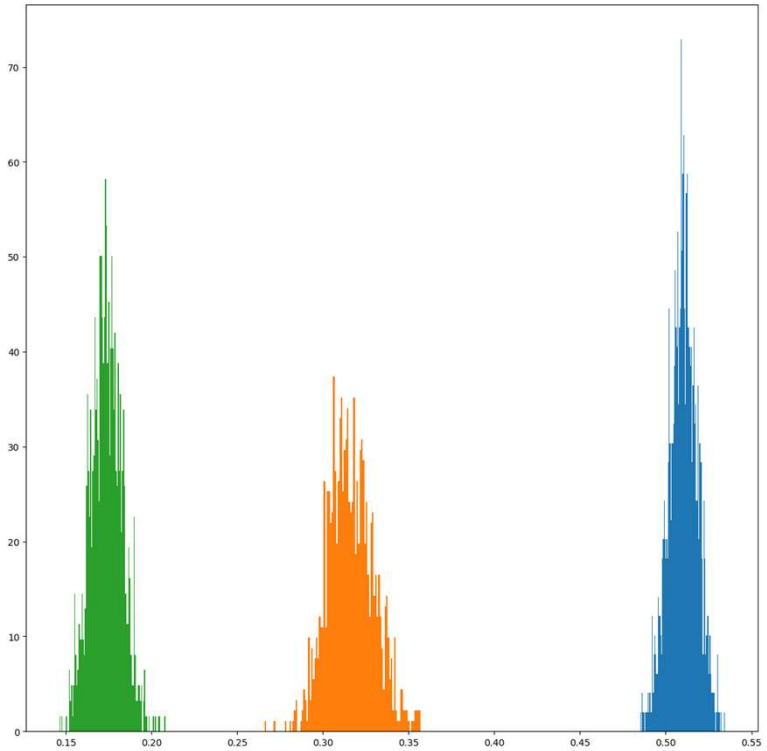


Histograms for Variance and Weights

Variances



Weights



The point estimates for the parameters are very close to the Variational Inference results

```
1 posterior_samples['locs'][500:].mean(0)
```

```
tensor([1.0240, 4.9632, 9.9358])
```

```
1 posterior_samples['scales'].numpy()[500:].mean(0)
```

```
array([1.0145655, 1.4315848, 2.0848947], dtype=float32)
```

```
1 posterior_samples['weights'].numpy()[500:].mean(0)
```

```
array([0.5091649 , 0.31857646, 0.17225857], dtype=float32)
```

Bayesian Neural Network

We shall do another example where we can use either MCMC or Variational Inference

A Neural Network is basically just a parametrized function that takes an input vector $\mathbf{x}$ and produces an output $\mathbf{y}$

$$\mathbf{y} = f(\mathbf{x}, \theta)$$

where θ denotes all the weights and biases of the NN.

A *Bayesian Neural Network* is a Neural Network where the parameters θ are random variables with some joint *distribution*.

This means that for each input vector, $\mathbf{x}$, the network outputs a *distribution* $f(\mathbf{x}, \theta)$, over the possible outputs.

If we want a point output, we can take the mean of the the distribution. The advantage is that we also get an estimate of the uncertainty of the output

Given a training set $\mathcal{D}$ and an input $\mathbf{x}$ we have the conditional distribution of the output

$$p(\mathbf{y}|\mathbf{x},\mathcal{D}) = \int p(\mathbf{y}|\mathbf{x},\theta)p(\theta|\mathcal{D})d\theta = \mathbb{E}_{p(\theta|\mathcal{D})}(p(\mathbf{y}|\mathbf{x},\theta))$$

If we have the posterior distribution $p(\theta|\mathcal{D})$ we can use Monte Carlo to estimate $p(\mathbf{y}|\mathbf{x},\mathcal{D})$

`pyro` does this for us automatically using the `pyro.infer.Predictive` class

Training the network then becomes the problem of determining the posterior distribution $p(\theta|\mathcal{D})$

We shall code a simple *denoising model*

We are given a noisy set of points from some curve in $\mathbb{R}^2$

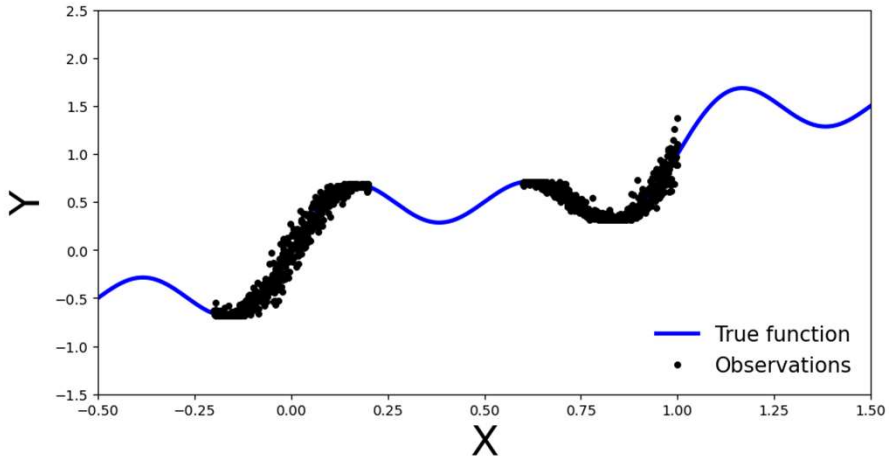
```
x_obs = np.hstack([np.linspace(-0.2,0.2,500),np.linspace(0.6,1,500)])
```

```
noise = 0.02 * np.random.randn(x_obs.shape[0])
```

```
y_obs = x_obs + 0.3 * np.sin(2 * np.pi * (x_obs+noise))+ 0.3 * \
                            np.sin(4 * np.pi * (x_obs+noise)) + noise
```

```
x_true = np.linspace(-0.5,1.5,1000)
```

```
y_true = x_true + 0.3*np.sin(2 * np.pi * x_true) + 0.3 * \
                            np.sin(4 * np.pi * x_true)
```



The network class is now derived from `PyroModule` rather than `nn.Module`

The two linear layers are now `PyroModules`

Weights and biases are samples from the prior distribution

This is basically the `pyro` model with prior and observations (we didn't actually need the return value `mu`

```
class BNN(PyroModule):
    def __init__(self,in_dim=1,out_dim=1,hid_dim=5,prior_scale=10.):
        super().__init__()

        self.activation = nn.Tanh()
        self.layer1 = PyroModule[nn.Linear](in_dim,hid_dim)
        self.layer2 = PyroModule[nn.Linear](hid_dim,out_dim)

        self.layer1.weight = PyroSample(dist.Normal(0.,prior_scale).expand([hid_dim,in_dim]).t
        self.layer1.bias = PyroSample(dist.Normal(0.,prior_scale).expand([hid_dim]).to_event(1

        self.layer2.weight = PyroSample(dist.Normal(0.,prior_scale).expand([out_dim,hid_dim]).
        self.layer2.bias = PyroSample(dist.Normal(0.,prior_scale).expand([out_dim]).to_event(1

    def forward(self,x,y=None):
        x = x.reshape(-1,1)
        x = self.activation(self.layer1(x))
        mu = self.layer2(x).squeeze()
        sigma = pyro.sample('sigma',dist.Gamma(.5,1))

        with pyro.plate('data',x.shape[0]):
            obs = pyro.sample('obs',dist.Normal(mu,sigma**2),obs=y)
        return mu

```

We can use HMC to generate samples from the posterior

We only generate 50 samples to save time

```
from pyro.infer import MCMC, NUTS

model = BNN()

pyro.set_rng_seed(42)

nuts_kernel = NUTS(model)

mcmc = MCMC(nuts_kernel,num_samples=50)

x_train = torch.from_numpy(x_obs).float()
y_train = torch.from_numpy(y_obs).float()

mcmc.run(x_train,y_train)
```

The inputs are the $\mathbf{x}$-coordinates. We take 3000 points on the x-axis.

The `Predictive` class generates 50 samples for each of the 3000 $\mathbf{x}$-values

```
from pyro.infer import Predictive
```

```
predictive = Predictive(model=model,posterior_samples=mcmc.get_samples())
x_test = torch.linspace(xlims[0],xlims[1],3000)
preds = predictive(x_test)
```

The predictions are returned in a `dict`

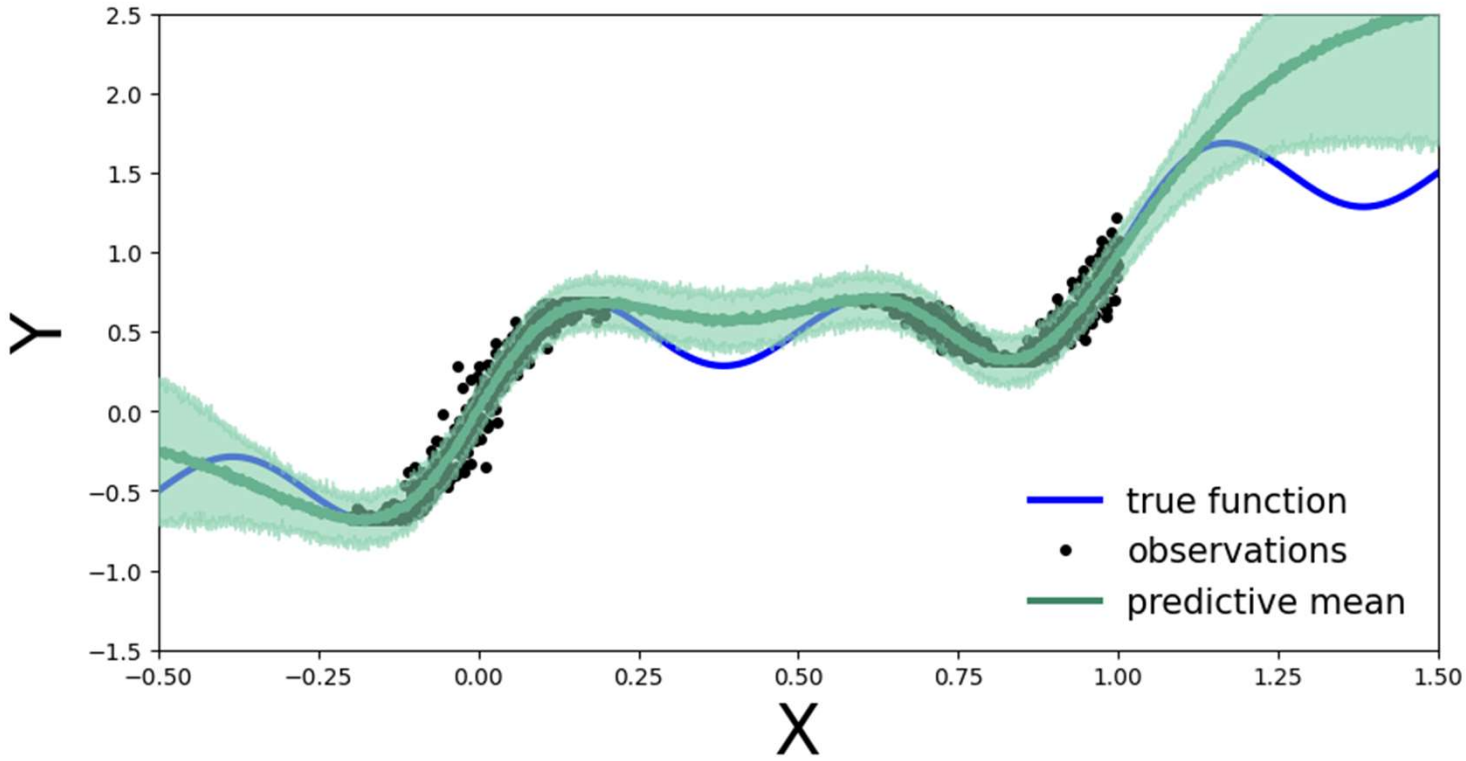
```
1 preds
```

```
{'obs': tensor([[-0.5782, -0.4745, -0.4330,  ...,  1.9330,  1.7531,  1.8977],
        [-0.3904, -0.3693, -0.2173,  ...,  2.0469,  2.0577,  2.0436],
        [-0.4403, -0.5386, -0.5305,  ...,  1.8439,  1.7718,  1.7824],
        ...,
        [-0.5244, -0.3689, -0.5213,  ...,  3.2782,  3.2007,  3.4162],
        [-0.3312, -0.3256, -0.4390,  ...,  3.2692,  3.1728,  3.2694],
        [-0.2058, -0.4267, -0.4103,  ...,  3.2530,  3.3596,  3.3291]])}
```

```
1 preds['obs'].shape
```

```
torch.Size([50, 3000])
```

Plotting the *means* with a 2 *std* band



Using Variational Inference with a `guide` (variational distribution) = product of Gaussians (this means that we consider the parameters as being independent, often called a *mean-field variational distribution*)

```
model = BNN()
mean_field_guide = AutoDiagonalNormal(model)
optimizer = pyro.optim.Adam({'lr':0.01})
```

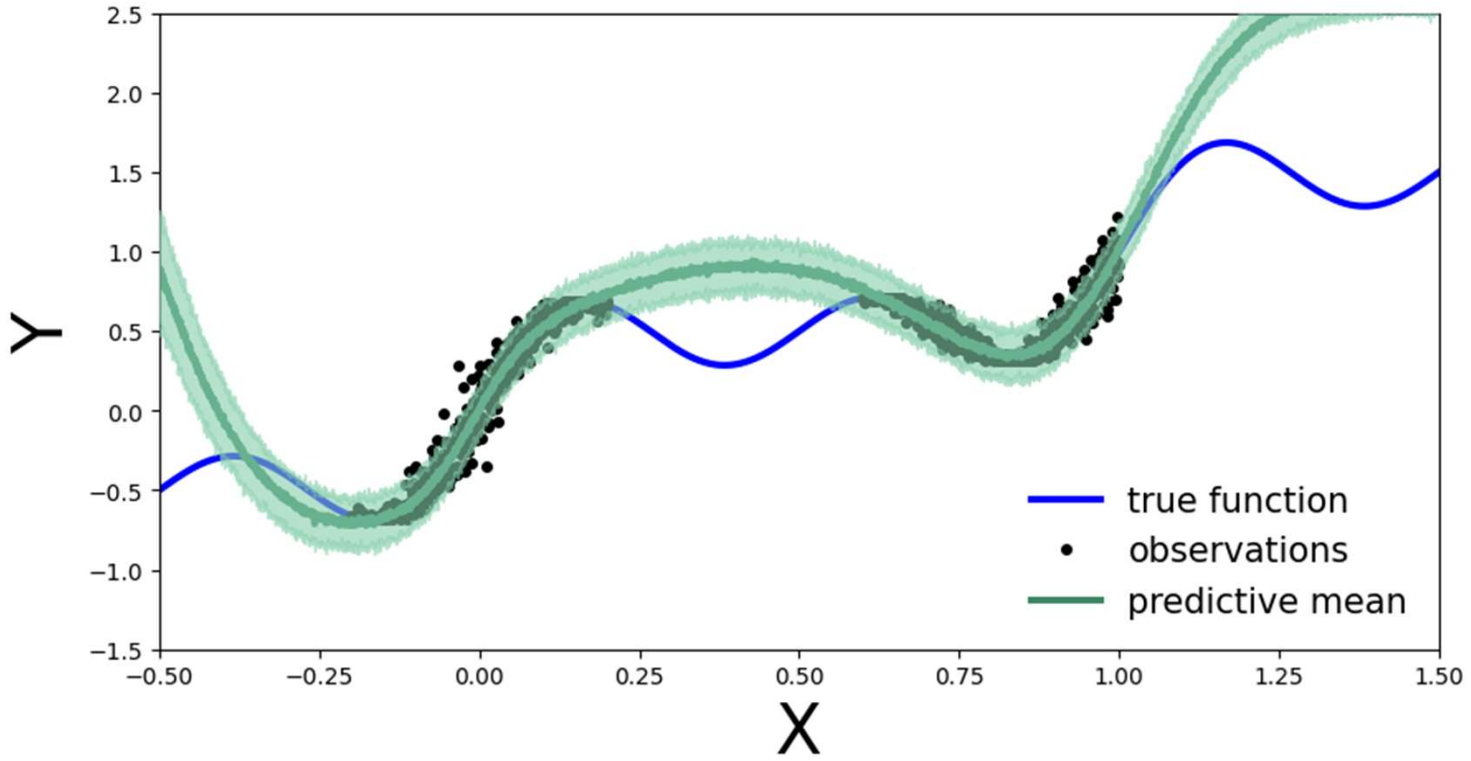
```
svi = SVI(model,mean_field_guide,optimizer,loss=Trace_ELBO())
```

```
num_epochs = 25000
progress_bar = trange(num_epochs)
```

```
  0%|                                                                  | 0/2
5000 [00:00<?, ?it/s]
```

```
for epoch in progress_bar:
    loss = svi.step(x_train,y_train)
    progress_bar.set_postfix(loss=f"{loss /x_train.shape[0]:.3f}")
```

```
100%|██████████████████████████████████████████████████████████| 25000/25000 [05:00<00:00, 8
3.26it/s, loss=-1.006]
```


2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
Y
X
true function
observations
predictive mean