FINAL PROJECT

CS 615: Deep Learning

Hamiltonian Monte Carlo - Bayesian Neural Network

Mathilda

qtn35@drexel.edu

September 2023

Hamiltonian Monte Carlo belongs to the family of Markov Chain Monte Carlo (MCMC) method. MCMC can approximate some distribution which can be hard to sample from, by constructing a markov chain in such a way that our desired distribution is the chain's stationary distribution. This is done with the help of Metropolis-Hastings (MH) algorithm. Hamiltonian Monte Carlo is built upon Metropolis-Hasting MCMC: by switching out one of MH processes with Hamiltonian dynamics, it's found out that we can overcome MH's weaknesses and speed up the sampling process.

I found HMC to be an intriguing method - a beautiful amalgamation of many beautiful ideas: Markov Chain and Monte Carlo methods (from probability theory and statistics), Hamiltonian dynamics and Leapfrog integration (from classical mechanics), coming about to solve the very hard problem of multidimensional integration. Because of this, I chose the topic of Hamiltonian Monte Carlo sampled Bayesian Neural Network. More on this later.

ON SUBJECT OF THE CODE: Originally I intend to have this project be: using HMC-BNN to do classification on galaxy morphology images. There were a few foreseen setbacks: the dataset is too big, and HMC-BNN doesn't scale well. However, it turned out that I also overestimated my ability to understand and code it up within reasonable time. With that out of the way, my project turns out to be: **"Let's see what I can do with HMC-BNN and how it fares in terms of tackling basic regression and classification problem"**

Contents

ACKNOWLEDGEMENTS AND CITATIONS **4**
0.1 HAMILTONIAN MONTE CARLO section . . . 4
0.2 BAYESIAN NEURAL NETWORK section . . . 4
0.3 CODE EXAMPLE section . . . 4
0.4 General notes . . . 4

HAMILTONIAN MONTE CARLO **5**

1 Monte Carlo method and Markov Chain **5**
1.1 Defining Monte Carlo method . . . 5
1.2 Intuitive explanation of a Markov Chain . . . 5
1.3 Overview of Markov Chain and it's role in MCMC . . . 5
1.4 Some use cases . . . 6

2 Metropolis-Hastings **7**
2.1 General Metropolis-Hastings . . . 7
2.2 Random walk Metropolis . . . 7
2.3 Pseudocode . . . 8
2.4 Advantages . . . 8
2.5 Drawbacks . . . 8

3 Hamiltonian Monte Carlo **8**
3.1 The Hamiltonians . . . 8
3.2 Hamiltonian Monte Carlo . . . 9
3.3 On numerical approximation of the Hamiltonian and implementation of HMC . . . 9
3.4 Pseudocode . . . 10
3.5 Advantages . . . 10
3.6 Drawbacks . . . 10

4 Hamiltonian Monte Carlo on infinite dimension **10**
4.1 Hilbert space HMC . . . 10
4.2 Pseudocode . . . 10
4.3 Advantages . . . 11
4.4 Drawbacks . . . 11

5 Appendix A **12**
5.1 Parts of a Markov Chain . . . 12
5.2 Hilbert space . . . 13
5.3 HMC cartoon . . . 13

BAYESIAN NEURAL NETWORK **14**

6 Bayesian Neural Network **14**
6.1 Neural Network as function approximator, and glosarry of terms . . . 14
6.2 Traditional Neural Network . . . 15
6.3 Bayesian Neural Network . . . 15

7 Hamiltonian Monte Carlo sampled Bayesian Neural Network - an in-depth look into implementation **16**
7.1 Similarities between Traditional and Bayesian Neural Network 16
7.2 Traditional forward-backward neural network 17
7.3 Hamiltonian Monte Carlo sampled Bayesian Neural Network 17

8 Going infinity **18**

9 Appendix B **19**
9.1 Neal's HMC pseudocode 19

CODE EXAMPLE **20**

10 Foreword **20**

11 Classification - MNIST 100 **20**
11.1 Architecture and parameters 20
11.2 Traditional result 20
11.3 BNN result 21
11.4 Key takeaway 21

12 Regression - Boston Housing **25**
12.1 Architecture and parameters 25
12.2 Traditional result 25
12.3 BNN result 25
12.4 Key takeaway 25

ACKNOWLEDGEMENTS AND CITATIONS

0.1 HAMILTONIAN MONTE CARLO section

To fully grasp the theory behind HMC, it's best to have knowledge of probability theory and analysis, however, my notes are made so that it hopefully can be read by more or less any audience with some understanding of undergraduate real analysis. Most definitions not directly related to MCMC, but is beneficial for the understanding of which, will be presented in appendix A.

The theories in section 1 and 2 are reliant on the 1995 paper Understanding the Metropolis-Hastings algorithm by Chib and Greenberg.

Section 3 and 4 are reliant on (1) the 2012 papers MCMC using Hamiltonian dynamics by Radford Neal, (2) the 2018 paper A Conceptual Introduction to Hamiltonian Monte Carlo by Michael Betancourt, and (3) the 2011 paper Hybrid Monte Carlo on Hilbert Spaces by Beskos et al.

Most of Appendix A, section 1, are from MCMC lecture note by Shimkin, except for the part on transition kernel and markov kernel, which definitions were taken from Klenke's Probability Theory book. This appendix are mainly used to present mathematical properties of importance to MCMC.

0.2 BAYESIAN NEURAL NETWORK section

The main spark of HMC-BNN project comes from the paper What Are Bayesian Neural Network Posteriors Really Like? by Izmailov et al (2021) - the structure of which (which datasets, what is possible, how to tweak parameters) is used to produce materials in this report. But the actual coding of the Neural Network is from following Tufts' course "Bayesian Deep Learning".

Note that section 6.2-6.3 relies heavily on a post by Prof Duvenaud from UToronto

Section 8 is mainly a discussion on the 2023 paper Trace-class Gaussian priors for Bayesian learning of neural networks with MCMC by Sell and Singh.

0.3 CODE EXAMPLE section

Datasets are (1) Boston housing and (2) MNIST 100. Both can be easily found on the internet.

0.4 General notes

The TeX template used is by Kevin Zhou)

HAMILTONIAN MONTE CARLO

1 Monte Carlo method and Markov Chain

1.1 Defining Monte Carlo method

Interestingly, according to Wikipedia, there are no consensus on how Monte Carlo should be defined.

Professor Xiaojin Zhu of University of Wisconsin-Madison, in his lecture notes on MCMC, put the definition of Monte Carlo in the context of a machine learning problem, stating that monte carlo is a method which approximate the expectation μ of a function $\phi(x)$, defined as

$$\mu \equiv \mathbb{E}_p[\phi(x)] = \int \phi(x)p(x)dx$$

with some sample average

$$\hat{\mu} \equiv \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)$$

for $x_1...x_n \sim p(x)$, with $p(x)$ the distribution we wish to sample from.

1.2 Intuitive explanation of a Markov Chain

Simplistically speaking, this is just a state-transition model, where one state reaches another according to some probability. It can be visualized as the figure to the right: there are three states a, B, and C. If currently we are at state a, there are 90% chances of us transitioning to state C, and 10% chances to state B. This markov chain has the transition probability matrix P, also presented in the figure.

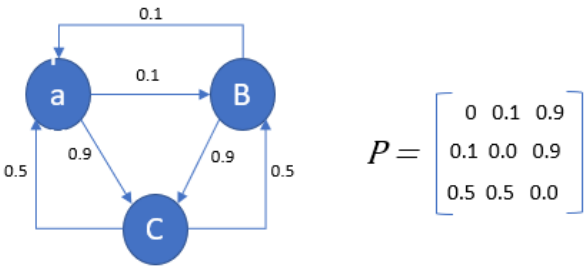

Figure 1: Visualized markov chain and it's transition matrix P

1.3 Overview of Markov Chain and it's role in MCMC

A markov chain can be defined as a collection of random variables which obeys the markov property - a property which asserts that the next state is dependent on the current state, and not any states before that. The probability of moving from one state to another is defined by a transition matrix in the discrete case, or transition kernel in the continuous case. Markov chain can have an invariant distribution. Once the chain reached the invariant distribution, it stays there.

For these notes, we will denote the transition kernel $P(x, A)$, for $x \in \mathcal{R}^d$, $A \in \mathcal{B}$, where $\mathcal{B}$ is the $\sigma-$field on $\mathcal{R}^d$. Moreover, since transition kernels are probability distributions, $P(x, \mathcal{R}^d) = 1$. It's permissible that $P(x, \{x\})$ be nonzero.

We denote the invariant distribution π^*, and denote π the density w.r.t Lebesgue measure of π^* (i.e. $\pi^*(dy) = \pi(y)dy$). The invariant distribution satisfy

$$\pi^*(dy) = \int_{\mathcal{R}^d} P(x, dy)\pi(x)dx$$

The nth iteration of the chain is

$$P^{(n)}(x, A) = \int_{\mathcal{R}^d} P^{(n-1)}(x, dy)P(y, A)$$

.

If the chain is aperiodic and irreducible, $P^{(n)}$ would approach the invariant distribution as n approach infinity.

Normally, it is the case that the transition kernel P that is known, and the invariant density $\pi(\cdot)$ is what we're computing. MCMC flips this around: it is the invariant density that is "known", and we seek to construct a kernel that would approximate this distribution.

Suppose $P(x, A)$ has the form

$$P(x, y) = p(x, y)dy + r(x)\delta_x(dy)$$

- $p(x, y)$: the function with $P(x, A)$ as the transition kernel. $p(x, x) = 0$.
- $\delta_x(dy) = 1$ if $x \in dy$ and 0 otherwise
- $r(x)$: the probability that the chain remains at x

If we can find $p(x, y)$ which satisfied reversibility, then $\pi(\cdot)$ is the invariant density of $P(x, \cdot)$ and our job is done. This is where Metropolis-Hastings comes in.

1.4 Some use cases

From what I've seen, MCMC is mostly used for:

(1) simulation of values of parameter's posterior / generating draws from posteriors and
(2) use that to infer properties of that posterior.

This lends MCMC to situations such as:

- Bayesian inference: given data and prior, calculate posterior of parameters
- Parameter estimation: generate draws from posterior to estimate parameter values
- Time series analysis: generate samples from posterior to make prediction about future events
- Estimating integrals: MCMC generate samples from some distribution $p(x)$ to calculate $\mathbb{E}[f(x)] = \int f(x)p(x)dx \approx (1/n) * \sum f(x_i)$ (or the empirical average of the samples)
- Optimization

2 Metropolis-Hastings

Whenever an MCMC method is employed, and if there are no further specification of which MCMC method it is, then chances are, the MCMC method in question is Random walk Metropolis - or that's what I used to think. From my recent experience, it seem like people tend to say "we use MCMC" without further clarification on *which* MCMC it is - if you want to know, you have to hunt down that information

2.1 General Metropolis-Hastings

We approximate $p(x, y)$, $x \neq y$, by some p_{MH} where

$$p_{MH}(x, y) \equiv q(x, y)\alpha(x, y)$$

.

The candidate-generating density $q(x, y)$ satisfies $\int q(x, y)dy = 1$. This density generate a value y when a process is at point x. If $q(x, y)$ satisfies reversibility condition $\pi(x)q(x, y) = \pi(y)q(y, x)$, then $p(x, y) = p_{MH}(x, y) = q(x, y)$ and our search is over. However, in most cases, one side of the equation might dominate the other, giving us, for example,

$$\pi(x)q(x, y) > \pi(y)q(y, x)$$

Which roughly translates into: the possibility of x going to y is higher than that of y going to x. There is where $\alpha(x, y)$ comes in. This is some "probability of move" function which would balance out the equation above, giving us

$$\pi(x)q(x, y)\alpha(x, y) = \pi(y)q(y, x)$$

- $\alpha(x, y) < 1$
- $\alpha(x, y) = min\left[\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right]$ if $\pi(x)q(x, y) > 1$. And $\alpha(x, y) = 1$ otherwise

Now, $r(x) = 1 - \int_{\mathcal{R}^d} q(x, y)\alpha(x, y)dy$. And we got the transition kernel

$$P_{MH}(x, dy) = q(x, y)\alpha(x, y)dy + \Big[1 - \int_{\mathcal{R}^d} q(x, y)\alpha(x, y)dy\Big]\delta_x(dy)$$

2.2 Random walk Metropolis

There are many choices for the family of distribution for $q(x, y)$, but the most popular one is a Gaussian distribution. This is largely due to the fact that the symmetry of the distribution would cause $q(x, y) = q(y, x)$, and so, $\alpha(x, y) = \frac{\pi(y)}{\pi(x)}$. Aside from increasing implementability, choosing Gaussian distribution also enable M-H to produce more desirable proposals. According to Betancourt (2018): *The [Gaussian] proposal distribution is biased towards large volumes, and hence the tails of the target distribution, while the Metropolis correction rejects those proposals that jump into neighborhoods where the density is too small. The combined procedure then preferentially selects out those proposals that fall into neighborhoods of high probability mass, concentrating towards the typical set as desired.*

Metropolis-Hasting algorithm with this choice of proposal is called "Random walk Metropolis"

2.3 Pseudocode

Algorithm 1 Metropolis-Hasting

for j=1,2,...,N **do**:
    $y \sim q(x^{(j)}, \cdot)$
    $u \sim U(0, 1)$
    if $u \leq \alpha(x^{(j)}, y)$ **then**
        $x^{(j+1)} = y$
    else
        $x^{(j+1)} = x^{(j)}$
return $x^{(1)}, x^{(2)}, ..., x^{(N)}$

2.4 Advantages

- It's a start
- Easy to implement, yet powerful, can be used to tackle extremely hard problem
- Still widely used
- The basis of other more powerful sampling method (such as Gibbs, HMC, Slice sampling, etc... all differs from MH in the way it generate proposals)

2.5 Drawbacks

- Slow to converge, mostly due to the random walk
- In "MCMC using Hamiltonian dynamics" - "the benefit of avoiding random walk" section, Neal illustrated why random walk makes sampling worse. In particular, it causes larger correlation between samples, negatively affecting efficiency of MCMC method. Hamiltonian Monte Carlo manages to avoid this random walk behavior of Metropolis-Hastings.

3 Hamiltonian Monte Carlo

3.1 The Hamiltonians

In general, the Hamiltonian equations describes a system based on it's position q and momentum p, as well as their corresponding functions: the potential energy $U(q)$ and kinetic energy $K(p)$. The Hamiltonian $H(q, p)$ is the sum of the potential energy and kinetic energy, and is described by the system of differential equation

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \tag{1}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \tag{2}$$

For Hamiltonian Monte Carlo, this is usually written as $H(q, p) = U(q) + K(p)$

Hamiltonian dynamics is attractive for MCMC in that it is **reversible** (which leave the desired distribution invariant), **keeps the Hamiltonian invariant** (which makes possible the Metropolis update step), **volume-preserving** (which affords us to not care about computing the Jacobian matrix brought about by Metropolis updates), and **symplectic** (a stronger condition to volume

preservation). Of all the properties presented, the second property is the only one which can only be approximated as the Hamiltonian be discretized, the rest can be maintained exactly.

3.2 Hamiltonian Monte Carlo

Firstly, it's worth noting that HMC is used to sample only from continuous distribution on R^d.[1]

To use HMC, the desired distribution's density is translated to $U(q)$, with q correspond to the variable of interest. Note that $U(q)$ is equals to minus log of probability density of qs. Momentum variables were introduced and resampled (generally from a Gaussian distribution[2]) each Monte Carlo iteration. Proposals were presented through computing the Hamiltonian dynamics.

HMC works thanks to a concept known as the canonical distribution - an equation which describe a distribution over states of a system as a function of some energy function over those states.[3]

3.3 On numerical approximation of the Hamiltonian and implementation of HMC

To implement HMC, the Hamiltonian must be discretized. The leapfrog method, a type of symplectic integrators, is the most widely used method. Regardless, proper tuning is required for leapfrog to perform efficiently: if the number of step size ϵ is chosen to be too small, HMC essentially behaves like random walk metropolis, and if number of steps T/ϵ (with T being integration time) are taken too large, we waste computation.

Algorithm 2 Leapfrog

$q_0 = q, p_0 = p$
for n=0,1,...,T/ϵ **do**:
    $p_{n+\frac{1}{2}} = p_n - \frac{\epsilon}{2}\frac{\partial V}{\partial q}(q_n)$
    $q_{n+1} = q_n + \epsilon p_{n+\frac{1}{2}}$
    $p_{n+1} = p_{n+\frac{1}{2}} - \frac{\epsilon}{2}\frac{\partial V}{\partial q}(q_{n+1})$
return q_{n+1}, p_{n+1}

Hoffman and Gelman's 2011 paper introduced the No-U-Turn sampler, which negates the need for the user to set the number of step. This method is used in the Stan package, which is a popular package for HMC.

So far, research in adaptive stepsize for leapfrog doesn't produce very great result[4]. Skeel & Gear, in their 1992 paper paper indicates that a variable step size might ruin a symplectic integrator.

[1] for discrete case, maybe it's better to use Gibb?...There are actually a paper which modified HMC to sample from a discontinuous densities, but reviews were not favourable. See stan forum

[2] Livingston et al wrote a paper Kinetic energy choice in Hamiltonian/hybrid Monte Carlo that discuss how kinetic energy choice affects HMC's stability and convergence

[3] Note that there exist an HMC method called Microcanonical HMC which has a fixed energy Hamiltonian dynamic. This method is useful when dealing with problems where the total energy is a conserved quantity

[4] See Richardson & Finn (2011), Okudo & Suzuki (2017), and Du-ruisseaux et al (2021)

Under certain circumstances, leapfrog might not be the best numerical integrator for HMC. Under certain conditions which necessitates leapfrog's stepsize to be very small, the midpoint method can be employed with better results, as demonstrated by Pourzanjani and Petzold.

3.4 Pseudocode

Note: $\Psi_h^{(T)}$ is a geometric integrator, and M is the mass-matrix.

Algorithm 3 HMC on $\mathbb{R}^N$

1: $q^{(0)} \in \mathbb{R}^N$
2: n = 0
3: **for** n=0,1,... **do**:
4: $p^{(n)} \sim N(0, M)$
5: $(q^*, p^*) = \Psi_h^{(T)}(q^{(n)}, p^{(n)})$
6: $a = min(1, exp(H(q^{(n)}, p^{(n)}) - H(q^*, p^*)))$
7: $q^{(n+1)} = q^*$ with probability a, otherwise $q^{(n+1)} = q^{(n)}$

3.5 Advantages

Leveraging the structure of the density of the desired distribution, provides proposals with higher acceptance rate, which reduces correlation and slow convergence rate caused by random walk

3.6 Drawbacks

- Still bad when scaling dimensionality
- Bad with distributions with discontinuities
- Computational complexity (gradients computation)
- Tunings parameters[5]

4 Hamiltonian Monte Carlo on infinite dimension

4.1 Hilbert space HMC

A generalized HMC algorithm that works for sampling from measures which are finite-dimensional approximation of some measure π, which has density w.r.t a Gaussian measure π_0 on an infinite-dimensional Hilbert space.

$$\frac{d\pi}{d\pi_0}(q) \propto \exp(-\phi(q))$$

4.2 Pseudocode

Note that:

[5]Some HMC methods exists now which permits less user interference for setting up HMC parameters, one popular example is NUTS

$$\Delta H(q_0, v_0) = \phi(q_I) - \phi(q_0) + \frac{h^2}{8}(|\ \mathcal{C}^{\frac{1}{2}} f(q_0)\ |^2 - |\ \mathcal{C}^{\frac{1}{2}} f(q_I)\ |^2)$$
$$+ h \sum_{i=1}^{I-1} < f(q_i), v_i > + \frac{h}{2}(< f(q_0), v_0 > + < f(q_I), v_I >)$$

and

$$a(q, v) = min(1, exp(-\Delta H(q, v))$$

For $I = T/h$ steps.

Algorithm 4 Hamiltonian Monte Carlo on Hilbert space

$q^{(0)} \sim \Pi_0$
for n=0,1,2,...,N **do**:
    $v^{(n)} \sim N(0, \mathcal{C})$
    $(q^*, v^*) = \psi_h^{(T)}(q^{(n)}, v^{(n)})$
    $a = a(q^{(n)}, v^{(n)})$
    if $u = U(0, 1) \leq a$ **then**
        $q^{(n+1)} = q^*$
    else
        $q^{(n+1)} = q^{(n)}$
return $x^{(1)}, x^{(2)}, ..., x^{(N)}$

4.3 Advantages

Measures of this form arises i n conditioned diffusion and the Bayesian approach to inverse problems. Citing Stuart, 2010

4.4 Drawbacks

General HMC drawbacks still apply

5 Appendix A

When I first encountered MCMC, there were no resources that are friendly to people who did not have knowledge of higher mathematics. It is an appendix like this that I wish was available to me at the time. The below notions are presented mainly for the sake of reference. However, hopefully, through providing these, I can make MCMC more accessible for people who are in the same position as I were.

5.1 Parts of a Markov Chain

The Markov property

$\mathbb{P}(X_n = x_n \mid X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}) = \mathbb{P}(X_n = x_n \mid X_{n-1} = x_{n-1})$

In other words: the probability of reaching the current state only depends on the state previous to it, and not those further in the past. Because of this, the markov property is also called **memoryless**.

Transition matrix/ transition probability

$\mathbb{P} = (p_{ij})_{i,j \in X}$ is what defined a (in this case, discrete-time and time-homogeneous) markov chain, which must satisfies three properties

(1) $\mathbb{P}(X_{t+1} = j \mid X_t = i) = p_{ij}$ (the markov property)

(2) $p_{ij} \geq 0$

(3) $\sum_j p_{ij} = 1 \forall i$

Transition kernel

Let $(\Omega_1, \mathcal{A}_1)$, $(\Omega_1, \mathcal{A}_1)$ be measurable spaces. A map $\kappa : \Omega_1 \times \mathcal{A}_2 \to [0, \infty]$ is called a $(\sigma-)$finite transition kernel (from Ω_1 to Ω_2) if:

(i) $\omega \mapsto \kappa(\omega_1, A_2)$ is $\mathcal{A}_1-$measurable for any $A_2 \in \mathcal{A}_2$

(ii) $A_2 \mapsto \kappa(\omega_1, A_2)$ is a $(\sigma-)$finite measure on $(\Omega_2, \mathcal{A}_2)$ for any $\omega_1 \in \Omega_1$

Stochastic/Markov kernel

If in (ii) above, the measure is a probability measure for all $\omega_1 \in \Omega_1$, then κ is called a stochastic kernel or a markov kernel.

Recurrence

Let $T_j = \inf(t \geq 1\text{: } X_t = j)$ be the first time that the chain hits state i. State i is *recurrence* if $\mathbb{P}(T_j < \infty \mid X_0 = j) = 1$. (Otherwise the state i is *transient*).

In other words: the process returns to state j with probability 1 if j is the start state.

Periodicity

A period d_i of a state i is smallest integer $d \geq 1$ s.t

$$\mathbb{P}(X_t \text{ for some } t \notin \{nd, n \geq 0\} \mid X_0 = i) = 0$$

If $d_i = 1$, the state is **a-periodic**.

In other words: periodicity tell us that the chain would return to a certain state after certain intervals

Irreducibility

State j is accessible from state i (written $i \to j$) if $(P^T)_{ij} \geq 0$ for some $t \geq 1$.

States i and j communicate if $i \to j$ and $j \to i$

The chain is irreducible if all states communicate with each others.

In other words: in an irreducible Markov chain, every state can be reached from every other states.

Ergodicity

A state is ergodic if it is (positive) recurrent and a-periodic. A markov chain is said to be ergodic if all its sates are ergodic.

Reversible Markov chain

A chain is reversible if $\exists \pi$, a probability vector, s.t $\pi p_{ij} = \pi_j p_{ij} \forall i, j$

Convergence of a Markov Chain

If a Markov chain is irreducible and a-periodic, the limit

$$\lim_{t\to\infty} \mathbb{P}(X_t = j \mid X_0 = i) \equiv \lim_{t\to\infty} P_{ij}^t \pi_j^\infty$$

exists $\forall i, j$.

In other words: provided that the chain is irreducible and a-periodic, then (1) the stationary distribution π exists, (2) is unique to the chain, and (3) coincides with π^∞.

Stationary/equilibrium/invariant distribution of a Markov Chain

A probability vector π (i.e $\pi_i \geq 0$, $\Sigma_i \pi_i = 1$) where $\pi = \pi P$.

In other words: the transition probability preserves the stationary distribution

5.2 Hilbert space

A Hilbert space is a vector space H with an inner product $< f, g >$ such that the norm defined by $f = \sqrt{(< f, f >)}$ turns H into a complete metric space.

5.3 HMC cartoon

An amazing visualization/cartoon of HMC at work can be found here.

BAYESIAN NEURAL NETWORK

6 Bayesian Neural Network

6.1 Neural Network as function approximator, and glosarry of terms

It was established in Hornik et al's 1989 paper Multilayer Feedforward Networks are Universal Approximators that "*standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universul approximators.*"

With that being said, we introduce a list of notations that will now be used henceforth:

- $f(W, X, B)$: neural network "functional" form
- $W, B \in \mathbb{R}^{\mathrm{N}}$: neural network parameters (weight and bias)
- $\mathcal{D} = \{x_i, y_i\}$: datasets of x_i (inputs) and y_i (targets)
- σ: activation (squashing) function

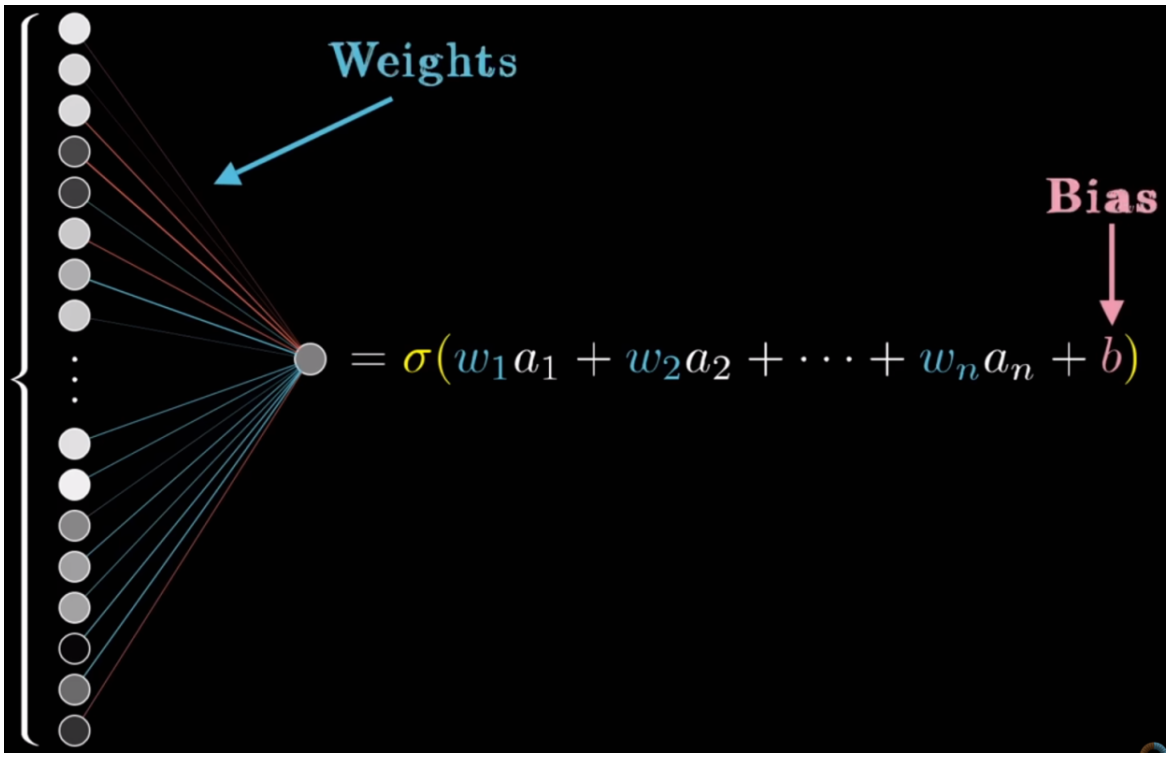

Figure 2: Visualization of the workings of Neural Network at one layer (3blue1brown video)

In a neural network, the output of layer l is (as shown in Figure 2)

$$a_j^{(l)} = \sigma\big(\sum w^{(l-1)} a^{(l-1)} + b^{(l)}\big)$$

6.2 Traditional Neural Network

Traditionally, neural network training implies finding the set of weights W that minimize the difference between the prediction and the true data value. This process is essentially a minimization of a certain Loss function $L(D, w)$. This difference is then back-propagated through the network, slowly adjusting the parameters towards the most desirable settings.

$$L(D, w) := \sum_{x_i, y_i \in D} (y_i - f(x_i, w))^2 + \lambda \sum_d w_d^2$$

Loss of our model on dataset D

is squared error between the target label y_i and output of network with input x_i and weights w

hyperparameter λ sets importance of regularizing large weight values w_d.

Figure 3: Loss function (source)

After training, predictions can be obtained using $y = f(x, W_{\text{trained}}, B_{\text{trained}})$

6.3 Bayesian Neural Network

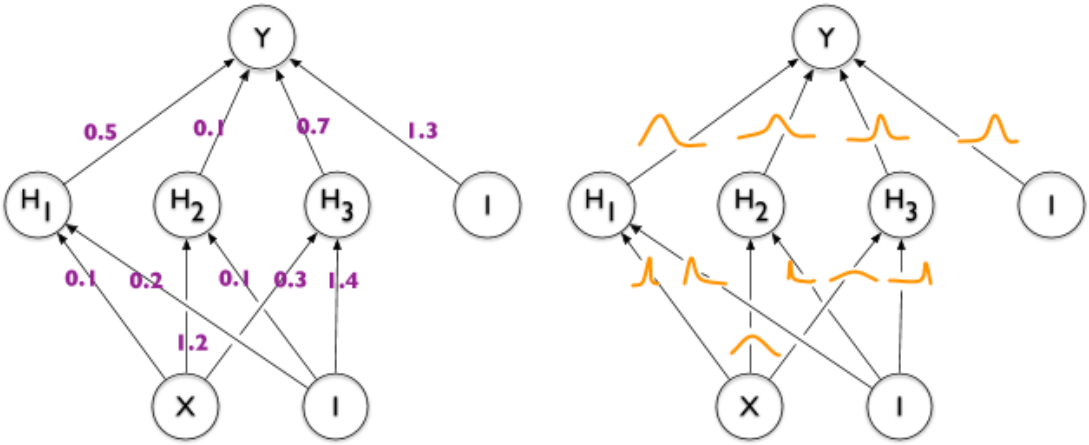

Figure 4: High level representation of
Traditional NN (left - point estimates) vs BNN (right - probability distribution)

Bayesian Neural Network (BNN) returns a probability distribution over predictions, instead of point estimates. This allows for uncertainty estimation, model selection, and reduces overfitting. Izmailov paper also asserts that, generally, BNN has better

The Bayesian settings of neural network is possible thanks to the fact that the problem of minimizing the loss function is equivalent to the problem of maximizing the log probability of the data and weights.

$$\log p(D, w) := \sum_{x_i, y_i \in D} \log \mathcal{N}(y_i | f(x_i, w), I) + \sum_d \log \mathcal{N}(w_d | 0, \frac{1}{\sqrt{\lambda}})$$

Log-likelihood of dataset D under model | is log-probability of target label y_i under gaussian distribution $\mathcal{N}$ with mean given by output of network $f(x_i, w)$ and variance 1. | and prior belief about weights given by log-probability of w_d under Gaussian $\mathcal{N}$ with 0 mean and hyperparameter-controlled variance $\frac{1}{\sqrt{\lambda}}$.

Figure 5: Log probability (source)

So now, instead of having one setting W, we have a probability distribution over weights

$$p(W \mid D) \propto p(D \mid W)p(W)$$

And our prediction becomes a multidimensional integral over probability distributions.

$$p(y \mid x, D) = \int_w p(y \mid x, w)p(w \mid D)dw$$

As this integral is usually intractable, we must employ a means of approximation. In literature, there exists two prominent methods (1) VI - Variational inference and (2) MCMC - Markov Chain Monte Carlo. It is more popular to use the first method as it scales better with large data, but we're more concerned with the second method - in particular, it's HMC variant, in this report.

MCMC, through obtaining samples, turn this intractable into a sum (a process called monte carlo integration):

$$p(y \mid x, D) = \int_w p(y \mid x, w)p(w \mid D)dw \approx \frac{1}{N} \sum_{w_i \sim p(w|D)}^{M} p(y \mid x, w_i)$$

7 Hamiltonian Monte Carlo sampled Bayesian Neural Network - an in-depth look into implementation

7.1 Similarities between Traditional and Bayesian Neural Network

Before training

May need to preprocess, add/de-noise, regularize data, as well as determining proper choices of activation and loss function, number of hidden layers, as well as means of diagnostics.

Architecture

In the case of our BNN being HMC-sampled, the architecture remains the same: they both consist of Input layers, hidden layers (fully connected), activation functions, and loss function.

An example architecture for a multiclass classification problem with one hidden layer:

```
L1 = layers.InputLayer(X)

L2 = layers.FullyConnectedLayer(input_size,hidden_units)

L3 = layers.ReLuLayer()

L4 = layers.FullyConnectedLayer(hidden_units, output_size)

L5 = layers.SoftmaxLayer()

L6 = layers.CrossEntropy()
```

7.2 Traditional forward-backward neural network

Before training

Preprocessing steps may involves one or more of: choosing appropriate parameter initializations, setting up Adam parameters

Training

The training steps are straightforward:

1. Make a forward pass, obtaining prediction `y_pred`
2. Calculate loss (difference between prediction and truth value)
3. Back propagate the gradient of the loss through the architecture to adjust neural network parameters, such that loss function is increasingly minimized

Obtaining prediction

Is done by feeding a new data through the trained neural network (keep forwarding new data point through each layers)

7.3 Hamiltonian Monte Carlo sampled Bayesian Neural Network

Additional parts of Architecture

In addition to the architecture, we need the implementation of a few more functions - probability density function calculator, to be exact.

- HMC sampler, that can take in current parameters, run HMC, and propose new set of parameters
- $\log p(w, b)$: Log prior calculator. Usually the prior are set to be gaussian
- $\log p(D \mid w, b)$: Log likelihood calculator. The functional form of the likelihood is dependent on the problem at hand (e.g: multinomial for classification, (multivariate)normal for regression)
- $\log p(w, b \mid D)$: Log posterior calculator. This is the sum of log prior and log likelihood
- $K(p)$: Kinetic energy calculator, taking in p the momentum

- $U(q)$: Potential energy calculator, taking in q the position. This is the negative log posterior
- $\nabla U(q)$: Gradient of potential energy

Before training

Need to choose functional form of prior, hamiltonian monte carlo parameters, parameter initialization also needed. Most importantly: note that choices of appropriate parameters can decide whether your training will take 200 days or 20 minutes!

Training

The training steps are less straightforward

1. Make a forward pass, obtaining point prediction `y_pred`.
2. This prediction is then fed through an HMC
3. Within this HMC sampling step, new network parameters will be proposed, such that these parameters maximize the log probability (or maximize it's negation) of the data and weight

Obtaining prediction

Is done by feeding a new data through the trained neural network (keep forwarding new data point through each layers). Then sampled through use of HMC, through the monte carlo integration process discussed

8 Going infinity

Despite the fact that BNN in general a better alternative to the traditional settings, it suffers from some setback: mainly, that there are not many great posterior samplers for it - seeing that a BNN is only as good as it's sampler, this is something more research should be done on; and partly, BNN, like most neural network, suffer in performance as the number of parameters goes up - an aspect especially transparent in HMC-BNN.

Sell and Singh published a paper in 2023 outlining a new type of neural network - called "Trace-class Neural Network (tcNN)", which would allow us to work on BNN with infinite parameter - in other words, rendering scaling of parameters irrelevant. This is an exciting area to explore, and is where I wish to take this current project to next.

A few words on tcNN: this works on an infinite width neural network. With prior being independent, centred, and Gaussian, and likelihood a Radon-Nikodym derivative. This induce a posterior that is defined on a separable Hilbert space, and can be sampled using preconditioned Crank-Nicolson algorithm (a type of MCMC)

9 Appendix B

9.1 Neal's HMC pseudocode

```
MC = function (U, grad_U, epsilon, L, current_q)
{
    q = current_q
    p = rnorm(length(q),0,1) # independent standard normal variates
    current_p = p
    # Make a half step for momentum at the beginning
    p = p - epsilon * grad_U(q) / 2
    # Alternate full steps for position and momentum
    for (i in 1:L)
    {
        # Make a full step for the position
        q = q + epsilon * p
        # Make a full step for the momentum, except at end of trajectory
        if (i!=L) p = p - epsilon * grad_U(q)
    }
    # Make a half step for momentum at the end.
    p = p - epsilon * grad_U(q) / 2
    # Negate momentum at end of trajectory to make the proposal symmetric
    p = -p
    # Evaluate potential and kinetic energies at start and end of trajectory
    current_U = U(current_q)
    current_K = sum(current_p^2) / 2
    proposed_U = U(q)
    proposed_K = sum(p^2) / 2
    # Accept or reject the state at end of trajectory, returning either
    # the position at the end of the trajectory or the initial position
    if (runif(1) < exp(current_U-proposed_U+current_K-proposed_K))
    {
        return (q) # accept
    }
    else
    {
    return (current_q) # reject
    }
}
```

(source)

CODE EXAMPLE

10 Foreword

Parameters and architectures are chosen based on (1) their general use (2) personal experience with them

11 Classification - MNIST 100

11.1 Architecture and parameters

MNIST architecture: FC (100 outputs), ReLu, FC, Softmax, Cross entropy

Traditional neural network has Adam implemented.

11.2 Traditional result

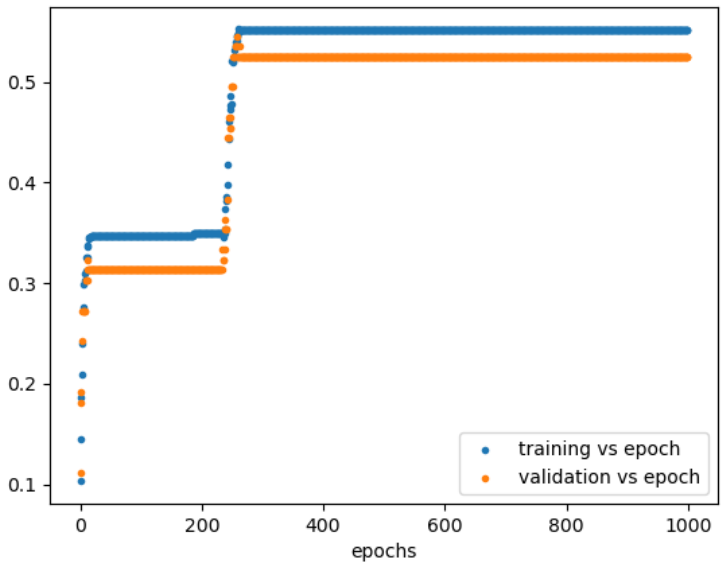

Figure 6: Trained (pretrained)

```
First RMSE: 84.97591925123854
First training accuracy: 0.5515515515515516
First validation accuracy: 0.5252525252525253
Weights and biases are saved as weights_nn_mnist.pkl
Training vs Validation plot: nn_mnist.png
With adam
Time took: ~ 3 minutes

>> 100 more epoch:
```

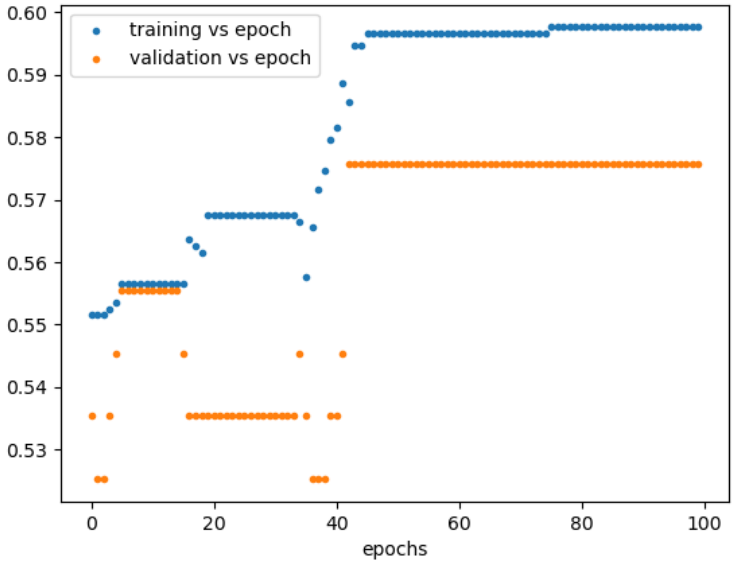

Figure 7: Trained for another 100 epochs

```
Final RMSE: 80.49518282056779
Final training accuracy: 0.5975975975975976
Final validation accuracy: 0.5757575757575758
Time took: few seconds
```

Remarks: accuracy is good, not amazing. Kept training it shows overfitting

11.3 BNN result

```
First training accuracy: 0.08708708708708708
First validation accuracy: 0.0707070707070707
Time took: ~ 2.5 hours

>> 100 more epoch after pretrained using nn_mnist weights + biases:
Final training accuracy: 0.5425425425425425
Final validation accuracy: 0.5252525252525253

Time took: ~12 minutes
```

Remarks: It took too long and time was running out so I'm abandoning getting more plots with the results or train it more. I finally understand why people said HMC-BNN does not scale well.

11.4 Key takeaway

- HMC-BNN does take a lot more to run

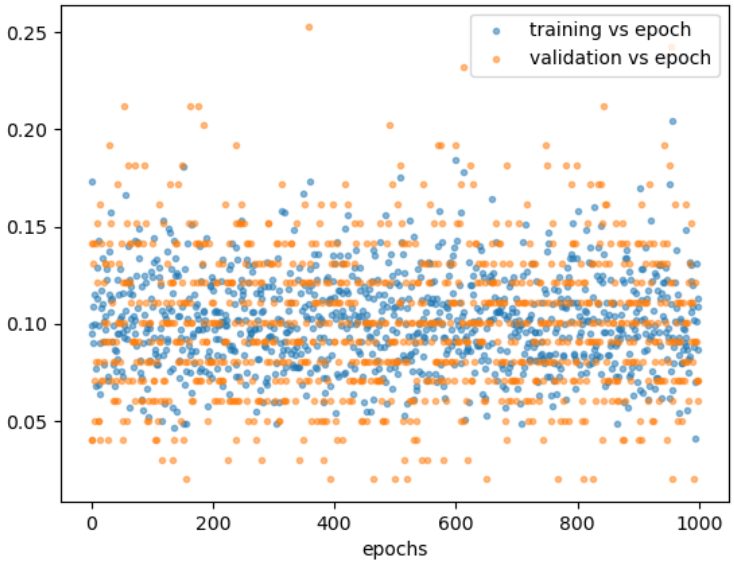

Figure 8: No pretraining.
First run was supposed to take 99 hours. Tweaked parameters so it can come down to 2 hours and this is the result.

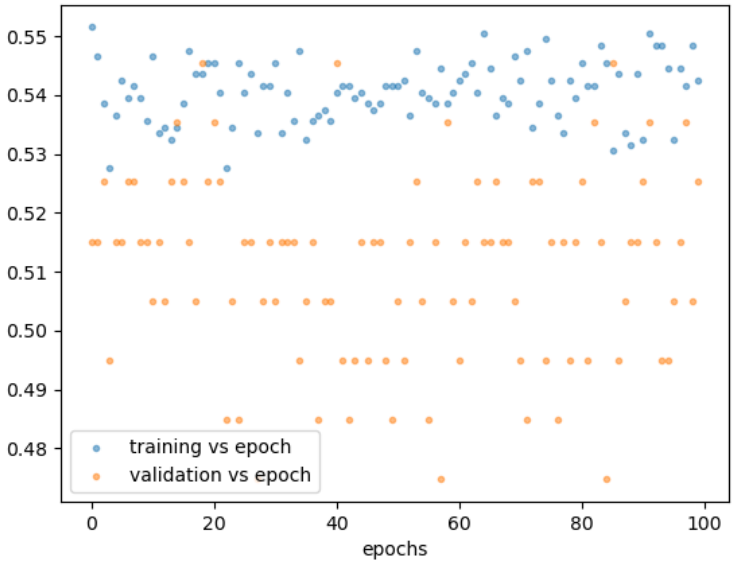

Figure 9: Trained, parameters initialized using traditional neural network's pretrained's weights

- Pre-training will save the day. I couldn't get diagnostic plots for BNN prediction due to time running out however
- Choosing good parameter is everything

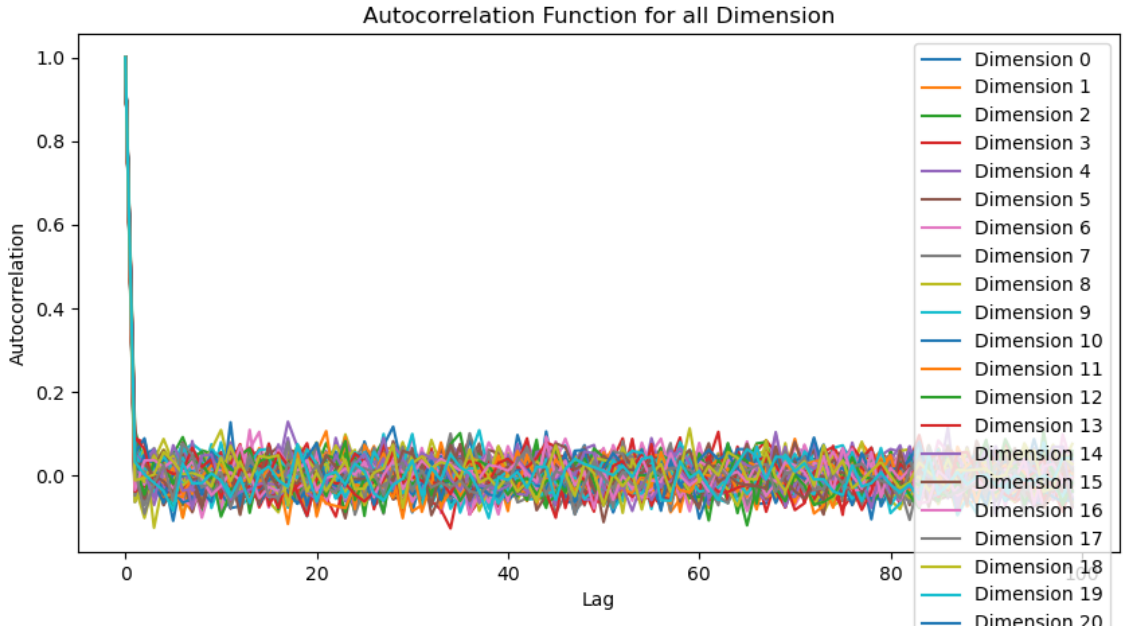

Figure 10: Autocorrelation of weight of first FC. Chain looks well mixed. Which is great.

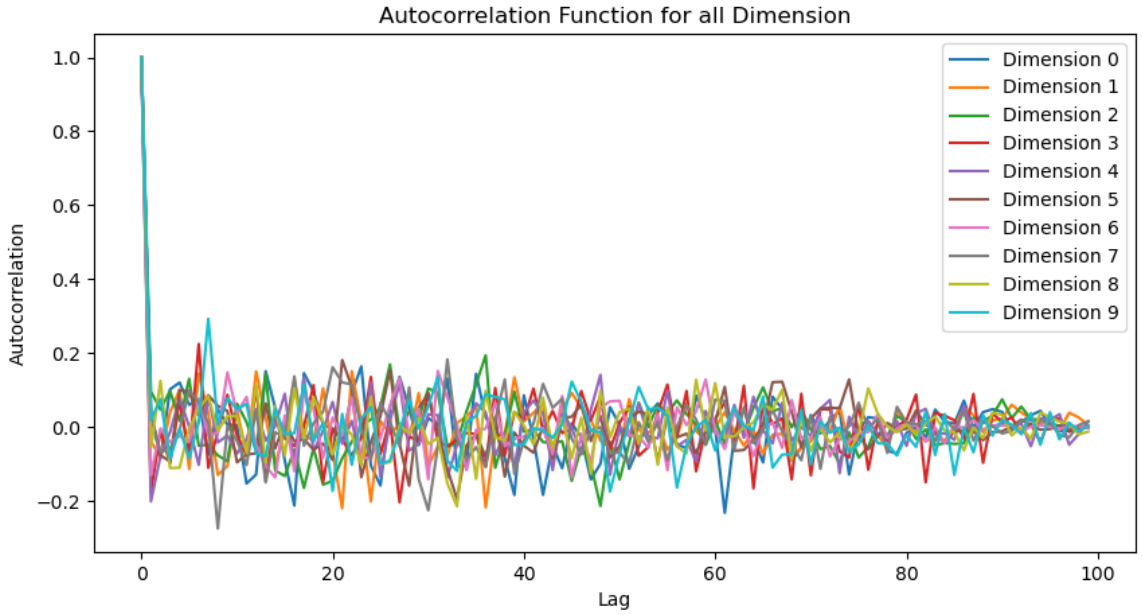

Figure 11: Autocorrelation of weight of second FC. Chain looks well mixed, which is great

- I forgot to discard the first few samples for burn-in. This is bad if i want to obtain results
- As you can see, traditional neural network starts to suffer from overfitting. In theory BNN prediction won't do such thing as it doesn't output point estimates (in a sense you can also look at accuracy?)

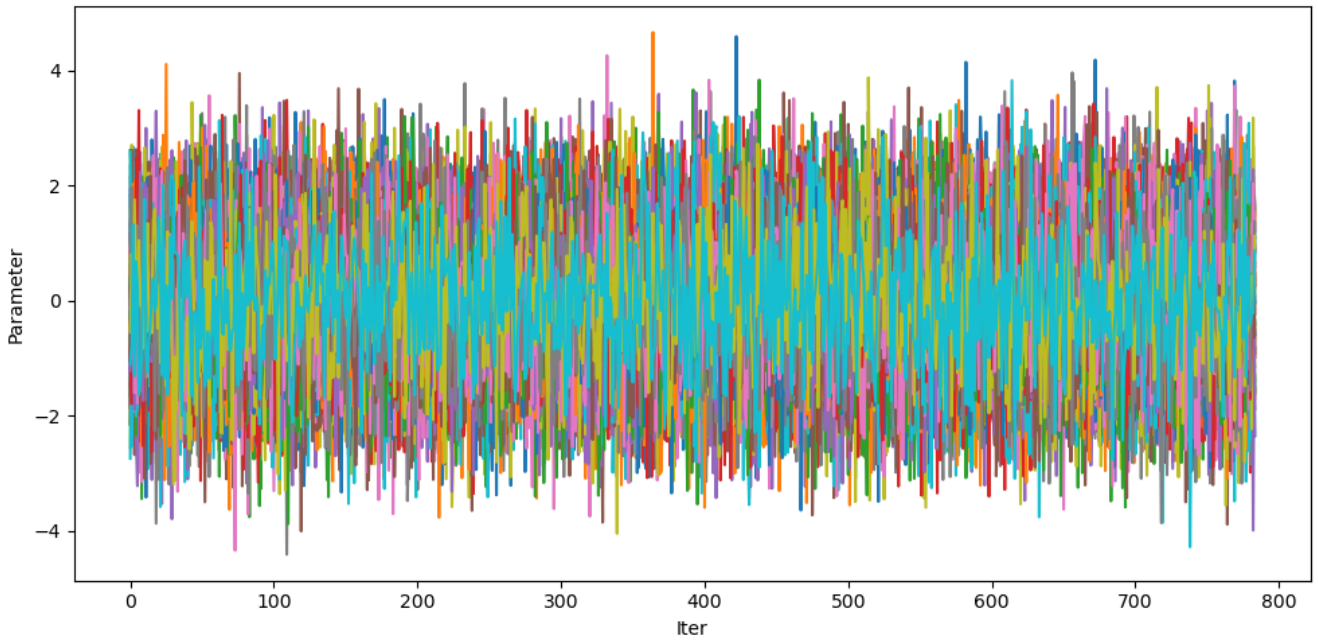

Figure 12: Individual sample of weight of first FC. Looks good? It does look like it's centering around one value (0) which is great, but too great variance

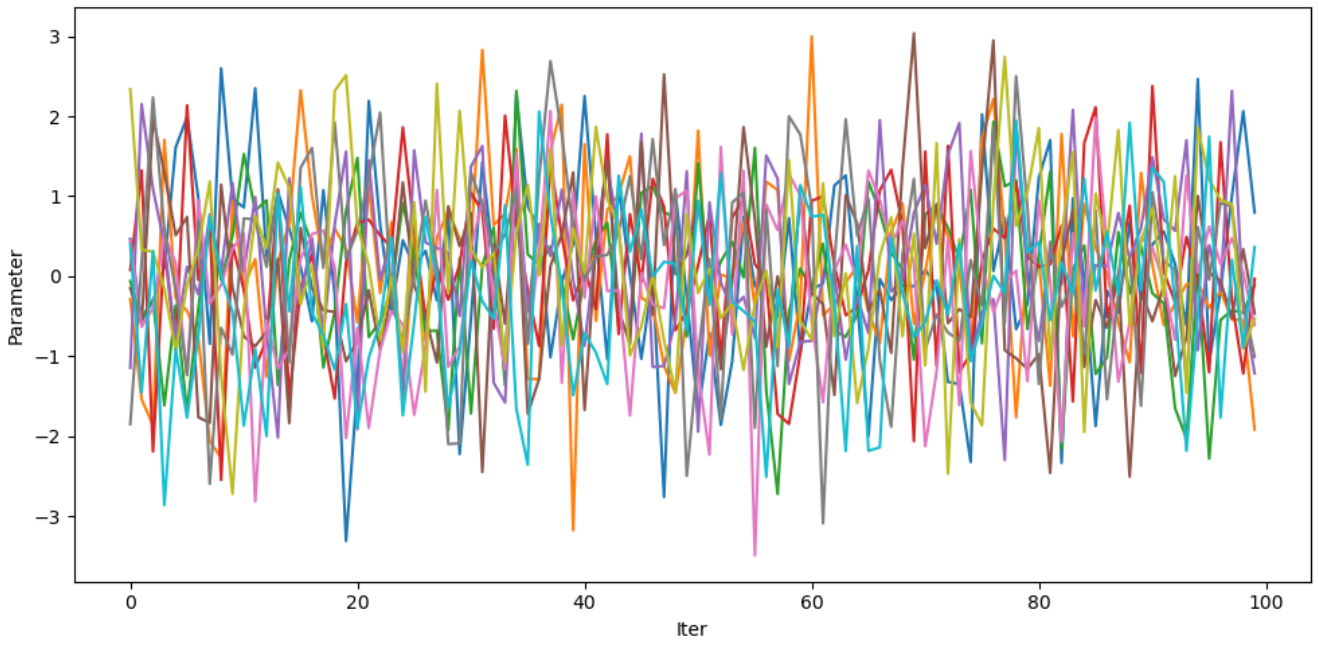

Figure 13: Individual sample of weight of second FC. Same comment as above.

12 Regression - Boston Housing

12.1 Architecture and parameters

Boston housing architecture: FC (10 output), ReLu, FC, Linear, SE.

No Adam, just vanila MLP under all these.

This is a lot less data-heavy dataset, and I also tweak parameters such that everything is really "mild". Architecture does not need to be tweak much: core MLP activation and loss function, as well as likelihood, must be changed. But overall not that bad.

I decided to also code up a BNN regression model out of hope that it can actually give me something worth reporting

12.2 Traditional result

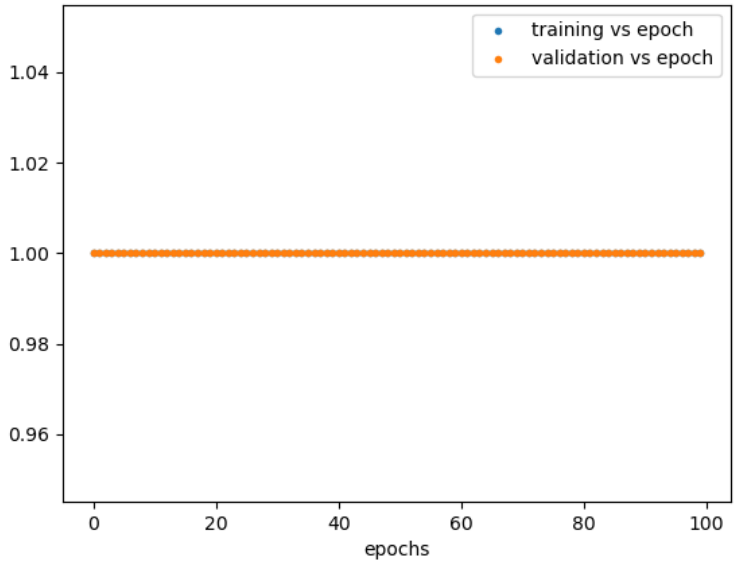

Figure 14: Traditional result kept crashing

Remark: interestingly, bnn works well in this case, but traditional neural network kept encountering nan errors

12.3 BNN result

12.4 Key takeaway

- BNN works but traditional neural network doesn't. So there must be something there. Training time is also very short (less than 10 mins)

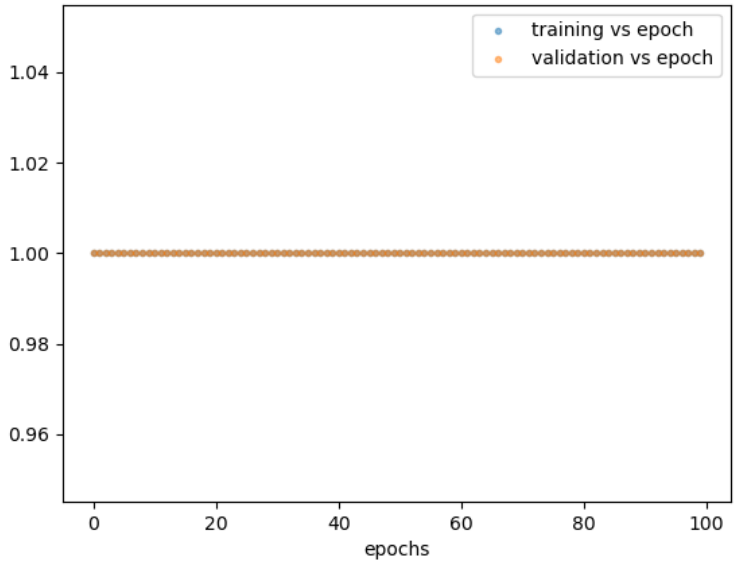

Figure 15: Accuracies plot - looks quite sus and makes me doubt the correctness of my model. But then again it could also be because of the availability of data.

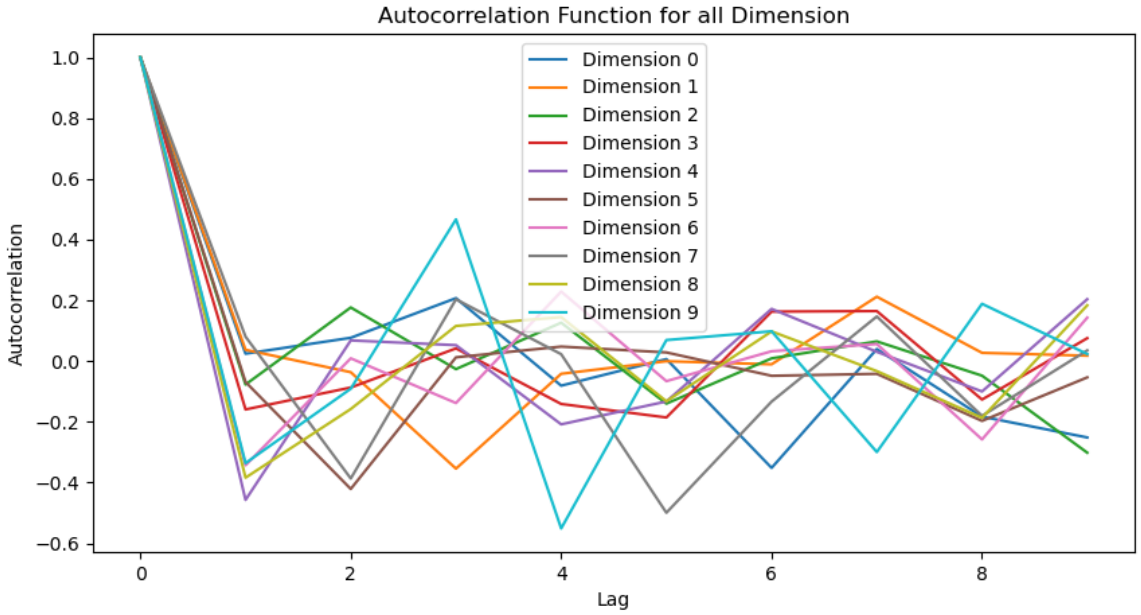

Figure 16: Autocorrelation plot for weight of first FC. By dimension I meant weight. Looks quite good

- Predicted probability looks ... ok. Maybe with more data and better model I can make better prediction

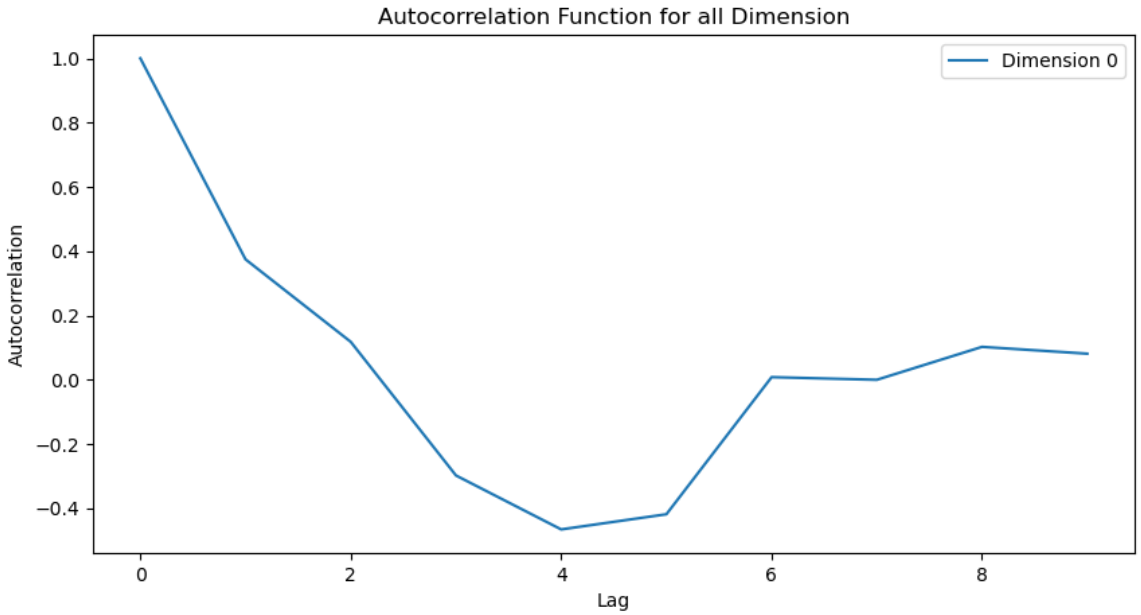

Figure 17: Autocorrelation plot for weight of second FC. Next time I really should employ more walkers (I only use 1)

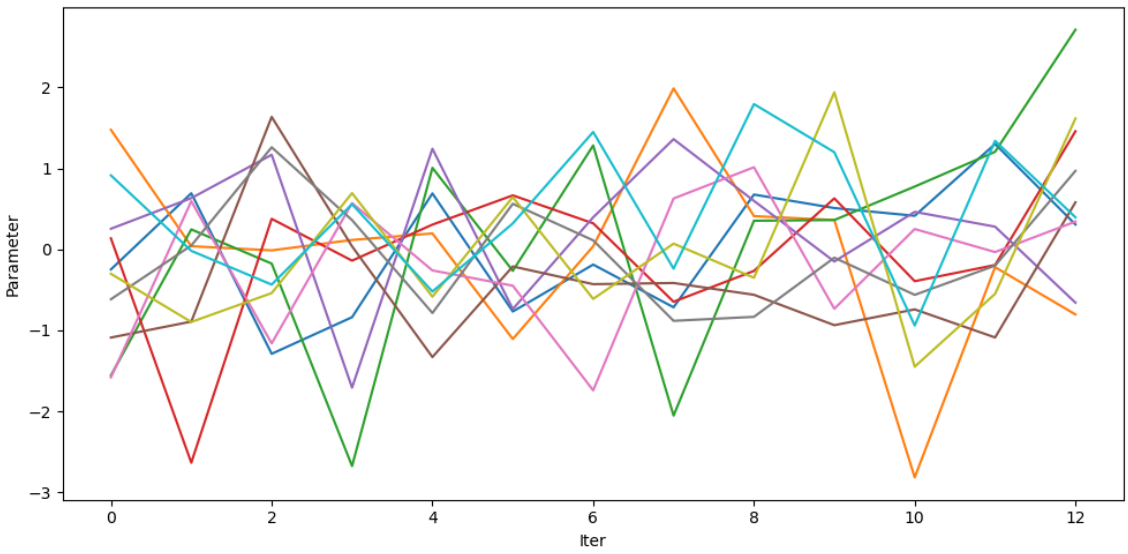

Figure 18: Individual sample of weight of first FC. Not that great

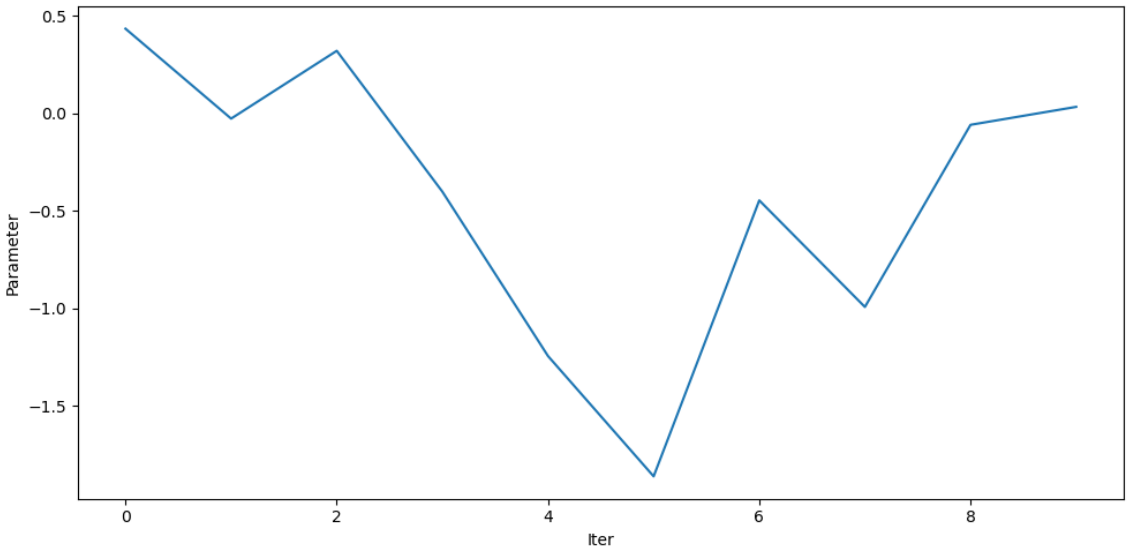

Figure 19: Individual sample of weight of second FC

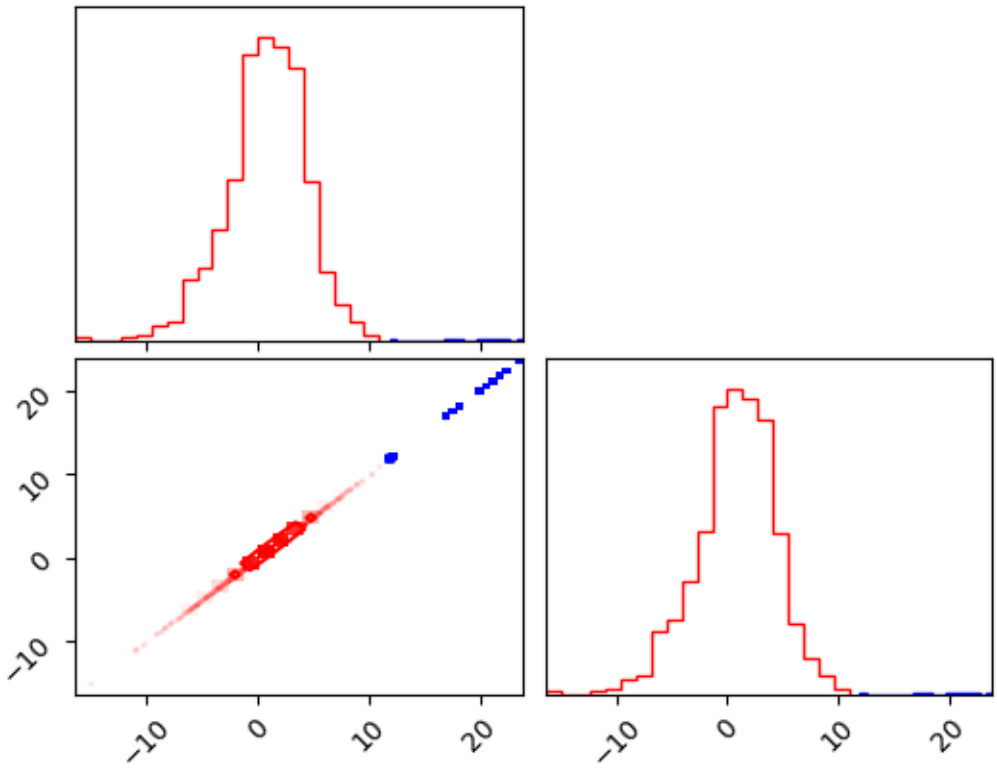

Figure 20: Predicted probability (red) vs where real values clumps to (blue). This really convinced me that I don't have enough data