Bayesian Neural Networks - Presenters

Group 1: A Practical Bayesian Framework for Backpropagation Networks - Slides 2-40

- Paul Vicol
- Shane Baccas
- George Alexandru Adam

Group 2: Priors for Infinite Networks - Slides 41-64

- Soon Chee Loong

Group 3: MCMC using Hamiltonian Dynamics - Slides 65-91

- Tristan Aumentado-Armstrong
- Guodong Zhang
- Chris Cremer

Group 4: Stochastic Gradient Langevin Dynamics - Slides 92-110

- Alexandra Poole
- Yuxing Zhang
- Jackson K-C Wang

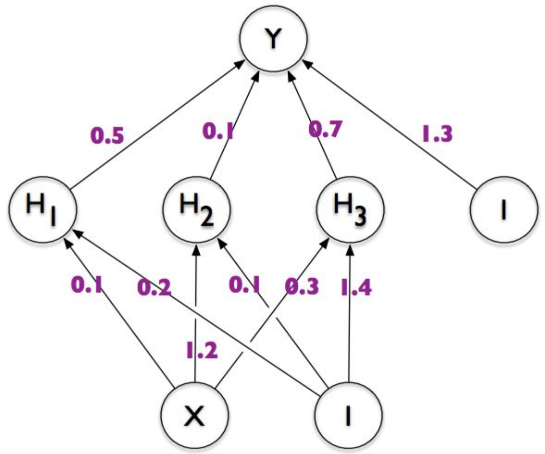
Bayesian Neural Networks

CSC2541:
Scalable and
Flexible Models of Uncertainty

Motivation - Why use Bayesian Neural Nets?

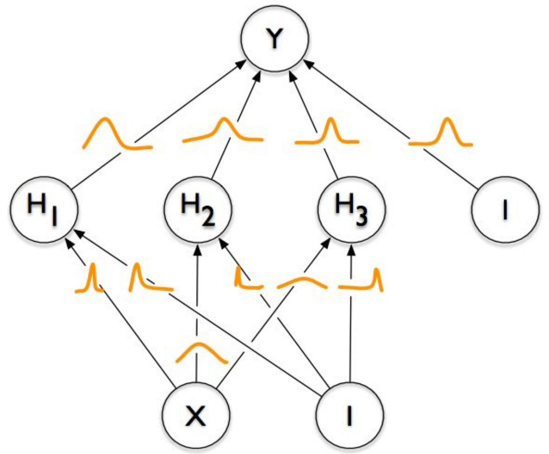
- With Bayesian methods, we obtain a distribution of answers to a question rather than a point estimate
- This can help address regularization and model comparison without a held-out validation set
    - Compare and choose architectures, regularizers, and other hyperparameters
- Can also compute a distribution over outputs: place error bars on $P(y \mid x, \mathcal{D})$

Standard Neural Net



- Parameters represented by *single, fixed values (point estimates)*
- Conventional approaches to training NNs can be interpreted as *approximations* to the full Bayesian method (equivalent to MLE or MAP estimation)

Bayesian Neural Net



- Parameters represented by *distributions*
- Introduce a prior distribution on the weights $P(\mathbf{w})$ and obtain the posterior $P(\mathbf{w} \mid \mathcal{D})$ through Bayesian learning
- Regularization arises naturally through the prior $P(\mathbf{w})$
- Enables principled model comparison

Images from: Blundell, C. et al. Weight Uncertainty in Neural Networks. *ICML 2015*.

Conventional Training as Bayesian Approximation

Minimizing:

Is Equivalent To:

Squared error (no regularization)

$$\sum_{i=1}^{N}(y(x^{(i)};\mathbf{w})-t^{(i)})^2$$

$\rightarrow$

Maximum likelihood estimation

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} P(\mathcal{D} \mid \mathbf{w})$$

Squared error (+ L^2 regularization)

$$\beta\sum_{i=1}^{N}(y(x^{(i)};\mathbf{w})-t^{(i)})^2 + \alpha||\mathbf{w}||^2$$

$\rightarrow$

MAP estimation with a Gaussian prior

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} P(\mathbf{w} \mid \mathcal{D})$$

where $\mathbf{w} \sim \mathcal{N}(0, \sigma_w^2)$

The Role of Integration in Bayesian Methods

- Many problems addressed by Bayesian methods involve *integration:*
    - Evaluate *distribution of network outputs* by integrating over weight space

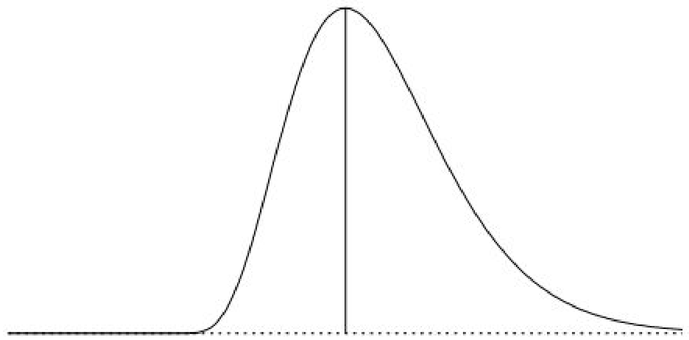
$$P(y \mid x, \mathcal{D}) = \int P(y \mid \mathbf{w}, x) P(\mathbf{w} \mid \mathcal{D}) d\mathbf{w}$$

- Compute the *evidence* for *Bayesian model comparison*

$$P(\mathcal{D} \mid \mathcal{H}_i) = \int P(\mathcal{D} \mid \mathbf{w}, \mathcal{H}_i) P(\mathbf{w} \mid \mathcal{H}_i) d\mathbf{w}$$

- These integrals are often intractable, and must be approximated

Methods of Approximating Integrals



- Gaussian approximation (1/2): Allows the integrals to be evaluated *analytically*
  - Optimization involved in finding the mean of the Gaussian
- Monte Carlo methods (2/2): Draw samples to evaluate the integral directly
- Variational inference (next week): Convert integration into optimization
  - Minimize KL divergence between the posterior and a proposed parametric function

Image from: MacKay, D. Information Theory, Inference, and Learning Algorithms. *Cambridge University Press. p. 341.* 2003.

BNNs - Topics

- Understand how to apply Bayesian model comparison to neural networks
- Explore the connection between neural networks and Gaussian Processes
  - Understand the meaning behind the prior distributions imposed over network parameters as the first step in Bayesian inference
  - What types of functions do BNNs compute when their weights are drawn from certain types of prior distributions?
- Understand how the computations required by the Bayesian approach can be performed using Markov Chain Monte Carlo methods
  - Hamiltonian Monte Carlo
  - Stochastic Gradient Langevin Dynamics (SGLD)

A Practical Bayesian Framework for Backpropagation Networks

Paper by: David J. C. MacKay

Presented by: Paul Vicol, Shane Baccas, George-Alexandru Adam

Overview

- Bayesian methods can be applied at two stages of a data modeling problem:
  1. Fitting a specific model to data by inferring its parameters
  2. *Comparing and ranking alternative models*
- Bayesian evidence enables principled comparisons between alternative **neural network architectures** and **regularization methods**
- **Key result:** For models well-suited to a problem, **Bayesian evidence** and **generalization error** are correlated

A Neural Network Model for Regression

Dataset

$$\mathcal{D} = \{(x^{(m)}, t^{(m)})\}_{m=1}^{N}$$

Data Error: Sum of Squared Errors

$$E_D(\mathcal{D} \mid \mathbf{w}, \mathcal{A}) = \frac{1}{2} \sum_{m=1}^{N} \left(y(x^{(m)}; \mathbf{w}, \mathcal{A}) - t^{(m)} \right)^2$$

Model

A neural network with architecture $\mathcal{A}$ and parameters $\mathbf{w}$ that defines a mapping $y(x; \mathbf{w}, \mathcal{A})$

Regularization

Add **regularizing term** E_w to penalize large weights for a smoother mapping:

$$E_w(\mathbf{w}|\mathcal{A}) = \frac{1}{2} \sum_i w_i^2$$

Obtain a regularized cost function: $M = \alpha E_W(\mathbf{w} \mid \mathcal{A}) + \beta E_D(\mathcal{D} \mid \mathbf{w}, \mathcal{A})$

Motivation for Bayesian Methods

$$\boxed{M = \alpha E_w(\mathbf{w} \mid \mathcal{A}) + \beta E_D(\mathcal{D} \mid \mathbf{w}, \mathcal{A})}$$

- Many free parameters, including:
  - The architecture $\mathcal{A}$
  - The regularization strength α
  - The regularizer $\mathcal{R}$ (E_W)

Common method to compare networks

- Split data into disjoint training/validation sets
- Optimize network parameters $\mathbf{w}$ on training set
- Optimize control parameters like α and $\mathcal{A}$ on validation set

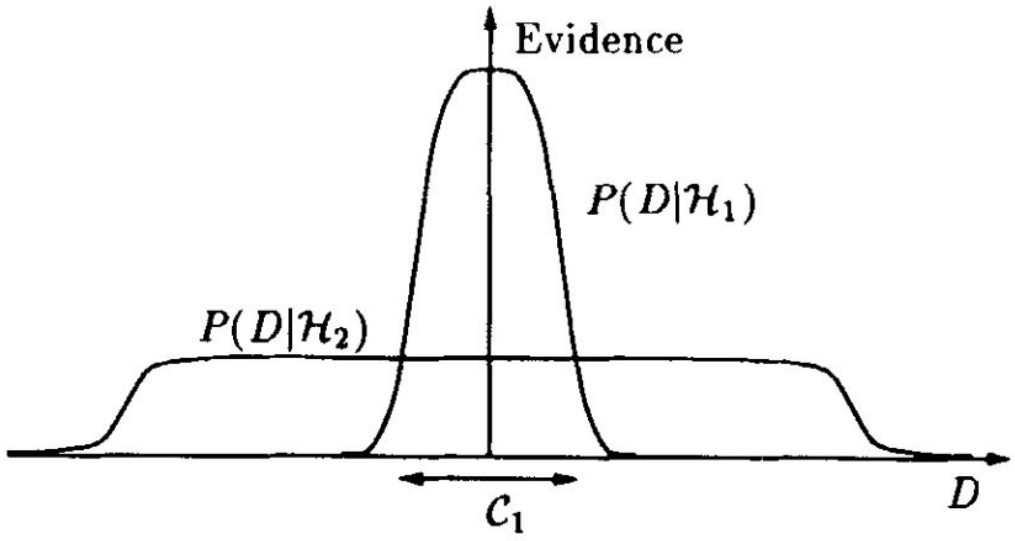
Drawbacks

- Need a large val set to achieve a good signal to noise ratio
- Cross-validation is computationally demanding
- Grid search over many parameters is not tractable

What We Want

- We would like objective criteria to set control parameters and compare alternative solutions
- In particular, techniques that do not require creating a held-out test set
  - Critical in situations where data is limited
- We want to use *all* the data for both:
  - Optimizing the weights $\mathbf{w}$, and
  - Optimizing control parameters α, $\mathcal{A}$, and $\mathcal{R}$

The Difficulty of Model Comparison: Occam's Razor



- We can't just choose the model that fits the data best
  - More complex models will always fit better, but fail to generalize
- Should account for Occam's Razor: balance data fit and model complexity

Image from: MacKay D., Bayesian Interpolation. *Neural Computation 4*, 415-447. 1992.

Bayesian Model Comparison

- *Bayesian model comparison*: determine which class of models $\mathcal{H}_i$ is best suited to explain the observed data $\mathcal{D}$
- We rank alternative models $\mathcal{H}_i$ by computing the *evidence*: $P(\mathcal{D} \mid \mathcal{H}_i)$

$$\begin{aligned} P(\mathcal{D} \mid \mathcal{H}_i) &= \int P(\mathcal{D}, w \mid \mathcal{H}_i) dw \\ &= \int P(\mathcal{D} \mid w, \mathcal{H}_i) P(w \mid \mathcal{H}_i) dw \end{aligned}$$

- *Bayesian evidence embodies Occam's razor naturally*
  - Penalizes overly complex models
  - Helps detect poor assumptions in learning models

Bayesian Neural Networks

- Retains the same topology of regular Neural Nets, however we assume a *prior distribution* over the weights and we follow an iterative procedure for estimating the hyperparameters.
- Under the Bayesian regime, we are not interested in the values of the weights, instead we make predictions using the marginal likelihood function (predictive distribution) whose mean is $\mathbf{w}_{\mathrm{MAP}}$
- **Two advantages:**
  - We can estimate hyperparameters iteratively using entire data set.
  - We can provide "well calibrated" confidence intervals around a prediction:

$$\mathbf{E}\left[\mathbf{t}|X = x^{(m)}\right] = y(x^{(m)}, \mathbf{w}_{\mathrm{MAP}})$$

Bayesian Neural Networks Overview

Prior distribution over target variables

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

Prior distribution over weights

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

Likelihood function (assuming iid)

$$p(\mathcal{D}|\mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1})$$

Posterior

Then $p(\mathbf{w}|\mathcal{D}, \alpha, \beta) \propto p(\mathbf{w}|\alpha)p(\mathcal{D}|\mathbf{w}, \beta)$

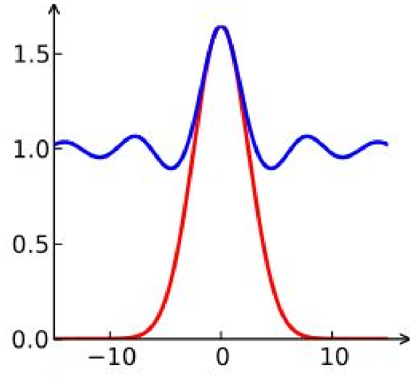
*Which is not Gaussian because the function y is a neural network whose dependence on **w** is non-linear*

→ We will build a Gaussian Approximation to the log-posterior.

Laplace Approximation

- Replace the posterior by a Gaussian centered at a peak of the posterior (a *mode*)
- The covariance of the Gaussian is small
- Assumes that the posterior distribution P has a global maximum

$$\int e^{Mf(\mathbf{x})}\, d\mathbf{x} \approx \qquad \approx \left(\frac{2\pi}{M}\right)^{d/2} |-H(f)(\mathbf{x}_0)|^{-1/2} e^{Mf(\mathbf{x}_0)} \text{ as } M \to \infty$$



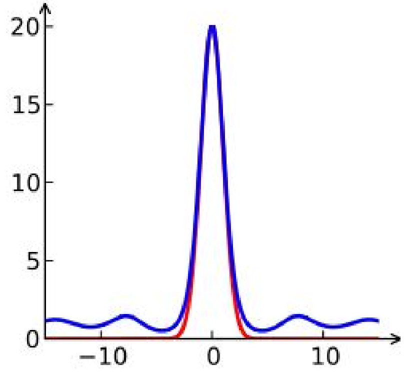


Image from: https://en.wikipedia.org/wiki/Laplace%27s_method

Gaussian Approximation to the Posterior

- The main idea in the procedure is to find a **non-unique** mode $\mathbf{w}_{MAP}$ of the log-posterior (through BP and SGD) and use this mode with Laplace approximation to find the Gaussian approximation about that particular mode:

$$\boldsymbol{w}_{MAP} = \underset{w}{argmax} \left[-\frac{1}{2} (\alpha E_w + \beta E_D) \right]$$

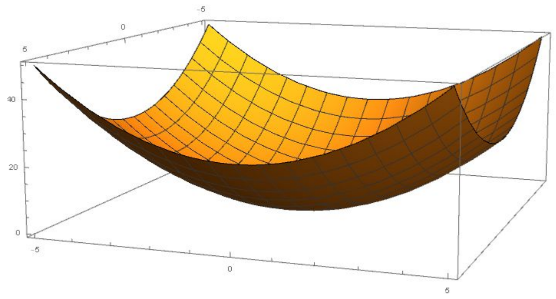
We then find the Hessian:

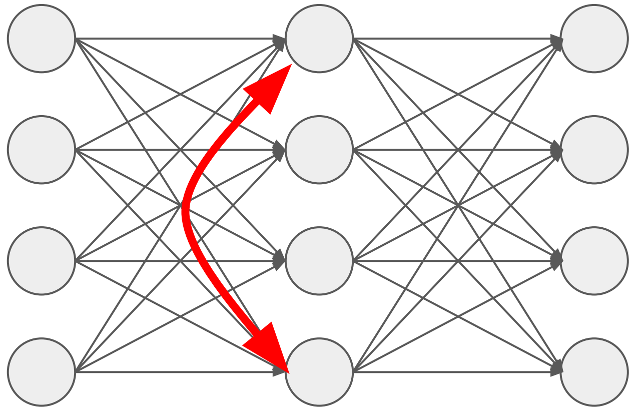
$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H} \qquad \mathbf{H} = \nabla\nabla E_D(\mathbf{w})$$

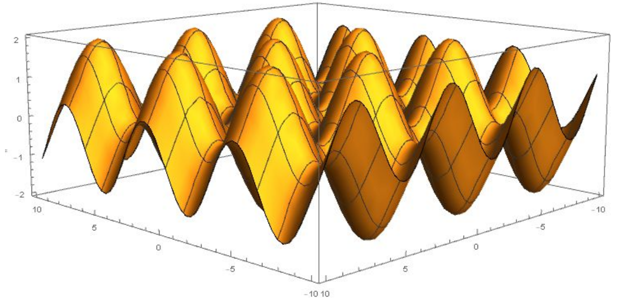
Thus by Laplace approximation the posterior Gaussian approximation becomes:

$$q(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}}, \mathbf{A}^{-1})$$

Neural Net Weight Space Symmetry







- If we permute all connections of one neuron with another in the same layer (e.g., swap positions of the neurons), the network output is unchanged
    - Because the weighted contributions of neurons are *summed* (order-invariant).
    - This causes many local minima (see figure on right)

Original figures: Left and right by G. A. Adam, center by P. Vicol

Towards the Predictive Distribution

- The predictive distribution is then given by: $p(t|\mathbf{x}, \mathcal{D}) = \int p(t|\mathbf{x}, \mathbf{w})q(\mathbf{w}|\mathcal{D})\,\mathrm{d}\mathbf{w}$
- Unfortunately this is still analytically intractable because our hypothesis function is a non-linear neural net. So we approximate $y(\mathbf{x}, \mathbf{w})$ with its first-order Taylor series expansion: $y(\mathbf{x}, \mathbf{w}) \simeq y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}) + \mathbf{g}^{\mathrm{T}}(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}})$

  where $\mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\mathrm{MAP}}}$
- This gives us: $p(t|\mathbf{x}, \mathbf{w}, \beta) \simeq \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}) + \mathbf{g}^{\mathrm{T}}(\mathbf{w} - \mathbf{w}_{\mathrm{MAP}}), \beta^{-1}\right)$
- Now we can approximate the predictive distribution with

$$p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{MAP}}), \sigma^2(\mathbf{x})\right)$$

  Where: $\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g}$

Learning Hyperparameters Online

- To learn α and β, the precision parameters over our prior distributions, online we need to find marginal likelihood over α and β by integrating over the network weights.

$$p(\mathcal{D}|\alpha, \beta) = \int p(\mathcal{D}|\mathbf{w}, \beta) p(\mathbf{w}|\alpha)\, \mathrm{d}\mathbf{w}$$

- Which gives us:

$$\ln p(\mathcal{D}|\alpha, \beta) \simeq -E(\mathbf{w}_{\mathrm{MAP}}) - \frac{1}{2}\ln|\mathbf{A}| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi)$$

Where:

$$E(\mathbf{w}_{\mathrm{MAP}}) = \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{MAP}}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}$$

Maximizing Joint Log-Marginal for α and β

- To find maximum likelihood point estimates for α define the eigenvalue equation

$$\beta \mathbf{H} u_i = \lambda_i u_i$$

where $\mathbf{H} = \nabla\nabla E_D(\mathbf{w})$ evaluated at $\mathbf{w}_{\mathrm{MAP}}$

- This implies

$$\alpha = \frac{\gamma}{\mathbf{w}_{\mathrm{MAP}}^{\mathrm{T}}\mathbf{w}_{\mathrm{MAP}}} \quad \text{where} \quad \gamma = \sum_{i=1}^{W} \frac{\lambda_i}{\alpha + \lambda_i} \quad \text{and} \quad \frac{1}{\beta} = \frac{1}{N-\gamma}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{MAP}}) - t_n\}^2$$

- Notice these equations only *implicitly* define α and β. We must first begin with guesses for α and β and we update our guesses using the posterior distribution, as new batches of data come in.

Energy-Based Probabilistic Interpretation

- Using the standard Bayesian framework of likelihood, prior and posterior, we define the following probabilities:

Prior

$$P(\mathbf{w} \mid \alpha, \mathcal{A}, \mathcal{R}) = \frac{\exp[-\alpha E_W(\mathbf{w} \mid \mathcal{A})]}{Z_W(\alpha)}$$

Likelihood

$$P(\mathcal{D} \mid \mathbf{w}, \beta, \mathcal{A}) = \frac{\exp[-\beta E_D(\mathcal{D} \mid \mathbf{w}, \mathcal{A})]}{Z_D(\beta)}$$

Posterior

$$P(\mathbf{w} \mid \mathcal{D}, \alpha, \beta, \mathcal{A}, \mathcal{R}) = \frac{\exp(-\alpha E_W - \beta E_D)}{Z_M(\alpha, \beta)}$$

Bayesian Approach over Architectures

- Probabilistic assumptions on the random variables of interest:
  - $P(t^{(m)}) = \mathcal{N}(y(x^{(m)}; \mathbf{w}, \mathcal{A}), \frac{1}{\beta})$ where $t^{(m)}$ corresponds to particular $x^{(m)}$ with additive Gaussian noise
  - $P(\mathbf{w}) = \mathcal{N}(0, \alpha^{-1})$ the random vector $\mathbf{w}$ is Gaussian with mean **0** and α precision parameter
- This implies:

Prior: $$P(\mathbf{w} \mid \alpha, \mathcal{A}, \mathcal{R}) = \frac{\exp[-\alpha E_W(\mathbf{w} \mid \mathcal{A})]}{Z_W(\alpha)}$$

Likelihood: $$P(\mathbf{t}^m \mid \mathbf{x}^m, \mathbf{w}, \beta, \mathcal{A}) = \frac{\exp[-\beta E(\mathbf{t}^m \mid \mathbf{x}^m, \mathbf{w}, \mathcal{A})]}{Z_m(\beta)}$$

Posterior: $$P(\mathbf{w} \mid D, \alpha, \beta, \mathcal{A}, \mathcal{R}) = \frac{\exp(-\alpha E_W - \beta E_D)}{Z_M(\alpha, \beta)}$$

Gaussian Integrals

$$Z_W = \int d^k\mathbf{w} \exp(-\alpha E_W)$$

$$Z_m(\beta) = \int d\mathbf{t} \exp(-\beta E)$$

$$Z_M(\alpha, \beta) = \int d^k\mathbf{w} \exp(-\alpha E_W - \beta E_D)$$

Finding the Posterior

- The posterior is given by $P(\mathbf{w} \mid \mathcal{D}, \alpha, \beta, \mathcal{A}, \mathcal{R}) = \dfrac{\exp(-\alpha E_W - \beta E_D)}{Z_M(\alpha, \beta)}$

- Finding the most probable value of $\mathbf{w}$ for the posterior, i.e., $\mathbf{w}_{MAP}$ is equivalent to minimizing the regularized cost function M, defined as $M = \alpha E_w(\mathbf{w} \mid \mathcal{A}) + \beta E_D(\mathcal{D} \mid \mathbf{w}, \mathcal{A})$, for $\mathbf{w}$

But how do we find the parameters α and β?

MacKay Bayesian Framework

By Bayes Theorem: $P(\alpha, \beta \mid \mathcal{D}, \mathcal{A}, \mathcal{R}) = \dfrac{P(\mathcal{D} \mid \alpha, \beta, \mathcal{A}, \mathcal{R}) P(\alpha, \beta)}{P(\mathcal{D} \mid \mathcal{A}, \mathcal{R})}$

We assign a uniform prior to α and β and have $P(\mathcal{D} \mid \alpha, \beta, \mathcal{A}, \mathcal{R}) = \dfrac{Z_M(\alpha, \beta)}{Z_W(\alpha) Z_D(\beta)}$

Now let $N = |\mathcal{D}| \cdot dim(t^{(m)})$ and $k = dim(\mathbf{w})$

$$\left.\begin{aligned} Z_W(\alpha) &= \int \exp(-\alpha E_W) d^k \mathbf{w} = \boxed{(2\pi/\alpha)^{k/2}} \\ Z_D(\beta) &= \int \exp(-\beta E_D) d^N D = \boxed{(2\pi/\beta)^{N/2}} \end{aligned}\right\} \text{Evaluate integrals directly}$$

For $Z_M(\alpha, \beta) = \int \exp[-M(\mathbf{w}, \alpha, \beta)] d^k \mathbf{w}$ we must use Laplace approximation for Gaussian integrals

Estimating α and β for NNs

- Assume:
  - The posterior probability of α and β consists of *well separated islands in parameter space* each centered around a minimum of M
- Consider $\mathbf{w}^*$ a minimum of M and define the solution $S_{\mathbf{w}^*}$ as the ensemble of networks A in the neighborhood of $\mathbf{w}^*$ and all symmetric permutations of that ensemble.

- Posterior probability of solution is:

$$P(S_{\mathbf{w}^*}, \alpha, \beta, \mathcal{A}, \mathcal{R} \mid D) \propto g \frac{Z_M^*(\mathbf{w}^*, \alpha, \beta)}{Z_W(\alpha) Z_D(\beta)} P(\alpha, \beta) P(\mathcal{A}, \mathcal{R})$$

where $$Z_M^*(\mathbf{w}^*, \alpha, \beta) = \int_{S_{\mathbf{w}^*}} d^k\mathbf{w} \exp[-M(\mathbf{w}, \alpha, \beta)]$$

How do we Calculate Z_M^*?

- We have expressions for every quantity but Z_M^*; because of our assumption, we can use Laplace approximation:

$$Z_M^* \simeq e^{-M(\mathbf{w}^*)}(2\pi)^{k/2}\det^{-1/2}\mathbf{A}$$

- Where $\mathbf{A} = \nabla\nabla M$ is the Hessian of M evaluated at $\mathbf{w}^*$

- This approximation works when $\frac{|\mathcal{D}|}{dim(x^{(m)})}$ is “large” by C.L.T.
- Also:
  - Recent paper from Pennington and Bahri *(JMLR, 2017)* also treats Hessian estimation for NNs using Random Matrix Theory

Comparing Models

- To assign preference to alternative *architectures* and *regularizers,* we evaluate the *evidence* $P(D, S_{\mathbf{w}^*}|\mathcal{A}, \mathcal{R})$ of the solutions we found by marginalizing out α and β:

$$P(D, S_{\mathbf{w}^*}|\mathcal{A}, \mathcal{R}) = \int g \frac{Z *_M (\mathbf{w}^*, \alpha, \beta)}{Z_W(\alpha) Z_D(\beta)} P(\alpha, \beta) d\alpha d\beta$$

- It is important to note we do NOT need the posterior of α and β over the entire set of possible architectures (this would be computationally infeasible)
- Instead we wish to compare pre-trained NNs which have found their own $\mathbf{w}^*$ minimum and rank them in some objective manner

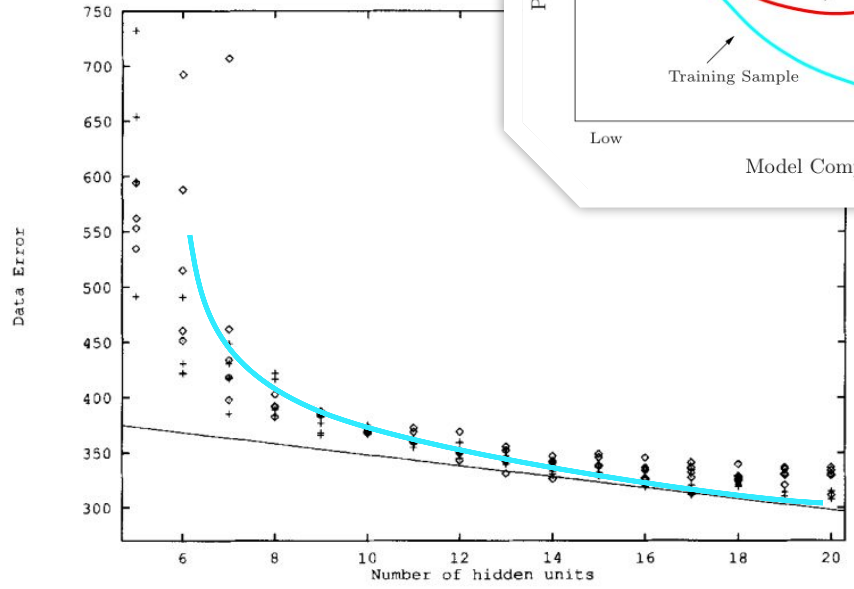
Evidence and Generalization Error

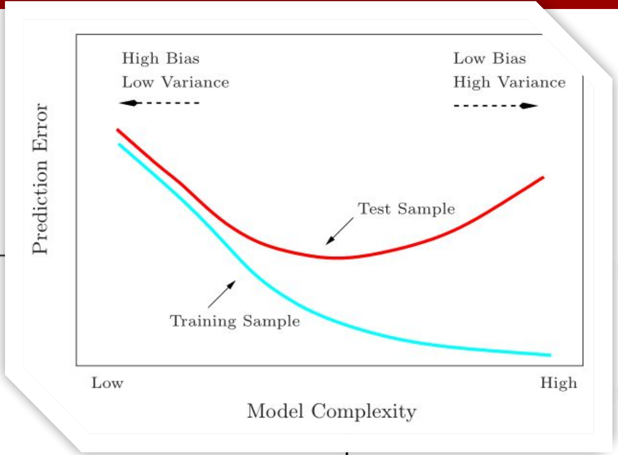
- Evidence and generalization error are correlated
- But evidence is not always a good predictor of generalization error
  - Validation error is noisy, and requires a large held-out dataset
  - If two models yield identical regression results, their gen errors are the same, but evidence may differ due to model complexity (penalized by the Occam factor)

Model Complexity and Generalization Error

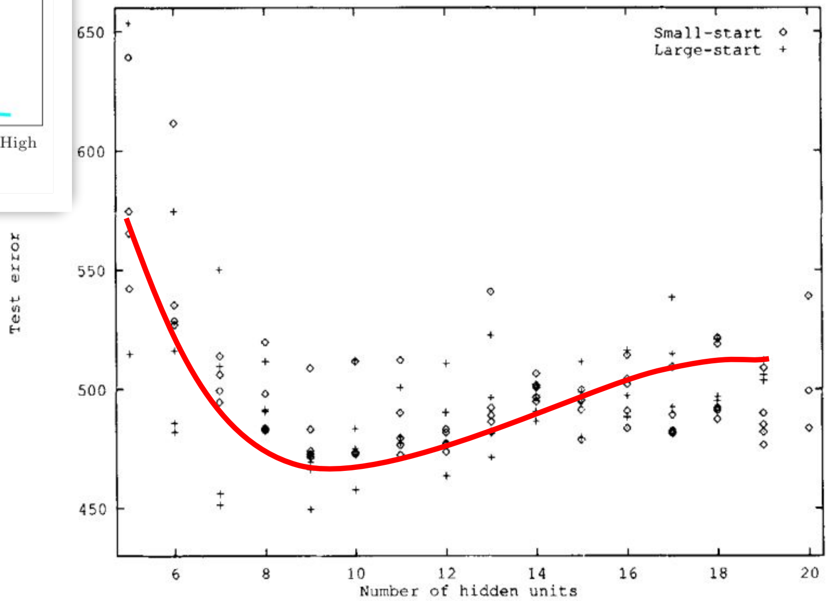
Image from: Hastie, T. et al., Elements of Statistical Learning. *Springer*. p. 38. 2013.

Train Error

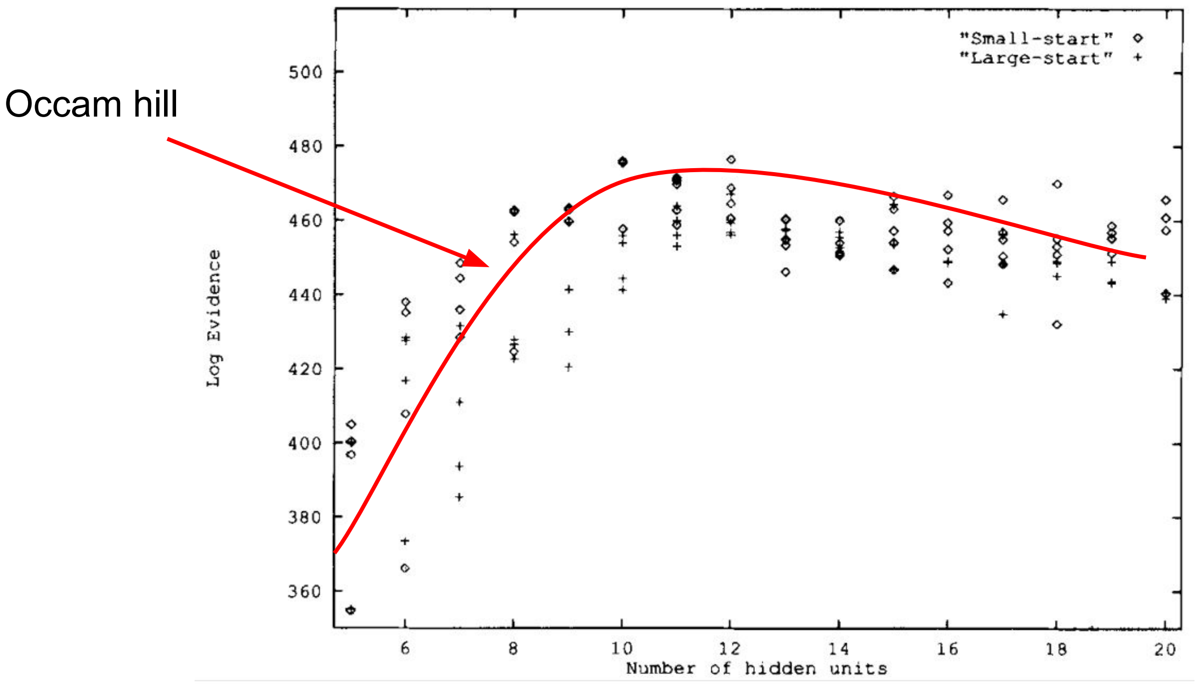




Test Error



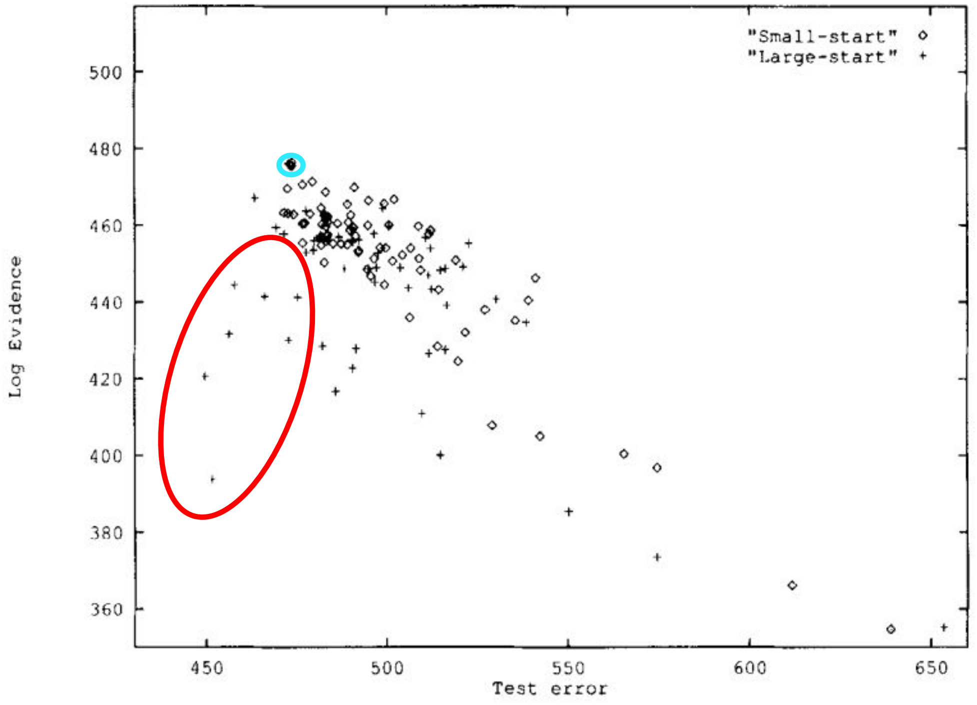
Evidence - Occam Hill



Evidence vs Num Hidden Units

Comparing Models

- The second last slide showed us that using training error on its own will lead to overfitting and poor generalization
- The red circled region shows models with good generalization but low evidence
- This contradicts what we thought Bayesian model comparison does
- We must be missing something!



Test Error vs Log Evidence

Failure as an Opportunity to Learn

- What if the evidence is *low*, but generalization error is *good* (low)?
  - i.e., we have poor correlation between evidence and gen error
- Then the model likely *does not match* the real world
- Learn from the failure: Check and evaluate model assumptions, try new models until one achieves better fit with the data
  - This is a benefit of using Bayesian methods; from gen error, can't discover the inconsistency between the model and data

Inconsistent Prior

- Our loss function is standard, so let's look at our prior more closely

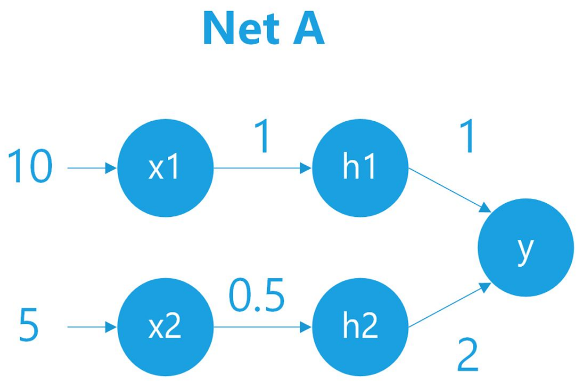
$$E_W(\mathbf{w} \mid \mathcal{A}) = \sum_{i \in c} \frac{w_i^2}{2}$$

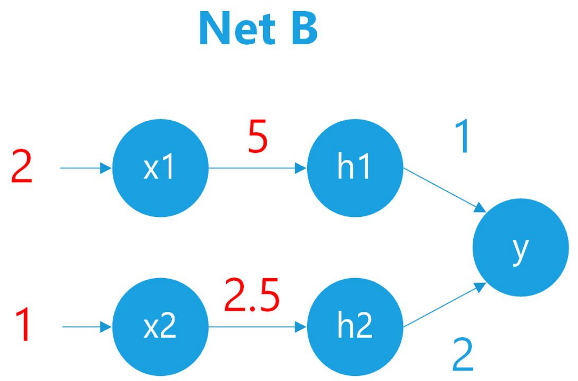
- Suppose we *rescale* the inputs

  Then we could rescale the weights in the first layer and end up with the same mapping

- Our prior is *inconsistent*

  Net B does the same thing yet the *prior penalizes Net B more than Net A*





Original figure by G. A. Adam

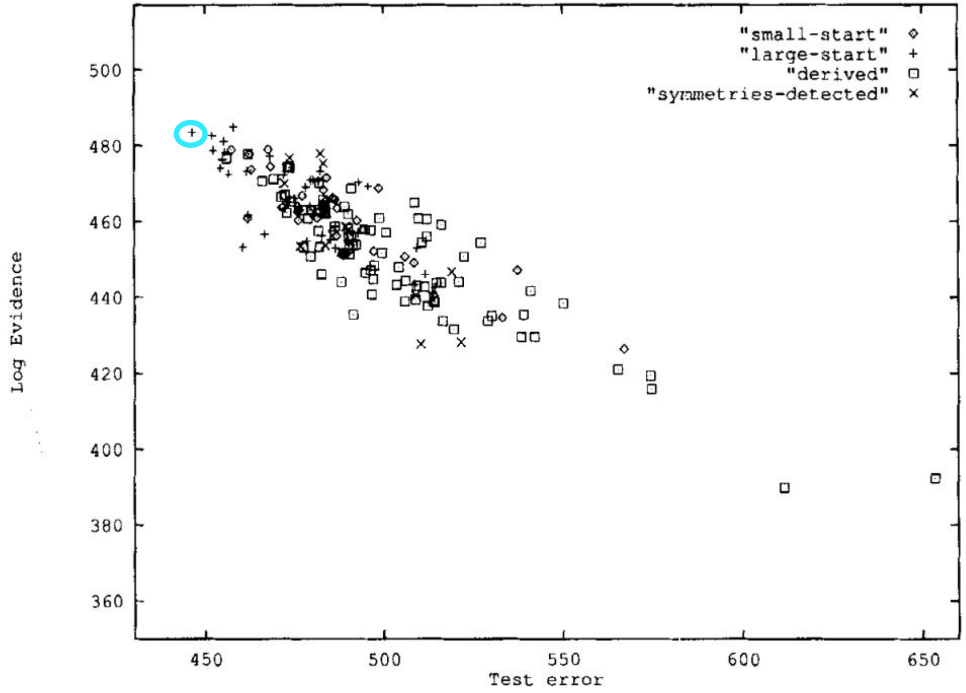
Adjusting the Prior

- The previous prior assumed dependence in the scale of the weights between layers

- Let's use a prior that has independent regularizing constants for each layer:

$$\sum_c \alpha_c E_W^c \qquad E_W^c = \sum_{i \in c} \frac{w_i^2}{2}$$

- Notice how the bottom left region of high evidence but poor generalization no longer exists



Summary from Neal

- Variance of the Gaussian prior for the weights and biases is a hyperparameter
  - Allows the model to adapt to the degree of smoothness indicated by the data
- Improved by using several variance hyperparameters, one for each type of parameter (input-to-hidden weights, hidden biases, and output weights/biases)
- This emphasizes the advantages of hierarchical models
- *Makes sense:* Network inputs and outputs are different quantities with different scales; using a single variance hyperparameter depends on arbitrary choice of measurement units.

Conclusion

- Bayesian evidence is in fact a good predictor of generalization ability
- Combined with generalization error, it can help us determine if we are using an inconsistent regularizer and change our worldview
- Evidence is maximized for neural nets with reasonable numbers of hidden units
- Computational difficulty arises in calculating the Hessian, its inverse, and determinant
- This framework is also applicable to classification problems
  - Error landscape would look totally different since we would be using different loss functions

References

- Blundell, C. et al. Weight Uncertainty in Neural Networks. *ICML 37,* 1613-1622, 2015.
- MacKay, D. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. 2003.
- MacKay, D. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation 4*, 448-472, 1992.
- MacKay, D. Bayesian Interpolation. *Neural Computation 4*, 415-447, 1992.
- Laplace Approximation. https://en.wikipedia.org/wiki/Laplace%27s_method. Retrieved September 25, 2017.
- Pennington, J. and Bahri, Y. Geometry of Neural Network Loss Surfaces via Random Matrix Theory. *PMLR 70*, 2798-2806, 2017.

Priors for Infinite Networks

Based on: *Chapter 2 of Bayesian Learning for Neural Networks*

By: Radford M. Neal

Presented by: Soon Chee Loong

Distributions over Weights and Functions

- The *weights* of a network determine the *function* computed by the network
- In a BNN, the weights are drawn from a *probability distribution*; intuitively, we can interpret the BNN as representing a *distribution over functions*
- The first step in Bayesian inference is the specification of a *prior*
    - e.g., a prior over weights, $P(\mathbf{w})$
- Given a prior over the weights, what is the prior over computed functions?
- Connection between bayesian neural networks and Gaussian Processes

Overview

- How do we decide priors for neural networks?
- A single hidden layer neural network with infinite hidden units **converges** to a Gaussian Process.
- A single hidden layer neural network with infinite hidden units approaches a **limiting distribution**.
- The choice of **hidden function** activation influences the type of functions sampled from the prior.
- Not covered from Radford's Chapter 2 Thesis: ~~**Hierarchical Models**~~

Motivation: Selecting Priors

- Prior represents our beliefs about the problem.
    - Recap: Coin toss problem, heads and tails with 50% probability makes sense
        - *Solve problem due to M.L.E. with coin toss by introducing uniform priors.*
- Neural Networks
    - priors over weights/biases has no obvious connection to input.
- The use of infinite networks makes sense from the standpoint of prior beliefs
    - We usually don't believe that the function we want to learn can be perfectly captured by a finite network.
- Properties of gaussian priors for infinite networks can be found analytically.

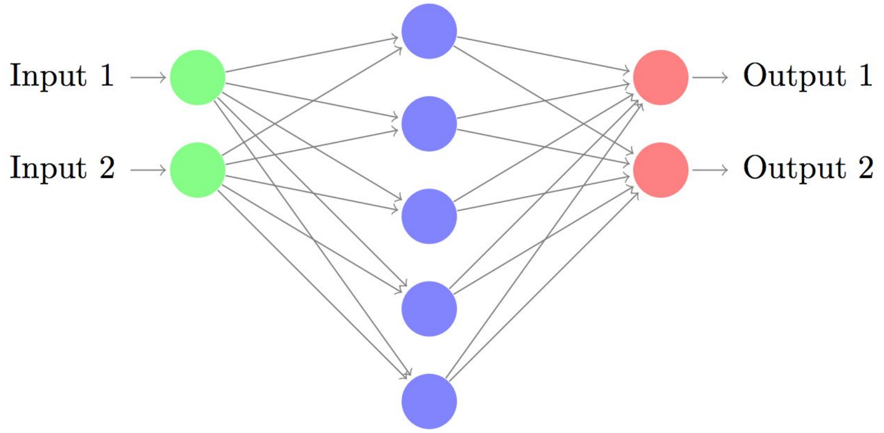
Motivation: Bigger is Better

- **Bayesian Occam Razor:** Bayesian approach:
  - can increase model parameters to infinity without overfitting.
- In practice, limited by computational resources (memory, time)
- NOT restricted based on size of training set.
- Wouldn't we overfit? No, if we regularize it properly.
  - Analogous to Neural Networks:
    - "Make network as big as possible"
    - "Then, regularize using weight decay, dropout." (Jimmy Ba, ECE521 2017 Lecture)
- What should we increase in Neural Networks to Infinity?

Universal Approximation Theorem.

- **Universal Approximation Theorem:**
  - can approximate any continuous function with a neural network with 1 hidden layer.
- Hence, focus on extending hidden layer to infinite number of hidden units.
- Assumption: Computationally feasible to produce mathematically correct results for infinite hidden units.

$$h_j(x) = \tanh\left(a_j + \sum_{i=1}^{I} u_{ij} x_i\right)$$



- Online Open Access Textbooks, 9.3 Neural Network Models

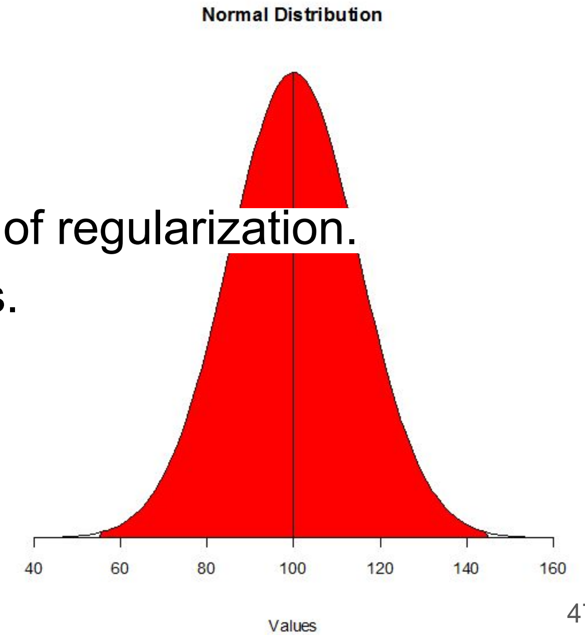
What Priors to Use

- Since no obvious connection, what priors do we use?
- Gaussian with zero mean is standard.
- Historically by David Mackay,
    - Gaussian with zero mean has worked well for his work.
    - Minimize standard deviation of priors as a form of regularization.
        - Similar to weight decay for neural networks.

Hidden to Output Parameters

$$Weights, v_{jk} \sim \mathcal{N}(0,\ \sigma_v^2)$$

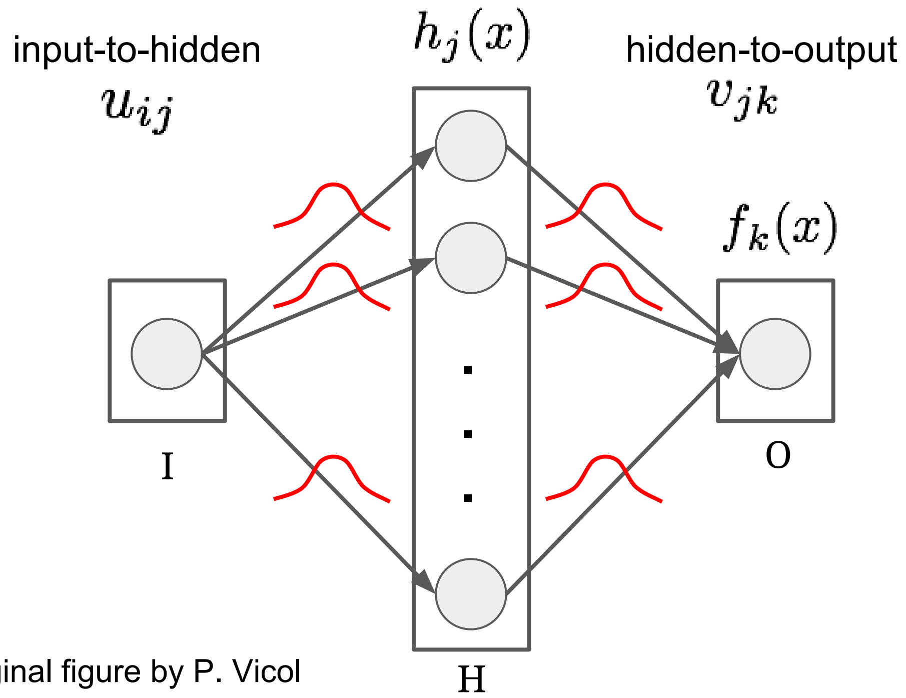
$$Bias, b_k \sim \mathcal{N}(0,\ \sigma_b^2)$$



- Natural Resource Biometrics, NR3110

Infinite Networks → Gaussian Processes

- Consider a Bayesian Neural Network with a single hidden layer:



$$\boxed{\begin{aligned} v_{jk} &\sim \mathcal{N}(0, \sigma_v^2) \\ u_{ij} &\sim \mathcal{N}(0, \sigma_u^2) \end{aligned}}$$

- Examine the prior distribution on the output value for a fixed input $x^{(1)}$

➡ We want to find $P(f_k(x^{(1)}))$

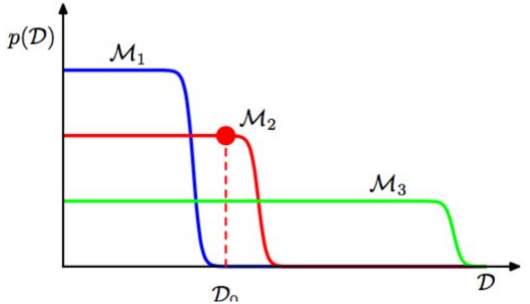
Original figure by P. Vicol

Output Variance Limit Behavior Without Regularization

- Behavior of output
- Total Variance
  - Central Limit Theorem

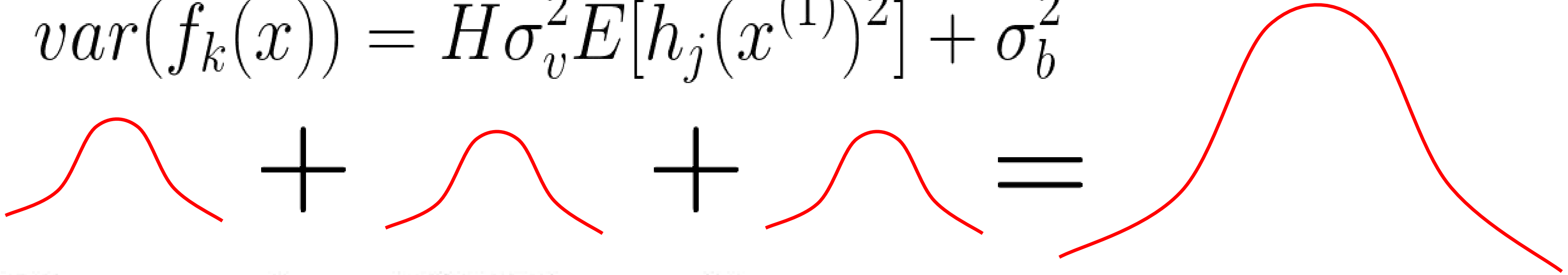
$$f_k(x) = b_k + \sum_{j=1}^{H} v_{jk} h_j(x)$$

$$Bias, b_k \sim \mathcal{N}(0,\ \sigma_b^2)$$



- Bishop, Pattern Recognition and Machine Learning

$$var(f_k(x)) = H\sigma_v^2 E[h_j(x^{(1)})^2] + \sigma_b^2$$

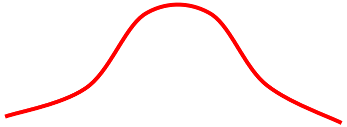


$$H = number\ of\ hidden\ units$$

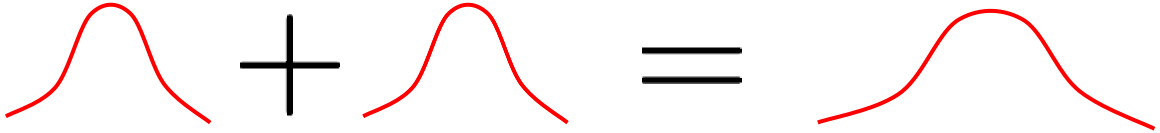
Need to get rid of dependence on H

Output Variance Limit Behavior With Regularization

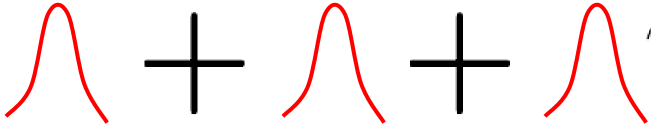
- Reduce initialization variance as form of regularization



$$Weights, v_{jk} \sim \mathcal{N}(0, \sigma_v^2)$$



$$\sigma_v = \omega_v H^{-1/2}$$



$$var(f_k(x)) = H\sigma_v^2 E[h_j(x^{(1)})^2] + \sigma_b^2$$

Output Variance With Regularization

$$f_k(x) = b_k + \sum_{j=1}^{H} v_{jk} h_j(x)$$

$$\sigma_v = w_v H^{\frac{-1}{2}}$$

$$E[v_{jk} h_j(x^{(1)})] = E[v_{jk}] E[h_j(x^{(1)})]$$

$$= 0 * E[h_j(x^{(1)}] = 0$$

$$E[X] = 0$$

$$Var(X) = E[X^2] - (E[X])^2 = E[X^2]$$

Output variance finite!

$$E[(v_{jk} h_j(x^{(1)}))^2] = E[v_{jk}^2] E[h_j(x^{(1)})^2] = w_v^2 E[h_j(x^{(1)})^2] \notin \infty$$

Output Variance With Regularization

$$f_k(x) = b_k + \sum_{j=1}^{H} v_{jk} h_j(x)$$

$$\sigma_v = w_v H^{\frac{-1}{2}}$$

$$E[v_{jk} h_j(x^{(1)})] = E[v_{jk}] E[h_j(x^{(1)})]$$
$$= 0 * E[h_j(x^{(1)}] = 0$$

Proof works for any zero mean, finite variance prior distribution.

$$E[X] = 0$$

$$Var(X) = E[X^2] - (E[X])^2 = E[X^2]$$

Output variance finite!

$$E[(v_{jk} h_j(x^{(1)}))^2] = E[v_{jk}^2] E[h_j(x^{(1)})^2] = w_v^2 E[h_j(x^{(1)})^2] \notin \infty$$

Priors Converge to Gaussian Process

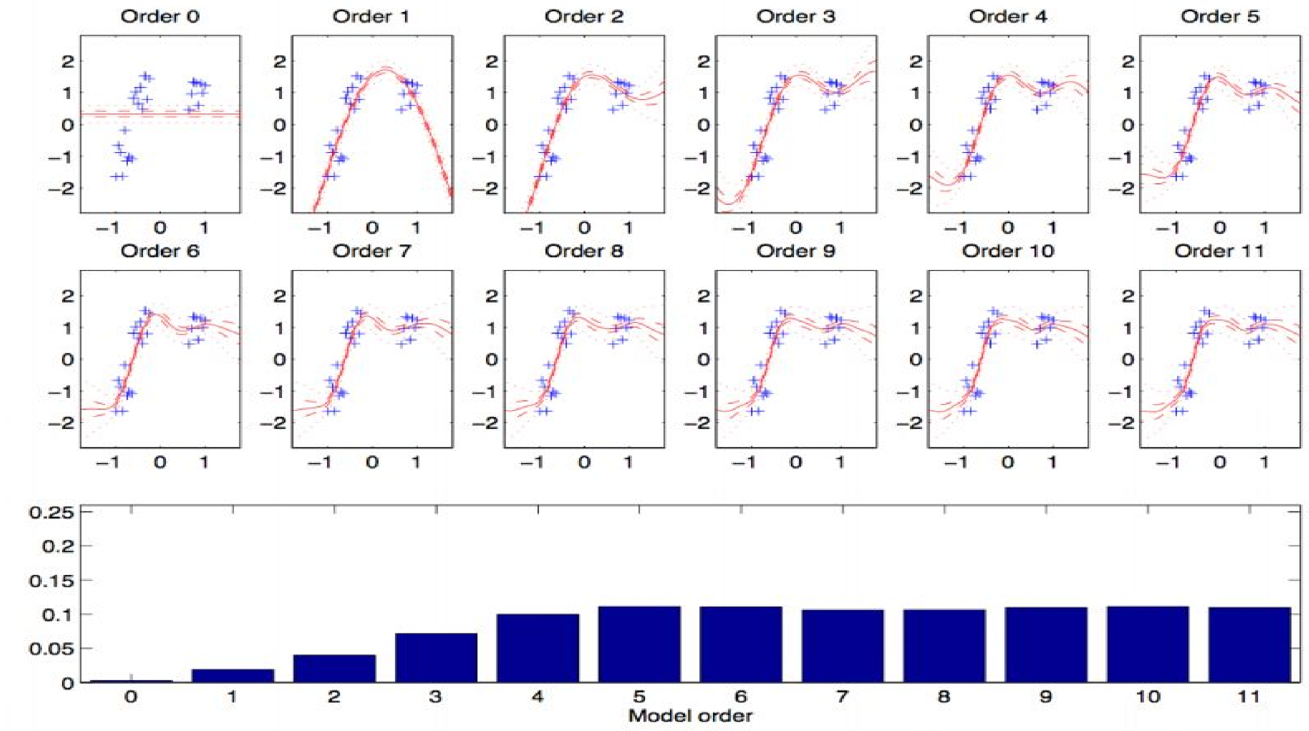
- Similarly, (mathematical proof for Prior Joint Distribution)
- Gaussian Priors converge to Gaussian Process

  as number of hidden units increases.

$$\begin{aligned} E[f_k(x^{(p)})f_k(x^{(q)})] &= \sigma_b^2 + \sum_j \sigma_v^2 E[h_j(x^{(p)})h_j(x^{(q)})] \\ &= \sigma_b^2 + \omega_v^2 C(x^{(p)}, x^{(q)}) \end{aligned}$$

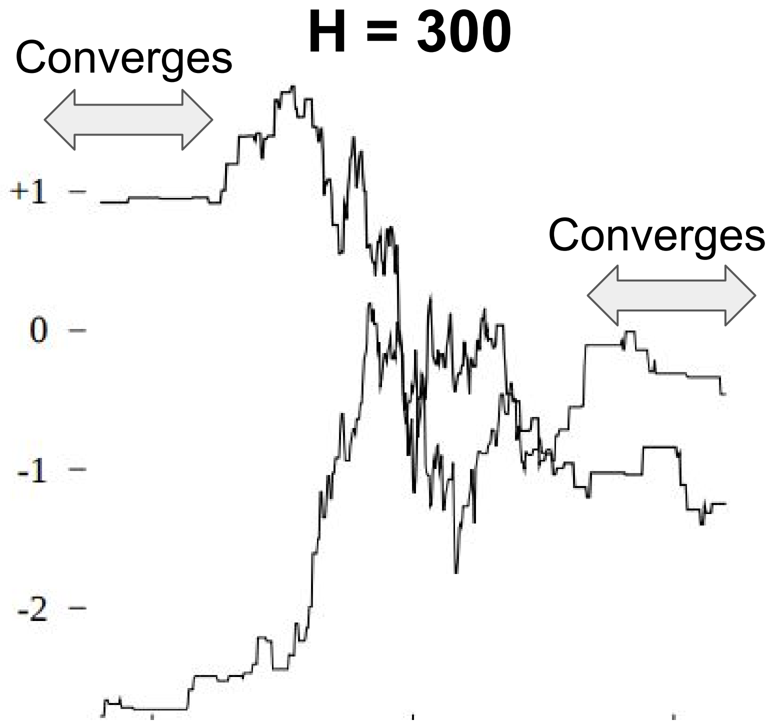
Functions Drawn Approaches Limiting Distribution (From CSC2541 2017: Lecture 2 pg. 36 of 55)

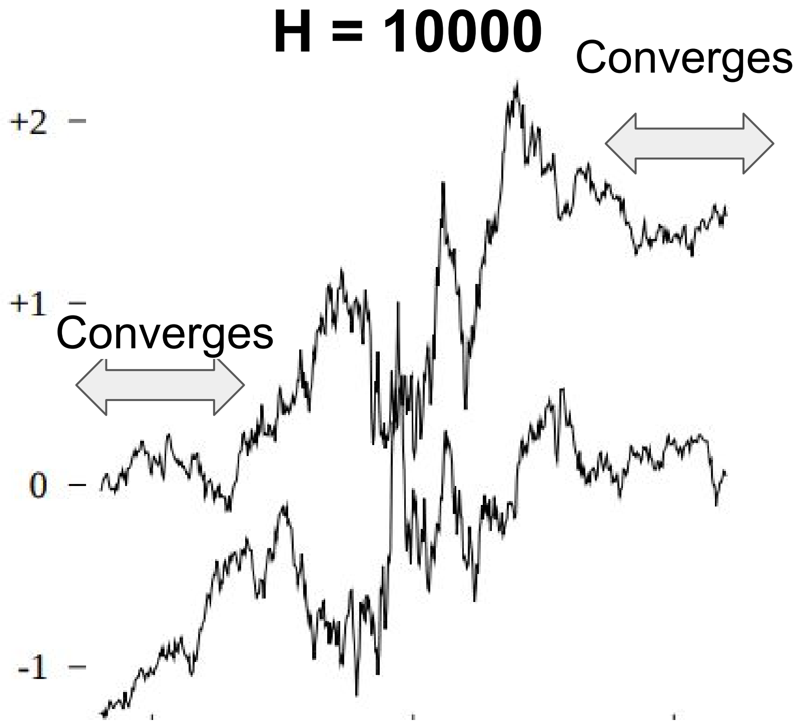
Behavior w/ smoothness prior as we add more basis functions:



— Rasmussen and Ghahramani, "Occam's Razor"

Functions Drawn From Prior Distribution With Step(z) Hidden Activation Approaches Limiting Distribution





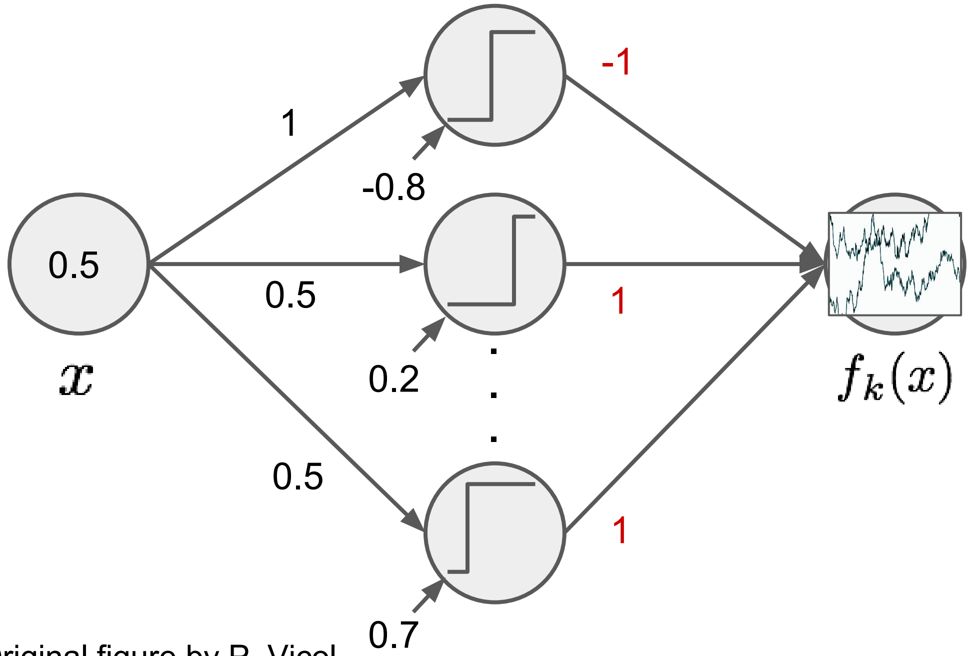
- Radford Neal, PhD Thesis, Chapter 2

Priors to Brownian or Smooth Function

- How does **Hidden Unit Activation** affect output function sampled?
  - h(z) = sign(z)
  - h(z) = tanh(z)
- Gaussian Prior with zero mean.
- Priors properties are determined by the covariance function.
  - Smooth
  - Fractional Brownian
  - Brownian

Priors that Lead to Brownian Functions

- Let input-to-hidden weights and hidden biases have Gaussian distributions

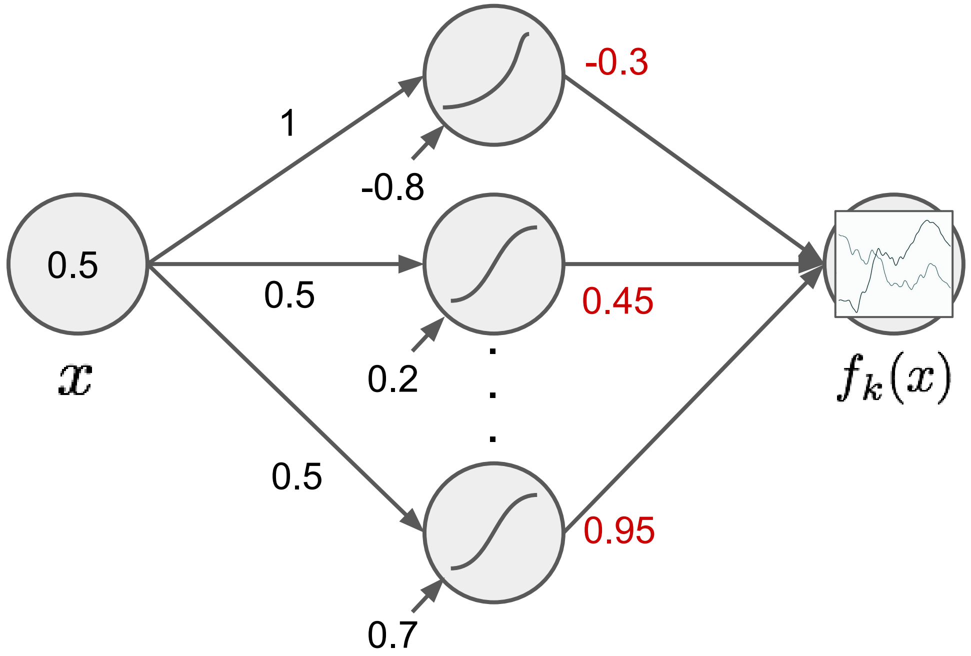


- **Step** activations
- The function is built up of small, independent, **non-differentiable** steps contributed by hidden units.
- **Brownian**

Original figure by P. Vicol

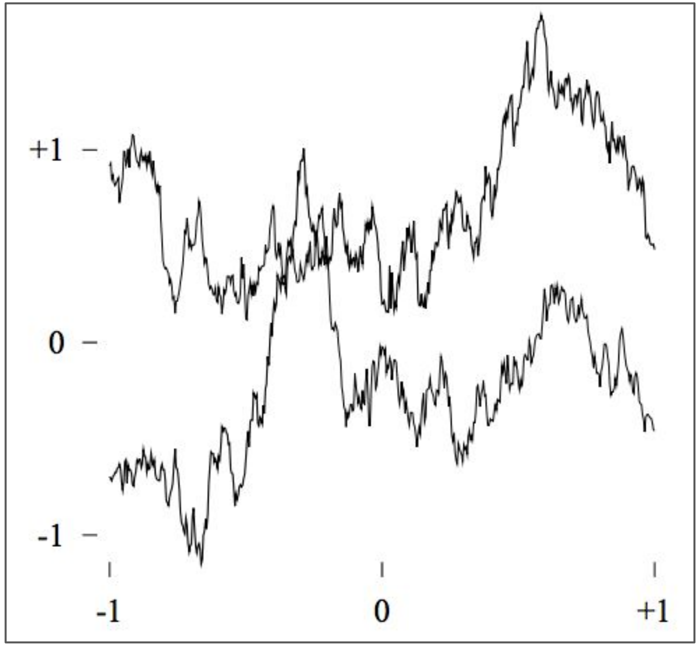
Priors that Lead to Smooth Functions

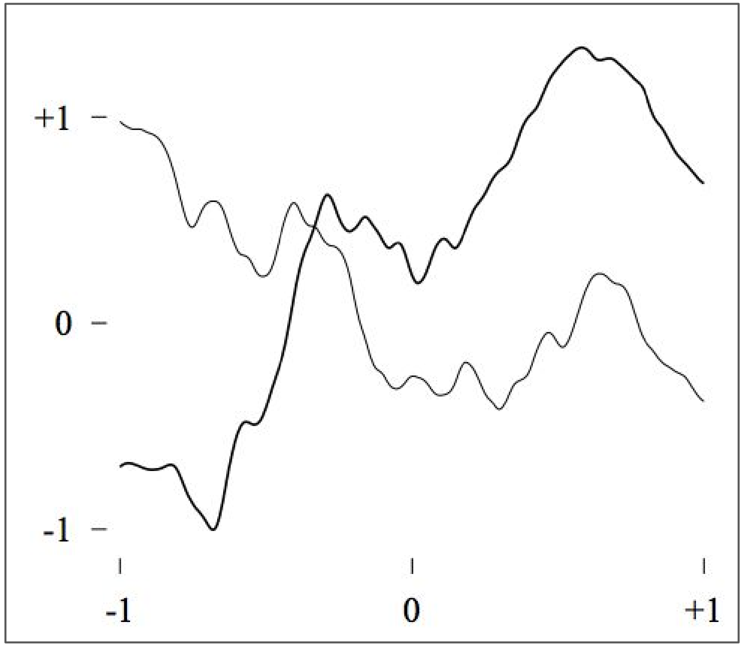
- Let input-to-hidden weights and hidden biases have Gaussian distributions



- **Tanh** activations
- The function is built up of small, independent, **differentiable** tanh contributed by hidden units
- **Smooth**

Priors that Lead to Smooth and Brownian Functions





- Radford Neal, PhD Thesis, Chapter 2

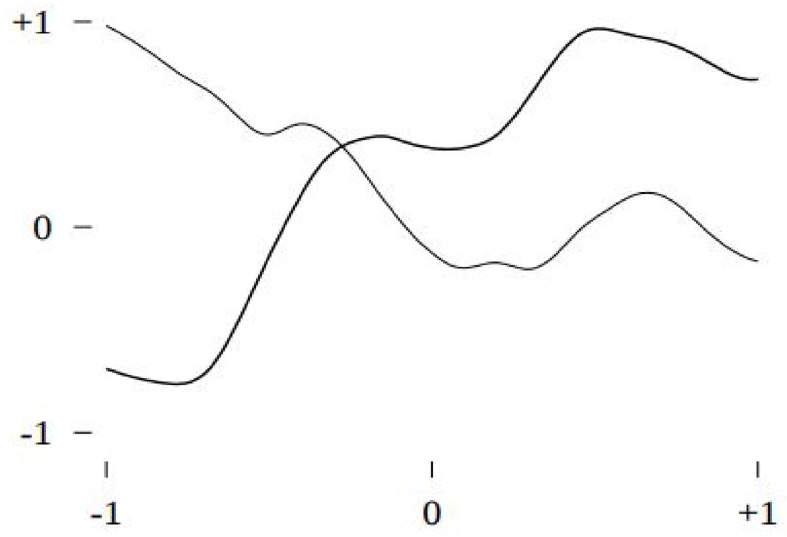
Gaussian priors, ***step()*** hidden units

Gaussian priors, ***tanh()*** hidden units

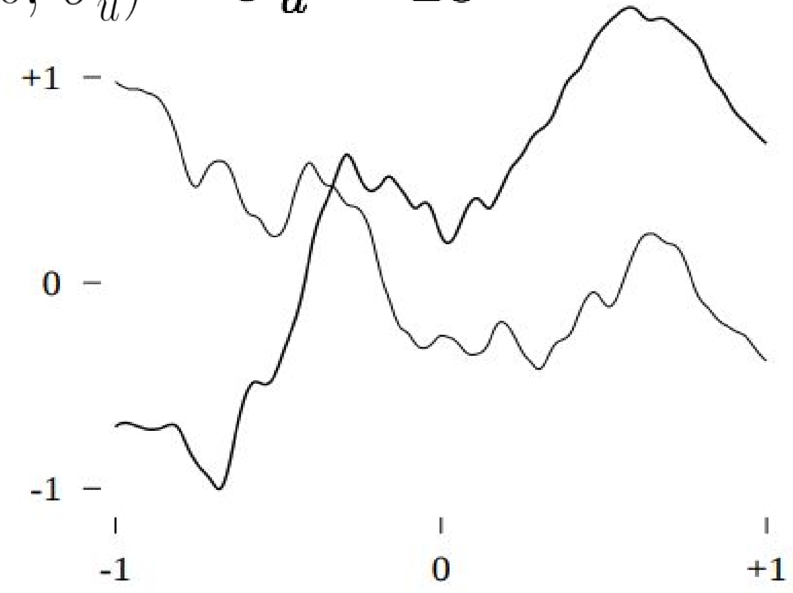
Tanh(): Smooth Converges to Brownian as the prior over the mean increases to infinity

$u_{ij} \sim \mathcal{N}(0, \sigma_u^2)$

$\sigma_u = 5$



$\sigma_u = 20$



- Radford Neal, PhD Thesis, Chapter 2

Gaussian priors, $\tanh$ hidden units

Priors to Brownian or Smooth Function

- Behavior is based on Covariance Function
- Re-write covariance function in terms of the differences.

$$E[h_j^{x^{(p)}} h_j^{x^{(q)}}]$$

$$= \frac{1}{2}(2E[h_j^{x^{(p)}} h_j^{x^{(q)}}])$$

$$= \frac{1}{2}(0 + 2E[h_j^{x^{(p)}} h_j^{x^{(q)}}])$$

$$= \frac{1}{2}(E[(h_j^{x^{(p)}})^2] + E[(h_j^{x^{(q)}})^2] - E[(h_j^{x^{(p)}})^2] - E[(h_j^{x^{(q)}})^2] + 2E[h_j^{x^{(p)}} h_j^{x^{(q)}}])$$

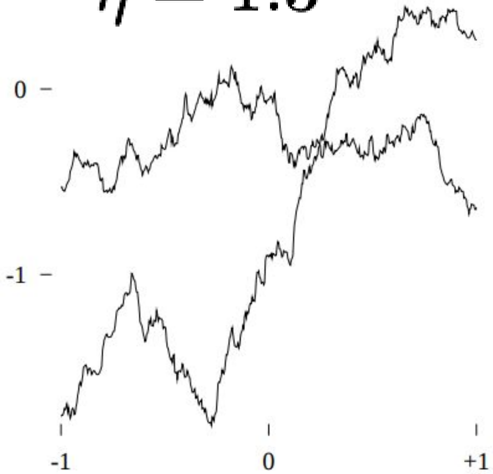
$$= -\frac{1}{2}(E[(h_j^{x^{(q)}} - h_j^{x^{(p)}})^2] + other\ terms)$$

Prior over Function Covariance Properties

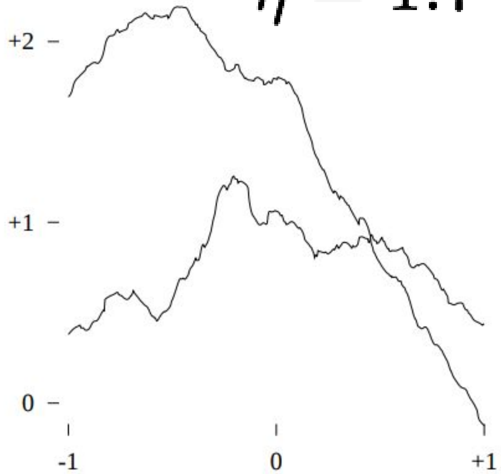
- Priors properties are determined by the covariance function.
  - Smooth
    - $\eta = 2$
  - Fractional Brownian
    - $\eta \in (1, 2)$
  - Brownian
    - $\eta = 1$

$$E[(h_j^{x^{(p)}} - h_j^{x^{(q)}})^2] = |x^{(p)} - x^{(q)}|^\eta$$

$\eta = 1.3$



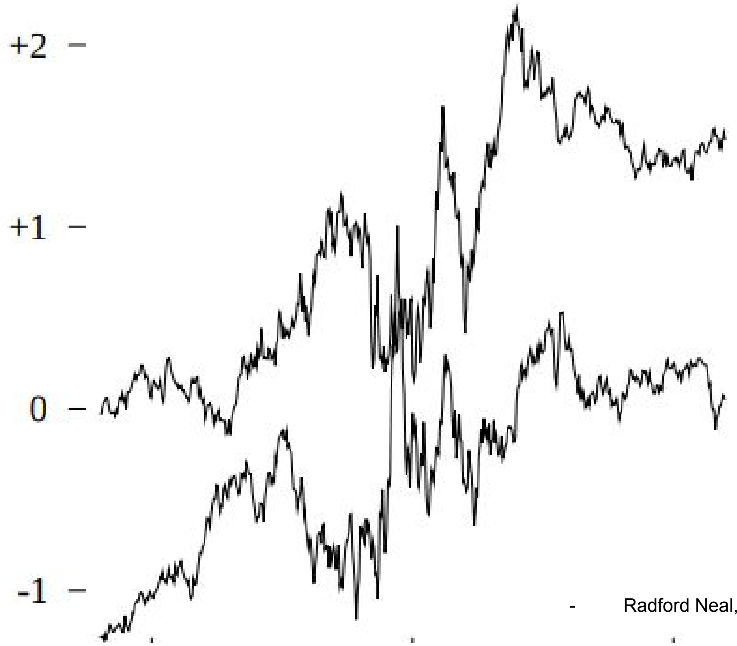
$\eta = 1.7$



- Radford Neal, PhD Thesis, Chapter 2

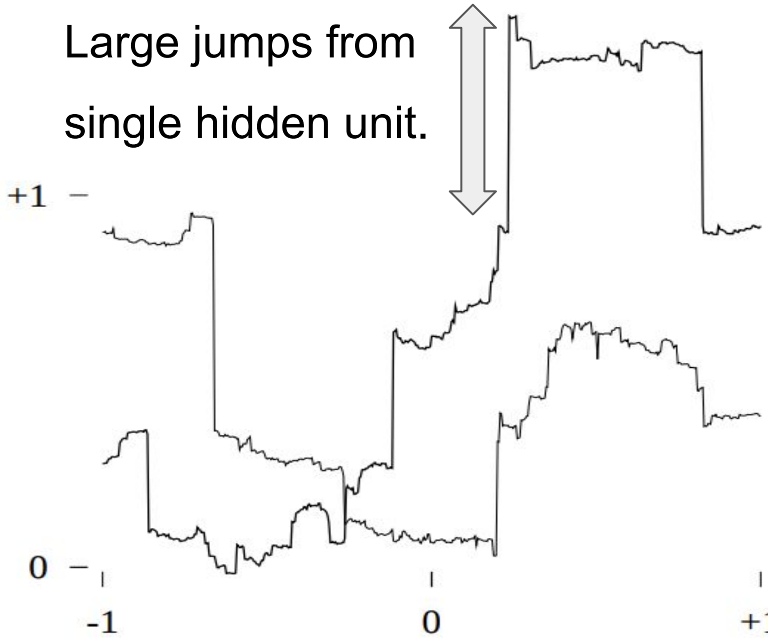
Non Gaussian Prior (Cauchy) with Step Hidden Activation

Gaussian Prior with Step()



Non-Gaussian Prior (Cauchy) with Step()

Large jumps from single hidden unit.



GPs vs NNs

- When would we use NNs vs GPs?
- GPs are just “smoothing devices” (MacKay, 2003)
  - Are NNs over-hyped, or are GPs underestimated? (MacKay says *both*.)

Neural Networks

- Optimizing parameters of a network
- Can solve many problems not solvable by GPs: e.g., representation learning

Gaussian Processes

- Simple matrix operations on the covariance matrix of the GP
- GPs are easy to implement and use - few parameters must be specified by hand
- Inverting an NxN matrix is expensive - Can’t scale to large datasets (N > 1000)

Hamiltonian Monte Carlo

Based on: *MCMC using Hamiltonian Dynamics* by Radford Neal

Presented by: Tristan Aumentado-Armstrong, Guodong Zhang, Chris Cremer

Markov Chain Monte Carlo (MCMC) I

- Recall: in Bayesian analysis, we often desire integrals like

$$P(y|x,\mathcal{D}) = \int P(y|x,\theta)P(\theta|\mathcal{D})d\theta$$ (Posterior Predictive Dist.)

$$\mathbb{E}[f(x|\theta)] = \int f(x|\theta)P(\theta|\mathcal{D})d\theta$$ (Expectation of y=f(x|θ))

- But how can we actually evaluate these integrals when θ is very high dimensional? Use *Markov Chain Monte Carlo (MCMC),* which is much less affected by dimensionality.

Markov Chain Monte Carlo (MCMC) II

- Monte Carlo is a way of performing this integration, by transforming the problem in the following way:

$$\mathbb{E}[f(x|\theta)] = \int f(x|\theta)P(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_{i=1}^{N} f(x|\theta_i)$$

where θ_i is distributed according to the posterior for the parameters, P(θ|D)

- This transforms our integral into a sampling problem, so that we now just need a way to sample from the posterior

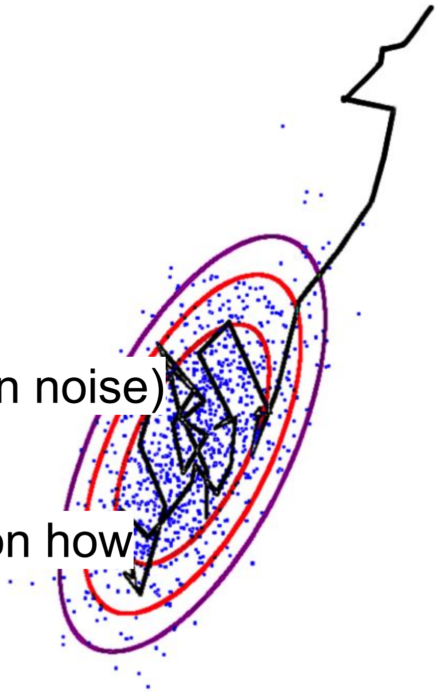
The Metropolis-Hastings (MH) Algorithm I

- MH is an MCMC algorithm for sampling from the posterior P(θ|D)
- Intuition: Run a Markov chain with stationary distribution P(θ|D)
- Algorithm (Assume: Q(θ) α P(θ|D))
  - Start from initial state θ_0
  - Iterate i = 1 to n:
    - Propose: $\hat{\theta} \sim q_{\text{proposal}}(\theta|\theta_{i-1})$
    - Acceptance Probability

$$\theta_i = \begin{cases} \hat{\theta} & \text{with prob. } \min\left\{\frac{q_{\text{proposal}}(\theta_{i-1}|\hat{\theta}))Q(\hat{\theta})}{q_{\text{proposal}}(\hat{\theta}|\theta_{i-1}))Q(\theta_{i-1})}, 1\right\} \\ \theta_{i-1} & \text{otherwise} \end{cases}$$

The Metropolis-Hastings (MH) Algorithm II

- Given enough time, MH converges to sampling from the stationary distribution
  - At this point, states are samples from the posterior (as desired)
- Common proposal choice: *Random Walk MH*
  - The proposal perturbs the current state (e.g. Gaussian noise) and is symmetric
  - The new proposed state is accepted/rejected based on how likely the parameters are according to the posterior



Murray, MCMC Slides, Machine Learning Summer School 2009

The Metropolis-Hastings (MH) Algorithm III

- This accept-reject step ensures that the state update (transition) satisfies the *equations of detailed balance*

$$T(\theta|\hat{\theta})Q(\hat{\theta}) = T(\hat{\theta}|\theta)Q(\theta)$$

  where $T(\theta|\hat{\theta})$ is the state transition probability

- Such chains are *reversible*, which is desirable for MCMC algorithms
  - Reversibility can be used to show the MCMC updates don’t alter Q
  - This is sufficient for the chain’s stationary distribution to exist and, in this case, equal our posterior by construction

The Metropolis-Hastings (MH) Algorithm IV

- Choosing a proposal distribution: e.g. $q_{\text{proposal}}(\hat{\theta}|\theta) = \mathcal{N}(\theta, \sigma)$
    - Balance exploration (reach areas with support) & visiting high probability areas more
    - Control the random walk step size with $\boldsymbol{\sigma}$
        - Too large: too many rejections
        - Too small: explores the space too slowly
- Drawbacks of the Random Walk MH Algorithm
    - The algorithm may find it very difficult to move long distances in parameter space
        - Random walks are not very efficient explorers
    - If $\boldsymbol{\sigma}$ is too large or small, then the samples will be too dependent (lower effective sample size)
- We want a way to move larger distances, yet still have a decent chance of acceptance - Idea: prefer moving along level sets of an energy related to Q

Hamiltonian Monte Carlo - Motivation

- For MCMC, the distribution we wish to sample can be related to a potential energy function via the concept of **canonical distribution** from statistical mechanics

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

Canonical distribution

- We can draw samples from canonical distribution using random walk Metropolis (**guess-and-check**). But it cannot produce **distant** proposals with **high** acceptance probability.

Hamiltonian Monte Carlo - Motivation

- The **key** here is to exploit additional information to guide us through the neighborhood with high target probability.

Gradient !!!

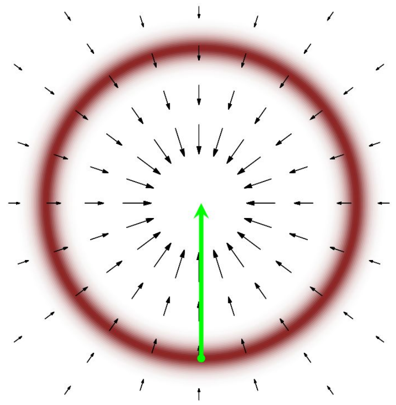


image credit: A Conceptual introduction to Hamiltonian Monte Carlo

Hamiltonian Monte Carlo - Motivation

- Following along the gradient would pulls us towards the **mode of the target density**. We lose the chance to explore new and unexplored areas.

Momentum !!!

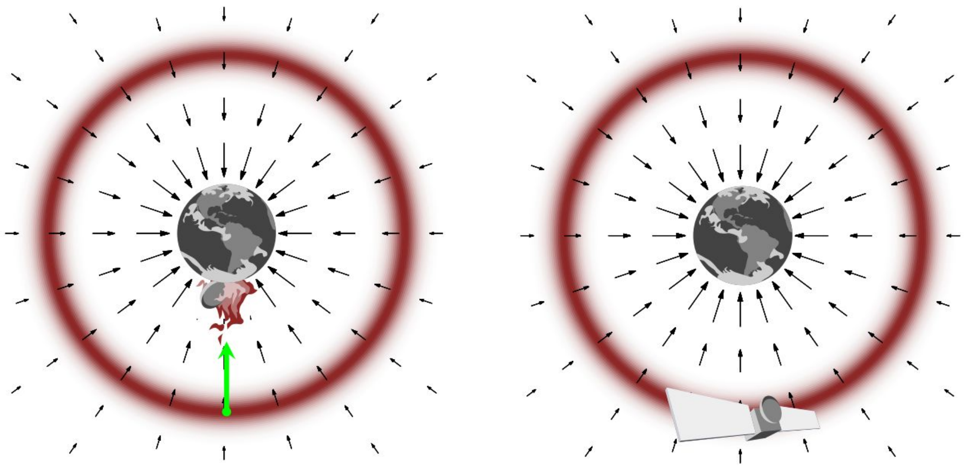


image credit: A Conceptual introduction to Hamiltonian Monte Carlo

Hamiltonian energy

- Introduce the momentum p (where q is variable of interest, say position)

$$q \rightarrow (q, p)$$

- We can now lift up the target distribution onto a joint distribution

$$\pi(q, p) = \pi_1(q)\pi_2(p)$$

- By the definition of canonical distribution, the expanded system defines a **Hamiltonian energy** that decomposes into a potential energy and kinetic energy.

$$\begin{tikzcd} & H(q,p) = U(q) + K(p) \arrow[dl] \arrow[dr] & \\ U(q) = -\log \pi_1(q) + \text{const} & & K(p) = -\log \pi_2(p) + \text{const} \arrow[d] \\ & & \frac{p^2}{m} \end{tikzcd}$$

Note: for convenience, I just use one-dimensional notations in my slide.

Hamiltonian dynamics

- Hamiltonian dynamics describe how kinetic energy is converted to potential energy (and vice versa) as a particle moves throughout a system in time.

- This description is implemented quantitatively via a set of differential equations known as the **Hamilton's equations**:

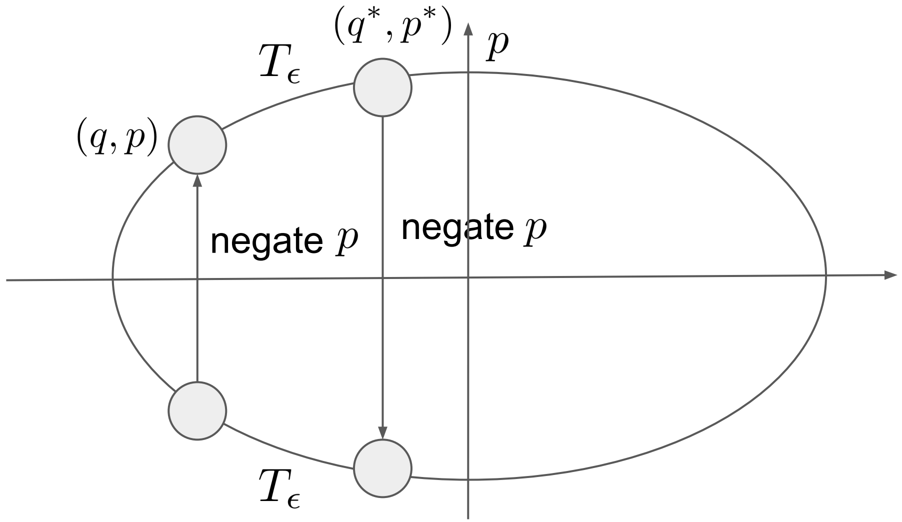
$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K(p)}{\partial p} = \frac{p}{m}$$

$$\frac{dp}{dt} = ma = -\frac{\partial H}{\partial q} = -\frac{\partial U(q)}{\partial q}$$

- These equations define a mapping T_ϵ from t to $t + \epsilon$

Property - Reversibility

Property 1: For the mapping T_ϵ from (q, p) to (q^*, p^*), we can find inverse mapping by first negating p, applying T_ϵ, and negating p again.



Note: Detailed balance requires each transition is reversible.

Property - Hamiltonian Conservation

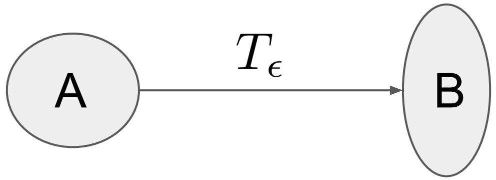
Property 2: Hamiltonian H doesn't have a functional dependence on time. It's invariant over time.

$$\begin{aligned}\frac{dH}{dt} &= \frac{dq}{dt}\frac{\partial H}{\partial q} + \frac{dp}{dt}\frac{\partial H}{\partial p} \\ &= \frac{\partial H}{\partial p}\frac{\partial H}{\partial q} - \frac{\partial H}{\partial q}\frac{\partial H}{\partial p} = 0\end{aligned}$$

Note: In practice, Hamiltonian is **approximately** invariant.

Property - Volume Preservation

Property 3: In Hamiltonian dynamics, any contraction or expansion in position space must be compensated by a respective expansion or compression in momentum space.



Sufficient and necessary condition: the determinant of Jacobian matrix of the mapping having absolute value one

Leave target distribution invariant

- Hamiltonian Conservation

$$\pi(q, p) = \pi(q^*, p^*)$$

- Reversibility

$$\pi^*(q*, p*) = \pi(q, p)|J_\epsilon|$$

- Volume Preservation

$$|J_\epsilon| = 1$$

$$\pi^*(q^*, p^*) = \pi(q^*, p^*)$$

Discretizing Hamilton's equations

- For computer implementation, Hamilton's equations must be approximated by discretizing time, using some small stepsize, ϵ.
- The best known way to approximate the solution is **Euler's method**

$$p(t+\epsilon) = p(t) + \epsilon \frac{dp}{dt}(t) = p(t) - \epsilon \frac{\partial U(q(t))}{\partial q}$$

$$q(t+\epsilon) = q(t) + \epsilon \frac{dq}{dt}(t) = q(t) + \epsilon \frac{p(t)}{m}$$

- Euler's method is **not** volume preserving and **not** reversible.

$$|J_\epsilon| = 1 + \epsilon^2 \frac{\partial^2 H}{\partial q^2} \frac{\partial^2 H}{\partial p^2}$$

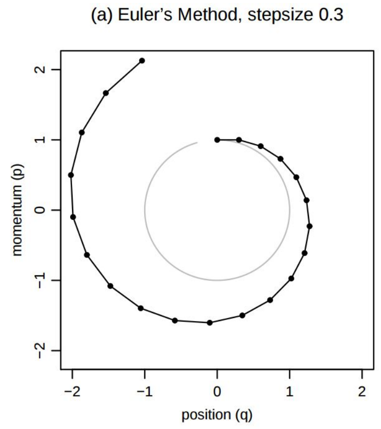


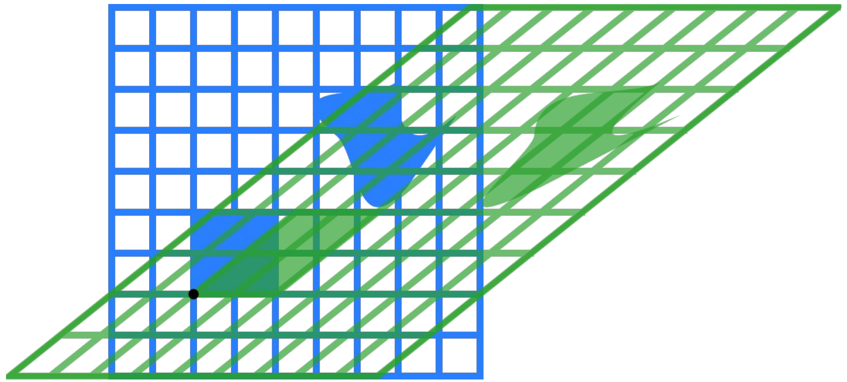
image credit: MCMC using Hamiltonian dynamics

Leapfrog method

$$p(t+\frac{\epsilon}{2}) = p(t) - \frac{\epsilon}{2}\frac{\partial U}{\partial q}(q(t))$$

$$q(t+\epsilon) = q(t) - \epsilon\frac{p(t+\epsilon/2)}{m}$$

$$p(t+\epsilon) = p(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial U}{\partial q}(q(t+\epsilon))$$


https://en.wikipedia.org/wiki/Shear_mapping

Note: each step is **"shear" transformation** which is volume preservation.

Accept / Reject

- In practice, using finite stepsizes ϵ will not preserve the Hamiltonian exactly and will introduce bias in the simulation.

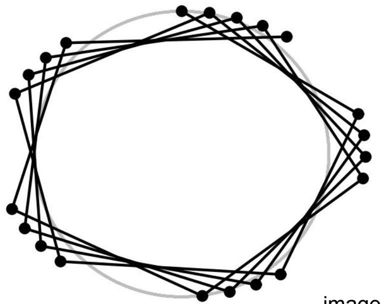
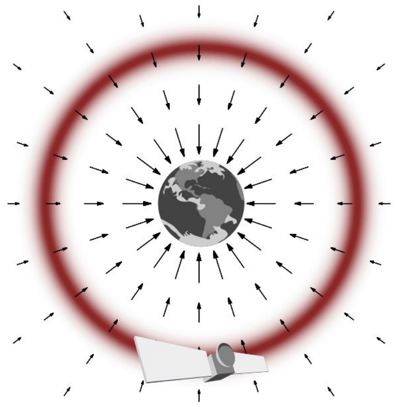


image credit: MCMC using Hamiltonian dynamics

- HMC cancels these effects **exactly** by adding a Metropolis accept/reject stage, after n leapfrog steps, the proposed state will be accepted with the probability $A((q^*, p^*), (q, p))$, defined as

$$\min\left\{1, \frac{exp(-H(q^*, p^*)}{exp(-H(q, p))}\right\}$$

Summary



- **Sample momentum** from its canonical distribution
- Perform n leapfrog steps and obtain the proposed state
- Accept/reject the proposed state

MH vs HMC

Metropolis-Hastings

```
● Initialize x
● For s=1,2...N:
    ○ Sample from proposal
      $x' \sim q(x'|x)$
    ○ Accept sample with probability:
      $\min(1, p(x')q(x|x')/p(x)q(x'|x))$
    ○ If accept:
          $x = x'$
    ○ Else:
          $x = x$
```

Hamiltonian Monte Carlo

```
● Initialize x
● For s=1,2...N:
    ○ Sample momentum
      $v = N(0,M)$
    ○ Simulate Hamiltonian dynamics
      $x',v' = LF(x,v)$
    ○ Accept sample with probability:
      $\min(1, p(x',v')/p(x,v))$
    ○ If accept:
          $x = x'$
    ○ Else:
          $x = x$
```

Visualization

- MH: https://chi-feng.github.io/mcmc-demo/app.html#RandomWalkMH,banana
- HMC: https://chi-feng.github.io/mcmc-demo/app.html#HamiltonianMC,banana

HMC - Bayesian Perspective

- Goal: Sample x ~ p(x)
  - Construct Markov chain p(x'|x)
  - Want:
    - High acceptance probability: p(x')/p(x) (more efficient, less rejections)
    - Distant proposals: x' far from x (better mixing)
- Introduce auxiliary variable v and integrate it out
  - x ~ p(x) = $\int_v p(x,v)$ (sampling momentum)
  - Chain becomes: p(x',v'|x,v)
  - Acceptance becomes: p(x',v')/p(x,v)
    - This ratio can stay high while x' can be very different than x
    - Hamiltonian dynamics achieves the desired properties

How do we know we’re sampling the correct distribution?

- Detailed Balance (sufficient but not necessary)

$$p(x)p(x'|x) = p(x')p(x|x')$$

Why: $\int_{x'} p(x')p(x|x') = \int_{x'} p(x)p(x'|x) = p(x)\int_{x'} p(x'|x) = p(x)$

Metropolis-Hastings

$$\begin{aligned} p(x)p(x'|x) &= p(x)q(x'|x)A(x'|x) \\ &= p(x)q(x'|x)\min\left(1, \frac{p(x')q(x|x')}{p(x)q(x'|x)}\right) \\ &= \min\left(p(x)q(x'|x), p(x')q(x|x')\right) \\ &= \min\left(p(x')q(x|x'), p(x)q(x'|x)\right) \\ &= p(x')p(x|x') \end{aligned}$$

Hamiltonian Monte Carlo

$$\begin{aligned} p(x,v)p(x',v'|x,v) &= p(x,v)A(x',v'|x,v) \\ &= p(x,v)\min\left(1, \frac{p(x',v')}{p(x,v)}\right) \\ &= \min\left(p(x,v), p(x',v')\right) \\ &= \min\left(p(x',v'), p(x,v)\right) \\ &= p(x',v')p(x,v|x',v') \end{aligned}$$

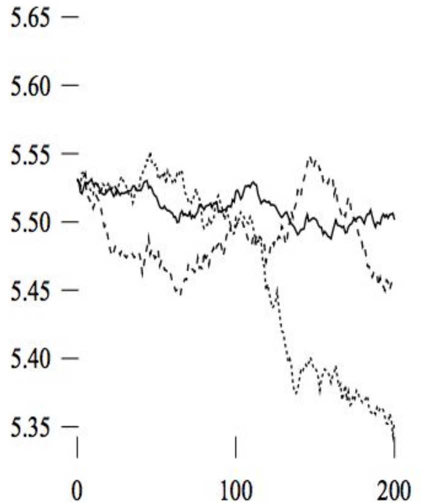
HMC for BNNs

- Radford Neal Thesis - Chapter 3

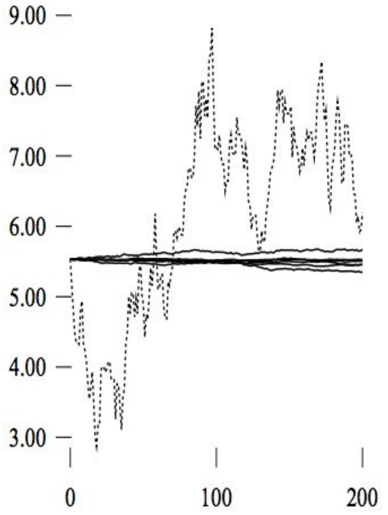
Exploration Experiment

Y-axis: square root of the average squared magnitude of the hidden-output weights

X-axis: super-transitions (2000 steps)



Simple Metropolis



Vs. Hybrid Monte Carlo

	Proposal	Acceptance
Solid:	.1	76%
Dotted:	.3	39%
Dashed:	.9	4%
HMC	.3	87%

Data Sub-Sampling in MCMC

- Problem:
  - Computing likelihood for MH acceptance step requires the whole dataset
  - For HMC, also need gradient of whole dataset

- Stochastic Gradient HMC (2014)
- Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach (2014)
- Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget (2014)

References

- MCMC & MH Introduction/Tutorials
  - Neal, Bayesian Learning for Neural Networks, Chapter 1, 1995
  - Robert, The Metropolis-Hastings Algorithm, 2016
  - Yildirim, Bayesian Inference: Metropolis-Hastings Sampling, 2012

Stochastic Gradient Langevin Dynamics

Based on: *Bayesian Learning via Stochastic Gradient Langevin Dynamics*
By: Max Welling, Yee Whye Teh

Bayesian Dark Knowledge
By: Balan et al.

Presented by: Alexandra Poole, Yuxing Zhang, Jackson K-C Wang

Overview

- Bayesian learning for small mini-batches
  - ☆ Bridging **optimization** and **Bayesian learning**
    - Recall Lecture 1: learning MAP versus learning the posterior distribution
- Simple framework that transitions from optimization to posterior sampling
- 2 perspectives on the algorithm
  - Adding Gaussian noise to Stochastic Gradient Descent (SGD) updates
  - Mini-batch Langevin Dynamics (LD)
- This paper is **not**
  - just proposing a new optimizer

Langevin Dynamics

Bayesian Learning

Full Batch GD

$$\Delta\Theta_t = \frac{\epsilon}{2}\left(\nabla \log\, p(\Theta_t) + \sum_{i=1}^{N} \nabla \log\, p(x_i|\Theta_t)\right)$$

Langevin Dynamics

$$\Delta\Theta_t = \frac{\epsilon}{2}\left(\nabla \log\, p(\Theta_t) + \sum_{i=1}^{N} \nabla \log\, p(x_i|\Theta_t)\right) + \eta_t$$

$$\eta_t \sim \mathcal{N}(0, \epsilon)$$

- Langevin Dynamics is used for the proposal step in Metropolis-adjusted Langevin Algorithm(MALA)
  - MALA is a technique of MCMC **(proper posterior sampling technique)**
- Only a slight modification of Full Batch GD
  - Injects Gaussian noise to parameter updates

The reject/accept step from the classic MALA framework is dropped here, because when epsilon is small enough, the acceptance rate approaches 1.

$$A(\Theta'|\Theta, X) = \min\left(1, \frac{P(\Theta'|X)g(\Theta|\Theta', X)}{P(\Theta|X)g(\Theta'|\Theta, X)}\right)$$

Langevin Dynamics

- **MALA**: animation
    - Href: https://chi-feng.github.io/mcmc-demo/app.html#MALA,banana
    - Why is having a small stepsize important in this paper? (try it!)
        - Notice how when stepsize is reduced, the acceptance rate goes up!

SGD

Optimization

Full Batch GD

$$\Delta\Theta_t = \frac{\epsilon}{2}\left(\nabla \log p(\Theta_t) + \sum_{i=1}^{N} \nabla \log p(x_{ti}|\Theta_t)\right)$$

Mini Batch GD (SGD)

$$\Delta\Theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\Theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\Theta_t)\right)$$

$$\epsilon_t \text{ should satisfy } \sum_{t=1}^{\infty} \epsilon_t = \infty \text{ and } \sum_{t=1}^{\infty} \epsilon_t^2 < \infty, \text{ e.g. } \epsilon_t = a(b+t)^{-\gamma},\ \gamma \in (.5, 1]$$

In SGD, at each iteration t, update is performed based on a subset of data, $X_t = \{x_{t1}, ..., x_{tn}\}$

- approximating the true gradient over the whole dataset
- N, and n can differ by orders of magnitude (e.g. 128 vs 1,000,000)
    - In practice, optimization of NN (non-Bayesian) appears to take a long time (large number of *iterations*), but it usually translates to <50 passes over the full dataset (*epoch*).
    - But 50 samples for MCMC is definitely not enough

SGLD

Full Batch GD

$$\Delta\Theta_t = \frac{\epsilon}{2}\left(\nabla \log p(\Theta_t) + \sum_{i=1}^{N} \nabla \log p(x_i|\Theta_t)\right)$$

Langevin Dynamics

$$\Delta\Theta_t = \frac{\epsilon}{2}\left(\nabla \log p(\Theta_t) + \sum_{i=1}^{N} \nabla \log p(x_i|\Theta_t)\right) + \eta_t$$

$$\eta_t \sim \mathcal{N}(0, \epsilon)$$

Mini Batch GD (SGD)

$$\Delta\Theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\Theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\Theta_t)\right)$$

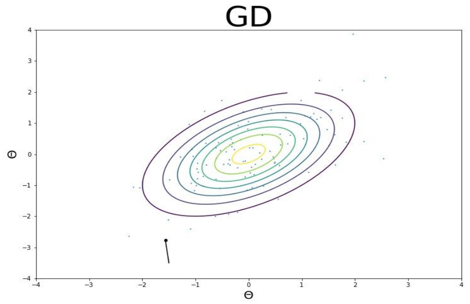
SGLD

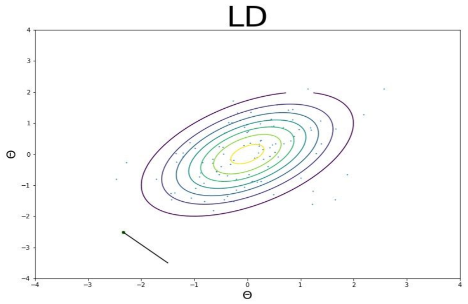
$$\Delta\Theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\Theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\Theta_t)\right) + \eta_t$$

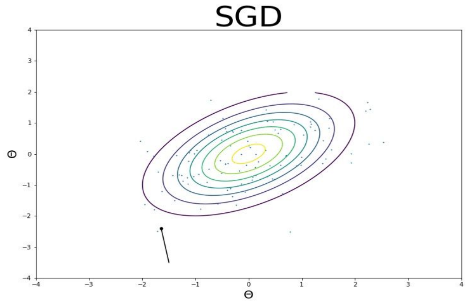
$$\eta_t \sim \mathcal{N}(0, \epsilon_t)$$

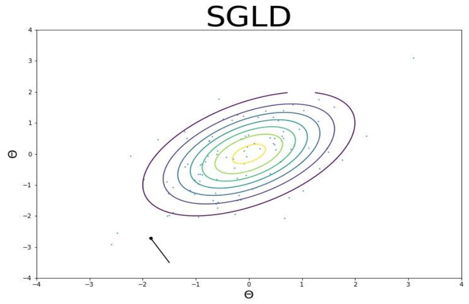
Visually speaking…

The figures are actually animations in the presentation.
Please visit: https://github.com/wangkua1/SGLD-presentation-supp/tree/master









Algorithm 1 SGLD

```
1: procedure TRAIN-PHASE($X$)
2:     initialize $\Theta, \epsilon_0$                    ▷ as you would in typical NN training
3:     $\boldsymbol{\Theta} \leftarrow \{\}$
4:     $\boldsymbol{\epsilon} \leftarrow \{\}$
5:     for $t = 0, ..., $ max iterations do
           $$\Delta\Theta_t = \frac{\epsilon_t}{2}\Big(\nabla \log p(\Theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\Theta_t)\Big) + \eta_t$$
           $$\eta_t \sim \mathcal{N}(0, \epsilon)$$
6:         $\Theta_{t+1} \leftarrow \Theta_t + \Delta\Theta_t$
7:         if in posterior sampling phase then        ▷ e.g. $\epsilon_t \leq$ some threshold
8:             $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}||\Theta_{t+1}$
9:             $\boldsymbol{\epsilon} \leftarrow \boldsymbol{\epsilon}||\epsilon_t$
10:        end if
11:    end for
12:    return $\boldsymbol{\Theta}, \boldsymbol{\epsilon}$
13: end procedure
14: ________________________________________
15: procedure TEST-PHASE($\boldsymbol{\Theta}, \boldsymbol{\epsilon}, f$)
16:     return
           $$\frac{\sum_{t=1}^{T} \epsilon_t f(\Theta_t)}{\sum_{t=1}^{T} \epsilon_t}$$
                                                    ▷ MC estimate according to the posterior
17: end procedure
```

Justify SGLD

- What approximation is SGLD making to MALA, and why is it still valid MCMC?
  - First approximation: when epsilon is small enough, the accept/reject is ignored
  - Second approximation: using subsampled gradient to approximate true gradient

Let's rewrite SGLD

$$g(\theta) = \nabla \log p(\theta) + \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$$

$$h_t(\theta) = \nabla \log p(\theta) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta) - g(\theta)$$

$$\Delta\theta_t = \frac{\epsilon_t}{2}(g(\theta_t) + h_t(\theta)) + \eta_t, \quad \eta_t \sim N(0, \epsilon_t)$$

$$\text{Find } t_1 < t_2 < \cdots \text{ such that } \sum_{t=t_s+1}^{t_{s+1}} \epsilon_t \to \epsilon_0 \text{ as } s \to \infty$$

Assumptions: g is Lipschitz continuous

$h_t(\theta_t)$ for each t is iid

Approximately recovers LD

$$\sum_{t=t_s+1}^{t_{s+1}} \left[\frac{\epsilon_t}{2}(g(\theta_t) + h_t(\theta_t)) + \eta_t\right]$$

$$= \sum_{t=t_s+1}^{t_{s+1}} \frac{\epsilon_t}{2}(g(\theta_t) + h_t(\theta_t)) + \sum_{t=t_s+1}^{t_{s+1}} \eta_t$$

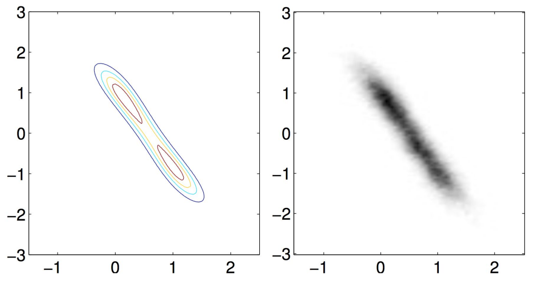
$$= \frac{\epsilon_0}{2} g(\theta_{t_s}) + O(\epsilon_0) + \sum_{t=t_s+1}^{t_{s+1}} \frac{\epsilon_t}{2} h_t(\theta_t) + O(\sqrt{\epsilon_0})$$

$$= \frac{\epsilon_0}{2} g(\theta_{t_s}) + O(\epsilon_0) + O\left(\sqrt{\sum_{t=t_s+1}^{t_{s+1}} \frac{\epsilon_t^2}{4}}\right) + O(\sqrt{\epsilon_0})$$

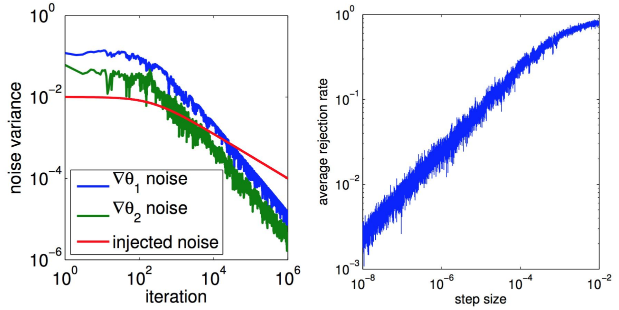
$$= \frac{\epsilon_0}{2} g(\theta_{t_s}) + O(\sqrt{\epsilon_0})$$

Experiments

Gaussian Mixture



True and estimated posterior distribution. Adapted from Welling, Max, and Yee W. Teh, 2011



Left: variances of stochastic gradient noise and injected noise. Right: rejection probability versus step size. Adapted from Welling, Max, and Yee W. Teh, 2011

$$\theta_1 \sim N(0,10); \quad \theta_2 \sim N(0,1)$$

$$x \sim \frac{1}{2}N(\theta_1, 2) + \frac{1}{2}N(\theta_1+\theta_2, 2)$$

Transition threshold

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(g(\theta_t) + h_t(\theta_t)\right) + \eta_t, \;\; \eta_t \sim N(0, \epsilon_t)$$

$$V[h_t(\theta_t)] \approx \frac{N^2}{n^2}\sum_{i=1}^{n}(s_{ti} - \bar{s}_t)(s_{ti} - \bar{s}_t)^\top,$$

$$\text{where } s_{ti} = \nabla \log p(x_{ti}|\theta_t), \;\; \bar{s}_t = \frac{1}{n}\sum_{i=1}^{n} s_{ti}$$

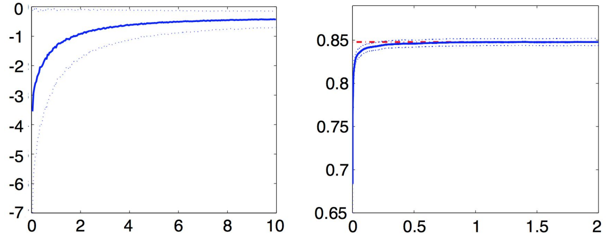
$$V[\frac{\epsilon_t}{2}(g(\theta_t) + h_t(\theta_t))] = \frac{\epsilon_t^2}{4}V[h_t(\theta_t)]$$

$$V(\eta_t) = \epsilon_t I$$

$$\boxed{\frac{\epsilon_t}{4}\lambda_{\max}(V[h_t(\theta_t)]) \ll 1}$$

Experiments

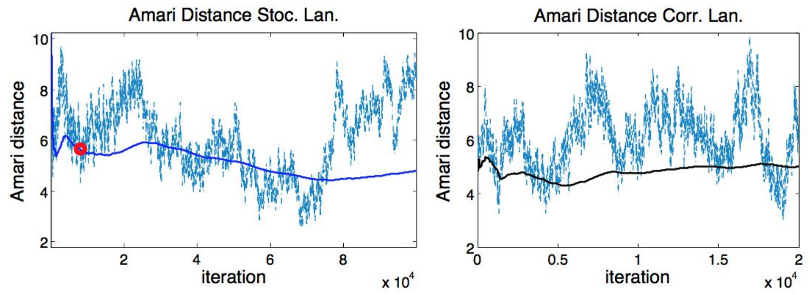
Logistic Regression



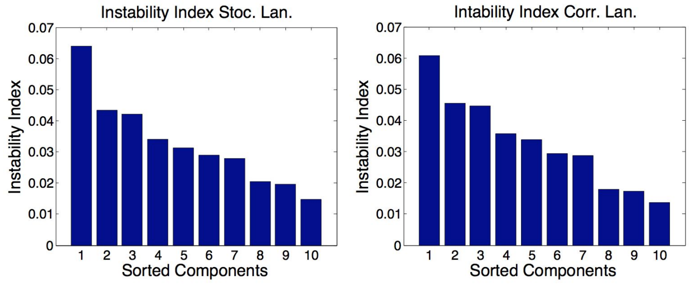
Average log joint probability per data item (left) and accuracy on test set (right) as functions of the number of sweeps through the whole dataset. Red dashed line represents accuracy after 10 iterations. Results are averaged over 50 runs; blue dotted lines indicate 1 standard deviation. Adapted from Welling, Max, and Yee W. Teh, 2011

$$p(y_i|x_i) = \sigma(y_i\beta^\top x_i)$$
$$p(\beta) = \frac{1}{2}\exp(-|\beta|)$$

Independent Components Analysis



Amari distance over time for stochastic Langevin dynamics and corrected Langevin dynamics. Thick line represents the online average. Adapted from Welling, Max, and Yee W. Teh, 2011



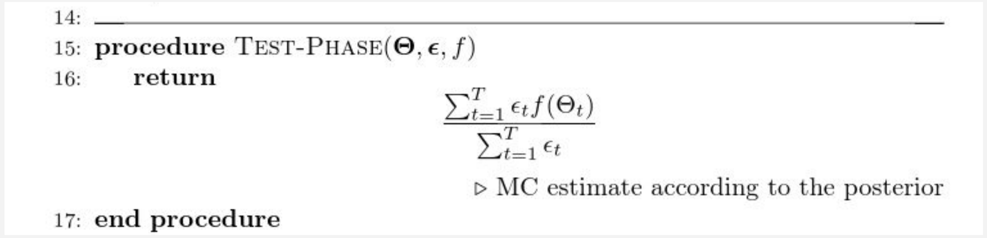
$$\mathcal{I}_i = \sum_i var(W_{ij})var(x_j)$$

Instability index for the 10 independent components computed for stochastic Langevin dynamics and corrected Langevin dynamics on MEG. Adapted from Welling, Max, and Yee W. Teh, 2011

Problems with SGLD

- Wasting memory:
  - Many copies of the parameters need to be stored
  - For S number of samples, memory requirements are S times bigger
- Wasting time:
  - Makes predictions using many versions of the model
  - For S number of samples, speed will be S times slower than an ML estimate

Recall from SGLD:



```
procedure TEST-PHASE($\boldsymbol{\Theta}, \boldsymbol{\epsilon}, f$)
    return
        $\frac{\sum_{t=1}^{T} \epsilon_t f(\Theta_t)}{\sum_{t=1}^{T} \epsilon_t}$
        ▷ MC estimate according to the posterior
end procedure
```

Bayesian Dark Knowledge - Distillation

Algorithm 1: Distilled SGLD

```
Input: $\mathcal{D}_N = \{(x_i, y_i)\}_{i=1}^N$, minibatch size $M$, number of iterations $T$, teacher learning schedule $\eta_t$, student learning schedule $\rho_t$, teacher prior $\lambda$, student prior $\gamma$
for $t = 1 : T$ do
    // Train teacher (SGLD step)
    Sample minibatch indices $S \subset [1, N]$ of size $M$
    Sample $z_t \sim \mathcal{N}(0, \eta_t I)$
    Update $\theta_{t+1} := \theta_t + \frac{\eta_t}{2}\left(\nabla_\theta \log p(\theta|\lambda) + \frac{N}{M}\sum_{i \in S} \nabla_\theta \log p(y_i|x_i, \theta)\right) + z_t$
    // Train student (SGD step)
    Sample $\mathcal{D}'$ of size $M$ from student data generator
    $w_{t+1} := w_t - \rho_t\left(\frac{1}{M}\sum_{x' \in \mathcal{D}'} \nabla_w \hat{L}(w, \theta_{t+1}|x') + \gamma w_t\right)$
```

$$\hat{L}(w|\theta^s, x) = -\sum_{k=1}^{K} p(y = k|x, \theta^s) \log \mathcal{S}(y = k|x, w) \qquad (4)$$

If we denote the predictions of the teacher by $p(y|x, \mathcal{D}_N)$ and the parameters of the student network by w, our objective becomes

$$\begin{aligned} L(w|x) &= \mathrm{KL}(p(y|x, \mathcal{D}_N)||\mathcal{S}(y|x, w)) = -\mathbb{E}_{p(y|x,\mathcal{D}_N)} \log \mathcal{S}(y|x, w) + \text{const} \\ &= -\int \left[\int p(y|x, \theta)p(\theta|D_N)d\theta \right] \log \mathcal{S}(y|x, w)dy \\ &= -\int p(\theta|D_N) \int p(y|x, \theta) \log \mathcal{S}(y|x, w)dy \; d\theta \\ &= -\int p(\theta|D_N) \left[\mathbb{E}_{p(y|x,\theta)} \log \mathcal{S}(y|x, w) \right] d\theta \end{aligned} \tag{1}$$

Unfortunately, computing this integral is not analytically tractable. However, we can approximate this by Monte Carlo:

$$\hat{L}(w|x) = -\frac{1}{|\Theta|} \sum_{\theta^s \in \Theta} \mathbb{E}_{p(y|x,\theta^s)} \log \mathcal{S}(y|x, w) \tag{2}$$

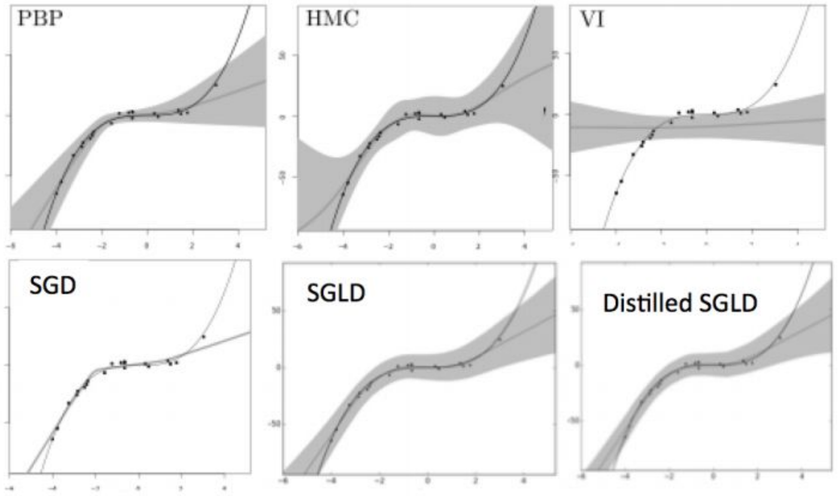
where Θ is a set of samples from $p(\theta|\mathcal{D}_N)$.

$$\begin{aligned} \hat{L}(w) &= \int p(x)L(w|x)dx \approx \frac{1}{|\mathcal{D}'|} \sum_{x' \in \mathcal{D}'} L(w|x') \\ &\approx -\frac{1}{|\Theta|} \frac{1}{|\mathcal{D}'|} \sum_{\theta^s \in \Theta} \sum_{x' \in \mathcal{D}'} \mathbb{E}_{p(y|x',\theta^s)} \log \mathcal{S}(y|x', w) \end{aligned} \tag{3}$$

SGD [BCKW15]	Dropout	BBB	SGD (our impl.)	SGLD	Dist. SGLD
1.83	1.51	1.82	1.536 ± 0.0120	1.271 ± 0.0126	1.307 ± 0.0169

Table 3: Test set misclassification rate on MNIST for different methods using a 784-400-400-10 MLP. SGD (first column), Dropout and BBB numbers are quoted from [BCKW15]. For our implementation of SGD (fourth column), SGLD and distilled SGLD, we report the mean misclassification rate over 10 runs and its standard error.

We start with a toy 1d regression problem, in order to visually illustrate the performance of different methods. We use the same data and model as [HLA15]. In particular, we use $N = 20$ points in $D = 1$ dimensions, sampled from the function $y = x^3 + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, 9)$. We fit this data with an MLP with 10 hidden units and ReLU activations. For SGLD, we use $S = 2000$ samples. For distillation, the teacher uses the same architecture as the student.



Takeaways from SGLD

- Perspective
  - ☆ Bridging **optimization** and **Bayesian learning**
- Justification
  - ☆ First approximation: when epsilon is small enough, the accept/reject is ignored
  - ☆ Second approximation: using subsampled gradient to approximate true gradient
- Experiments/Evaluations
  - ☆ Small rejection rate
  - ☆ Fast optimization
  - ☆ Good approximation to MALA

Making SGLD better?

- Various angles to consider
  - From modern neural networks
    - Variants of SGD are popular (e.g. with momentum, 2nd order approx.) in

      use it within SGLD?
      - ➢ With momentum: SGHMC
      - ➢ 2nd order approx.: stochastic gradient Fisher scoreing (SGFS)
    - SGLD assumes smoothly changing gradients, but popular non-linearity in

      might not be
      - ➢ Use weight clipping like in WGAN?
  - From other MCMC techniques
    - SGHMC, and many extensions (see page 2 on the Bayesian Dark Knowledge paper)

References

Originally Assigned Readings from instructor (href: https://csc2541-f17.github.io/)
Welling, Max, and Yee W. Teh. "Bayesian learning via stochastic gradient Langevin dynamics." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.

Balan, Anoop Korattikara, et al. "Bayesian dark knowledge." *Advances in Neural Information Processing Systems*. 2015.

Extensions mentioned -
[SGHMC] Chen, Tianqi, Emily Fox, and Carlos Guestrin. "Stochastic gradient hamiltonian monte carlo." *International Conference on Machine Learning*. 2014.

[SGFS] Ahn, Sungjin, Anoop Korattikara, and Max Welling. "Bayesian posterior sampling via stochastic gradient Fisher scoring." *arXiv preprint arXiv:1206.6380* (2012).

[WGAN] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein gan." *arXiv preprint arXiv:1701.07875* (2017).

Thank you!

Q/A