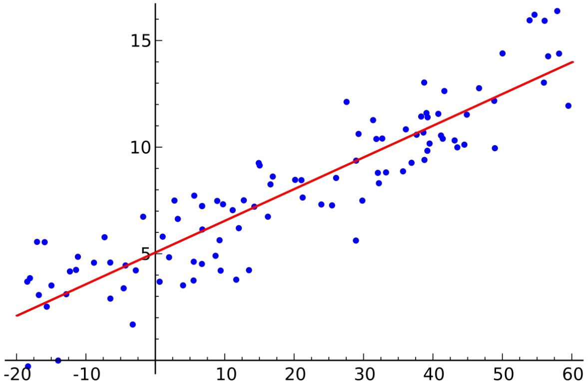
Bayesian deep learning

Dr. Luca Ambrogioni

Part I: From SGD to gradient-based MCMC

Dr. Luca Ambrogioni

Linear regression



Predictive model: $$y_j = W x_j$$

Loss function: $$\mathcal{L}(W) = \frac{1}{2}\sum_j (y_j - W x_j)^2$$

Linear regression by stochastic gradient descent I

SGD update:

$$W_{n+1} = W_n - \eta \sum_{j \in \text{minibatch}} \nabla \mathcal{L}_j(W)$$

Gradient:

$$\nabla \mathcal{L}_j(W) = \frac{1}{2} \nabla \left(y_j - W_j x_j \right)^2 = - \left(y_j - W_j x_j \right) x_j^T$$

Least squares as maximum likelihood

$$\mathcal{L}(W) = -\frac{1}{2}\sum_j (y_j - Wx_j)^2 = \log \prod_j \mathcal{N}(y_j \mid Wx_j, 1) + c$$

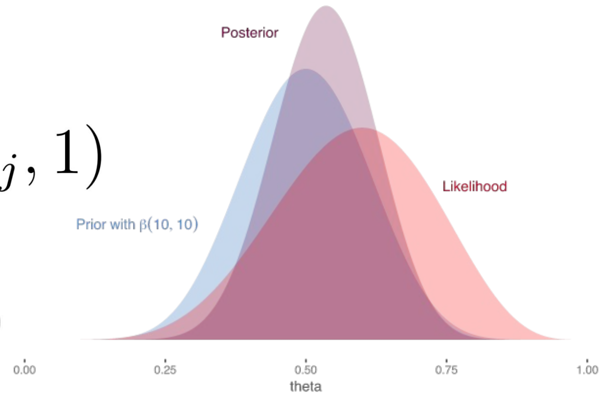
Bayesian linear regression

Likelihood:

$$p(D \mid W) = \prod_j \mathcal{N}(y_j \mid W x_j, 1)$$

Prior:

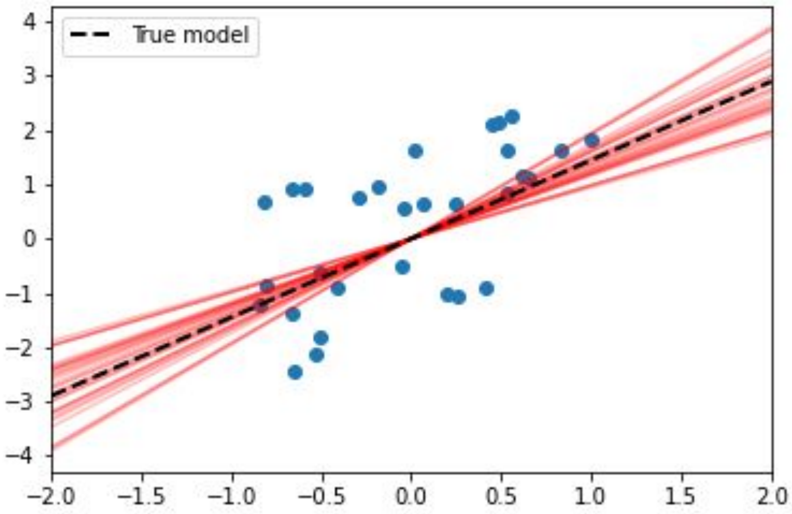
$$p(W) = \mathcal{N}(W \mid 0, \lambda I)$$

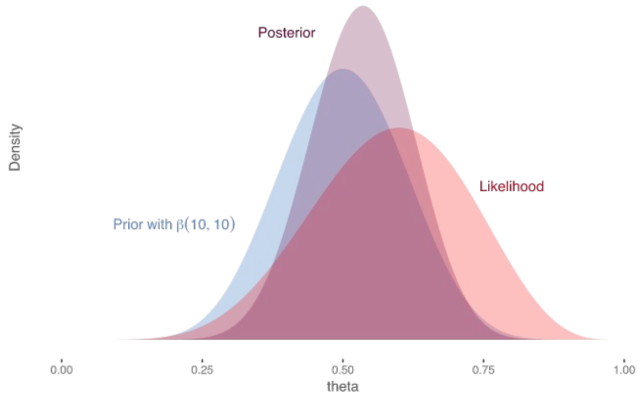


Posterior:

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

Bayesian linear regression



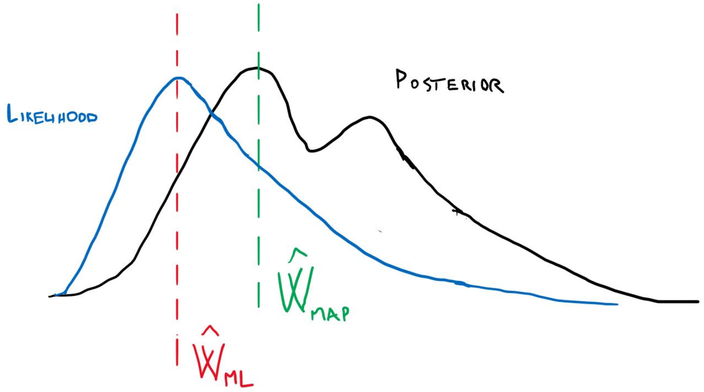


$$W_{\text{sampled}} \sim p(W \mid D)$$

Maximum-a-posteriori (MAP)

$$\hat{W}_{\text{MAP}} = \text{argmax}_w p(W \mid D) = \text{argmin}_W \mathcal{R}(W)$$

$$\mathcal{R}(W) = -\log P(D \mid W)p(W)$$



Maximum-a-posteriori and regularization

Regularized loss:

$$\mathcal{R}(W) = \sum_j \log p(y_j \mid Wx_j, 1)) + \log p(W \mid 0, \lambda I)$$

$$= -\frac{1}{2}\sum_j (y_j - Wx_j)^2 + \frac{1}{2\lambda}\sum_{k,l} W_{kl}^2 + c$$

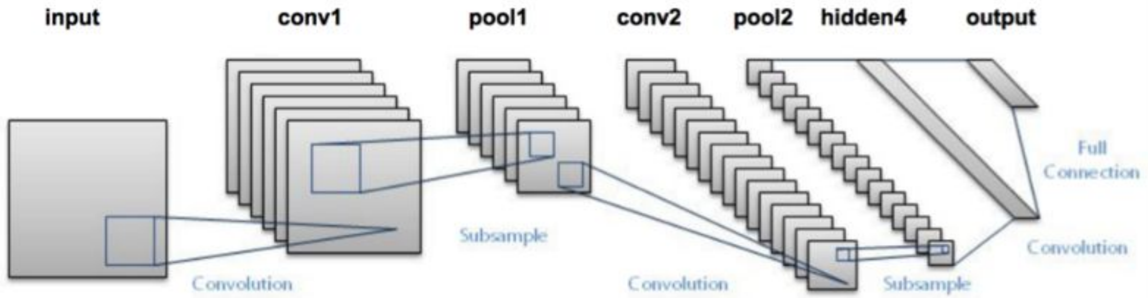
Regularized update:

$$\nabla \mathcal{R} = \nabla \mathcal{L}(W) - \lambda^{-1}W$$

Weight decay term

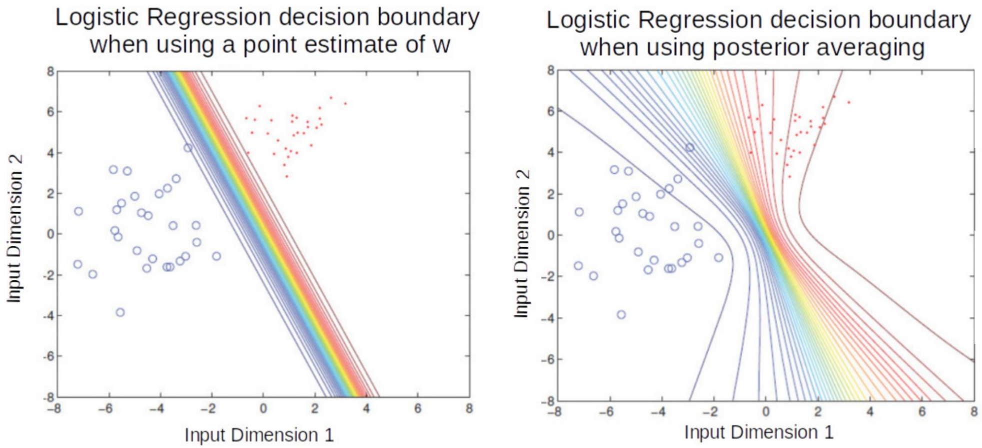
Bayesian deep networks

Place a prior on the weights of any regular deep net and estimate the posterior over the weights P(W | D). The MAP estimate is just regularized SGD.



Source:
https://davidstutz.de/a-short-introduction-to-bayesian-neural-networks/

Bayesian deep learning example: Logistic regression



Bayesian

$$p(y \mid x, D) = \int p(y \mid x, W) p(W \mid D) \mathrm{d}W$$

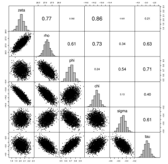
Point-estimate (SGD)

$$p(y \mid x, D) \approx p(y \mid x, \hat{W} = \mathrm{argmax}_W \mathcal{L}(W, D))$$

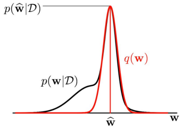
Source: https://www.cse.iitk.ac.in/users/piyush/courses/tpmi_winter19/tpmi_w19_lec7_slides_print.pdf

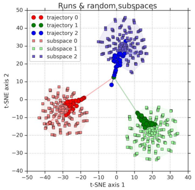
Bayesian deep networks

The posterior is intractable! However we can attempt to approximate it using several techniques:



- Gradient-based Markov Chain Monte Carlo
- Variational inference (to be discussed in later lectures)
- Ensembling (to be discussed in later lectures)





Source:
https://davidstutz.de/a-short-introduction-to-bayesian-neural-networks/

Likelihood

Example: Binary classification with sigmoid cross entropy loss

Generic variable for all the learnable parameters

$$p(y_j \mid W, x_j) = y_j \log\left(f(x_j; W)\right) + (1 - y_j)\log\left(1 - f(x_j; W)\right)$$

Forward pass of eep architecture with sigmoid output node

Prior

It is commod to assign a univariate centered Gaussian prior to each parameter in the network

$$p(W) = \prod_k \mathcal{N}\left(W_k \mid 0, \sigma^2_{\text{prior}}\right)$$

Do why have a reason to make this choice? Not really

Maximum-a-posteriori SGD for Bayesian deep networks

1-datapoint loss: $\mathcal{R}_j(W) = N \log p(y_j \mid W, x_j) - \log p(W)$

Dataset size (correct for sub-sampling)

Mini-batch SGD: $W_{n+1} = W_n - \eta \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n)$

Interlude: Markov Chain Monte Carlo sampling

Proposal distribution: $$p(W_{n+1} \mid W_n)$$

Correct for sampling bias

Acceptance rate: $$a_{n+1} = \max\left(\frac{\exp\left(-\mathcal{R}(W_{n+1})\right)}{\exp\left(-\mathcal{R}(W_n)\right)} \cdot \frac{p(W_n \mid W_{n+1})}{p(W_{n+1} \mid W_n)}, 1\right)$$

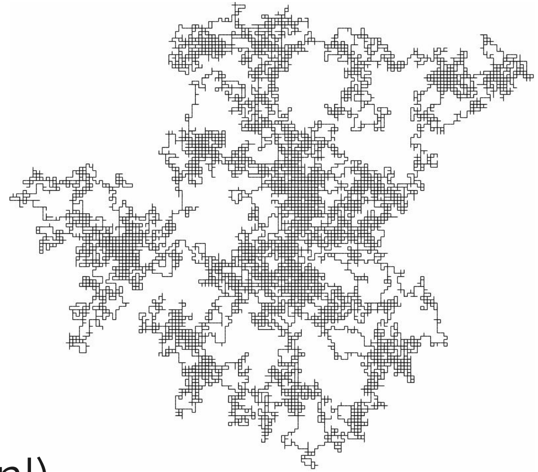
Decrease regularized loss

Interlude: Markov Chain Monte Carlo sampling

If you accept/reject samples with probability given by the acceptance ratio, the true posterior distribution is guaranteed to be the stationary sampling distribution of the Markov Chain!

Namely:

$$W_N \sim p(W \mid D)$$



When N is large enough.
(What does “large enough” mean? Very hard question!)

Stochastic Gradient Langevin Monte Carlo:

SGD ≈ MCMC

Regularized SGD gradient update

$$p(W_{n+1} \mid W_n) = \mathcal{N}\left(W_n - \eta \underbrace{\sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n)}, I\sigma^2\right)$$

Source:
https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf

Stochastic Gradient Langevin Monte Carlo:

SGD ≈ MCMC

Randomness from minibatch sampling

$$W_{n+1} = W_n - \eta \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n) + \sigma \epsilon_n$$

Gaussian random perturbation (only difference!)

Source:
https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf

Stochastic Langevin Monte Carlo:

SGD ≈ MCMC

Gaussian random perturbation: Only difference?

Well almost, we need to compute the acceptance rate and decide whether to accept or reject the sample.

Source:

https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf

Stochastic Langevin Monte Carlo

Unfortunately this is unpractical in a deep learning setting since the acceptance ration has to be computed over the whole dataset!

$$\log a_{n+1} = - \sum_{j \in \text{dataset}} \left(\mathcal{R}_j(W_{n+1}) - \mathcal{R}_j(W_n) \right) \; + \; \text{bias correction term}$$

Approximate Stochastic Langevin Monte Carlo

$$\log a_{n+1} \approx - \sum_{j \in \text{minibatch}} \left(\mathcal{R}_j(W_{n+1}) - \mathcal{R}_j(W_n) \right) \;+\; \text{bias correction term}$$

Practical but not guaranteed to converge to the true posterior!

Stochastic Langevin Dynamics

We can ignore the acceptance step if we let the learning rate get smaller and smaller!

$$a_n \to 1, \quad \text{when} \quad \eta_n \to 0$$

This is because the acceptance rate tends to one (check!)

Slow enough to explore

$$\sum_{n=1}^{\infty} \eta_t = \infty$$

$$\sum_{n=1}^{\infty} \eta_t^2 < \infty$$

Fast enough to make the acceptance step negligible

Source:
https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf

Stochastic Langevin Dynamics

$$W_{n+1} = W_n - \eta_n \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n) + \sqrt{\eta_t}\epsilon_n$$

$$\epsilon_n \sim \mathcal{N}(0, 1)$$

$$\sum_{n=1}^{\infty} \eta_t = \infty$$

$$\sum_{n=1}^{\infty} \eta_t^2 < \infty$$

Source:
https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf

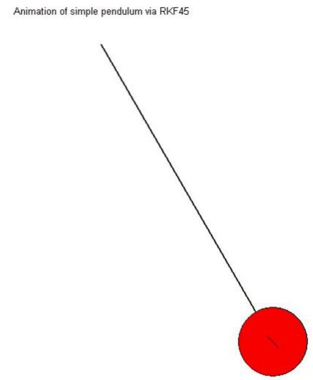
Part II: Momentum SGD and Hamiltonian MCMC

Dr. Luca Ambrogioni

Momentum in elementary physics

$$m\frac{d^2x}{dt^2} = F$$

Equation of motion



Momentum in elementary physics

Mass (how much the system "wants" to keep moving in the same direction)

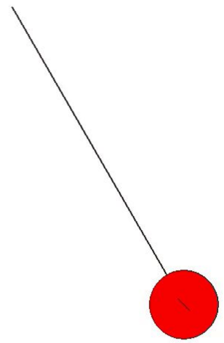
External force that changes motion

$$p = m\frac{dx}{dt} \qquad \frac{dp}{dt} = F$$

Definition of momentum

Equation of motion

Animation of simple pendulum via RKF45



From physics to deep learning

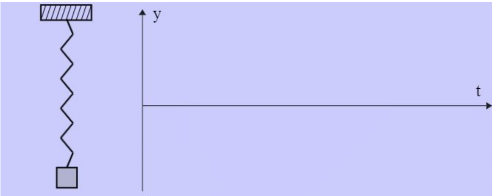
In deep learning, we want a force that pushes towards decreasing lower loss values. If we use mini-batches, this force is stochastic.

$$F(W) = -\eta \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W)$$

Random forces due to sub-sampling

From physics to deep learning

Equation “of motion” of a leaning system



Momentum damping

Momentum of the network parameters

$$\frac{dp}{dt} = -\eta \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W) - \frac{\beta}{m} p$$

“Learning force”

From physics to deep learning

Equation "of motion" of a leaning system

$$\frac{dp}{dt} = -\eta \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W) - \frac{\beta}{m} p$$

$$m \frac{dW}{dt} = p$$

Momentum gradient descent

Just the good old Euler discretization:

$$p_{n+1} = \left(1 - \mathrm{d}t\frac{\beta}{m}\right) p_n - \mathrm{d}t \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n)$$

$$W_{n+1} = W_n + \frac{\mathrm{d}t}{m} p_{n+1}$$

Visualization source:
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Momentum gradient descent

Let's clean up and re-define the free parameters

$$p_{n+1} = (1-b)p_n - b \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n)$$

$$W_{n+1} = W_n + \eta p_{n+1}$$

Weighted average between previous value and new gradient direction

Redefined learning rate

Adding Momentum to Monte Carlo: First attempt

Proposal distribution:

Gaussian random force

$$p_{n+1} = (1-b)p_n - b \sum_{j \in \text{minibatch}} \nabla \mathcal{R}_j(W_n) + \sigma \epsilon_n$$

$$W_{n+1} = W_n + \eta p_{n+1}$$

Adding Momentum to Monte Carlo: Problems

Problems:

- The resulting dynamics is not Markov since the momentum introduces memory
- The acceptance rate is burdensome to compute since the transition distribution is not reversible ($p(x',p'|x,p) \neq p(x,p|x',p')$)

Adding Momentum to Monte Carlo: Solutions

Solutions:

- Include the momentum as auxiliary variable with prior: $\mathcal{N}(0, I)$
- Make the dynamic reversible (We will talk about this later)

Momentum as auxiliary variables: An augmented probabilistic model

Likelihood (does not depend on the momentum)

$$P(W,p) = P(W|D)P(W)\mathcal{N}(p;0,1)$$

Prior

Auxiliary Prior

Momentum as auxiliary variables: An augmented probabilistic model

- The auxiliary variables do not change the original Bayesian learning problem since the likelihood does not depend on them!
- However, they turn the momentum dynamics into a Markov process so that we can do MCMC with it!

Momentum as auxiliary variables: The Hamiltonian loss function

This new probabilistic model gives us a new augmented loss function

$$\mathcal{H}(W, p) = \mathcal{R}(W) + \frac{1}{2m} p \cdot p$$

“Potential Energy” “Kinetic Energy”

Momentum as auxiliary variables: The Hamiltonian loss function

We can define a reversible dynamics on the augmented space by using one of the most beautiful equations of physics: the Hamilton equations of motion!

$$\frac{dp}{dt} = -\nabla_x \mathcal{H}(W, p)$$

$$\frac{dW}{dt} = \nabla_p \mathcal{H}(W, p)$$

Momentum as auxiliary variables: The Hamiltonian loss function

Just momentum gradient descent without damping!

$$\frac{dp}{dt} = -\nabla_x \mathcal{R}(W)$$
$$\frac{dW}{dt} = \frac{1}{m} p$$

Momentum as auxiliary variables: The Hamiltonian loss function

We need to discretize the dynamics without breaking the reversibility property. A solution is the leapfrog integrator:

$$p_{n+1/2} = p_n - \frac{\eta}{2}\nabla\mathcal{R}(W_n)$$

$$W_{n+1} = W_n + \eta p_{n+1/2}$$

$$p_{n+1} = p_{n+1/2} - \frac{\eta}{2}\nabla\mathcal{R}(W_{n+1})$$

Acceptance ratio and conservation of energy

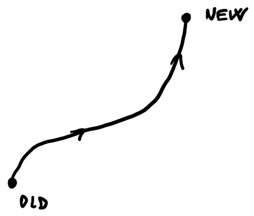
Since the dynamics is reversible, the acceptance ratio is given by the difference between the (augmented) loss functions:

$$\log a_n = -\mathcal{H}(W_{\mathrm{new}}, p_{\mathrm{new}}) + \mathcal{H}(W_{\mathrm{old}}, p_{\mathrm{old}})$$

Acceptance ratio and conservation of energy

If the new sample was obtained using the true (not discretized) Hamiltonian dynamics, we would always accept with probability one since H is the energy of the system and is therefore conserved!

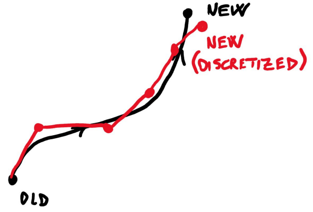
$$-\mathcal{H}(W_{\text{new}}, p_{\text{new}}) + \mathcal{H}(W_{\text{old}}, p_{\text{old}}) = 0$$



Acceptance ratio and conservation of energy

However, the discretization introduces a violation of energy conservation, leading to the need to reject some samples

$$-\mathcal{H}(W_{\text{new}}, p_{\text{new}}) + \mathcal{H}(W_{\text{old}}, p_{\text{old}}) \neq 0$$



Hamiltonian Monte Carlo Sampling

A three steps procedure:

- Sample the momentum from its auxiliary prior distribution
- Run the discretized Hamiltonian dynamics with L steps of leapfrog numerical integration
- Accept/reject the resulting sample with probability given by the acceptance ratio

Hamiltonian Monte Carlo Sampling

Open source interactive sampler: https://github.com/chi-feng/mcmc-demo

Bayesian neural networks in modern deep learning

What Are Bayesian Neural Network Posteriors Really Like?

Pavel Izmailov
New York University

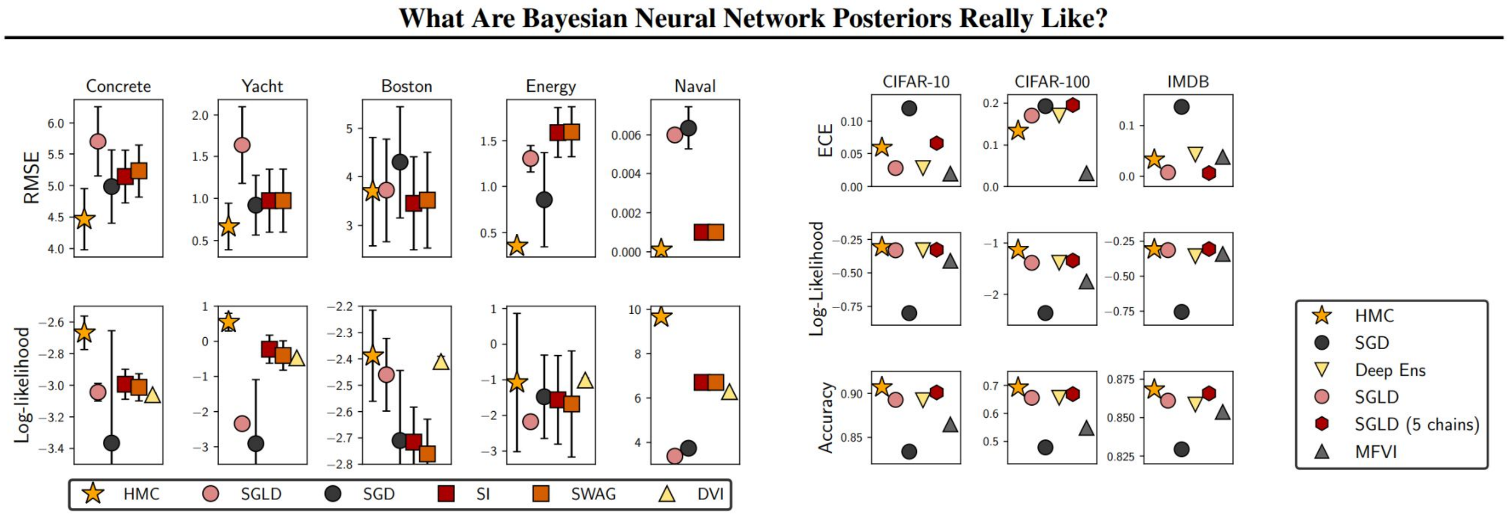
Sharad Vikram
Google Research

Matthew D. Hoffman
Google Research

Andrew Gordon Wilson
New York University

Papers: https://arxiv.org/pdf/2104.14421.pdf

Bayesian neural networks in modern deep learning



Paper: https://arxiv.org/pdf/2104.14421.pdf

Open problems

- In deep learning, the true gradient is often prohibitively expensive to compute as it involves a pass through all samples in the dataset. This can be solved using stochastic gradients (SGHMC).

- Unfortunately, the resulting algorithm is biased unless we perform Metropolis-Hasting (M-H) correction (accept/reject sampling step) which requires the often prohibitively expensive acceptance ratio.

Stochastic gradient Hamiltonian MC paper: http://proceedings.mlr.press/v32/cheni14.html
Problems with sub-sampling: https://arxiv.org/pdf/1502.01510.pdf

Thank you.