MCMC and Bayesian neural networks

Arto Klami
University of Helsinki, Finland
arto.klami@helsinki.fi

Nordic Probabilistic AI School, June 2025

Introduction

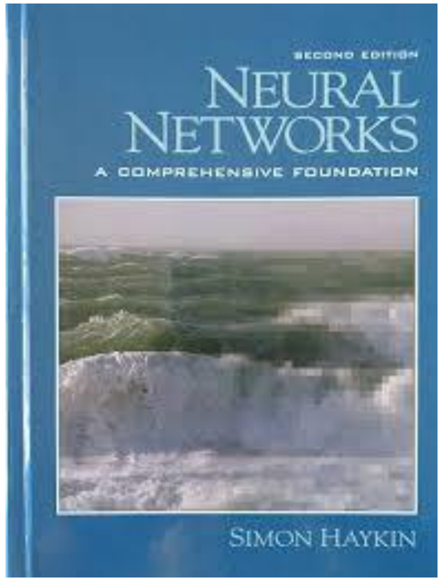
In 90s, all the cool kids in Finland were doing graphics, music and demo competitions

- This is why we are a superpower of games (Angry birds, Clash of Clans, Alan Wake, Max Payne, ...)

The less cool did AI

- This is why we now have Finnish Center for Artificial Intelligence, the second ELLIS Institute, etc.

Assembly demo event in 2002



Regular reading for highschool kids around 1995

Introduction

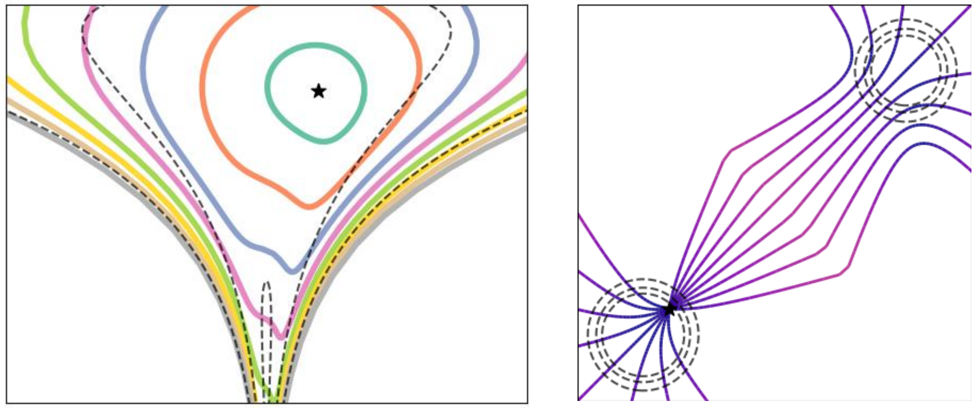
I’m more of the latter kind

- Part of Helsinki Probabilistic Machine Learning Lab at Univ. of Helsinki

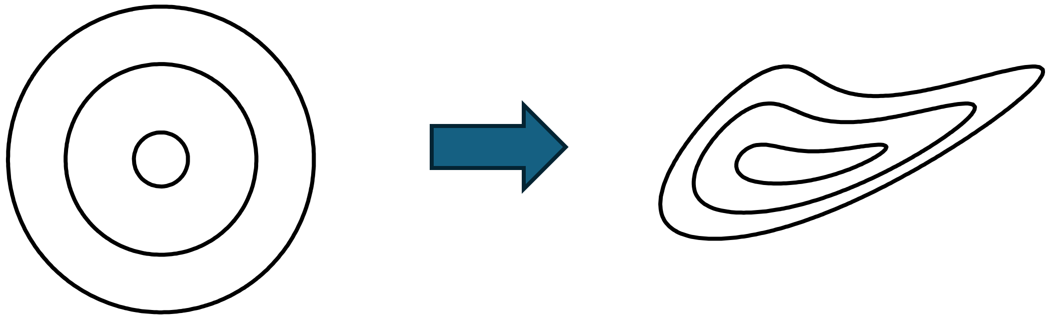
Some 20 years of research on

- Bayesian models for data integration
- Decision-making
- Inference
    - Variational and Laplace approximation
    - MCMC
    - Differential geometry in inference
- Priors and knowledge elicitation

Talked about variational inference and priors in previous ProbAI



Improving inference by adapting to local geometry of the posterior (Williams et al. 2025)



Arbitrarily flexible priors in statistical modelling (Mikkola et al. 2024)

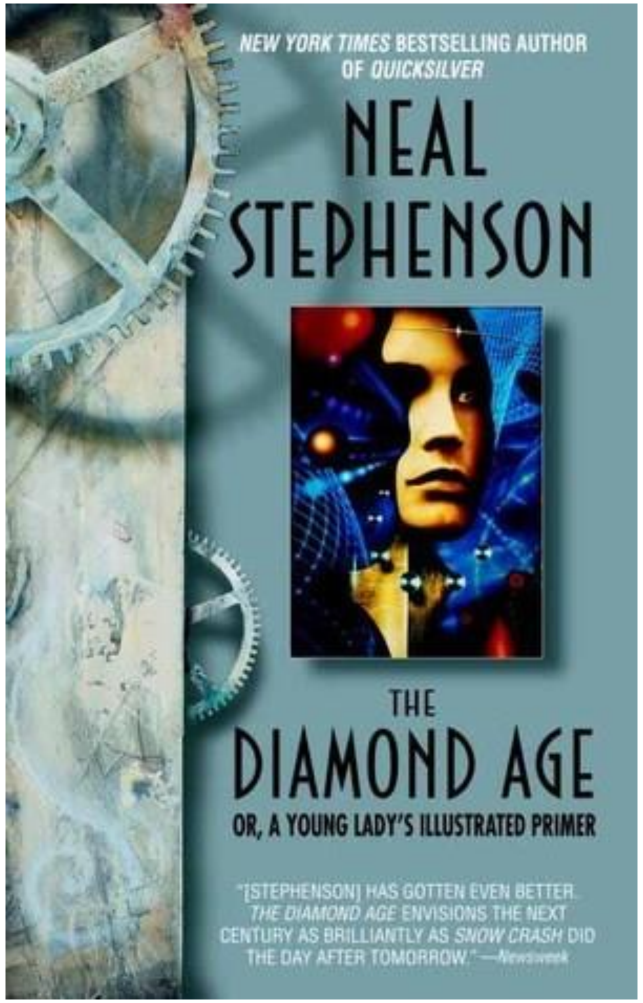
Introduction

25 years ago, neural networks were hard but Bayesian statistics easy

Backpropagation in C++
vs
random-walk Metropolis-Hastings

Now everyone can do neural networks, but statistics is perceived difficult

Keras and companions
vs
'I need to actually learn math'


NEW YORK TIMES BESTSELLING AUTHOR OF QUICKSILVER
NEAL STEPHENSON
THE DIAMOND AGE
OR, A YOUNG LADY'S ILLUSTRATED PRIMER
"[STEPHENSON] HAS GOTTEN EVEN BETTER. THE DIAMOND AGE ENVISIONS THE NEXT CENTURY AS BRILLIANTLY AS SNOW CRASH DID THE DAY AFTER TOMORROW." —Newsweek

The Bayesian Age
Or, a Young AI researcher's illustrated primer on MCMC

Objectives

To convince you that **Bayesian neural networks are easy** and explain some basic principles behind sampling-based inference

Mostly about the easiest possible algorithms

- Focus on connecting samplers to optimization
- Implicit assumption: You are better at optimization

We talk about NNs, but not DL

- Small NNs, very little on scalability

Contents

Background: Neural networks

A neural network defined by a combination of

- Architecture
  - Fully-connected, CNN, transformer, …
- Objective
  - Mean-square error, cross-entropy, ..
- Learning algorithm
  - Stochastic gradient descent, Adam, …

Background: Neural networks

Absolutely **nothing** related to architecture today
- Everything in principle applies to any architecture
  - But: Various heuristics and tricks, like batch normalization, may be difficult to handle
- Small fully-connected networks in examples

Learning algorithm
- We start with SGD and proceed to MCMC
- Fundamental difference in the task

Search for best possible parameters

vs

Represent the posterior distribution of all possible parameters

Background: Neural networks

For simplicity, we consider only supervised learning tasks with i.i.d. samples, not modelling the inputs as random variables

- NN outputs the parameters of the predictive distribution of a known form

Objective is always $\mathcal{L}(\boldsymbol{\theta}) = \log \left[p(\boldsymbol{\theta}) \prod_n^N p(\mathbf{y}_n, \boldsymbol{\theta} | \mathbf{x}_n) \right] = \log p(\boldsymbol{\theta}) + \sum_n^N \log p(\mathbf{y}_n, \boldsymbol{\theta} | \mathbf{x}_n)$

...of for easier mini-batching $\mathcal{L}(\boldsymbol{\theta}) = N \left[\frac{1}{N} \log p(\boldsymbol{\theta}) + \frac{1}{N} \sum_n^N \log p(\mathbf{y}_n, \boldsymbol{\theta} | \mathbf{x}_n) \right]$

Don’t worry:

- All losses are like this, we just write them more carefully now
- Very little focus on the prior term: It is not important (but the 1/N scaling is!)

Neural networks: Implementation

Define the structure

```
class network(nn.Module):
 def __init__(self, D_i, D_o, K):
    super().__init__()
    self.fc = nn.Linear(D_i, K)
    self.fc2 = nn.Linear(K, K)
    self.out = nn.Linear(K, D_o)

 def forward(self, x):
    x = self.fc(x)
    x = nn.functional.tanh(x)
    x = self.fc2(x)
    x = nn.functional.tanh(x)
    x = self.out(x)
    return x
```

Define the computation

Define the objective

```
class loglikelihoodGaussian(nn.Module):
  def __init__(self, variance):
     super(NLLLossNormal, self).__init__()
     self.variance = torch.tensor(variance)

  def forward(self, predictions, targets):
     # Negative log-likelihood over samples
     ll = -0.5*(torch.log(2.*torch.pi * self.variance) +
            (predictions - targets) ** 2 / self.variance)
     return torch.sum(ll)

objective = loglikelihoodGaussian(pred_variance) + log_prior
```

Neural networks: Optimization

Stochastic descent (SGD)

- Initialize somehow
- Loop over $$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \epsilon \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$

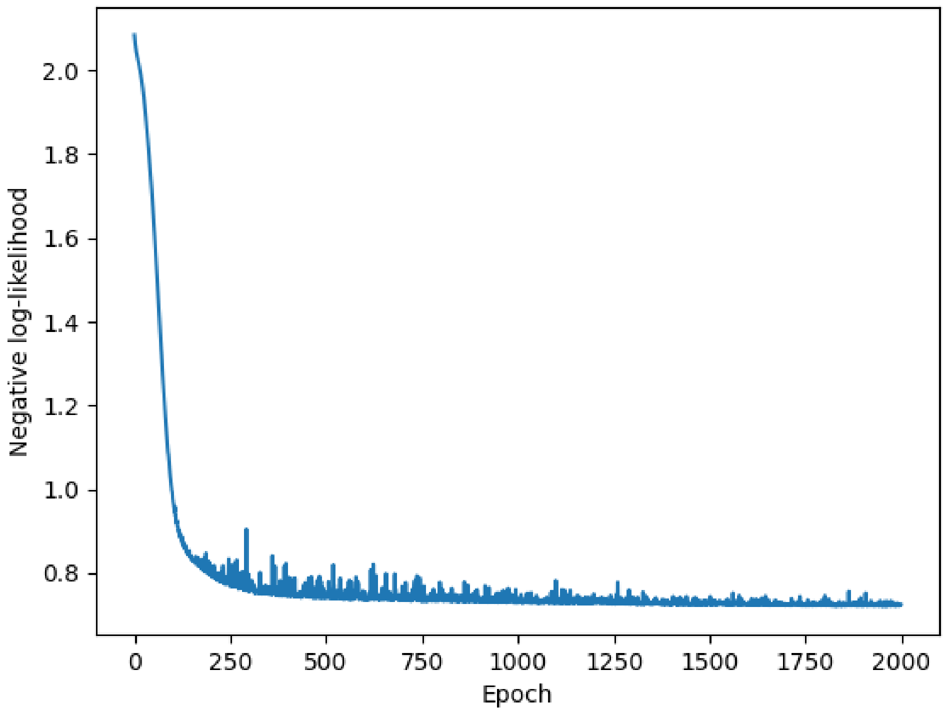
  where the expected gradient is estimated using a mini-batch

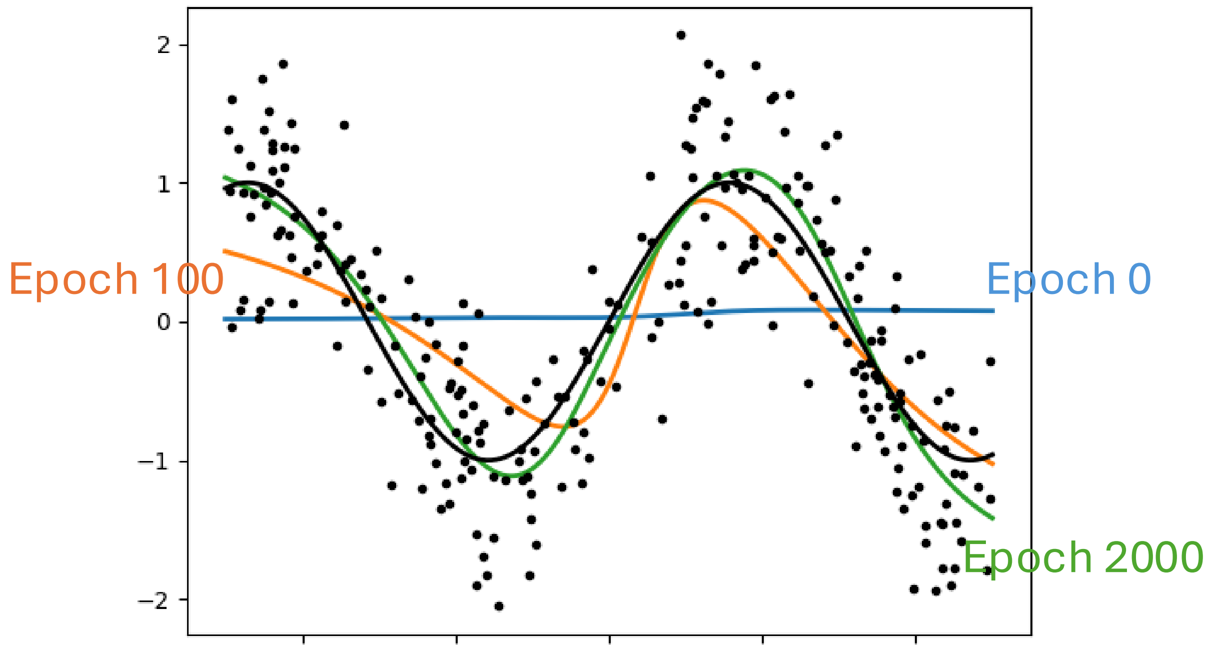
$$\nabla \mathcal{L}(\boldsymbol{\theta}) \approx \frac{1}{N} \nabla \log p(\boldsymbol{\theta}) + \left[\frac{1}{S} \sum_{s \in \mathcal{S}} \nabla \log p(\mathbf{y}_s, \boldsymbol{\theta} | \mathbf{x}_s) \right]$$

- Make the step length ε smaller
- Store all iterations for calculation of validation losses $P = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T\}$

Adam and others are minor variants, replacing the scalar step-length with a vector that is adapted during the iterations

Neural networks: Optimization





Background: Bayesian inference

Bayesian modelling split into two parts

1. Estimate the posterior using the Bayes rule

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}, \boldsymbol{\theta})}{p(\mathcal{D})}$$

2. Average predictions over the posterior

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

Background: Bayesian inference for NNs

The posterior itself is often interesting in classical statistical modelling

…but for neural networks very much not

- No interpretation for individual parameters
- Completely non-identifiable (e.g. permutations)
- No tools for interpretation: How to plot a billion-dimensional posterior?

Hence: **We (almost) only care about the predictions**!

Background: Posterior predictive distribution

The posterior predictive is difficult even if we knew the posterior

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

Monte Carlo integration as the go-to technique

$$p(\mathbf{y}|\mathcal{D}, \mathbf{x}) \approx \frac{1}{M} \sum_{m=1}^{M} p(\mathbf{y}|\boldsymbol{\theta}_m, \mathbf{x}) \qquad \boldsymbol{\theta}_m \sim p(\boldsymbol{\theta}|\mathcal{D})$$

We need only samples from the posterior

- VI used some q(θ) that was optimized to look like the posterior
- MCMC algorithms provide such samples directly

Background: Random-walk MCMC

When I said Bayesian inference was easy, I meant random-walk Metropolis Hastings

- Propose $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ e.g. $\boldsymbol{\theta}' \sim \boldsymbol{\theta} + \mathcal{N}(0, \sigma)$
- Check whether to accept the proposal

$$p(\text{accept}) = \frac{p(\mathcal{D}, \boldsymbol{\theta}')}{p(\mathcal{D}, \boldsymbol{\theta})} \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})} \Rightarrow \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}' \text{ or } \boldsymbol{\theta}$$

- Store all samples after converging to the target $P = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T\}$

Works well in low dimensions, but random proposals are **ridiculously** inefficient in higher dimesnions

Backround: Gradients in MCMC

We don’t optimize functions with random walk, but use gradients instead: They are massively helpful and (today) easy to compute

For BNNs, any sampling algorithm also **needs** to use gradients

- Most of MCMC today uses gradients, with Hamiltonian Monte Carlo (e.g. NUTS in Stan) as the main example

So, MCMC stopped being easy.

Or, it was only easy for small problems to begin with!

The predictive distribution

$$p(\mathbf{y}|\mathcal{D}, \mathbf{x}) \approx \frac{1}{M}\sum_{m=1}^{M} p(\mathbf{y}|\boldsymbol{\theta}_m, \mathbf{x})$$

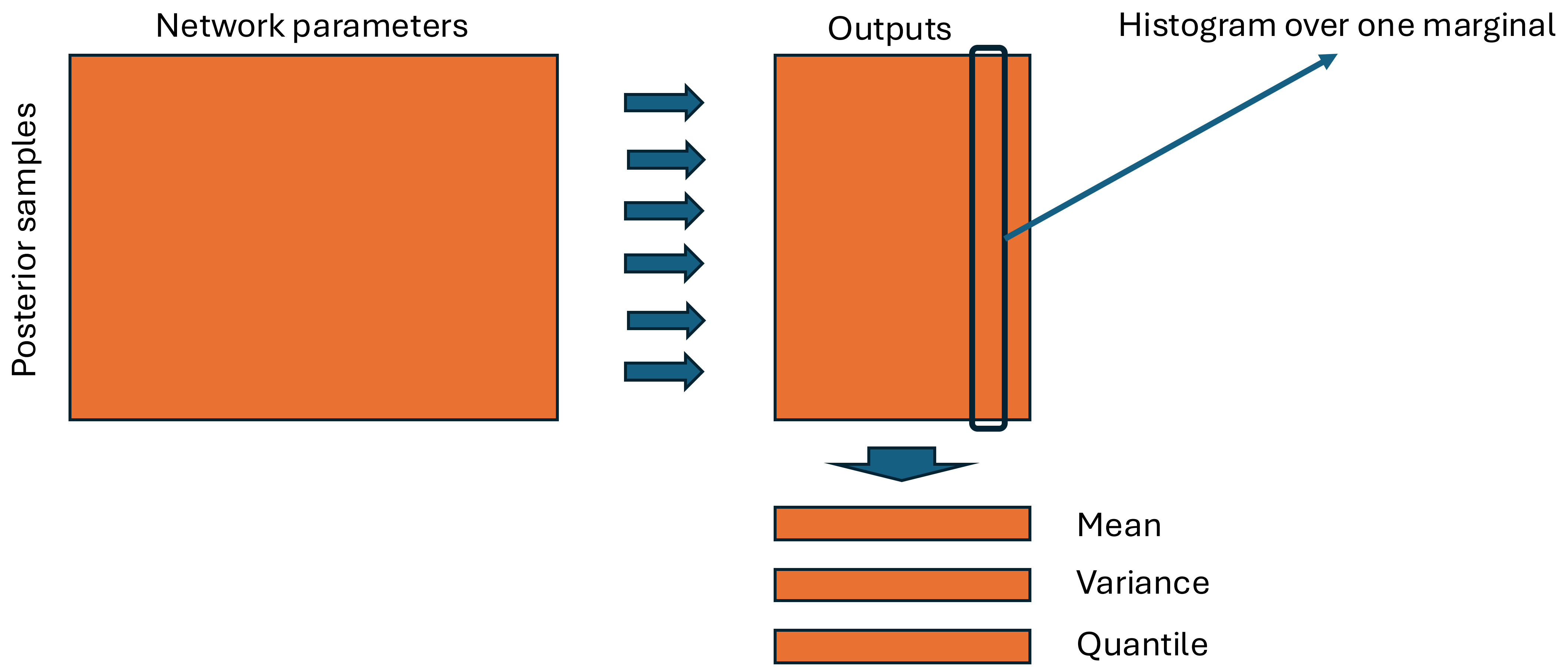
Assume we already have $\boldsymbol{\theta}_m \sim p(\boldsymbol{\theta}|\mathcal{D})$

As a placeholder, let's use the samples stored during optimization

- Definitely **not** samples from the posterior, but helps understanding how the predictive distribution is formed

1. Every test sample passed through the network M times
2. Both computational and memory load goes up by factor of M
3. We get M different predictions

The predictive distribution (for one input)



$$p(\mathbf{y}|\mathcal{D}, \mathbf{x}) \approx \frac{1}{M} \sum_{m=1}^{M} p(\mathbf{y}|\boldsymbol{\theta}_m, \mathbf{x})$$

Monte Carlo as ensemble

- We can think of this as an **ensemble** model, like e.g. random forest
- MC integration is an ensemble over parameter configurations, not models (for random forest we train several decision trees)
- Ensembles can be useful as such, but if the parameters are from the posterior we get the real posterior predictive distribution

From optimization to Bayesian NNs

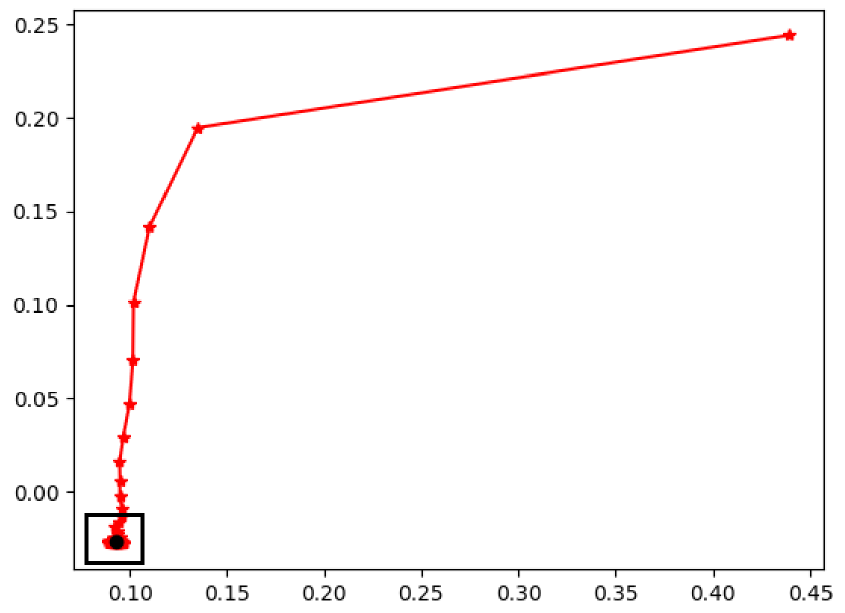
The only missing piece is the actual sampling algorithm: If you plug in the parameters collected during SGD then you get some ensemble, but not samples from the predictive distribution

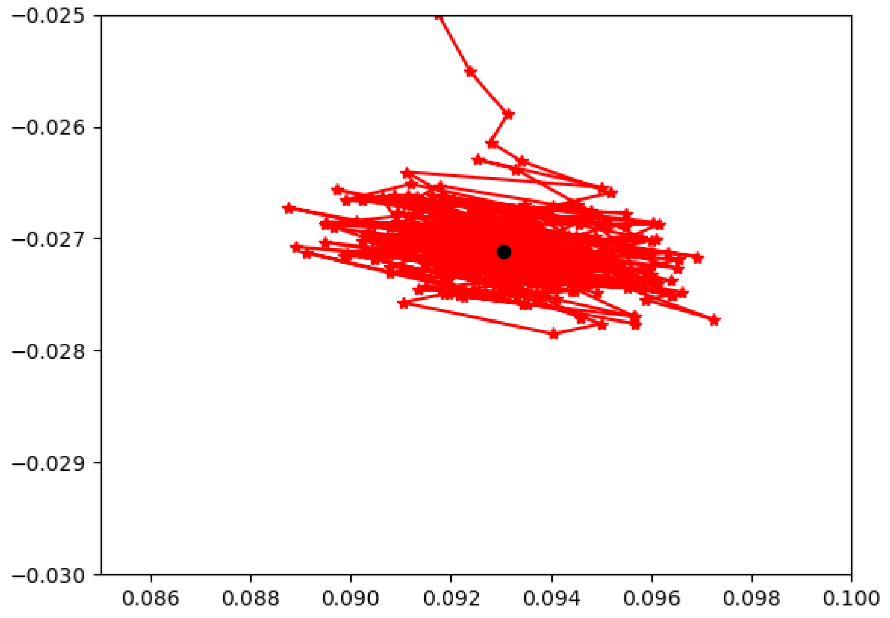
…except that you kind of do, if doing things right!

Next we discuss this relationship, as a stepping stone to concrete gradient-based MCMC algorithms for BNNs

How SGD works

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \epsilon \nabla \mathcal{L}(\boldsymbol{\theta}_t)$$





After reaching the optimum, **SGD fluctuates around it**

Caused by the gradient noise, and prevented in optimization by reducing the step size gradually

SGD samples from something

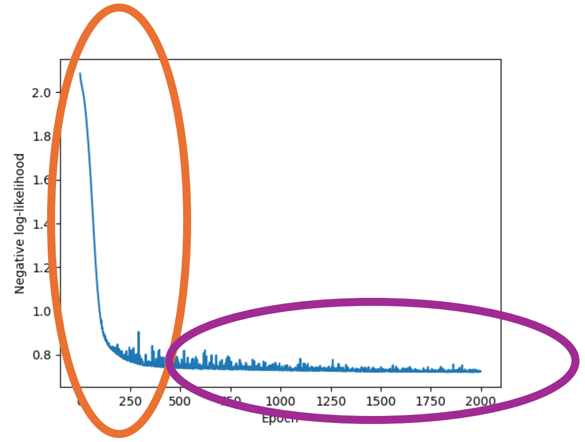
Let's split SGD into two phases:

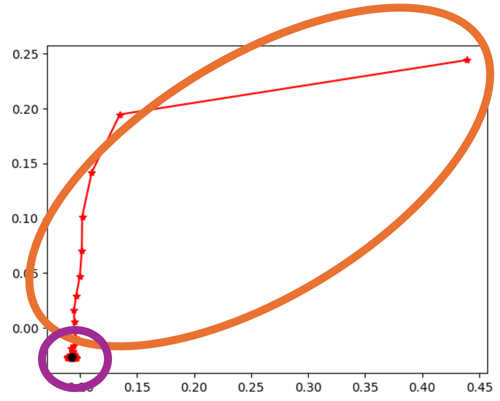
- Find the optimum
- Move around the optimum

During the latter phase:

- We move around the optimum in **some** way

Let's find out how exactly, following Mandt et al. (2017)





Gradient noise

For $g_N = \frac{1}{N}\sum_{n=1}^{N} \nabla \log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$ and $g_S = \frac{1}{S}\sum_{s=1}^{S} \nabla \log p(\mathbf{y}_s|\mathbf{x}_s, \boldsymbol{\theta})$

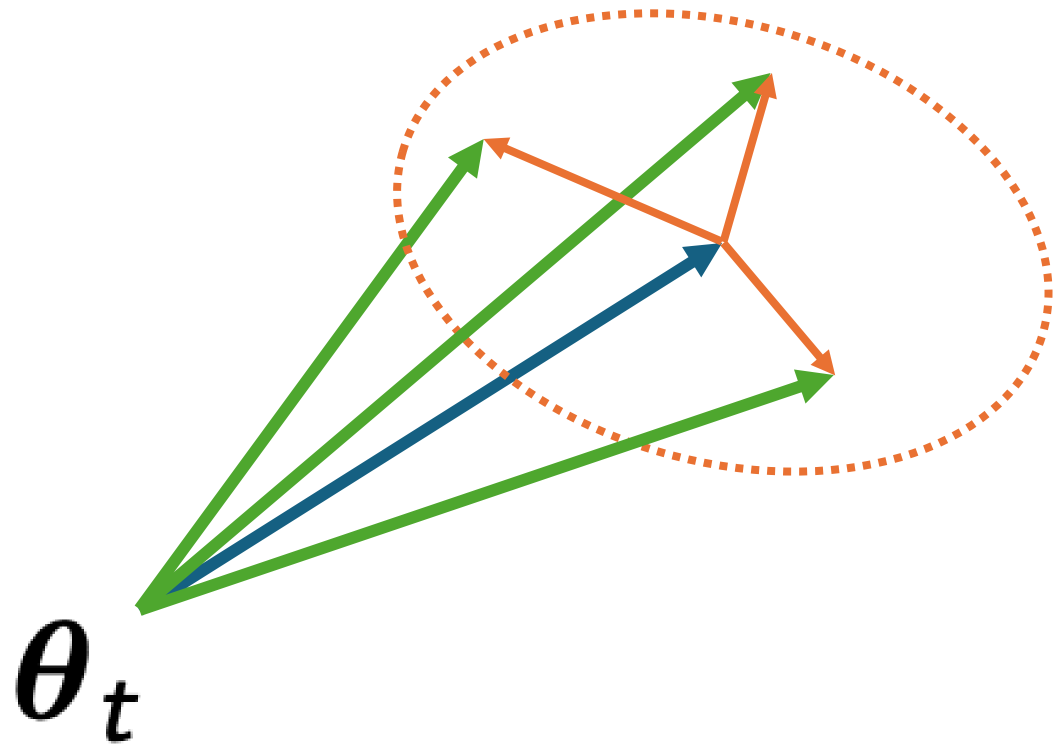
we have $g_S = g_N + ?$ but that is the error term?

1. **Central limit theorem**: We have S independent terms, so their sum tends to Gaussian
2. **Stochastic estimate is unbiased**: The mean is zero
3. **Variance scales inversely with sample size**

$$g_S = g_N + \frac{1}{\sqrt{S}}\gamma \qquad \gamma \sim \mathcal{N}(0, \mathbf{B}\mathbf{B}^T)$$

Covariance of per-sample noise (unknown)

Gradient noise



Full gradient

Mini-batch estimates

Gradient noise

SGD with noise

Re-write the SGD update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \epsilon g_S = \boldsymbol{\theta}_t - \epsilon \left[g_N + \gamma\right]$$

as $\Delta\boldsymbol{\theta}_t = -\epsilon \left[g_N + \frac{1}{\sqrt{S}} B \Delta W\right]$ with $\Delta W \sim \mathcal{N}(0, \mathbf{I})$

At the limit of small step size this corresponds to a differential equation with Brownian motion

$$\partial\boldsymbol{\theta}_t = -\epsilon \left[g_N + \frac{1}{\sqrt{S}} B \partial W(t)\right]$$

SGD with noise

The differential equation

$$\partial \boldsymbol{\theta}_t = -\epsilon \left[g_N + \frac{1}{\sqrt{S}} B \partial W(t) \right]$$

has a Gaussian stationary distribution with covariance

$$\Sigma = \frac{\epsilon}{2S} \mathbf{B}\mathbf{B}^T$$

Hence: SGD samples from some Gaussian with covariance proportional to the noise covariance of the gradient estimates

Local objective

Every log-posterior is **locally** quadratic

- Think of Taylor approximation for any continuous function

$$f(x) \approx f(x_0) + (x - x_0)f'(x) + \frac{1}{2}(x - x_0)^2 f''(x) + \ldots$$

...and hence well approximated by a Gaussian

- Laplace approximation $q(\boldsymbol{\theta}) = \mathcal{N}(\hat{\boldsymbol{\theta}}, \mathbf{H}^{-1})$

- The mean is the MAP estimate we find by optimization
- The covariance is inverse Hessian at that point $\mathbf{H}_{ij} = \left[\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta}_i \partial \boldsymbol{\theta}_j}\right]$

Matching the two

The target is locally Gaussian with inverse Hessian covariance and SGD samples from a Gaussian with covariance $\Sigma = \frac{\epsilon}{2S}\mathbf{B}\mathbf{B}^T$

The only moving piece here is the step-length, which we can select to make the two similar: $\epsilon = 2\frac{S}{N}\frac{D}{\mathrm{Tr}(\mathbf{B}\mathbf{B}^T)}$

- We don't know **B**, but we can estimate it
- We don't know **H**, but we do not need it (skipping the details here)

  QUESTION: Can you prove why this is the case? What kind of assumptions are needed and why don't we need to estimate **H**?

Estimating gradient noise

We need the expected gradient noise (for one sample)

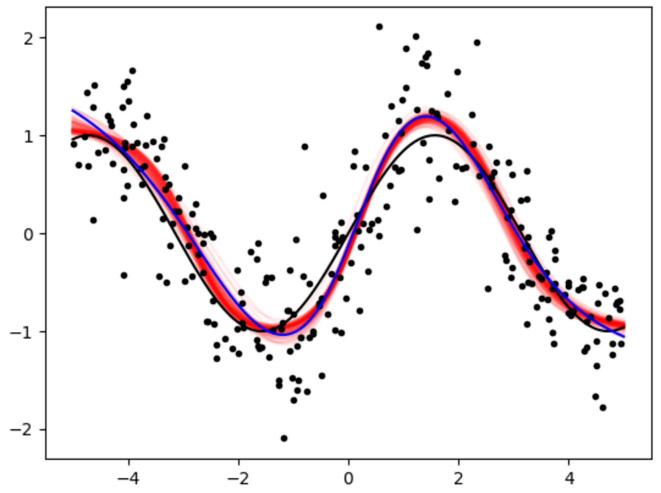
Could estimate directly by feeding individual data points, but this would be slow

In practice: Estimate covariance for the mini-batch gradient and re-scale back to get an estimate for per-sample covariance

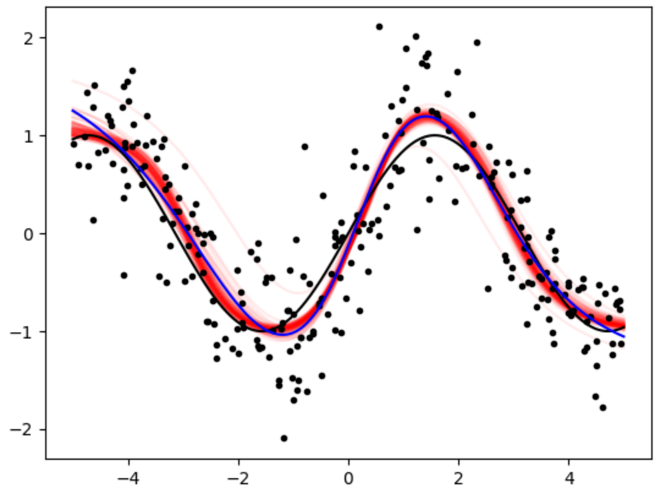
$$g_S = g_N + \frac{1}{\sqrt{S}}\gamma \qquad \gamma \sim \mathcal{N}(0, \mathbf{B}\mathbf{B}^T)$$

SGD in action (N=256)

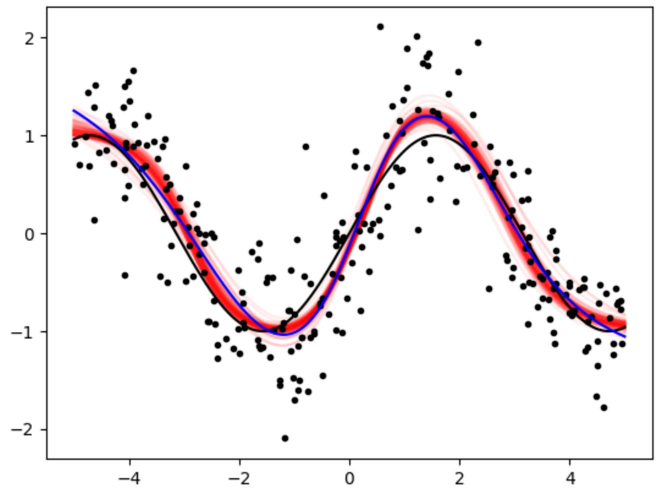
S = 8, lr = 0.007



S = 16, lr = 0.016

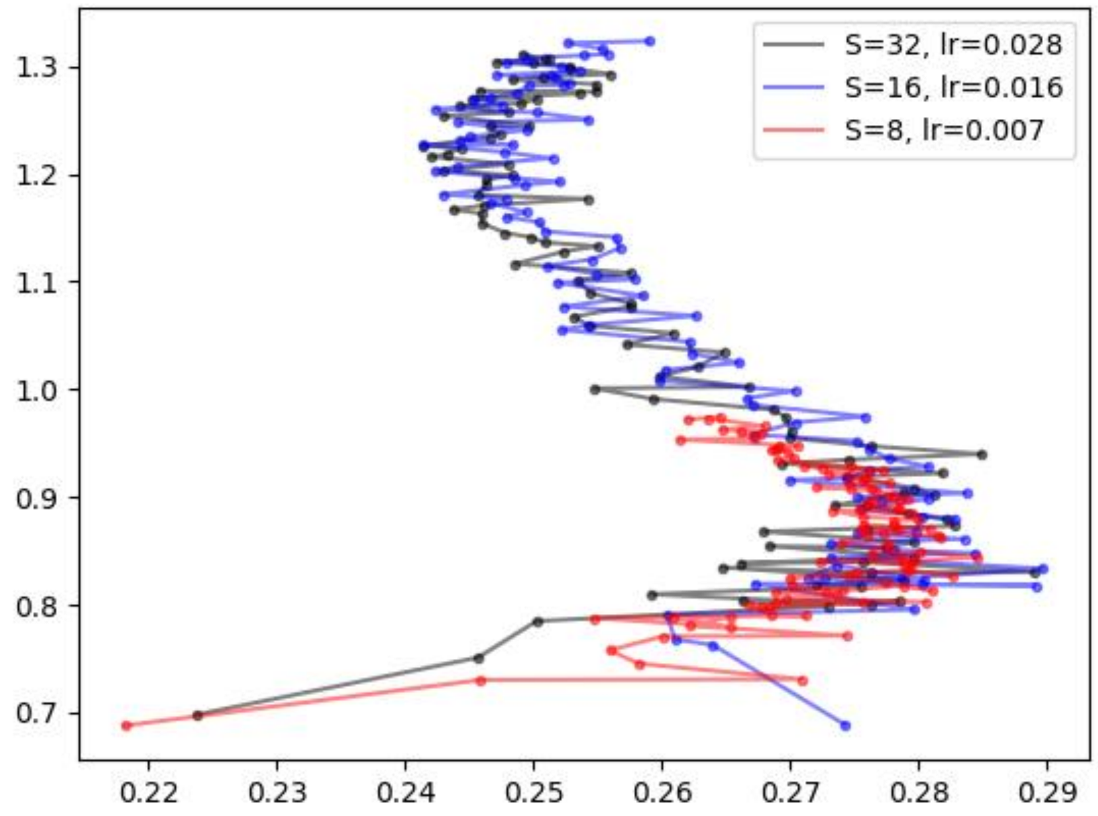


S = 32, lr = 0.028



For smaller mini batches we have smaller learning rate, due to more noise. The results are still highly similar, as they should!

SGD in action (N=256)



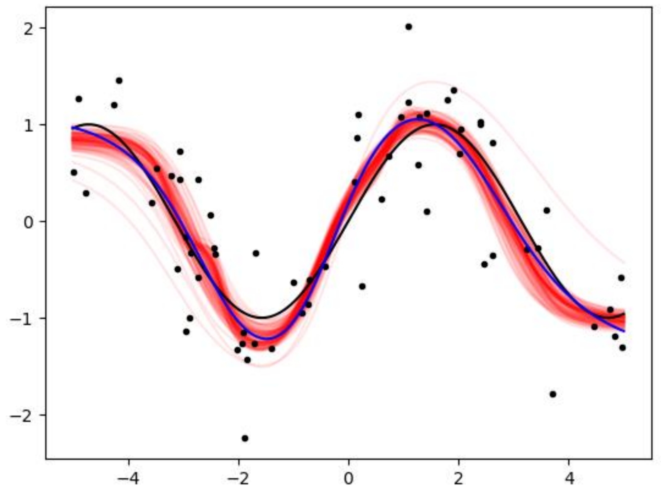
Appear to also sample from the same posterior

...that does not look very Gaussian

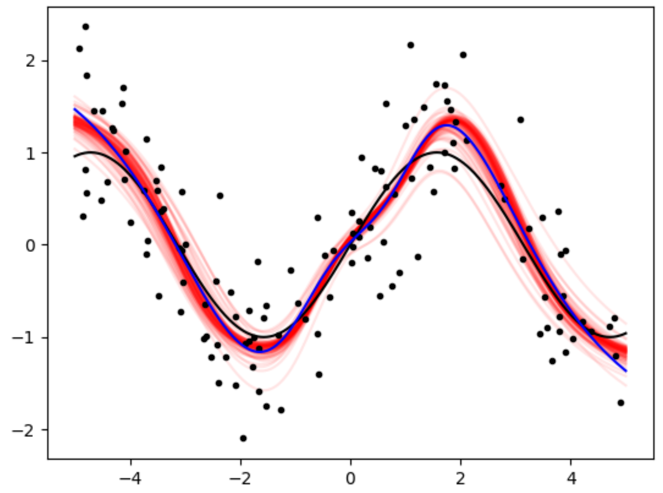
Note: Stops working when S approaches N, due to no noise and infinite step length

SGD in action (varying N)

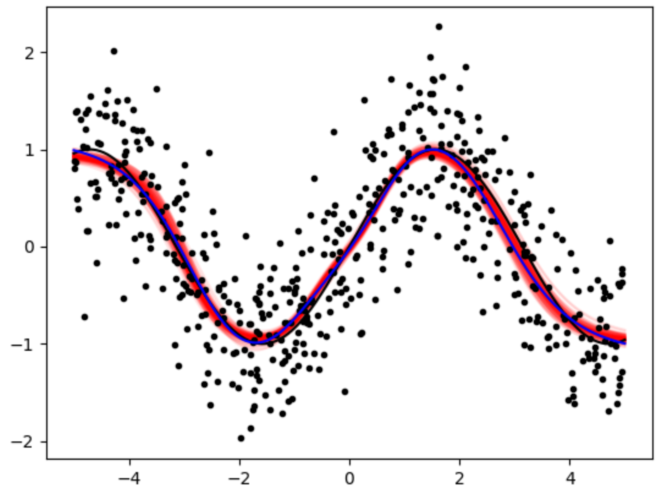
N=64



N=128



N=512



Works as expected, less uncertainty when N grows

...but we don’t (for now) really know whether this is correct!

Beyond scalar learning rates

Instead of a constant learning rate we can use a matrix preconditioner as in optimization

$$\Delta\boldsymbol{\theta}_t = -\mathbf{M}\left[g_N + \frac{1}{\sqrt{S}}B\Delta W\right] \quad \Longrightarrow \quad \mathbf{M}_{kk} = \frac{2S}{N\mathbf{B}\mathbf{B}^T_{kk}}$$

Full preconditioners don’t work for NNs, but diagonal ones do

- Effectively just a vector of learning rates instead of a scalar
- Most SGD variants (Adam etc) anyway do this

SGD as sampler

1. Find optimum, with any step length
2. Estimate gradient noise at the optimum by averaging over the minibatches
3. Solve for the optimal learning rate (scalar or vector)

$$\epsilon = 2\frac{S}{N}\frac{D}{\mathrm{Tr}(\mathbf{B}\mathbf{B}^T)} \qquad \mathbf{M}_{kk} = \frac{2S}{N\mathbf{B}\mathbf{B}^T_{kk}}$$

4. Keep running with this constant learning rate, storing the parameters every now and then
5. Monte Carlo integration over the samples for predictions

Hands-on exercise

Convert SGD into a sampler!

First implement the scalar learning rate case

Continue to diagonal preconditioner if time permits

Plotting code etc already given

Beyond SGD

SGD as a sampler is useful in understanding the basic idea, but not really something we would use in practice

Stochastic Gradient Langevin Dynamics (SGLD) (Welling and Teh, 2011) is perhaps the simplest actual sampling algorithm that follows essentially the same idea

Langevin dynamics

Langevin dynamics is a differential equation

$$\partial \boldsymbol{\theta}_t = -\frac{\epsilon}{2} g_N + \sqrt{\epsilon}\partial W(t)$$

that by construction has the posterior as its stationary distribution

It's discrete-time approximation is $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\epsilon}{2} g_N + \sqrt{\epsilon}\Delta W$

Otherwise **exactly standard GD** but we add explicit noise, and now the target is correct (not just some Gaussian)

But: Computationally inefficient as we need the full gradient

Stochastic gradient Langevin dynamics (SGLD)

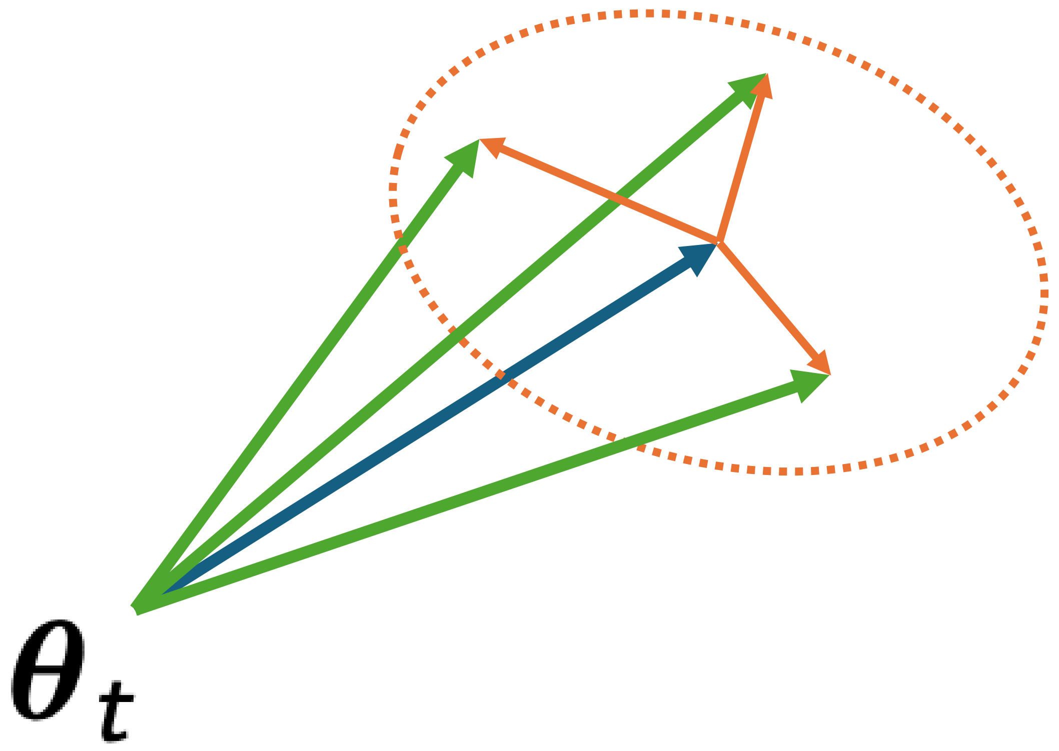
Combine LD with stochastic gradients

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\epsilon}{2} g_S + \sqrt{\epsilon}\Delta W \qquad g_S = g_N + \frac{1}{\sqrt{S}}\gamma \qquad \gamma \sim \mathcal{N}(0, \mathbf{B}\mathbf{B}^T)$$

Now two noise sources: **Gradient noise** and the **injected noise**

- Gradient noise variance proportional to ε^2 and the injected noise to ε
- For small ε the latter dominates => This is still Langevin dynamics!
- For really small ε the discretization error disappears => Samples from the right target

Gradient noise

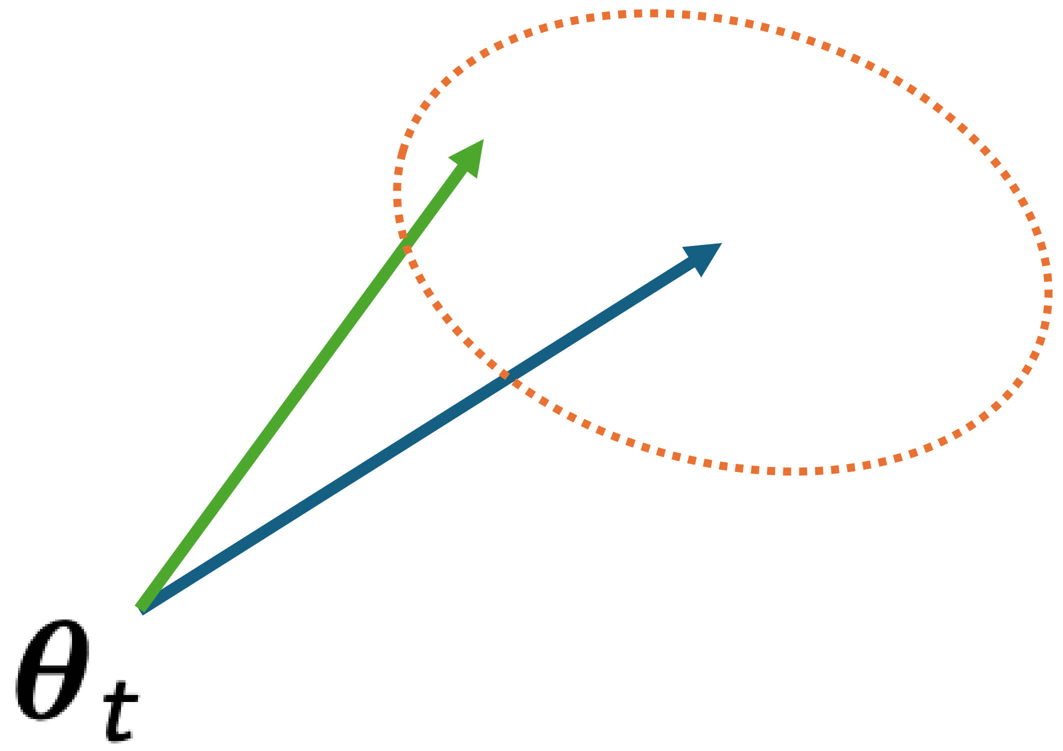


Full gradient

Mini-batch estimates

Gradient noise

Gradient noise and injected noise

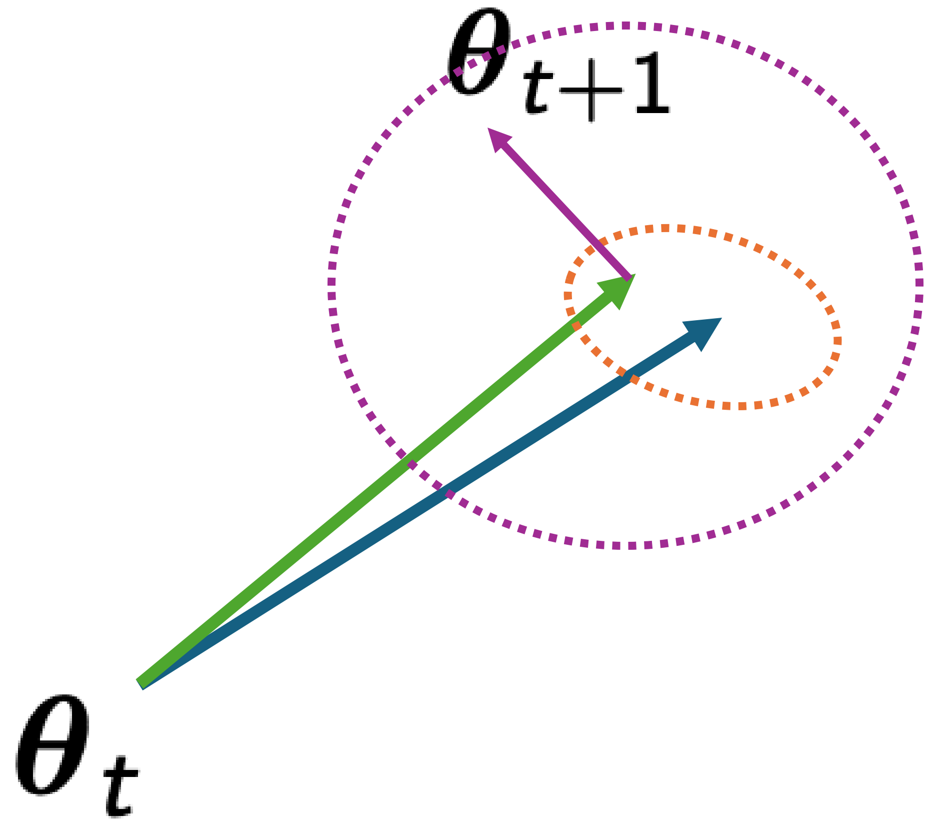


Full gradient

Mini-batch estimates

Gradient noise

Gradient noise and injected noise



Full gradient

Mini-batch estimates

Gradient noise

Injected noise

SGLD

Run SGD but add independent Gaussian noise after making the gradient step, with variance of the step-length (times square root of 2)

Reduce step-length gradually as usual, to transition from optimization to sampling

1. **Beginning**: Gradients are large and works as optimizer; neither noise matters!
2. **Middle**: Both noises play a role – a bit muddy phase
3. **End**: Injected noise dominates and samples from the right target!

SGLD

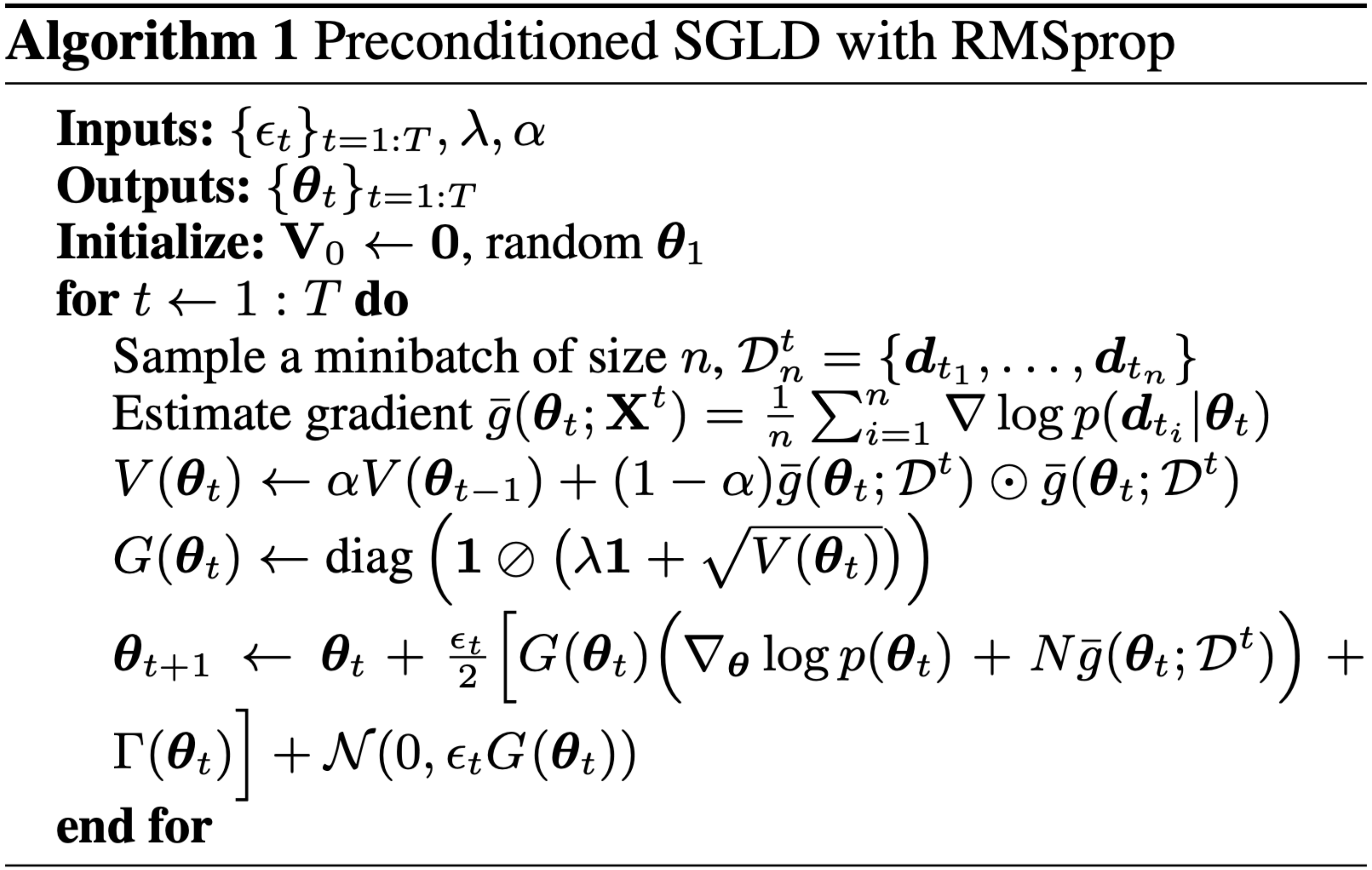
Similar to SGD we can (and should) use preconditioners

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\epsilon}{2}\mathbf{M} g_S + \sqrt{\epsilon}\,\mathsf{Chol}(\mathbf{M})\Delta W$$

Position-dependent scaling is even better: Instead of fixed preconditioner, we have $\mathbf{M}(\boldsymbol{\theta})$

- Valid sampler in a Riemannian geometry; see Girolami et al. (2013)
- pSGLD by Li et al. (2015) uses the RMSProp princple for updating $\mathbf{M}(\boldsymbol{\theta})$
    - Not just the preconditioner, but also momentum for gradient and variance estimates

pSGLD (Li et al. 2015)



Algorithm 1 Preconditioned SGLD with RMSprop

```
Inputs: $\{\epsilon_t\}_{t=1:T}, \lambda, \alpha$
Outputs: $\{\boldsymbol{\theta}_t\}_{t=1:T}$
Initialize: $\mathbf{V}_0 \leftarrow \mathbf{0}$, random $\boldsymbol{\theta}_1$
for $t \leftarrow 1 : T$ do
    Sample a minibatch of size $n$, $\mathcal{D}_n^t = \{\boldsymbol{d}_{t_1}, \ldots, \boldsymbol{d}_{t_n}\}$
    Estimate gradient $\bar{g}(\boldsymbol{\theta}_t; \mathbf{X}^t) = \frac{1}{n}\sum_{i=1}^n \nabla \log p(\boldsymbol{d}_{t_i}|\boldsymbol{\theta}_t)$
    $V(\boldsymbol{\theta}_t) \leftarrow \alpha V(\boldsymbol{\theta}_{t-1}) + (1-\alpha)\bar{g}(\boldsymbol{\theta}_t; \mathcal{D}^t) \odot \bar{g}(\boldsymbol{\theta}_t; \mathcal{D}^t)$
    $G(\boldsymbol{\theta}_t) \leftarrow \text{diag}\left(\mathbf{1} \oslash \left(\lambda\mathbf{1} + \sqrt{V(\boldsymbol{\theta}_t)}\right)\right)$
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \frac{\epsilon_t}{2}\Big[G(\boldsymbol{\theta}_t)\Big(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t) + N\bar{g}(\boldsymbol{\theta}_t; \mathcal{D}^t)\Big) + \Gamma(\boldsymbol{\theta}_t)\Big] + \mathcal{N}(0, \epsilon_t G(\boldsymbol{\theta}_t))$
end for
```

SGLD

Unforunately not a valid sampler!

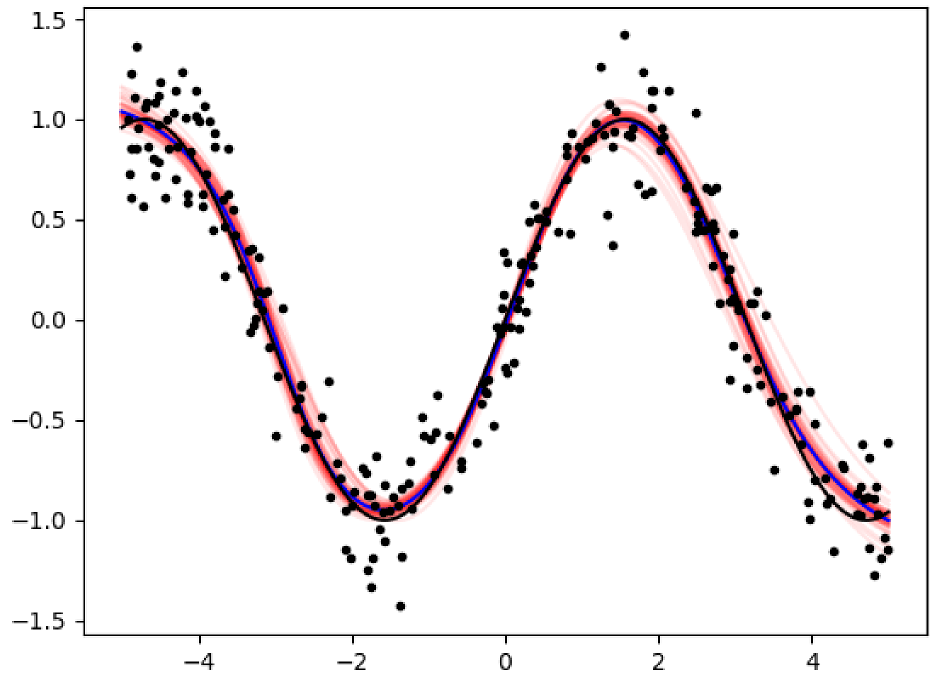
- MH rejection rate indeed goes to zero with the step length, but we also stop moving around altogether
- For any useful learning rate SGLD is biased: The samples are not from the correct target

Nevertheless, people do use SGLD and especially pSGLD

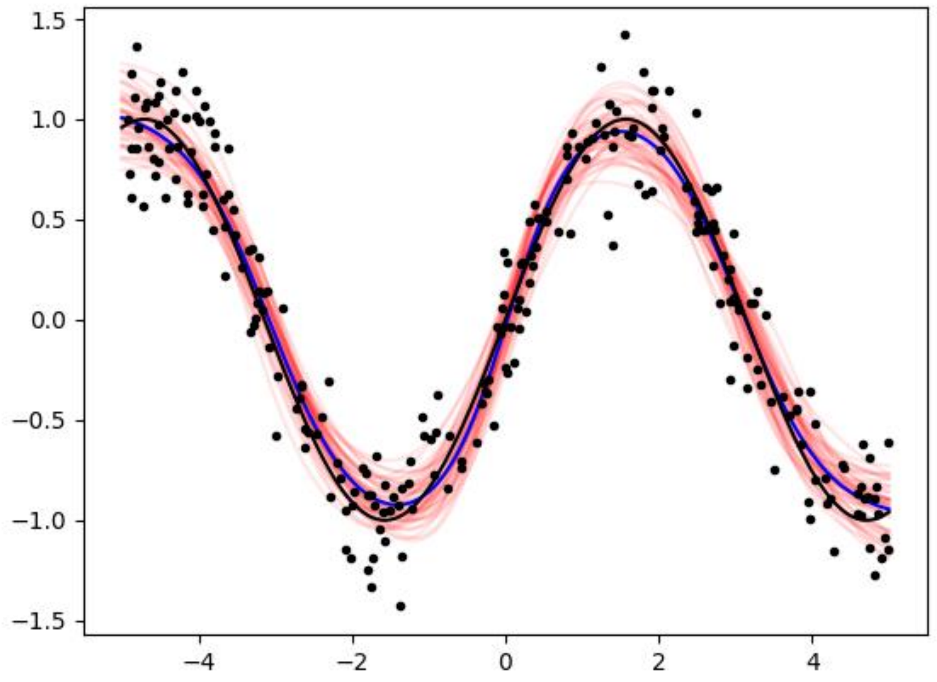
- Extremely simple algorithm, just SGD or RMSProp with added noise
- No need to explicitly estimate the gradient noise as in SGD

SGLD in action (N=256)

SGD



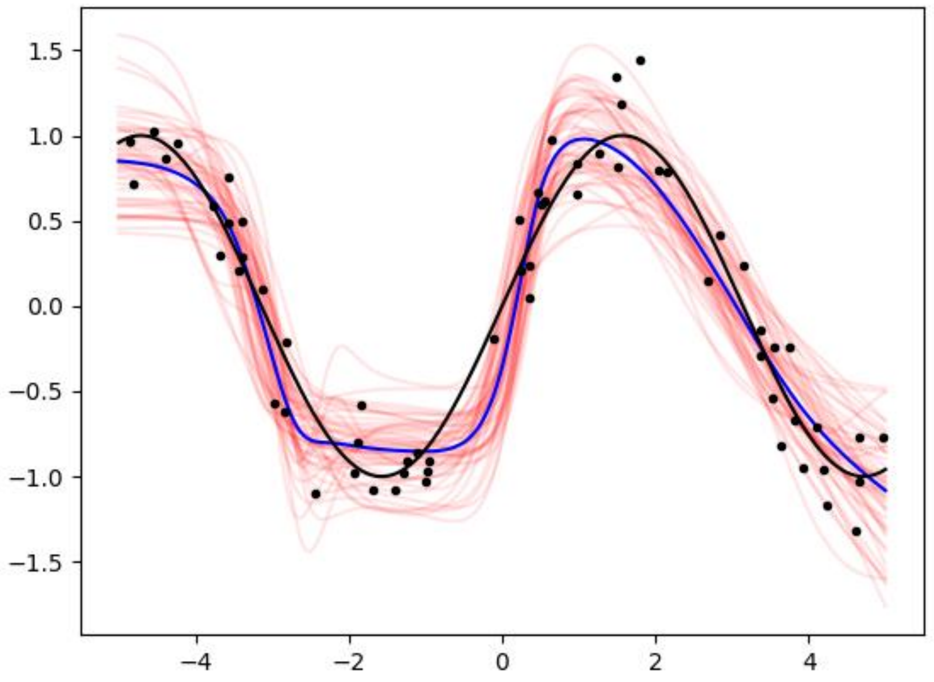
SGLD



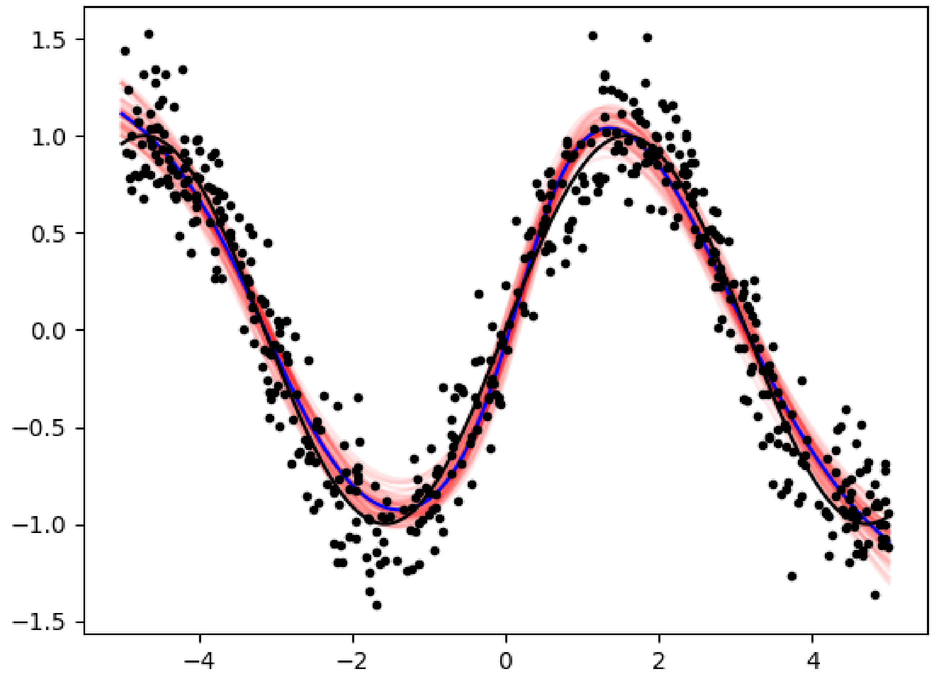
Not identical but similar. Which one is more correct?

SGLD in action (varying N)

N=64



N=512



Fixing the bias

$$p(\text{accept}) = \frac{p(\mathcal{D}, \boldsymbol{\theta}')}{p(\mathcal{D}, \boldsymbol{\theta})} \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})}$$

MALA for *Metropolis-Adjusted Langevin Algorithm* fixes the bias by adding a standard Metropolis-Hastings check

- In early phase accepts all proposals as the gradient update dominates
- For small step-lengths should be accepting around 60% or so

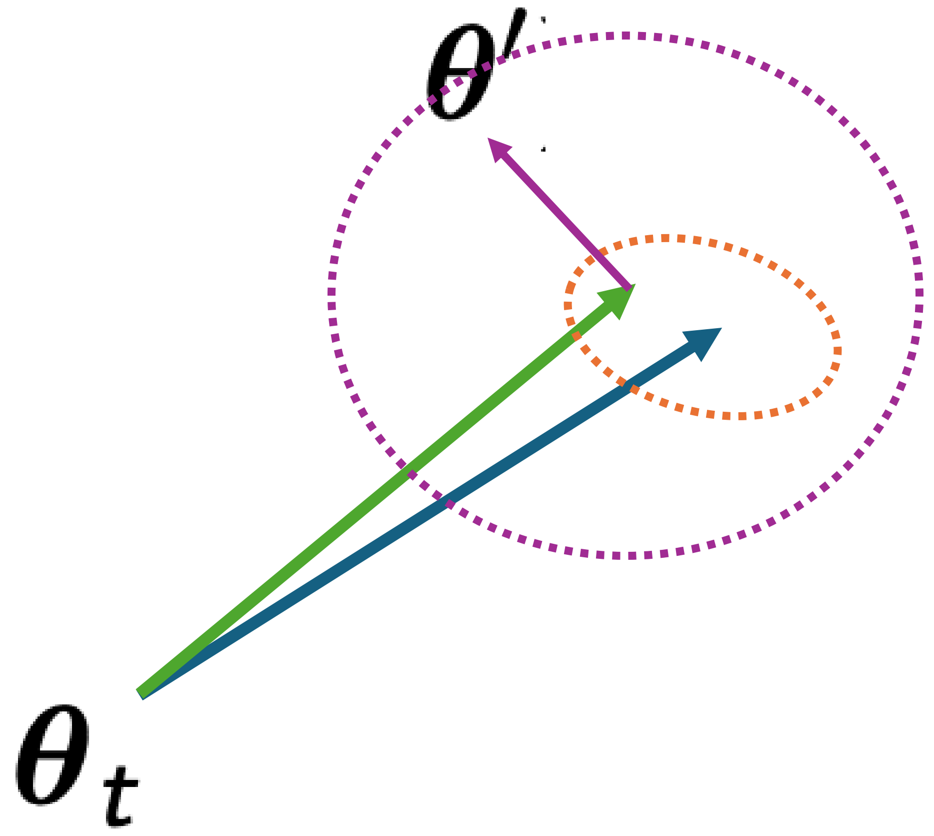
Doubles computation

- We need to compute gradients also at the proposal, to evaluate the proposal distribution

Optimization template no longer good for implementation

- Need to form explicitly the distribution from which the proposal was sampled, for density computation
- Building on optimization routine no longer optimal: Requires a bit tedious manipulation of the neural network state
- Switch to MCMC templates!

Gradient noise and injected noise



Full gradient

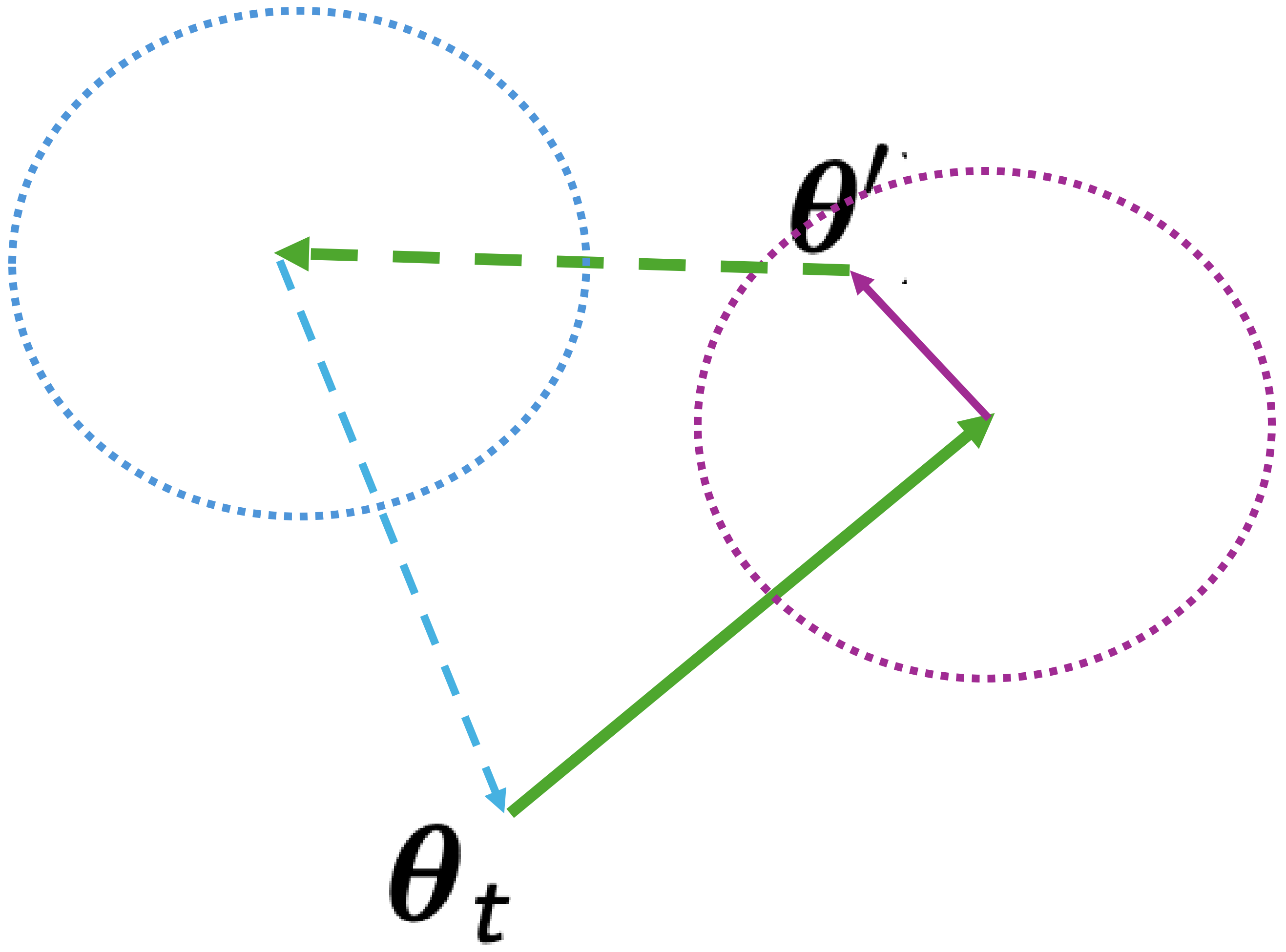
Mini-batch estimates

Gradient noise

~~Injected noise~~
Proposal $q(\theta' | \theta_t)$

Metropolis-Hastings

$$p(\text{accept}) = \frac{p(\mathcal{D}, \boldsymbol{\theta}')}{p(\mathcal{D}, \boldsymbol{\theta})} \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})}$$

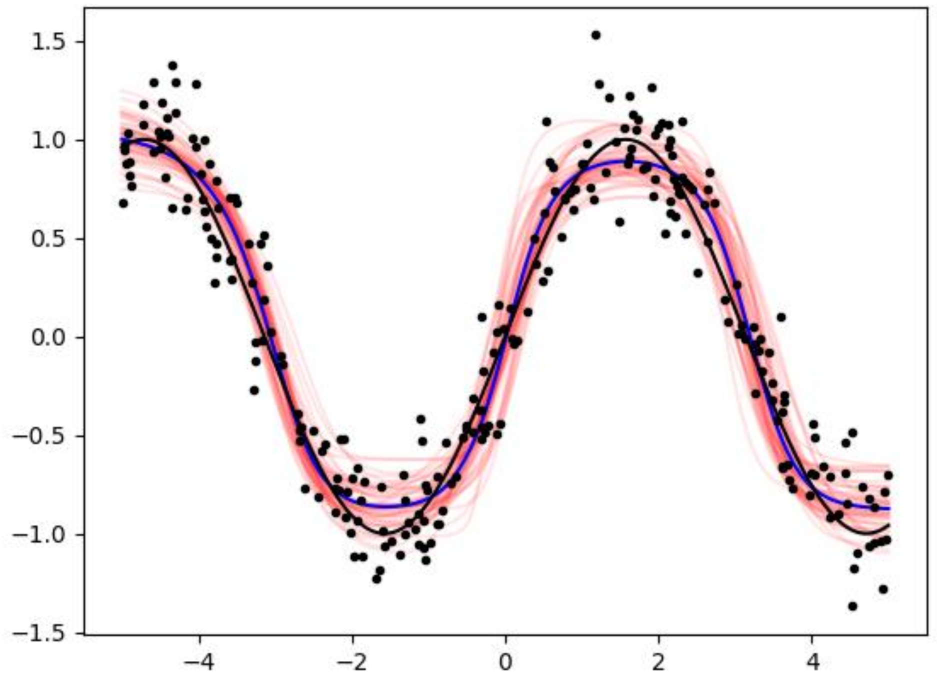


Proposal $q(\theta'|\,\theta_t)$

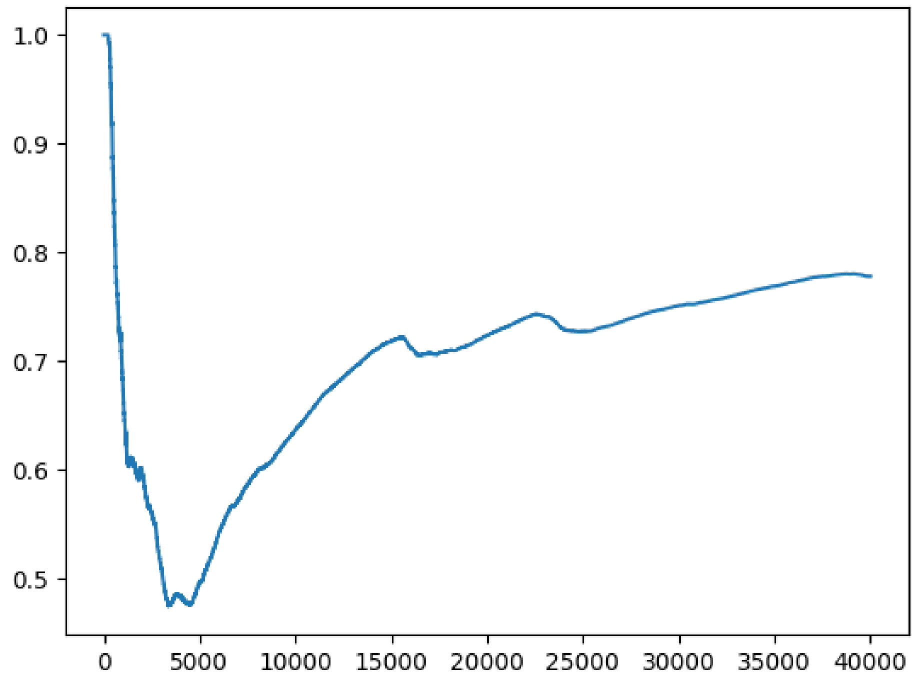
Proposal $q(\theta_t|\,\theta')$

What will $\boldsymbol{\theta}_{t+1}$ be?

MALA in action (N=256)



Cumulative acceptance probability



Historical remark

We went from SGD to SGLD and then MALA, but the actual history goes the other way around

- MALA is from 1994/2003, with Riemannian preconditioners from 2011
- SGLD is from 2011 and pSGLD from 2015
- SGD as a sampler is from 2017

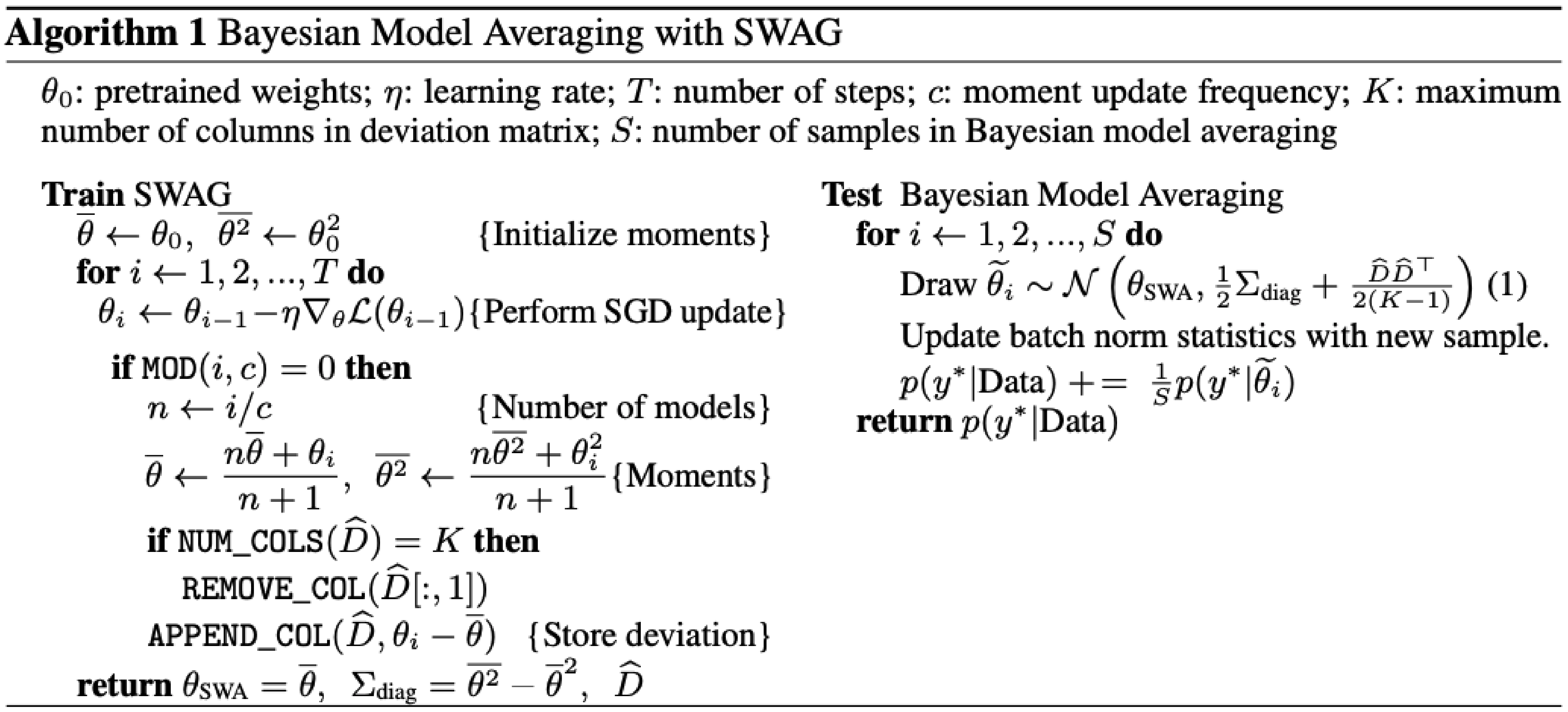
This is all fine: Making the (already quite simple) algorithms approachable for the masses and easy for neural networks

- For instance, the SGLD paper may still be hard to read but the algorithm is ridiculously simple
- RMSProp works well in optimization, so why not leverage the ideas in sampling

Simplicity vs exactness

- SWAG (Stochastic Weight Averaging Gaussian) approximates things even further (Maddox et al. 2019)
- Run SGD and form a Gaussian approximation by estimating the mean and variance of the parameters themselves over the iterations
    - Forms explicit approximation instead of storing the samples, but has similar theoretical justification
- Predictions by Monte Carlo integration over samples from this distribution as usual
    - Resolves the memory overhead

Simplicity vs exactness



Algorithm 1 Bayesian Model Averaging with SWAG

θ_0: pretrained weights; η: learning rate; T: number of steps; c: moment update frequency; K: maximum number of columns in deviation matrix; S: number of samples in Bayesian model averaging

```
Train SWAG
  $\overline{\theta} \leftarrow \theta_0, \ \overline{\theta^2} \leftarrow \theta_0^2$    {Initialize moments}
  for $i \leftarrow 1, 2, ..., T$ do
    $\theta_i \leftarrow \theta_{i-1} - \eta \nabla_\theta \mathcal{L}(\theta_{i-1})$    {Perform SGD update}
    if MOD$(i, c) = 0$ then
      $n \leftarrow i/c$    {Number of models}
      $\overline{\theta} \leftarrow \frac{n\overline{\theta} + \theta_i}{n+1}, \ \overline{\theta^2} \leftarrow \frac{n\overline{\theta^2} + \theta_i^2}{n+1}$    {Moments}
      if NUM_COLS$(\widehat{D}) = K$ then
        REMOVE_COL$(\widehat{D}[:, 1])$
      APPEND_COL$(\widehat{D}, \theta_i - \overline{\theta})$    {Store deviation}
  return $\theta_{\text{SWA}} = \overline{\theta}, \ \Sigma_{\text{diag}} = \overline{\theta^2} - \overline{\theta}^2, \ \widehat{D}$
```

```
Test  Bayesian Model Averaging
  for $i \leftarrow 1, 2, ..., S$ do
    Draw $\widetilde{\theta}_i \sim \mathcal{N}\left(\theta_{\text{SWA}}, \frac{1}{2}\Sigma_{\text{diag}} + \frac{\widehat{D}\widehat{D}^\top}{2(K-1)}\right)$ (1)
    Update batch norm statistics with new sample.
    $p(y^*|\text{Data}) \mathrel{+}= \frac{1}{S} p(y^*|\widetilde{\theta}_i)$
  return $p(y^*|\text{Data})$
```

Hands-on break

Let’s do SGLD, starting from our previous SGD code

…and MALA if you want to try your hands on something more complex

Discussion

We have been going through fairly old algorithms, some pre-dating many of the advances in deep learning

Still, pSGLD and MALA are very much concrete tools

- pSGLD builds effectively on RMSProp as an optimizer and inherits at least some of its good properties
- MALA is a practical method to use e.g. inside more complex algorithms, such as diffusion models

Discussion

We have only been talking about **local** posterior around the mode, similar to Laplace approximation

- SGD derivation directly about the local properties, and SWAG makes explicit Gaussian assumption
- SGLD/MALA in principle are general samplers but often get stuck in the mode in practice

Not necessarily a problem: It is okay to stay within one mode

- In fact, we should not even monitor mixing in the parameter space but in the space of the induced functions (Izmailov et al. 2021)

Hamiltonian Monte Carlo helps in making bigger steps

- Stochastic variants of HMC are somewhat more challenging, but possible (Chen et al. 2014)

Discussion

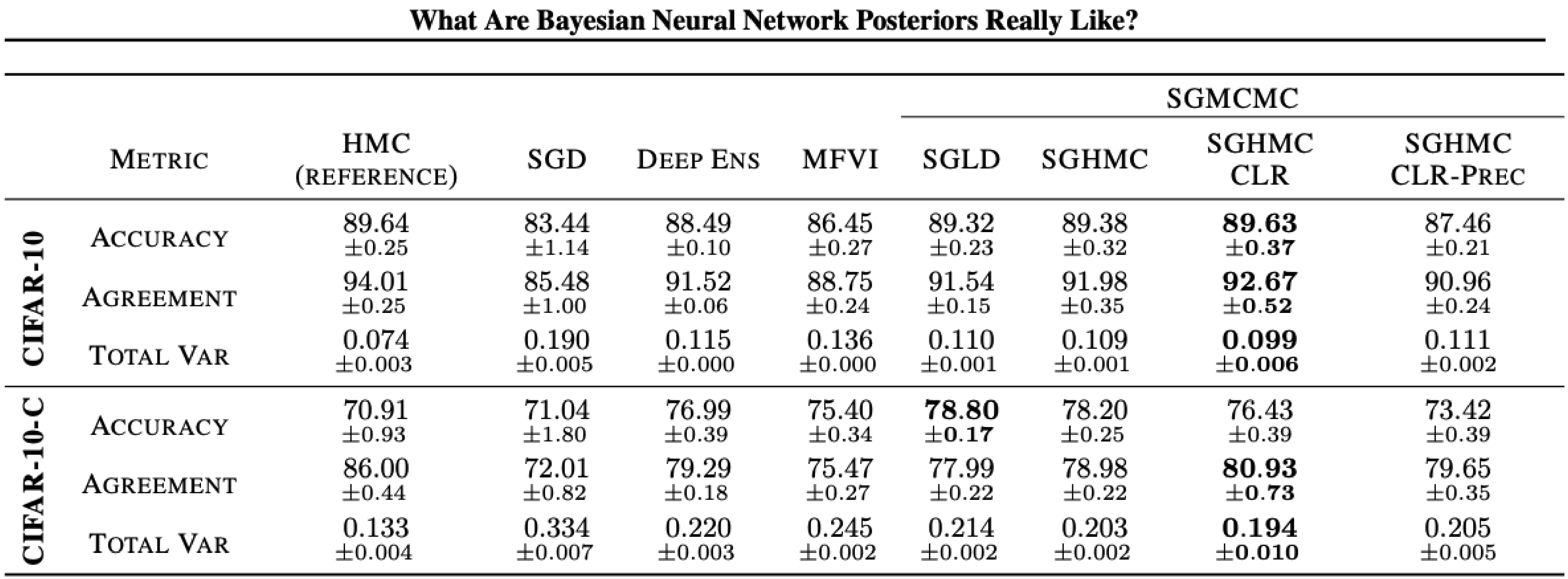
Priors ignored here, but they do play some role

- Priors often quite simple (wide Gaussians), and perhaps matter less than architecture and various other choices (Izmailov et al. 2021)
- More complex priors likely require more accurate samplers (Yu et al. 2023)
- We likely should use function-space priors anyway, not priors over parameters

Are the models good?

- Predictive accuracy: Predictive accuracy in isolation is highly valuable even if we did not sample from the correct posterior
- Explicit quantification of calibration of the predictive distribution

Discussion

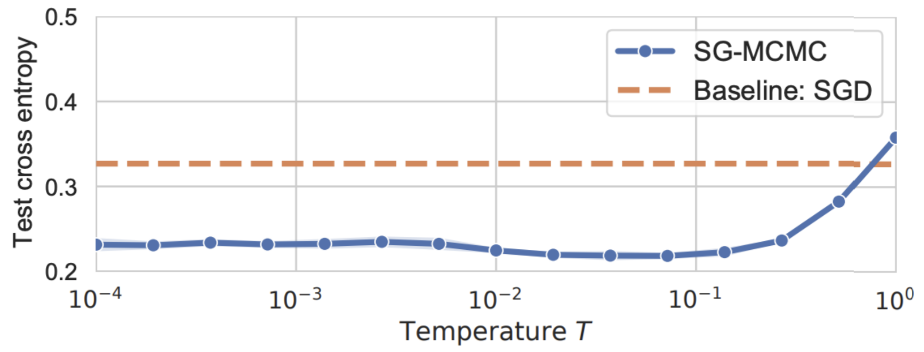


What Are Bayesian Neural Network Posteriors Really Like?

						SGMCMC			
	METRIC	HMC (REFERENCE)	SGD	DEEP ENS	MFVI	SGLD	SGHMC	SGHMC CLR	SGHMC CLR-PREC
CIFAR-10	ACCURACY	89.64 ±0.25	83.44 ±1.14	88.49 ±0.10	86.45 ±0.27	89.32 ±0.23	89.38 ±0.32	**89.63 ±0.37**	87.46 ±0.21
	AGREEMENT	94.01 ±0.25	85.48 ±1.00	91.52 ±0.06	88.75 ±0.24	91.54 ±0.15	91.98 ±0.35	**92.67 ±0.52**	90.96 ±0.24
	TOTAL VAR	0.074 ±0.003	0.190 ±0.005	0.115 ±0.000	0.136 ±0.000	0.110 ±0.001	0.109 ±0.001	**0.099 ±0.006**	0.111 ±0.002
CIFAR-10-C	ACCURACY	70.91 ±0.93	71.04 ±1.80	76.99 ±0.39	75.40 ±0.34	**78.80 ±0.17**	78.20 ±0.25	76.43 ±0.39	73.42 ±0.39
	AGREEMENT	86.00 ±0.44	72.01 ±0.82	79.29 ±0.18	75.47 ±0.27	77.99 ±0.22	78.98 ±0.22	**80.93 ±0.73**	79.65 ±0.35
	TOTAL VAR	0.133 ±0.004	0.334 ±0.007	0.220 ±0.003	0.245 ±0.002	0.214 ±0.002	0.203 ±0.002	**0.194 ±0.010**	0.205 ±0.005

Izmailov et al. 2021

Discussion

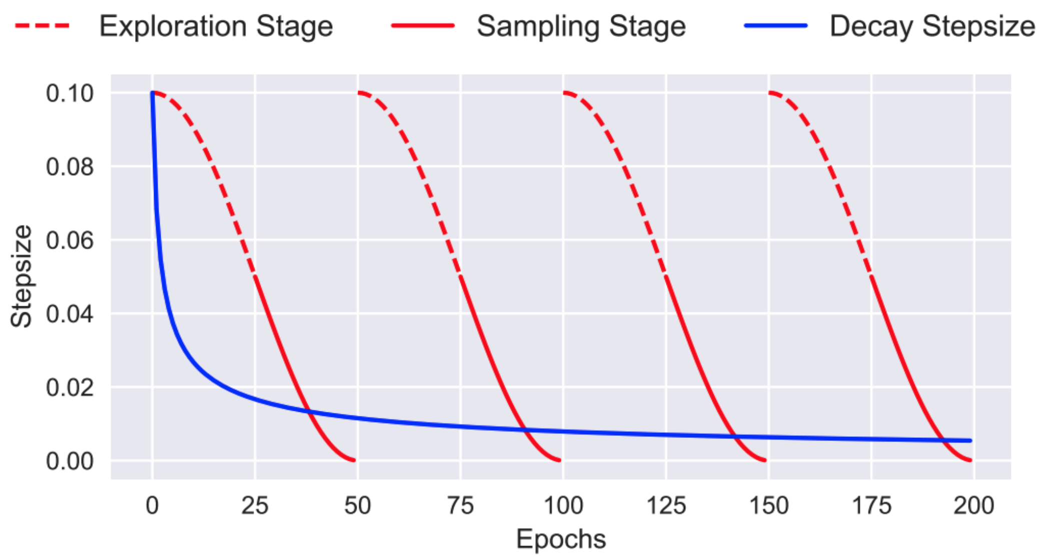


Some papers discussing the 'current' state of BNNs

- Papamarkou et al. (2021) focuses on small models
- Wenzel et al. (2020) claims we should 'cool' the log-density to make the posterior smother
- Izmailov et al. (2021) corrects Wenzel
  - Various DL heuristics, like data augmentation, interact with sampling

Various heuristics tend to help

- Cyclical learning rates (Zhang et al. 2020)
- Layer-wise preconditioning (Yu et al. 2023)



Conclusions

Don’t be afraid to try modelling predictive uncertainty with NNs

Simple variants of optimization algorithms (SGLD, pSGLD, SWAG etc) have solid theoretical basis and provide useful approximations of the uncertainty

They are not exact, but perhaps nothing will ever be for complex enough models

- Better or worse than explicit (variational) approximation methods?

Role of Bayesian inference for really large models?

ProbAI party trick

When your colleague says “Bayesian NNs are hard”:

1. Offer a bet that you can change their current solution into a Bayesian one in five minutes

2. Open the code and implement SGLD
   1. Add noise to the gradient
   2. If they already store the weights during iteration, just loop over them at prediction time. If not, say that the code gives one sample from the posterior.

3. Lose the bet because the objective is not properly scaled

References

1) Chen et al. Stochastic gradient Hamiltonian Monte Carlo, 2014.
2) Girolami and Calderhead. Riemann manifold Langeving and Hamiltonian Monte Carlo, 2011.
3) Izmailov et al. What are Bayesian neural network posteriors really like?, 2021.
4) Li et al. Preconditioned stochastic gradient Langeving dynamics for deep neural networks, 2015.
5) Maddox et al. A simple baseline for Bayesian uncertainty in deep learning, 2019.
6) Mandt et al. Stochastic gradient descent as approximate Bayesian inference, 2017.
7) Mikkola et al. Preferential normalizing flows, 2024.
8) Papamarkou et al. Challenges in Markov chain Monte Carlo for Bayesian neural networks, 2021.
9) Welling and Teh. Bayesian learning via stochastic gradient Langevin dynamics, 2011.
10) Wenzel et al. How good is the Bayes posterior in deep neural networks really?, 2020.
11) Williams et al. Geodesic slice sampler for multimodal distributions with strong curvature, 2025.
12) Yu et al. Scalable stochastic gradient Riemannian Langevin dynamics in non-diagonal metrics, 2023.
13) Zhang et al. Cyclical stochastic gradient MCMC for Bayesian deep learning, 2020.