Meta-learning for stochastic gradient MCMC

Yingzhen Li

University of Cambridge → Microsoft Research Cambridge

Joint work with Wenbo Gong & José Miguel Hernández-Lobato (U Cambridge)
arXiv 1806.04522

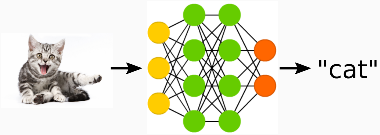


.

Bayesian neural networks 101

Goal: classifying different types of cats from images

- $\boldsymbol{x}$: input images; $\boldsymbol{y}$: output label
- build a neural network (with param. $\boldsymbol{\theta}$):
  $\hat{\boldsymbol{y}} = \mathrm{NN}_{\boldsymbol{\theta}}(\boldsymbol{x})$

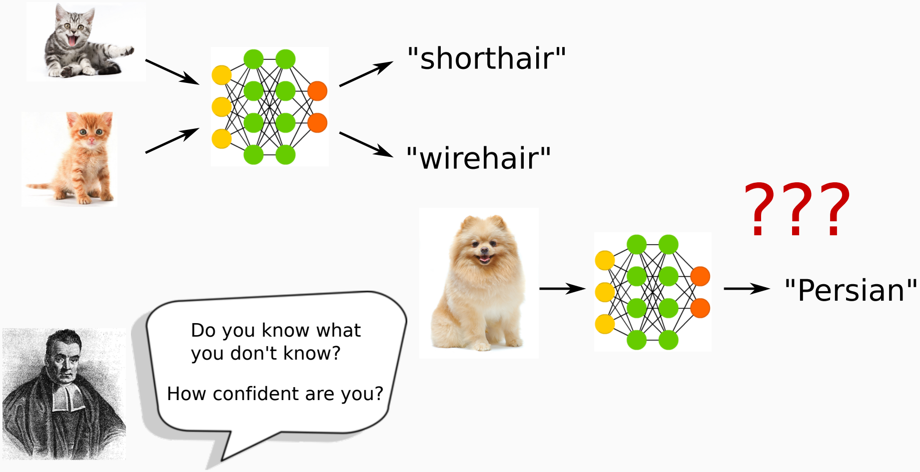


- find the best parameter $\boldsymbol{\theta}$ given a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log p(\boldsymbol{y}_n | \boldsymbol{x}_n, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

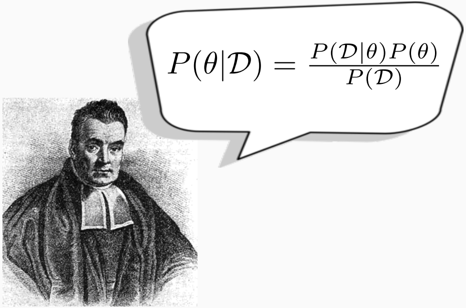
Maximum a posteriori (MAP)

Bayesian neural networks 101



Bayesian neural networks 101

Bayesian inference: given some function $F(\boldsymbol{\theta})$, want $\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\left[F(\boldsymbol{\theta})\right]$



- predictive mean

$$\hat{\boldsymbol{y}}_{\text{mean}} = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\left[\text{NN}_{\boldsymbol{\theta}}(\boldsymbol{x})\right]$$

- predictive distribution

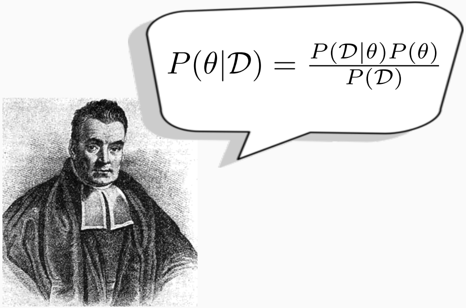
$$p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\left[p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})\right]$$

- evaluate posterior

$$p(\boldsymbol{\theta} \in A|\mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\left[\delta_A\right]$$

Bayesian neural networks 101

Bayesian inference: given some function $F(\boldsymbol{\theta})$, want $\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[F(\boldsymbol{\theta})]$



- Monte Carlo estimation:

$$\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[F(\boldsymbol{\theta})] \approx \frac{1}{K}\sum_{k=1}^{K} F(\boldsymbol{\theta}^k)$$

$$\boldsymbol{\theta}^k \sim p(\boldsymbol{\theta}|\mathcal{D}) \qquad \text{(intractable)}$$

- Stochastic gradient MCMC (SG-MCMC): efficient ways to (approximately) draw samples from $p(\boldsymbol{\theta}|\mathcal{D})$

From SGD to SG-MCMC

- The MAP problem can be rewritten as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} U(\boldsymbol{\theta}),$$

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto \exp[-U(\boldsymbol{\theta})], \quad -U(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

From SGD to SG-MCMC

- The MAP problem can be rewritten as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} U(\boldsymbol{\theta}),$$

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto \exp[-U(\boldsymbol{\theta})], \quad -U(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

- In the MAP problem, we find $\boldsymbol{\theta}^*$ by gradient descent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} U(\boldsymbol{\theta}_t),$$

$$-\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) = \underbrace{\sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{\theta})}_{\text{full gradient of LL}} + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}), \quad \{(\boldsymbol{x}_m, \boldsymbol{y}_m)\}_{m=1}^{M} \sim \mathcal{D}^M$$

From SGD to SG-MCMC

- The MAP problem can be rewritten as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} U(\boldsymbol{\theta}),$$

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto \exp[-U(\boldsymbol{\theta})], \quad -U(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

- In the MAP problem, we find $\boldsymbol{\theta}^*$ by stochastic gradient descent (for big data)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \tilde{U}(\boldsymbol{\theta}_t),$$

$$-\nabla_{\boldsymbol{\theta}} \tilde{U}(\boldsymbol{\theta}) = \underbrace{\frac{N}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}_m|\boldsymbol{x}_m, \boldsymbol{\theta})}_{\text{stochastic gradient of LL}} + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}), \quad \{(\boldsymbol{x}_m, \boldsymbol{y}_m)\}_{m=1}^{M} \sim \mathcal{D}^M$$

From SGD to SG-MCMC

- In the Bayesian inference problem, we need to (approximately) draw $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})$
- Big data: Stochastic gradient Langevin dynamics (SGLD)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \tilde{U}(\boldsymbol{\theta}_t) + \sqrt{2\eta}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Welling and Teh (2011), Chen et al. (2014), Li et al. (2016), Chen et al. (2016)

From SGD to SG-MCMC

- In the Bayesian inference problem, we need to (approximately) draw $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})$
- Big data: Stochastic gradient Langevin dynamics (SGLD)

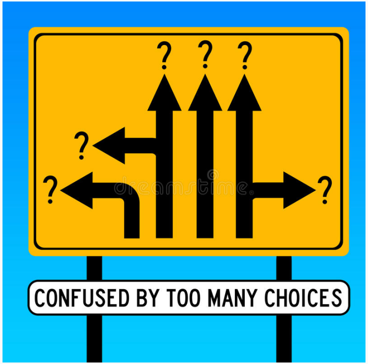
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \tilde{U}(\boldsymbol{\theta}_t) + \sqrt{2\eta}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- SGLD = SGD + properly scaled Gaussian noise
- Other optimisation algorithms can be transformed into SG-MCMC samplers:
    - SGD + momentum $\rightarrow$ SGHMC; RMSprop $\rightarrow$ preconditioned SGLD; Adam $\rightarrow$ Santa

Welling and Teh (2011), Chen et al. (2014), Li et al. (2016), Chen et al. (2016)

“I’m bored of tuning my optimiser & sampler”

- Which SG-MCMC algorithm should I use?
- How do I tune the hyper-parameters?



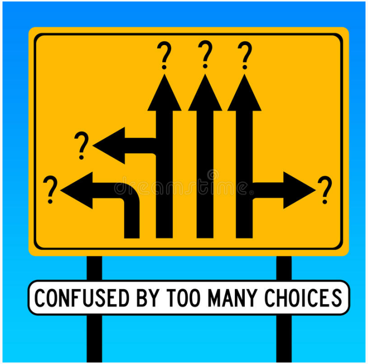
Salimans et al. (2015), Song et al. (2017), Levy et al. (2018)

"I'm bored of tuning my optimiser & sampler"

- Which SG-MCMC algorithm should I use?
- How do I tune the hyper-parameters?

Learn it from data!

- Want a general solution for similar tasks
- Train on low-dim, generalise to high-dim



Salimans et al. (2015), Song et al. (2017), Levy et al. (2018)

Learning to learn

Meta-learning for optimisers:

- Define an optimiser with parameters ϕ:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \boldsymbol{f}_\phi(\boldsymbol{z}_t, H(\cdot))$$

- Run it on some training objective functions $H(\boldsymbol{z})$, provide learning signals to train ϕ
- Once learned, apply this optimiser to test objective functions

Andrychowicz et al. (2016), Li and Malik (2017), Wichrowska et al. (2017), Li and Turner (2018)

Learning to learn

Meta-learning for SG-MCMC: can we just naively

- Define a sampler with parameters ϕ:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \boldsymbol{f}_\phi(\boldsymbol{z}_t, H(\cdot), \boldsymbol{\epsilon}), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- Run it on some training distributions $\pi(\boldsymbol{z}) \propto \exp[-H(\boldsymbol{z})]$, provide learning signals to train ϕ
- Once learned, apply this sampler to test distributions

Andrychowicz et al. (2016), Li and Malik (2017), Wichrowska et al. (2017), Li and Turner (2018)

Learning to learn

Meta-learning for SG-MCMC: can we just naively

- Define a sampler with parameters ϕ:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \boldsymbol{f}_\phi(\boldsymbol{z}_t, H(\cdot), \boldsymbol{\epsilon}), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- Run it on some training distributions $\pi(\boldsymbol{z}) \propto \exp[-H(\boldsymbol{z})]$, provide learning signals to train ϕ
- Once learned, apply this sampler to test distributions

Not quite yet! We need to make sure it is a valid sampler!

Andrychowicz et al. (2016), Li and Malik (2017), Wichrowska et al. (2017), Li and Turner (2018)

The complete framework: Ma et al. NIPS 2015

To sample from $\pi(\boldsymbol{z}) \propto \exp[-H(\boldsymbol{z})]$:

- Let's take the step-size $\eta \to 0$ and use exact gradient:

$$d\boldsymbol{z} = -\nabla_{\boldsymbol{z}} H(\boldsymbol{z}) dt + \sqrt{2} dW(t) \qquad \text{(Langevin dynamics)}$$

- $W(t)$ is a Wiener process
  (think about $dW(t)$ as some Gaussian noise with variance dt)
- Langevin dynamics is a special case of Itô diffusion

$$d\boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{z}) dt + \sqrt{2\boldsymbol{D}(\boldsymbol{z})} dW(t)$$

The complete framework: Ma et al. NIPS 2015

- Itô diffusion

$$d\boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{z})dt + \sqrt{2\boldsymbol{D}(\boldsymbol{z})}dW(t) \qquad (1)$$

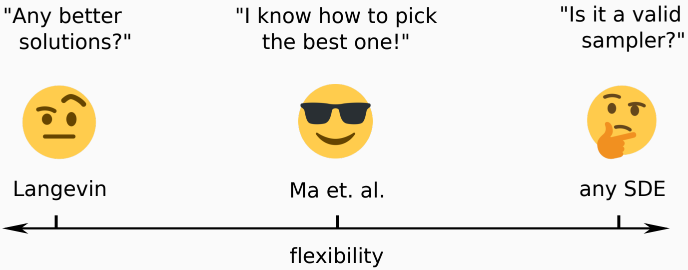
- To make sure $\pi(\boldsymbol{z}) \propto \exp[-H(\boldsymbol{z})]$ is a stationary distribution:

$$\boldsymbol{\mu}(\boldsymbol{z}) = -[\boldsymbol{D}(\boldsymbol{z}) + \boldsymbol{Q}(\boldsymbol{z})]\nabla_{\boldsymbol{z}} H(\boldsymbol{z}) + \boldsymbol{\Gamma}(\boldsymbol{z}), \quad \boldsymbol{\Gamma}(\boldsymbol{z})_i = \sum_{j=1}^{d} \frac{\partial}{\partial \boldsymbol{z}_j}[D_{ij}(\boldsymbol{z}) + Q_{ij}(\boldsymbol{z})] \qquad (2)$$

- $\boldsymbol{D}(\boldsymbol{z})$: diffusion matrix, PSD
- $\boldsymbol{Q}(\boldsymbol{z})$: curl matrix, skew-symmetric
- $\boldsymbol{\Gamma}(\boldsymbol{z})$: correction vector

Ma et al. (2015) completeness result: under some mild conditions
"Any Itô diffusion that has the unique stationary $\pi(\boldsymbol{z})$ is governed by (1)+(2)"

The complete framework: Ma et al. NIPS 2015



- Searching the best sampler within the complete framework:
  - Guaranteed to be correct
  - Retains the most flexibility
  - Only needs to learn how to parameterise $\boldsymbol{D}(\boldsymbol{z})$ and $\boldsymbol{Q}(\boldsymbol{z})$ matrices!

Our recipe: dynamics design

- Goal: train an SG-MCMC sampler to sample from $p(\boldsymbol{\theta}|\mathcal{D}) \propto \exp[-U(\boldsymbol{\theta})]$
- We augment the state space with momentum variable $\boldsymbol{p}$:

$$\boldsymbol{z} = (\boldsymbol{\theta}, \boldsymbol{p}), \quad \pi(\boldsymbol{z}) \propto \exp[-H(\boldsymbol{z})], \quad H(\boldsymbol{z}) = U(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{p}^{\mathsf{T}}\boldsymbol{p}$$

- Recall the complete recipe

$$d\boldsymbol{z} = -[\boldsymbol{D}(\boldsymbol{z}) + \boldsymbol{Q}(\boldsymbol{z})]\nabla_{\boldsymbol{z}} H(\boldsymbol{z})dt + \boldsymbol{\Gamma}(\boldsymbol{z})dt + \sqrt{2}dW(t)$$

Our recipe: dynamics design

- Our recipe:

$$\boldsymbol{Q}(\boldsymbol{z}) = \begin{bmatrix} \mathbf{0} & -\boldsymbol{Q}_f(\boldsymbol{z}) \\ \boldsymbol{Q}_f(\boldsymbol{z}) & \mathbf{0} \end{bmatrix}, \quad \boldsymbol{D}(\boldsymbol{z}) = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{D}_f(\boldsymbol{z}) \end{bmatrix}, \quad \boldsymbol{\Gamma}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{\Gamma}_{\boldsymbol{\theta}}(\boldsymbol{z}) \\ \boldsymbol{\Gamma}_{\boldsymbol{p}}(\boldsymbol{z}) \end{bmatrix}$$

$$\boldsymbol{Q}_f(\boldsymbol{z}) = \text{diag}[\boldsymbol{f}_{\phi_Q}(\boldsymbol{z})], \quad \boldsymbol{D}_f(\boldsymbol{z}) = \text{diag}[\alpha \boldsymbol{f}_{\phi_Q}(\boldsymbol{z}) \odot \boldsymbol{f}_{\phi_Q}(\boldsymbol{z}) + \boldsymbol{f}_{\phi_D}(\boldsymbol{z}) + c], \quad \alpha, c > 0$$

- Resulting update rules (rearrange terms & discretise & stochastic gradient):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \overbrace{\eta \boldsymbol{Q}_f(\boldsymbol{z}_t)\boldsymbol{p}_t}^{\text{momentum SGD}} + \overbrace{\eta \boldsymbol{\Gamma}_{\boldsymbol{\theta}}(\boldsymbol{z}_t)}^{\text{correction}}$$

$$\boldsymbol{p}_{t+1} = \boldsymbol{p}_t - \underbrace{\eta \boldsymbol{D}_f(\boldsymbol{z}_t)\boldsymbol{p}_t}_{\textit{friction}} - \eta \boldsymbol{Q}_f(\boldsymbol{z}_t)\nabla_{\boldsymbol{\theta}_t}\tilde{U}(\boldsymbol{\theta}_t) + \eta \boldsymbol{\Gamma}_{\boldsymbol{p}}(\boldsymbol{z}_t) + \sqrt{\Sigma(\boldsymbol{z}_t)}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\Sigma(\boldsymbol{z}_t) = 2\eta \boldsymbol{D}_f(\boldsymbol{z}_t) - \eta^2 \boldsymbol{Q}_f(\boldsymbol{z}_t)\boldsymbol{B}(\theta_t)\boldsymbol{Q}_f(\boldsymbol{z}_t), \quad \boldsymbol{B}(\theta_t) = \mathbb{V}[\nabla_{\boldsymbol{\theta}_t}\tilde{U}(\boldsymbol{\theta}_t)]$$

Our recipe: dynamics design

Designing $\boldsymbol{f}_{\phi_Q}(\boldsymbol{z})$ (responsible for the drift):
the i^{th} element is defined as

$$\boldsymbol{f}_{\phi_Q,i}(\boldsymbol{z}) = \beta + f_{\phi_Q}(\tilde{U}(\boldsymbol{\theta}), p_i)$$

- We want $\boldsymbol{f}_{\phi_Q}(\boldsymbol{z})$ to depend on the energy landscape:
  - Fast traversal through low-density regions
  - Better exploration in high-density regions
- But we don't want $\boldsymbol{\Gamma}_{\boldsymbol{\theta}}(\boldsymbol{z})$ to be too expensive!
  (using $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$ as input here leads to an extra term $\langle \nabla, \nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) \rangle$ in $\boldsymbol{\Gamma}_{\boldsymbol{\theta}}(\boldsymbol{z})$)

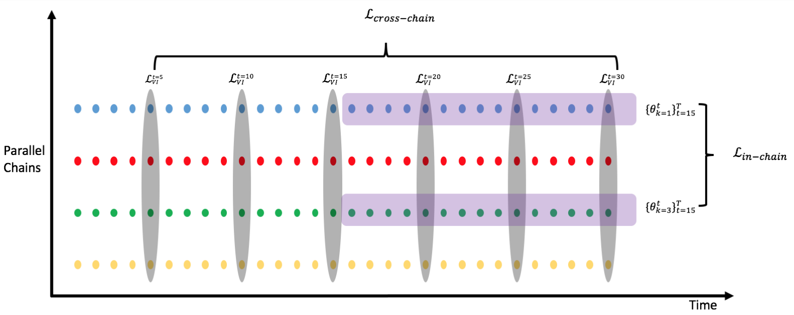
Our recipe: dynamics design

Designing $\boldsymbol{f}_{\phi_D}(\boldsymbol{z})$ (responsible for friction):
the i^{th} element is defined as

$$\boldsymbol{f}_{\phi_D,i}(\boldsymbol{z}) = f_{\phi_D}(\tilde{U}(\boldsymbol{\theta}), p_i, \partial_{\theta_i}\tilde{U}(\boldsymbol{\theta}))$$

- $\boldsymbol{\Gamma}_{\boldsymbol{p}}(\boldsymbol{z})$ only requires computing $\nabla_{\boldsymbol{p}} \boldsymbol{D}_{\boldsymbol{f}}(\boldsymbol{z})$
- ...so we can use the gradient information $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$
- prevent overshoot by "comparing" $\boldsymbol{p}$ and $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$

Our recipe: loss function design

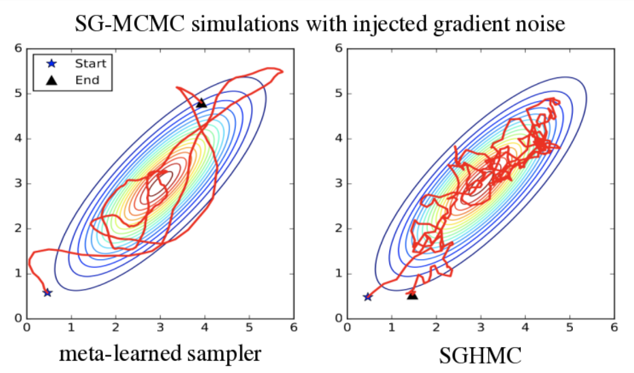


Use KL divergence $\mathrm{KL}[q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})]$ to define loss.

Define $q(\boldsymbol{\theta})$ *implicitly*: run parallel chains for several steps, then

- Cross-chain loss: at time t, collect samples across chains
- In-chain loss: for each chain, collect samples by *thinning*

A toy example



- trained on factorised Gaussians, tested on correlated Gaussians
- manually injected Gaussian noise to the gradients
  (and assume we don't know noise variance $\boldsymbol{B}(\theta)$)

Bayesian NN on MNIST

Goal: sample from BNN posterior

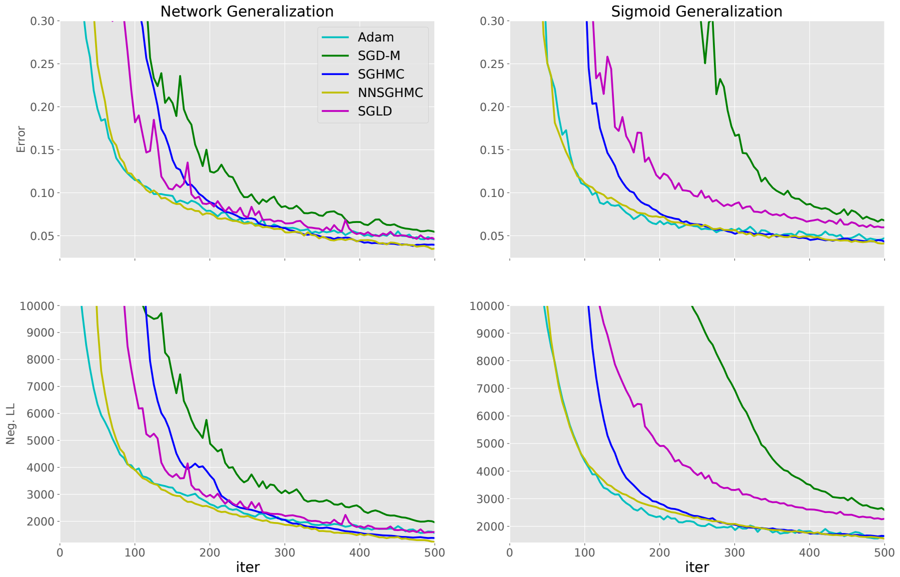
Training: meta sampler trained to sample from the posterior of a BNN (1-hidden layer, 20 hidden units, ReLU)

Three generalisation tests:

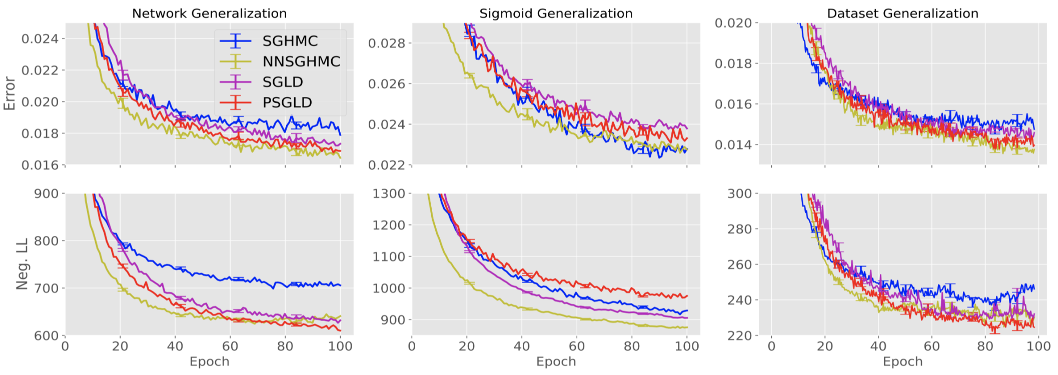
- to bigger network architecture: 2-hidden layer MLP (40 units, ReLU)
- to different activation function: 1-hidden layer MLP (20 units, Sigmoid)
- to different dataset: train on MNIST 0-4, test on MNIST 5-9

Also consider long-time horizon generalisation

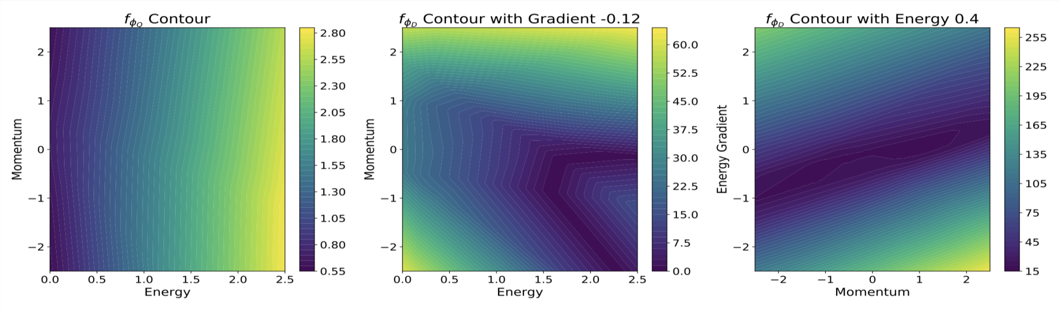
Bayesian NN on MNIST: speed improvements



Bayesian NN on MNIST: long-time generalisation



Bayesian NN on MNIST: understanding the learned sampler



$$Q_f(z) = \text{diag}[f_{\phi_Q}(z)], \quad D_f(z) = \text{diag}[\alpha f_{\phi_Q}(z) \odot f_{\phi_Q}(z) + f_{\phi_D}(z) + c]$$

- f_{ϕ_Q} (left): nearly linear wrt. energy (fast traversal, better exploration)
- f_{ϕ_D} (middle): decrease friction around high energy regions
- f_{ϕ_D} (right): increase friction when gradient & momentum "disagree" (prevent overshoot)

Summary

MCMC and meta-learning can be friends:

- MCMC can be improved using meta-learning
- Meta-learning works better when searching in a theoretically sound framework



Thank you!

Summary

MCMC and meta-learning can be friends:

- MCMC can be improved using meta-learning
- Meta-learning works better when searching in a theoretically sound framework

Future work:

- add in tempering, adaptive learning rate...
- meta-learn the Hamiltonian
- improving samplers for discrete variables



Thank you!

References

Welling and Teh (2011). Bayesian learning via stochastic gradient Langevin dynamics. ICML 2011

Chen et al. (2014). Stochastic gradient Hamiltonian Monte Carlo. ICML 2014

Li et al. (2016). Learning weight uncertainty with stochastic gradient MCMC for shape classification. CVPR 2016

Chen et al. (2016) Bridging the gap between stochastic gradient MCMC and stochastic optimization. AISTATS 2016

Salimans et al. (2015). Markov chain Monte Carlo and variational inference: Bridging the gap. ICML 2015

Song et al. (2017). A-NICE-MC: Adversarial training for MCMC. NIPS 2017

Levy et al. (2018). Generalizing Hamiltonian Monte Carlo with neural networks. ICLR 2018

Andrychowicz et al. (2016). Learning to learn by gradient descent by gradient descent. NIPS 2016

Li and Malik (2017). Learning to optimize. ICLR 2017

Wichrowska et al. (2017). Learned optimizers that scale and generalize. ICML 2017

Li and Turner (2018). Gradient estimators for implicit models. ICLR 2018

Ma et al. (2015). A complete recipe for stochastic gradient MCMC. NIPS 2015