Gradient Estimation for Implicit Models with Stein's Method

Yingzhen Li

Microsoft Research Cambridge

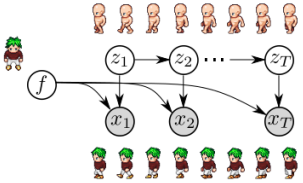
Joint work with Rich Turner, Wenbo Gong, and José Miguel Hernández-Lobato

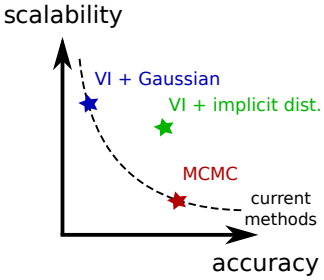



.

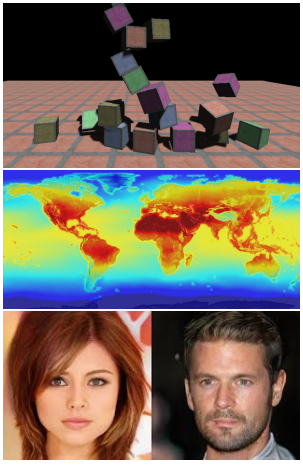
A little about my research...

Bayesian Deep Learning





Examples for implicit (generative) models



Implicit distributions:

+ easy to sample:
  $\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x}) \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\epsilon), \boldsymbol{z} = \boldsymbol{f}(\epsilon, \boldsymbol{x})$
+ super flexible

Examples for implicit (generative) models

- Bayesian inference goal: compute $\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[F(\boldsymbol{z})]$
- Approximate inference: find $q(\boldsymbol{z}|\boldsymbol{x})$ in some family $\mathcal{Q}$ such that $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$
- At inference time: Monte Carlo integration:

$$\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[F(\boldsymbol{z})] \approx \frac{1}{K}\sum_{k=1}^{K} F(\boldsymbol{z}^k), \quad \boldsymbol{z}^k \sim q(\boldsymbol{z}|\boldsymbol{x})$$

Examples for implicit (generative) models

- Bayesian inference goal: compute $\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[F(\boldsymbol{z})]$
- Approximate inference: find $q(\boldsymbol{z}|\boldsymbol{x})$ in some family $\mathcal{Q}$ such that $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$
- At inference time: Monte Carlo integration:

$$\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[F(\boldsymbol{z})] \approx \frac{1}{K}\sum_{k=1}^{K} F(\boldsymbol{z}^k), \quad \boldsymbol{z}^k \sim q(\boldsymbol{z}|\boldsymbol{x})$$

Tractability requirement: fast sampling from q

(no need for point-wise density evaluation)

Examples for implicit (generative) models

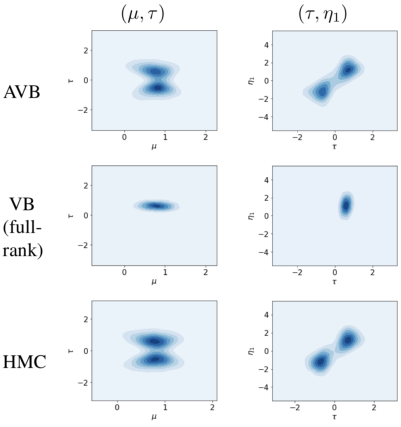


Fig. source: Mescheder et al. (2017)

Implicit distributions:

- \+ easy to sample: $\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x}) \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{\epsilon}, \boldsymbol{x})$
- \+ super flexible
- \+ better approximate posterior

Examples for implicit (generative) models

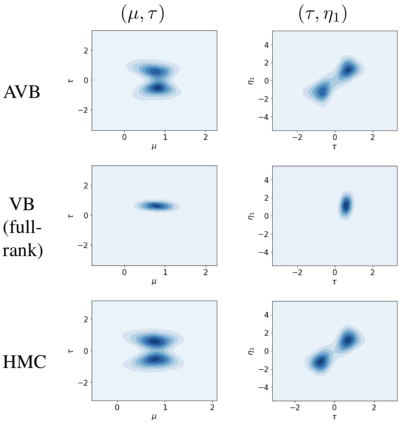


Fig. source: Mescheder et al. (2017)

Implicit distributions:

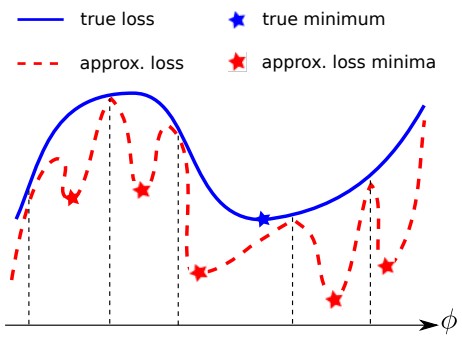
- \+ easy to sample: $\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x}) \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{\epsilon}, \boldsymbol{x})$
- \+ super flexible
- \+ better approximate posterior
- − hard to evaluate density, need some tricks for training

Loss approximation vs gradient approximation

To train the implicit generative model $p_\phi(x)$:

E.g. the generative adversarial network (GAN) method (Goodfellow et al. 2014):

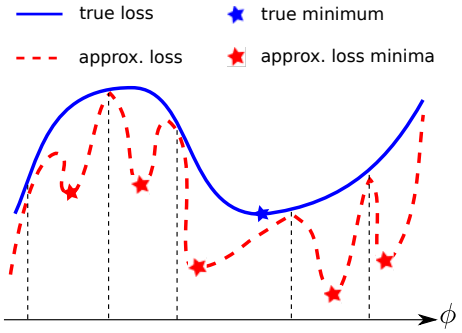
$$\min_\phi \mathrm{JS}[p_\mathcal{D}||p_\phi] = \min_\theta \max_D \mathbb{E}_{p_\mathcal{D}}[\log D(x)] + \mathbb{E}_{p_\phi}[\log(1 - D(x))]$$

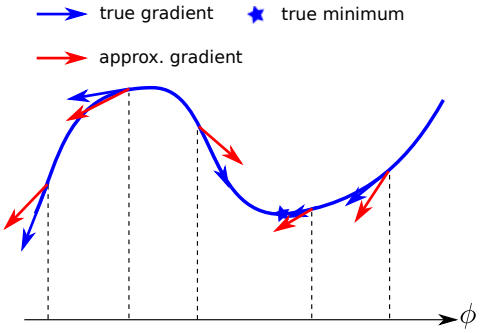


Loss approximation vs gradient approximation

Often we use gradient-based optimisation methods to train machine learning models.

... which only require evaluating the gradient, rather than the loss function!





Gradient approximation for VI

Variational inference with q distribution parameterised by ϕ:

$$\phi^* = \arg\min_{\phi} \mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})] = \arg\max_{\phi} \mathcal{L}_{\mathrm{VI}}(q_\phi)$$

$$\begin{aligned}
\mathcal{L}_{\mathrm{VI}}(q_\phi) &= \log p(\boldsymbol{x}) - \mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})] \\
&= \log p(\boldsymbol{x}) - \mathbb{E}_q\left[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_q\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_q\left[\log p(\boldsymbol{x},\boldsymbol{z})\right] + \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})]
\end{aligned}$$

$\mathcal{L}_{\mathrm{VI}}(q_\phi)$ is also called the variational lower-bound

Gradient approximation for VI

Variational lower-bound: assume $\boldsymbol{z} \sim q_{\boldsymbol{\phi}} \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x})$

$$\begin{aligned}\mathcal{L}_{\text{VI}}(q_{\boldsymbol{\phi}}) &= \mathbb{E}_q \left[\log p(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{H}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \\ &= \mathbb{E}_\pi \left[\log p(\boldsymbol{x}, \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}))\right] + \mathbb{H}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \quad // \textit{ reparam. trick}\end{aligned}$$

Gradient approximation for VI

Variational lower-bound: assume $\boldsymbol{z} \sim q_{\boldsymbol{\phi}} \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x})$

$$\begin{aligned}\mathcal{L}_{\text{VI}}(q_{\boldsymbol{\phi}}) &= \mathbb{E}_q \left[\log p(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{H}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \\ &= \mathbb{E}_\pi \left[\log p(\boldsymbol{x}, \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \boldsymbol{x}))\right] + \mathbb{H}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})] \quad // \textit{ reparam. trick}\end{aligned}$$

If you use gradient descent for optimisation, then you only need gradients!

Gradient approximation for VI

Variational lower-bound: assume $\boldsymbol{z} \sim q_\phi \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{z} = \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$

$$\begin{aligned}\mathcal{L}_{\mathrm{VI}}(q_\phi) &= \mathbb{E}_q\left[\log p(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\ &= \mathbb{E}_\pi\left[\log p(\boldsymbol{x}, \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}))\right] + \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})] \quad \textit{// reparam. trick}\end{aligned}$$

If you use gradient descent for optimisation, then you only need gradients!

The gradient of the variational lower-bound:

$$\nabla_\phi \mathcal{L}_{\mathrm{VI}}(q_\phi) = \mathbb{E}_\pi\left[\nabla_{\boldsymbol{f}} \log p(\boldsymbol{x}, \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}))^\mathsf{T} \nabla_\phi \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})\right] + \nabla_\phi \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})]$$

Gradient approximation for VI

Variational lower-bound: assume $\boldsymbol{z} \sim q_\phi \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{z} = \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$

$$\begin{aligned}\mathcal{L}_{\text{VI}}(q_\phi) &= \mathbb{E}_q\left[\log p(\boldsymbol{x}, \boldsymbol{z})\right] + \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\ &= \mathbb{E}_\pi\left[\log p(\boldsymbol{x}, \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}))\right] + \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})] \quad \textit{// reparam. trick}\end{aligned}$$

If you use gradient descent for optimisation, then you only need gradients!

The gradient of the variational lower-bound:

$$\nabla_\phi \mathcal{L}_{\text{VI}}(q_\phi) = \mathbb{E}_\pi\left[\nabla_{\boldsymbol{f}} \log p(\boldsymbol{x}, \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}))^\mathsf{T} \nabla_\phi \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})\right] + \nabla_\phi \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})]$$

The gradient of the entropy term:

$$\nabla_\phi \mathbb{H}[q_\phi(\boldsymbol{z}|\boldsymbol{x})] = -\mathbb{E}_\pi\left[\nabla_{\boldsymbol{f}} \log q(\boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})|\boldsymbol{x})^\mathsf{T} \nabla_\phi \boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})\right] - \cancel{\mathbb{E}_q\left[\nabla_\phi \log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right]}$$

this term is 0

It remains to approximate $\nabla_{\boldsymbol{z}} \log q(\boldsymbol{z}|\boldsymbol{x})$!

Stein gradient estimator

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$

Stein gradient estimator

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$

Stein's identity:
Define $\boldsymbol{h}(\boldsymbol{x})$: a (column vector) test function satisfying the boundary condition

$$\lim_{\boldsymbol{x} \to \infty} q(\boldsymbol{x}) \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}.$$

Then we can derive Stein's identity using integration by parts:

$$\mathbb{E}_q[\boldsymbol{h}(\boldsymbol{x}) \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})^{\mathsf{T}} + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})] = \boldsymbol{0}$$

Stein gradient estimator

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$

Stein's identity:
Define $\boldsymbol{h}(\boldsymbol{x})$: a (column vector) test function satisfying the boundary condition

$$\lim_{x \to \infty} q(\boldsymbol{x}) \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}.$$

Then we can derive Stein's identity using integration by parts:

$$\mathbb{E}_q[\boldsymbol{h}(\boldsymbol{x}) \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})^{\mathsf{T}} + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})] = \mathbf{0}$$

Invert Stein's identity to obtain $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$!

Stein gradient estimator (kernel based)

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$
Main idea: invert Stein's identity:

$$\mathbb{E}_q[\boldsymbol{h}(\boldsymbol{x})\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})^{\mathsf{T}} + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})] = \mathbf{0}$$

1. Monte Carlo (MC) approximation to Stein's identity:

$$\frac{1}{K}\sum_{k=1}^{K} -\boldsymbol{h}(\boldsymbol{x}^k)\nabla_{\boldsymbol{x}^k} \log q(\boldsymbol{x}^k)^{\mathsf{T}} + \text{err} = \frac{1}{K}\sum_{k=1}^{K} \nabla_{\boldsymbol{x}^k} \boldsymbol{h}(\boldsymbol{x}^k), \quad \boldsymbol{x}^k \sim q(\boldsymbol{x}^k),$$

Stein gradient estimator (kernel based)

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$

Main idea: invert Stein's identity:

$$\mathbb{E}_q[\boldsymbol{h}(\boldsymbol{x})\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})^{\mathsf{T}} + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})] = \mathbf{0}$$

1. Monte Carlo (MC) approximation to Stein's identity:

$$\frac{1}{K}\sum_{k=1}^{K} -\boldsymbol{h}(\boldsymbol{x}^k)\nabla_{\boldsymbol{x}^k} \log q(\boldsymbol{x}^k)^{\mathsf{T}} + \text{err} = \frac{1}{K}\sum_{k=1}^{K} \nabla_{\boldsymbol{x}^k} \boldsymbol{h}(\boldsymbol{x}^k), \quad \boldsymbol{x}^k \sim q(\boldsymbol{x}^k),$$

2. Rewrite the MC equations in matrix forms: denoting

$$\mathbf{H} = \left(\boldsymbol{h}(\boldsymbol{x}^1), \cdots, \boldsymbol{h}(\boldsymbol{x}^K)\right), \quad \overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}} = \frac{1}{K}\sum_{k=1}^{K} \nabla_{\boldsymbol{x}^k} \boldsymbol{h}(\boldsymbol{x}^k),$$

$$\mathbf{G} := \left(\nabla_{\boldsymbol{x}^1} \log q(\boldsymbol{x}^1), \cdots, \nabla_{\boldsymbol{x}^K} \log q(\boldsymbol{x}^K)\right)^{\mathsf{T}},$$

Then $-\frac{1}{K}\mathbf{H}\mathbf{G} + \text{err} = \overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}}$.

Stein gradient estimator (kernel based)

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$
Main idea: invert Stein's identity:

$$\mathbb{E}_q[\boldsymbol{h}(\boldsymbol{x})\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})^{\mathsf{T}} + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})] = \mathbf{0}$$

Matrix form (MC): $-\frac{1}{K}\mathbf{H}\mathbf{G} + \text{err} = \overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}}$.

3. Now solve a ridge regression problem:

$$\hat{\mathbf{G}}_V^{\text{Stein}} := \underset{\hat{\mathbf{G}} \in \mathbb{R}^{K \times d}}{\arg\min} ||\overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}} + \frac{1}{K}\mathbf{H}\hat{\mathbf{G}}||_F^2 + \frac{\eta}{K^2}||\hat{\mathbf{G}}||_F^2,$$

Stein gradient estimator (kernel based)

Goal: approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for a given distribution $q(\boldsymbol{x})$

Main idea: invert Stein's identity:

$$\mathbb{E}_q[\boldsymbol{h}(\boldsymbol{x})\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})^\top + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})] = \mathbf{0}$$

Matrix form (MC): $-\frac{1}{K}\mathbf{H}\mathbf{G} + \text{err} = \overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}}$.

3. Now solve a ridge regression problem:

$$\hat{\mathbf{G}}_V^{\text{Stein}} := \underset{\hat{\mathbf{G}} \in \mathbb{R}^{K \times d}}{\arg\min} ||\overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}} + \frac{1}{K}\mathbf{H}\hat{\mathbf{G}}||_F^2 + \frac{\eta}{K^2}||\hat{\mathbf{G}}||_F^2,$$

Analytic solution: $\hat{\mathbf{G}}_V^{\text{Stein}} = -(\mathbf{K} + \eta \boldsymbol{I})^{-1}\langle \nabla, \mathbf{K} \rangle$,

with

$\mathbf{K} := \mathbf{H}^\top \mathbf{H}, \quad \mathbf{K}_{ij} = \mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) := \boldsymbol{h}(\boldsymbol{x}^i)^\top \boldsymbol{h}(\boldsymbol{x}^j)$,

$\langle \nabla, \mathbf{K} \rangle := K\mathbf{H}^\top \overline{\nabla_{\boldsymbol{x}} \boldsymbol{h}}, \quad \langle \nabla, \mathbf{K} \rangle_{ij} = \sum_{k=1}^K \nabla_{x_j^k} \mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^k)$.

Stein gradient estimator (kernel based)

Kernelized Stein Discrepancy:

$$\mathcal{S}^2(q, \hat{q}) = \mathbb{E}_{\boldsymbol{x}, \boldsymbol{x}' \sim q} \left[\hat{\boldsymbol{g}}(\boldsymbol{x})^{\mathsf{T}} \mathcal{K}_{\boldsymbol{x}\boldsymbol{x}'} \hat{\boldsymbol{g}}(\boldsymbol{x}') + \hat{\boldsymbol{g}}(\boldsymbol{x})^{\mathsf{T}} \nabla_{\boldsymbol{x}'} \mathcal{K}_{\boldsymbol{x}\boldsymbol{x}'} + \nabla_{\boldsymbol{x}} \mathcal{K}_{\boldsymbol{x}\boldsymbol{x}'}^{\mathsf{T}} \hat{\boldsymbol{g}}(\boldsymbol{x}') + \mathsf{Tr}(\nabla_{\boldsymbol{x}, \boldsymbol{x}'} \mathcal{K}_{\boldsymbol{x}\boldsymbol{x}'}) \right],$$

$$\boldsymbol{g}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x}), \quad \hat{\boldsymbol{g}}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log \hat{q}(\boldsymbol{x}), \quad \mathcal{K}_{\boldsymbol{x}\boldsymbol{x}'} = \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}').$$

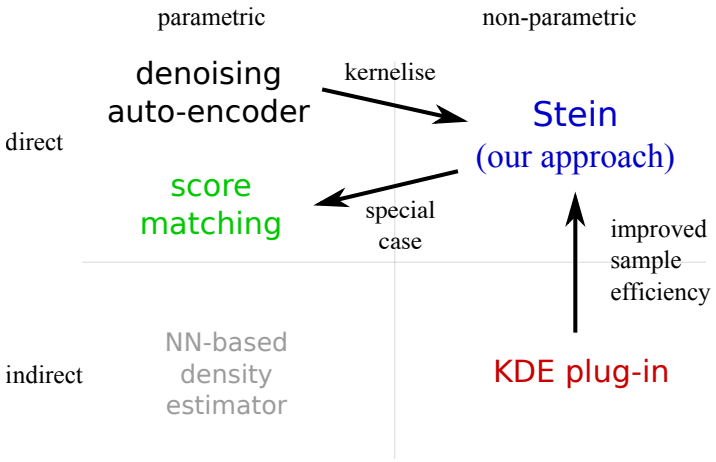
One can show that the V-statistic of KSD is

$$\mathcal{S}_V^2(q, \hat{q}) = \frac{1}{K^2} \mathsf{Tr}(\hat{\mathbf{G}}^{\mathsf{T}} \mathbf{K} \hat{\mathbf{G}} + 2 \hat{\mathbf{G}}^{\mathsf{T}} \langle \nabla, \mathbf{K} \rangle) + C$$

This means

$$\hat{\mathbf{G}}_V^{Stein} = \underset{\hat{\mathbf{G}} \in \mathbb{R}^{K \times d}}{\arg\min} \quad \mathcal{S}_V^2(q, \hat{q}) + \frac{\eta}{K^2} ||\hat{\mathbf{G}}||_F^2$$

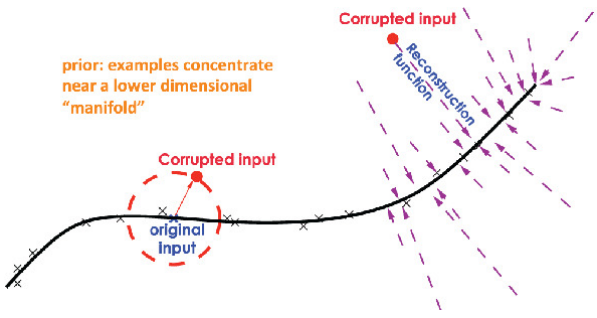
Comparisons to existing approaches



KDE plug-in estimator: Singh (1977)
Score matching estimator: Hyvärinen (2005), Sasaki et al. (2014), Strathmann et al. (2015)
Denoising auto-encoder: Vincent et al. (2008), Alain and Bengio (2014)

Comparisons to existing approaches

Compare to denoising auto-encoder (DAE):



Vincent et al. (2008), Alain and Bengio (2014)
with Wenbo Gong and José Miguel Hernández-Lobato

- DAE: for $\boldsymbol{x} \sim q(\boldsymbol{x})$, denoise $\hat{\boldsymbol{x}} = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}$ to $\boldsymbol{x}$ (by min. ℓ_2 loss, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$)
- When $\sigma \to 0$, $\text{DAE}^*(\hat{\boldsymbol{x}}) \approx \boldsymbol{x} + \sigma^2 \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$

– unstable estimate: depends on σ

\+ functional gradient in RKHS: $||\nabla\text{DAE loss}||^2_{\mathcal{H}} \propto \text{KSD}$

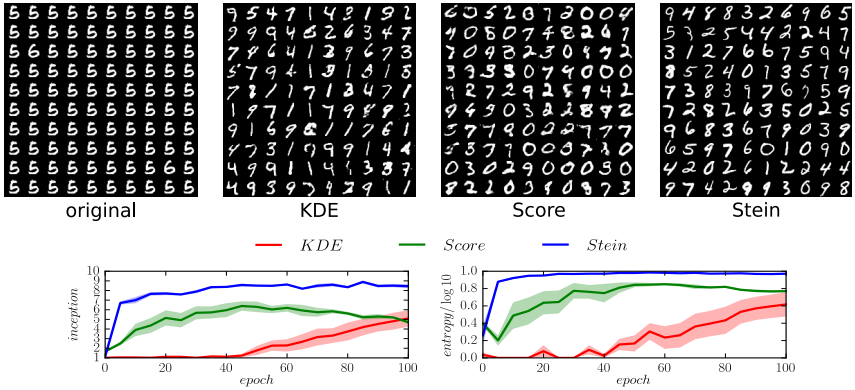
Example: entropy regularised GANs

- Addressing mode collapse: train your generator using entropy regularisation:

$$\min \ \mathcal{L}_{\text{gen}}(p_{\text{gen}}) - \mathbb{H}[p_{\text{gen}}]$$

- $\mathcal{L}_{\text{gen}}(p_{\text{gen}})$ is the generator loss of your favourite GAN method
- Again the gradient of $\mathbb{H}[p_{\text{gen}}]$ is approximated by the gradient estimators

Example: entropy regularised GANs



Significant improvement on sample diversity with the Stein approach

BEGAN: Berthelot et al. (2017)

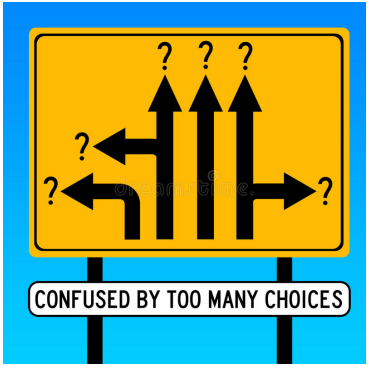
Meta learning for posterior samplers

Many existing posterior samplers in the literature...

- Which sampler should I use?
- How do I tune the hyper-parameters?

Learn a sampler 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 SG-MCMC

- Define a sampler with parameters ϕ:

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

- 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)

The complete framework: Ma et al. NIPS 2015

- Itô diffusion

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

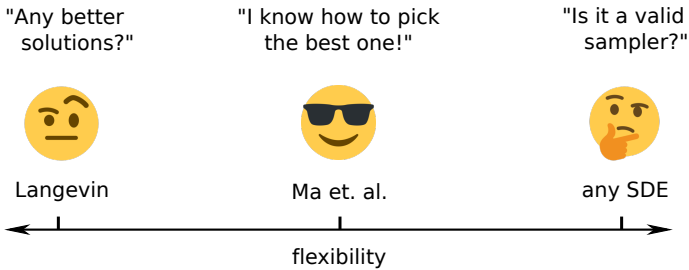
- To make sure $\pi(\boldsymbol{z}) \propto \exp[-H(\boldsymbol{z})]$ is a stationary distribution:

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

- $\boldsymbol{D}(\boldsymbol{z})$: 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 $\boldsymbol{p}$:

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

- 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:

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

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

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

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

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

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

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 $\boldsymbol{\Gamma}_{\boldsymbol{\theta}}(\boldsymbol{z})$)

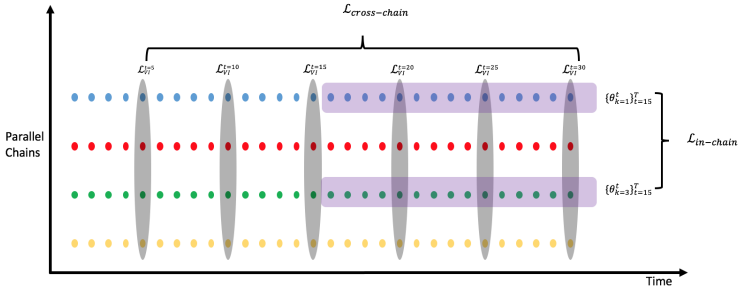
Our recipe: dynamics design

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

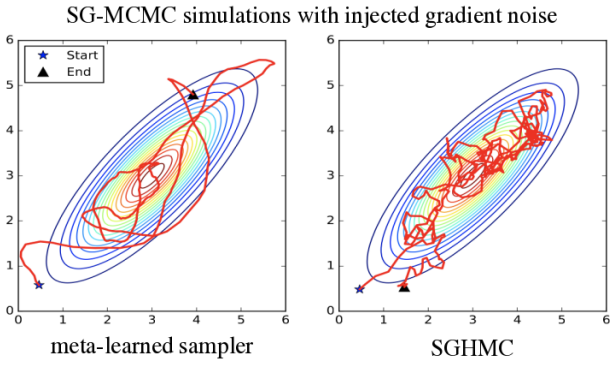


Use KL divergence $\mathrm{KL}[q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})]$ to define loss.
Define $q(\boldsymbol{\theta})$ *implicitly*: run parallel chains for several steps, then

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

Gradient of KL approximated by Stein gradient estimator!

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 $\boldsymbol{B}(\boldsymbol{\theta})$)

Bayesian NN on MNIST

Goal: sample from BNN posterior

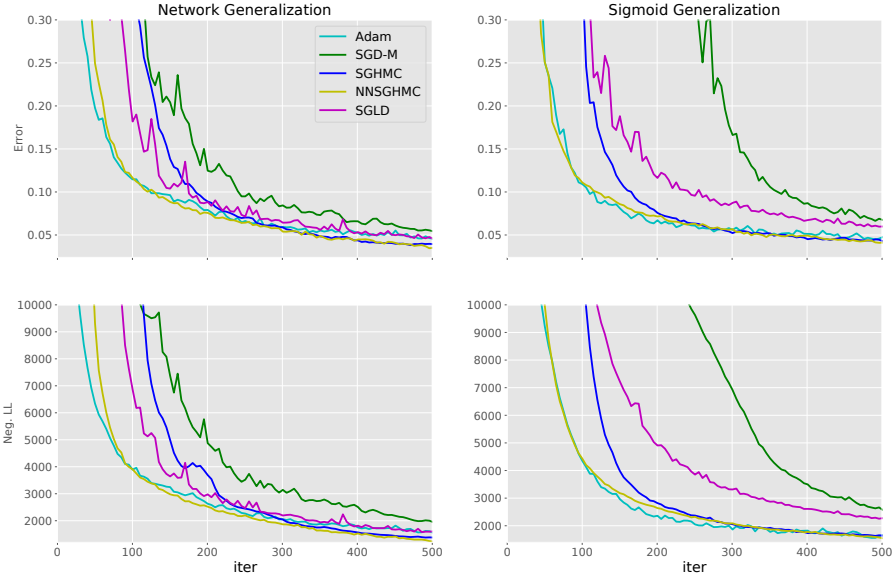
Training: meta sampler trained to sample from the posterior of a BNN (1-hidden layer, 20 hidden units, ReLU)

Three generalisation tests:

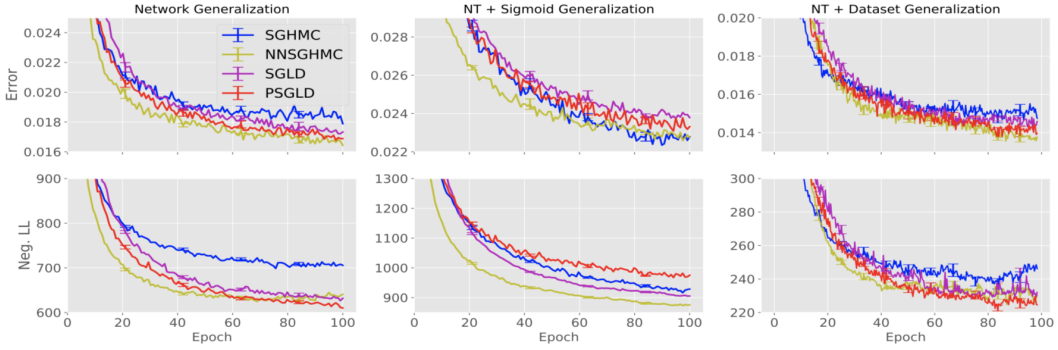
- to bigger network architecture: 2-hidden layer MLP (40 units, ReLU)
- to different activation function: 1-hidden layer MLP (20 units, Sigmoid)
- to different dataset: train on MNIST 0-4, test on MNIST 5-9

Also consider long-time horizon generalisation

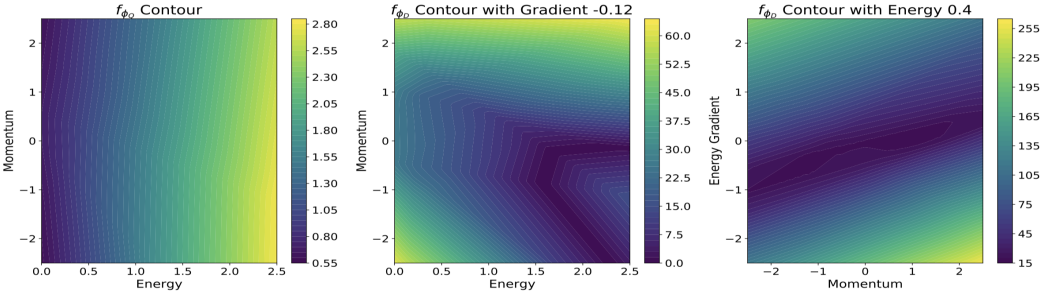
Bayesian NN on MNIST: speed improvements



Bayesian NN on MNIST: long-time generalisation



Bayesian NN on MNIST: understanding the learned sampler

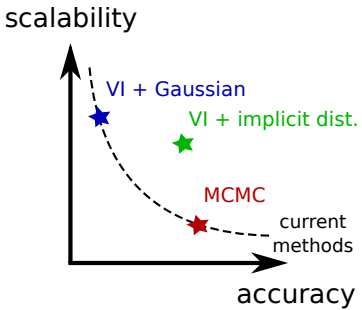


$$\boldsymbol{Q}_f(\boldsymbol{z}) = \text{diag}[\boldsymbol{f}_{\phi_Q}(\boldsymbol{z})], \quad \boldsymbol{D}_f(\boldsymbol{z}) = \text{diag}[\alpha \boldsymbol{f}_{\phi_Q}(\boldsymbol{z}) \odot \boldsymbol{f}_{\phi_Q}(\boldsymbol{z}) + \boldsymbol{f}_{\phi_D}(\boldsymbol{z}) + c]$$

- 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)

Stein gradient estimator: summary

- We derived a non-parametric gradient estimator
- We used the gradient estimator for entropy-regularised GANs
- We explored meta-learning for approximate inference



Related work by colleagues

Doucet et al. (2013). Derivative-Free Estimation of the Score Vector and Observed Information Matrix with Application to State-Space Models. ArXiv 1304.5768

Shi et al. (2018). A Spectral Approach to Gradient Estimation for Implicit Distributions. ICML 2018

Song et al. (2019). Sliced Score Matching: A Scalable Approach to Density and Score Estimation. UAI 2019

Andrew Duncan's talk. Minimum Stein discrepancy estimators. This workshop

Spectral Estimators for Gradient Fields of Log-Densities. This workshop

Thank you!

Y. Li and R.E. Turner. Gradient Estimators for Implicit Models. ICLR 2018

W. Gong*, Y. Li* and J.M. Hernández-Lobato. Meta learning for stochastic gradient MCMC. ICLR 2019.