Wild Approximate Inference: Why and How

Yingzhen Li

Machine Learning Group, University of Cambridge
Joint work with Rich Turner (Cambridge) and Qiang Liu (Dartmouth → UT Austin)

Tractability in approximate inference

- Bayesian inference: integrating out the unobserved variables in your model
  - latent variables in a generative model
  - weight matrices in a Bayesian neural network
- "We do approximate inference because exact inference is intractable."

Tractability in approximate inference

- Bayesian inference: integrating out the unobserved variables in your model
  - latent variables in a generative model
  - weight matrices in a Bayesian neural network
- "We do approximate inference because exact inference is intractable."

What does "tractability" really mean for an *approximate* inference algorithm?

Tractability in approximate inference

- In Bayesian modelling, we specify the model:

$$\text{prior:} \quad p(\boldsymbol{z}), \quad \text{likelihood:} \quad p(\boldsymbol{x}|\boldsymbol{z})$$

- We want to compute the exact posterior

$$\text{exact posterior:} \quad p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})}{p(\boldsymbol{x})}$$

- Inference: given some function $F(\boldsymbol{z})$ in interest, want $\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[F(\boldsymbol{z})]$
  - predictive distribution $p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{z}|\mathcal{D})}[p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})]$
  - evaluate posterior $p(\boldsymbol{z} \in A|\boldsymbol{x}) = \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}[\delta_A]$
- In this talk we assume $F(\boldsymbol{z})$ is cheap to compute given $\boldsymbol{z}$

Tractability in approximate inference

- In most of the time we cannot compute $p(\boldsymbol{z}|\boldsymbol{x})$ efficiently
- Approximate inference: find $q(\boldsymbol{z}|\boldsymbol{x})$ in some family $\mathcal{Q}$ such that $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$
- and in inference time, approximate

$$\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}\left[F(\boldsymbol{z})\right] \approx \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[F(\boldsymbol{z})\right]$$

Tractability in approximate inference

- In most of the time we cannot compute $p(\boldsymbol{z}|\boldsymbol{x})$ efficiently
- Approximate inference: find $q(\boldsymbol{z}|\boldsymbol{x})$ in some family $\mathcal{Q}$ such that $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$
- and in inference time, approximate

$$\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}\left[F(\boldsymbol{z})\right] \approx \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[F(\boldsymbol{z})\right]$$

Tractability: fast computation of the RHS term

Tractability in approximate inference

Optimisation-based methods, e.g. variational inference:

- Optimise a (usually parametric) q distribution to approximate the exact posterior

$$q^*(\boldsymbol{z}|\boldsymbol{x}) = \arg\min_{q\in\mathcal{Q}} \mathrm{KL}[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})] = \arg\max_{q\in\mathcal{Q}} \mathbb{E}_q[\log p(\boldsymbol{z},\boldsymbol{x})] + \mathbb{H}[q(\boldsymbol{z}|\boldsymbol{x})]$$

- When q or p is complicated, usually approximate the expectation by Monte Carlo

$$\mathcal{L}_{\mathrm{VI}}^{\mathrm{MC}}(q) = \frac{1}{K}\sum_{k=1}^{K} \log p(\boldsymbol{x},\boldsymbol{z}^k) - \log q(\boldsymbol{z}^k|\boldsymbol{x}), \quad \boldsymbol{z}^k \sim q(\boldsymbol{z}|\boldsymbol{x})$$

- With Monte Carlo approximation methods, inference is done by

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

Tractability in approximate inference

Optimisation-based methods, e.g. variational inference:

- Optimise a (usually parametric) q distribution to approximate the exact posterior

$$q^*(\boldsymbol{z}|\boldsymbol{x}) = \arg\min_{q\in\mathcal{Q}} \mathrm{KL}[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})] = \arg\max_{q\in\mathcal{Q}} \mathbb{E}_q[\log p(\boldsymbol{z},\boldsymbol{x})] + \mathbb{H}[q(\boldsymbol{z}|\boldsymbol{x})]$$

- When q or p is complicated, usually approximate the expectation by Monte Carlo

$$\mathcal{L}_{\mathrm{VI}}^{\mathrm{MC}}(q) = \frac{1}{K}\sum_{k=1}^{K} \log p(\boldsymbol{x},\boldsymbol{z}^k) - \log q(\boldsymbol{z}^k|\boldsymbol{x}), \quad \boldsymbol{z}^k \sim q(\boldsymbol{z}|\boldsymbol{x})$$

- With Monte Carlo approximation methods, inference is done by

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

Tractability requirement: fast sampling

Tractability in approximate inference

Optimisation-based methods, e.g. variational inference:

- Optimise a (usually parametric) q distribution to approximate the exact posterior

$$q^*(\boldsymbol{z}|\boldsymbol{x}) = \arg\min_{q\in\mathcal{Q}} \mathrm{KL}[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})] = \arg\max_{q\in\mathcal{Q}} \mathbb{E}_q[\log p(\boldsymbol{z},\boldsymbol{x})] + \mathbb{H}[q(\boldsymbol{z}|\boldsymbol{x})]$$

- When q or p is complicated, usually approximate the expectation by Monte Carlo

$$\mathcal{L}_{\mathrm{VI}}^{\mathrm{MC}}(q) = \frac{1}{K}\sum_{k=1}^{K} \log p(\boldsymbol{x},\boldsymbol{z}^k) - \log q(\boldsymbol{z}^k|\boldsymbol{x}), \quad \boldsymbol{z}^k \sim q(\boldsymbol{z}|\boldsymbol{x})$$

- With Monte Carlo approximation methods, inference is done by

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

Tractability requirement: fast sampling
and fast density (ratio) evaluation (only for optimisation!)

Is it necessary to evaluate the (approximate) posterior density?

Three reasons why I think it is not necessary:

- if yes, might restrict the approximation accuracy
- if yes, visualising distributions in high dimensions is still an open research question;
- most importantly, MC integration do not require density evaluation!

Wild approximate inference: Why

Research for wild approximate inference:

Can we design efficient approximate inference algorithms that enables fast inference, without adding more constraints to q?

Wild approximate inference: Why

Research for wild approximate inference:

Can we design efficient approximate inference algorithms that enables fast inference, without adding more constraints to q?

Why this research problem is interesting:

- Having the best from both MCMC and VI
- Allowing exciting new applications

Wild approximate inference: Why

VI

- Need fast density (ratio) evaluation
- Less accurate
- Fast inference
- Easy to amortise (memory efficient)

MCMC

- Just need to do sampling
- Very accurate
- Need large T thus slow (MCMC)
- Not re-usable when p is updated

We want to have the best from both worlds!

Wild approximate inference: Why

Meta learning for approximate inference:

- Currently we handcraft the MCMC algorithms and/or approximate inference optimisation objectives
- Can we learn them?

Wild approximate inference: How

We have seen/described/developed 4 categories of approaches:

- Variational lower-bound approximation (based on density ratio estimation)
  Li and Liu (2016), Karaletsos (2016); Huszár (2017); Tran et al. (2017); Mescheder et al. (2017); Shi et al. (2017)
- Alternative objectives other than minimising KL
  Ranganath et al. (2016); Liu and Feng (2016)
- Gradient approximations
  Huszár (2017); Li and Turner (2017)
- Amortising deterministic/stochastic dynamics
  Wang and Liu (2016); Li, Turner and Liu (2017); Chen et al. (2017); Pu et al. (2017)

See Chapter 5 of my draft thesis for a discussion
Also see Liu and Lee (2017), Titsias (2017)

Variational lower-bound approximation via density ratio estimation

Recall the variational lower-bound

$$\mathcal{L}_{\mathrm{VI}}(q) = \mathbb{E}_q \left[\log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})} \right]$$

- let's assume $p(\boldsymbol{x}|\boldsymbol{z})$ is tractable
- then it only remains to estimate the density ratio $\frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})}$
- classifier-based GAN (Goodfellow et al. 2014): the optimal discriminator satisfies

$$D^*(\boldsymbol{z}, \boldsymbol{x}) = \left[1 + \exp \left[- \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})} \right] \right]^{-1}$$

Therefore,

$$\mathcal{L}_{\mathrm{VI}}(q) \approx \mathbb{E}_q \left[\log p(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{D(\boldsymbol{z}, \boldsymbol{x})}{1 - D(\boldsymbol{z}, \boldsymbol{x})} \right].$$

In practice need to do "adaptive contrast" (Mescheder et al. 2017)
Can extend this to IS/SMC (NIPS workshop)

Alternative objectives other than minimising KL

Minimising Stein's discrepancy

$$\mathcal{S}^2(p(\boldsymbol{z}|\boldsymbol{x}), q(\boldsymbol{z}|\boldsymbol{x})) = \left(\sup_{h \in \mathcal{H}} \mathbb{E}_q \left[\nabla_z \log p(\boldsymbol{z}, \boldsymbol{x}) h(\boldsymbol{z}) + \langle \nabla, h(\boldsymbol{z}) \rangle \right] \right)^2$$

- this is nice: only need samples from q and evaluate the joint distribution $p(\boldsymbol{z}, \boldsymbol{x})$
- OPVI (Ranganath et al. 2016): $\mathcal{H}$ is a set of neural network functions
- Wild VI via KSD minimisation (Liu and Feng 2016): $\mathcal{H}$ is the unit ball in an RKHS

Gradient Approximations

Alternative idea: approximate the gradient

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

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

During optimisation we only care about the gradients!

The gradient of the variational lower-bound:

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

Alternative idea: approximate the gradient

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

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

During optimisation we only care about the gradients!

The gradient of the variational lower-bound:

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

The gradient of the entropy term:

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

this term is 0

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

(in a cheap way)

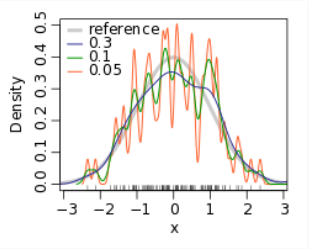
Gradient estimators (kernel based)

KDE plug-in gradient estimator for $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^d$:

- first approximate $q(\boldsymbol{x})$ using kernel density estimator $\hat{q}(\boldsymbol{x})$:

$$\hat{q}(\boldsymbol{x}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^k), \quad \boldsymbol{x}^k \sim q(\boldsymbol{x})$$

- then approximate $\nabla_{\boldsymbol{x}} \log q(\boldsymbol{x}) \approx \nabla_{\boldsymbol{x}} \log \hat{q}(\boldsymbol{x})$



Gradient estimators (kernel based)

Score matching gradient estimators: find $\hat{\boldsymbol{g}}(\boldsymbol{x})$ to minimise the ℓ_2 error

$$\mathcal{F}(\hat{\boldsymbol{g}}) := \mathbb{E}_q \left[||\hat{\boldsymbol{g}}(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log q(\boldsymbol{x})||_2^2 \right]$$

Using integration by parts we can rewrite: (Hyvärinen 2005)

$$\mathcal{F}(\hat{\boldsymbol{g}}) = \mathbb{E}_q \left[||\hat{\boldsymbol{g}}(\boldsymbol{x})||_2^2 + 2 \sum_{j=1}^{d} \nabla_{x_j} \hat{g}_j(\boldsymbol{x}) \right] + C$$

Sasaki et al. (2014) and Strathmann et al. (2015): define

$$\hat{\boldsymbol{g}}(\boldsymbol{x}) = \sum_{k=1}^{K} a_k \nabla_{\boldsymbol{x}} \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^k), \quad \boldsymbol{x}^k \sim q(\boldsymbol{x})$$

and find the best $\boldsymbol{a} = (a_1, ..., a_K)$ by minimising the ℓ_2 error.

Stein gradient estimator (kernel based)

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

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

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

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

Stein gradient estimator (kernel based)

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

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

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)

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. 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)

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. 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)

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

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)

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

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

Gradient estimators: comparisons

Comparing KDE plugin gradient estimator and Stein gradient estimator:
for translation invariant kernels:

$$\hat{\mathbf{G}}^{\text{KDE}} = -\text{diag}(\mathbf{K1})^{-1}\langle\nabla, \mathbf{K}\rangle$$

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

When approximating $\nabla_{\boldsymbol{x}^k} \log q(\boldsymbol{x}^k)$:

- KDE: only use $\mathcal{K}(\boldsymbol{x}^j, \boldsymbol{x}^k)$
- Stein: use all $\mathcal{K}(\boldsymbol{x}^j, \boldsymbol{x}^i)$ even for those $i \neq k$

more sample efficient!

Gradient estimators: comparisons

Compare to the score matching gradient estimator:

Score matching	Stein
• Min. expected ℓ_2 error (a stronger divergence)	• Min. KSD (a weaker divergence)
• Parametric approx. (introduce approx. error)	• Non-parametric approx. (no approx. error)
• Repeated derivations for different kernels	• Ubiquitous solution for any kernel

Complexity: both need matrix inversion, $\mathcal{O}(K^3 + K^2 d)$

KSD: Kernelised Stein discrepancy (Liu et al. 2016; Chwialkowski et al. 2016)

Adding predictive power

Use Stein gradient estimator at unseen locations $\boldsymbol{y}$:

- non-parametric predictive model:
  - compute $\hat{\mathbf{G}}_V^{\text{Stein}}$ on $\{\boldsymbol{y}\} \cup \{\boldsymbol{x}^k\}_{k=1}^K$
  - retrieve the row corresponding to the approximation of $\log q(\boldsymbol{y})$
- parametric predictive model:
  - define $\hat{\boldsymbol{g}}(\boldsymbol{x}) = \sum_{k=1}^K a_k \nabla_{\boldsymbol{x}} \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^k), \quad \boldsymbol{x}^k \sim q(\boldsymbol{x})$
  - find the best $\boldsymbol{a} = (a_1, ..., a_K)$ by minimising KSD
  - compute $\hat{\boldsymbol{g}}(\boldsymbol{y})$ using the fitted $\boldsymbol{a}$ and the stored samples $\{\boldsymbol{x}^k\}_{k=1}^K$

Example: meta-learning for approximate inference

- learn an approx. posterior sampler for NN weights

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \zeta \Delta_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \nabla_t) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \nabla_t) \odot \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I}),$$

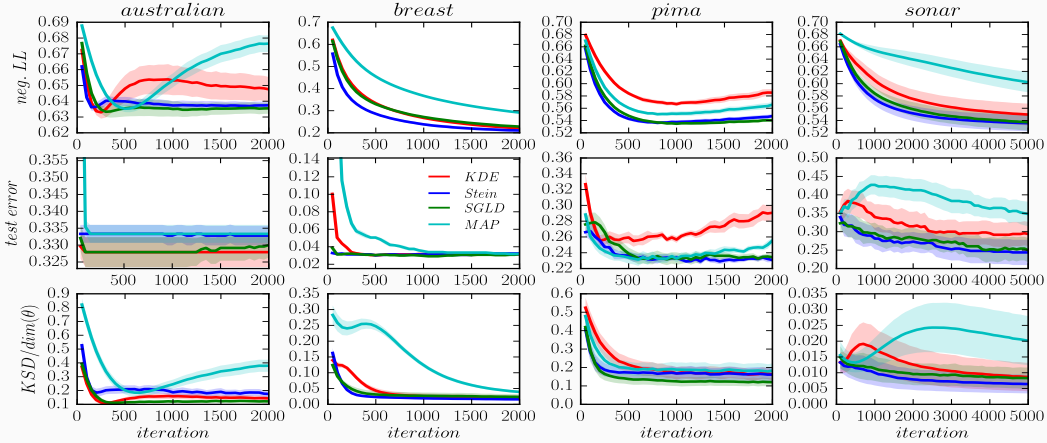
$$\nabla_t = \nabla_{\boldsymbol{\theta}_t} \left[\frac{N}{M} \sum_{m=1}^{M} \log p(y_m | \boldsymbol{x}_m, \boldsymbol{\theta}_t) + \log p_0(\boldsymbol{\theta}_t) \right].$$

- coordinates of $\Delta_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \nabla_t)$ and $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \nabla_t)$ are parameterised by MLPs
- training objective: an MC approximation of

$$\sum_t \mathcal{L}_{\mathrm{VI}}(q_t), \quad q_t \text{ is the marginal distribution of } \boldsymbol{\theta}_t$$

- see whether it generalises to diff. architectures and datasets:
  - train: 1-hidden-layer BNN with 20 hidden units + ReLU, on *crabs* dataset
  - test: 1-hidden-layer BNN with 50 hidden units + sigmoid, on other datasets

Example: meta-learning for approximate inference



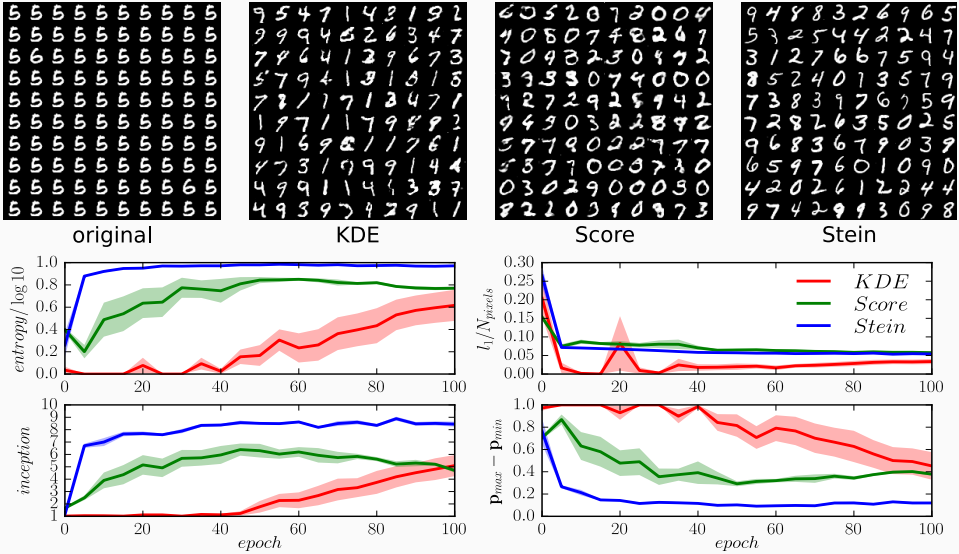
Example: entropy regularised GANs

- 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
- We pick BEGAN because it has severe mode collapse problem

Example: entropy regularised GANs



Example: training implicit generative models

- training a generative model $q_{\boldsymbol{\theta}}(\boldsymbol{x})$ by minimising

$$\min_{\boldsymbol{\theta}} \mathrm{KL}[q_{\boldsymbol{\theta}}(\boldsymbol{x})||p_{\mathcal{D}}(\boldsymbol{x})]$$

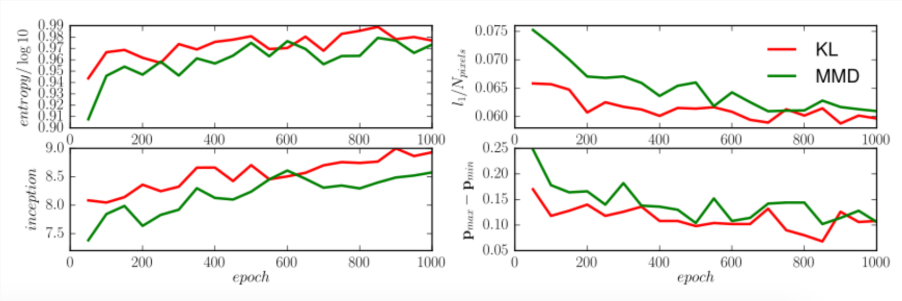
- the gradient of KL: say $\boldsymbol{x} \sim q \Leftrightarrow \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}), \boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon})$

$$\nabla_{\boldsymbol{\theta}} \mathrm{KL} = \mathbb{E}_{\pi}[(\nabla_{\boldsymbol{x}} \log q_{\boldsymbol{\theta}}(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log p_{\mathcal{D}}(\boldsymbol{x}))|_{\boldsymbol{x}=\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{\epsilon})} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}]$$

- Use the gradient estimator for both $\nabla_{\boldsymbol{x}} \log q_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $\nabla_{\boldsymbol{x}} \log p_{\mathcal{D}}(\boldsymbol{x})$!

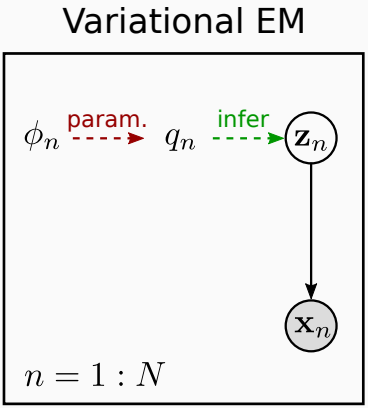
Example: training implicit generative models

(a) test data samples (b) minimising MMD V-statistic (c) approximate KL SGD



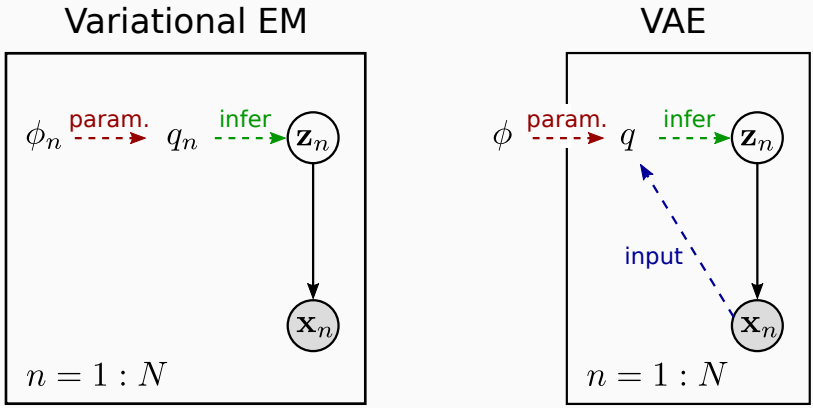
Amortised MCMC

Recap: from Variational EM to VAE



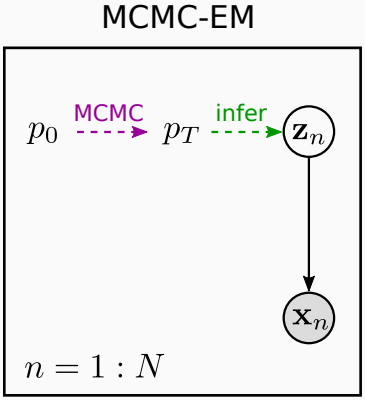
(For simplicity we omit the model parameter $\boldsymbol{\theta}$ but it would be trained by approx. MLE)

Recap: from Variational EM to VAE



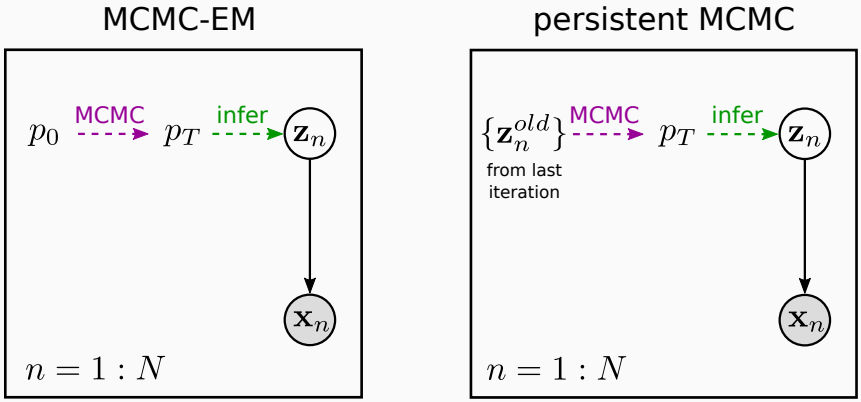
Amortised inference: memory efficient & no need to run VI optimisation in test time!

New: from MCMC-EM to amortised MCMC



For every $\boldsymbol{x}_n$, need to simulate MCMC for $T >> 0$ steps (slow!)

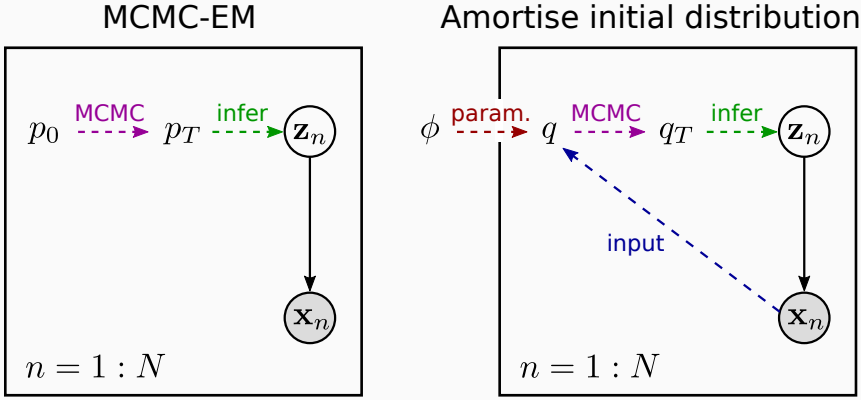
New: from MCMC-EM to amortised MCMC



Need to store all samples from the previous iteration, memory cost $\mathcal{O}(NKD)$.

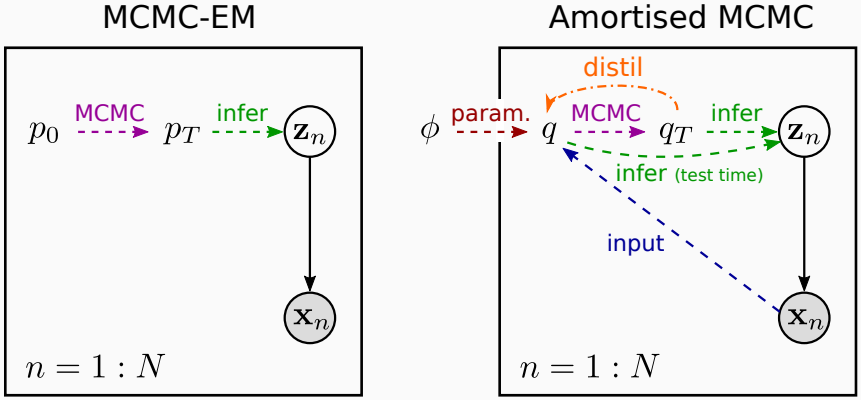
For a new datapoint, still need to run MCMC with $T >> 0$ starting from p_0 (slow!)

New: from MCMC-EM to amortised MCMC



This method essentially cares about q_T only, so no need for $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$.
In test time still need to run MCMC to obtain samples from q_T (slow!)

New: from MCMC-EM to amortised MCMC

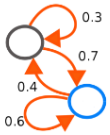


Distillation happens at the same time during training (thus also improving q_T),
and now $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$ – no need to run MCMC in test time!

Understanding “distillation during training”

I've generated some samples. Do they approximate the target distribution well?

You can do better. Returned you some MCMC samples improved upon yours.



OK thanks I'll try to generate these improved samples next time.

Understanding "distillation during training"

Sampling-based methods, e.g. Markov chain Monte Carlo:

- Define a transition kernel $\mathcal{K}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ that leaves $p(\boldsymbol{z}|\boldsymbol{x})$ invariant
- Pick an initial distribution $q_0(\boldsymbol{z}_0|\boldsymbol{x})$, and run T-step transitions

$$\boldsymbol{z}_T^k \sim q_T(\boldsymbol{z}_T|\boldsymbol{x}) = \int q_0(\boldsymbol{z}_0|\boldsymbol{x}) \prod_{t=1}^{T} \mathcal{K}(\boldsymbol{z}_T|\boldsymbol{z}_{t-1}) d\boldsymbol{z}_{0:T-1}, \quad k = 1, 2, ..., K$$

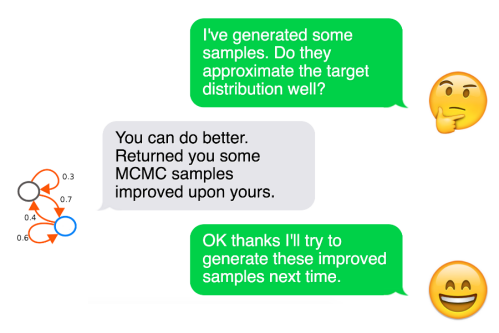
- We rely on two assumptions:
  - $\mathcal{K}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ has a unique stationary distribution $p(\boldsymbol{z}|\boldsymbol{x})$
  - when $T \to +\infty$ the Markov chain converges to the stationary
- Then for all $T > 0$, $q_0(\boldsymbol{z}|\boldsymbol{x}) = q_T(\boldsymbol{z}|\boldsymbol{x})$ iff. $q_0(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z}|\boldsymbol{x})$

Summarising amortised MCMC in one slide!

Three main ingredients of the framework:

- "Student": an architecture of $q(\boldsymbol{z}|\boldsymbol{x})$
- "Teacher": a transition kernel $\mathcal{K}$ invariant to $p(\boldsymbol{z}|\boldsymbol{x})$, which produces $q_T(\boldsymbol{z}|\boldsymbol{x})$
- "Feedback": an update rule to match $q(\boldsymbol{z}|\boldsymbol{x})$ to $q_T(\boldsymbol{z}|\boldsymbol{x})$ (say divergence minimisation)

After looping, $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$, and use $q(\boldsymbol{z}|\boldsymbol{x})$ for inference!



Connections to deep RL

- TD-learning and Q-learning:
  also tries to update the current V- or Q-value to using T-step roll-out
- Can borrow RL ideas for convergence proof:
  TD/Q-learning can show convergence when using linear approximations
- using NNs as $q_0 \leftrightarrow$ using DQNs
- Can borrow deep RL tricks to design distillation rules!

Amortised MCMC: distillation rules

We want q to be implicit!
(i.e. can sample from q but cannot evaluate density)

idea: match samples $\{z_0^k\} \sim q$ to samples $\{z_T^k\} \sim q_T$!

- We tested the original GAN idea [1]
- In general, any GAN-like technique is applicable!

[1] Goodfellow et al. Generative Adversarial Networks. NIPS 2014.

Training generative models with implicit q distributions

Generative model: a small convolutional decoder

- VAE training:
  - Gaussian encoder: a symmetric flip of the decoder
- Amortised MCMC (AMC) training:
  - CNN-G encoder: $\boldsymbol{z} = MLP([CNN(\boldsymbol{x}), \epsilon])$, $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  - CNN-B encoder: $\boldsymbol{z} = MLP([CNN(\boldsymbol{x}) \odot \epsilon])$, $\epsilon \sim Bern(0.5)$
  - MCMC: Langevin Dynamics w/out rejection

data

Gaussian encoder + VAE

CNN-G + AMC

CNN-B + AMC

An open question for test-LL evaluation

Table 1: Average Test LL and effective sample size (ESS). η as the stepsize for Langevin dynamics.

Encoder	**Method**	IW-LL	IW-ESS	HAIS-LL	HAIS-ESS
Gaussian	VAE	**-81.31**	**104.11**	-80.64	91.59
	MCMC-VI, $T = 5$, $\eta = 0.2$	-90.06	110.58	-89.79	85.63
	AMC, $T = 5$, $\eta = 0.2$	-90.71	49.02	-89.64	87.93
CNN-G	AMC, $T = 5$, $\eta = 0.2$	-90.84	31.60	-89.35	87.49
	AMC, $T = 50$, $\eta = 0.02$	-83.30	6.84	**-78.23**	**77.78**
	AVB	-94.97	11.30	-85.92	57.21
CNN-B	AMC, $T = 5$, $\eta = 0.2$	-90.75	34.17	-89.42	88.10
	AMC, $T = 50$, $\eta = 0.02$	-83.62	8.88	-80.03	80.71
	AVB	-89.47	8.98	-82.66	76.90
N/A	persistent MCMC, $T = 50$, $\eta = 0.02$	-84.43	9.14	-78.88	77.29

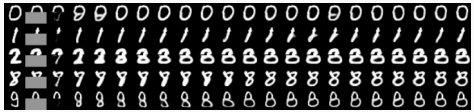
- HAIS seems to be more reliable ($K = 100$), compared to importance sampling (IS, $K = 5000$)
- The best case (CNN-G) is better than persistent MCMC

Missing data imputation

- Given an image $\boldsymbol{x} = [\boldsymbol{x}_o, \boldsymbol{x}_m]$ with missing values, repeat the following for T steps:
  - sample $\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x}_o, \boldsymbol{x}_m)$
  - sample $\boldsymbol{x}^* \sim p(\boldsymbol{x}|\boldsymbol{z})$ and set $\boldsymbol{x}_m \leftarrow \boldsymbol{x}^*_m$

Table 2: Label entropy on nearest neighbours. The l_1 distance is divided by the number of pixels.

Dataset	VAE	CNN-G	CNN-B
Entropy	0.411±0.0389	0.701±0.0476	0.933±0.0491
l_1-norm	0.061±0.0002	0.059±0.0001	0.064±0.0002



Gaussian encoder + VAE

CNN-G + AMC

CNN-B + AMC

Amortised MCMC: distillation rules (cont.)

We want q to be implicit!

If you don't like discriminators/minimax optimisation: do moment matching!

- Simply match the first M moments (not minimising a divergence)
  - How to choose M and how to reweigh the importance of them?
- We propose energy matching:

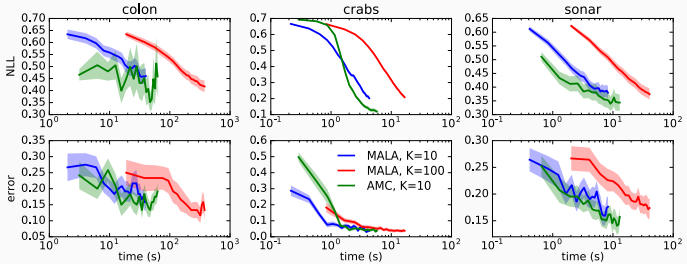
$$\min_{\phi} \left| \mathbb{E}_{q_T} \left[\log p(\boldsymbol{x}, \boldsymbol{z}_T) \right] - \mathbb{E}_{q_\phi} \left[\log p(\boldsymbol{x}, \boldsymbol{z}) \right] \right|^2$$

  - Let the log-joint distribution tell you which moments you want to match
  - Related to Contrastive Divergence (CD-T), works well with small T

Bayesian neural network classification

Table 3: BNN classification experiment results. MALA = Langevin dynamics with rejection

	Average Test Log-likelihood			Average Test Error		
Dataset	**VI+Gaussian**	**AMC**	**MALA**	**VI+Gaussian**	**AMC**	**MALA**
australian	**-0.633±0.008**	-0.666±0.015	-0.636±0.010	**0.315±0.014**	0.360±0.011	0.344±0.013
breast	-0.096±0.010	**-0.091±0.008**	-0.094±0.010	**0.029±0.003**	0.030±0.004	0.037±0.004
colon	-0.799±0.246	-0.491±0.104	**-0.420±0.027**	**0.125±0.023**	0.167±0.031	0.167±0.026
crabs	-0.221±0.012	**-0.115±0.011**	-0.179±0.010	0.070±0.013	0.040±0.010	**0.035±0.010**
ionosphere	-0.241±0.019	-0.230±0.031	**-0.179±0.013**	0.099±0.011	0.077±0.013	**0.064±0.010**
pima	-0.503±0.010	-0.506±0.013	**-0.498±0.012**	0.262±0.008	**0.245±0.008**	0.247±0.008
sonar	-0.389±0.025	**-0.347±0.030**	-0.366±0.021	0.179±0.014	**0.150±0.016**	0.171±0.020



Summary

What we covered today:

- Is density evaluation really necessary for inference tasks?
- Fitting implicit approx. posterior:
  - objective approximation via density ratio estimation
  - minimising Stein's discrepancy
  - gradient approximations
  - amortise MCMC/SVGD
- Designing implicit posterior approximations: big challenge

Thank you!