Gradient Estimators for Implicit Models

Yingzhen Li

University of Cambridge

Joint work with Rich Turner (Cambridge)
Special thanks to Qiang Liu (Dartmouth $\rightarrow$ UT Austin)

Based on Chapter 5,6 of my (draft) thesis
arXiv 1705.07107

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

- Bayes Rule

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})}{p(\boldsymbol{x})}$$

- 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 $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$
- In inference time: Monte Carlo approximation:

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

Tractability in approximate inference

- 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 $q(\boldsymbol{z}|\boldsymbol{x}) \approx p(\boldsymbol{z}|\boldsymbol{x})$
- In inference time: Monte Carlo approximation:

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

Tractability requirement: fast sampling from q

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
and fast density (gradient) 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 does not require density evaluation

Wild approximate inference: Why

Can we design efficient approximate inference algorithms that enables fast inference, without adding more requirements 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
- Faster inference
- Easy to amortise (memory efficient)

MCMC

- Just need to do sampling
- Very accurate
- Need large T thus slower
- 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 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)
- Amortising deterministic/stochastic dynamics
  Wang and Liu (2016); Li, Turner and Liu (2017); Chen et al. (2017); Pu et al. (2017)
- Gradient approximations (this talk)
  Huszár (2017); Li and Turner (2017)

Also see Titsias (2017)

Alternative idea: approximate the gradient

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

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

During optimisation we only care about the gradients!

The gradient of the variational lower-bound:

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

Alternative idea: approximate the gradient

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

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

During optimisation we only care about the gradients!

The gradient of the variational lower-bound:

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

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, don't want double-loop)

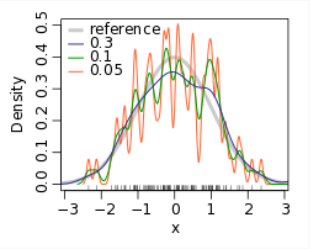
Gradient estimators (kernel based)

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:

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

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

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

Example: meta-learning for approximate inference

- learn an approx. posterior sampler for NN weights

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

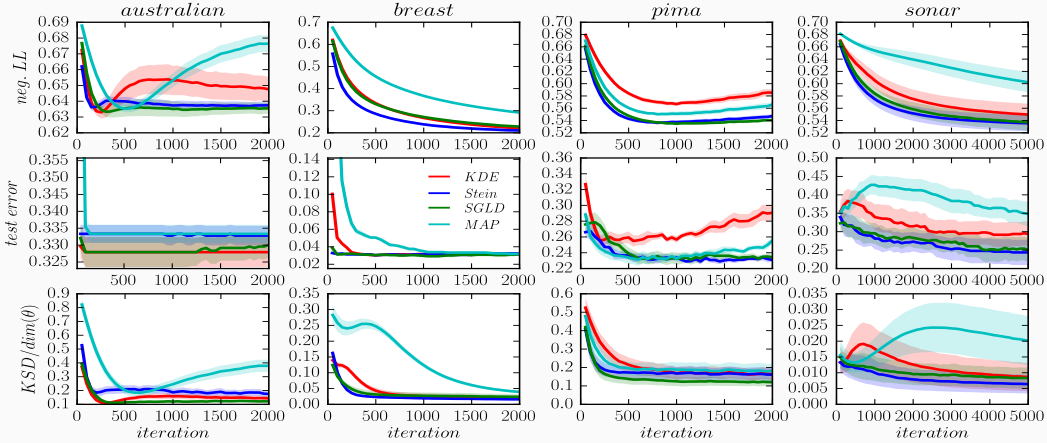
$$\nabla_t = \nabla_{\boldsymbol{\theta}_t} \left[\frac{N}{M} \sum_{m=1}^{M} \log p(y_m | \boldsymbol{x}_m, \boldsymbol{\theta}_t) + \log p_0(\boldsymbol{\theta}_t) \right].$$

- coordinates of $\Delta_\phi(\boldsymbol{\theta}_t, \nabla_t)$ and $\boldsymbol{\sigma}_\phi(\boldsymbol{\theta}_t, \nabla_t)$ 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



Summary

What we covered today:

- Is density evaluation really necessary for inference tasks?
- Fitting implicit approx. posterior
  by approximating variational lower-bound's gradients
- Designing implicit posterior approximations: big challenge

Thank you!

(BDL workshop tomorrow: training implicit generative models)