Sandwiching the marginal likelihood using bidirectional Monte Carlo

Roger Grosse

Ryan Adams

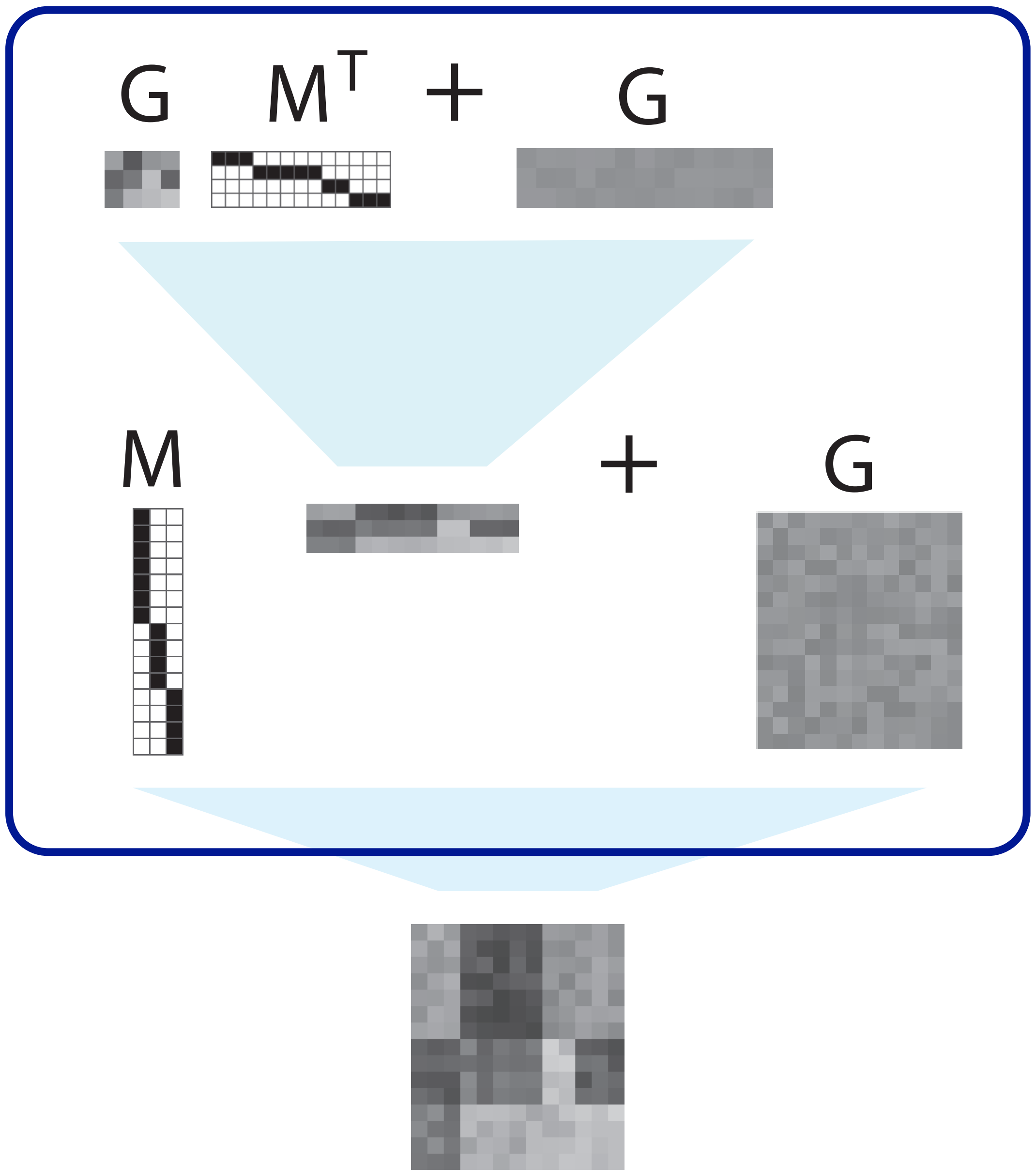
Zoubin Ghahramani

Introduction

- When comparing different statistical models, we'd like a quantitative criterion which trades off model complexity and fit to the data
- In a Bayesian setting, we often use marginal likelihood
  - Defined as the probability of the data, with all parameters and latent variables integrated out
- Motivation: plug into Bayes' Rule

$$p(\mathcal{M}_i \,|\, \mathcal{D}) = \frac{p(\mathcal{M}_i)\, p(\mathcal{D} \,|\, \mathcal{M}_i)}{\sum_j p(\mathcal{M}_j)\, p(\mathcal{D}|\mathcal{M}_j)}$$

Introduction: marginal likelihood



need to integrate out all of the component matrices and their hyperparameters

Introduction

- Advantages of marginal likelihood (ML)
  - Accounts for model complexity in a sophisticated way
  - Closely related to description length
  - Measures the model's ability to generalize to unseen examples
- ML is used in those rare cases where it is tractable
  - e.g. Gaussian processes, fully observed Bayes nets
- Unfortunately, it's typically very hard to compute because it requires a very high-dimensional integral
- While ML has been criticized on many fronts, the proposed alternatives pose similar computational difficulties

Introduction

- Focus on latent variable models
  - parameters $\boldsymbol{\theta}$, latent variables $\mathbf{z}$, observations $\mathbf{y}$
  - assume i.i.d. observations
- Marginal likelihood requires summing or integrating out latent variables and parameters

$$p(\mathbf{y}) = \int p(\boldsymbol{\theta}) \prod_{i=1}^{N} \sum_{\mathbf{z}_i} p(\mathbf{z}_i \,|\, \boldsymbol{\theta})\, p(\mathbf{y}_i \,|\, \mathbf{z}_i, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}$$

- Similar to computing the partition function

$$\mathcal{Z} = \sum_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

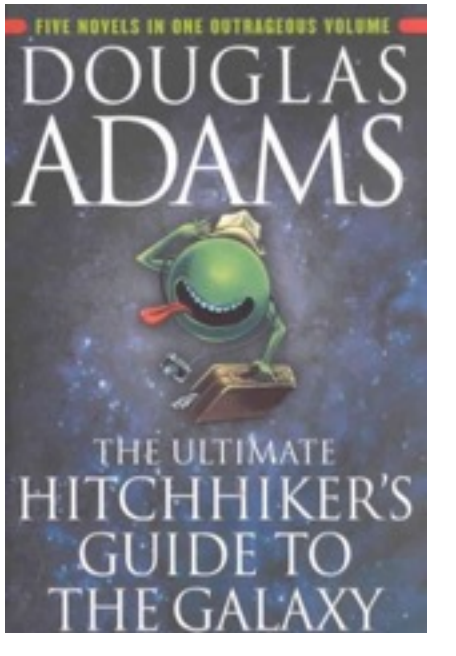
Introduction

- Problem: exact marginal likelihood computation is intractable
- There are many algorithms to approximate it, but we don't know how well they work

Why evaluating ML estimators is hard

The answer to life, the universe, and everything is...

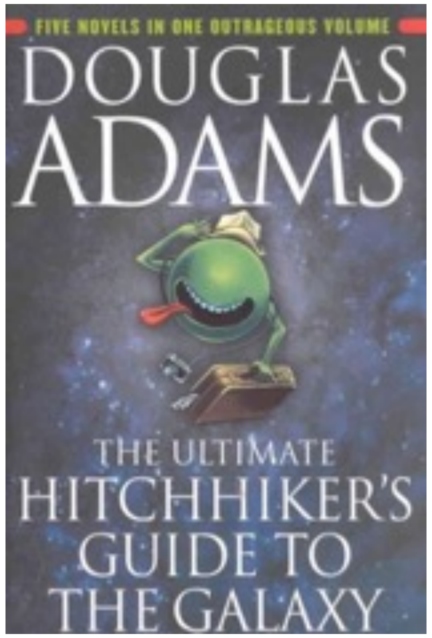
42



Why evaluating ML estimators is hard

The marginal likelihood is…

$$\log p(\mathcal{D}) = -23814.7$$



Why evaluating ML estimators is hard

- How does one deal with this in practice?
  - polynomial-time approximations for partition functions of ferromagnetic Ising models
  - test on very small instances which can be solved exactly
  - run a bunch of estimators and see if they agree with each other

Log-ML lower bounds

- One marginal likelihood estimator is simple importance sampling:

$$\{\boldsymbol{\theta}^{(k)}, \mathbf{z}^{(k)}\}_{k=1}^{K} \sim q \qquad \hat{p}(\mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} \frac{p(\boldsymbol{\theta}^{(k)}, \mathbf{z}^{(k)}, \mathcal{D})}{q(\boldsymbol{\theta}^{(k)}, \mathbf{z}^{(k)})}$$

- This is an unbiased estimator

$$\mathbb{E}[\hat{p}(\mathcal{D})] = p(\mathcal{D})$$

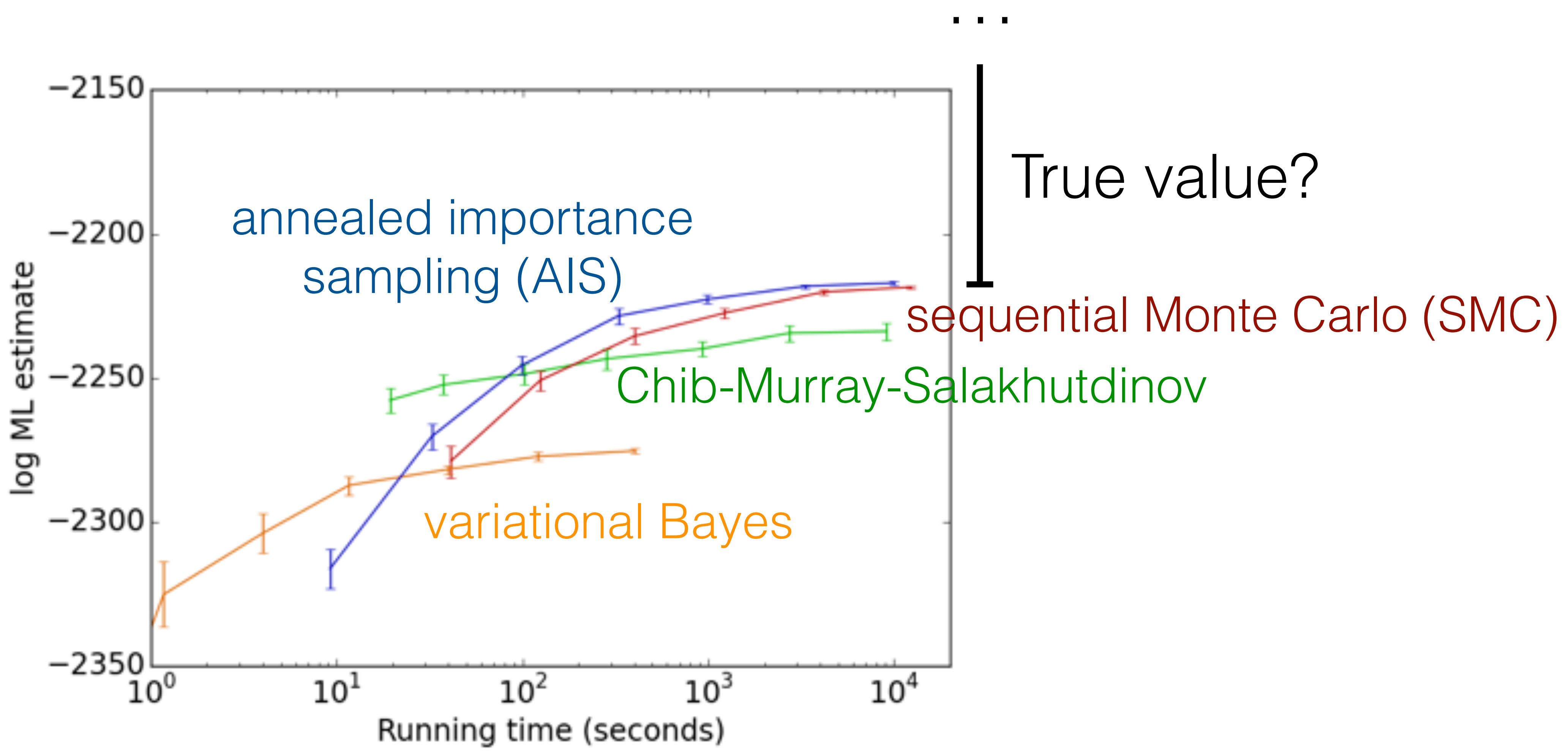
- Unbiased estimators are stochastic lower bounds

(Jensen's inequality) $$\mathbb{E}[\log \hat{p}(\mathcal{D})] \leq \log p(\mathcal{D})$$

(Markov's inequality) $$\Pr(\log \hat{p}(\mathcal{D}) > \log p(\mathcal{D}) + b) \leq e^{-b}$$

- Many widely used algorithms have the same property!

Log-ML lower bounds



How to obtain an upper bound?

- Harmonic Mean Estimator:

$$\{\boldsymbol{\theta}^{(k)}, \mathbf{z}^{(k)}\}_{k=1}^{K} \sim p(\boldsymbol{\theta}, \mathbf{z} \,|\, \mathcal{D}) \qquad \hat{p}(\mathcal{D}) = \frac{K}{\sum_{k=1}^{K} 1/p(\mathcal{D} \,|\, \boldsymbol{\theta}^{(k)}, \mathbf{z}^{(k)})}$$

- Equivalent to simple importance sampling, but with the role of the proposal and target distributions reversed

- Unbiased estimate of the *reciprocal* of the ML

$$\mathbb{E}\left[\frac{1}{\hat{p}(\mathcal{D})}\right] = \frac{1}{p(\mathcal{D})}$$

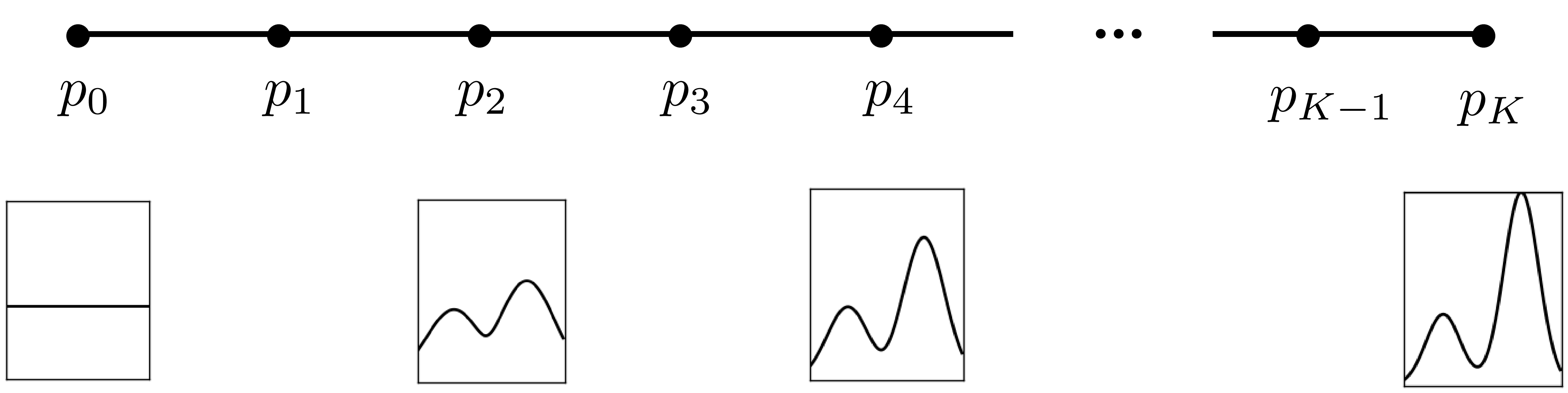
- Gives a stochastic *upper* bound on the log-ML

- Caveat 1: only an upper bound if you sample *exactly* from the posterior, which is generally intractable

- Caveat 2: this is the Worst Monte Carlo Estimator (Neal, 2008)

Annealed importance sampling

(Neal, 2001)



tractable initial distribution (e.g. prior)

intractable target distribution (e.g. posterior)

Annealed importance sampling
(Neal, 2001)

Given:

unnormalized distributions $f_0, \ldots, f_K$

MCMC transition operators $\mathcal{T}_0, \ldots, \mathcal{T}_K$

f_0 easy to sample from, compute partition function of

$\mathbf{x} \sim f_0$

$w = 1$

For $i = 0, \ldots, K-1$

$w := w \frac{f_{i+1}(\mathbf{x})}{f_i(\mathbf{x})}$

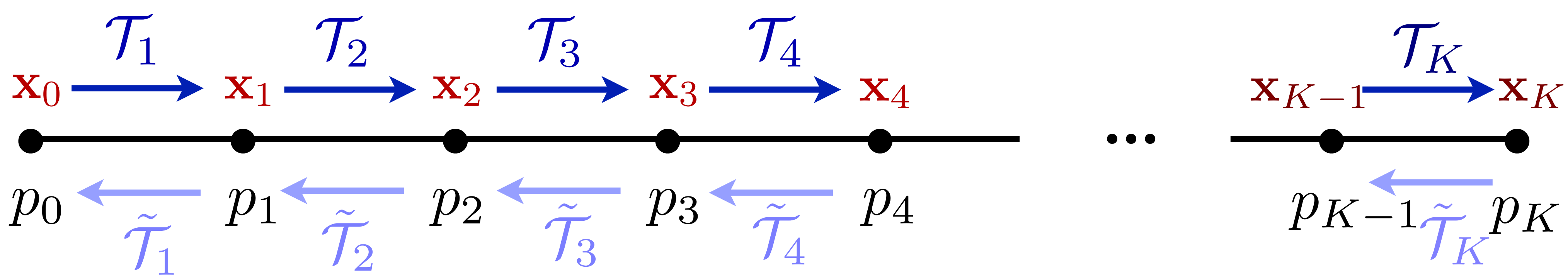
$\mathbf{x} :\sim \mathcal{T}_{i+1}(\mathbf{x})$

Then, $\mathbb{E}[w] = \frac{\mathcal{Z}_K}{\mathcal{Z}_0}$

$$\hat{\mathcal{Z}}_K = \frac{\mathcal{Z}_0}{S} \sum_{s=1}^{S} w^{(s)}$$

Annealed importance sampling

(Neal, 2001)



Forward: $$w := \prod_{i=1}^{K} \frac{f_i(\mathbf{x}_{i-1})}{f_{i-1}(\mathbf{x}_{i-1})} = \frac{\mathcal{Z}_K}{\mathcal{Z}_0} \frac{q_{\text{back}}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K)}{q_{\text{fwd}}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K)} \qquad \mathbb{E}[w] = \frac{\mathcal{Z}_K}{\mathcal{Z}_0}$$

Backward: $$w := \prod_{i=1}^{K} \frac{f_{i-1}(\mathbf{x}_i)}{f_i(\mathbf{x}_i)} = \frac{\mathcal{Z}_0}{\mathcal{Z}_K} \frac{q_{\text{fwd}}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K)}{q_{\text{back}}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K)} \qquad \mathbb{E}[w] = \frac{\mathcal{Z}_0}{\mathcal{Z}_K}$$

Bidirectional Monte Carlo

- Initial distribution: prior $p(\boldsymbol{\theta}, \mathbf{z})$
- Target distribution: posterior $p(\boldsymbol{\theta}, \mathbf{z} \,|\, \mathcal{D}) = \dfrac{p(\boldsymbol{\theta}, \mathbf{z}, \mathcal{D})}{p(\mathcal{D})}$
- Partition function: $\mathcal{Z} = \int p(\boldsymbol{\theta}, \mathbf{z}, \mathcal{D}) \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\mathbf{z} = p(\mathcal{D})$
- Forward chain

$$\mathbb{E}[w] = \frac{\mathcal{Z}_K}{\mathcal{Z}_0} = p(\mathcal{D})$$

stochastic lower bound

- Backward chain (requires exact posterior sample!)

$$\mathbb{E}[w] = \frac{\mathcal{Z}_0}{\mathcal{Z}_K} = \frac{1}{p(\mathcal{D})}$$

stochastic upper bound

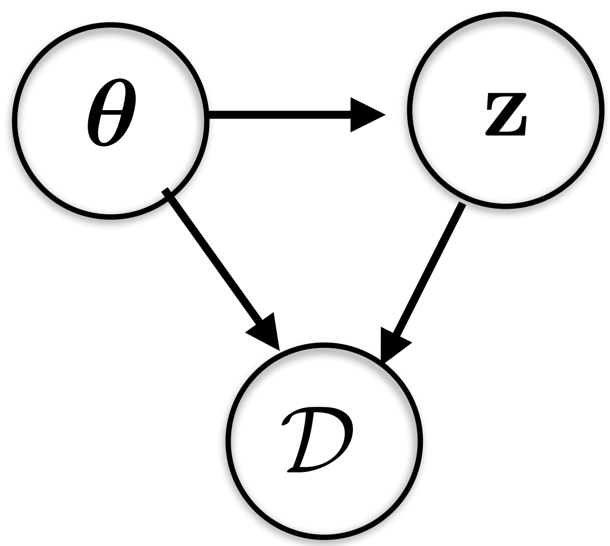
Bidirectional Monte Carlo

How to get an exact sample?

Two ways to sample from $p(\boldsymbol{\theta}, \mathbf{z}, \mathcal{D})$

$p(\boldsymbol{\theta}, \mathbf{z})\, p(\mathcal{D} \,|\, \boldsymbol{\theta}, \mathbf{z})$

forward sample



$p(\mathcal{D})\, p(\boldsymbol{\theta}, \mathbf{z} \,|\, \mathcal{D})$

generate data, then perform inference

Therefore, the parameters and latent variables used to generate the data are an exact posterior sample!

Bidirectional Monte Carlo

Summary of algorithm:

$$\boldsymbol{\theta}^\star, \mathbf{z}^\star \sim p_{\boldsymbol{\theta},\mathbf{z}}$$

$$\mathbf{y} \sim p_{\mathbf{y}\,|\,\boldsymbol{\theta},\mathbf{z}}(\cdot\,|\,\boldsymbol{\theta}^\star, \mathbf{z}^\star)$$

Obtain a stochastic lower bound on $\log p(\mathbf{y})$ by running AIS forwards

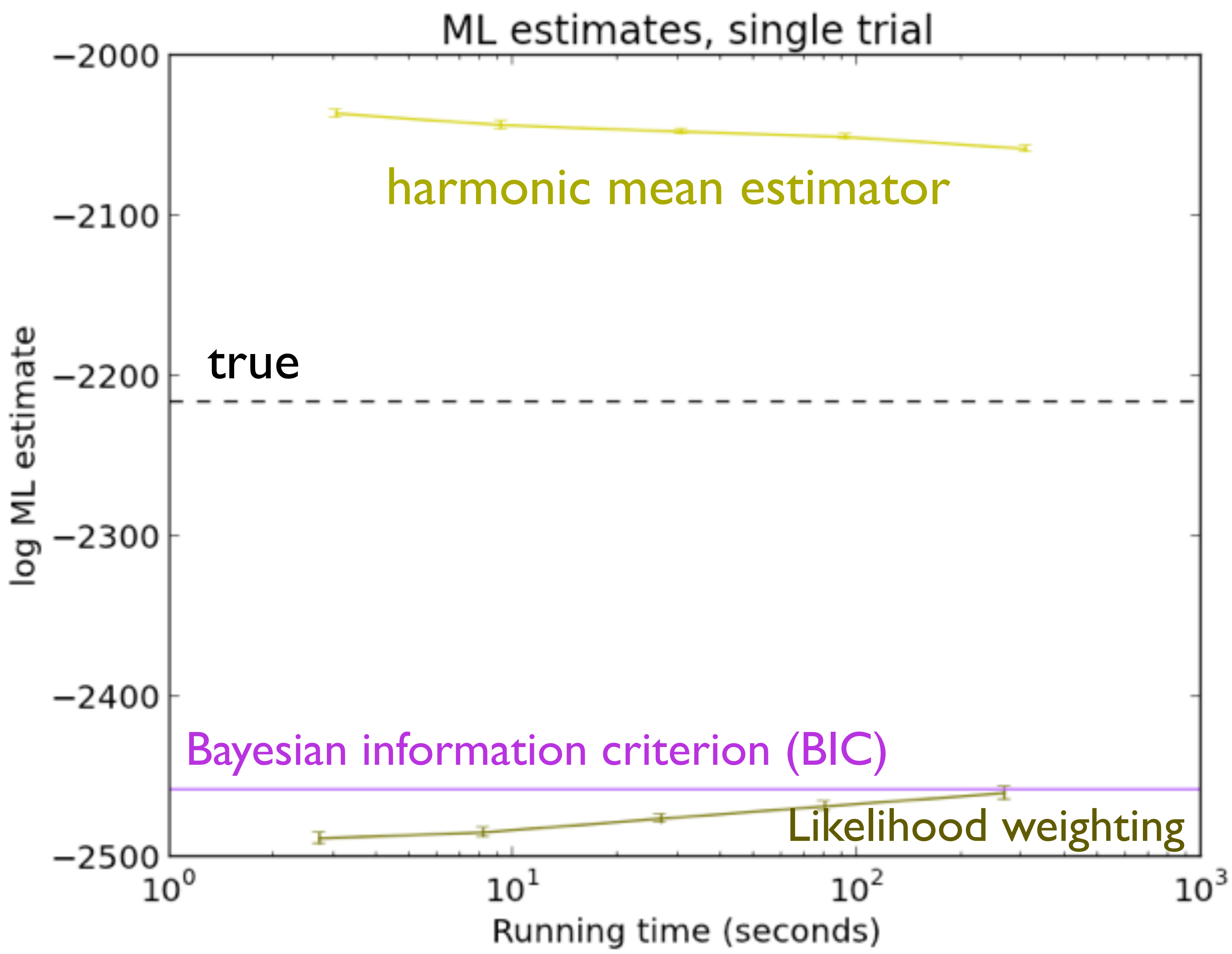
Obtain a stochastic upper bound on $\log p(\mathbf{y})$ by running AIS backwards, starting from $(\boldsymbol{\theta}^\star, \mathbf{z}^\star)$

The two bounds will converge given enough intermediate distributions.

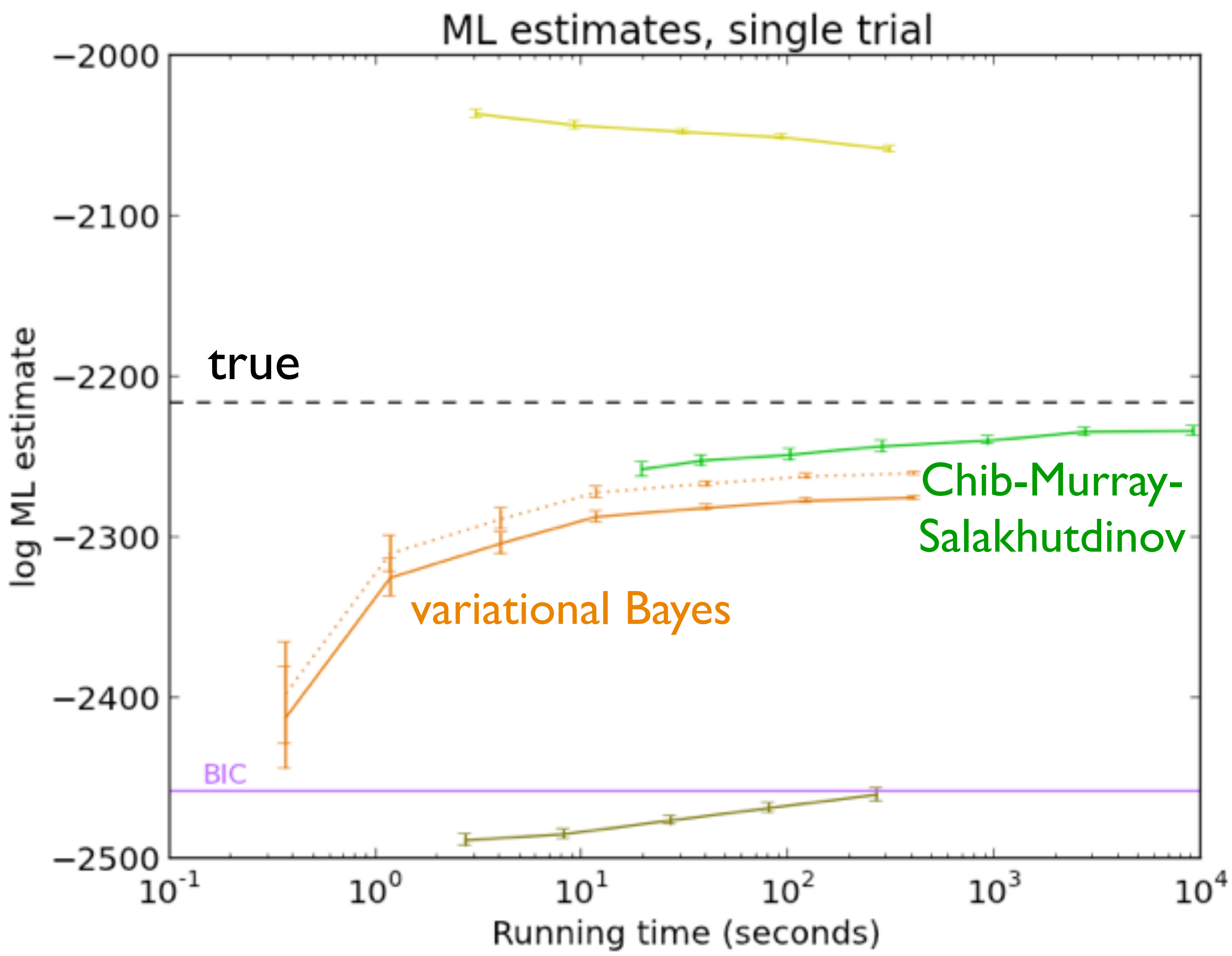
Experiments

- BDMC lets us compute ground truth log-ML values for data simulated from a model
  - We can use these ground truth values to benchmark log-ML estimators!
- Obtained ground truth ML for simulated data for
  - clustering
  - low rank approximation
  - binary attributes
- Compared a wide variety of ML estimators
- MCMC operators shared between all algorithms wherever possible

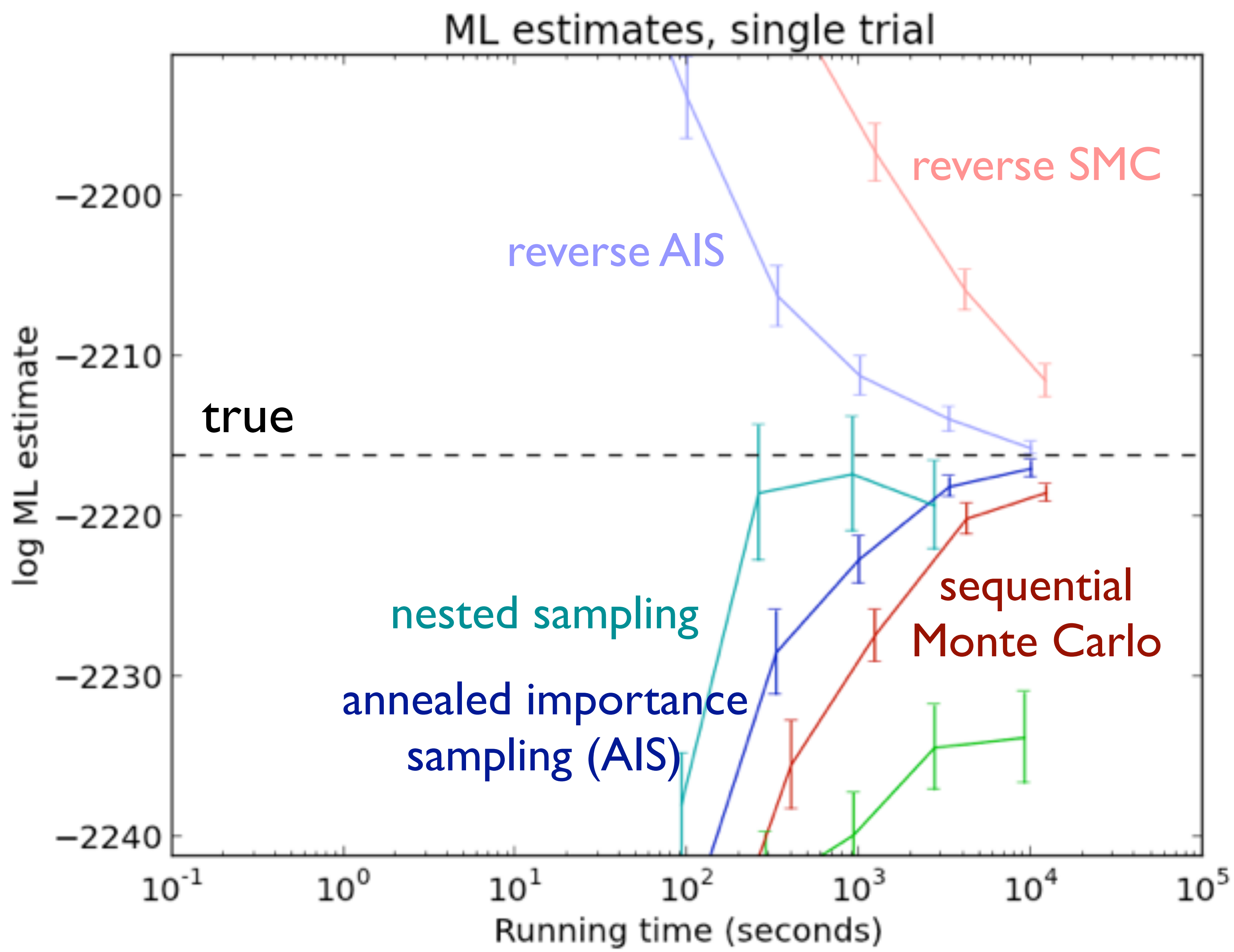
Results: binary attributes



Results: binary attributes

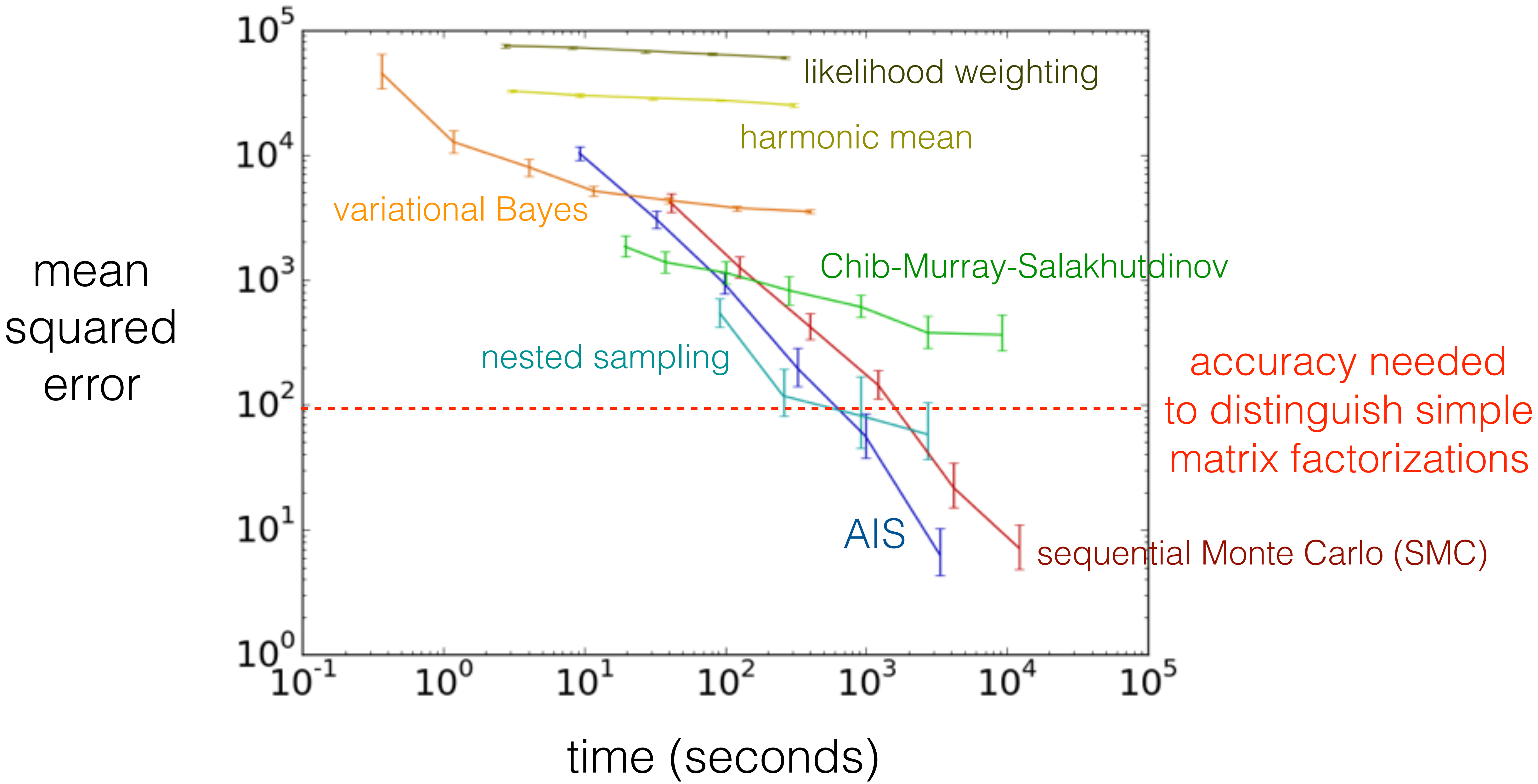


Results: binary attributes (zoomed in)

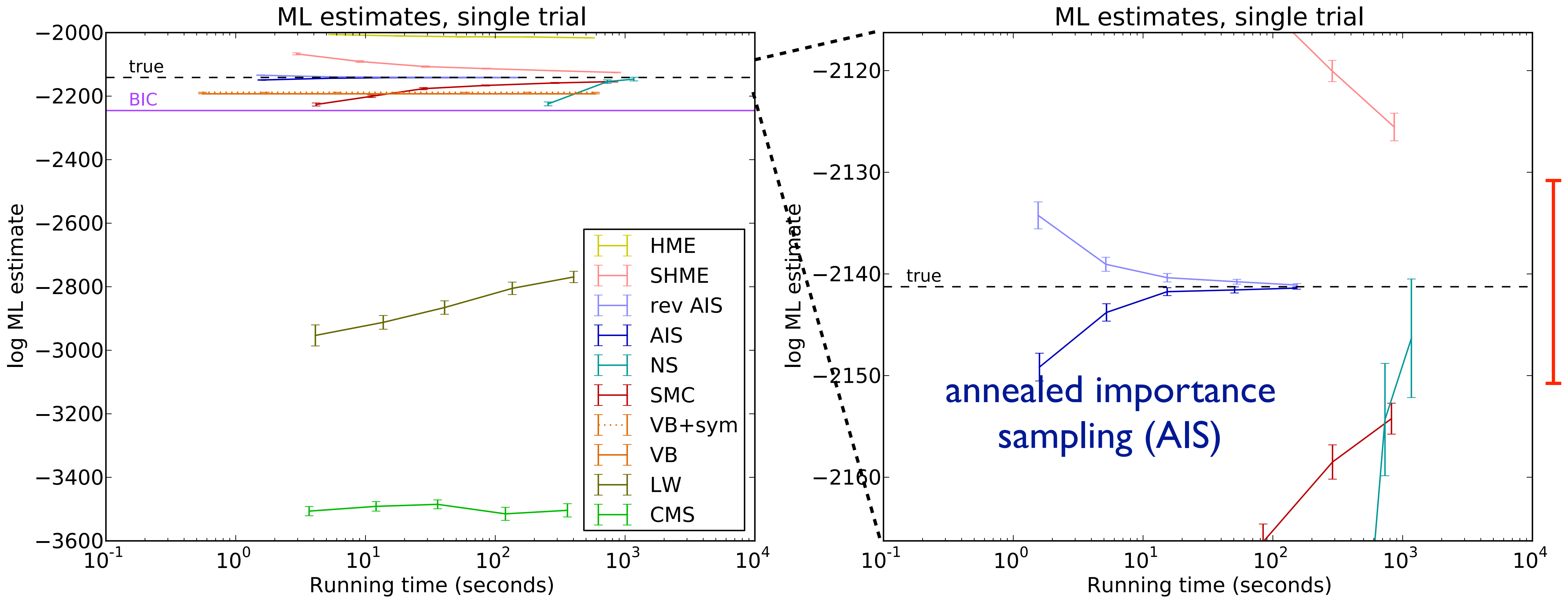


Results: binary attributes

Which estimators give accurate results?



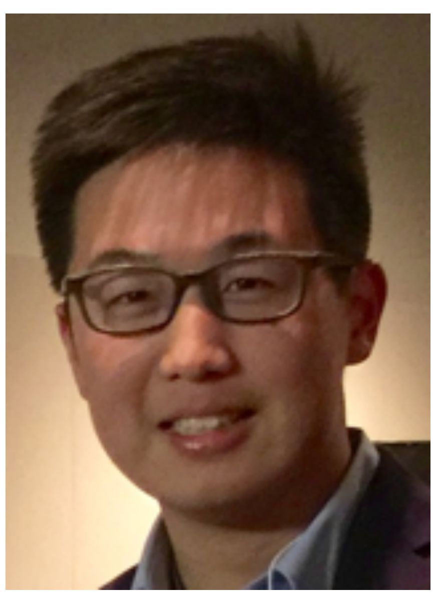
Results: low rank approximation



Recommendations

- Try AIS first
- If AIS is too slow, try sequential Monte Carlo or nested sampling
- Can’t fix a bad algorithm by averaging many samples
- Don’t trust naive confidence intervals -- need to evaluate rigorously

On the quantitative evaluation of decoder-based generative models



Yuhuai Wu

Yuri Burda

Ruslan Salakhutdinov

Decoder-based generative models

- Define a generative process:
  - sample latent variables z from a simple (fixed) prior p(z)
  - pass them through a decoder network to get x = f(z)
- Examples:
  - variational autoencoders (Kingma and Welling, 2014)
  - generative adversarial networks (Goodfellow et al., 2014)
  - generative moment matching networks (Li et al., 2015; Dziugaite et al., 2015)
  - nonlinear independent components estimation (Dinh et al., 2015)

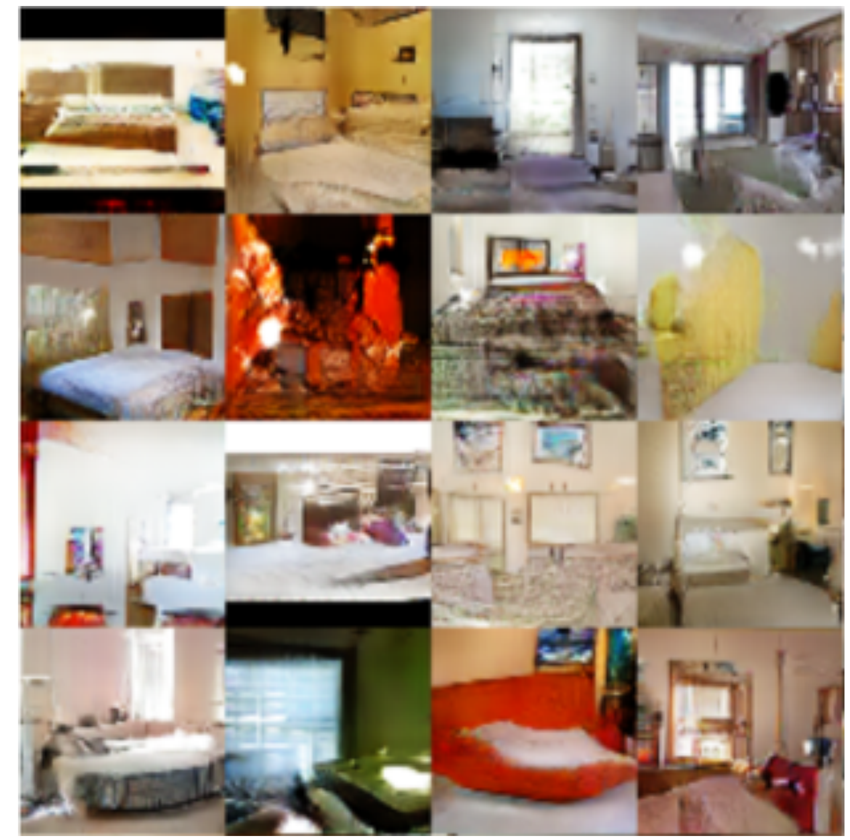
Decoder-based generative models

- Variational autoencoder (VAE)
  - Train both a generator (decoder) and a recognition network (encoder)
  - Optimize a variational lower bound on the log-likelihood
- Generative adversarial network (GAN)
  - Train a generator (decoder) and a discriminator
  - Discriminator wants to distinguish model samples from the training data
  - Generator wants to fool the discriminator
- Generative moment matching network (GMMN)
  - Train a generative network such that certain statistics match between the generated samples and the data

Decoder-based generative models

Some impressive-looking samples:

Denton et al. (2015)



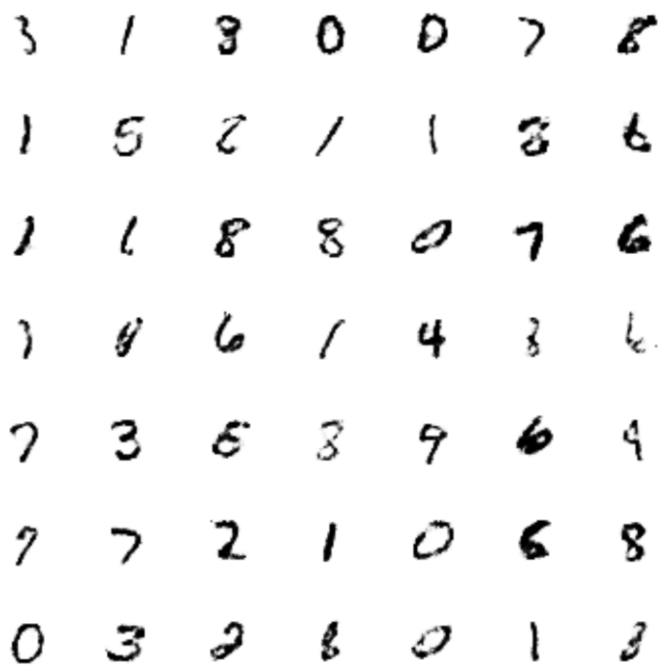
Radford et al. (2016)

But how well do these models capture the distribution?

Decoder-based generative models

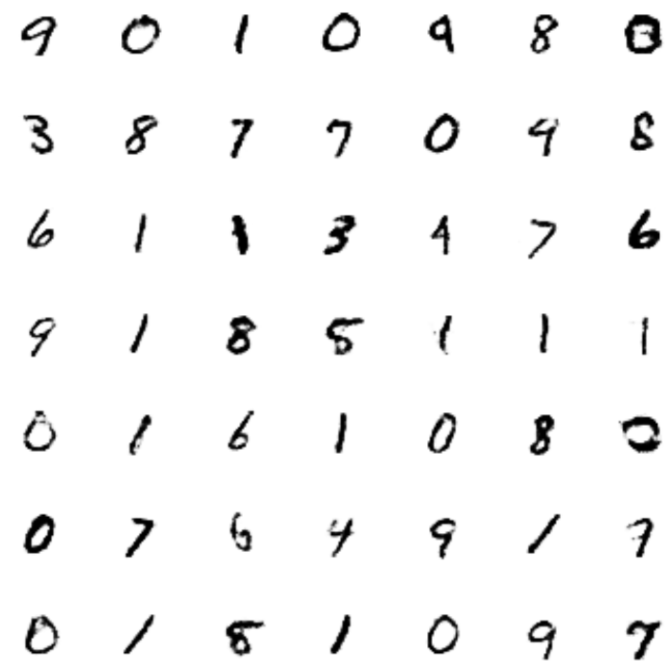
Looking at samples can be misleading:

Decoder-based generative models



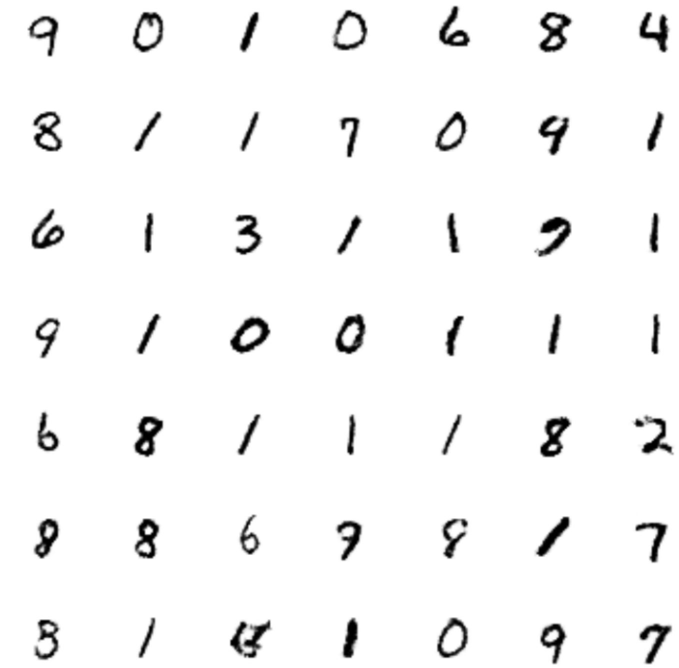
GAN, 10 dim

LLD = 328.7



GAN, 50 dim, 200 epochs

LLD = 543.5



GAN, 50 dim, 1000 epochs

LLD = 625.5

Evaluating decoder-based models

- Want to quantitatively evaluate generative models in terms of the probability of held-out data
- Problem: a GAN or GMMN with k latent dimensions can only generate within a k-dimensional submanifold!
- Standard (but unsatisfying) solution: impose a spherical Gaussian observation model

$$p_\sigma(\mathbf{x} \,|\, \mathbf{z}) = \mathcal{N}(f(\mathbf{z}), \sigma \mathbf{I})$$

  - tune σ on a validation set
- Problem: this still requires computing an intractable integral:

$$p_\sigma(\mathbf{x}) = \int p(\mathbf{z})\, p_\sigma(\mathbf{x} \,|\, \mathbf{z})\, \mathrm{d}\mathbf{z}$$

Evaluating decoder-based models

- For some models, we can tractably compute log-likelihoods, or at least a reasonable lower bound

- Tractable likelihoods for models with reversible decoders (e.g. NICE)

- Variational autoencoders: ELBO lower bound

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z} \,|\, \mathbf{x})}[\log p(\mathbf{x} \,|\, \mathbf{z})] - \mathrm{D}_{\mathrm{KL}}(q(\mathbf{z} \,|\, \mathbf{x}) \,\|\, p(\mathbf{z}))$$

- Importance Weighted Autoencoder

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z} \,|\, \mathbf{x})}\left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z} \,|\, \mathbf{x})}\right]$$

- In general, we don't have accurate and tractable bounds
  - Even in the cases of VAEs and IWAEs, we don't know how accurate the bounds are

Evaluating decoder-based models

- Currently, results reported using kernel density estimation (KDE)

$$\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(S)} \sim p(\mathbf{z})$$

$$\hat{p}_\sigma(\mathbf{x}) = \frac{1}{S} \sum_{s=1}^{S} p_\sigma(\mathbf{x} \,|\, \mathbf{z}^{(s)})$$

  - Can show this is a stochastic lower bound:

$$\mathbb{E}[\log \hat{p}_\sigma(\mathbf{x})] \leq \log p_\sigma(\mathbf{x})$$

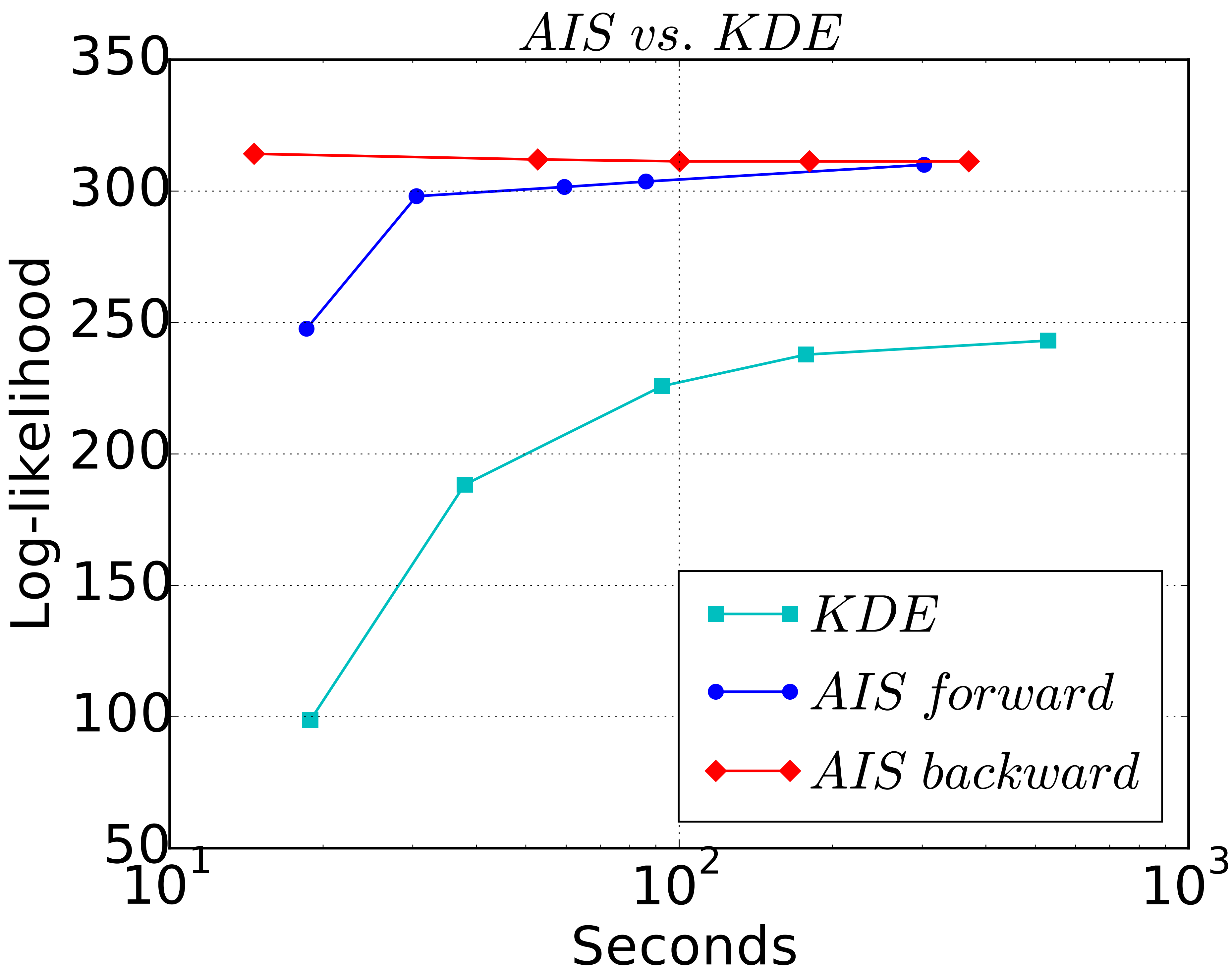
  - Unlikely to perform well in high dimensions
  - Papers caution the reader not to trust the results

Evaluating decoder-based models

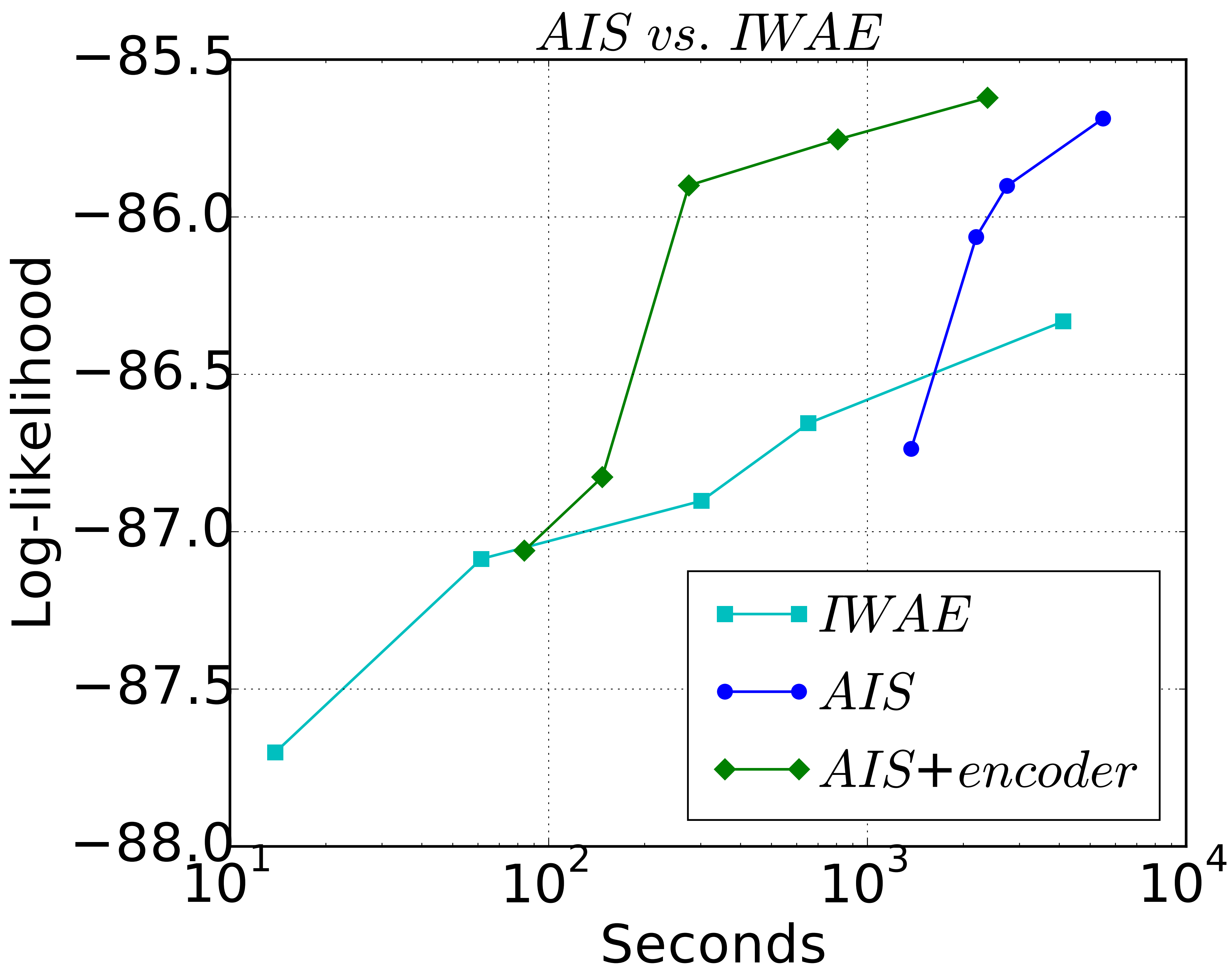
- Our approach: integrate out latent variables using AIS, with Hamiltonian Monte Carlo (HMC) as the transition operator
- Validate the accuracy of the estimates on simulated data using BDMC
- Experiment details
  - Real-valued MNIST dataset
  - VAEs, GANs, GMMNs with the following decoder architectures:
    - 10-64-256-256-1024-784
    - 50-1024-1024-1024-784
  - Spherical Gaussian observations imposed on all models (including VAE)

How accurate are AIS and KDE?

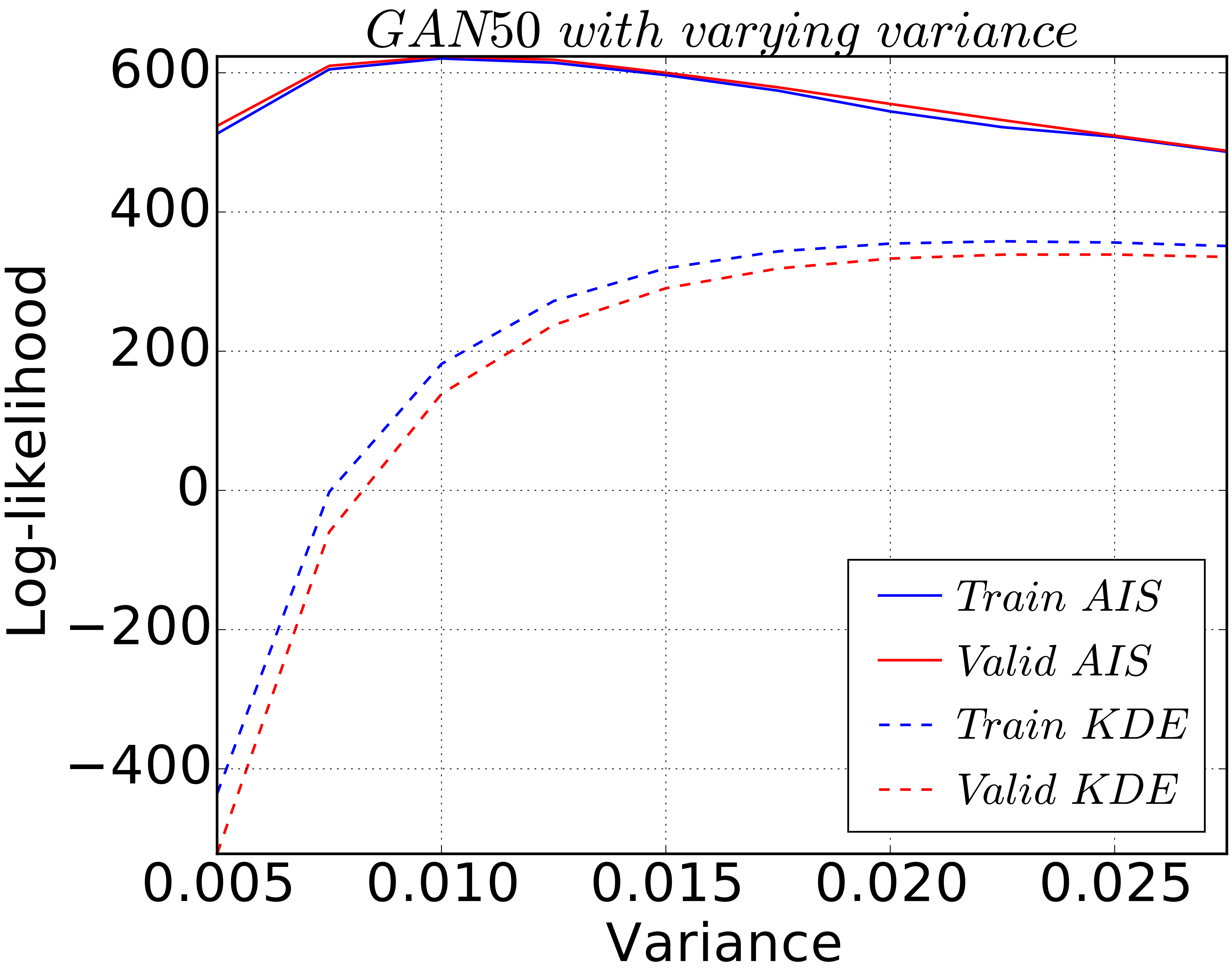
(GMMN-50)



How accurate is the IWAE bound?



Estimation of variance parameter



Comparison of different models

(Nats)	AIS Test (1000ex)	AIS Train (100ex)	BDMC gap	KDE Test	IWAE Test
VAE-50	991.435±6.477	1272.586±6.759	1.540	351.213	826.325
GAN-50	627.297±8.813	620.498±31.012	10.045	300.331	/
GMMN-50	593.472±8.591	571.803±30.864	1.146	277.193	/
VAE-10	705.375±7.411	780.196±19.147	0.832	408.659	486.466
GAN-10	328.772±5.538	318.948±22.544	0.934	259.673	/
GMMN-10	346.679±5.860	345.176±19.893	0.605	262.73	/

AIS estimates are accurate (small BDMC gap)

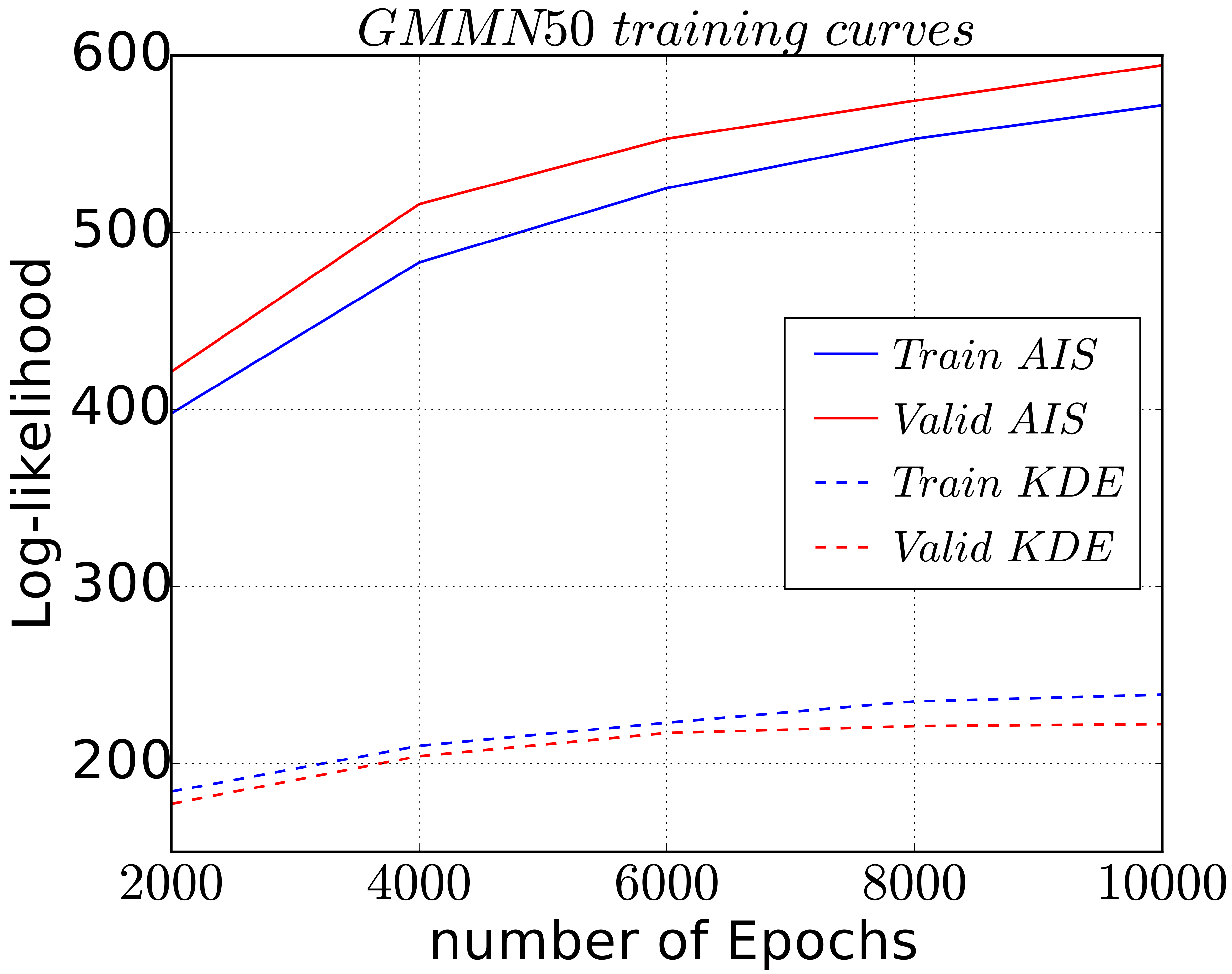
Larger model ==> much higher log-likelihood

VAEs achieve much higher log-likelihood than GANs and GMMNs

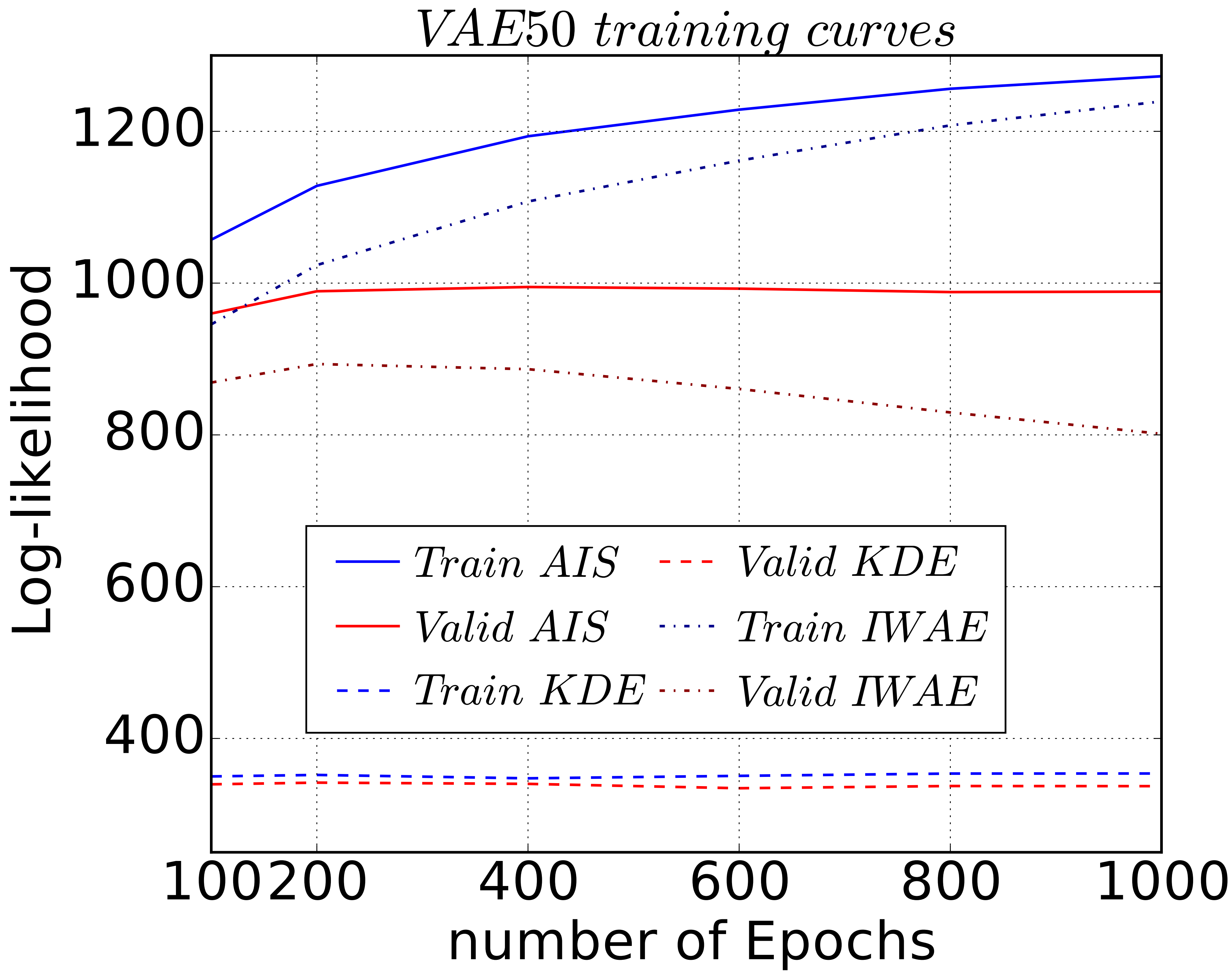
For GANs and GMMNs, no statistically significant difference between training and test log-likelihoods!

These models are not just memorizing training examples.

Training curves for a GMMN

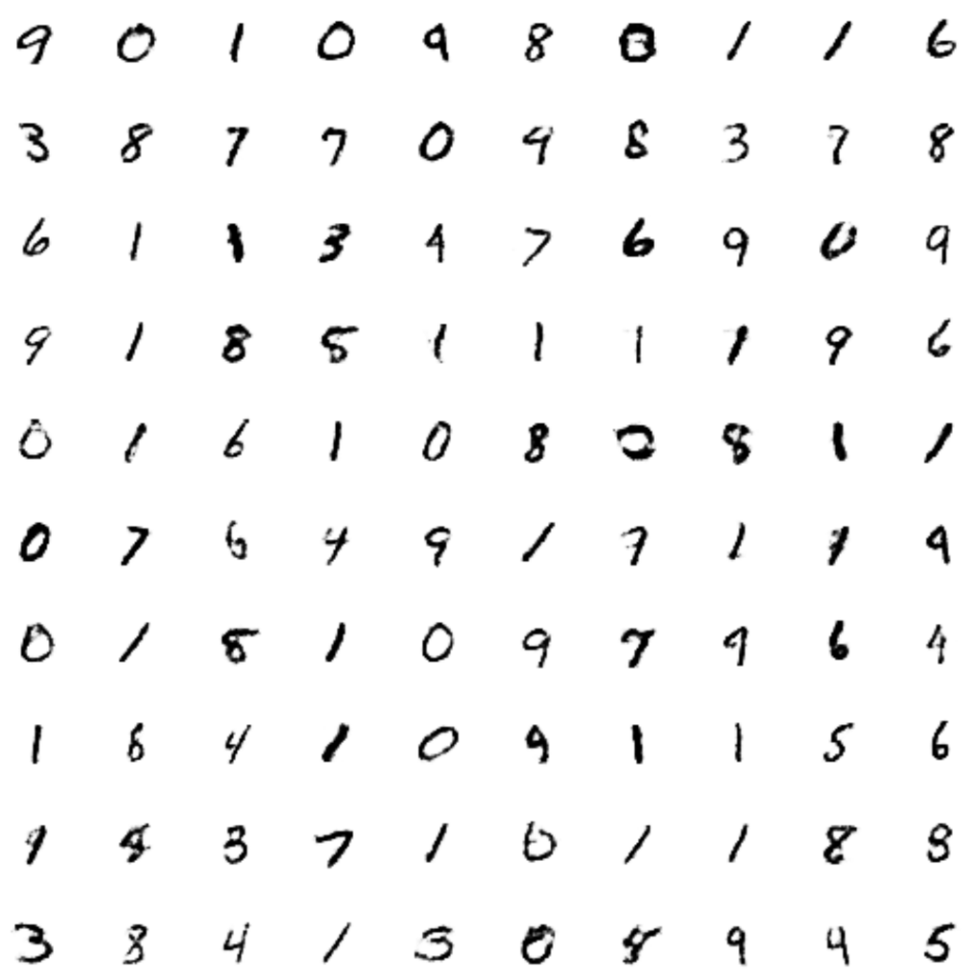


Training curves for a VAE

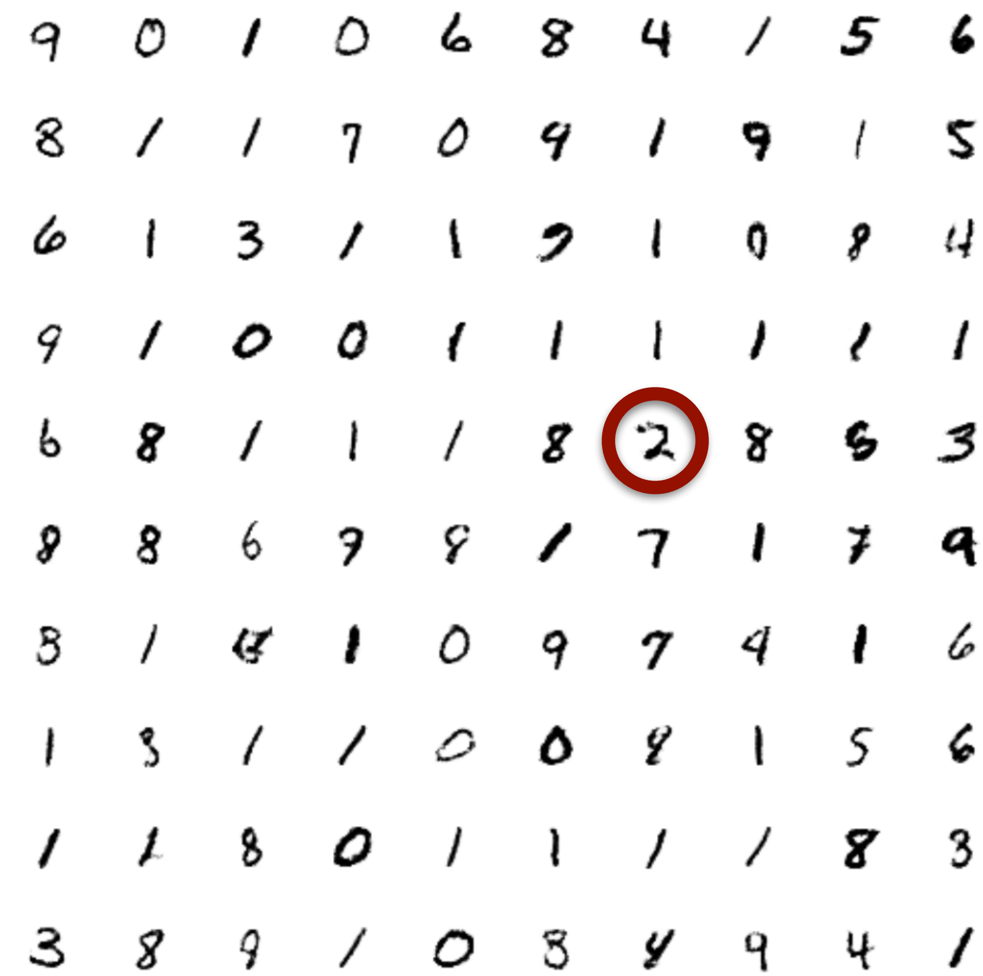


Missing modes

The GAN seriously misallocates probability mass between modes:



200 epochs



1000 epochs

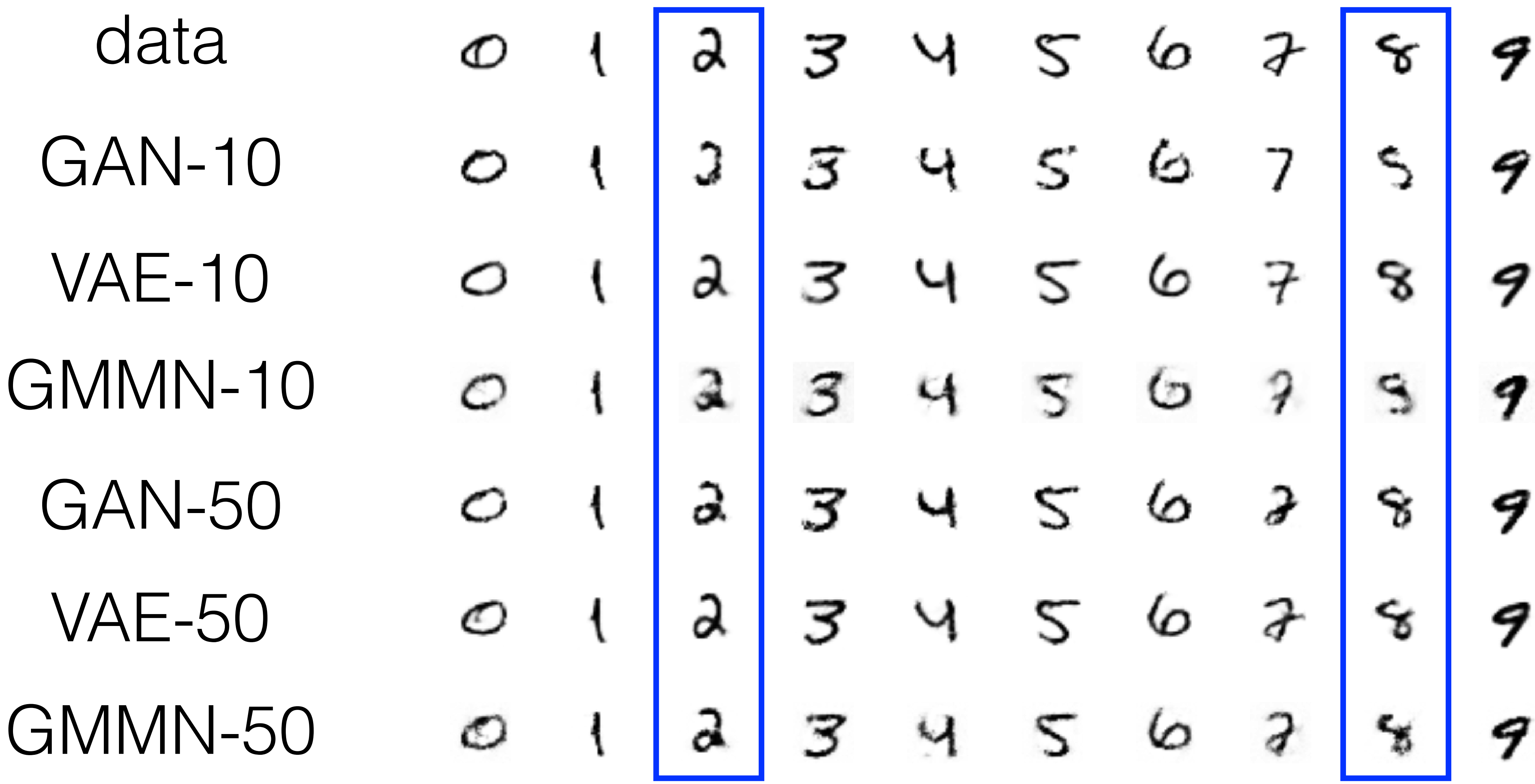
But this effect by itself is too small to explain why it underperforms the VAE by over 350 nats

Missing modes

- To see if the network is missing modes, let's visualize posterior samples given observations.
- Use AIS to approximately sample z from p(z | x), then run the decoder
- Using BDMC, we can validate the accuracy of AIS samples on simulated data

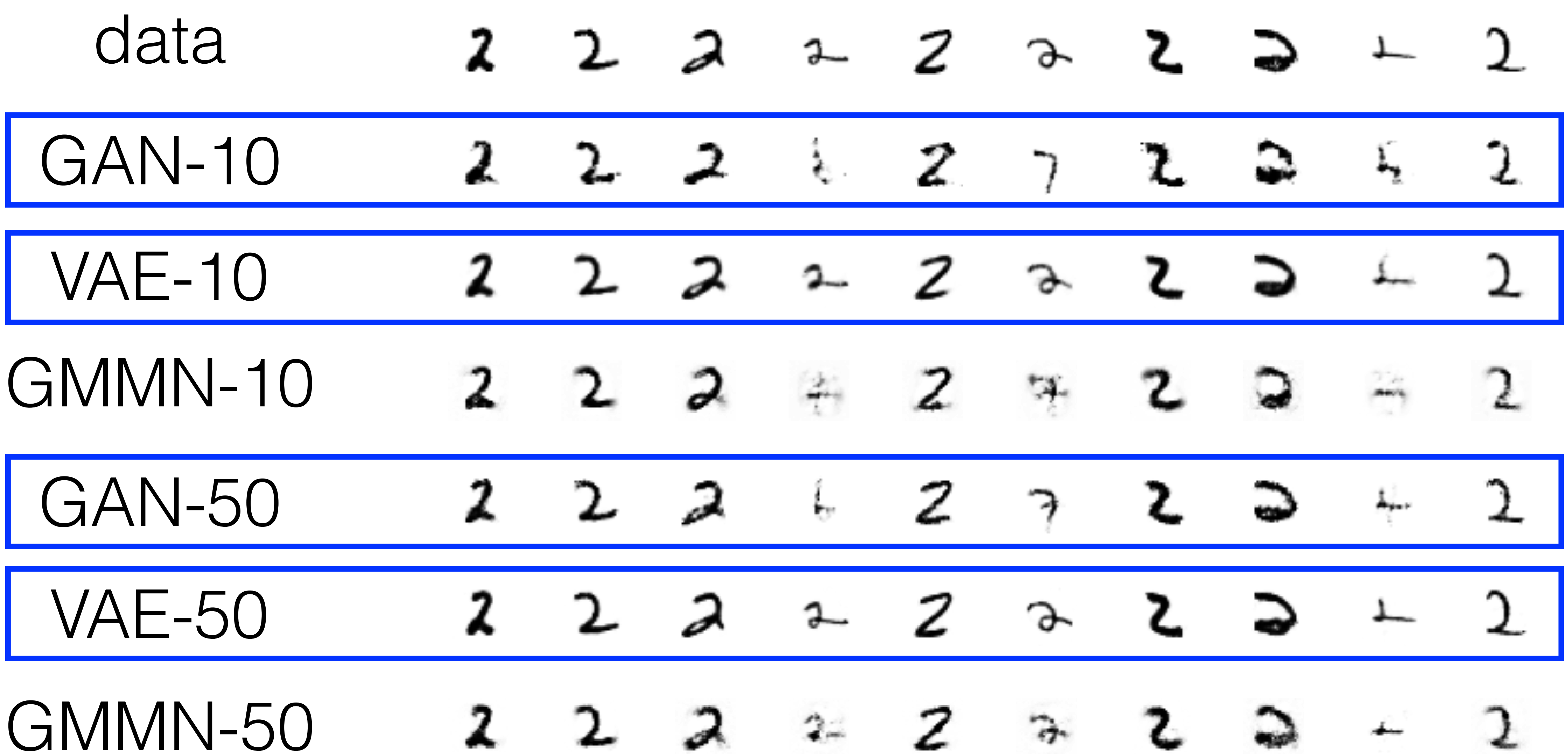
Missing modes

Visualization of posterior samples for validation images



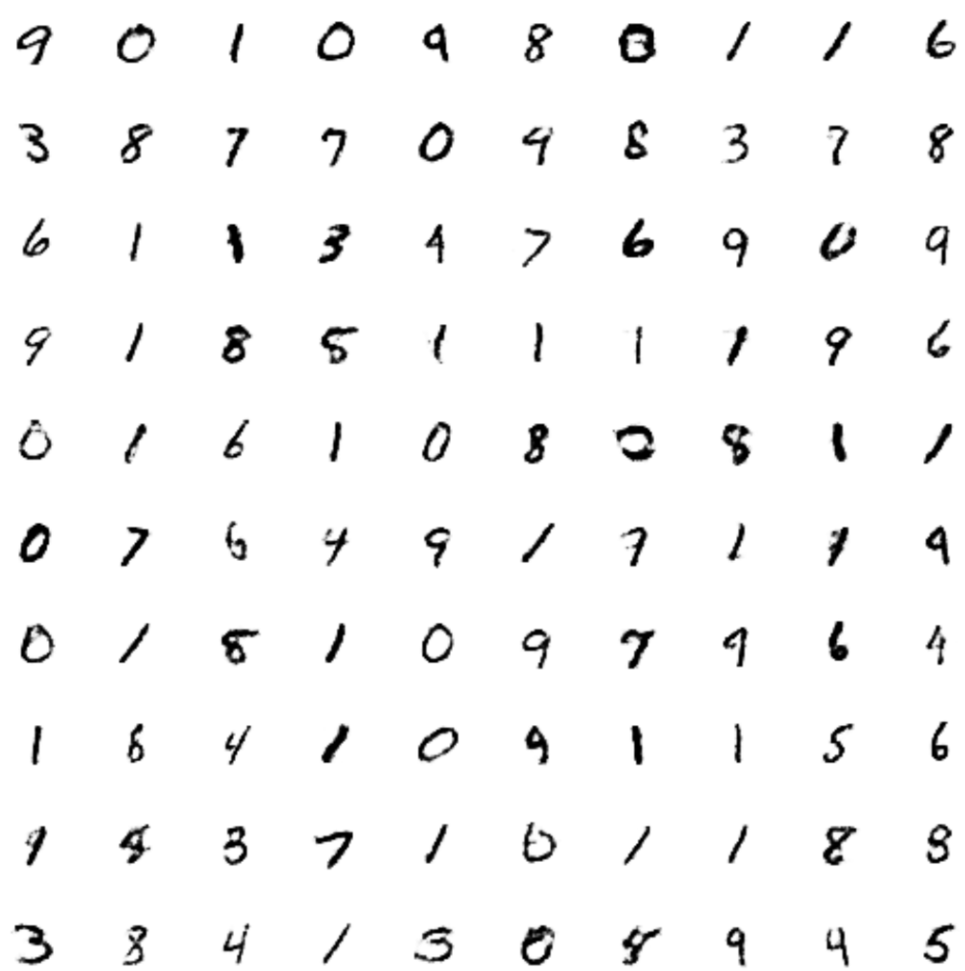
Missing modes

Posterior samples on *training* set

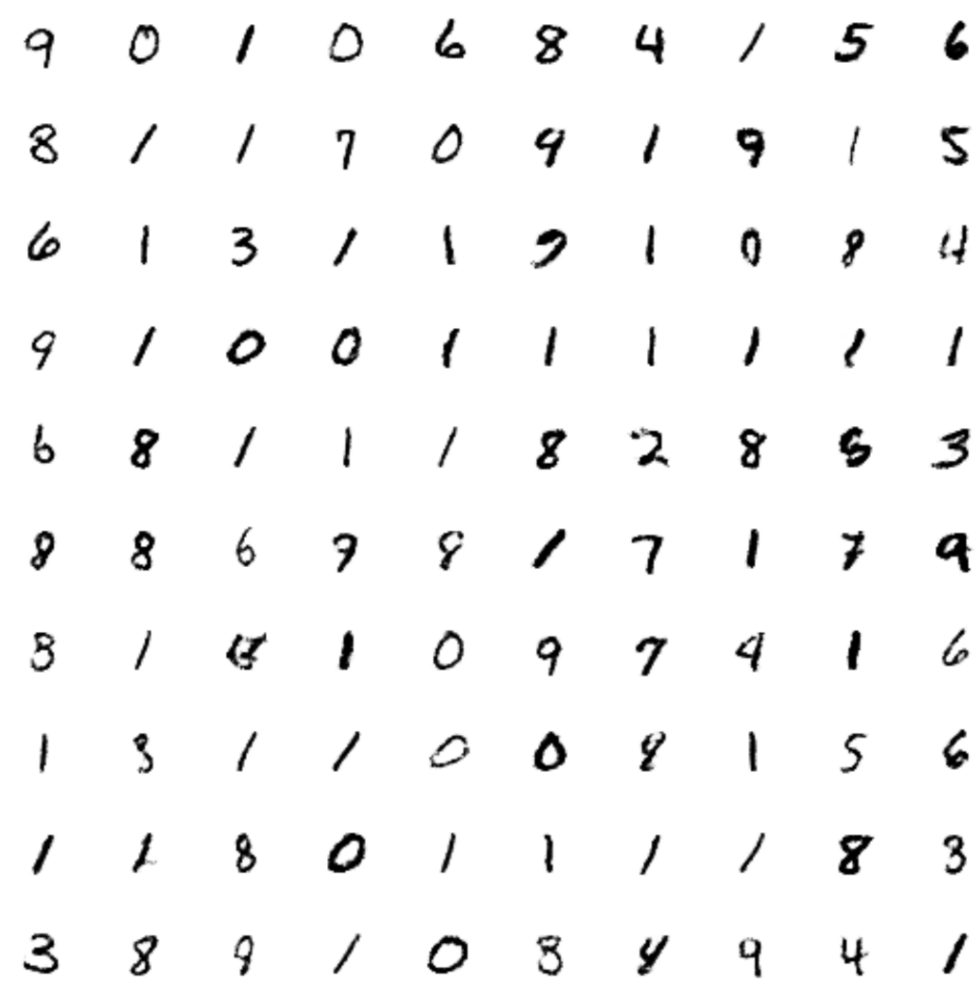


Missing modes

Conjecture: the GAN acts like a frustrated student



200 epochs



1000 epochs

Conclusions

- AIS gives high-accuracy log-likelihood estimates on MNIST (as validated by BDMC)
- This lets us observe interesting phenomena that are invisible to KDE
- GANs and GMMNs are not just memorizing training examples
- VAEs achieve substantially higher log-likelihoods than GANs and GMMNs
  - This appears to reflect failure to model certain modes of the data distribution
- Recognition nets can overfit
- Networks may continue to improve during training, even if KDE estimates don't reflect that
- Will be interesting to measure the effects of other algorithmic improvements to these networks