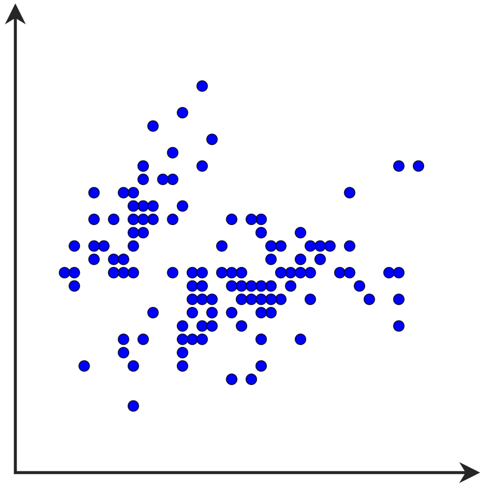
Density Estimation for Likelihood-free Inference

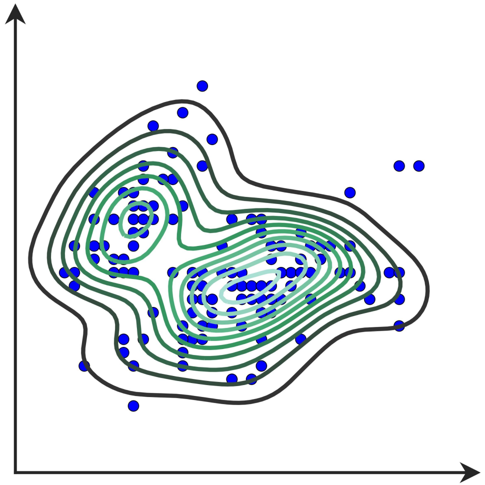
George Papamakarios

THE UNIVERSITY of EDINBURGH

Density estimation

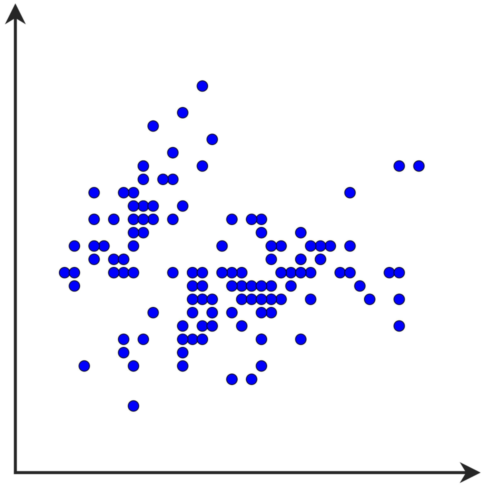


Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

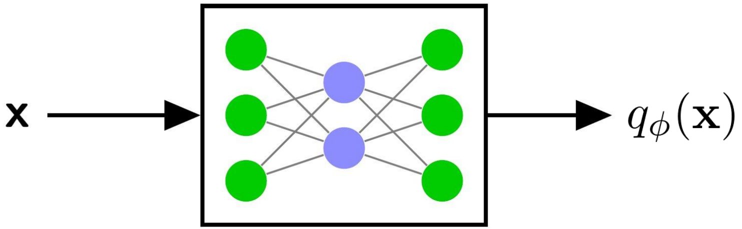


Goal: estimate density function $p(\mathbf{x})$

Neural density estimation



Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



$q_\phi(\mathbf{x})$ must be a density function:

$$\int q_\phi(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1 \qquad q_\phi(\mathbf{x}) \geq 0$$

Neural density estimation: Training

Average log likelihood:

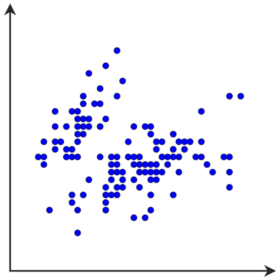
$$L(\phi) = \frac{1}{N} \sum_{n=1}^{N} \log q_\phi(\mathbf{x}_n)$$

Maximize $L(\phi)$ w.r.t. ϕ

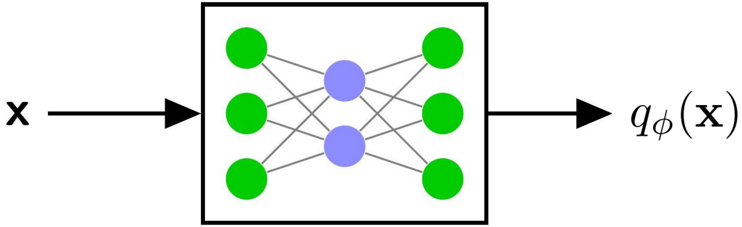
Calculate gradients with backpropagation

Use favourite type of stochastic-gradient ascent (e.g. Adam)

Use favourite machine-learning framework (e.g. TensorFlow, PyTorch)



Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



Training as KL-divergence minimization

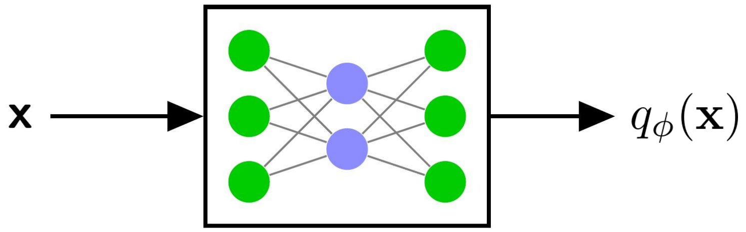
For $N \to \infty$:

$$L(\phi) = \frac{1}{N}\sum_{n=1}^{N} \log q_\phi(\mathbf{x}_n) \to \mathbb{E}_{\mathbf{x}\sim p}(\log q_\phi(\mathbf{x}))$$

Maximizing average log likelihood is asymptotically equivalent to minimizing KL divergence from the true data density, since:

$$D_{\mathrm{KL}}(p \,\|\, q_\phi) = -\mathbb{E}_{\mathbf{x}\sim p}(\log q_\phi(\mathbf{x})) + \mathrm{const}$$

How to design flexible neural density models?



$q_\phi(\mathbf{x})$ must be a density function:

$$\int q_\phi(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1 \qquad q_\phi(\mathbf{x}) \geq 0$$

Parameters ϕ should be unconstrained

$q_\phi(\mathbf{x})$ must be differentiable w.r.t. ϕ

Mixture models

Model is a weighted average of simple density models

e.g. Gaussian mixture model:

$$q_\phi(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$$

Reparameterize so that parameters are unconstrained

$$(\alpha_1, \ldots, \alpha_K) = \mathrm{softmax}(\alpha'_1, \ldots, \alpha'_K)$$
$$\Sigma_k = L_k L_k^T$$

Optimize with respect to:

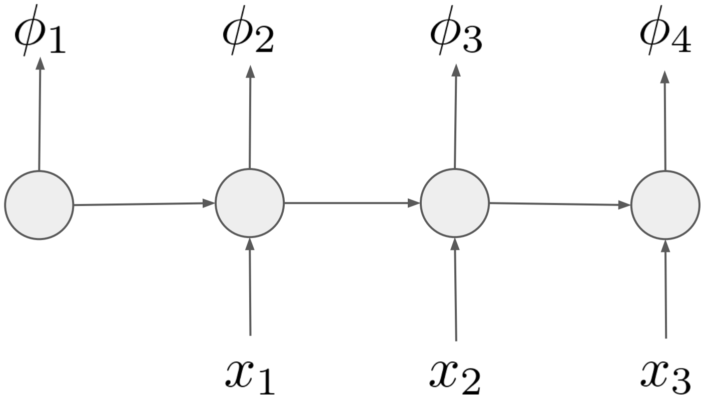
$$\phi = \{\alpha'_k, \mu_k, L_k\}_{k=1:K}$$

Autoregressive models

$$\mathbf{x} = (x_1, \dots, x_I)$$

Chain rule of probability:

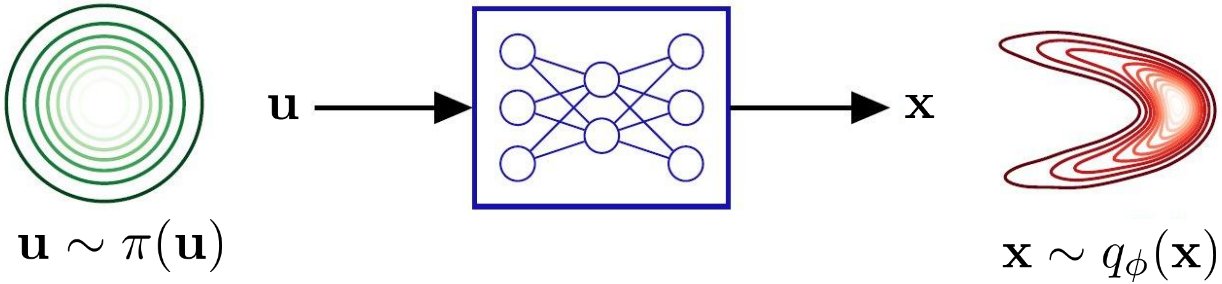
$$p(\mathbf{x}) = \prod_i p(x_i \,|\, x_1, \dots, x_{i-1})$$



$$\phi_i = \phi_i(x_1, \dots, x_{i-1})$$

$$q_\phi(\mathbf{x}) = \prod_i q_{\phi_i}(x_i)$$

Normalizing flows



Design $f_\phi : \mathbf{u} \mapsto \mathbf{x}$ so that:

- It's differentiable and 1-to-1
- f_ϕ^{-1} is tractable
- $\det \nabla f_\phi$ is tractable

Calculate model density by:

$$\mathbf{u} = f_\phi^{-1}(\mathbf{x})$$

$$q_\phi(\mathbf{x}) = \pi(\mathbf{u}) \left|\det \nabla f_\phi(\mathbf{u})\right|^{-1}$$

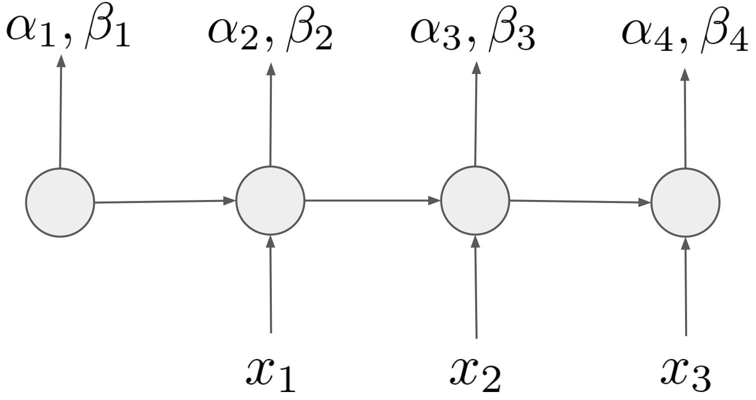
Normalizing flows are composable

$$\mathbf{u}_1 \longrightarrow \boxed{f_{\phi_1}} \longrightarrow \mathbf{u}_2 \longrightarrow \boxed{f_{\phi_2}} \longrightarrow \mathbf{u}_3$$

$$\mathbf{u}_k = f_{\phi_k}^{-1}(\mathbf{u}_{k+1})$$

$$q_\phi(\mathbf{x}) = \pi(\mathbf{u}_1) \prod_k \left|\det \nabla f_{\phi_k}(\mathbf{u}_k)\right|^{-1}$$

Masked Autoregressive Flow



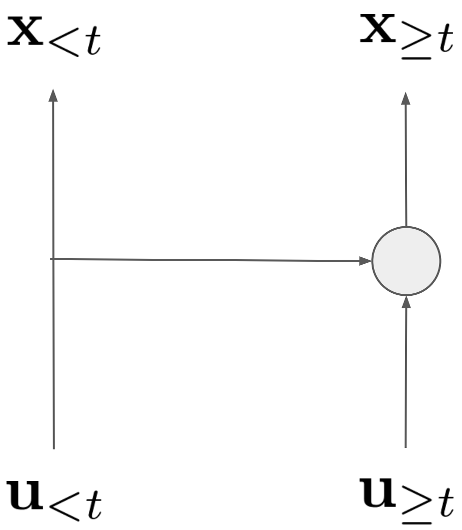
$$\alpha_i = \alpha_i(x_1, \ldots, x_{i-1})$$
$$\beta_i = \beta_i(x_1, \ldots, x_{i-1})$$

$$\mathbf{x} = f_\phi(\mathbf{u}) \;\Rightarrow$$
$$x_i = \alpha_i u_i + \beta_i \quad \forall i$$

$$\det \nabla f_\phi(\mathbf{u}) = \prod_i \alpha_i$$

Papamakarios, et al., *Masked Autoregressive Flow for Density Estimation*, NeurIPS 2017

Real NVP



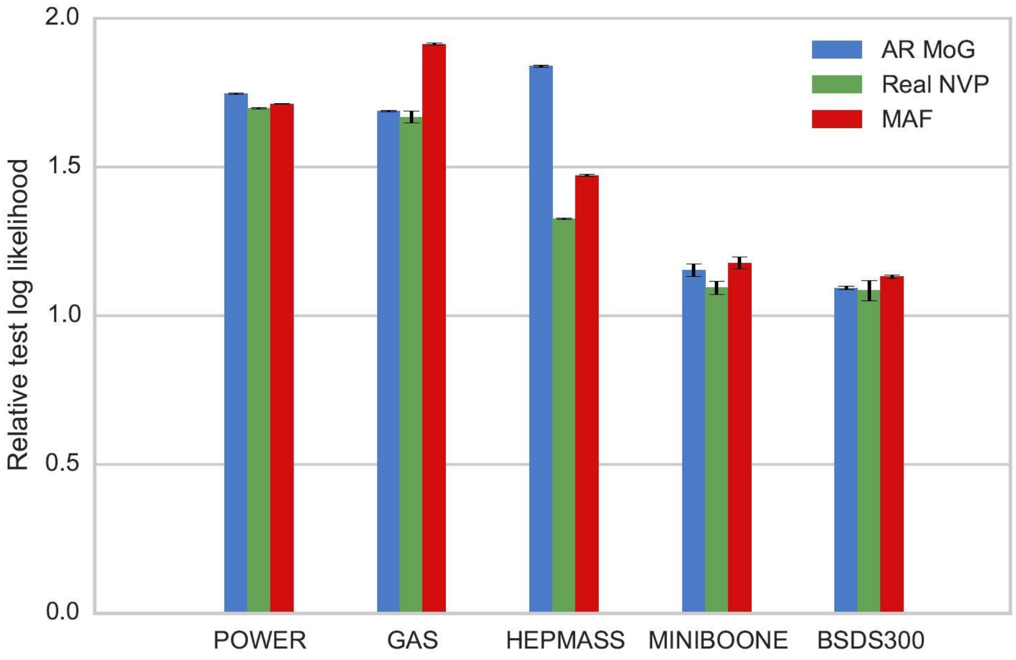
$$\alpha = \alpha(\mathbf{x}_{<t}) = \alpha(\mathbf{u}_{<t})$$
$$\beta = \beta(\mathbf{x}_{<t}) = \beta(\mathbf{u}_{<t})$$

$$\mathbf{x} = f_\phi(\mathbf{u}) \Rightarrow$$
$$\begin{cases} \mathbf{x}_{<t} = \mathbf{u}_{<t} \\ \mathbf{x}_{\geq t} = \alpha \odot \mathbf{u}_{\geq t} + \beta \end{cases}$$

$$\det \nabla f_\phi(\mathbf{u}) = \prod_i \alpha_i$$

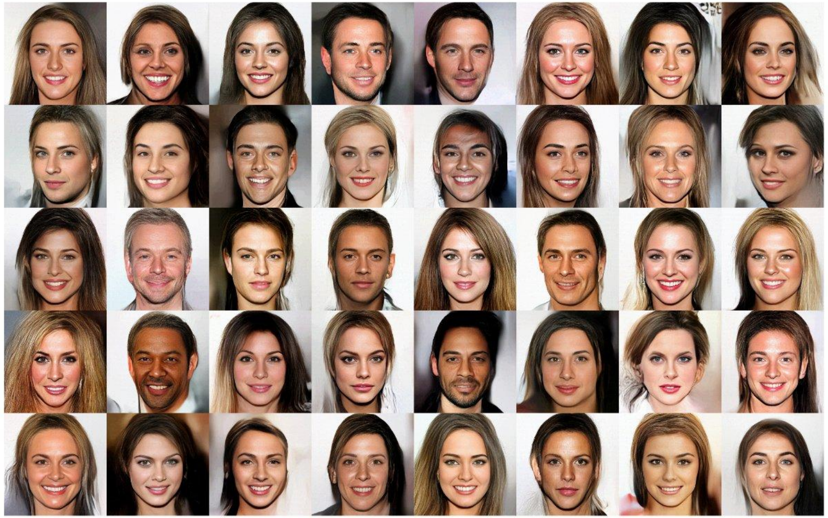
Dinh, et al., *Density Estimation with Real NVP*, ICLR 2017

Comparison: AR, MAF, Real NVP



Papamakarios, et al., *Masked Autoregressive Flow for Density Estimation*, NeurIPS 2017

Convolutional Real NVP for images



Kingma & Dhariwal, *Glow: Generative Flow with Invertible 1x1 Convolutions*, NeurIPS 2018

Neural density estimation: Summary

Goal: estimate density function from data

Neural density estimation:

- Represent density function with a neural network
- Maximize average log likelihood on data
- Train with backprop

Types of neural density models:

- Mixture models
- Autoregressive models
- Normalizing flows

Masked Autoregressive Flow & Real NVP

- Implemented in TensorFlow Probability

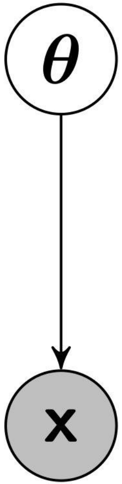
Bayesian inference

Observe data $\mathbf{x}_o$

Infer parameters θ

$$p(\theta \mid \mathbf{x} = \mathbf{x}_o) \propto p(\mathbf{x} = \mathbf{x}_o \mid \theta)\, p(\theta)$$

posterior likelihood prior



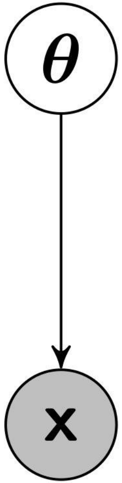
Likelihood-free inference

Observe data $\mathbf{x}_o$

Infer parameters θ

Likelihood $p(\mathbf{x} = \mathbf{x}_o \mid \theta)$ is not available

Can simulate $\mathbf{x}_n \sim p(\mathbf{x} \mid \theta)$ for any θ



A toy example from ecology: Lotka-Volterra model

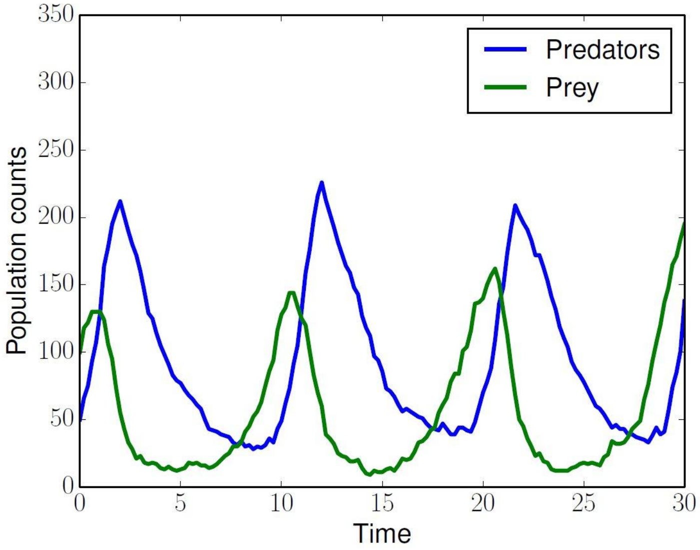
Parameters:

$$\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$$

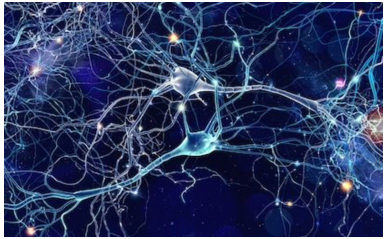
Model is a Markov Jump Process:

$$A = \text{Predators}$$
$$B = \text{Prey}$$

$$\frac{\partial A}{\partial t} = \theta_1 AB - \theta_2 A$$
$$\frac{\partial B}{\partial t} = \theta_3 B - \theta_4 AB$$

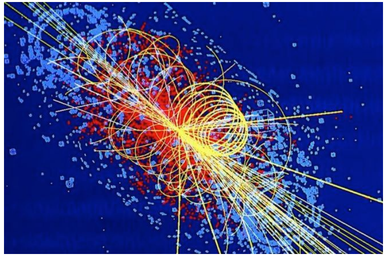


Many important applications

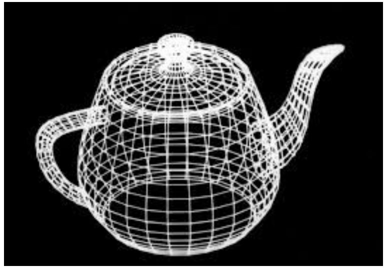


Neuroscience

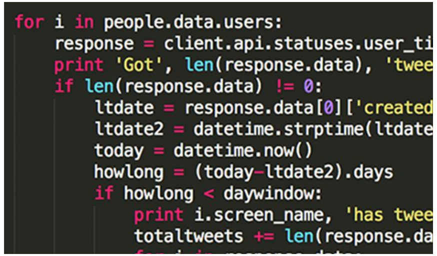
Astrophysics & cosmology



High-energy physics



Computer vision as inverse graphics



Probabilistic programming

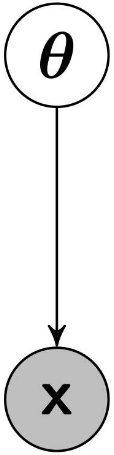
Approximate Bayesian Computation

Simulate $\theta_n \sim p(\theta)$
$\mathbf{x}_n \sim p(\mathbf{x} \,|\, \theta_n)$

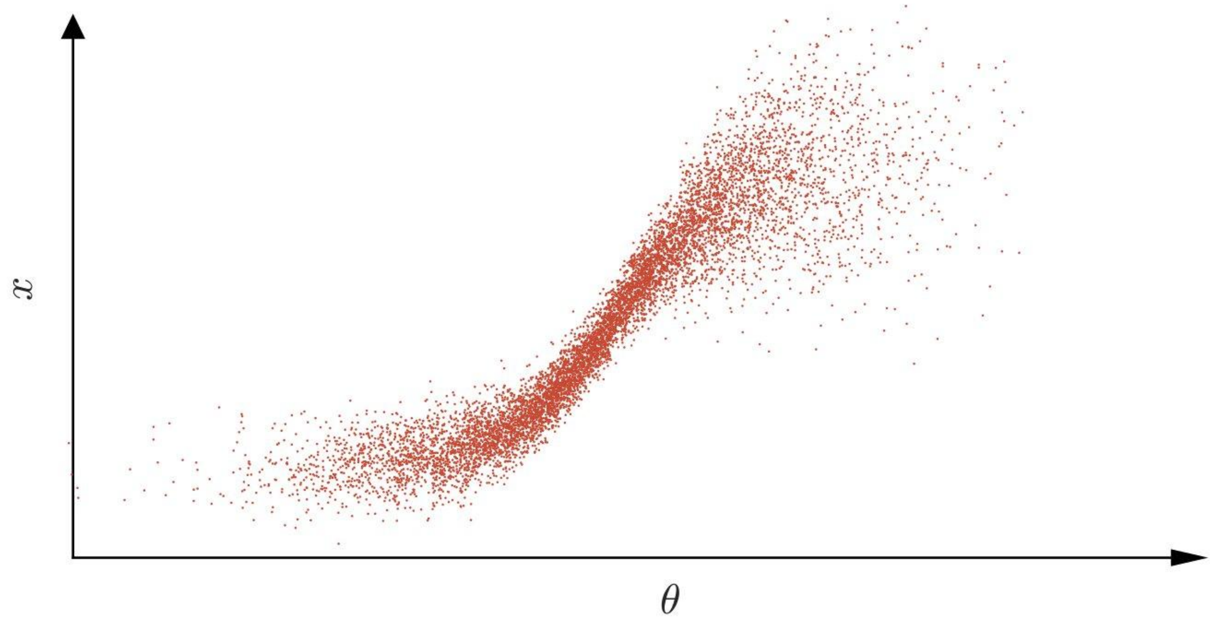
If $\mathbf{x}_n = \mathbf{x}_o$ then θ_n is a sample from $p(\theta \,|\, \mathbf{x} = \mathbf{x}_o)$

In practice:

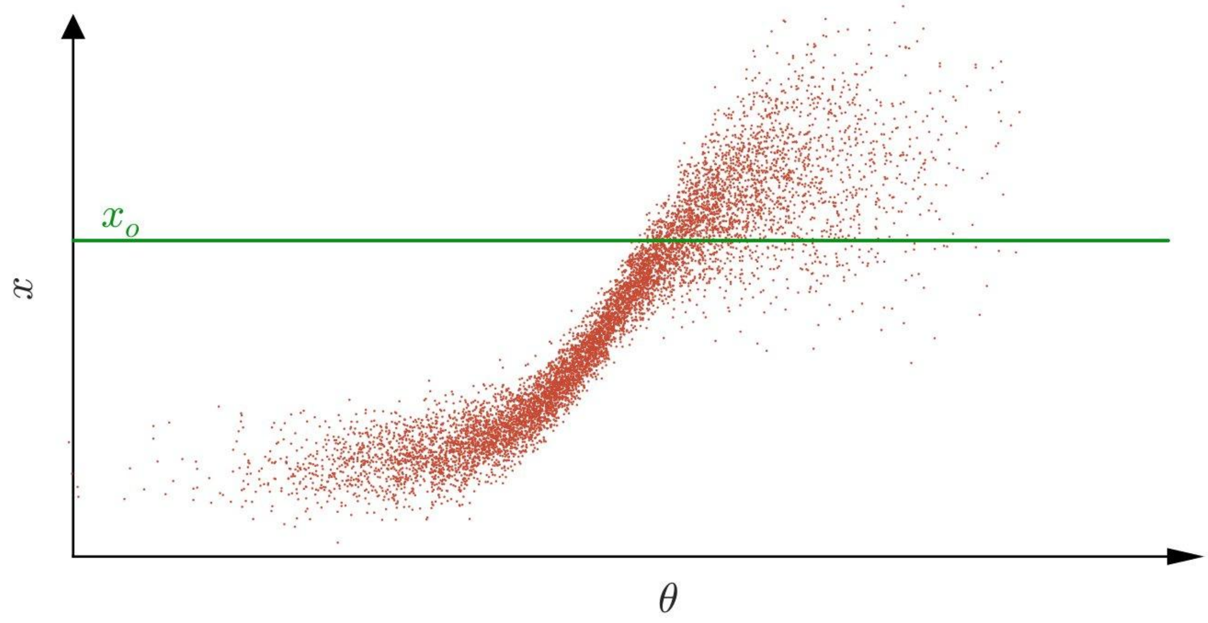
- Reduce data to summary statistics
- Accept whenever $\|\mathbf{x}_n - \mathbf{x}_o\| < \epsilon$
- Propose new parameters by 'perturbing' accepted parameters
  - MCMC-ABC, SMC-ABC, many others …



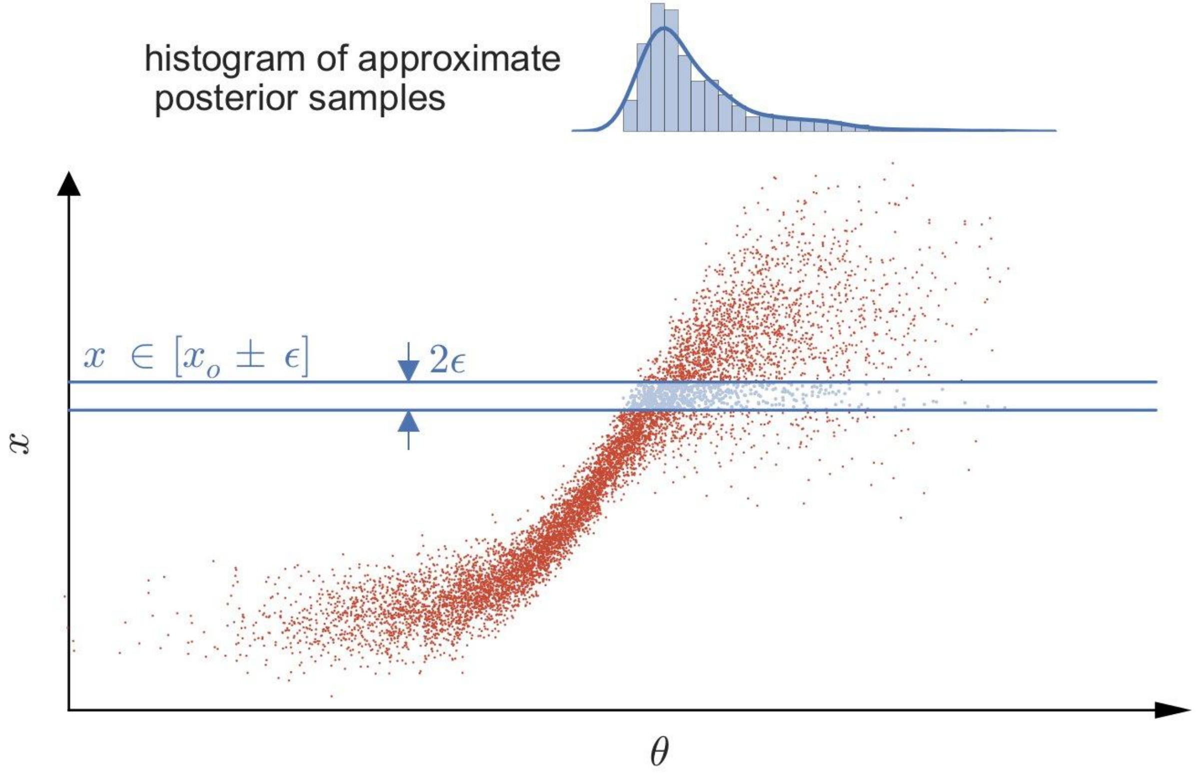
Approximate Bayesian Computation



Approximate Bayesian Computation



Approximate Bayesian Computation



Neural Likelihood

<u>Step 1</u>

Simulate $\theta_n \sim p(\theta) \quad \mathbf{x}_n \sim p(\mathbf{x} \,|\, \theta_n)$

Collect training data $\{(\theta_1, \mathbf{x}_1), \ldots, (\theta_N, \mathbf{x}_N)\}$

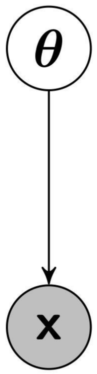
<u>Step 2</u>

Train a **conditional** neural density estimator $q_\phi(\mathbf{x} \,|\, \theta)$ on data

With enough data & capacity: $q_\phi(\mathbf{x} \,|\, \theta) \approx p(\mathbf{x} \,|\, \theta)$

<u>Step 3</u>

Sample from $\hat{p}(\theta \,|\, \mathbf{x} = \mathbf{x}_o) \propto q_\phi(\mathbf{x} = \mathbf{x}_o \,|\, \theta)\, p(\theta)$ (e.g. with MCMC)



Neural Likelihood with proposal

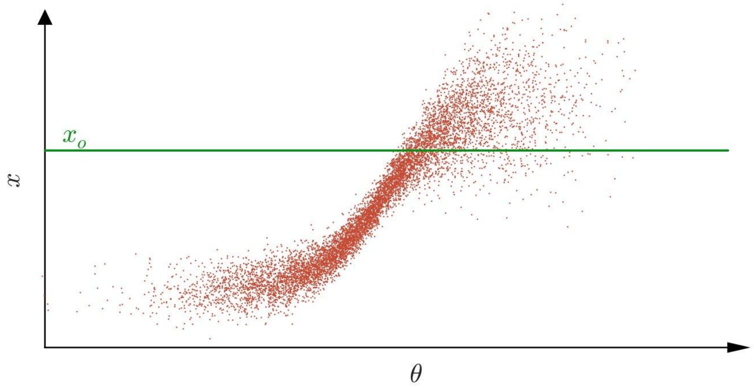
Neural Likelihood can be **inefficient**

- Wastes computation in irrelevant regions

Can use a proposal to control where likelihood approximation should be accurate

- Sample $\theta_n \sim \tilde{p}(\theta)$ instead of $p(\theta)$
- $q_\phi(\mathbf{x} \mid \theta)$ will be most accurate where $\tilde{p}(\theta)$ large

Empirically: Posterior $p(\theta \mid \mathbf{x} = \mathbf{x}_o)$ is a good proposal



Sequential Neural Likelihood

Initialize $\hat{p}(\theta \,|\, \mathbf{x} = \mathbf{x}_o) = p(\theta)$

<u>Step 1</u>

Sample $\theta_n \sim \hat{p}(\theta \,|\, \mathbf{x} = \mathbf{x}_o)$ e.g. with MCMC

Simulate $\mathbf{x}_n \sim p(\mathbf{x} \,|\, \theta_n)$

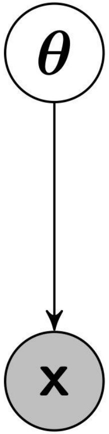
Append new data $\{(\theta_1, \mathbf{x}_1), \ldots, (\theta_N, \mathbf{x}_N)\}$ to all data

<u>Step 2</u>

Re-train a **conditional** neural density estimator $q_\phi(\mathbf{x} \,|\, \theta)$ on all data

Set $\hat{p}(\theta \,|\, \mathbf{x} = \mathbf{x}_o) \propto q_\phi(\mathbf{x} = \mathbf{x}_o \,|\, \theta)\, p(\theta)$

Go to step 1

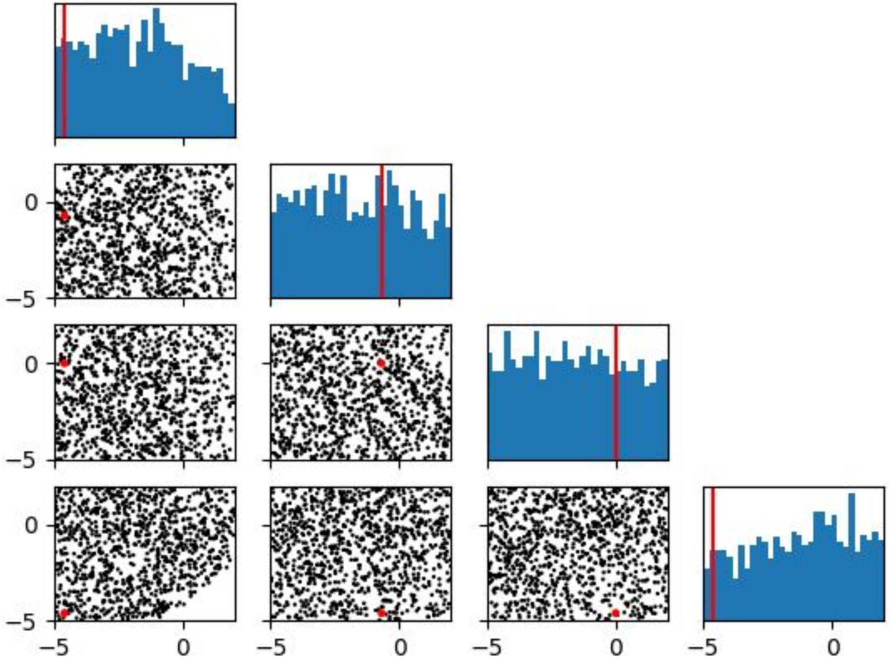


Papamakarios et al., *Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows*, AISTATS 2019

Sequential Neural Likelihood: Demo

Lotka-Volterra model

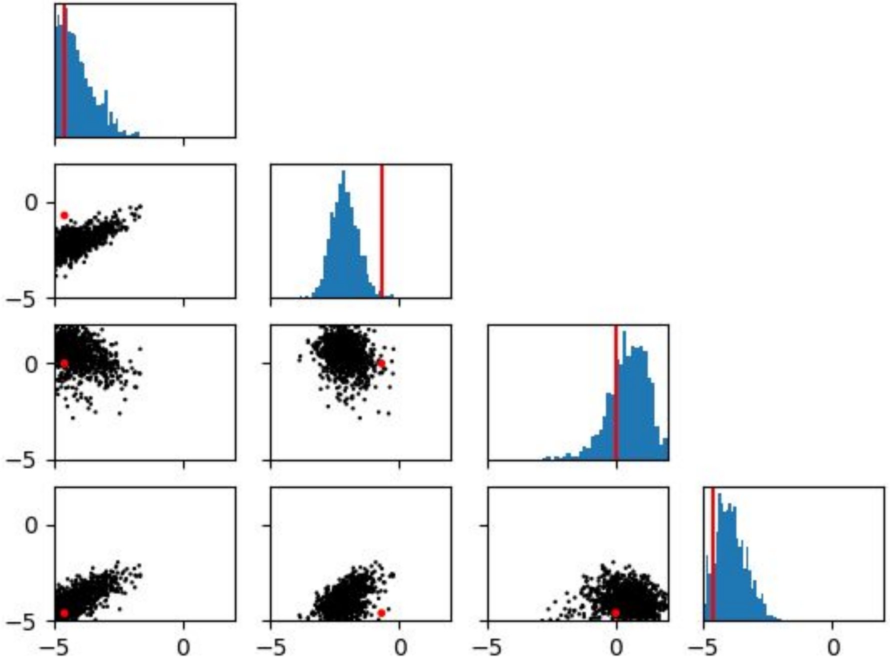
Round 1
(simulating from prior)



Sequential Neural Likelihood: Demo

Lotka-Volterra model

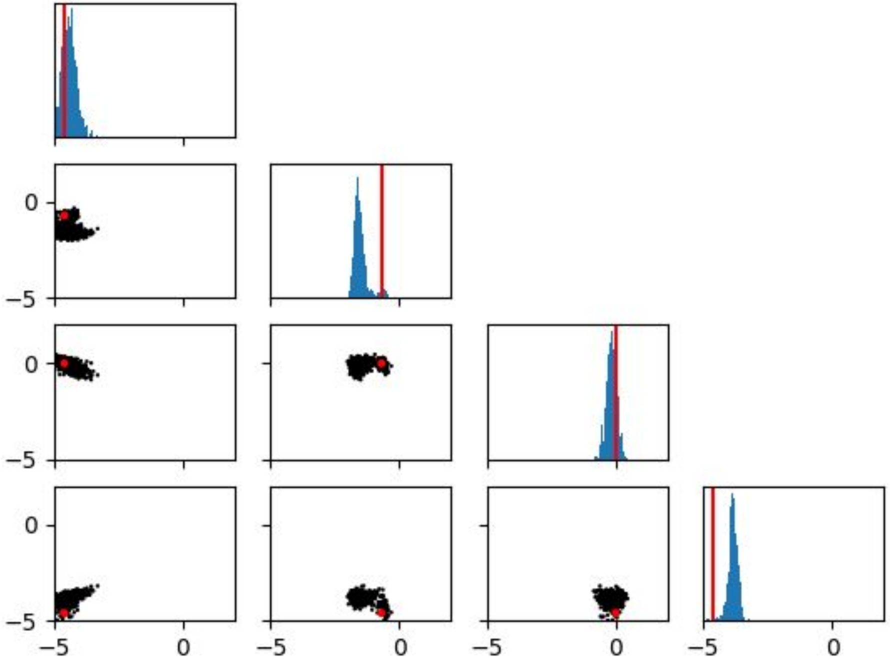
Round 2



Sequential Neural Likelihood: Demo

Lotka-Volterra model

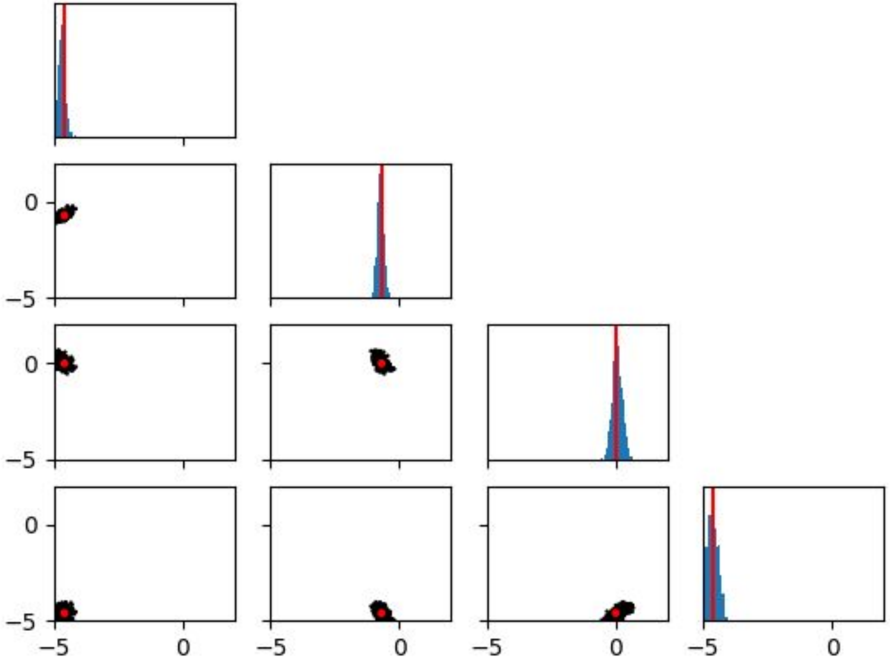
Round 3



Sequential Neural Likelihood: Demo

Lotka-Volterra model

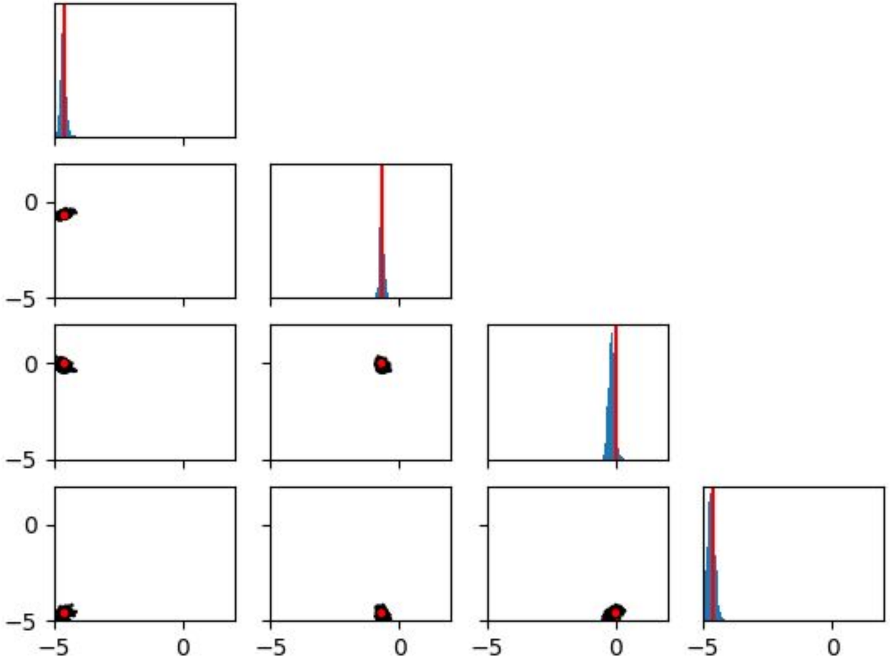
Round 4



Sequential Neural Likelihood: Demo

Lotka-Volterra model

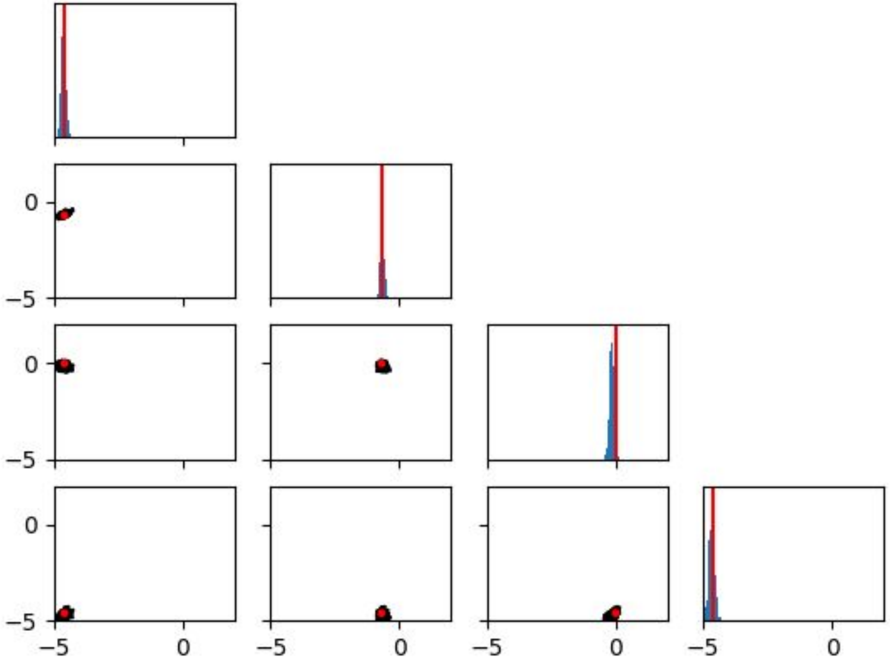
Round 5



Sequential Neural Likelihood: Demo

Lotka-Volterra model

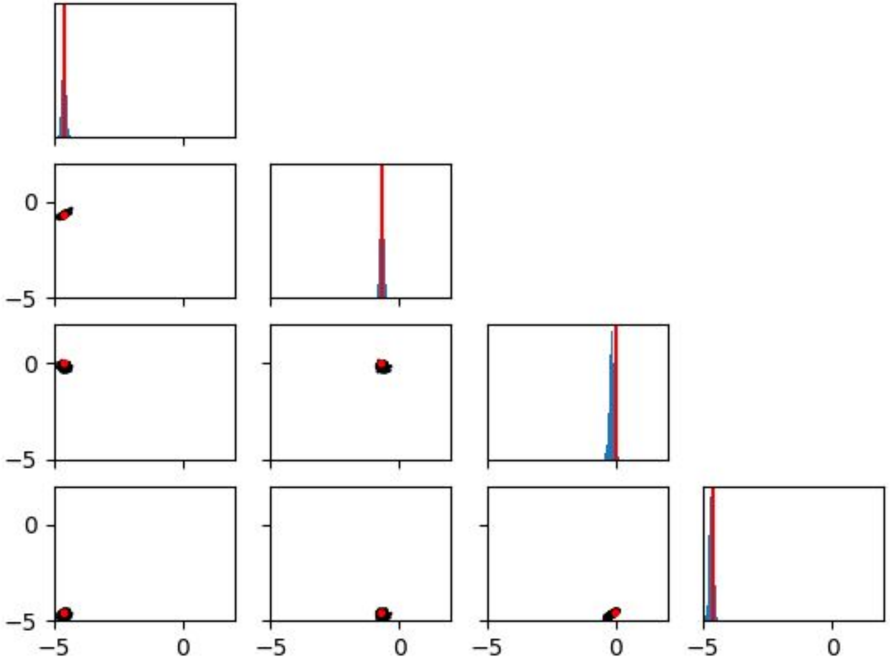
Round 6



Sequential Neural Likelihood: Demo

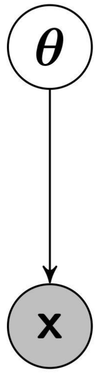
Lotka-Volterra model

Round 7



Neural Posterior Estimation

Instead of estimating the likelihood, we can train $q_\phi(\theta \,|\, \mathbf{x})$
on simulated data $\{(\theta_1, \mathbf{x}_1), \ldots, (\theta_N, \mathbf{x}_N)\}$
and estimate the posterior directly



Then inference is as simple as $\hat{p}(\theta \,|\, \mathbf{x} = \mathbf{x}_o) = q_\phi(\theta \,|\, \mathbf{x} = \mathbf{x}_o)$
→ no MCMC needed

However:

If we propose parameters from proposal $\tilde{p}(\theta)$ instead of prior $p(\theta)$
then $q_\phi(\theta \,|\, \mathbf{x}) \propto p(\mathbf{x} \,|\, \theta)\, \tilde{p}(\theta)$ i.e. estimated posterior will be biased

Sequential Neural Posterior Estimation

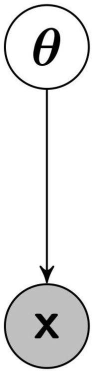
To enable sequential estimation of posterior, we need to correct for sampling from the “wrong” prior

Two methods for Sequential Neural Posterior Estimation

SNPE-A: Estimate posterior by analytically adjusting $q_\phi(\theta \,|\, \mathbf{x})$

$$\hat{p}(\theta \,|\, \mathbf{x} = \mathbf{x}_o) \propto \frac{p(\theta)}{\tilde{p}(\theta)} \, q_\phi(\theta \,|\, \mathbf{x} = \mathbf{x}_o)$$

- Neural density estimator is restricted to a conditional Gaussian mixture model
- Fails if division by $\tilde{p}(\theta)$ yields improper posterior



Papamakarios & Murray, *Fast ε-free Inference of Simulation Models with Bayesian Conditional Density Estimation*, NeurIPS 2016

Sequential Neural Posterior Estimation

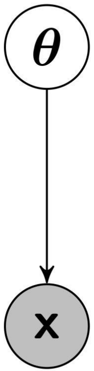
To enable sequential estimation of posterior, we need to correct for sampling from the "wrong" prior

Two methods for Sequential Neural Posterior Estimation

SNPE-B: Importance re-weight samples when training $q_\phi(\theta \,|\, \mathbf{x})$

$$L(\phi) = \frac{1}{N} \sum_{n=1}^{N} \frac{p(\theta_n)}{\tilde{p}(\theta_n)} \log q_\phi(\theta_n \,|\, \mathbf{x}_n)$$

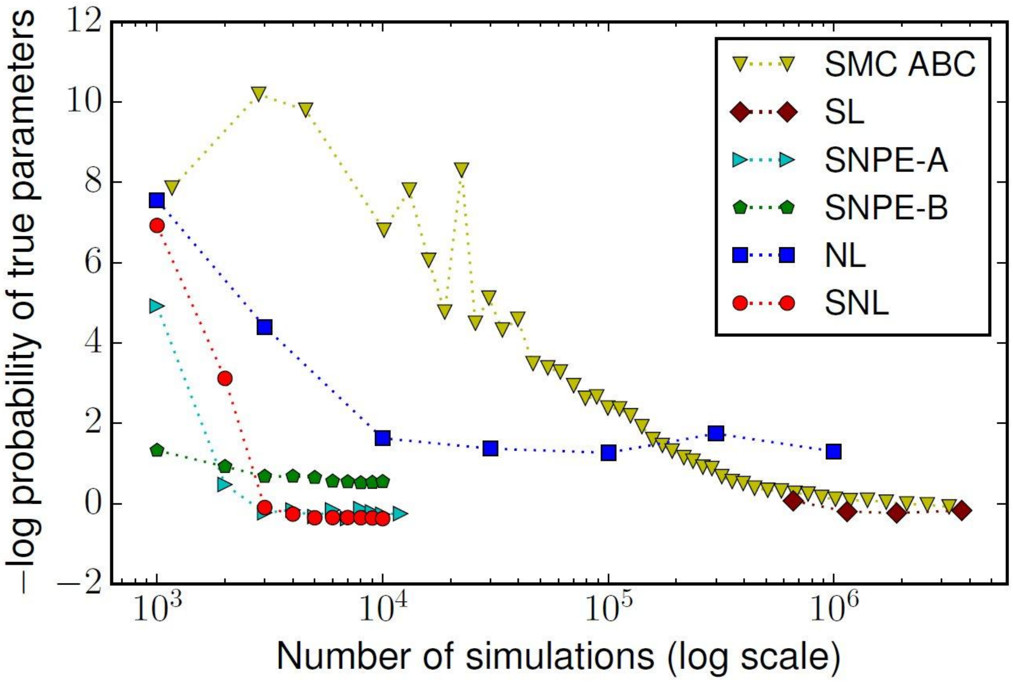
- $q_\phi(\theta \,|\, \mathbf{x})$ estimates the posterior correctly
- Training may become less stable → high-variance gradients



Lueckmann et al., *Flexible statistical inference for mechanistic models of neural dynamics*, NeurIPS 2017

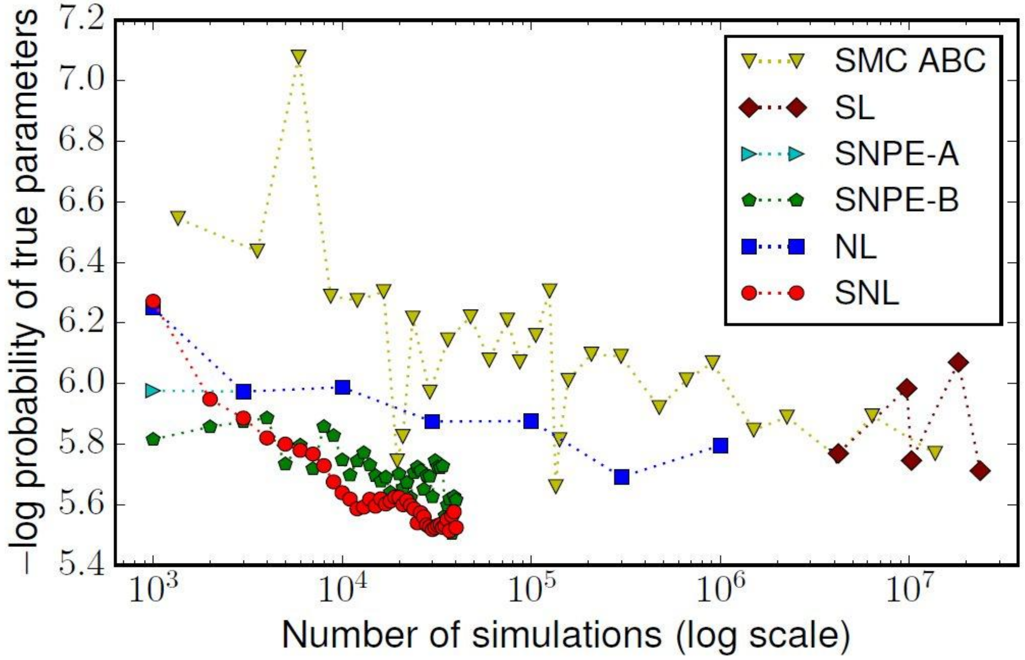
Results: Accuracy vs simulation cost

Lotka-Volterra
population model



Results: Accuracy vs simulation cost

Hodgkin-Huxley
neuron model



Diagnostics: Simulation-based calibration

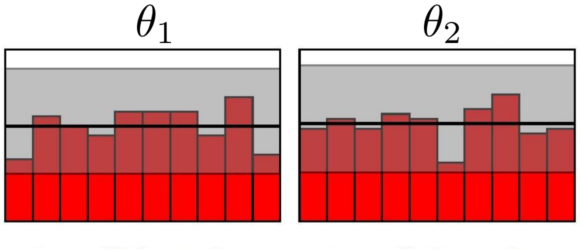
Draw “true parameters” from prior

Run inference, obtain posterior

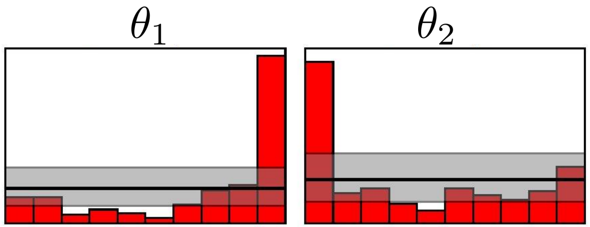
Draw K i.i.d. samples from posterior and calculate the rank statistic {0, 1, … , K} of true parameters

Repeat many times, plot histogram of all rank statistics

If inference calibrated → histogram is uniform



Well calibrated



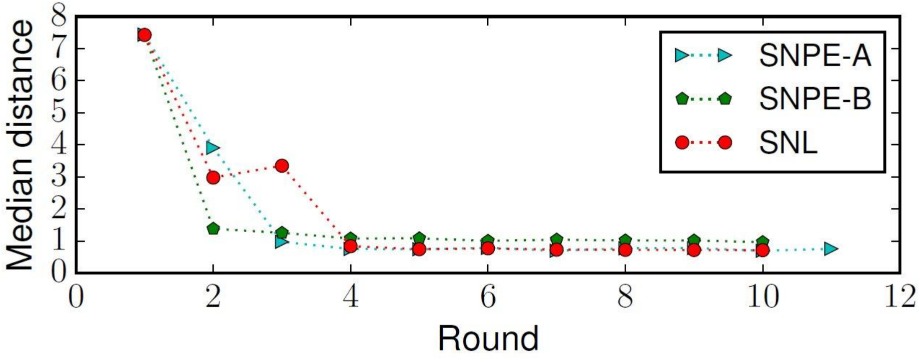
Badly calibrated

Diagnostics: Distance to observed data

In each round, calculate the median distance of simulated data to observed data $\mathbf{x}_o$

Can be used to assess convergence

Shows that posterior zones in relevant parameter region



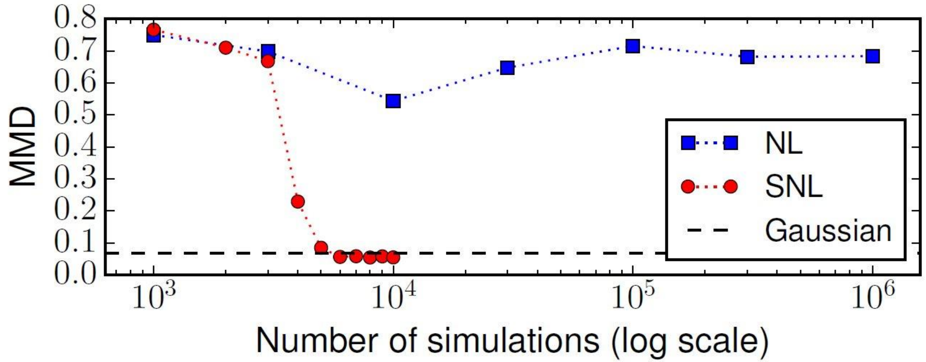
Diagnostics: Likelihood goodness-of-fit

For e.g. true parameters θ^*:

Simulate data from $p(\mathbf{x} \mid \theta = \theta^*)$

Sample data from $q_\phi(\mathbf{x} \mid \theta = \theta^*)$

Compare two datasets using e.g.
Maximum Mean Discrepancy



Summary

Two ideas for likelihood-free inference

Neural density estimation:

- Estimate likelihood / posterior from simulated data
- Use state-of-the-art neural density estimators

Sequential inference:

- Guide simulations by using so-far posterior as proposal
- Can lead to orders-of-magnitude savings in simulation cost