Inverting the Beamline

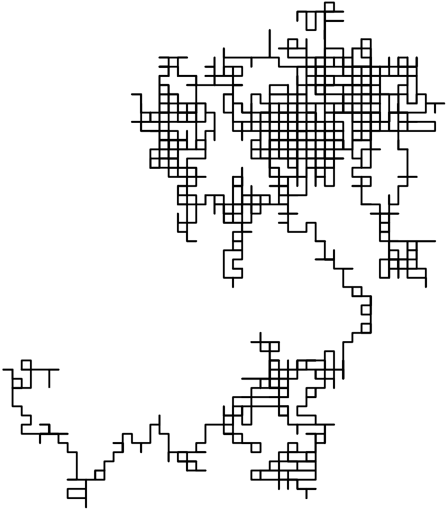
a random walk of simulation-based inference using machine learning

Peter Steinbach

HZDR / IKTP Dresden, June 10, 2021

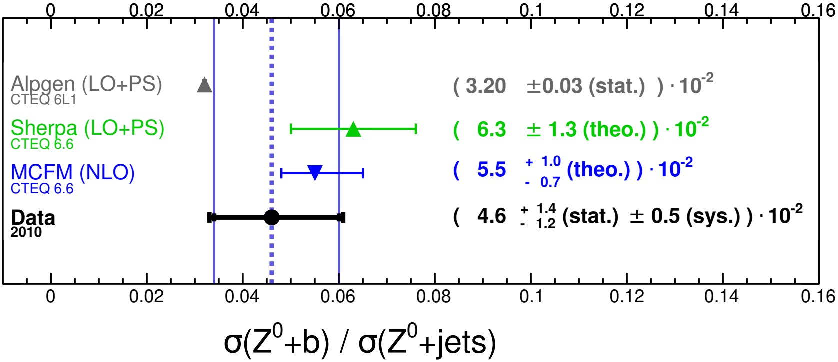
Some context first

A disclaimer



(anything presented below is the result of a long random walk)

whoami

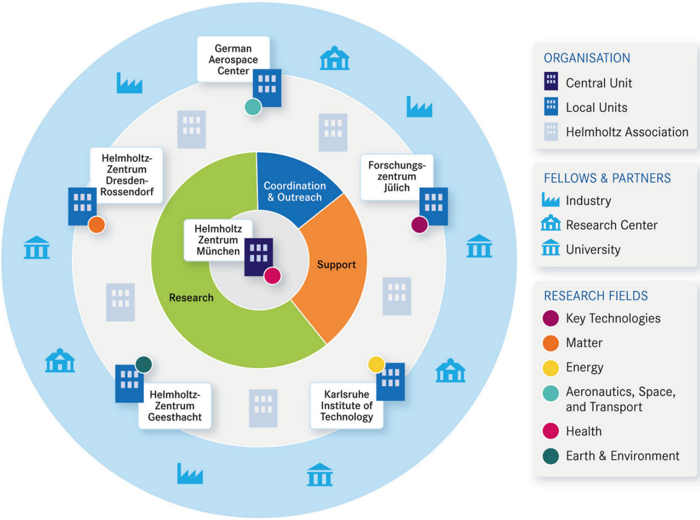


until 2012 PhD on $pp \to Z^0 + b$ cross-section at ATLAS (LHC, CERN)

2012-2019 HPC developer and Scientific Software Engineer at Scionics/MPI CBG

since 2019 Helmholtz AI team lead for Matter consulting

Helmholtz AI

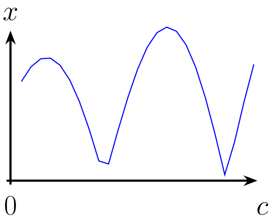


- one central, five local units
- each unit: research + consulting team
- since 2019 for 7+X years
- consulting team: **collaboration-as-a-service**
- more details: helmholtz.ai

Why are we doing this?

A common situation with simulations

$$\vec{\vartheta} = \begin{pmatrix} 42. \\ 12. \\ 1. \\ -0.8 \\ 2. \\ 320 \end{pmatrix} \xrightarrow[\vec{x} = f_{sim}(\vec{\vartheta})]{\text{simulation}}$$



$$\begin{pmatrix} .50 \\ .60 \\ .67 \\ \dots \\ \dots \\ \dots \\ .40 \\ .58 \\ .75 \end{pmatrix} = \vec{x}$$

- simulations used in many domains (physics, biology/medicine, chemistry, epidemiology, ...)
- approaches to simulations vary (mechanistic, agent based, distribution based, ...)
- simulations can be computationally challenging
- here: simulations = **forward process**

Inversion of Simulations?

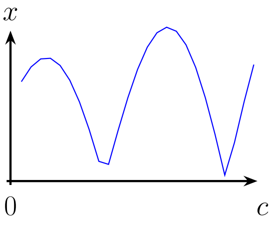
$$\vec{\vartheta} = \begin{pmatrix} 42. \\ 12. \\ 1. \\ -0.8 \\ 2. \\ 320 \end{pmatrix}$$

simulation

$$\vec{x} = f_{sim}(\vec{\vartheta})$$

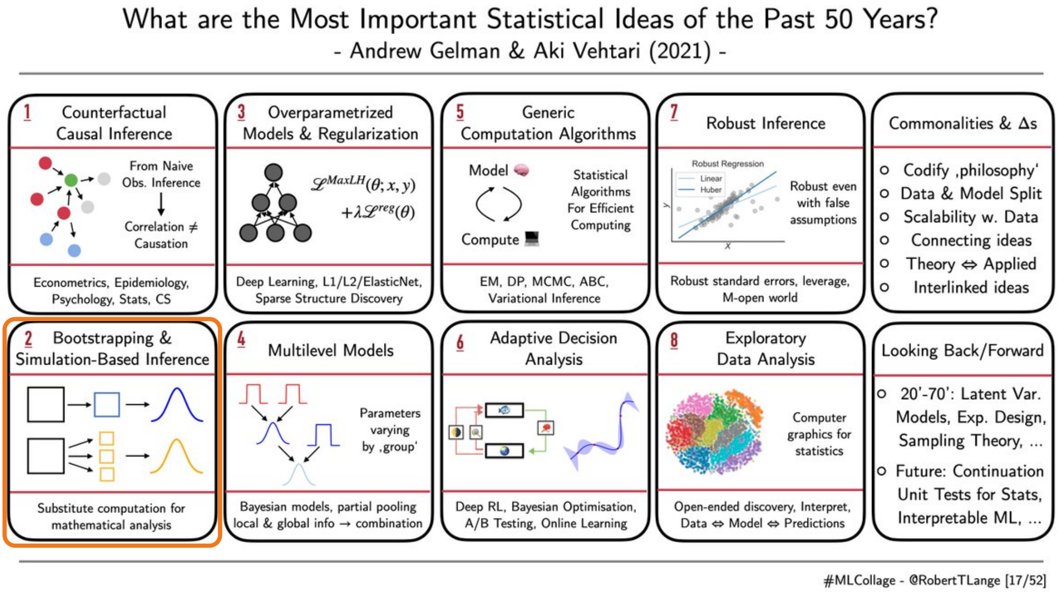
inversion?

$$\hat{\vec{\vartheta}} = f_{sim}^{-1}(\vec{x})$$



$$\begin{pmatrix} .50 \\ .60 \\ .67 \\ \ldots \\ \ldots \\ \ldots \\ .40 \\ .58 \\ .75 \end{pmatrix} = \vec{x}$$

- inverse process hard to do (if at all tried)
- often, only single observables “fitted”
- simulations updated based on singular observables
- considerable human tuning involved (heuristics)



Simulation-based inference discovered recently

(Cranmer et al, 2020, Gelman & Vehtari, 2021)

Bayes Law

$$p(\vec{\vartheta}|\vec{x}) = \frac{p(\vec{x}|\vec{\vartheta}) \cdot p(\vec{\vartheta})}{\int p(\vec{x}|\vec{\vartheta})p(\vec{\vartheta})d\vec{\vartheta}}$$

Bayes Law

$$p(\vec{\vartheta}|\vec{x}) = \frac{\overbrace{p(\vec{x}|\vec{\vartheta})}^{\text{likelihood}} \cdot p(\vec{\vartheta})}{\int p(\vec{x}|\vec{\vartheta})p(\vec{\vartheta})d\vec{\vartheta}}$$

Bayes Law

$$p(\vec{\vartheta}|\vec{x}) = \frac{\overbrace{p(\vec{x}|\vec{\vartheta})}^{\text{likelihood}} \cdot \overbrace{p(\vec{\vartheta})}^{\text{prior}}}{\int p(\vec{x}|\vec{\vartheta}) p(\vec{\vartheta}) d\vec{\vartheta}}$$

Bayes Law

$$\underbrace{p(\vec{\vartheta}|\vec{x})}_{\text{posterior}} = \frac{\overbrace{p(\vec{x}|\vec{\vartheta})}^{\text{likelihood}} \cdot \overbrace{p(\vec{\vartheta})}^{\text{prior}}}{\int p(\vec{x}|\vec{\vartheta})p(\vec{\vartheta})d\vec{\vartheta}}$$

Bayes Law

$$\underbrace{p(\vec{\vartheta}|\vec{x})}_{\text{posterior}} = \frac{\overbrace{p(\vec{x}|\vec{\vartheta})}^{\text{likelihood}} \cdot \overbrace{p(\vec{\vartheta})}^{\text{prior}}}{\int p(\vec{x}|\vec{\vartheta})p(\vec{\vartheta})d\vec{\vartheta}}$$

- likelihood $p(\vec{x}|\vec{\vartheta})$ provided by (forward) simulation
- prior $p(\vec{\vartheta})$ given by how we sample the simulation parameters $\vec{\vartheta}$

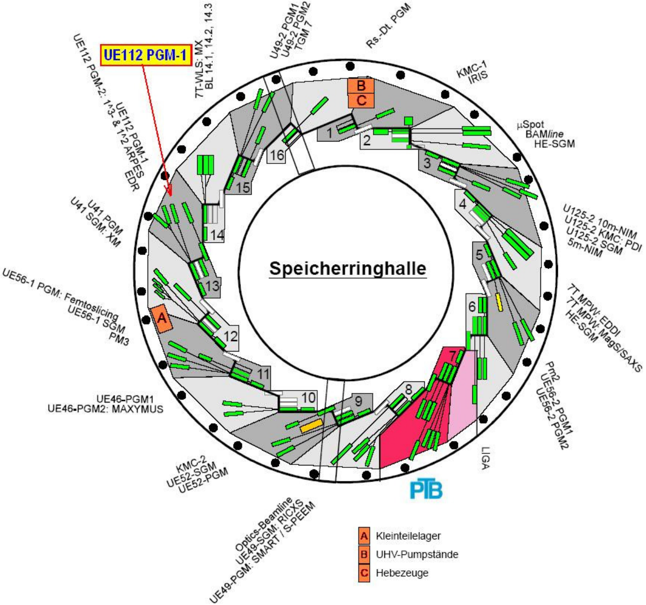
Bayes Law

$$\underbrace{p(\vec{\vartheta}|\vec{x})}_{\text{posterior}} = \frac{\overbrace{p(\vec{x}|\vec{\vartheta})}^{\text{likelihood}} \cdot \overbrace{p(\vec{\vartheta})}^{\text{prior}}}{\int p(\vec{x}|\vec{\vartheta})p(\vec{\vartheta})d\vec{\vartheta}}$$

- likelihood $p(\vec{x}|\vec{\vartheta})$ provided by (forward) simulation
- prior $p(\vec{\vartheta})$ given by how we sample the simulation parameters $\vec{\vartheta}$

Goal: predict posterior to the best of our abilities!

Why do I care?



Scientific Question

Given a beam profile, what were the beamline optics parameters that likely produced it?

- beamline UE112 PGM-1 at BESSY
- beam characteristics fixed at electron storage ring outlet
- forward simulations by rayUI

Normalizing Flows and INNs

Goal: Learn Invertible Mapping

basic assumption

f ... invertible mapping X ... data distribution Z ... generative distribution

$$f : X \rightarrow Z$$

Goal: Learn Invertible Mapping

basic assumption

f ... invertible mapping X ... data distribution Z ... generative distribution

$$f : X \rightarrow Z$$

What we get ...

$$f^{-1}(z) = \hat{x}_{gen}$$

- sample z from Z with $p_Z(z)$ (Z is a normal Gaussian distribution)
- "obtain" inverse transformation f^{-1} from learned forward function f

- **Normalizing Flows are generative models!**

Change of Variables Trick, part 1/2

- given a random variable $X \in \mathbb{R}^d$ and $Z \in \mathbb{R}^d$ of d dimensions
- given an invertible function $f : X \rightarrow Z$ and $z = f(x)$

For interval β over X, there has to be β' over Z such that

$$\int_{\beta} p_X dx = \int_{\beta'} p_Z dz$$

Change of Variables Trick, part 1/2

- given a random variable $X \in \mathbb{R}^d$ and $Z \in \mathbb{R}^d$ of d dimensions
- given an invertible function $f : X \rightarrow Z$ and $z = f(x)$

For interval β over X, there has to be β' over Z such that

$$\int_{\beta} p_X dx = \int_{\beta'} p_Z dz$$

$$p_X(x)dx = p_Z(z)dz$$

Change of Variables Trick, part 2/2

$$p_X(x)dx = p_Z(z)dz$$

Change of Variables Trick, part 2/2

$$p_X(x)dx = p_Z(z)dz$$

$$p_X(x) = |\frac{dz}{dx}|p_Z(z)$$

Change of Variables Trick, part 2/2

$$p_X(x)dx = p_Z(z)dz$$

$$p_X(x) = |\frac{dz}{dx}|p_Z(z)$$

$$p_X(x) = |\frac{df(x)}{dx}|p_Z(f(x))$$

Change of Variables Trick, part 2/2

$$p_X(x)dx = p_Z(z)dz$$

$$p_X(x) = |\frac{dz}{dx}|p_Z(z)$$

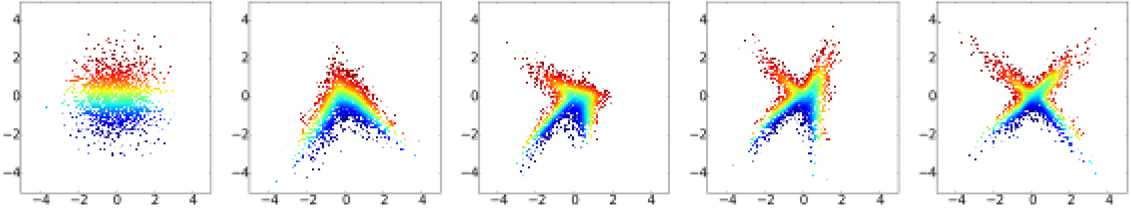
$$p_X(x) = |\frac{df(x)}{dx}|p_Z(f(x))$$

For $d > 1$ dimensions

- given a random variable $X \in \mathbb{R}^d$ and $Z \in \mathbb{R}^d$ of d dimensions
- given an invertible function $f : X \rightarrow Z$ and $z = f(x)$

$$p_X(x) = |det(\frac{df(x)}{dx})|p_Z(f(x))$$

Achievement: from x to z and back



from G. Papamakarios et al, “Normalizing Flows for Probabilistic Modeling and Inference”, arXiv:1912.02762, 2019.

Quiz!

Why are normalizing flows called normalizing flows?

Quiz!

Why are normalizing flows called normalizing flows?

They *normalize x into z*!

- $X \in \mathbb{R}^d$ and $Z \in \mathbb{R}^d$ of d dimensions
- sample z from Z with $p_Z(z)$, Z being a (multivariate) normal Gaussian distribution

Take Aways

- NFs infer exact latent-variable values z!

Take Aways

- NFs infer exact latent-variable values z!
  - VAEs infers a distribution over latent-variable values

Take Aways

- NFs infer exact latent-variable values z!
  - VAEs infers a distribution over latent-variable values
  - GANs do not have a latent-distribution

Take Aways

- NFs infer exact latent-variable values z!
  - VAEs infers a distribution over latent-variable values
  - GANs do not have a latent-distribution
- NFs optimize the exact log-likelihood of the data, $log(p_X(x))$

Take Aways

- NFs infer exact latent-variable values z!
  - VAEs infers a distribution over latent-variable values
  - GANs do not have a latent-distribution
- NFs optimize the exact log-likelihood of the data, $log(p_X(x))$
  - VAEs optimize the lower bound (ELBO)

Take Aways

- NFs infer exact latent-variable values z!
  - VAEs infers a distribution over latent-variable values
  - GANs do not have a latent-distribution
- NFs optimize the exact log-likelihood of the data, $log(p_X(x))$
  - VAEs optimize the lower bound (ELBO)
  - GANs learn to fool a discriminator network

Take Aways

- NFs infer exact latent-variable values z!
  - VAEs infers a distribution over latent-variable values
  - GANs do not have a latent-distribution
- NFs optimize the exact log-likelihood of the data, $log(p_X(x))$
  - VAEs optimize the lower bound (ELBO)
  - GANs learn to fool a discriminator network
- **requirements of invertibility and efficient Jacobian calculations restrict model architecture**

Conditional Invertible Neural Networks

arxiv:1907.02392, Code on github

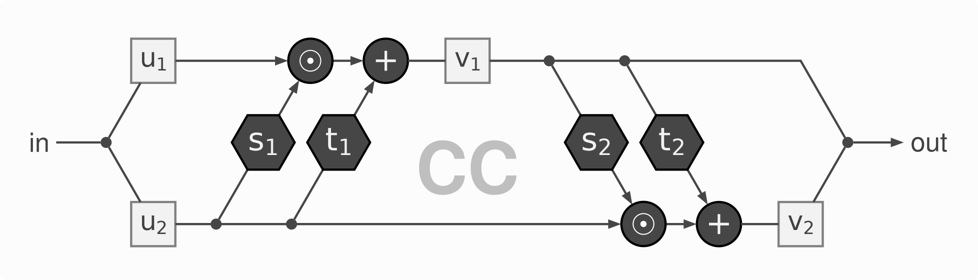
Guided Image Generation with Conditional Invertible Neural Networks

Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, Ullrich Köthe
Visual Learning Lab Heidelberg

Abstract

In this work, we address the task of natural image generation guided by a conditioning input. We introduce a new architecture called conditional invertible neural network (cINN). The cINN combines the purely generative INN model with an unconstrained feed-forward network, which efficiently preprocesses the conditioning input into useful features. All parameters of the cINN are jointly optimized with a stable, maximum likelihood-based training procedure. By construction, the cINN does not experience mode collapse and generates diverse samples, in contrast to e.g. cGANs. At the same time our model produces sharp images since no reconstruction loss is required, in contrast to e.g. VAEs. We demonstrate these properties for the tasks of MNIST digit generation and image colorization. Furthermore, we take advantage of our bi-directional cINN architecture to explore and manipulate emergent properties of the latent space, such as changing

INNs



edited from arxiv:1907.02392
(inspired by arxiv:1808.04730)

Forward (f)

$$v_1 = u_1 \otimes exp(s_1(u_2)) + t_1(u_2)$$

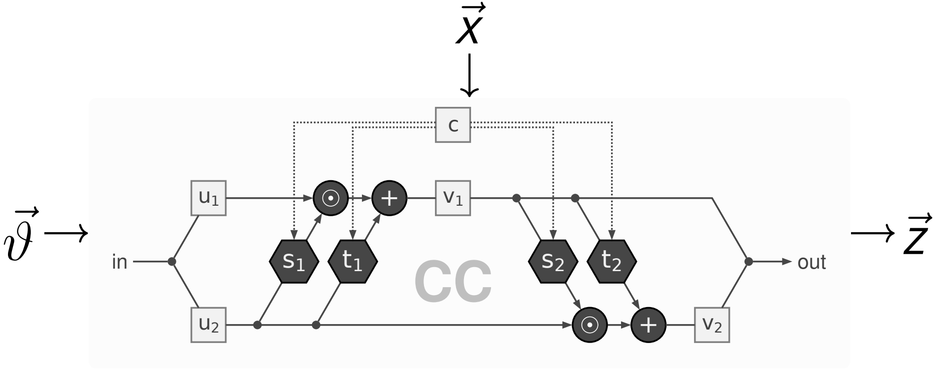
$$v_2 = u_2 \otimes exp(s_2(u_1)) + t_2(u_1)$$

Backward (f^{-1})

$$u_2 = (v_2 - t_2(v_1)) \oslash exp(s_2(v_1))$$

$$u_1 = (v_1 - t_1(u_2)) \oslash exp(s_1(u_2))$$

Conditioning



from arxiv:1907.02392

- this **does not compromise invertibility**
- *concatenate output of conditioning network c to inputs of s_j and t_j*, e.g.

$$s_1(u_2) \rightarrow s_1([u_2, c(u_2)])$$

cINN Loss from Maximum Likelihood Optimisation

$$\mathcal{L}_i = \mathbb{E}_i\Big[- \log(p(\vartheta_i|x_i))\Big]$$

cINN Loss from Maximum Likelihood Optimisation

$$\mathcal{L}_i = \mathbb{E}_i\Big[-\log(p(\vartheta_i|x_i))\Big]$$

- using $J_i = det(\frac{df_w}{d\vartheta}|_{\vartheta_i})$
- with $p(\vec{\vartheta}|\vec{x}) = |J_i|p_Z(z = f_w(\vec{\vartheta};\vec{x}))$ from change of variables
- p_z describes unit gaussian distribution, $\mathcal{N}(z|0, I) \approx exp(||-\frac{1}{2}z||_2^2)$

cINN Loss from Maximum Likelihood Optimisation

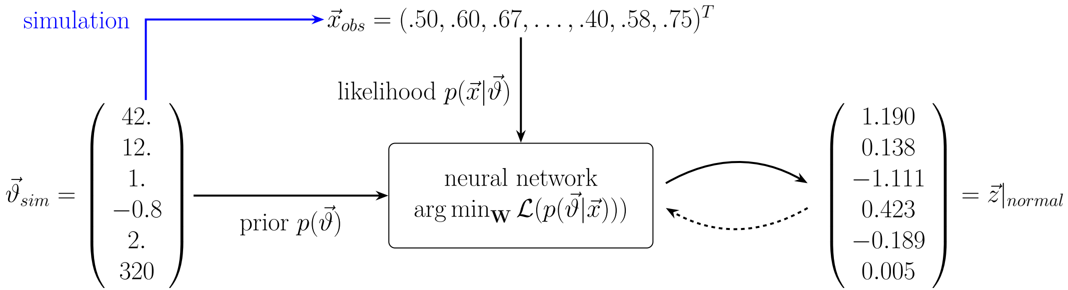
$$\mathcal{L}_i = \mathbb{E}_i\Big[-\log(p(\vartheta_i|x_i))\Big]$$

- using $J_i = det(\frac{df_w}{d\vartheta}|_{\vartheta_i})$
- with $p(\vec{\vartheta}|\vec{x}) = |J_i| p_Z(z = f_w(\vec{\vartheta};\vec{x}))$ from change of variables
- p_z describes unit gaussian distribution, $\mathcal{N}(z|0,I) \approx exp(||-\frac{1}{2}z||_2^2)$

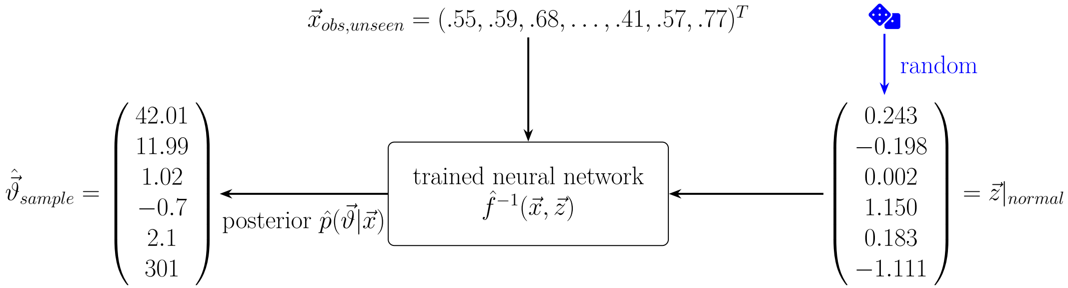
$$\mathcal{L} = \mathbb{E}_i\Big[-\frac{||f_w(\vec{\vartheta};c_u(\vec{x}))||_2^2}{2} - \log(J_i)\Big]$$

for complete derivation see Radev et al, 2020
(footnote: the paper starts from the Kullback-Leibler (KL) divergence, which is equivalent to a Maximum Likelihood Optimisation)

In Practice: Training



In Practice: Inference



Take Home Messages

- (c)INNs based on normalizing flows

Take Home Messages

- (c)INNs based on normalizing flows
- conditional INNs allow inversion of $\vec{\vartheta} \in \mathbb{R}^d \rightleftharpoons \vec{x} = f(\vec{\vartheta}) \in \mathbb{R}^k$, $(d \neq k)$

Take Home Messages

- (c)INNs based on normalizing flows
- conditional INNs allow inversion of
  $\vec{\vartheta} \in \mathbb{R}^d \rightleftharpoons \vec{x} = f(\vec{\vartheta}) \in \mathbb{R}^k$, $(d \neq k)$
- inversion *sensible on latent space* $z \in \mathbb{R}^d$

Take Home Messages

- (c)INNs based on normalizing flows
- conditional INNs allow inversion of
  $\vec{\vartheta} \in \mathbb{R}^d \rightleftharpoons \vec{x} = f(\vec{\vartheta}) \in \mathbb{R}^k$, $(d \neq k)$
- inversion *sensible on latent space* $z \in \mathbb{R}^d$
- promising avenue beyond/aside VAE and GANs

Take Home Messages

- (c)INNs based on normalizing flows
- conditional INNs allow inversion of
  $\vec{\vartheta} \in \mathbb{R}^d \rightleftharpoons \vec{x} = f(\vec{\vartheta}) \in \mathbb{R}^k$, $(d \neq k)$
- inversion *sensible on latent space* $z \in \mathbb{R}^d$
- promising avenue beyond/aside VAE and GANs

Take Home Messages

- (c)INNs based on normalizing flows
- conditional INNs allow inversion of
  $\vec{\vartheta} \in \mathbb{R}^d \rightleftharpoons \vec{x} = f(\vec{\vartheta}) \in \mathbb{R}^k$, $(d \neq k)$
- inversion *sensible on latent space* $z \in \mathbb{R}^d$
- promising avenue beyond/aside VAE and GANs

Quiz: What are the core assumptions for (c)INNs to work?

Take Home Messages

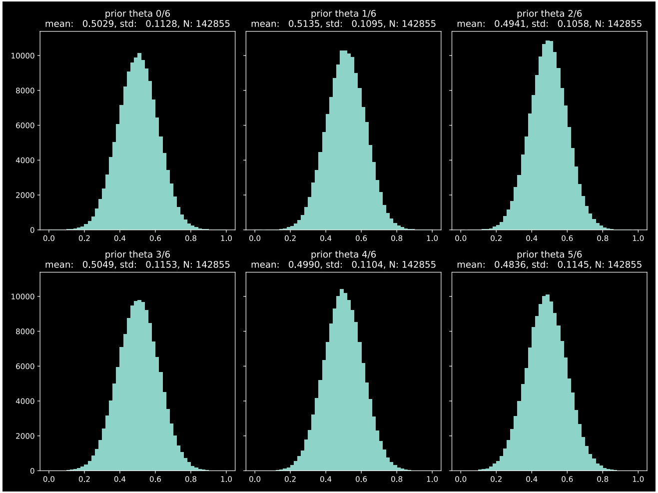
- (c)INNs based on normalizing flows
- conditional INNs allow inversion of $\vec{\vartheta} \in \mathbb{R}^d \rightleftharpoons \vec{x} = f(\vec{\vartheta}) \in \mathbb{R}^k$, $(d \neq k)$
- inversion *sensible on latent space* $z \in \mathbb{R}^d$
- promising avenue beyond/aside VAE and GANs

Quiz: What are the core assumptions for (c)INNs to work?

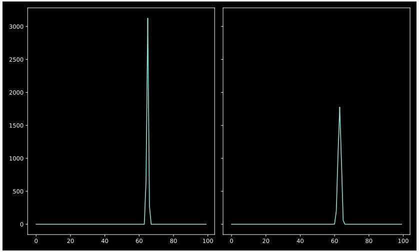
- $f : \vartheta \to x$ is invertible
- generative distribution z is drawn from a normal distribution
- tractable jacobian so that $det(J_f)$ can be computed

cINNs in the wild

My data

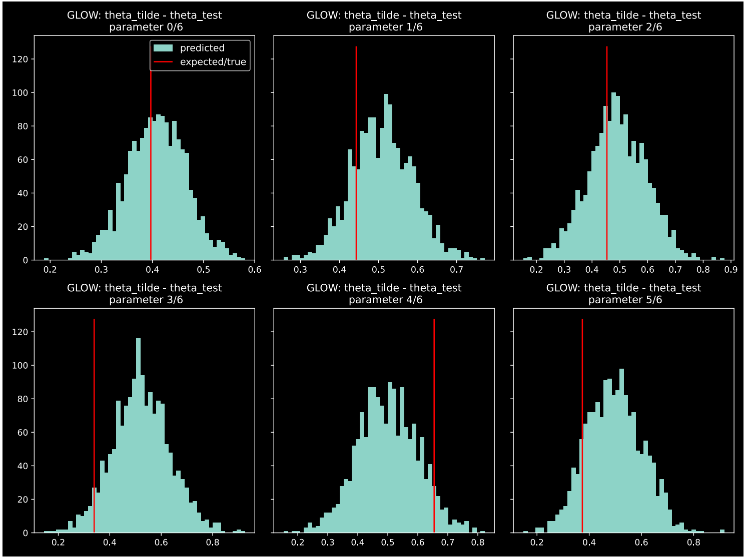


(forward) simulation parameters $\vec{\vartheta} \in \mathbb{R}^6$



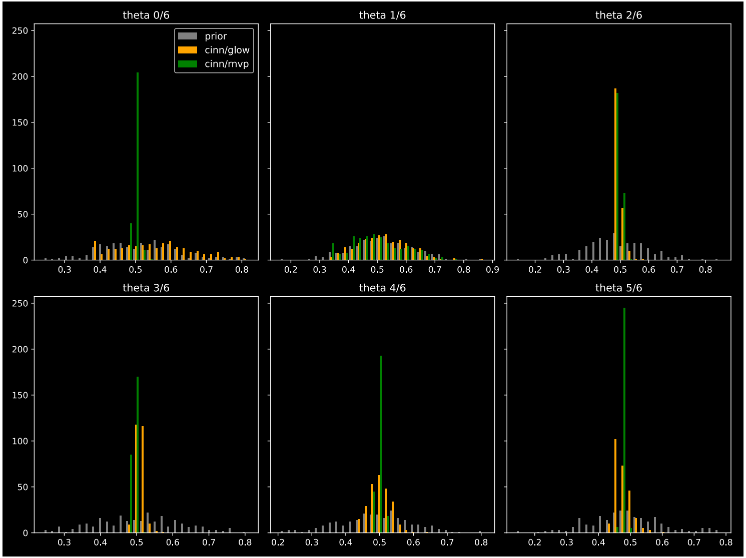
knife-edge scans to quantify beam quality at experimental station, $\vec{x} \in \mathbb{R}^{200}$

cINN Inference on the validation set



- cINN provides posterior that can be sampled
- extract Maximum a posteriori estimation (MAP) estimate by mean/median

cINN MAPs on the validation set



- training:
  - 30 epochs only
  - fixed arch: 8 layers
  - 256 units per dense layer
- inference: 256 draws per validation sample, MAP by mean
- good: posterior stays within prior support
- to improve: posterior misses out for some dimensions

What is going on?

core assumption(s)

- network f is sufficiently expressive
- as $N_{\text{simulations}} \to \infty$, network allows mapping of $\vec{x}$ onto $p(\vec{\vartheta}|\vec{x})$
- training dataset imposes the entire “truth” through implicit prior $p_{\text{simulation}}(\vec{\vartheta})$

What is going on?

core assumption(s)

- network f is sufficiently expressive
- as $N_{\text{simulations}} \rightarrow \infty$, network allows mapping of $\vec{x}$ onto $p(\vec{\vartheta}|\vec{x})$
- training dataset imposes the entire "truth" through implicit prior $p_{\text{simulation}}(\vec{\vartheta})$

Let's reconsider

- goal: infer $\vec{\vartheta}|\vec{x_o}$ on observation $\vec{x_o}$
- **but:** the global learned posterior may not be too informative at $p(\vec{\vartheta}|\vec{x_o})$ (as we fixed the prior)

What is going on?

core assumption(s)

- network f is sufficiently expressive
- as $N_{\text{simulations}} \to \infty$, network allows mapping of $\vec{x}$ onto $p(\vec{\vartheta}|\vec{x})$
- training dataset imposes the entire "truth" through implicit prior $p_{\text{simulation}}(\vec{\vartheta})$

Let's reconsider

- goal: infer $\vec{\vartheta}|\vec{x_o}$ on observation $\vec{x_o}$
- **but:** the global learned posterior may not be too informative at $p(\vec{\vartheta}|\vec{x_o})$ (as we fixed the prior)

sequential neural density estimation

Sequential neural density estimation

SNPE

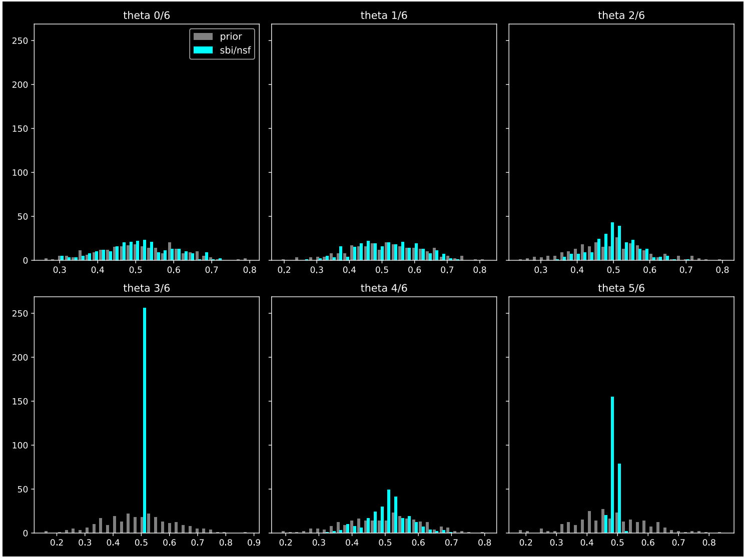
Algorithm 1 APT with per-round proposal updates

Input: simulator with (implicit) density $p(x|\theta)$, data x_o, prior $p(\theta)$, density family q_ψ, neural network $F(x,\phi)$, simulations per round N, number of rounds R.

$\tilde{p}_1(\theta) := p(\theta)$
for $r = 1$ **to** R **do**
  for $j = 1$ **to** N **do**
    Sample $\theta_{r,j} \sim \tilde{p}_r(\theta)$
    Simulate $x_{r,j} \sim p(x|\theta_{r,j})$
  end for
  $\phi \leftarrow \underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{r} \sum_{j=1}^{N} -\log \tilde{q}_{x_{i,j},\phi}(\theta_{i,j})$ using (2)
  $\tilde{p}_{r+1}(\theta) := q_{F(x_o,\phi)}(\theta)$
end for
return $q_{F(x_o,\phi)}(\theta)$

- sequentially update prior to form a proposal prior $\tilde{p}(\vartheta)$ (and a $\tilde{p}(\vartheta|\vec{x})$)
- loss function has to be adapted
- **cINNs can be used as conditional density estimator**
- Greenberg et al, 2019:
  "Learning with such 'atomic' proposals has an intuitive interpretation: we are training the network to solve multiple choice test problems, of the format "which of these ϑ's generated this x?"

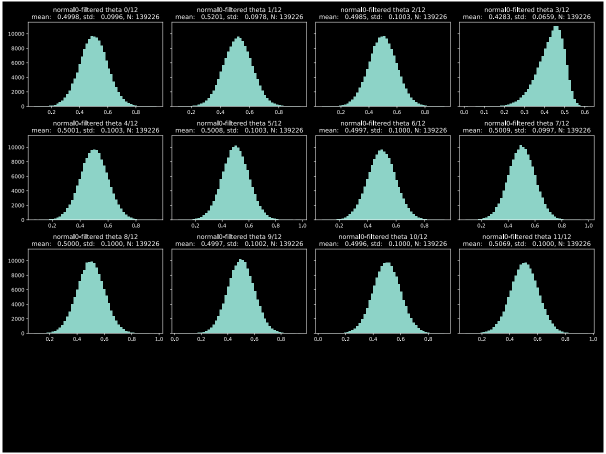
sbi MAPs on the validation set



- training:
  - >100 epochs only
  - fixed arch: 8 layers, neural spine flow Durkan et al, 2019
  - 256 units per layer
- inference: 256 draws per validation sample, MAP by mean
- good: posterior stays within prior support
- to improve: posterior misses out for two dimensions (expected)

www.mackelab.org/sbi

the real world



reproducibility, openness & team work = key!

(all results from above were from toy simulations)

Next Steps

- quality control of predictions:
  - sample based metrics (Naeem et al, 2020)
  - simulation based calibration (Talts et al, 2018)
- integrating cINNs from above into SNPE
- experiment with gradient based MAP estimation
- uncertainties of MAPs?

Summary

Take Aways

- **normalizing flows** have emerged as a learnable transformation between distributions

Take Aways

- **normalizing flows** have emerged as a learnable transformation between distributions
- they lend themselves as a central building block for **conditional density estimation**

Take Aways

- **normalizing flows** have emerged as a learnable transformation between distributions
- they lend themselves as a central building block for **conditional density estimation**
- a new field of statistics, **simulation-based inference**, for multi-variate n-dimensional parameter estimation is currently explored in many fields of science (astronomy, lattice QCD, life science, ...)

Take Aways

- **normalizing flows** have emerged as a learnable transformation between distributions
- they lend themselves as a central building block for **conditional density estimation**
- a new field of statistics, **simulation-based inference**, for multi-variate n-dimensional parameter estimation is currently explored in many fields of science (astronomy, lattice QCD, life science, ...)
- **openness, communication and interdisciplinary work** can drive all of the above

Take Aways

- **normalizing flows** have emerged as a learnable transformation between distributions
- they lend themselves as a central building block for **conditional density estimation**
- a new field of statistics, **simulation-based inference**, for multi-variate n-dimensional parameter estimation is currently explored in many fields of science (astronomy, lattice QCD, life science, ...)
- **openness, communication and interdisciplinary work** can drive all of the above

Take Aways

- **normalizing flows** have emerged as a learnable transformation between distributions
- they lend themselves as a central building block for **conditional density estimation**
- a new field of statistics, **simulation-based inference**, for multi-variate n-dimensional parameter estimation is currently explored in many fields of science (astronomy, lattice QCD, life science, ...)
- **openness, communication and interdisciplinary work** can drive all of the above

Questions? Feedback? Concerns?

Further Reading

- nice blog post with pytorch code samples
- “Normalizing Flows for Probabilistic Modeling and Inference,” G. Papamakarios et al, arXiv:1912.02762, 2019.
- “Normalizing Flows: An Introduction and Review of Current Methods”, I. Kobyzev et al, arXiv:1908.09257, 2019.
- “Glow: Generative Flow with Invertible 1x1 Convolutions”, Kingma et al, arXiv:1807.03039, 2018.

Backup

Converging to a loss function

$$p_X(x) = |det(\frac{df(x)}{dx})|p_Z(f(x))$$

Going from one f to multiple: $z_n = f_n \circ \cdots \circ f_1(x)$:

Converging to a loss function

$$p_X(x) = |det(\frac{df(x)}{dx})|p_Z(f(x))$$

Going from one f to multiple: $z_n = f_n \circ \cdots \circ f_1(x)$:

$$p_X(x) = \prod_{i=1}^{n} |det(\frac{dz_i}{dz_{i-1}})|p_Z(f(x)) \; , x = z_0$$

Converging to a loss function

$$p_X(x) = |det(\frac{df(x)}{dx})|p_Z(f(x))$$

Going from one f to multiple: $z_n = f_n \circ \dots \circ f_1(x)$:

$$p_X(x) = \prod_{i=1}^{n} |det(\frac{dz_i}{dz_{i-1}})|p_Z(f(x)) \; , x = z_0$$

$$\log(p_X(x)) = \sum_{i=1}^{n} \log|det(\frac{dz_i}{dz_{i-1}})|p_Z(f(x)) \; , x = z_0$$

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:
  - train h outside of INN (only feed $\tilde{c}_i = h(c_i)$ to INN)

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:
  - train h outside of INN (only feed $\tilde{c}_i = h(c_i)$ to INN)
  - train h jointly with INN

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:
  - train h outside of INN (only feed $\tilde{c}_i = h(c_i)$ to INN)
  - train h jointly with INN
- for image inputs:

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:
  - train h outside of INN (only feed $\tilde{c}_i = h(c_i)$ to INN)
  - train h jointly with INN
- for image inputs:
  - train classification network

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:
  - train h outside of INN (only feed $\tilde{c}_i = h(c_i)$ to INN)
  - train h jointly with INN
- for image inputs:
  - train classification network
  - remove last softmax layer

conditioning network details

- recall: outputs of conditioning network h concatenated to inputs of s_i and t_i
- feeds higher semantic features of data into INN
- 2 options:
  - train h outside of INN (only feed $\tilde{c}_i = h(c_i)$ to INN)
  - train h jointly with INN
- for image inputs:
  - train classification network
  - remove last softmax layer
  - use latent features as conditioning