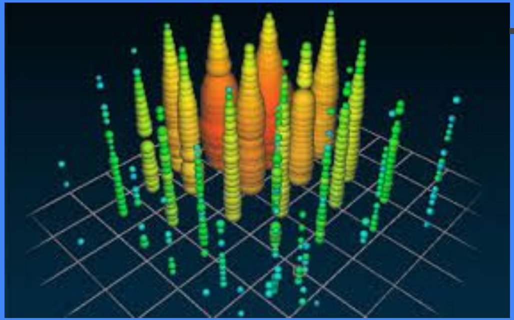
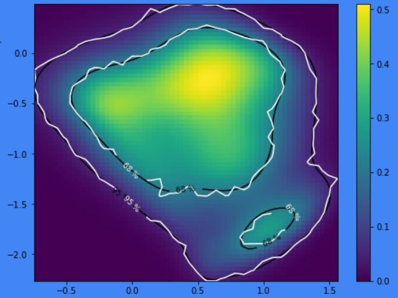
Normalizing flows

An overview and its revolutionary potential for high-energy physics when combined with deep learning



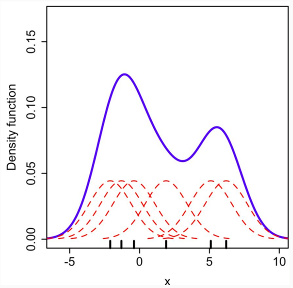


Friedrich-Alexander-Universität Erlangen-Nürnberg

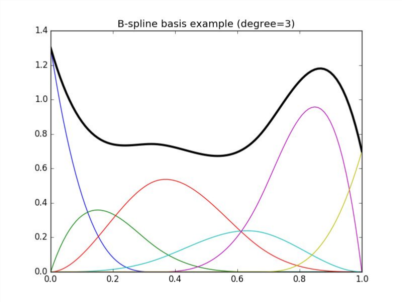
Thorsten Glüsenkamp, 22.4. 2022, IoP Machine Learning workshop

What are normalizing flows? … specific probability density functions

Some well-known techniques to construct PDFs:



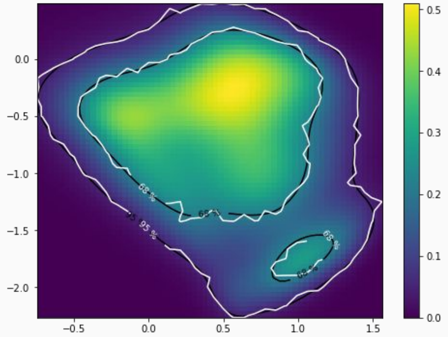
Mixture models / Kernel Density Functions



B-Spline representation of a PDF

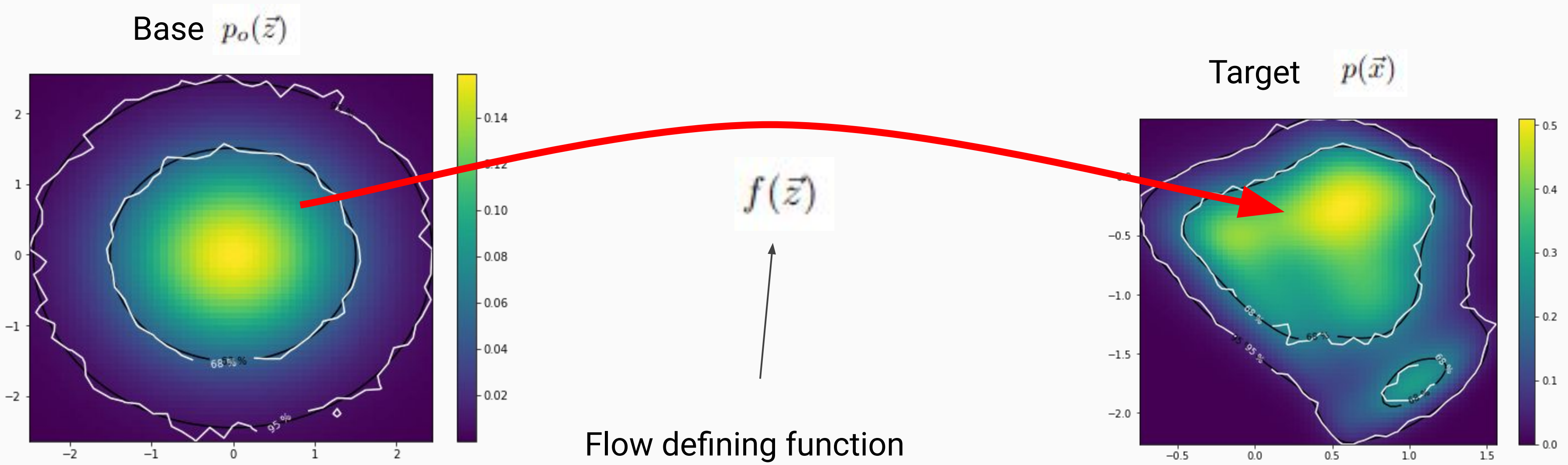
- (arbitrarily) Complex PDF shape
- Evaluate probability analytically
- Works in D > 1

Normalizing flows are also PDFs, but richer functionality:



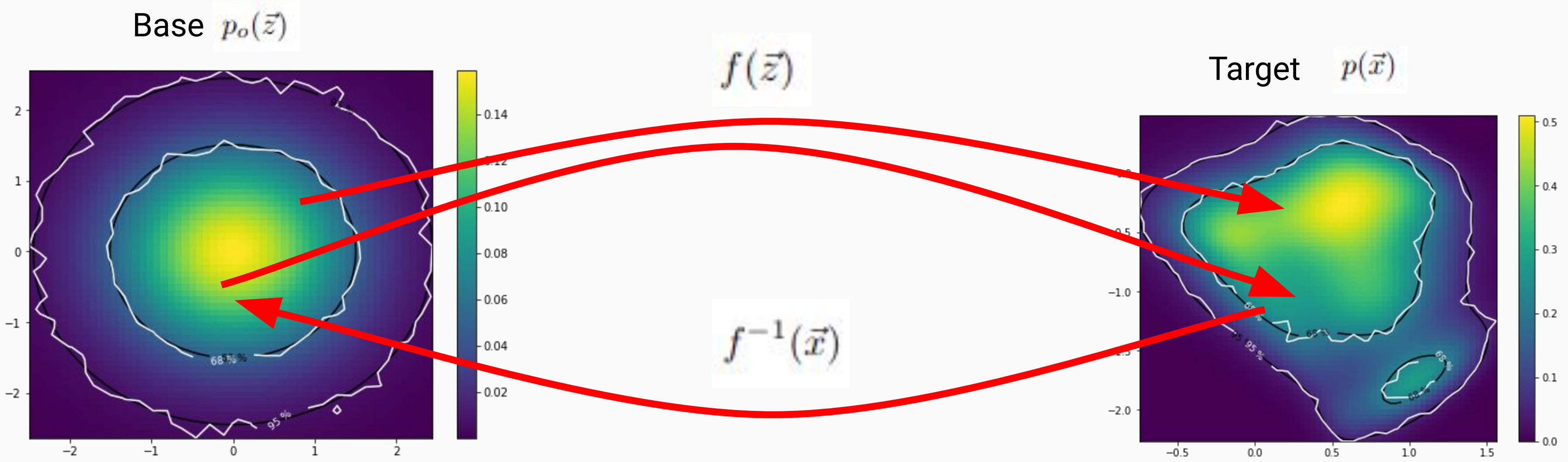
- (arbitrarily) Complex PDF shape
- Evaluate probability analytically
- Works in D > 1
- Generate differentiable samples (-> differentiable expectation values)
- Coverage of the PDF
- Can be interpreted as generalizations of the Gaussian distribution
- Works on manifolds
- …

Definition



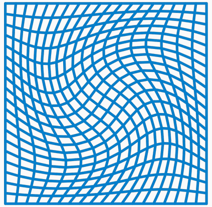
Definition

$$p(\vec{x}) = p_0(f^{-1}(\vec{x})) \cdot \left| \det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}} \right|$$



1) The function has to be **bijective** for the inverse!
2) The function has to be **differentiable!**

= Diffeomorphism



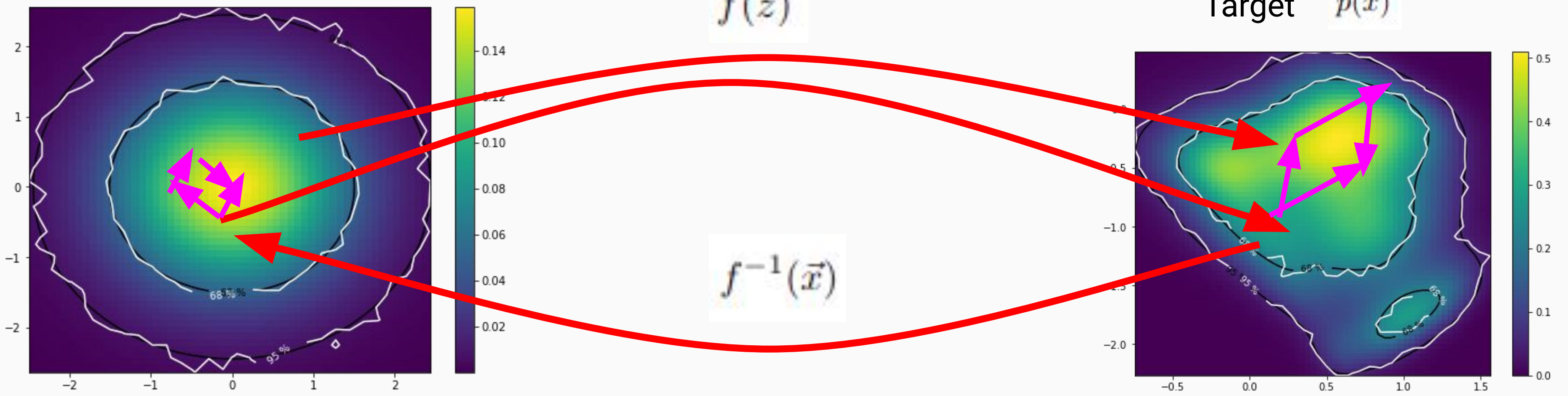
Definition

$$p(\vec{x}) = p_0(f^{-1}(\vec{x})) \cdot \left|\det J^{-1}(\vec{x})\right|$$

← Log probability includes
Local “stretching”/”squeezing”
Of volume element

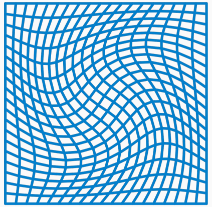
Base $p_o(\vec{z})$

Target $p(\vec{x})$



1) The function has to be **bijective** for the inverse!
2) The function has to be **differentiable!**

= Diffeomorphism

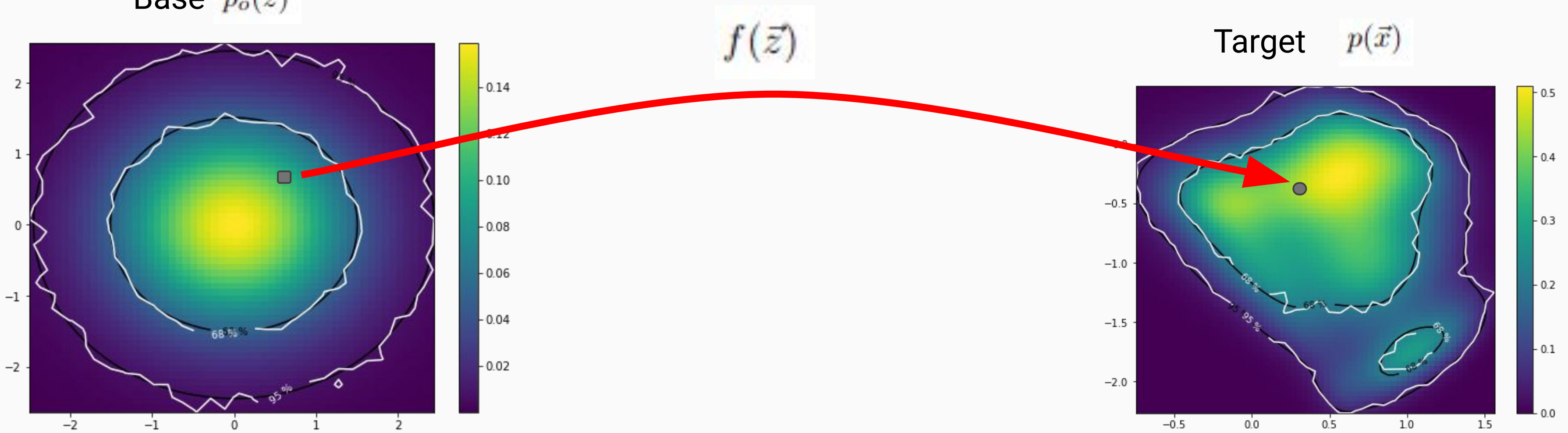


Definition

Sampling is also straight forward: $\vec{x} = f(\vec{z})$

Must be able to sample from base distribution!

Base $p_o(\vec{z})$



Definition

Sampling is also straight forward: $\vec{x}_\theta = f_\theta(\vec{z})$

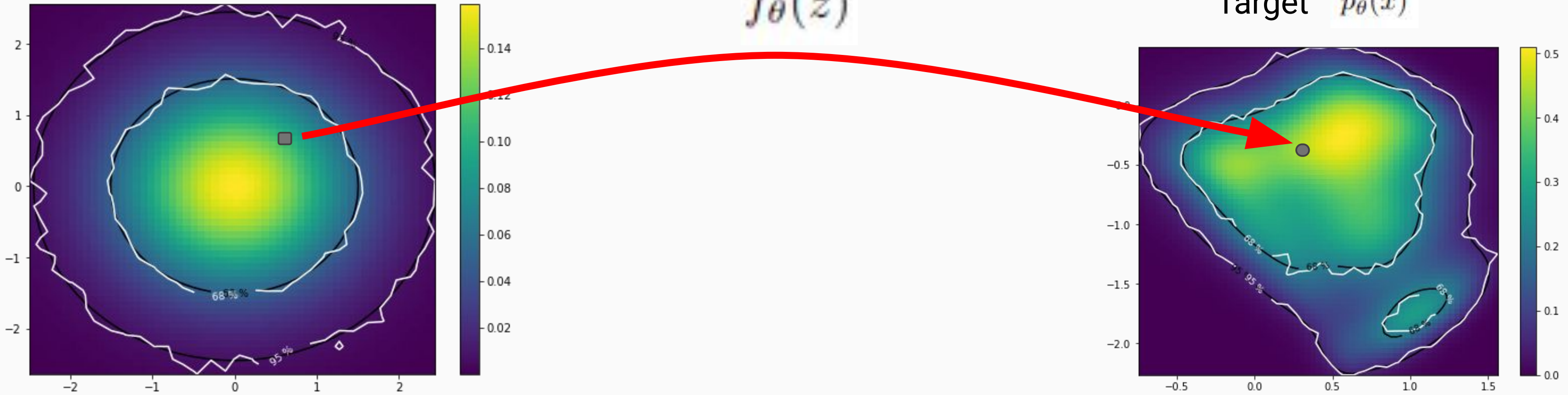
Must be able to sample from base distribution!

$$p_\theta(\vec{x}) = p_0(f_\theta^{-1}(\vec{x})) \cdot \left|\det \frac{\partial f_\theta^{-1}(\vec{x})}{\partial \vec{x}}\right|$$

Base $p_o(\vec{z})$

$f_\theta(\vec{z})$

Target $p_\theta(\vec{x})$



Definition

Sampling is also straight forward:

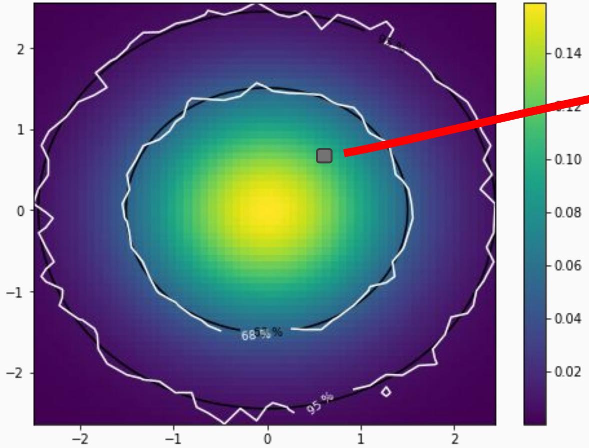
$$\vec{x}_\theta = f_\theta(\vec{z})$$

Must be able to sample from base distribution!

1) Differentiable prob.

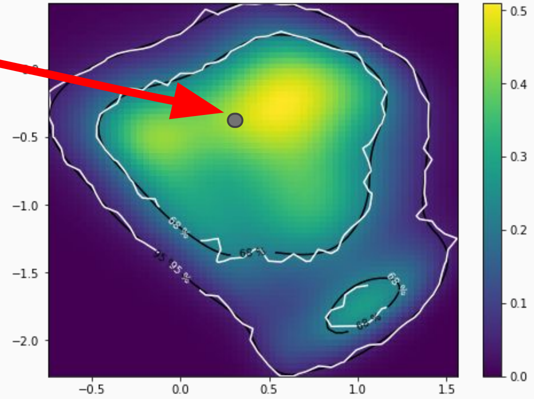
$$p_\theta(\vec{x}) = p_0(f_\theta^{-1}(\vec{x})) \cdot \left|\det \frac{\partial f_\theta^{-1}(\vec{x})}{\partial \vec{x}}\right|$$

Base $p_o(\vec{z})$



$f_\theta(\vec{z})$

Target $p_\theta(\vec{x})$



2) Differentiable samples!

Sampling is also straight forward:

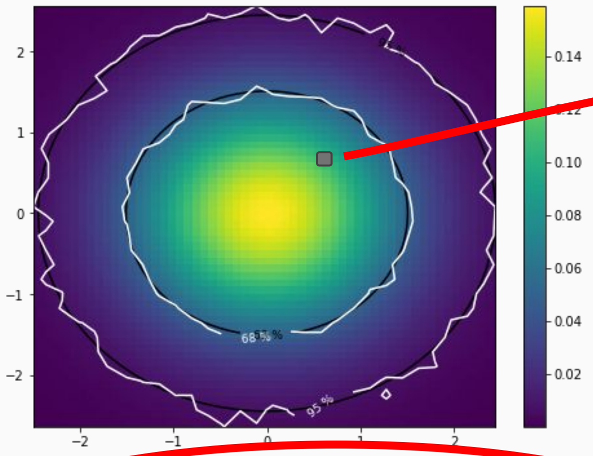
Must be able to sample from base distribution!

$$\vec{x}_\theta = f_\theta(\vec{z})$$

1) Differentiable prob.

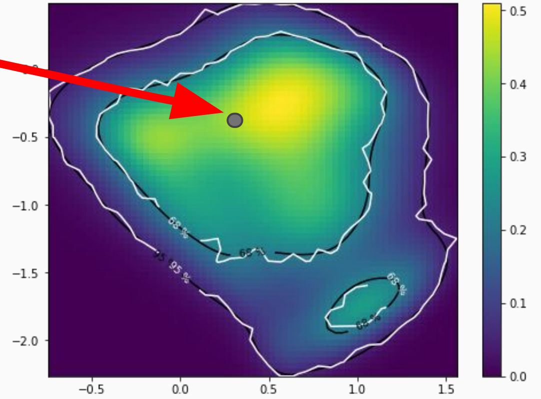
$$p_\theta(\vec{x}) = p_0(f_\theta^{-1}(\vec{x})) \cdot \left|\det \frac{\partial f_\theta^{-1}(\vec{x})}{\partial \vec{x}}\right|$$

Base $p_o(\vec{z})$



$f_\theta(\vec{z})$

Target $p_\theta(\vec{x})$



2) Differentiable samples!

Unique and important!

Differentiable Samples -> Differentiable expectation values

$$I_\theta = \int p_\theta(x) F_\theta(x) dx \approx \frac{1}{N} \cdot \sum_i^N F_\theta(x_\theta)$$

Differentiable Samples -> Differentiable expectation values

$$I_\theta = \int p_\theta(x) F_\theta(x) dx \approx \frac{1}{N} \cdot \sum_i^N F_\theta(x_\theta)$$

Examples:

n-th moment of p

$$\int p_\theta(x) x^n dx \approx \frac{1}{N} \cdot \sum_i^N x_\theta^n$$

entropy

$$-\int p_\theta(x) \log\left(p_\theta(x)\right) dx \approx \frac{1}{N} \cdot \sum_i^N -\log p_\theta(x_\theta)$$

…. Many other integrals in information theory

$$I_\theta = \int p_\theta(x) F_\theta(x) dx \approx \frac{1}{N} \cdot \sum_i^N F_\theta(x_\theta)$$

Examples:

n-th moment of p

$$\int p_\theta(x) x^n dx \approx \frac{1}{N} \cdot \sum_i^N x_\theta^n$$

entropy

$$-\int p_\theta(x) \log\left(p_\theta(x)\right) dx \approx \frac{1}{N} \cdot \sum_i^N -\log p_\theta(x_\theta)$$

Wrong gradient!

$$\frac{1}{N} \cdot \sum_i^N -\log p_\theta(x)$$

!

…. Many other integrals in information theory

The simplest possible normalizing flow (1-d)

Let us try to see the distribution of the simplest possible normalizing flow in 1 dimension!

$$x = f(\vec{z}) =$$

The simplest possible normalizing flow (1-d)

Let us try to see the distribution of the simplest possible normalizing flow in 1 dimension!

$$x = f(\vec{z}) = a \cdot z + b \qquad \theta = \{a, b\}$$

$$p_\theta(\vec{x}) = p_0(f_\theta^{-1}(\vec{x})) \cdot \left| \det \frac{\partial f_\theta^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

The simplest possible normalizing flow (1-d)

Let us try to see the distribution of the simplest possible normalizing flow in 1 dimension!

$$x = f(\vec{z}) = a \cdot z + b$$

Use Standard normal base distribution (we will always use the standard normal for convenience)

Base distribution

$$p_0(z) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot \exp(-0.5 \cdot z^2)$$

Change of variables formula

$$p_\theta(\vec{x}) = p_0(f_\theta^{-1}(\vec{x})) \cdot \left| \det \frac{\partial f_\theta^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

Inv. Flow function

$$z = f^{-1}(\vec{x}) = \frac{x - b}{a}$$

$$p(x) = \frac{1}{\sqrt{2 \cdot \pi \cdot a^2}} \cdot \exp\left(-0.5 \cdot \left(\frac{x - b}{a}\right)^2\right)$$

The simplest possible normalizing flow (1-d)

Let us try to see the distribution of the simplest possible normalizing flow in 1 dimension!

$$x = f(\vec{z}) = a \cdot z + b$$

Use Standard normal base distribution (we will always use the standard normal for convenience)

"Linear flow" = "general Gaussian distribution" in 1d

Base distribution

$$p_0(z) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot \exp(-0.5 \cdot z^2)$$

Change of variables formula

$$p_\theta(\vec{x}) = p_0(f_\theta^{-1}(\vec{x})) \cdot \left| \det \frac{\partial f_\theta^{-1}(\vec{x})}{\partial \vec{x}} \right|$$

Inv. Flow function

$$z = f^{-1}(\vec{x}) = \frac{x - b}{a}$$

$$p(x) = \frac{1}{\sqrt{2 \cdot \pi \cdot a^2}} \cdot \exp\left(-0.5 \cdot \left(\frac{x - b}{a}\right)^2\right)$$

The simplest possible normalizing flow (1-d)

Let us try to see the distribution of the simplest possible normalizing flow in 1 dimension!

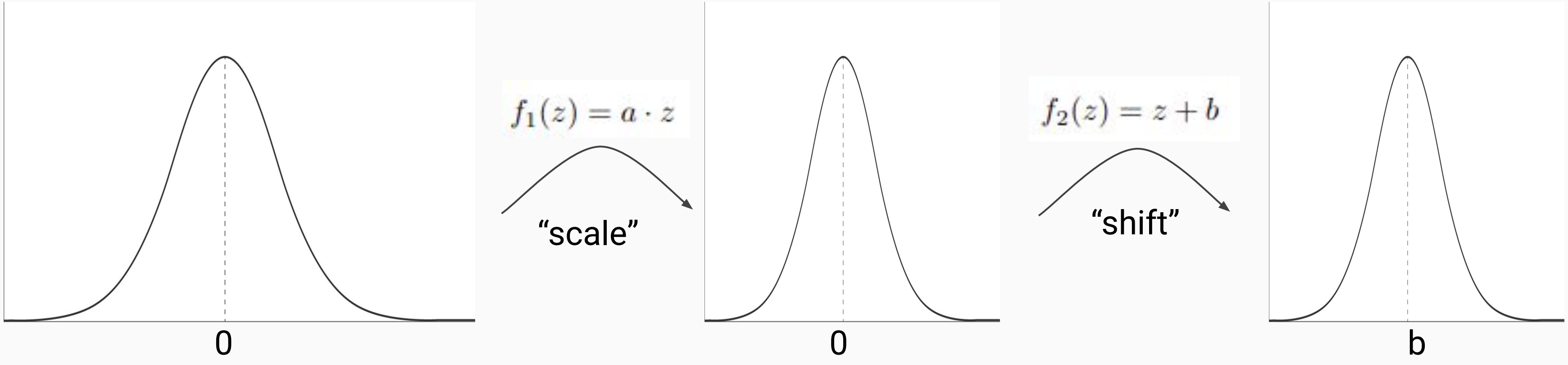
$$x = f(\vec{z}) = a \cdot z + b$$

Use Standard normal base distribution (we will always use the standard normal for convenience)

> “Linear flow” = “general Gaussian distribution” in 1d

Technically, this is 2-step flow:

$$f(z) = f_2(f_1(z))$$



The simplest possible normalizing flow (n-d)

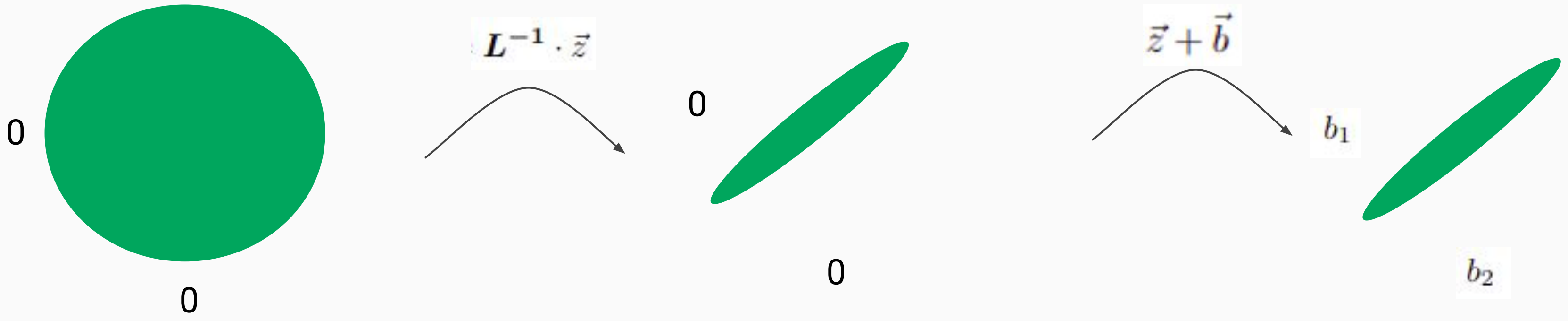
Let us try to see the distribution of the simplest possible normalizing flow in n dimensions!

$$\vec{x} = f(\vec{z}) = \boldsymbol{L}^{-1} \cdot \vec{z} + \vec{b}$$

Use Standard normal base distribution (we will always use the standard normal for convenience)

$$\boldsymbol{C}^{-1} = \boldsymbol{L}^T \cdot \boldsymbol{L}$$

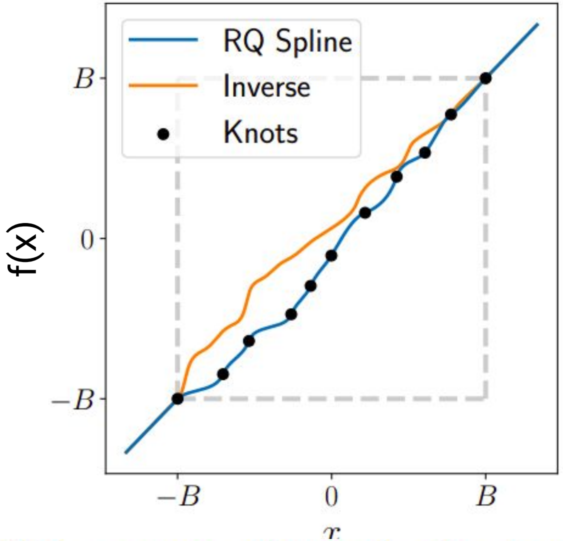
“Affine flow” = “general Gaussian distribution” in n-d



How can we ensure bijectivity in 1-d for nonlinear functions?

One possibility:
Neural Spline Flows (Durkan et al. 2019)

$$p(\vec{x}) = p_0(f^{-1}(\vec{x})) \cdot \left|\det \frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}}\right|$$



Flow function: $f^{-1}(\vec{x})$

$$\frac{\alpha^{(k)}(\xi)}{\beta^{(k)}(\xi)} = y^{(k)} + \frac{(y^{(k+1)} - y^{(k)})\left[s^{(k)}\xi^2 + \delta^{(k)}\xi(1-\xi)\right]}{s^{(k)} + \left[\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}\right]\xi(1-\xi)}$$

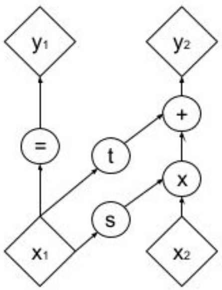
Derivative: $\frac{\partial f^{-1}(\vec{x})}{\partial \vec{x}}$

$$\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{\alpha^{(k)}(\xi)}{\beta^{(k)}(\xi)}\right] = \frac{\left(s^{(k)}\right)^2\left[\delta^{(k+1)}\xi^2 + 2s^{(k)}\xi(1-\xi) + \delta^{(k)}(1-\xi)^2\right]}{\left[s^{(k)} + \left[\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}\right]\xi(1-\xi)\right]^2}$$

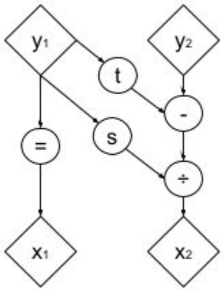
which passes through the knots, with the given derivatives at the knots. Defining $s_k = (y^{k+1} - y^k)/(x^{k+1} - x^k)$ and $\xi(x) = (x - x^k)/(x^{k+1} - x^k)$, the expression for the rational-

Certain N-d non-linear normalizing-flow functions

1) "Affine Coupling layer" (Dinh et al. 2014 (NICE) / Dinh et al. 2017 (real-NVP))



(a) Forward propagation



(b) Inverse propagation

$$
\begin{aligned}
y_{1:d} &= x_{1:d} \\
y_{d+1:D} &= x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}),
\end{aligned}
$$

$$
\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x^T_{1:d}} & \operatorname{diag}\big(\exp\left[s\left(x_{1:d}\right)\right]\big) \end{bmatrix},
$$

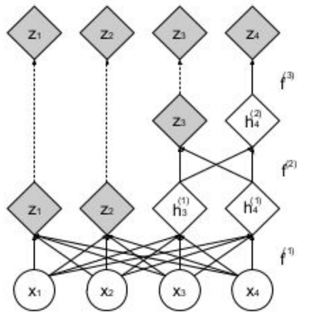
Induces coupling between different layers

Functions ***s*** and ***t*** can be arbitrarily complex (neural networks)

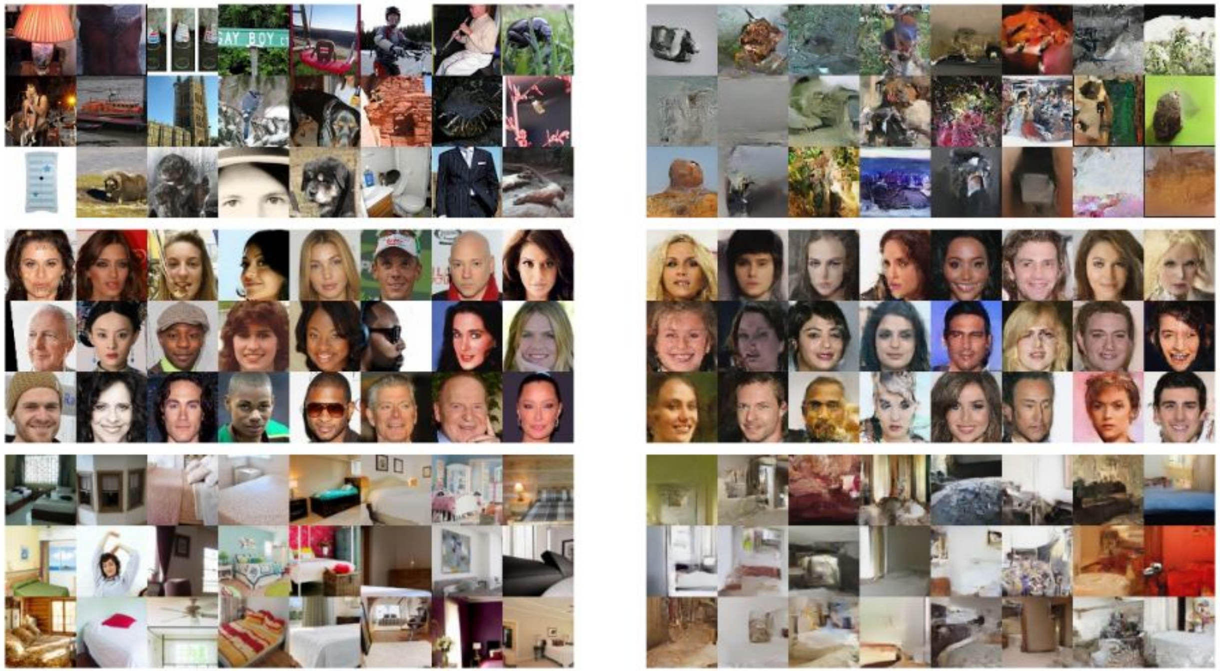
These functions are also called "conditioners"

Certain N-d non-linear normalizing-flow functions

1) "Affine Coupling layer" (Dinh et al. 2014 (NICE) / Dinh et al. 2017 (real-NVP))



(b) Factoring out variables. At each step, half the variables are directly modeled as Gaussians, while the other half undergo further transformation.

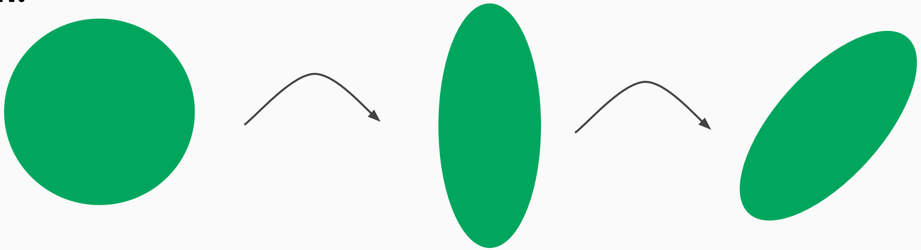


Can be scaled to very high dimensions (here images)

Certain N-d non-linear normalizing-flow functions

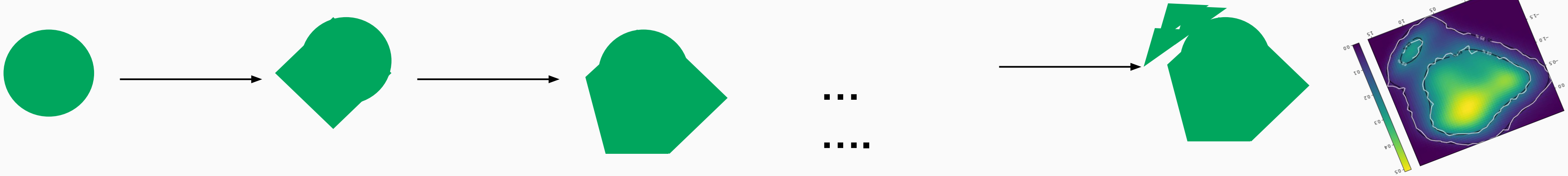
4) Gaussianization Flow(arXiv:2003.01941)

Gaussian:



Gaussianization flow generalized by non-linear scalings instead of “linear scalings”

$$T_{\boldsymbol{\theta}}(\mathbf{x}) = \Psi_{\boldsymbol{\theta}_L} \circ R_L \circ \Psi_{\boldsymbol{\theta}_{L-1}} \circ \cdots \circ \Psi_{\boldsymbol{\theta}_1} \circ R_1 \mathbf{x}$$

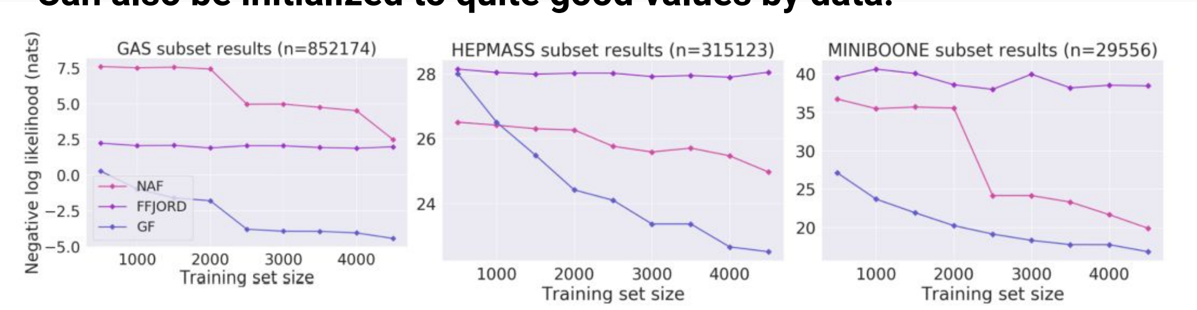


Step by step refinement of the PDF - provably approximates any distribution

Certain N-d non-linear normalizing-flow functions

4) Gaussianization Flow(arXiv:2003.01941)

Can also be initialized to quite good values by data!



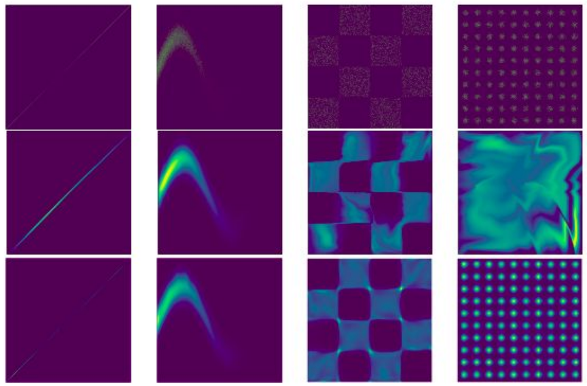


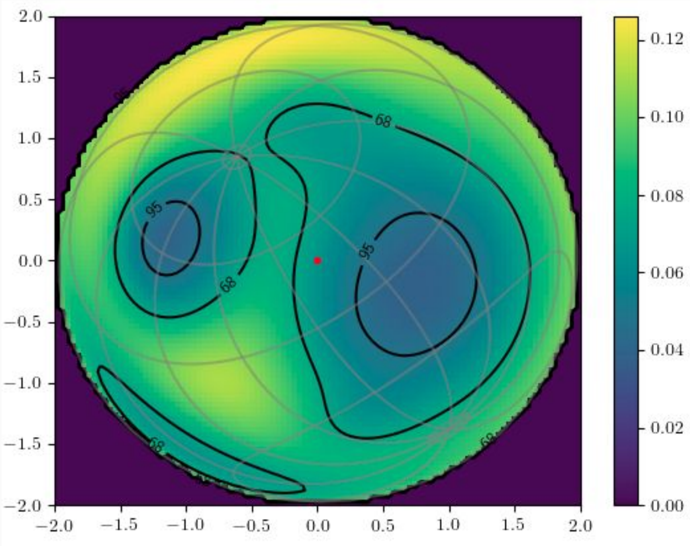
Figure 2: 2D density estimation results. **Top:** Ground truth samples. **Middle:** Glow. **Bottom:** GF.

Method	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	FMNIST
Real NVP	-0.17	-8.33	18.71	13.55	-153.28	1.06	**2.85**
Glow	-0.17	-8.15	18.92	11.35	-155.07	1.05	2.95
FFJORD	-0.46	-8.59	**14.92**	10.43	**-157.40**	**0.99**	-
RBIG	1.02	0.05	24.59	25.41	-115.96	1.71	4.46
GF(ours)	**-0.57**	**-10.13**	17.59	**10.32**	-152.82	1.29	3.35
MADE	3.08	-3.56	20.98	15.59	-148.85	2.04	4.18
MAF	-0.24	-10.08	17.70	11.75	-155.69	1.89	-
TAN	-0.48	-11.19	15.12	11.01	-157.03	-	-
MAF-DDSF	-0.62	-11.96	15.09	8.86	-157.73	-	-

Other interesting current research

"Manifold" normalizing flows

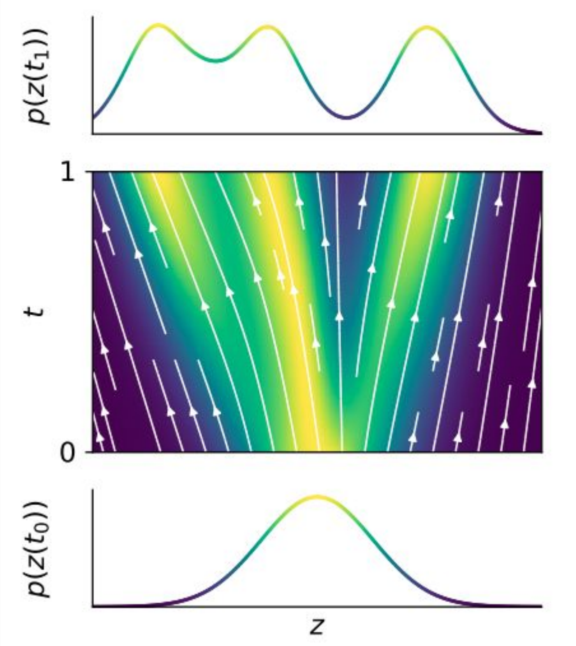
$$p(T(x)) = \frac{\pi(x)}{\sqrt{\det(E^\top J^\top J E)}}$$



(2002.02428)

"Continuous" normalizing flows

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt.$$



(FFJORD, 1810.01367)

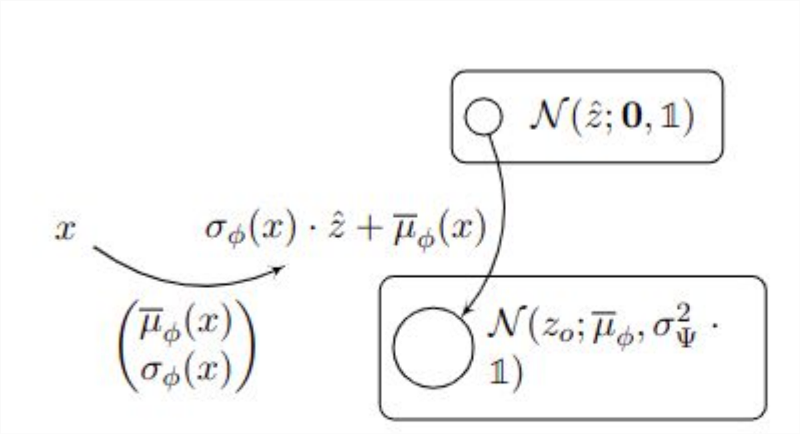
What about conditional PDFs?

p(x) -> p(x;y) ?

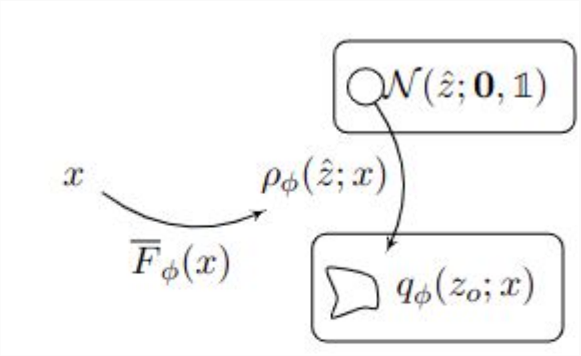
Connection to neural networks…

Conditional PDFs .. parameters of flow are output of a neural network

Conditional affine flow



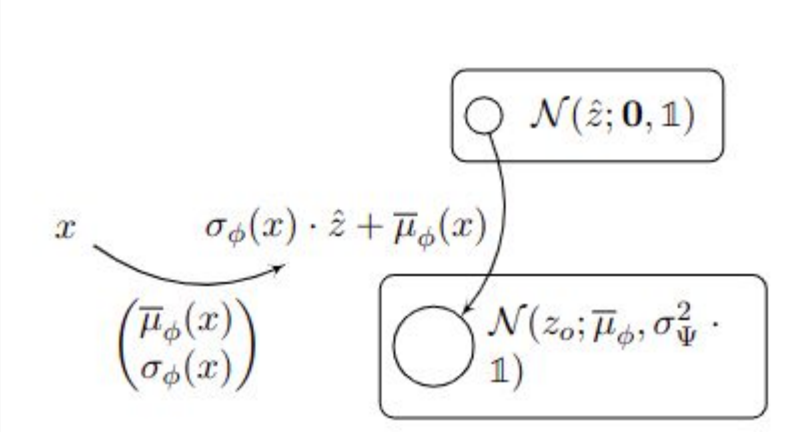
Conditional general flow



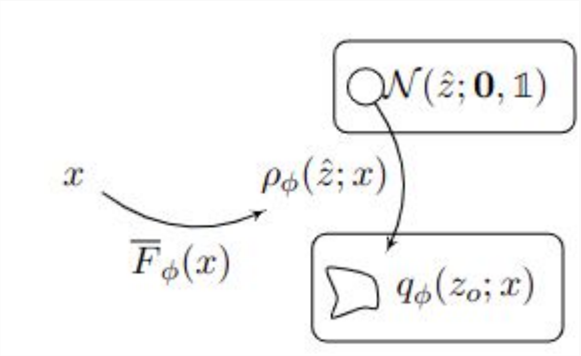
→ Instead of flow parameters, one optimizes NN parameters

Conditional PDFs .. parameters of flow are output of a neural network

Conditional affine flow



Conditional general flow

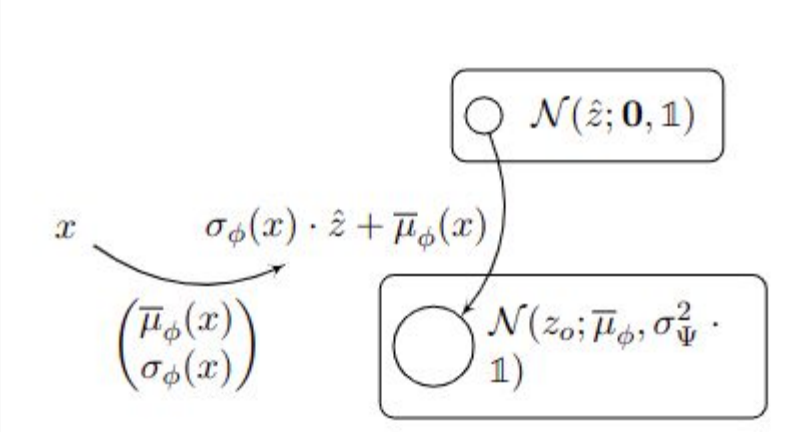


→ Instead of flow parameters, one optimizes NN parameters

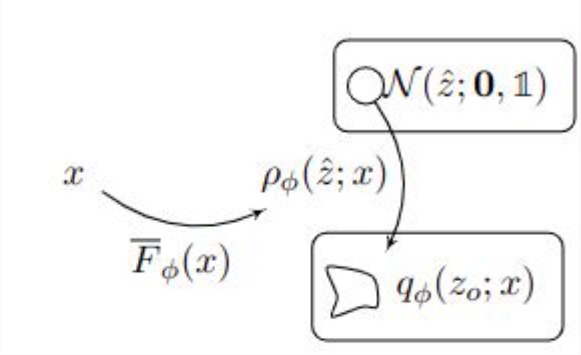
→ Conditional normalizing flow shows that **MSE loss comes from conditional Flow that only consists of a shift** (and unit scaling) $0.5 \cdot (x-\mu)^2 = \ln(p(x))$

Conditional PDFs .. parameters of flow are output of a neural network

Conditional affine flow



Conditional general flow



- Instead of flow parameters, one optimizes NN parameters
- Conditional normalizing flow shows that **MSE loss comes from conditional Flow that only consists of a shift** (and unit scaling) $0.5 \cdot (x-\mu)^2 = \ln(p(x))$
- The process of **predicting parameters by a neural network** is also called "**amortization**"

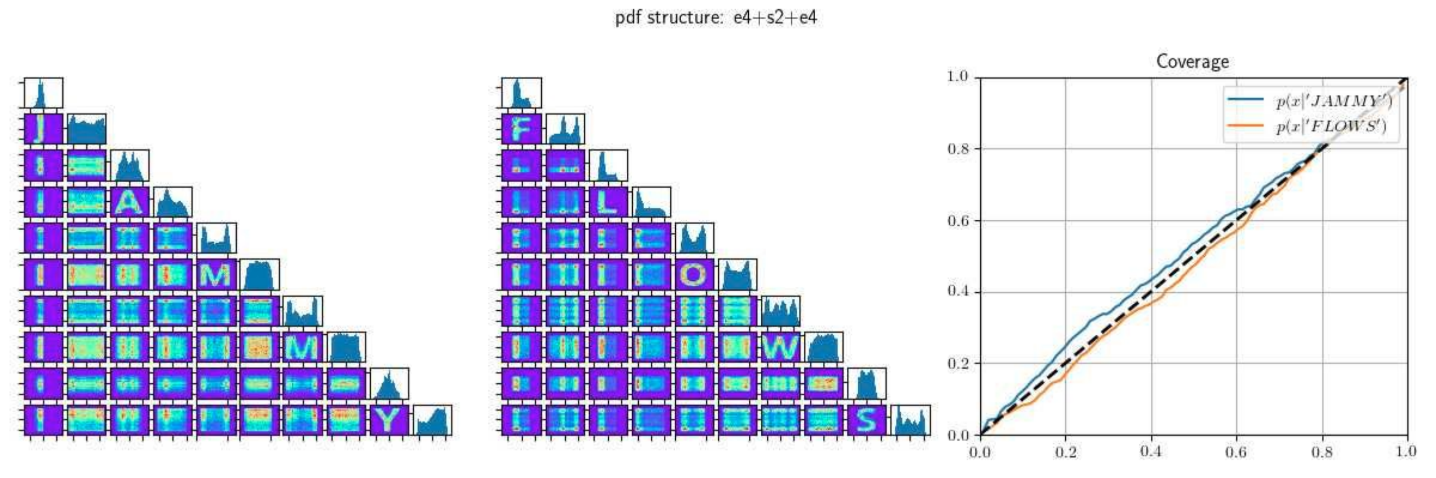
A package for flows on manifolds: jammy_flows (https://github.com/thoglu/jammy_flows)

import jammy_flows

pdf=jammy_flows.pdf("e4+s2+e4", "gggg+n+gggg")

pdf.sample(nsamples=1000)

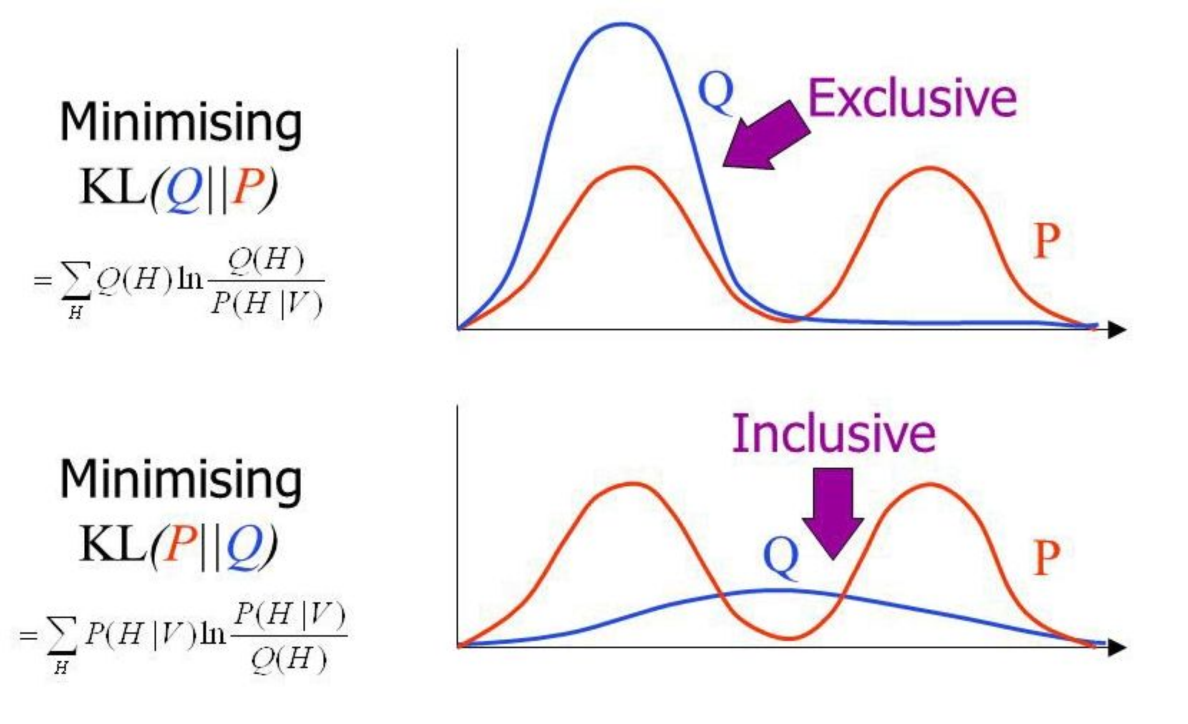
A package to describe amortized (conditional) normalizing-flow PDFs defined jointly on tensor products of manifolds with coverage control. The connection between different manifolds is fixed via an autoregressive structure.



Why are normalizing flows interesting for physics?

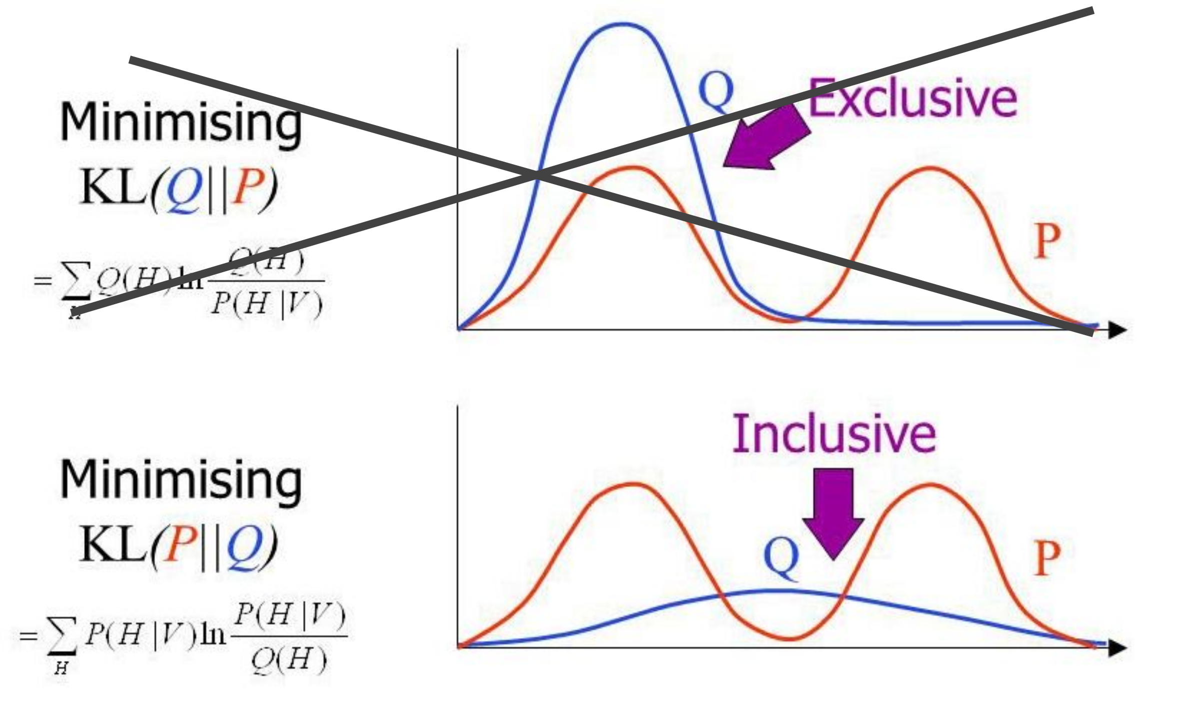
They allow to do statistical analysis with probabilistic machine learning…

"Inclusive" KL-diverengce vs "exclusive" KL-divergence



KL-divergence introduction

"Inclusive" KL-diverengce vs "exclusive" KL-divergence



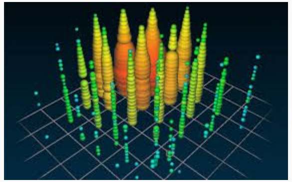
In most settings:

P = "True" PDF
(not accessible,
"Nature",
Samples from MC simulation
Draw from P)

Q = "Approximating PDF",
Parametrized by us,
"Surrogate model"

Supervised learning as (inclusive) KL-divergence minimization

What is a Monte Carlo simulation?
Samples from some "true" distribution



$$\arg\min_{\phi} \hat{D}_{\mathrm{KL,joint(x,z_o)}}(\mathcal{P}_t; q_\phi) = \arg\min_{\phi} \frac{1}{N} \sum_{S \in x_i, z_{o,i}} \ln\left(\frac{\mathcal{P}_t(z_{o,i}; x_i)}{q_\phi(z_{o,i}; x_i)}\right) + \ln\left(\frac{\mathcal{P}_t(x_i)}{q(x_i)}\right)$$

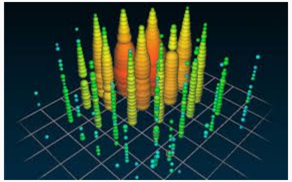
$$= \arg\min_{\phi} \frac{1}{N} \sum_{S \in x_i, z_{o,i}} -\ln\left(q_\phi(z_{o,i}; x_i)\right)$$

Computer scientists call this
"**conditional ML objective**"
Of supervised learning

Physicists should call it
"**variational inference objective**"
For the **variational Posterior**

Supervised learning as (inclusive) KL-divergence minimization

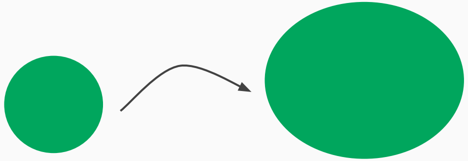
What is a Monte Carlo simulation?
Samples from some “true” distribution



$$\arg\min_{\phi} \hat{D}_{\mathrm{KL,joint(x,z_o)}}(\mathcal{P}_t; q_\phi) = \arg\min_{\phi} \frac{1}{N} \sum_{S \in x_i, z_{o,i}} \ln\left(\frac{\mathcal{P}_t(z_{o,i}; x_i)}{q_\phi(z_{o,i}; x_i)}\right) + \ln\left(\frac{\mathcal{P}_t(x_i)}{q(x_i)}\right)$$

$$= \arg\min_{\phi} \frac{1}{N} \sum_{S \in x_i, z_{o,i}} -\ln\left(q_\phi(z_{o,i}; x_i)\right)$$

Sample from “systematics distribution” during MC generation



Computer scientists call this
“**conditional ML objective**”
Of supervised learning

Including systematics is trivial!

Physicists should call it
“**variational inference objective**”
For the **variational Posterior**

Deep learning generalizes classical statistical approaches

"Bayesian"

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

MCMC

"Frequentist"

Maximum likelihood principle

Deep learning generalizes classical statistical approaches

KL-Div posterior

KL-Div likelihood

"Bayesian"

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

(90's)

"Frequentist"

(Aikake 70's)

MCMC

Variational inference

Maximum likelihood principle

Deep learning generalizes classical statistical approaches

KL-Div posterior

Joint KL
(deep learning)

KL-Div likelihood

"Bayesian"

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

(90's)

"Frequentist"

(Aikake 70's)

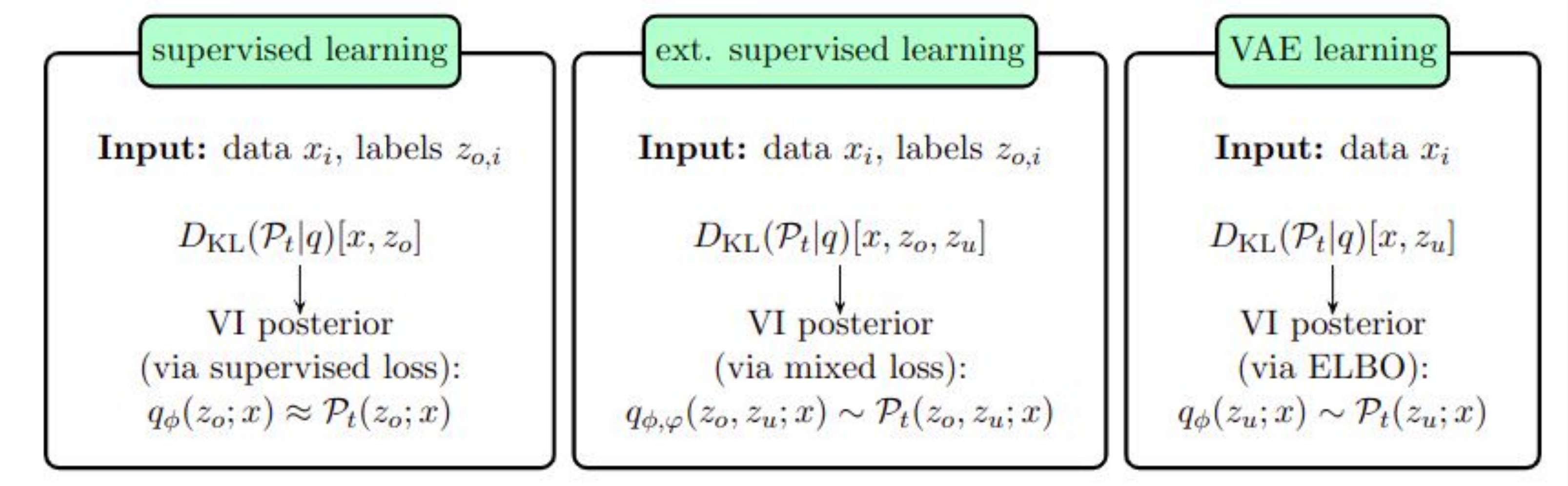
MCMC

Variational inference

Maximum likelihood principle

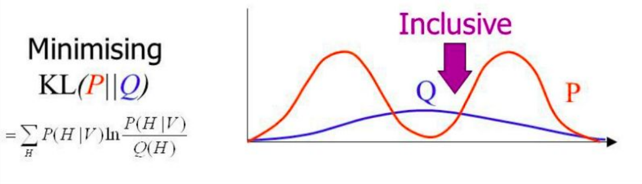
Probabilistic deep learning

"All of deep learning is probability distribution matching"



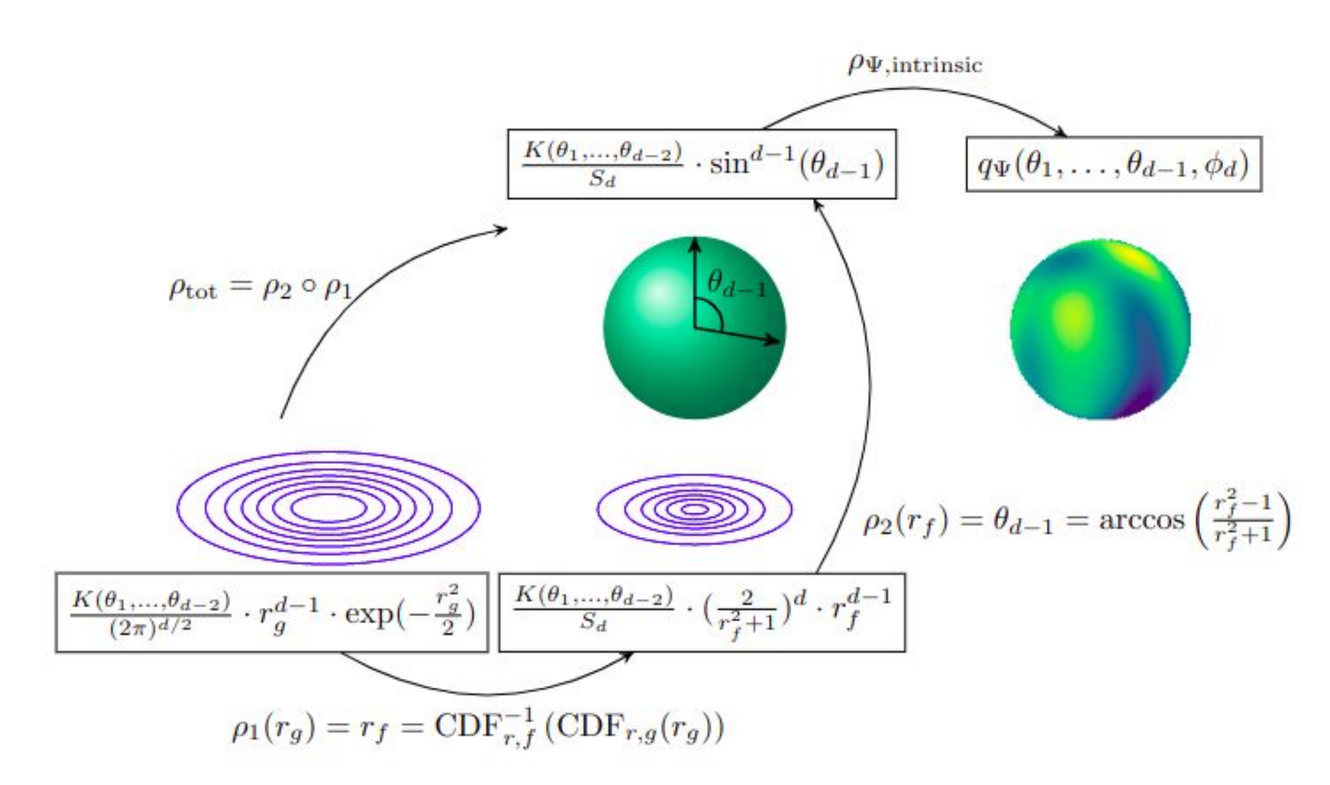
"Inklusive KL divergence"

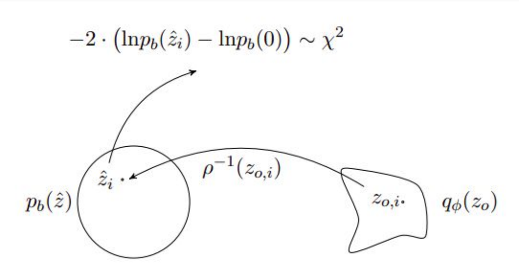
"exclusive KL divergence"

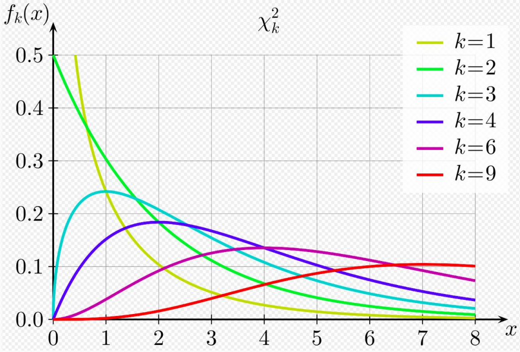


(2008.05825)

Coverage for NFs, including NFs on manifolds

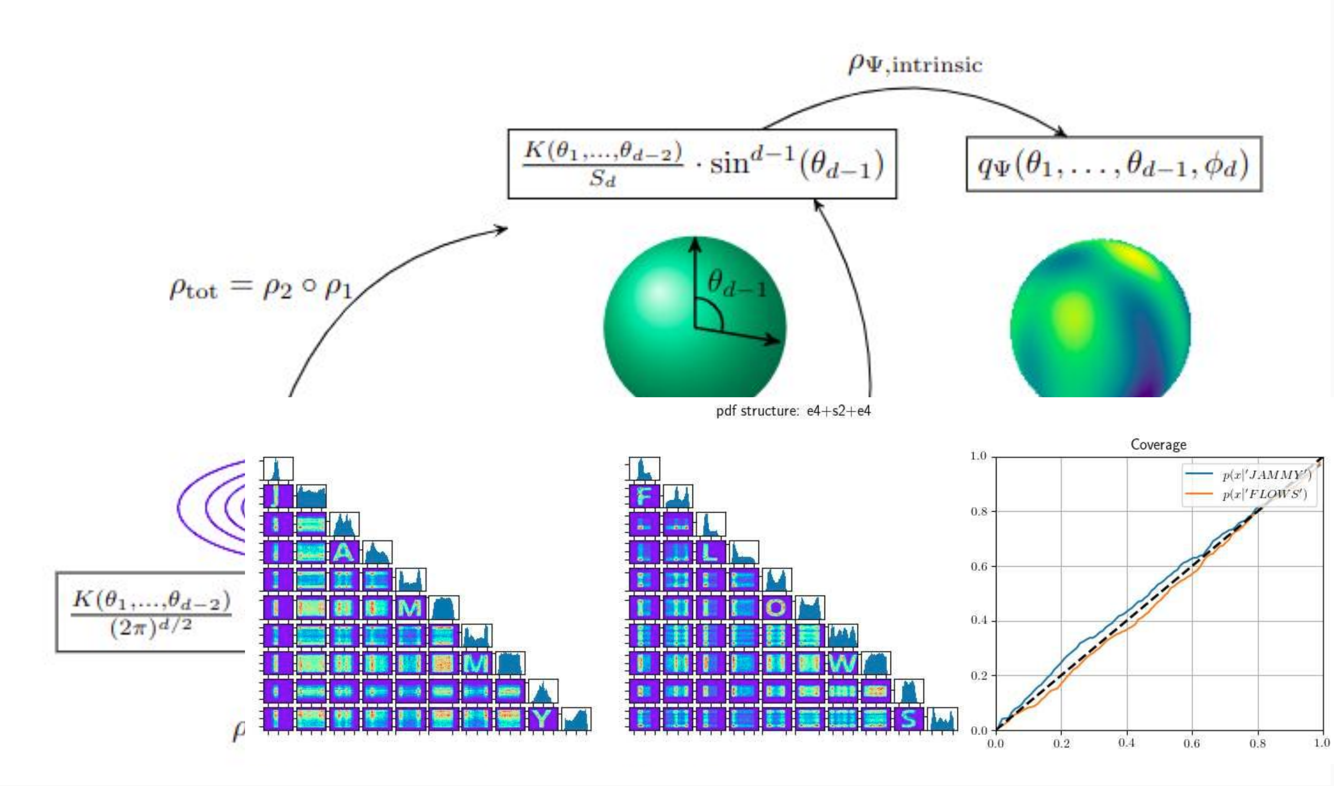


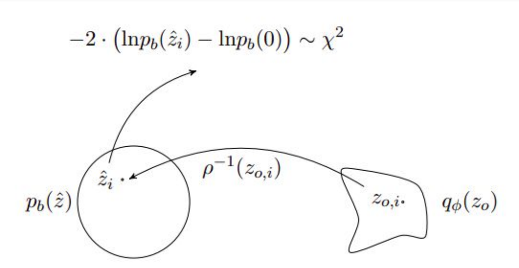


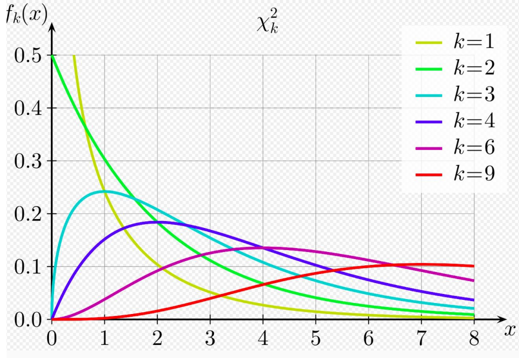


(2008.05825)

Coverage for NFs, including NFs on manifolds

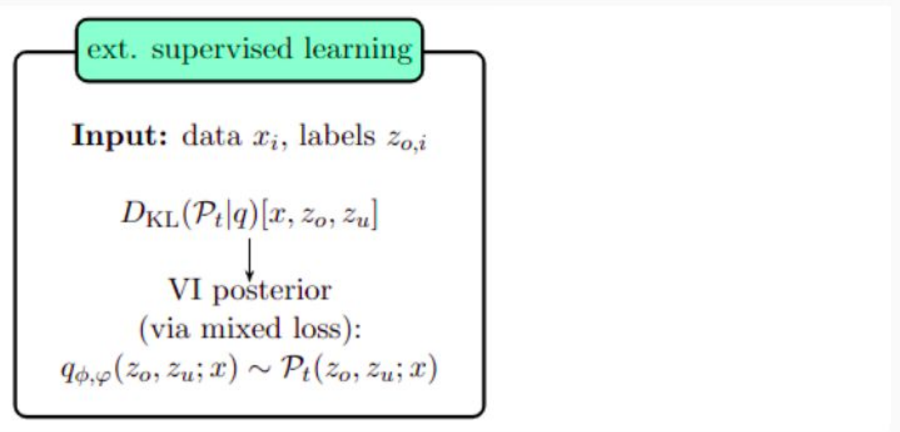


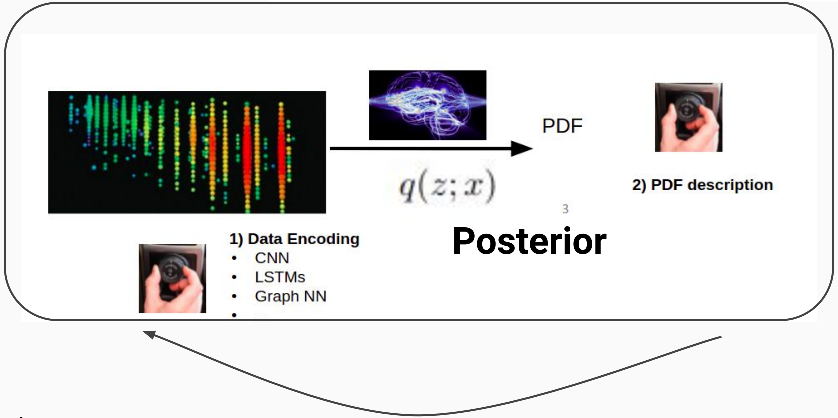




(2008.05825)

Goodness of Fit





(2008.05825)

Likelihood $p(x;z)$

Goodness of Fit

ext. supervised learning

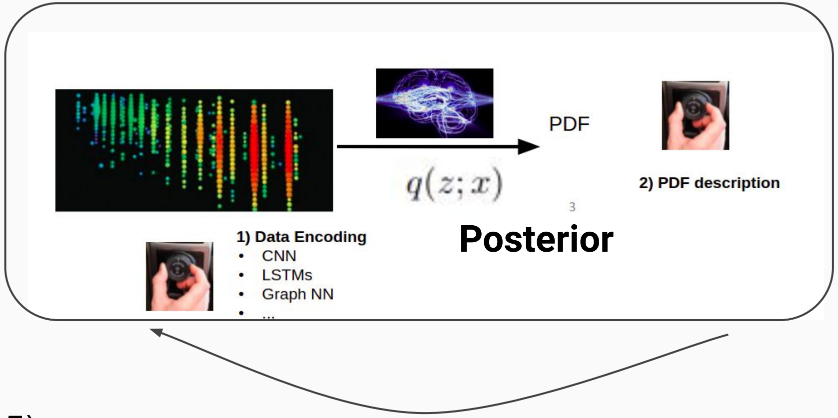
Input: data x_i, labels $z_{o,i}$

$$D_{\mathrm{KL}}(\mathcal{P}_t|q)[x, z_o, z_u]$$

↓

VI posterior
(via mixed loss):

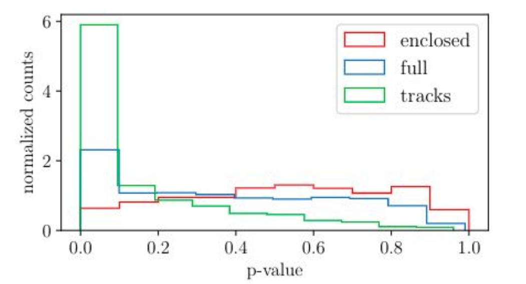
$$q_{\phi,\varphi}(z_o, z_u; x) \sim \mathcal{P}_t(z_o, z_u; x)$$



Likelihood $p(x;z)$

$$p_{\mathrm{val}} = \int_{x,z} I_{T(x,z)>T(x_{\mathrm{obs}},z)} p_\theta(x;z) q_\phi(z;x_{\mathrm{obs}}) dx dz$$

$$T(x,z) = \ln p_\theta(x;z)/N_d$$



(2008.05825)

Summary

- Normalizing flows + neural networks allow to model **complex (conditional) PDFs**

- Probabilistic interpretation of machine possible with normalizing flows (supervised / unsupervised learning etc. are just **PDF matching of Posterior/likelihood**)

- **Systematics** trivial (include in training)
- **Coverage** for arbitrary shaped distributions
- **Goodness-of-Fit** (potentially “single-cut” final level event selections that are “optimal”)

(2008.05825 for more info)

https://github.com/thoglu/jammy_flows
(first release in ~ few weeks)