**Notizen Einarbeitung erste Phase**

**Analyzing Inverse Problems w/ INNs (Ardizzone et al 2019)**

* Forward process: hidden variables **x** -> observables **y**, often well-defined
* Backward process: posterior of hidden variables p(x|y)
* INNs learn forward process and implicitly learn backward too, lost info encoded in additional output latent variables
* INNs discover multi-modalities, parameter correlations and identify unrecoverable hidden parameters

### 1. Intro

* model should learn full posterior of hidden params to estimate diversity of solutions
* standard y -> x approach needs supervised loss, which causes problems if x is ambiguous for y
* info about x which is not contained in y is captured in latent z, forward training optimizes f(x) = [z,y] with g as f´s inverse
* p(z) made to be gaussian, therefore INN represents a push of p(z) to x-space, conditioned on y
* direct posterior training requires a loss that respects the shape of the posterior, vicious circle
* learning easier forward process circumvents this Findings:full posterior can be represented with INNs with theoretical zero asymptotical loss
* invertibility restriction does not detrimentally limit INNs representative power
* forward training is asymptotically sufficient, but combining with unsupervised backward learning makes necessary training set size finite
* approach better than approximative Bayesian computation (ABC) or VAE

### 2. Related Work

* exact bayesian treatment of real-world problems (to learn posterior distr.) is usually intractable, usually sampling (monte carlo Markov chain) or, if model for forward process is available, ABC (rejection sampling scheme)
* normalizing flows: learning non-linear trafo between simple prior and true data distribution
* normalizing flows are not exactly INNs, usually inversion of probability is too costly

### 3. Methods

* assume a scientific model/formula y = s(x) w/ information loss, and want p(x|y) approximated by q(x|y). can sample infinitely with prior p(x) and forward model s(x) -> would allow for regression, but we want full posterior distr. Therefore introduce latent variable z from normal distr. And reparametrize q(x|y) by deterministic function of [y,z] represented by INN: x=g(y,z,\theta), z~p(z)
* learn g jointly with f, the forward process for y: [y,z] = f(x,theta) = [f\_y(x,theta);f\_z(x,theta)] with f\_y(x;theta) approximately s(x)
* f and g share parameters, joint bi-directional training avoids problems arising with cVAEs and Bayesian NNs
* f=g^-1 enforced by architecture, intrinsic and nominal dimensions must match (m<=M is intrinsic dimension of y: R^M), x:R^D -> z must be from R^K, K=D-m, assuming d=D.
* if nominal output dimension M+K exceeds D, add zeros to x
* > q(x=g(z,y,theta) | y) = p(z)\*|J\_x|^-1, J is jacobian of g w.r.t [y,z] (simple triangular if using coupling layers)

#### 3.2 Invertible Architecture

* to increase model capacity: pad in- and output with zeros (if D is small), and apply permutations to the splitting of the vector at each layer (Kingma and Dhariwal 2018 learn permutations)

#### 3.3 Bi-directional Training

* collect gradients from for- and backward and then apply gradient step
* forward: simple loss (e.g. squared) between theory predictions y=s(x) and f\_y(x)
* backward: mismatch of joint distr. q(y(x),z(x)) and product p(y)p(z)=p(s(x))p(z) (Loss\_z)
* Loss\_z enforces p(z) to match the proposed distr. and for p(z|y)=p(z), s.th z and y are independent and do not encode the same information (by blocking Loss\_z by y?????)
* Loss\_z implemented by Maximum Mean Discrepancy (only requires samples from the distr´s) -> Jacobians J\_yz and J\_s do not need to be known explicitly
* after finite training y and z are still dependent, to speed up convergence we also define loss on hidden variable x Loss\_x through mismatch of the predicted distr q(x) with the observed distr p(x)
* Loss\_x is zero if Loss\_y and Loss\_z are zero, e.g. it does not alter optimum but only aids convergence
* if padding is used, also need loss for extra dims to be zero

#### 3.4 maximum mean discrepancy

* kernel-based comparison of two distr.s which are only accessible through samples
* discrimantor based approach usually preferred, but MMD is easier cheaper and more stable
* needs kernel with heavier tail than gaussian as design parameter (paper: inv. Multiquadratic 1/(1+|x-x\*|/h))

### 5. Conclusion

* how can method be scaled to higher dimensions where MMD becomes less effective?

**Conditional INNs for Diverse Image-to-Image Translation (Ardizzone et al 2021)**

* cINN combines generative INN with unconstrained feed-forward net to preprocess image into maximally informative features
* all parameters trained jointly w/ max. likelihood procedure
* INNs not as popular as GANs even though advantages like apparent resistance to mode collapse
* Image-to-Image translation like night to day and image colorization, or changing image style

### 1. Intro

* INN loss is quantitatively meaningful
* mapping of real images into latent space for explainability, editing
* connection to information theory, lossless compression, outlier detection
* cINN: given conditional image Y, model conditional distr p(X|Y) over different domain
* supervised learning with {Xi,Yi}, unsupervised is ill-posed
* until now, conditional generation with INNs only for class-conditionals, or others where condition directly contains high-level information
* because conditional is not part of invertible step-wise transformations and cant be passed through
* cINN uses conditional coupling blocks (CCBs)
* to inject useful info about conditioning in each resolution layer, use conditioning network producing *feature pyramids* C
* invertible pooling scheme based on *wavelets*
* it can be shown that learned features C from condition are maximally informative

### 2. Related Work

* mode collapse in GANs: cannot produce multiple diverse outputs
* there are approaches which produce visually different images, but no way to prove that these cover the entire distr
* conditional INNs can be divided into ones with conditional latent spaces, or ones with conditional INNs
* paper covers conditional INNs, only one other paper known with this approach, but w/o diversity

#### 3.1 cINN architecture

* CCBs: s1(u2) -> s1(u2,c) etc.
* resolution does not stay the same in each layer (was?), need c^(k) for each section
* downsampling with Haar wavelets (previously using checkerboard patterns): average 2x2 pooling aswell as vertical, horizontal and diagonal derivatives; reduces spatial dimension and separates high from low frequencies

#### 3.2 max likelihood training of cINNs

* as usual, assign probability to image x by q(x|theta, c)=p(z=f(x;theta,c)\*|(del\_f/del\_x)| (denote jacobian determinant (evaluated at x\_i) as J\_i)
* maximizing posterior over model parameters and using bayes theorem yields following *conditional max likelihood* loss:

Loss\_cML = E\_i[(0,5\*||f(x\_i;c\_i,theta)||^2-log|J\_i|]

* yields max likelihood parameters, can now sample for some c with x\_gen=g(z;c,theta) with z sampled from p\_z
* mode collapse virtually impossible: if any mode lies outside distr q(x|c,theta), latent vector will lie far outside p\_z and receive high correction (f(x;c,theta) maps likely images to center of gaussian p\_Z)
* in contrast GAN loss penalizes ignored modes much less
* also propagate loss through c to the feed-forward net phi. Useful features lead to lower Loss\_cML
* for two variables a and b, the *mutual information* is given as the KL div between joint and factored distr.: I(a,b) = KL(p(a,b)||p(a)p(b))
* information theory: learned features are the ones that are maximally informative about x
* when using grayscale image directly as input, not with conditional feed-forward net, semantic information about images is largely ignored

**BayesFlow: Learning complex stochastic models with INNs (Radev et al 2020)**

**-**uses simulation to find mapping from observed parameters to hidden params

-we assume good understanding of forward process (hidden to observable) which allows simulation

-likelihood-based approaches: likelihood p(x|theta) is known (x are observables, theta hidden)

-likelihood-free approaches: only sampling from likelihood is necessary given pairs (x\_i, theta\_i); usually realized by simulation programs with deterministic g(theta) plus random noise, i.e. x~p(x|theta) becomes x=g(theta, chi), chi~p(chi)

-here likelihood cannot be calculated, statistical inference is impossible

-likelihood-free problems: no closed form likelihood, statistical diff equations

-paper proposes new Bayesian method in terms of INNs

-bayesian modeling: p(theta|x) = p(x|theta)…

-here x = {x\_i} from N runs of forward process and fixed theta

-bayesian inverse modeling complicated because RHS always has to be approximated, forward-process loses info and usually noise is involved

-standard solution: approximative bayesian computation ABC: approximate posterior by sampling theta multiple times from proposed prior, then sampling dataset by using forward process. If dataset is similar to training sets, keep theta, else discard. Depends strongly on similarity measure

-more efficient methods optimize sampling from proposed distr (monte-carlo etc)

-above approaches are *case-based*, i.e. posterior estimation must be done for each sequence x\_1:N from scratch

-*amortized inference*: estimation split up into expensive upfront training phase, but much cheaper inference: learn approximate posterior p^(theta|x) that works well for any sequence x\_1:N

-must work for arbitrary dataset size N (amount of samples with the same hidden params)

-model should show correct *posterior contraction*: posterior p(theta|x\_1:N) should become sharper for larger N (simple, single-modal case: variance should be 1/N)

-*summary network* generate informative data from the sequence x\_1:N, independent of size N (usually done manually)

-*inference network* learns posterior from the summary statistics,; is an INN

-during inference, model can predict posterior of given params theta, or sample different parameters from the learned posterior distribution

-**Contributions:** globally amortized Bayesian inference, learning informative summary statistics

2. Related Work

-BayesFlow can be viewed as parameterization of inverible transport maps via INNs

2.2 Learning the Posterior

-Use reparamterization again: theta ~ p(theta) becomes theta = f^-1(x;z), z~p(z)

-goal: minimize KL div between p and p\_phi: argmin int p(x,theta) log p\_phi(x|theta)dtheta dx

-want parameters that maximize likelihood of theta given x

-can express expectation values by simulations of forward-process

-determinant term in loss controls volume change by trafo from theta to z

-three sources of incorrect posteriors:

-using MC to estimate expectation values

-imperfect summary statistics by network

-imperfect inference INN

4 Conclusion

-BayesFlow is applicable to any forward model which can be computer simulated

-summary network makes model shape-independent, automatically learns info which is most helpful for learning posterior

-BayesFlow samples from true posterior under convergence, needs no info about posterior shape to begin with, and posterior contraction is observed

**Normalizing Flows: An Introduction and Overview of current methods (Kobyzev et al 2020)**

1. Introduction

-generative modeling: modeling prob distr given samples drawn from it

-makes use of unlabeled data

-density estimation, outlier detection, dataset summarization, prior estimation

-analytic approaches: model distr by function family

-variational approaches and maximum likelihood use latent spaces, additional flexibility but also complexity

-graphical models, neural generative models since 2014 (GANs and VAEs, don’t allow posterior evaluation of new points)

2. Background

-Normalizing Flows (NF) made popular in 2015 by Rezende and Mohamed

-NF is combination of differentiable and invertible trafos of simple distr to target distr

-sample density then is density of simple distr multiplied by change of volume by trafos (Jacobian)

-z from latent space (simple distr) and y=g(z), f=g^-1

-then p\_Y = p\_Z |det J(f(y))| = p\_Z |det J(g(z))|^-1 is the pushforward of density p\_Z to y-space (g\_\*p\_Z)

- pushes p\_Z (sometimes called noise) to complicated distr. “generative direction”

-f flows in normalizing direction (towards normal form p\_Z)

-if g=gN ° … g1 then f=f1 ° … fN, det J f = prod det J fi

-the normalizing flows need not be bijections, they can also be piecewise differentiable

-usually params are determined by maximum likelihood estimation, but other losses (e.g. adversarial, Flow-GAN 2018) are possible

2.2.2 Variational Inference

-latent variable model: p(x) = int p(x,y)dy

-posterior distr p(y|x) is used but usually intractable, usually introduce approximate posterior q(y|x,theta) and minimize KL div with true posterior, optimization needs non-trivial gradient of expectation value w.r.t. posterior of h(y) (entropy?)

-use flow to model posterior, take expectation value w.r.t. normal variable z

-then, only likelihood of sampled points is necessary

3. Methods

-NFs need to be invertible, because for sampling g is necessary, for likelihood approximation f is needed; expressive enough to reproduce desired distribution; f, g and jacobian det should be efficiently computable

-likelihood: likeliness of observables to arise given hidden parameters

-posterior: probability that, given an observation, these hidden params are true

3.4 Coupling and autoregressive flows

-two most used flow architectures

-multi-scale flows: start with low-dimensional z that gets transformed, z\_aux is not transformed. Each layer, more of the [z, z\_aux] are included (can capture multi-scale nature of e.g. images)

-how to partition x? RealNVP, Glow

3.4.2 Autoregressive flows

-autoregressive model is non-linear generalization of triangular matrix: yi = h(xi; Theta\_i(x\_1:i)) Theta\_i are the *conditioners*

-> diagonal jacobian, det is product of diagonal elements

-masked autoregressive flows by Papamariakos

-calculation of inverse inherently sequential, cant be parallelized xt = h^-1(yt,Theta\_t(x1:t))

-Inverse Autoregressive Flows: yt = h(xt, Theta\_t(y1:t)) same form as inverse of AF, like modeling flow in the generative direction (usually in the normalizing direction, so complicated to normal)

-IAFs better for fast sampling, (masked) AFs better for density estimation

-for several autoregressive flows, the universality property is proven, meaning that it can model any prob distr

-affine coupling function: h(x,theta) = theta1\*x + theta2 (NICE, Glow, RealNVP, MAF, IAF) (not very expressive, must be stacked)

-nonlinear squared flow h(x;theta) = a+ bx+ c/(dx+h)^2 can learn multimodal distributions

-continuous mixture CDFs

-Splines

-4 architecture categories: coupling, autoregressive, residual (x+F(x)), ordinary differential equation-like (no need to calculate determinant, need ODE solver)

4. Datasets and Performance

-Flow++ (continuous mixture CDFs) has best performance, change in dequantization approach

5. Discussion and open problems

-inductive bias through base measure p\_Z: one might use more complex base measure, trade-off between complexity in base measure and generative transformations was studied by Jaini et al 2019

-other diffeomorphisms than triangular (coupling, autoregressive), residual or ODEs?

-what inductive bias comes with the architecture of the flow?

-conditional normalizing flows? Proposed different flows for each condition (inefficient) Trippe and Turner 2017, Atanov et al 2018 propose affine couplings with parameters dependent on condition; conditional NFs are especially useful for time series

-different losses: adversarial (Flow-GAN) and minimizing Wasserstein distance

-flows on manifolds

-normalizing flows on discrete distributions, Tran et al 2019 show change of variable formula applies without the jacobian

-straight-through gradient estimator to backprop functions on discrete variables

-hoogeboom et al 2019 model flows between discrete spaces with additive coupling layers, other model discrete variables to continuous latent variables with autoencoders and then apply NFs to continuous latents

-dequantization for ordinal variables like image pixel values, dequantization can be learned

**Normalizing Flows for Probabilistic Modeling and Inference (Papamakarios et al 2021)**

2.1 Normalizing Flows Definitions and Basics

-invertible and differentiable functions are composable

-2 functions of flow-based models: sampling (needs to sampel from p\_u and to perform forward transform T), and density calculation (needs to apply T^-1, jacobian determinant, and evaluation of p\_u)

2.3 Using Flows for Modeling and Inference

Forward KL divergence optimization:

-D\_KL(px\*(x||px(x;theta))

-p\_u(T^-1(x)) depends on the parameters of base measures psi and params of the transformation phi, the jacobian determinant only depends on phi (it doesn’t yet consider the evaluation of p\_u(u; psi) at u=T-1(x))

-flow model can be trained without being able to compute T or being able to sample from p\_u, only need these operations when sampling from the model

-one estimates the expectation values through monte carlo estimation with samples from px\*(x)

Reverse KL optimization:

-D\_KL(px(x;theta)||px\*(x))=…=E\_pu[log pu(u;psi) – log |det JT(u,phi)| - log px\*(T(u,phi))]

-suitable if we can evaluate px\*, even if it is only known without normalizing factor C

-monte carlo estimate with samples from pu(u)

-one can assume the parameters of base measure psi to be fixed (WHY???) and therefore only need gradient w.r.t. phi of the loss

-no need to evaluate base density or compute T-1

-reverse KL often used for *variational inference*, or in *model distillation*, when px\* can be sampled from but is otherwise inconvenient

-other perspective: for given T, pu\* is what comes if px\* is passed through T-1. Therefore, fitting px to px\* is equivalent to fitting pu to pu\*

-change of variables can be used to show DKL(px||px\*)=DKL(pu||pu\*) (same goes for p<->p\*)

-using different divergences shows connection to more general class of implicit probabilistic models, which are not restricted to only compositions of changes of variables

2.4 Brief historical overview

-first predecessor: whitening transformations, transforming data into white noise (1966, 1987)

-2000 Chen and Gopinath *Gaussianization* first to use whitening for density estimation

-2010 tabak and vanden-Eijnden approach gaussianization from perspective of diffusion

-2013 tabak and turner introduce modern conception of normalizing flow with K transformations

-2013 rippel and adams: normalizing flows with neural networks

-2015 dinh et al: scalable architecture (NICE)

-many other predecessors if one looks at the aspect of change of variables (like Liouvilles theorem in statistical mechanics, 1838, flow through phase space as incompressible fluid)

3.1 Constructing Flows Part 1: Finite Compositions

-log |det(JT(z0))| = sum\_K log|det(JT\_k(z\_k-1))|

-computational complexity is of O(K)

-ensuring invertibility of f\_k does not mean that inverse is easily tractable (f\_k either being T\_k or T^-1\_k)

-inverse of f\_k always needed, either for density evaluation or for sampling, so it depends on use case whether to model T\_k or T^-1\_k

-in general, jacobian caluculation is O(d^3), but it should be at most O(d) to be tractable in most scenarios

3.1 Autoregressive flows

-function implemented as z´\_i = tau(z\_i;h\_i) with h\_i=c\_i(z<i)

-tau is the strictly monotonic transformer which is parametrized by h, the conditioner c calculates the parameters h of the transformer from the “previous” z\_i. the conditioners parameters are usually the parameters of f\_phi

-the conditioners do not need to be bijections

-forward does not need to be calculated in specified order, because z´\_i do not depend on each other. But for reverse calculations, the z\_i depend on the h\_i which depend on the “previous” z\_i

-in this setting the jacobian is triangular, and log|det(J\_f\_theta(z))|=sum log |del tau/ del z\_i (z\_i,h\_i)| which can be calculated in O(D)

-autoregressive flows are general approximators

-designing autoregressive flow depends on choice of transformers and choice of conditioners

-simple and first used type of transformers is the affine transformation: z´\_i=alpha\_i z\_i + beta\_i, h\_i={alpha\_i,beta\_i}

-invertible if alpha\_i isn’t zero, which can be achieved by alpha\_i=exp alpha`\_i

-log |det(J\_T)| =sum alpha´\_i  
-limited expressitivity, because if z is gaussian, conditionals p(z´\_i|z´<i) will also be gaussians. Stacking autoregressive layers can help, but it is not sure if autoregressive models are general approximators or not

-following combinations of monotonic functions tau\_k are also monotonic: conic: tau(z\_i) = w0i + sum omega\_k\*tau\_k(alpha\_ki z\_i + beta\_ki) (like a single-layer perceptron), composition

-can approximate any monotonic function arbitrarily well

-combination transformers can in general not be inverted analytically, but via bisection search etc.

-integration-based transformers

-spline-based transformers: monotonic simple functions defined on [x\_i, x\_i+1)

-usually meeting points z\_i and values here z`\_i and the derivatives at these points are taken as the transformers parameters h\_i

-locating right segment is O(log K), and invertibility and evaluation are easy

-conditioners c\_i(z<i) can be arbitrary, but usually params are shared or c=c\_i

-with recurrent NNs: h\_i = c(s\_i) with s\_i = RNN(z\_i-1, s\_i-1)

-turn an inherently parallel computation to sequential one

-masked conditioners: use feed forward NN which generated h from z, and make sure that there is no connection from z\_i to outputs h1,…,hi

-assign degree 1,…,D to each input, hidden, output neuron and make sure that no neuron feeds to other neuron with higher or equal degree

-inverting masked autoregressive flows is complicated, because to calculate z\_i = tau-1(z´\_i,h\_i), we need h\_i which depends on z<i. therefore one needs to calculate h1, then z1, then h2, then z2 etc.

-initialize z, calculate (h1,…,hD) = c(z), calculate z\_i=tau-1(z´\_i,h\_i) and update z

-this requires D calls of the conditioner network, in comparison to the forward calculation which only requires one pass of z through c

-alternative way to invert flow (song et al 2019) is using newton style method is to find z by zk+1=zk-alpha\*diag(J\_f\_phi(zk))^-1\*(f\_phi(zk)-z´), where z0 = z´ is convenient initialization

-if it converges, it must converge to z because only then does gradient step vanish because f\_phi(z) = z´

-more efficient than other method if number of steps for convergence is less than D

-only guaranteed to converge locally

-in spite of problems with inversion masking is most popular method for autoregressive flows

-Coupling layers

-with autoregressive flow, either sampling or density evaluation is D-times more complex than the other

-coupling layers are computationally symmetric

-conditions: for d (usually D/2): h1…hd are constant, and hd+1…hD depend only on z1…zd

-D coupling layers are proven to be a universal approximator if d of layer I is equal to i-1 for each layer, but do then not have a computational advantage over autoregressive flows

3.1.2 Connection between Autoregressive models and Autoregressive flows

NICHT VERSTANDEN

3.2 Linear Flows

-for models of finite capacity, which is always the case in practice, the order of input variables determines which distributions the flow can represent

-index permutations can be described by permutation matrices in between the flow (autoregressive or coupling) layers, z´=Wz where W is a permutation matrix. The jacobian of these trafos is just W, and |det W| is 1 for permutation matrices

-straightforward implemtation of learning W is problematic, because inversion is O(D^3) (if its invertible at all) and detW is also O(D^3)

-restrict W to structured matrix like triangular (inversion O(d^2), det O(d)) or multiplications of structured matrices

-it is impossible to parametrize all DxD invertible matrices in a continuous way, so every parametrization will leave out some invertible matrices (because det is continuous in matrix entries, and therefore universal parametrization of R^D^2 to DxD matrices will have to go through region with determinat zero, which is not invertible => at least two separate islands of invertible islands, one may choose to only parametrize invertible matrices with either pos or neg determinant)

3.3 Residual Flows

-residual trafos of form z´=z+g\_phi(z), invertible with special properties on g

3.3.1 contractive residual flows

-contractive trafo: if for some distance function d and some L<1 it holds: d(g(a),g(b) <= L\*d(a,b)

-Banachscher Fixpunktsatz: each such trafo has fixed point with g(a) = a

-if g\_phi is contractive, z´=z+g\_phi(z) is invertible

-z\_k+1 = z´-g\_phi(z\_k) converges to z=f\_phi^-1(z´)

-as L gets smaller, approximation of z gets faster, but the trafo function gets less expressive

-no general efficient procedure to compute residual flows jacobian determinant

-good expressability because jacobian is dense i.g., i.e. all outputs depend on all inputs

-but density evaluation is expensive and sampling is done iteratively

3.3.2 matrix determinant lemma based residual flow

-matrix determinant lemma: det(A+VW.t)=det(I+W.tAV)det(A)

-if A is DxD and V,W are MxD, LHS is O(D^3+MD^2) RHS is O(M^3+DM^2), RHS preferable if M<D

-flows making use of this simplified det calculation:

-planar flow: contraction/expansion perpendicularto a hyperplane characterized by v,w in R^D and b in R

-Sylvester flow: generalization of planar flow to v,w being in R^DxM

-both have invertibility conditions of the form w.t\*v > -[ sup\_x sigma´(x)]^-1

-radial flows: contraction/expansion with center z\_0

-all of these have jacobian determinants which take O(D) to compute and are invertible if params are suitably restricted

-but, no general analytic calculation of inverse possible (used for calculating posteriors in variational inference) and trafos are simple so its not clear how expressive these flows are

3.4 Practical Considerations when Combining Flows

-example: dhariwal et al used 320 subtrafos on 40 GPUs (Glow 2018)

-batch norm: has diagonal jacobian and (with mu and sigma (batch statistics) as given) has free params alpha(scale) and beta(translation)

-composite T\_k o BN o T\_k-1

-alternative for small batch sizes where batch statistics can become noisy and destabilize training: *activation norm*

-Multiscale architectures (Dinh et al 2017) clamping out features at certain layers

4. Constructing Flows Part 2: Continuous-time transformations

-previously one-step trafos zk=T(zk-1)

-continuous trafos by ODEs, step is replaced by “time”

-zt0 = u and zt1 = x, dzt/dt = g\_phi(zt,t)

-Picards existence theorem: if g\_phi(zt,t) is globally (w.r.t. t) Lipschitz-continuous in zt there exists a unique solution zt

-has to also be continuous in t

-many NNs fulfill these requirement, and g\_phi need not be invertible as in section 3

- x = z1 = u + int from t0 to t1 g\_phi(zt,t)dt; u = z0 = x – int

-> continuous flows are computationally symmetric

-chen et al: d log p(zt) / dt = -Tr{J\_g(t,-) (zt)}, jacobian needs O(D) backpropagations

-use Hutchinson trace estimator: Tr(A) ~ v.t A v, with v a Gaussian vector. This can be calculated with a single backpropagation

-can also constrain g\_phi s.th. trace can be calculated exactly with single backprop(Chen et al 2019)

-calculate log px = log pu + int -Tr(…) dt

-evaluate combined 2d integral with numerical integrator

4.2 Solving and Optimizing Time-continuous flow

-can leverage vast numerical ODE literature

4.2.1 Euler Flows and equivalence to residual flow

-Euler calculates small timestep discreetly (runge-kutta e.g.) and makes this approach equivalent to residual flows

-epsilon \* g\_phi(zt,t) is contractive if epsilon<1/L with L the Lipschitz constant (=> contractive residual flow)

-using taylor expansion we can deduce in epsilon->0 limit: d log p(zt)/dt = -Tr{J\_g\_phi(zt)}

4.2.2 Adjoint method

-for a general Loss Loss(x;phi) like log likelihood, chen et al in 2018 showed that d/dt (del Loss/del zt) = -(del Loss/del zt).t del g\_phi(t,zt)/del zt

-this result is widely known as the *adjoint sensitivity method* (pontryakin 1962)

-continuous-time equivalent of backpropagation

-del Loss / del phi = int from t1 to t0 (del Loss/del zt) del g\_phi(t,zt) / del phi

-both forward evaluation and backpropagation can be solved with blackbox ODE solvers

-This results in significant practical benefits since backpropagating through a solver is costly both in terms of computation and memory requirements wieso??

5. Generalizations

5.1 General Probability-Transformation Formula

-int\_(u is in omega is part of U) p(u) dmu(u) = int\_(x is in gamma is part of X) p(x) dnu(x), where gamma = T(omega)

-conservation of probability measure

5.2 Piecewise Invertible Transformations and Mixtures of Flows

-we can treat many-to-one transformations T, where multiple u´s are mapped to the same x

-piece-wise invertible transformations, where we define countable subsets U\_i such that T:U\_i to X are invertible

-px(x) = sum\_i pu(T\_i^-1(x))|det J\_T^-1\_i(x)| (*mixture of flows)*

-*Real and discrete (RAD)* can be thought as the inverse as mixture of flows, where the same u is assigned to multiple x

-can think of a setting where one of the u -> X trafos T\_i is chosen at random, U becomes U x I

-px(x) = pu(T^-1\_i(x))p(i(x)|T^-1\_i(x))|det J\_T\_i^-1(x)|

-where pu(T^-1\_i(x))p(i(x)|T^-1\_i(x)) = p(u,i)

-p(i) is the so-called *mixture weight*

5.3 Flows for Discrete Random Variables

-if the measures dmu(u) and dnu(x) of the general prob-trafo formula are the counting measures, one obtains sum (u in omega) pu(u) = sum (x in gamma) px(x)

-with gamma = x for arbitrary x, we get px(x) = sum (u in omega) pu(u), where omega is the set of all u with T(u) = x, and if T is bijective we get px(x) = pu(T-1(x))

-such *discrete flows* do not include a jacobian

-discrete flow based on autoregressive flows on whole numbers by Hogeboom et al 2019 implement transformer tau(zi;beta\_i) = zi + round(beta\_i), beta\_i = beta\_i(z<i)

-discrete flows can are not universal approximators, e.g. uniform distr. is necessarily transformed to uniform trafo, in general a discrete flow can not change the probabilities but only permute them, due to absence of jacobian term in change-of-variable formular´

-with standard flows, one can model any distr with fully factorized base distribution pu(u) = multiply pui(ui). For discrete flows this is in general not possible, because it would mean that the table p(u1,u2) would have to always be representable as an outer product of two vectors pu1(u1) and pu2(u2), which is only possible if there is a permutation of p(u1,u2) which has rank 1

-this problem can be circumvented if U and V are extended with zeros to make higher order matrix which can be permutated to rank 1 matrix

-one can also define the base distributions autoregressively

5.4 Flows on Riemannien Manifolds

-embedding of D-dim manifold into R^M via mapping T: R^D -> R^M with M>=D, x=T(u)

-metric of tangent space X in point x on manifold given by G(u) = J\_T(u).t J\_T(u)

-infinitesimal volume dnu(x) = sqrt(det(G(u))) du

-pu(u) = px(T(u)) sqrt(det(G(u)))

-invertible if the range of T is restricted to X

-results in usual change of variable formula if X = R^D and M=D

-invertibility of T requires manifolds to be homeomorphic to R^D (topologically equivalent)

-alternative approach focusses on cases where manifold is a also a Lie group

5.3 Bypassing topological constraints

-due to T being diffeomorphism, e.g. flow cant map R^D to S^D or R^D to R^D´

-each intermediate transformation must be topologically equivalent to beginning and end of transformation. A way to make the transformations more flexible, is to make the trafo from R^D+rho to R^D-rho, in artificially higher-dimensional space

5.6 Symmetric Densitites and Equivariant Flows

-often problems have known symmetries (like interaction between two particles is symmetric)

-one tries to incorporate these symmetries into the model

-symmetries are trafos under which prob density does not change. Preservation of probability yields that |det Jg|=1 if g is a symmetry trafo

-set of all symmetries of prob distr is closed under composition etc and is therefore a group

-T is *equivariant* if T(gu) = gT(u) for any symmetry g (T from R^D to R^D)

-Lemma of equivariant flows: if T is equivariant wrt G and pu(u) is invariant under G, then px(x) is invariant under G (G is the symmetry group)

-Lemma equivariance from invariance: f:R^D->R invariant wrt G (eg f(u)=f(gu)) and assume that g´s representation Rg is orthogonal for all g in G. then the gradient wrt u of f(u) is equivariant wrt G

-useful because usually its easier to find invariant functions then equivariant, e.g. all functions of the L2 norm are rotation invariant

6. Applications

6.1 Probabilistic Modeling

-normalizing flows are expressive but can still calculate exact likelihoods

-often data are discrete like pixel image values from 0 to 255, often dequantized adding continuous noise (variational dequantization: noise distr. is learned)

-usually fitting via *maximum likelihood estimation*, using KL divergence:

KL=-E\_p\*(x)[log p\_theta(x)]+const ~ -1/N sum log px(xn;theta) + const

~-1/N sum log pu(T-^(xn;phi);psi) + log |det J\_T-1(xn;phi)| +const

6.1.1. Density Estimation

-2011 first real data application: Laparra et al detect urban areas in satellite images

6.1.2 Generation

-text: direct way to apply normalizing flows to text generation is by discrete flow over vocabulary

6.2 Inference

-estimation of unknown quantities in a model

-*likelihood-free inference:* easy to simulate x from hidden variables eta, but intractable to calculate p(eta|x)

**FrEIA:**

FrEIA.framework: enthält die node classes (input, output, regular, conditional) und GraphINN, ReversibleGraphINN, SequenceINN und ReversibleSequential