Modern Normalising Flows

Denis Derkach

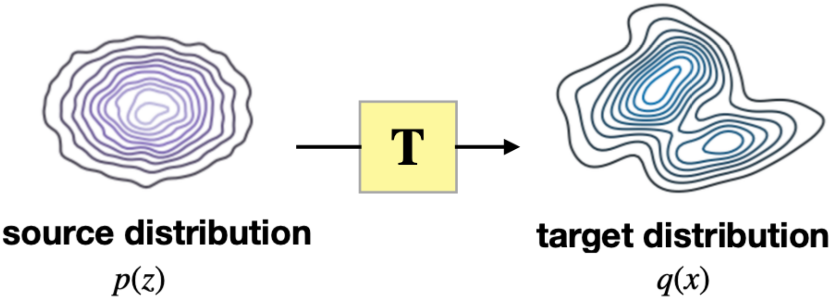
CS HSE faculty, Generative Models, spring 2020

Contents

Basic Understanding

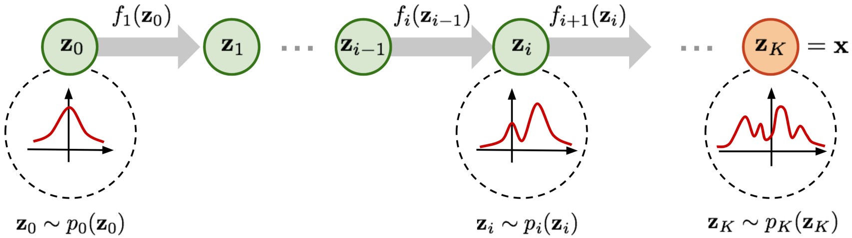
Reminder: Motivation

Quite often we want to sample from some distribution (but we do not know how to do this). What we know is how to sample from a simple pdf: Gaussian, uniform or something.



We want a deterministic map T from source to target density.

Definition



For definition:

$$z_m = f_\theta^m \circ f_\theta^{m-1} \cdots \circ f_\theta^1(z_0) = f_\theta^m(f_\theta^{m-1}(\ldots(f_\theta^1(z_0)))) = f_\theta(z_0),$$

we have $z_m \to x$:

$$p(\mathbf{x};\theta) = q(f_\theta^{-1}(\mathbf{z})) \prod_{i=1}^{m} \left| \det \frac{\partial (f_\theta^i)^{-1}(\mathbf{z^m})}{\partial \mathbf{z^m}} \right|.$$

Note that we have invertible transformations.

Triangular Jacobian

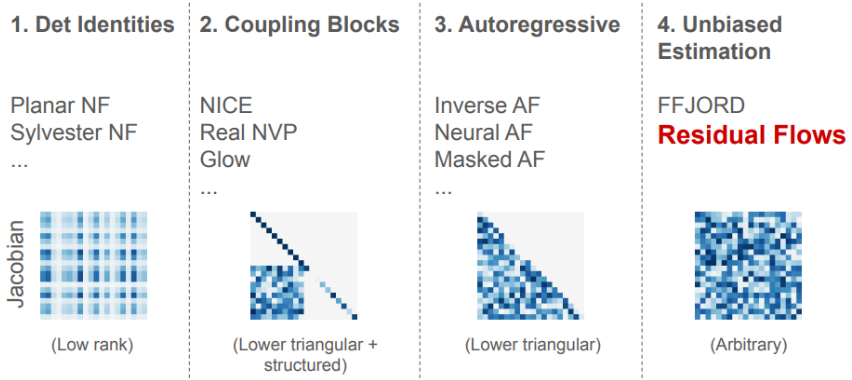
- There always exists a unique (up to ordering) increasing triangular map that transforms a source density to a target density (see Bogachev et al. for detail).
- In previous lectures we considered the triangular Jacobian:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial z_1} & \dots & 0 \\ \dots & \dots & \dots \\ \frac{\partial f_1}{\partial z_n} & \dots & \frac{\partial f_n}{\partial z_n} \end{pmatrix}$$

  It describes the flow with transformation $x_i = f_i(z)$ that only depends on $z \leq i$.

Developments

More results are produced in 2019. In general, they can be separated into following categories.



Masked Autoregressive Flows

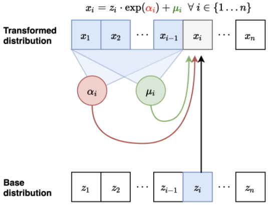
Reminder: Autoregressive models

- Take Autoregressive Model:

$$p(x) = \prod_{i=1}^{n} p(x_i|x_{i<1}).$$

  such that
  $p(x_i|x_{i<1}) = \mathcal{N}(\mu_i(x_1, \ldots, x_{i-1}), \exp\left(\alpha_i(x_1, \ldots, x_{i-1})\right)^2)$,
  with μ and α are Neural network outputs.
- We have a direct estimation of likelihood in this model.
- To sample, we need to go through consecutive steps:
  - $z_i \sim \mathcal{N}(0; 1)$;
  - $x_1 = \exp(\alpha_1) z_1 + \mu_1$
  - $x_2 = \exp(\alpha_2(x_1)) z_1 + \mu_2(x_1)$
  - and so on.
- Might be stacked as Flow from Gaussians to observable space.

Masked and Inverse Autoregressive Flow (MAF/IAF)

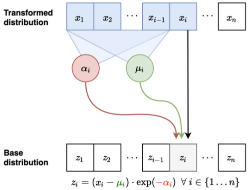


- looks similar to MADE;
- Forward mapping $z \mapsto x$:
  - $z_i \sim \mathcal{N}(0; 1)$;
  - $x_1 = \exp(\alpha_1) z_1 + \mu_1$
  - $x_2 = \exp(\alpha_2(x_1)) z_1 + \mu_2(x_1)$
  - and so on.
- sampling is sequential and slow.

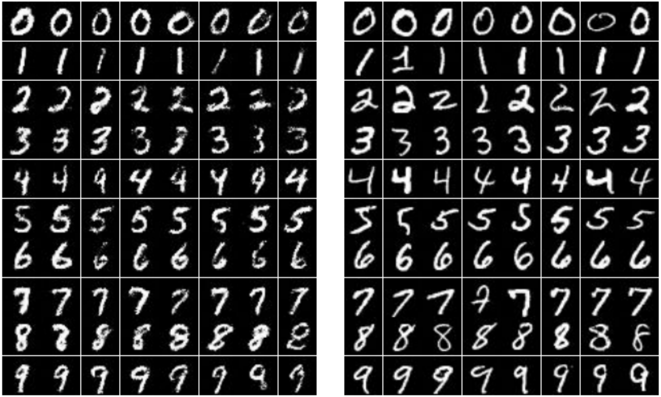
From Stanford GM lectures by S. Ermano G. Papamakarios et al. Masked Autoregressive Flow for Density Estimation

MAF: Inverse

- Inverse mapping $x \mapsto z$:
  - Compute all μ_i and α_i
  - $z = \exp(-\alpha_1) \odot (x - \mu)$
- Jacobian is lower diagonal, hence determinant can be computed efficiently.
- Likelihood evaluation is easy and parallelizable.
- Thus, the training is relatively fast.



MAF:results



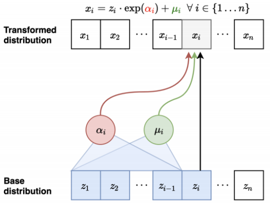
(a) Generated images

(b) Real images

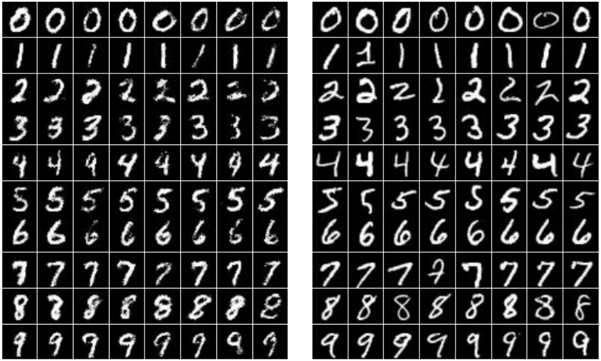
Figure 3: Class-conditional generated and real images from MNIST. Rows are different classes. Generated images are sorted by decreasing log likelihood from left to right.

Inverse Autoregressive Flow (IAF)

- Forward mapping $z \mapsto x$:
  - Sample all z_i;
  - Compute all μ_i and α_i.
  - $x = \exp(-\alpha) \odot (z - \mu)$.
- Inverse mapping $x \mapsto z$:
  - Sequential calculation.
  - $z_i = \exp(-\alpha_i(z_{<i}))(x - \mu_i(z_{<i}))$.
- Fast to sample from, slow to evaluate likelihoods of data points (train).



IAF:results



(a) Generated images (b) Real images

Figure 3: Class-conditional generated and real images from MNIST. Rows are different classes. Generated images are sorted by decreasing log likelihood from left to right.

MAF and IAF

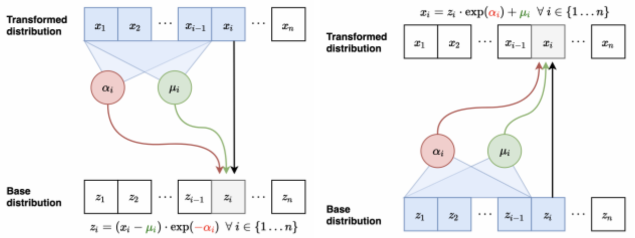


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- MAF and IAF use autoregressive transformations based on MADE building block.
- One can see that IAF forward mapping and MAF Inverse mapping are connected up to parameterisation.
- In fact, they are inverse of each other.

MAF and IAF

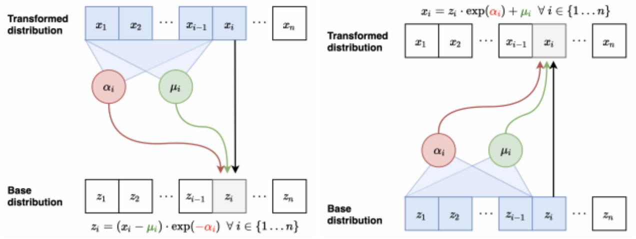
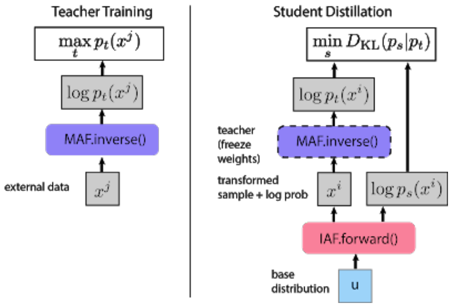


Figure: Inverse pass of MAF (**left**) vs. Forward pass of IAF (**right**)

- MAF: Fast likelihood evaluation, slow sampling - best for training based on MLE, density estimation.
- IAF: Fast sampling, slow likelihood evaluation - best for for real-time generation.

Teacher-Student Model



- Two part training with a teacher and student model.
- Teacher (MAF) trained first, than student (IAF) initialised.
- Student model cannot efficiently evaluate density for external datapoints but allows for efficient sampling.

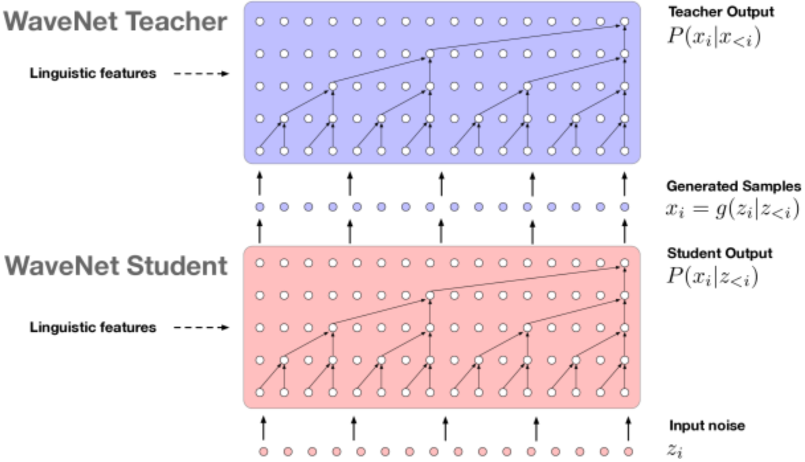
Probability Density Distillation

- Student, s, is trained to match the teachers' distribution t using KL divergence:

$$KL(s,t) = \mathbb{E}_{x\sim s}[\log s(x) \log t(x)]$$

- Training:
    - Train teacher via MLE and obtain likelihood.
    - Train student to minimize KL divergence.
    - Use student to sample.
- Improves sampling efficiencies by a factor 100 for Wavenet.

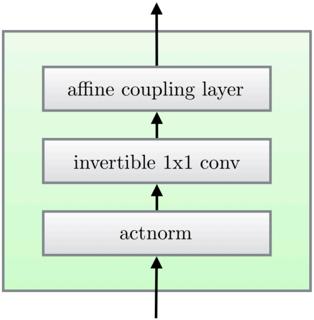
Parallel Wavenet



Gives fast and efficient in training algorithm for sound generation.

GLOW

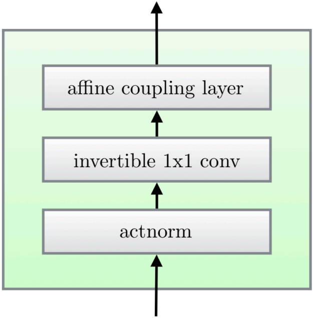
Generative Flow with Invertible 1x1 Convolutions



- Updates NICE and RealNV and follwing their idea.
- Uses block with several layers.

Blog by L. Weng
D. Kingma et al. Glow: Generative Flow with Invertible 1x1 Convolutions

GLOW: Layers

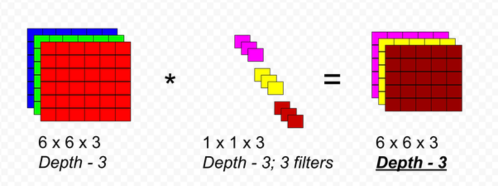


- Activation normalization (short for “actnorm”): affine transformation using two trainable parameters (scale and bias).
- Invertible 1x1 conv: generalization of any permutation (like r-NVP) of the channel ordering.
- Affine coupling layer. Similar to rNVP.

Invertible 1x1 conv layer

- We have an invertible 1x1 convolution:

$$f = \text{conv}2D(h, W).$$



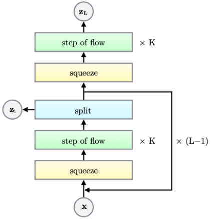
- We need to compute the Jacobian determinant $|\det \partial f / \partial h|$.
- In fact :

$$\log |\det \frac{\partial \text{conv}2D}{\partial h}| = \log(|\det W|^{h \cdot w}) = h \cdot w \log(|\det W|).$$

The latter operation can be computed using PU decomposition for matrix W of size $c \times c$ as $\mathcal{O}(c)$.

Summary of layers transformation

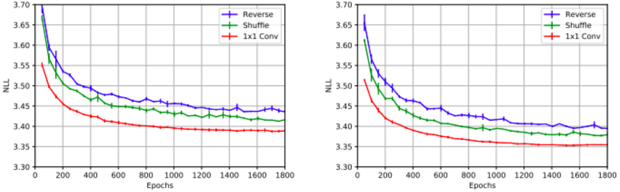
Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathtt{sum}(\log\lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log\lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \mathtt{sum}(\log\lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log\mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log\mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log\mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log\mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$



(b) Multi-scale architecture (Dinh et al., 2016).

Each layer is than followed by a subsequent multi-scale procedure (like it was done in NICE and r-NVP).

Additive vs Affine



(a) Additive coupling.

(b) Affine coupling.

Figure 3: Comparison of the three variants - a reversing operation as described in the RealNVP, a fixed random permutation, and our proposed invertible 1×1 convolution, with additive (left) versus affine (right) coupling layers. We plot the mean and standard deviation across three runs with different random seeds.

Authors claim that:

- Affine faster than additive.
- 1x1 convolution performs like better randomisation.

Unfortunately, to train on celeba, one needs a lot of GPU-days.

Sampling Temperature

- In order to get more realistic sampling, one can use a reduced-temperature model.
- In this work:

$$p_{\theta,T}(x) \sim p_{\theta}^{T^2}(x)$$

- Temperature is a free parameter for sampling.

R. Dahl et al. Pixel Recursive Super Resolution

GLOW: Results

Figure 4: Random samples from the model, with temperature 0.7

Dependence on the Depth

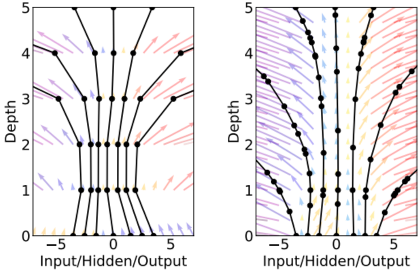
Figure 9: Samples from shallow model on left vs deep model on right. Shallow model has $L = 4$ levels, while deep model has $L = 6$ levels

Discussion

- Adds additional parameters

FFJORD

Motivation

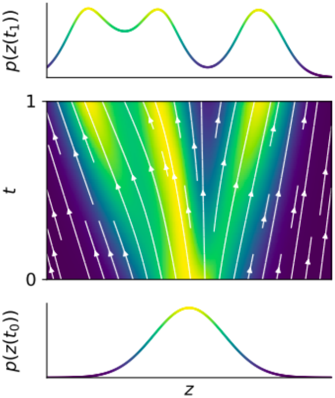


We can relax even more restrictions:

- Do we really care of having discrete steps?
- Can we change the Jacobian to something more stochastic?
- We than thing of system of continuous-time dynamics.
- This ideas led to a branch called NeuralODE.

Chen et al. Neural Ordinary Differential Equations

Continuous Normalizing Flows



Model the generative process with continuous dynamics:

$$z_0 \sim p(z_0)$$

$$\frac{\partial z}{\partial t} = f_\theta(z_t, t)$$

$$x = z_t = z_0 + \int_{t_0}^{t_1} f_\theta(z_t, t)dt$$

To obtain the density we solve the initial value problem (IVP) under mild conditions:

$$\log(p_x) = \log(p_{z_0}) - \int_{t_0}^{t_1} \mathrm{Tr}\frac{\partial f(z(t))}{\partial z(t)}dt$$

What this means

- log-probability of the data under the discrete model:

$$\log p(x) = \log p(z_0) + \sum_{t=0}^{T} \log |\log \partial F^{-1} / \partial z_t|$$

- log-probability of the data under continuous model:

$$\log(p_x) = \log(p_{z_0}) - \int_0^1 \mathsf{T}r \frac{\partial f(z(t))}{\partial z(t)} dt$$

- sum of jacobian log-determinants $\longrightarrow$ integral of jacobian trace. This give $\mathcal{O}(N^3)$ caclulations.

Unbiased Log-Density Estimation

We can use stochastic trace estimation. For any matrix A and a distribution $p(e)$ over vectors where $\mathbb{E}[e] = 0$, $cov[e] = I$, we used Hutchinson's estimator:

$$Tr(A) = \mathbb{E}_{p(e)}[e^T A e]$$

Which brings us to calculable:

$$\log(p_x) = \log(p_{z_0}) - \mathbb{E}_{p(e)} \int_{t_0}^{t_1} e^T \frac{\partial f(z(t))}{\partial z(t) e} dt$$

The existence and uniqueness of solution requires that f and its first derivatives be Lipschitz continuous and can be calculated in $\mathcal{O}(N)$.

Training with adjoint Backprop

- We need to compute $\partial L / \partial \theta$.
- Given scalar objective:

$$\mathcal{L}(z_1) = \mathcal{L}\left(\int_0^1 f(z(t), t, \theta) dt\right)$$

- we can obtain $\partial \mathcal{L} / \partial \theta$ for gradient-based optimization by solving another IVP.

We define a new quantity, the adjoint, a_t, which has dynamics $\frac{\partial a_t}{\partial t}$

$$a_t = -\frac{\partial L}{\partial z_t} \qquad \frac{\partial a_t}{\partial t} = -a_t^T \frac{\partial f(z_t, t, \theta)}{\partial z_t}$$

then solving backwards in time gives the desired gradients of the loss with respect to the parameters

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} a_t^T \frac{\partial f(z_t, t, \theta)}{\partial \theta} dt$$

This allows us to use a black-box ODE solver to compute z_1 and also $\partial L / \partial \theta$.

FFJORD: results

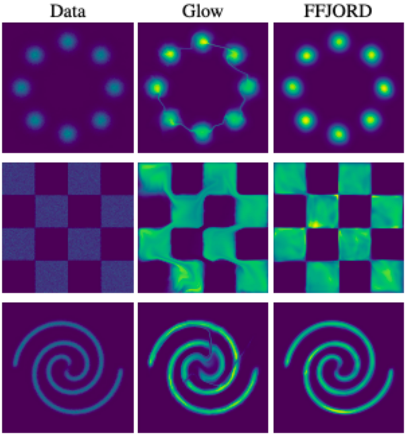


Figure 2: Comparison of trained FFJORD and Glow models on 2-dimensional distributions including multi-modal and discontinuous densities.

FFJORD: discussion

- Advantages
  - Guaranteed inverse by reversing order of integration, regardless of model parameterization
  - Efficient, unbiased log-probability estimation without restricting the Jacobian of the transformation
  - Does not require dimension splitting or ordering choices
  - Reversible generative models can now be defined with standard neural network architectures
- Disadvantages
  - Must rely on adaptive numerical ODE solvers for stable training
  - Computation time determined by solver, not user
  - Currently 4-5x slower than other reversible generative models (Glow, Real-NVP)

Conclusion

	Method	Train on data	One-pass Sampling	Exact log-likelihood	Free-form Jacobian
	Variational Autoencoders	✓	✓	✗	✓
	Generative Adversarial Nets	✓	✓	✗	✓
	Likelihood-based Autoregressive	✓	✗	✓	✗
Change of Variables	Normalizing Flows	✗	✓	✓	✗
	Reverse-NF, MAF, TAN	✓	✗	✓	✗
	NICE, Real NVP, Glow, Planar CNF	✓	✓	✓	✗
	FFJORD	✓	✓	✓	✓

Table 1: A comparison of the abilities of generative modeling approaches.