NORMALIZING FLOWS

IN THE SEARCH FOR MODELS THAT CORRECTLY DESCRIBE THE PROCESSES THAT PRODUCE THE DATA

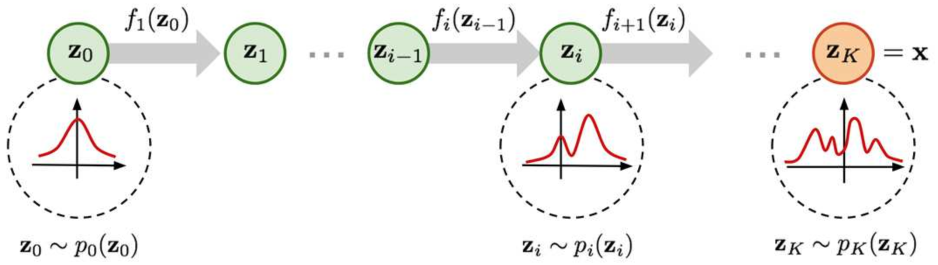
Source: PointFlow: 3D (Yang et. al)

Agenda

1. Intuition behind Normalizing Flows
2. Family of generative models and merits of Normalizing Flows
3. Mathematical definition
4. Constructing flows with finite composition
    1. Review of transformation methods
    2. Review of conditioning methods
5. Other common architectures
6. Comparison of different methods

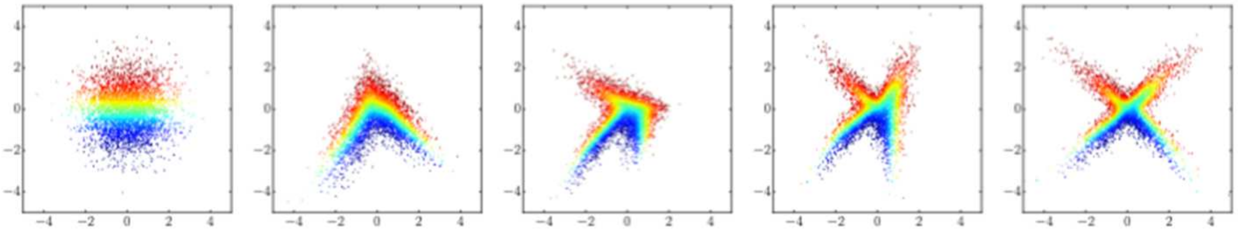
The intuition behind the normalizing flows

- Transformation T is expanding and contracting the space in order to mold the density $p_u(u)$ into $p_x(x)$
- $|\det J(T(u))|$ quantifies the relative change of volume of a small neighborhood du around u.



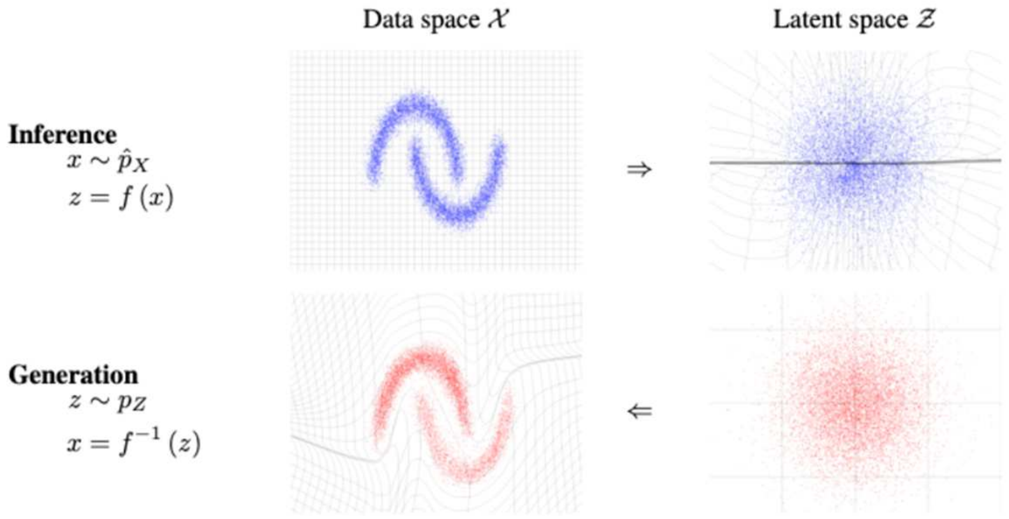
$f_1(\mathbf{z}_0)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$f_i(\mathbf{z}_{i-1})$
$\mathbf{z}_{i-1}$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_i$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html



Papamakarios et. al. (2019)

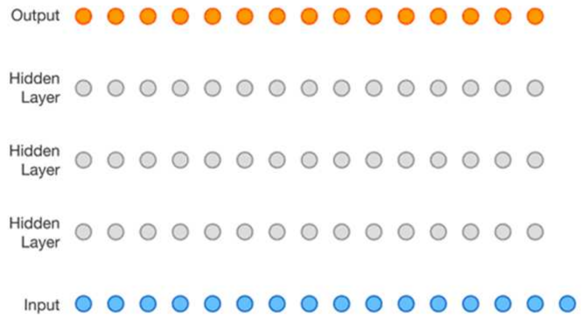
The intuition behind the normalizing flows



Dinh et. al. (2017) [1]

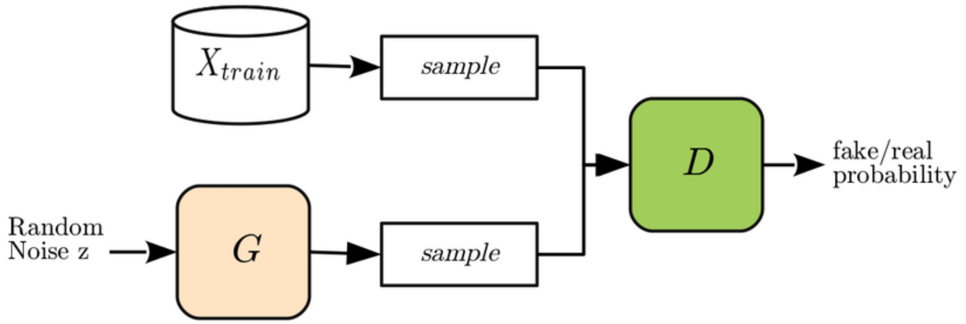
Familiy of generative models

Autoregressiv models



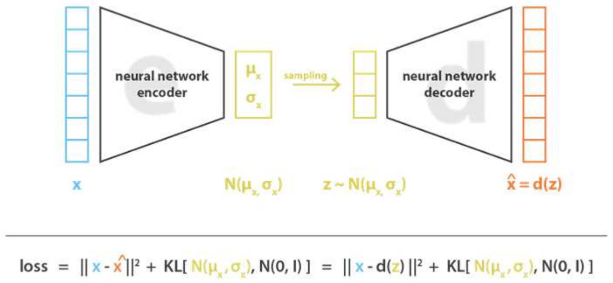
Source: https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

Generative Adversarial Network (GAN)



Source: https://www.researchgate.net/figure/Generative-Adversarial-Network-GAN_fig1_317061929

Variational Auto Encoders (VAE)

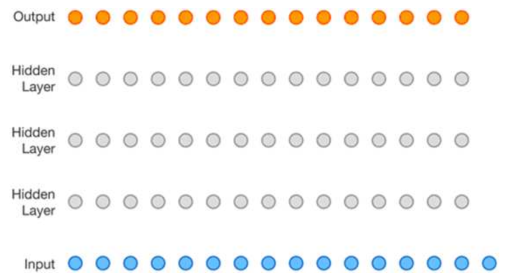

neural network encoder
μ_x
σ_x
sampling
neural network decoder
x
$N(\mu_x, \sigma_x)$
$z \sim N(\mu_x, \sigma_x)$
$\hat{x} = d(z)$
loss $= \| x - \hat{x} \|^2 + \mathrm{KL}[N(\mu_x, \sigma_x), N(0, I)] = \| x - d(z) \|^2 + \mathrm{KL}[N(\mu_x, \sigma_x), N(0, I)]$

Source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

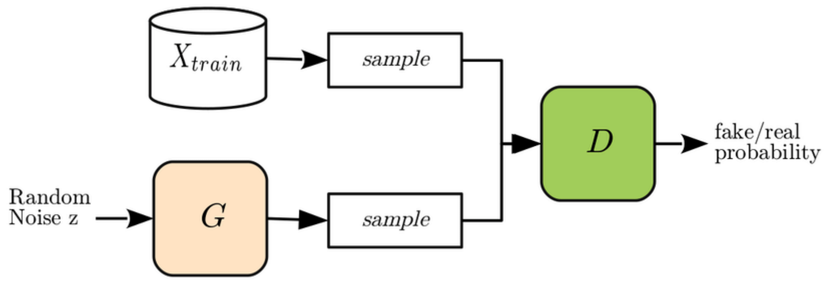
Merits of Normalizing Flows

Exact latent-variable inference and log-likelihood evaluation

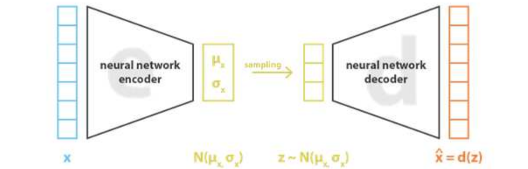
Autoregressiv models



Generative Adversarial Network



Variational Auto Encoders

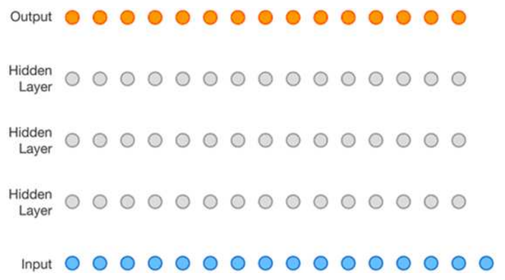


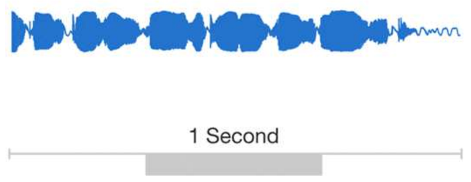
$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

Merits of Normalizing Flows

Efficient inference and efficient synthesis

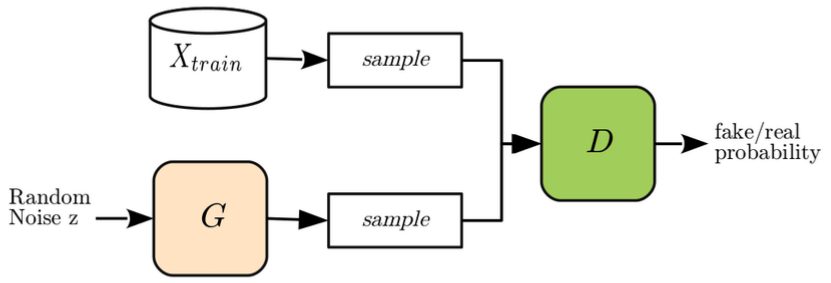
Autoregressiv models



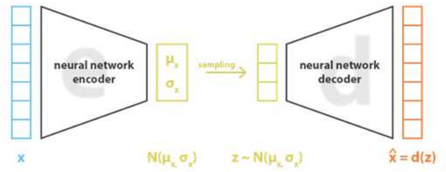


Source: https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

Generative Adversarial Network



Variational Auto Encoders

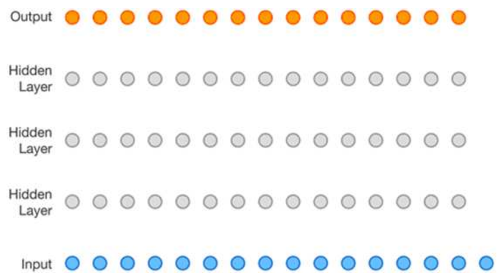


$$\text{loss} = \| x - \hat{x} \|^2 + \mathrm{KL}[N(\mu_x, \sigma_x), N(0, I)] = \| x - d(z) \|^2 + \mathrm{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

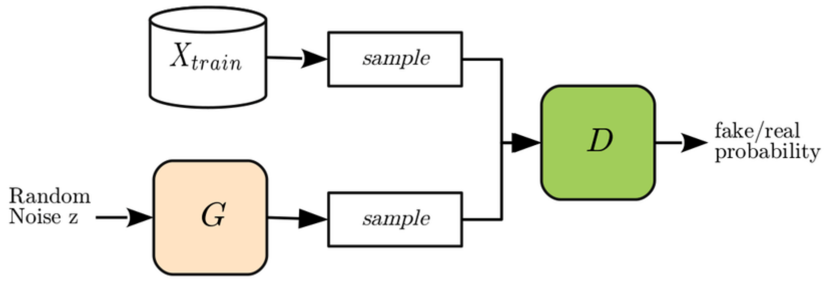
Merits of Normalizing Flows

Useful latent space for downstream tasks

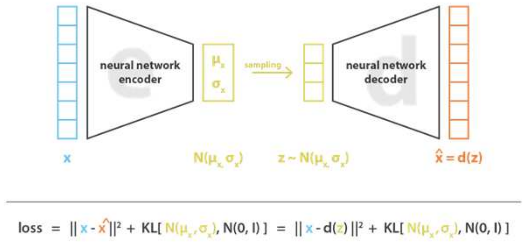
Autoregressiv models



Generative Adversarial Network



Variational Auto Encoders


neural network encoder
μ_x
σ_x
sampling
neural network decoder
x
$N(\mu_x, \sigma_x)$
$z \sim N(\mu_x, \sigma_x)$
$\hat{x} = d(z)$
loss $= \| x - \hat{x} \|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \| x - d(z) \|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$

Definition

General framework for constructing flexible probability distribution over continuse random variable

$$\boldsymbol{x} = T(\boldsymbol{u}), \text{where } \boldsymbol{u} \sim p_u(\boldsymbol{u})$$

- $\boldsymbol{x}\ - D$ dimensional real vector
- $\boldsymbol{u} - D$ dimensional real vector
- $p_u(\boldsymbol{u}) -$ base distribution (eg. Normal)
- $T\ \ -$ invertible transformation where both T and $T^{-1}$are differentiable (Diffeomorphishm)

Definition

- Density of x is well-defined
- Obtainable by a change of variables.

$$u \sim p_u(u)$$
$$x \sim p_x(x)$$
$$x = T(u)$$
$$p_x(x)dx = p_u(u)du$$
$$p_x(x) = p_u(u) \left|\frac{\partial u}{\partial x}\right|$$
$$p_x(x) = p_u(u)\left|\frac{\partial}{\partial x} T^{-1}(x)\right|$$
$$p_x(\boldsymbol{x}) = p_u(T^{-1}(\boldsymbol{x}))\left|\det J_{T^{-1}}(\boldsymbol{x})\right|$$

Implementing transformation in practice

T is implemented as a neural network taking $p_u(u)$ to be simple density such as a multivariate normal.

It is common to chain toogether multiple transformations T

Important properites of normalizing flows

- Differomorphic functions are composable – Given two such transformations T_1 and T_2 their composition is also invertible and differentiable
- As a consequence we can build complex transformations by composing multiple instances of simpler transformations

$$(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1}$$
$$\det J_{T_2 \circ T_1}(\mathbf{u}) = \det J_{T_2}(T_1(\mathbf{u})) \cdot \det J_{T_1}(\mathbf{u}).$$

The functionality of normalizing flows

- Sampling from the model
  - $x = T(u), \qquad \text{where } u \sim p_u(u)$
- Inference with the model
  - $u = T^{-1}(x)$
- Evaluating density
  - $p_x(x) = p_u(T^{-1}(x)) * |\det J_{T^{-1}}(x)|$

Different computational requirements. Application should dictate which need to be implemented efficiently.

The expressive power of flow based models

- For several autoregressive flows the universality property has been proven.
- Universality means that the flow can learn any target density to any required precision given sufficient capacity and data.

Expressive power of flow based models

- Suppose that $p_x(x) > 0$ for all $x \in R_D$
- Suppose all conditional probabilities $\Pr(x'i \leq xi | x < i)$ with $x'i$ being the random variable this probability refers to are differentiable to $(xi, x < i)$

$$p_{\mathrm{x}}(\mathbf{x}) = \prod_{i=1}^{D} p_{\mathrm{x}}(\mathrm{x}_i \mid \mathbf{x}_{<i}).$$

$$\mathrm{z}_i = F_i(\mathrm{x}_i, \mathbf{x}_{<i}) = \int_{-\infty}^{\mathrm{x}_i} p_{\mathrm{x}}(\mathrm{x}'_i \mid \mathbf{x}_{<i})\, d\mathrm{x}'_i = \Pr(\mathrm{x}'_i \leq \mathrm{x}_i \mid \mathbf{x}_{<i}).$$

$$\det J_F(\mathbf{x}) = \prod_{i=1}^{D} \frac{\partial F_i}{\partial \mathrm{x}_i} = \prod_{i=1}^{D} p_{\mathrm{x}}(\mathrm{x}_i \mid \mathbf{x}_{<i}) = p_{\mathrm{x}}(\mathbf{x}) > 0.$$

$$p_{\mathrm{z}}(\mathbf{z}) = p_{\mathrm{x}}(\mathbf{x})\ \left|\det J_F(\mathbf{x})\right|^{-1} = 1,$$

How to fit the model

- Forward KLD:
  - We have samples from the target distribution (or can generate them), but not necessarily know the target density.
  - Computing KLD on base distribution (require computing T^{-1})
- Backward KLD:
  - We can evaluate the target density but not necessarily sample from it.
  - We can minimize loss even if we can only evaluate target density up to a multiplicative normalizing constant C
  - Computing KLD on target distribution (require computing T)

How to fit the model

- Flow-based model is - $p_x(\boldsymbol{x};\ \boldsymbol{\theta})$
- Target distribution is – $p_x^*(\boldsymbol{x})$
- Models parameters - $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \boldsymbol{\psi}\}$
- Parameters of $T\ -\ \boldsymbol{\phi}$
- Parameters of $p_u(\boldsymbol{u})\ -\ \boldsymbol{\psi}$

How to fit the model - forward KLD

You have samples from the target distribution (or can generate them), but not necessarily know the target density.

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= D_{\mathrm{KL}}\left[\, p^*_{\mathrm{x}}(\mathbf{x}) \,\|\, p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta}) \,\right] \\ &= -\mathbb{E}_{p^*_{\mathrm{x}}(\mathbf{x})}\left[\, \log p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta}) \,\right] + \text{const.} \\ &= -\mathbb{E}_{p^*_{\mathrm{x}}(\mathbf{x})}\left[\, \log p_{\mathrm{u}}\left(T^{-1}(\mathbf{x};\boldsymbol{\phi});\boldsymbol{\psi}\right) + \log\left|\det J_{T^{-1}}(\mathbf{x};\boldsymbol{\phi})\right| \,\right] + \text{const.}\end{aligned}$$

$$\mathcal{L}(\boldsymbol{\theta}) \approx -\frac{1}{N}\sum_{n=1}^{N} \log p_{\mathrm{u}}(T^{-1}(\mathbf{x}_n;\boldsymbol{\phi});\boldsymbol{\psi}) + \log\left|\det J_{T^{-1}}(\mathbf{x}_n;\boldsymbol{\phi})\right| + \text{const.} \qquad (13)$$

Minimizing the above Monte Carlo approximation of the KL divergence is equivalent to fitting the flow-based model to the samples $\{\mathbf{x}_n\}_{n=1}^{N}$ by maximum likelihood estimation.

How to fit the model - reverse KLD

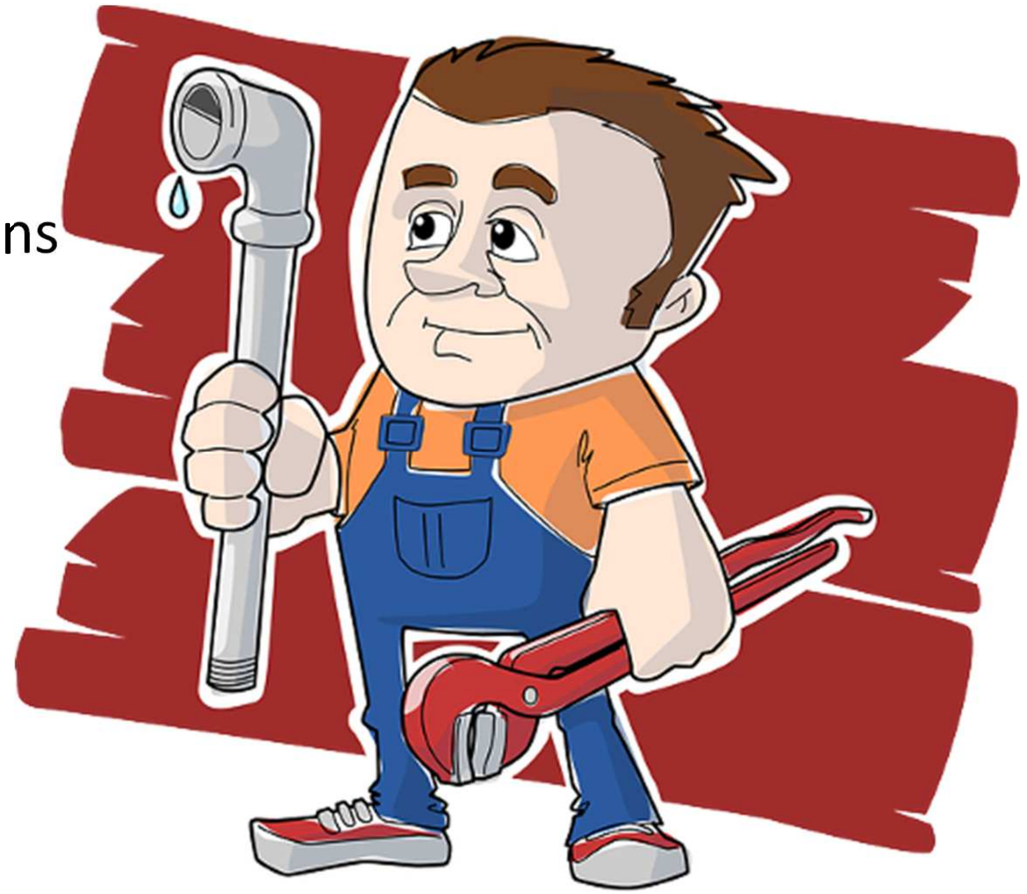
we have the ability to evaluate the target density but not necessarily sample from it. In fact, we can minimize L(θ) even if we can only evaluate target density up to a multiplicative normalizing constant C, since in that case log C will be an additive constant in the above expression for L(θ). EXAMPLES

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= D_{\mathrm{KL}}\left[\, p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta}) \,\|\, p^{*}_{\mathrm{x}}(\mathbf{x}) \,\right] \\
&= \mathbb{E}_{p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta})}\left[\, \log p_{\mathrm{x}}(\mathbf{x};\boldsymbol{\theta}) - \log p^{*}_{\mathrm{x}}(\mathbf{x}) \,\right] \\
&= \mathbb{E}_{p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})}\left[\, \log p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi}) - \log\left|\det J_{T}(\mathbf{u};\boldsymbol{\phi})\right| - \log p^{*}_{\mathrm{x}}(T(\mathbf{u};\boldsymbol{\phi})) \,\right].
\end{aligned}
$$

$$
\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})}\left[\, \log p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi}) - \log\left|\det J_{T}(\mathbf{u};\boldsymbol{\phi})\right| - \log \widetilde{p}_{\mathrm{x}}(T(\mathbf{u};\boldsymbol{\phi})) \,\right] + \mathrm{const.}
$$

Constructing Flows – finite composition

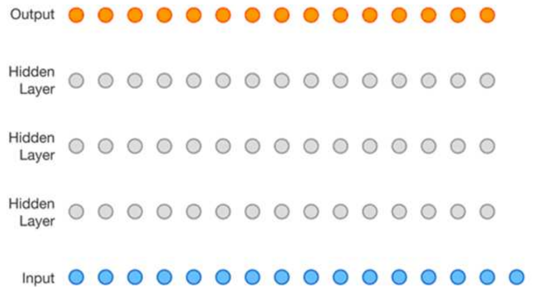
- Composition of transformations
- Efficiently tractable Jacobian



Autoregressive flow

Definition

- Transformation:
  - $z_i' = \tau(z_i; h_i)$ and $\mathrm{z_i} = \tau^{-1}(z_i'; h_i)$
  - Strictly monotonic function of z_i (due to that invertible)
  - Parametrized by conditioning h_i
- Conditioning :
  - $h_i = c_i(z_{<i})$
  - Determines the parameters of transformation
  - **Does not need to be a bijection**
  - i_{th} conditioner can depend only $z_{i<}$
- Consequences:
  - autoregressive flows are universal approximator



Autoregressive flows
Complexity

- Jacobian determiniet computation $O(D)$ - Lower triangle

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial \mathrm{z}_1}(\mathrm{z}_1; \boldsymbol{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial \tau}{\partial \mathrm{z}_D}(\mathrm{z}_D; \boldsymbol{h}_D) \end{bmatrix}.$$

$$\log \left|\det J_{f_\phi}(\mathbf{z})\right| = \log \left| \prod_{i=1}^{D} \frac{\partial \tau}{\partial \mathrm{z}_i}(\mathrm{z}_i; \boldsymbol{h}_i) \right| = \sum_{i=1}^{D} \log \left| \frac{\partial \tau}{\partial \mathrm{z}_i}(\mathrm{z}_i; \boldsymbol{h}_i) \right|.$$

- Forward pass $T(z)$ easily parallelizable though fast.
- Inverse $T^{-1}(x)$ slow

Autoregressive flows

Implementing transformation as an affine neural transform

- Shift and scale transformation:

  $$\tau(z_i; h_i) = \alpha_i z_i + \beta_i, \qquad h_i = \{\alpha_i, \beta_i\}$$

- Pros
  - Simplicity
  - Fast to compute Jacobian determinient $O(D)$

    $$\log \left| detJ_{\tau_{h_i}}(z_i) \right| = \sum_{i=1}^{D} \log |\alpha_i|$$

  - Analytical tractability
- Cons
  - Expressivity is limited

Autoregressive flows

Implementing Transformer as affine neural transformer

Examples:

- NICE (Dinh et. al. 2015)
- Inverse Autoregressive Flow (Diedrik et. al. 2016)
- Masked Autoregressiv Flow (Papamakarios et al., 2017)
- Parallel Wavenet (Oord et. al. 2017)
- RealNVP (Dinh et al., 2017)
- GLOW (Kingma and Dhariwal, 2018)
- WAVE GLOW (Prenger et. al. 2018)

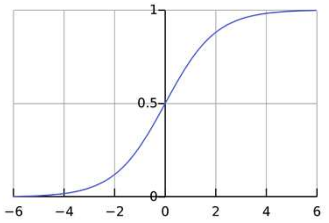
Results:

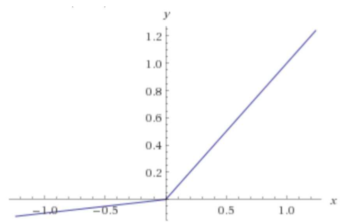
- Not state of the art but can achieve “good enough” results for Real Time applications
- Cannot stand to the results achieved with traditional autoregressive models:
  - WaveGlow or ParallelWavenet vs Wavenet
  - Glow, RealNVP, IAF, MAF vs PixelRNN

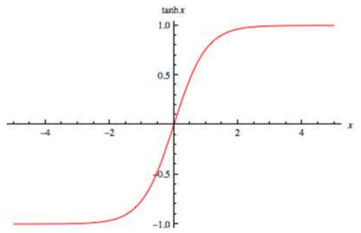
Autoregressive flows

Implementing Transformer as Non-affine Neural Transformer

- Constructed using a conic combination or composition of monotonically increasing activation functions such as:
    - logistic sigmoid
    - tanh
    - leaky ReLu
    - etc







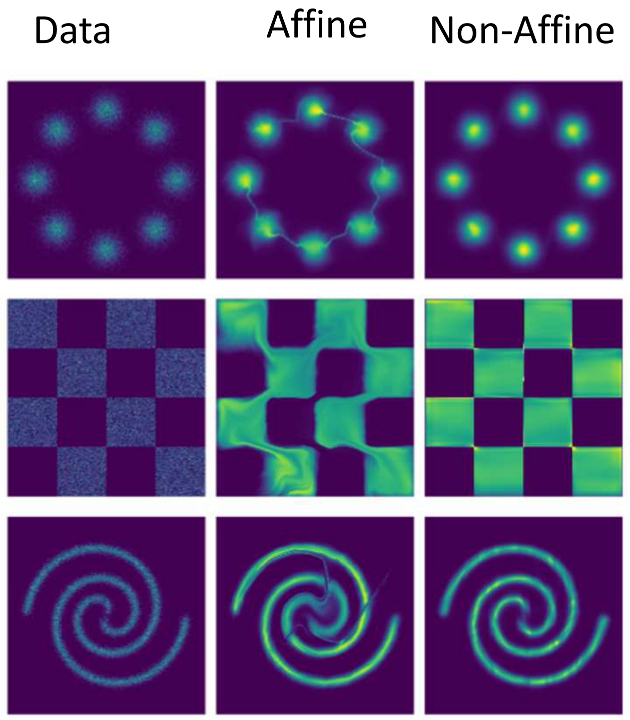
- Conic combination: $\tau(z) = \sum_{k=1}^{K} w_k \tau_k(z)\,,\ where\ w_k > 0$
- Composition: $\tau(z) = \tau_K \circ \cdots \circ \tau_1$

Autoregressive flows

Implementing Transformer as Non-affine Neural Transformer

- Pros
  - Can represent any monotonic function arbitrarily well, which follows directly from the universal-approximation capabilities of multi-layer perceptrons
- Cons:
  - In general they cannot be inverted analytically, and can be inverted only iteratively e.g. using bijection
- Examples
  - Neural Autoregressive Flows
  - Flow++

Modeling power on toy dataset



Source: B-NAF (De Cao et. al.)

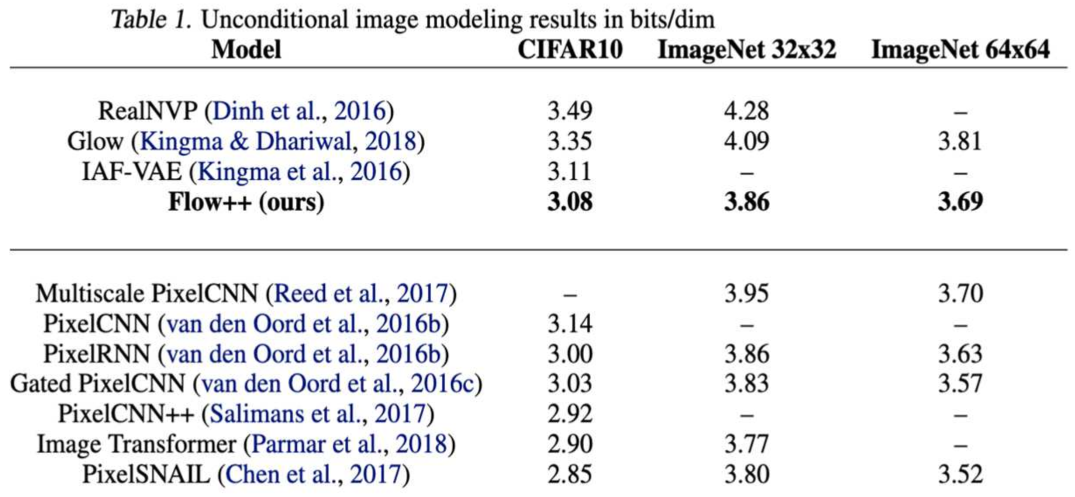


Table 1. Unconditional image modeling results in bits/dim

Model	**CIFAR10**	**ImageNet 32x32**	**ImageNet 64x64**
RealNVP (Dinh et al., 2016)	3.49	4.28	–
Glow (Kingma & Dhariwal, 2018)	3.35	4.09	3.81
IAF-VAE (Kingma et al., 2016)	3.11	–	–
Flow++ (ours)	**3.08**	**3.86**	**3.69**
Multiscale PixelCNN (Reed et al., 2017)	–	3.95	3.70
PixelCNN (van den Oord et al., 2016b)	3.14	–	–
PixelRNN (van den Oord et al., 2016b)	3.00	3.86	3.63
Gated PixelCNN (van den Oord et al., 2016c)	3.03	3.83	3.57
PixelCNN++ (Salimans et al., 2017)	2.92	–	–
Image Transformer (Parmar et al., 2018)	2.90	3.77	–
PixelSNAIL (Chen et al., 2017)	2.85	3.80	3.52

Autoregressive flows

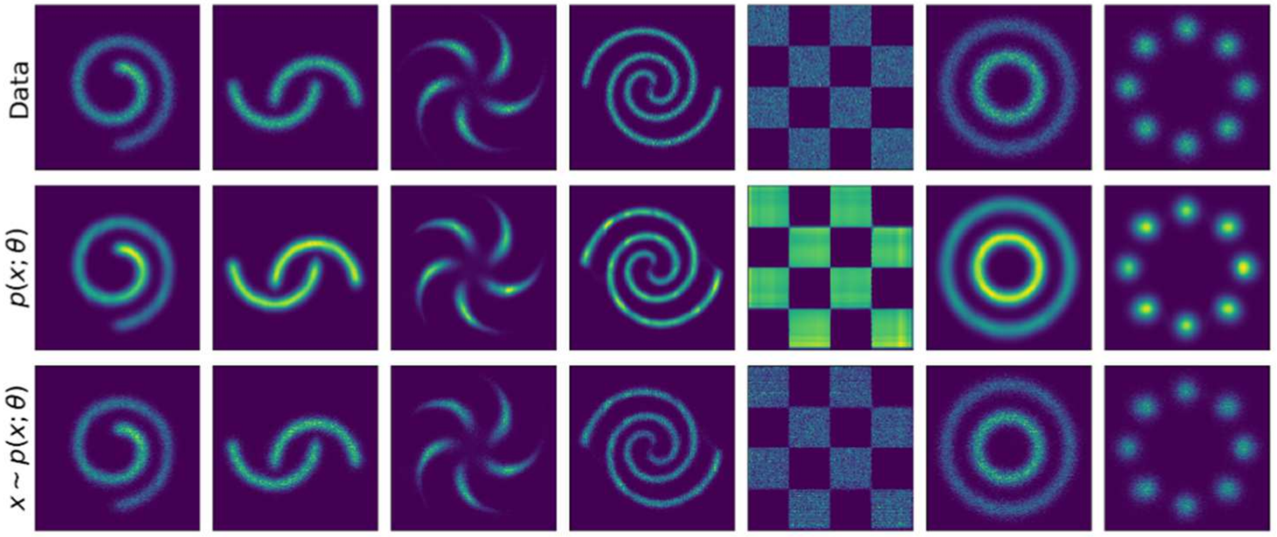
Implementing Transformer as Integration Transformer

- Constructed on observation that integral of some positive function is a monotonically increasing function

$$\tau(\mathrm{z}_i; \boldsymbol{h}_i) = \int_0^{\mathrm{z}_i} g(\mathrm{z}; \boldsymbol{\alpha}_i)\, d\mathrm{z} + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\boldsymbol{\alpha}_i, \beta_i\},$$

- Pros:
  - Arbitrarly flexible
- Cons:
  - Integral lacks analytical tractability. One possibility is to resort to a numerical approximation.
- Examples:
  - UMNN-MAF (Wehenkel and Louppe 2019)
  - Sum-of-Squares Polynomial Flow (Jaini and Yu 2019)

Modeling power on toy dataset

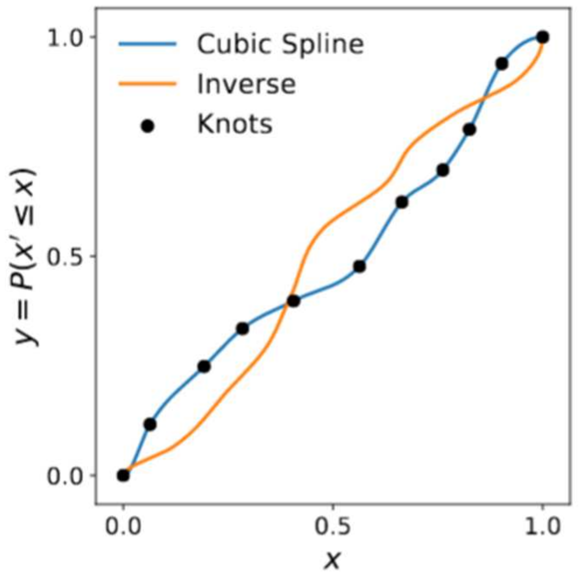


Source: UMNN-MAF (Wehenkel and Louppe 2019)

Autoregressive flows

Implementing Transformer as Neural Spline

- Implement transformer as monotonic spline with K semgents parametrized by neural network (for example using Steffens method).



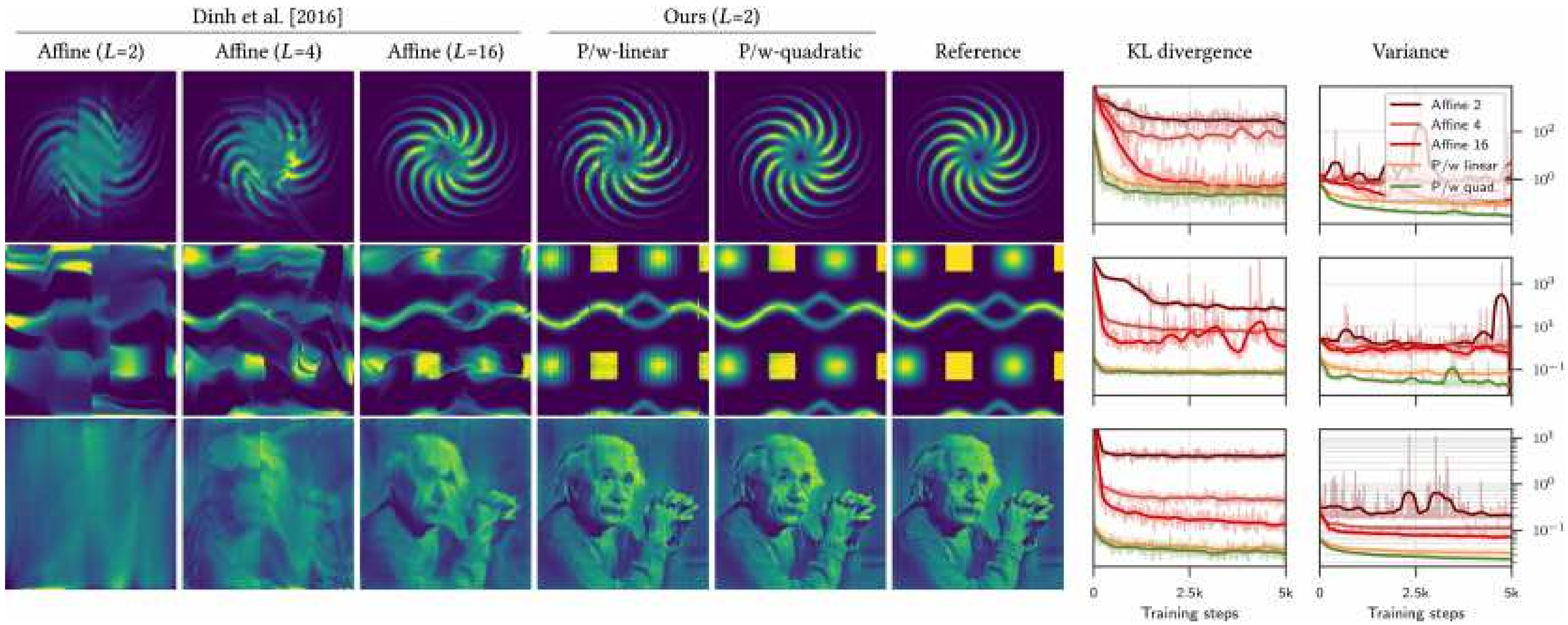
Source: Cubic Spline Flow (Durkan et. al. 2019)

Autoregressive flows

Implementing Transformer as Neural Spline

- Pros:
  - Arbitrarly flexible with increese of numer of segments
  - Deals with tradeoff between accuracy and computational cost of bijection search
  - Maintain exact analytical tractability
- Cons
  - Personally cannot find
- Expamples:
  - Neural Spline Flows (Durkan et. al. 2019)
  - Cubic Spline Flows (Durkan et. al. 2019)

Modeling on toy dataset



Source: Neural importnace sampling (Muller et. al. 2019)

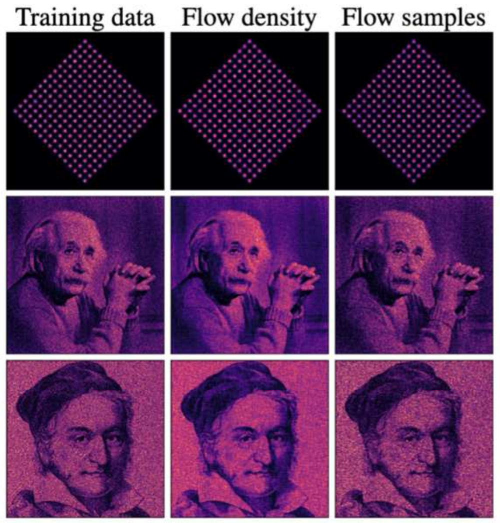


Figure 2: Qualitative results for two-dimensional synthetic datasets using cubic-spline flows with two coupling layers. Some previous flows struggle to model such fine details, as demonstrated by, e.g., Nash & Durkan [21].

Table 1: Test log likelihood (in nats) for UCI datasets and BSDS300; higher is better. Error bars correspond to two standard deviations (FFJORD do not report error bars). Apart from quadratic and cubic splines, all results are taken from existing literature. NAF† report error bars across five repeated runs rather than across the test set.

	MODEL	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
ONE-PASS FLOWS	FFJORD [8]	0.46	8.59	-14.92	-10.43	157.40
	QUADRATIC-SPLINE	0.65 ± 0.01	13.13 ± 0.02	-14.95 ± 0.02	-9.18 ± 0.43	157.49 ± 0.28
	CUBIC-SPLINE	0.65 ± 0.01	13.14 ± 0.02	-14.59 ± 0.02	-9.06 ± 0.44	157.24 ± 0.28
AUTO-REGRESSIVE FLOWS	MAF [23]	0.30 ± 0.01	10.08 ± 0.02	-17.39 ± 0.02	-11.68 ± 0.44	156.36 ± 0.28
	NAF† [13]	0.62 ± 0.01	11.96 ± 0.33	-15.09 ± 0.40	-8.86 ± 0.15	157.73 ± 0.04
	BLOCK-NAF [3]	0.61 ± 0.01	12.06 ± 0.09	-14.71 ± 0.38	-8.95 ± 0.07	157.36 ± 0.03
	TAN VARIOUS [22]	0.60 ± 0.01	12.06 ± 0.02	-13.78 ± 0.02	-11.01 ± 0.48	159.80 ± 0.07

Autoregressive flows

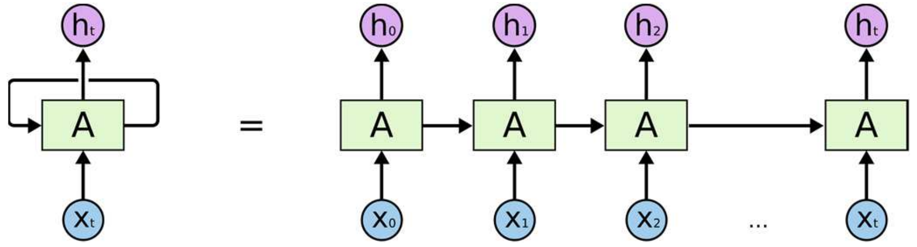
Implementing Conditioner

- Conditioner can be any function of $z_{i<}$
- Naive implementation would scale poorly with dimensionality D (on average $D/2$ forward passes)
- This problem can be overpassed through sharing parameters across the conditioners, or by combining the conditioners into a single net-work

Autoregressive flows

Implementing Conditioner – Recurent autoregressive flow

- Share parameters across conditioners using recurrent neural network (RNN).



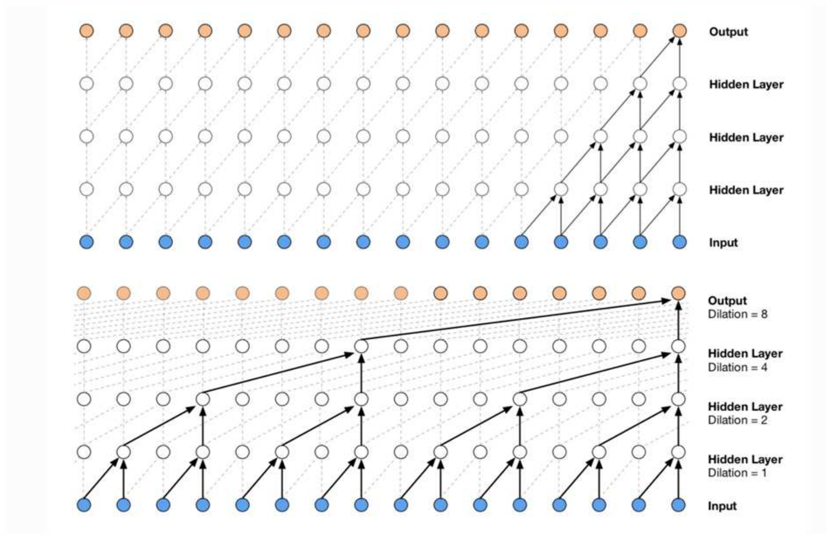
$$\boldsymbol{h}_i = c(\boldsymbol{s}_i) \quad \text{where} \quad \begin{aligned} \boldsymbol{s}_1 &= \text{initial state} \\ \boldsymbol{s}_i &= \text{RNN}(\mathrm{z}_{i-1}, \boldsymbol{s}_{i-1}) \text{ for } i > 1. \end{aligned}$$

- Pros:
  - Allow for sharing parameters saving memorry
- Cons:
  - Each state s_i must be computed sequentially even though each h_i can be computed independently and in parallel from $z_{i<}$.
  - Recurrent computation involves O(D)

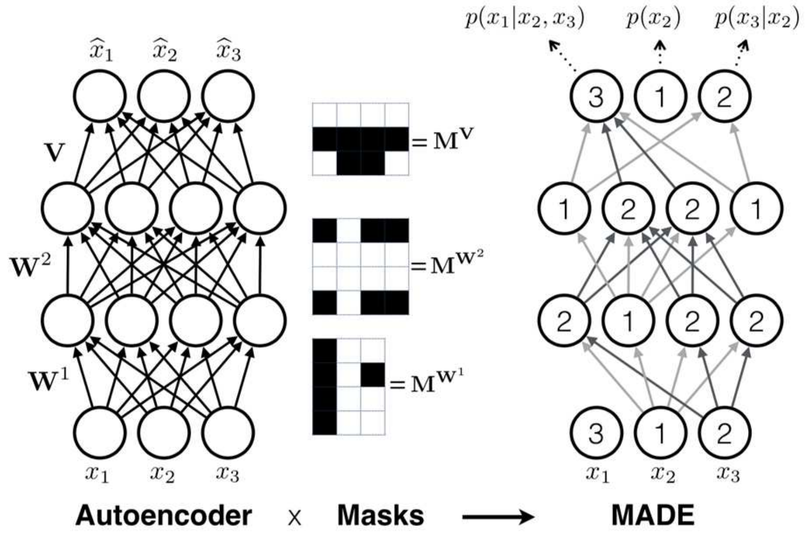
Autoregressive flows

Implementing Conditioner – masked autoregressive flow

- Feedforward neural network that takes z and outputs entire sequence $(h_1, \ \dots, h_D)$ in one pass
- Constructed through taking any Neural Network and masking any connections from $z_{\geq i}$ to h_i



Source: WaveNet (Oord et. al. 2016)



Source: MADE (Germain et al. 2015)

Autoregressive flows

Implementing Conditioner – masked autoregressive flow

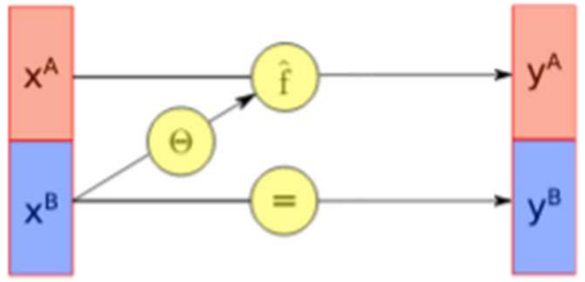
- Pros:
  - Efficient to evaluate
  - Universal aproximators given large enough conditioner and flexible enough transformer
- Cons:
  - Not efficient to invert
- Examples:
  - MAF
  - IAF
  - MintNet (Song et al. 2019)

Coupling flows
Implementing Conditioner – coupling layers

- Parameters $(h_1, \dots, h_d)$ are constants, i.e. not a function of z
- Parameters $(h_{d+1}, \dots, h_D)$ are functions of $z_{\leq d}$ only, i.e. they don't depend on $z_{>d}$.
- Coupling layers and fully autoregressive flows are two extremes on a spectrum of possible implementations

$$J_{f_\phi} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A} & \mathbf{D} \end{bmatrix}$$

$$\begin{aligned} \mathbf{z}'_{\leq d} &= \mathbf{z}_{\leq d} \\ (\boldsymbol{h}_{d+1}, \dots, \boldsymbol{h}_D) &= \mathrm{NN}(\mathbf{z}_{\leq d}) \\ \mathbf{z}'_i &= \tau(\mathbf{z}_i; \boldsymbol{h}_i) \text{ for } i > d. \end{aligned}$$



Coupling flows
Implementing Conditioner – coupling layers

- One of the most popular methods for implementing flow conditioners
- Coupling layers and fully autoregressive flows are two extremes on a spectrum of possible implementations
- Is not known if universal universal approximation cappabilities can be achieved with lower ammount of computations that with autorgressive flow

- Pros:
  - Faster computations for both T and T^{-1}
- Cons:
  - Comes at the cost of reduced epxressivity
  - Require permutations between layers

Other implementations of Normalizing Flows

- Linear Flows - find input ordering easier for modeling target distribution
- Residual Flows - all input variables to affect all output variables
- Continiouse flows – Instead of having finite compositions time is assumed to flow continiously for transformation
- **Conditional Flows** – we can use additional conditionings in conditioner network.

Autoregressive flows
Relation to Autoregressive models

We can think of autoregressive flows as subsuming and further extending autoregressive models for continuous variables.

- this view provides a framework for their composition, which opens up an avenue for enhancing their flexibility
- It separates the model architecture from the source of randomness, which gives us freedomin specifying the base distribution
- It allows us to compose autoregressive models with other types of flows, potentiallynon-autoregressive ones.

Linear flows

Definition

- Autoregressive flows depend on the order of the input variables.
- Target transformation may be easy to learn for some input orderings and hard to learn for others
- Permute the input variables between successive autoregressive layers.
- A permutation of the input variables is itself an easily invertible trans-formation, and its absolute Jacobian determinant is always 1
- A linear flow is essentially an invertible linear transformation of the form:

$$\mathbf{z}' = \mathbf{W}\mathbf{z},$$

Linear flows
Definition

- Pros:
  - Coupling layers without linear flows are limited
  - Allow to find input ordering easier for modeling target distribution
  - Special case - permutation – used with success in many applications such as RealNVP, Glow, Cubic Spline Flow
- Cons:
  - Straightforward implementation dooes not guarantee to be inversible
  - Finding Inverse of W and Jacobian Determiniet takes $O(D^3)$ - some approaches allow for respectively $O(D^2)$ and $O(D)$

Residual flows
Definition

Defined as: $$\mathbf{z}' = \mathbf{z} + g_{\phi}(\mathbf{z}),$$

Residual transformations are not always invertible, but can be made invertible if g_{φ} is constrained appropriately.

Residual flows

Contractive residual flows

- A residual transformation is guaranteed to be invertible if g_φ can be made contractive with respect to some distance function
- If 0 < $L < 1$ and $F\colon R^D \rightarrow R^D$

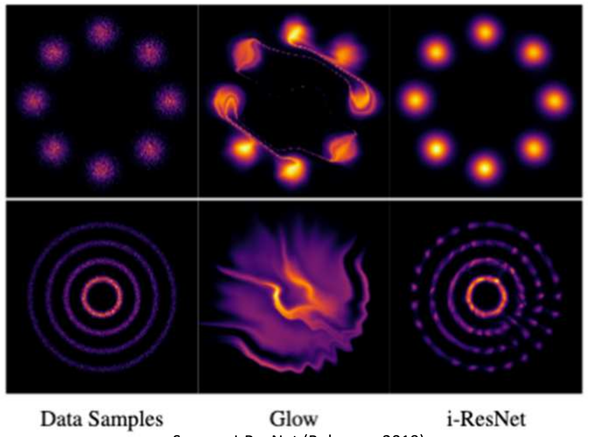
$$\delta(F(\mathbf{z}_A), F(\mathbf{z}_B)) \leq L\,\delta(\mathbf{z}_A, \mathbf{z}_B).$$

$$\mathbf{z}_{k+1} = \mathbf{z}' - g_{\boldsymbol{\phi}}(\mathbf{z}_k) \quad \text{for } k \geq 0.$$

Residual flows

Contractive residual flows

- Pros:
  - allows all input variables to affect all output variables
  - can be very flexible and have demonstrated good results in practice
- Cons:
  - Exact density estimation is computionally expensive
  - No general efficient procedurę for computing Jacobian
- Examples:
  - Invertible-ResNet



Source: I-ResNet (Behrman 2019)

Residual flows

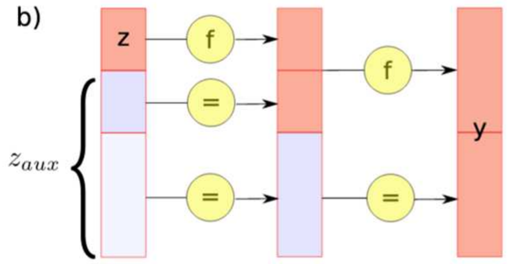
matrix determinant lemma

- Have $O(D)$ Jacobian determinants, and can be made invertible by suitably restricting their parameters
- No analytical way to compute invers
- It’s not clear how the flexibility of the flow can be increased other than by increasing the number of transformations
- Used to approximate posteriors for variational autoencoders and rarely as generative models in their own righ

- Examples:
    - Planar flow
    - Sylvester flow
    - Radial flow

Practical considerations

- Compose as many transformations as memory and computation will allow
- Use batch normalization between consecutive layers of flow:
  - Allow for training deeper models, through better gradient flow
  - Stabilize the training
  - With small mini-batches this can be noisy and negatively impact the training (Glow implements activation normalization instead)
- Use multi-scale architecture (skip-connections for flows)
  - Less costly though allow for deeper models
  - Help optimize through the whole depth of the flow



Constructing Flows – continiuse transformation

- Let z_t denote the flow's state at time t (or 'step' t, thinking in the discrete setting). Time t is assumed to run continuously from t_0 to t_1, such that $z_{t_0} = u$ and $z_{t_1} = x$.
- Continuous-time flow is constructed by parameterizing the time derivative of z_t with a neural networkg φ with parameters φ, yielding the following ordinary differential equation(ODE)

$$\frac{d\mathbf{z}_t}{dt} = g_\phi(t, \mathbf{z}_t).$$

Constructing Flows – continiuse transformation

- To compute the transformation $x = T(u)$, we need to run the dynamics forward in time by integrating

$$\mathbf{x} = \mathbf{z}_{t_1} = \mathbf{u} + \int_{t=t_0}^{t_1} g_{\boldsymbol{\phi}}(t, \mathbf{z}_t)\, dt.$$

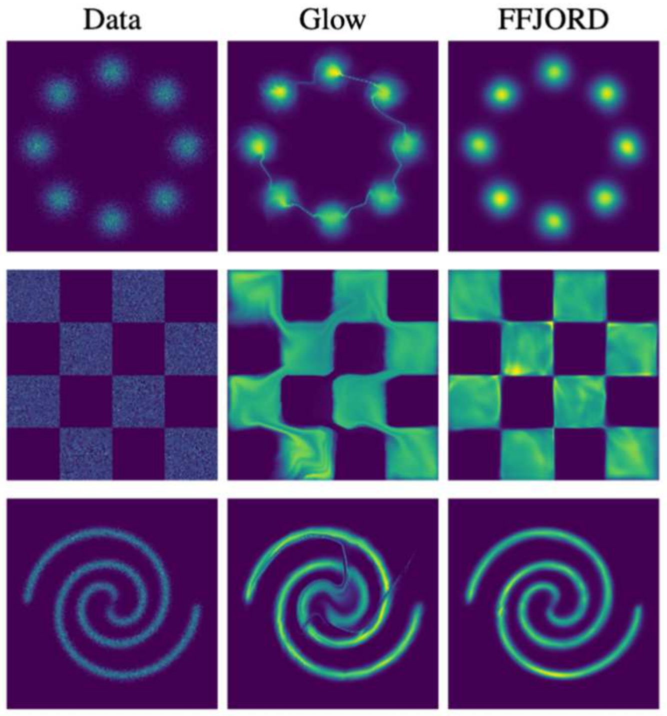
- Inverse transform is:

$$\mathbf{u} = \mathbf{z}_{t_0} = \mathbf{x} + \int_{t=t_1}^{t_0} g_{\boldsymbol{\phi}}(t, \mathbf{z}_t)\, dt = \mathbf{x} - \int_{t=t_0}^{t_1} g_{\boldsymbol{\phi}}(t, \mathbf{z}_t)\, dt,$$

- Optimization is done through numerical ODE solvers

Constructing Flows – continiuse transformation

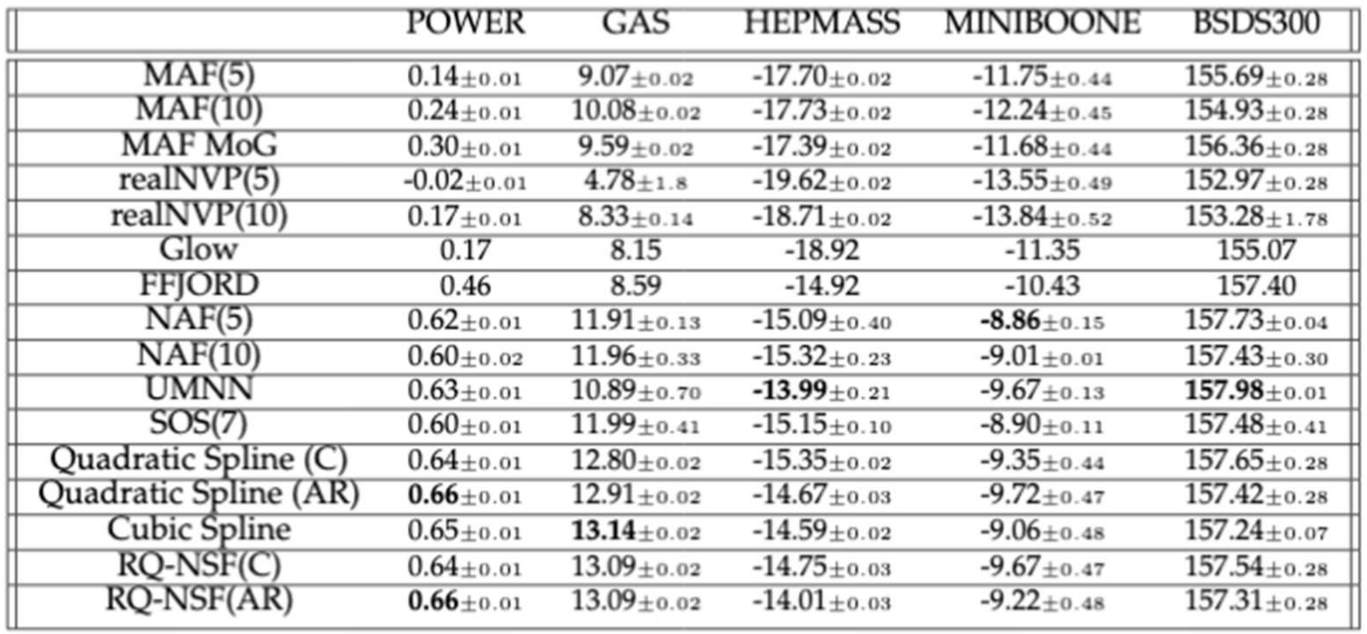
- FFJORD
- PointFlow



Sources: FFJORD (Grathwohl 2019)

Comparison of different methods

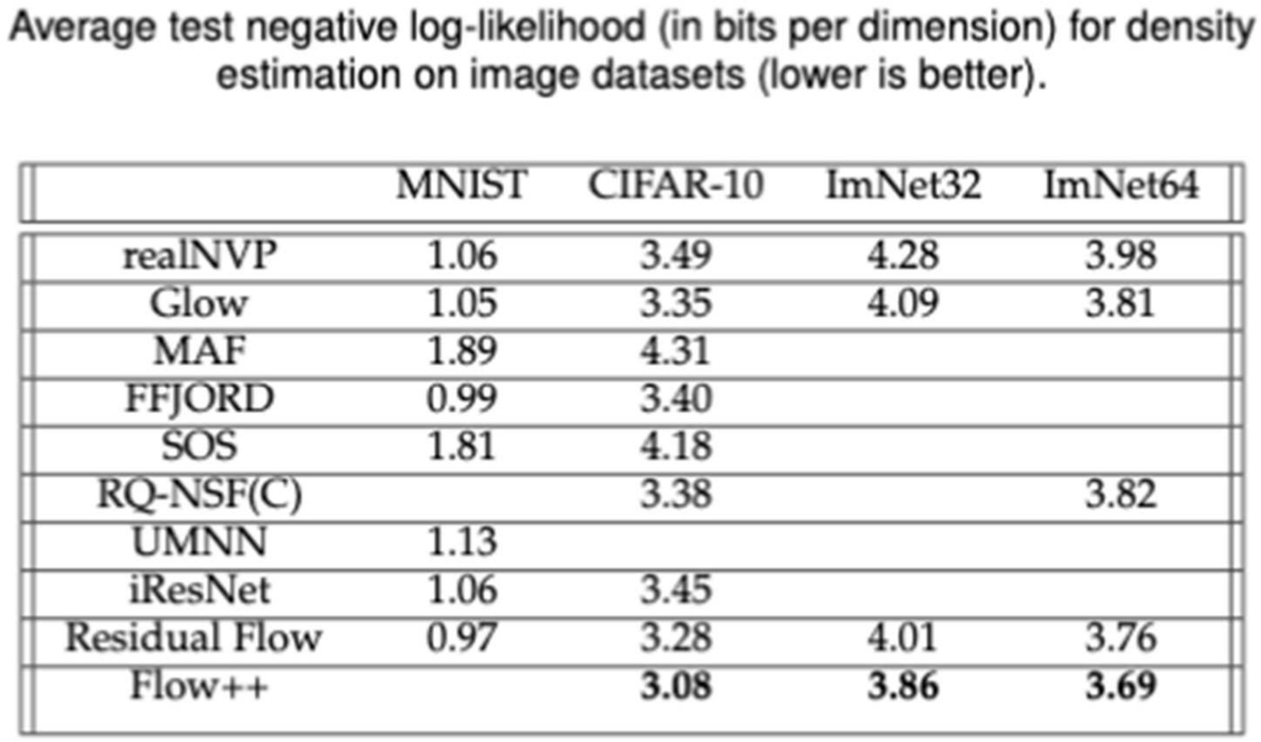
Average test log-likelihood (in nats) for density estimation on tabular datasets (higher the better). A number in parenthesis next to a flow indicates number of layers. MAF MoG is MAF with mixture of Gaussians as a base density.



	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
MAF(5)	0.14±0.01	9.07±0.02	-17.70±0.02	-11.75±0.44	155.69±0.28
MAF(10)	0.24±0.01	10.08±0.02	-17.73±0.02	-12.24±0.45	154.93±0.28
MAF MoG	0.30±0.01	9.59±0.02	-17.39±0.02	-11.68±0.44	156.36±0.28
realNVP(5)	-0.02±0.01	4.78±1.8	-19.62±0.02	-13.55±0.49	152.97±0.28
realNVP(10)	0.17±0.01	8.33±0.14	-18.71±0.02	-13.84±0.52	153.28±1.78
Glow	0.17	8.15	-18.92	-11.35	155.07
FFJORD	0.46	8.59	-14.92	-10.43	157.40
NAF(5)	0.62±0.01	11.91±0.13	-15.09±0.40	**-8.86**±0.15	157.73±0.04
NAF(10)	0.60±0.02	11.96±0.33	-15.32±0.23	-9.01±0.01	157.43±0.30
UMNN	0.63±0.01	10.89±0.70	**-13.99**±0.21	-9.67±0.13	**157.98**±0.01
SOS(7)	0.60±0.01	11.99±0.41	-15.15±0.10	-8.90±0.11	157.48±0.41
Quadratic Spline (C)	0.64±0.01	12.80±0.02	-15.35±0.02	-9.35±0.44	157.65±0.28
Quadratic Spline (AR)	**0.66**±0.01	12.91±0.02	-14.67±0.03	-9.72±0.47	157.42±0.28
Cubic Spline	0.65±0.01	**13.14**±0.02	-14.59±0.02	-9.06±0.48	157.24±0.07
RQ-NSF(C)	0.64±0.01	13.09±0.02	-14.75±0.03	-9.67±0.47	157.54±0.28
RQ-NSF(AR)	**0.66**±0.01	13.09±0.02	-14.01±0.03	-9.22±0.48	157.31±0.28

Source: Normalizing Flows: An Introduction and Reviewof Current Methods (Kobyzev et al. 2019)

Comparison of different methods



Average test negative log-likelihood (in bits per dimension) for density estimation on image datasets (lower is better).

	MNIST	CIFAR-10	ImNet32	ImNet64
realNVP	1.06	3.49	4.28	3.98
Glow	1.05	3.35	4.09	3.81
MAF	1.89	4.31		
FFJORD	0.99	3.40		
SOS	1.81	4.18		
RQ-NSF(C)		3.38		3.82
UMNN	1.13			
iResNet	1.06	3.45		
Residual Flow	0.97	3.28	4.01	3.76
Flow++		**3.08**	**3.86**	**3.69**

Source: Normalizing Flows: An Introduction and Reviewof Current Methods (Kobyzev et al. 2019)

Conclusions

- High level overview with intuition behind normalizing flows
- State of the art architectures and their characteristics

Most important references

- https://arxiv.org/abs/1908.09257 - **Normalizing Flows: An Introduction and Review of Current Methods**
- https://arxiv.org/abs/1910.13233 - **Neural Density Estimation and Likelihood-free Inference**
- https://arxiv.org/abs/1912.02762 - **Normalizing Flows for Probabilistic Modeling and Inference**