
2020
Swamps of Numerical Integration
Monte Carlo Heights


2020
Swamps of Numerical Integration
Monte Carlo Heights
Normalizing Flow

Change of Variables and Normalizing Flows

Cheng Soon Ong
Marc Peter Deisenroth

December 2020

Normalizing flows for density estimation

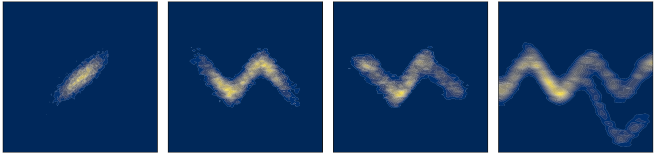


Figure: Generated with PyMC3 (Salvatier et al., 2016)

Key idea
(Tabak & Turner, 2013; Rippel & Adams, 2013; Rezende & Mohamed, 2015)

Build complex distributions from simple distributions via a flow of successive (invertible) transformations

Normalizing flows for density estimation

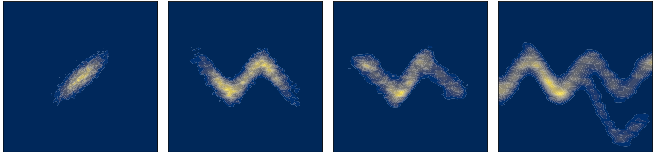


Figure: Generated with PyMC3 (Salvatier et al., 2016)

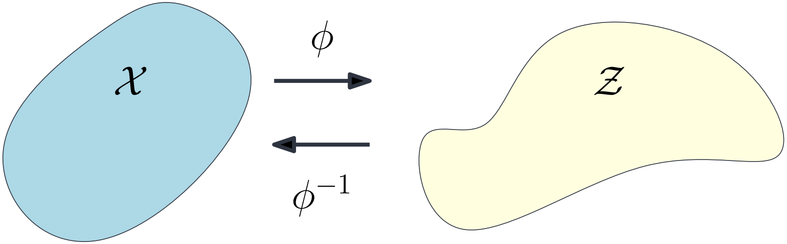
Key idea
(Tabak & Turner, 2013; Rippel & Adams, 2013; Rezende & Mohamed, 2015)

Build complex distributions from simple distributions via a flow of successive (invertible) transformations

Key ingredient: **Change-of-variables trick**

Change of Variables

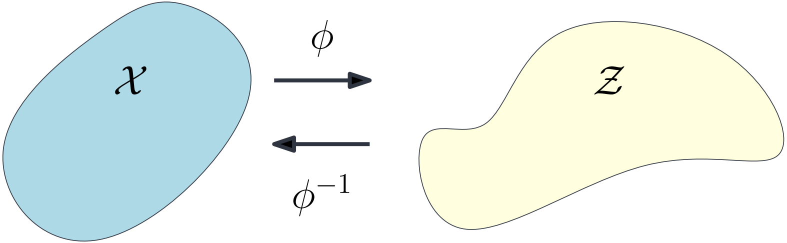
Change of variables: Key idea



Key idea

Transform random variable X into random variable Z using an invertible transformation ϕ, while keeping track of the change in distribution

Change of variables: Key idea

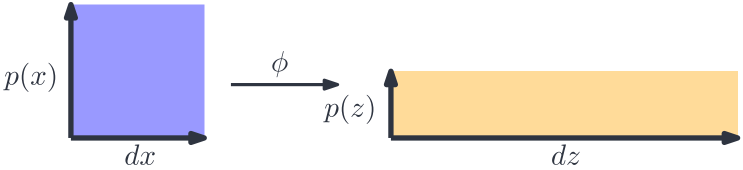


Key idea

Transform random variable X into random variable Z using an invertible transformation ϕ, while keeping track of the change in distribution

- Distribution p_X induces distribution p_Z via transformation ϕ
- Distribution p_Z induces distribution p_X via transformation ϕ^{-1}

Jacobian determinant



- Determinant of Jacobian

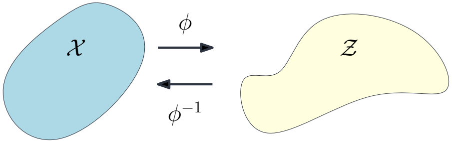
$$\left|\det\left(\frac{dz}{dx}\right)\right| = \left|\det\left(\frac{d\phi(x)}{dx}\right)\right|$$

tells us how much the domain dx is stretched to dz

How it works

- Constraint: volume preservation

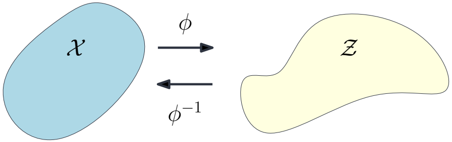
$$\int_{\mathcal{X}} p_X(\boldsymbol{x}) d\boldsymbol{x} = 1 = \int_{\mathcal{Z}} p_Z(\boldsymbol{z}) d\boldsymbol{z}$$



How it works

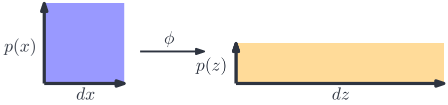
- Constraint: volume preservation

$$\int_{\mathcal{X}} p_X(\boldsymbol{x}) d\boldsymbol{x} = 1 = \int_{\mathcal{Z}} p_Z(\boldsymbol{z}) d\boldsymbol{z}$$

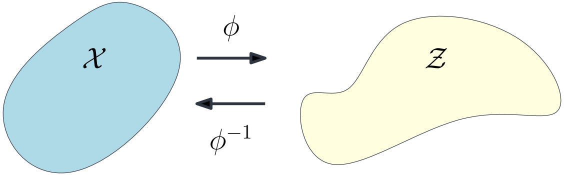


- Volume preservation: Rescale p_Z by the inverse of Jacobian determinant

$$p_Z(\boldsymbol{z}) = p_X(\boldsymbol{x}) \left| \det \left(\frac{d\phi(\boldsymbol{x})}{d\boldsymbol{x}} \right) \right|^{-1}$$



Considerations



$$p_Z(\boldsymbol{z}) = p_X(\boldsymbol{x}) \left| \det \left(\frac{d\phi(\boldsymbol{x})}{d\boldsymbol{x}} \right) \right|^{-1}$$

- Express target distribution p_Z in terms of known distribution p_X and the Jacobian determinant of an invertible mapping ϕ
- No need to invert ϕ explicitly
- Generate expressive distributions p_Z by simple p_X and flexible transformation ϕ

Applications

- Numerical integration (turn indefinite integrals into definite ones)
- Neural ODEs (E 2017, Chen et al., 2018)
- Learning in implicit generative models (e.g., GANs) and likelihood-free inference (e.g., ABC)
  (e.g., Mohamed & Lakshminarayanan, 2016; Sisson et al., 2007)
- **Normalizing flows** (Rezende & Mohamed, 2015)

Normalizing Flows

Normalizing flows for density estimation

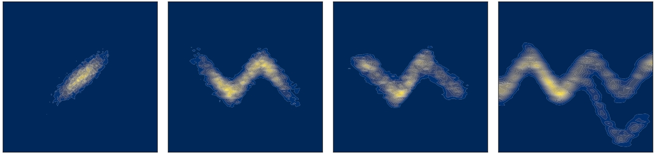
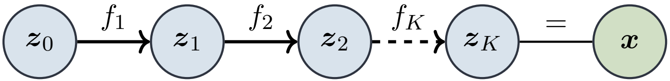


Figure: Generated with PyMC3 (Salvatier et al., 2016)

Key idea
(Tabak & Turner, 2013; Rippel & Adams, 2013; Rezende & Mohamed, 2015)

Build complex distributions from simple distributions via a flow of successive (invertible) transformations

How it works



- Random variable $\boldsymbol{z}_0 \sim p_0$
- Simple base distribution p_0, e.g. $p_0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

How it works

$$\begin{tikzcd} \boldsymbol{z}_0 \arrow[r, "f_1"] & \boldsymbol{z}_1 \arrow[r, "f_2"] & \boldsymbol{z}_2 \arrow[r, dashed, "f_K"] & \boldsymbol{z}_K \arrow[r, dash, "="] & \boldsymbol{x} \end{tikzcd}$$

- Random variable $\boldsymbol{z}_0 \sim p_0$
- Simple base distribution p_0, e.g. $p_0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
- Successive transformation of $\boldsymbol{z}_k$ via invertible transformations f_k:

$$\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1})$$

- Observed data $\boldsymbol{x} = \boldsymbol{z}_K$ at the end of the chain

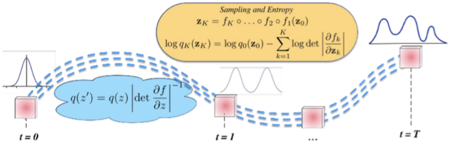
$$\boldsymbol{x} = \boldsymbol{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\boldsymbol{z}_0)$$

Marginal distribution

$$\boldsymbol{z}_0 \xrightarrow{f_1} \boldsymbol{z}_1 \xrightarrow{f_2} \boldsymbol{z}_2 \overset{f_K}{\dashrightarrow} \boldsymbol{z}_K = \boldsymbol{x}$$

- Repeated application of **change-of-variables** trick

$$p(\boldsymbol{x}) = p(\boldsymbol{z}_K) = p(\boldsymbol{z}_0) \prod_{k=1}^{K} \left| \det \frac{df_k(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right|^{-1}$$



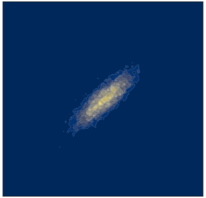
Distribution flows through a sequence of invertible transforms

Rezende and Mohamed, 2015

- Entropy

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_K) = \log p(\boldsymbol{z}_0) - \sum_{k=1}^{K} \log \left| \det \left(\frac{df_k(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right) \right|$$

Illustration with PyMC3 (Salvatier et al., 2016)

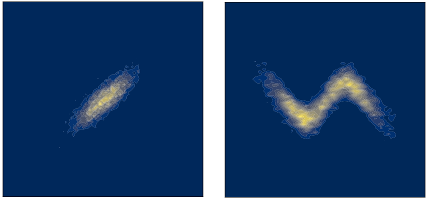


1 planar flow

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)

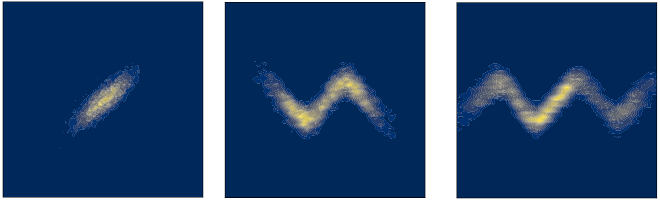


1 planar flow 2 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)

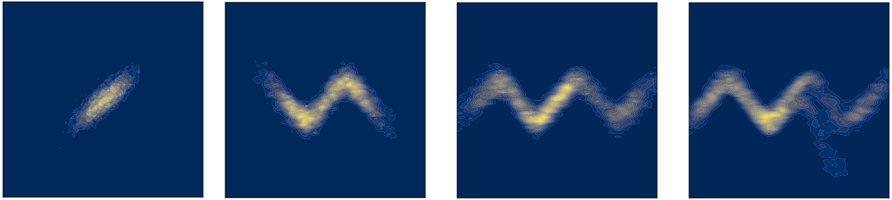


1 planar flow   2 planar flows   3 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)

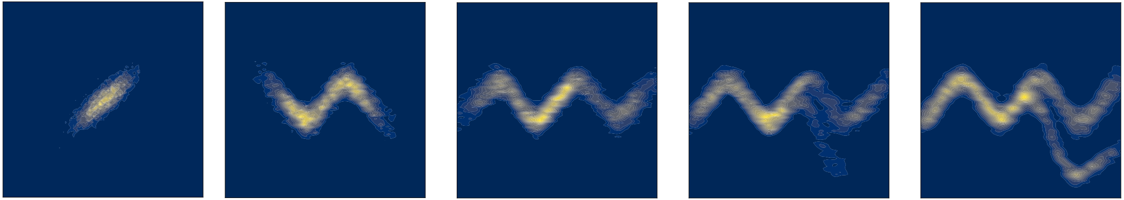


1 planar flow | 2 planar flows | 3 planar flows | 7 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)



1 planar flow | 2 planar flows | 3 planar flows | 7 planar flows | 12 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Computing expectations

$$\mathbb{E}_{p_X}[l(\boldsymbol{x})] = \mathbb{E}_{p_K}[l(\boldsymbol{z}_K)] = \mathbb{E}_{p_0}[l(f_K \circ \cdots \circ f_1(\boldsymbol{z}_0))]$$

Computing expectations

$$\mathbb{E}_{p_X}[l(\boldsymbol{x})] = \mathbb{E}_{p_K}[l(\boldsymbol{z}_K)] = \mathbb{E}_{p_0}[l(f_K \circ \cdots \circ f_1(\boldsymbol{z}_0))]$$

- Expectations w.r.t. p_K can be computed without explicitly knowing p_K or p_X
  - Sample $\boldsymbol{z}_0^{(s)} \sim p_0$
  - Push sample forward through sequence of deterministic transformations
    - Valid sample $\boldsymbol{x}^{(s)} \sim p_X(\boldsymbol{x})$
- Monte-Carlo estimation to get expected value

Computational considerations

- Compute log-determinant of Jacobian
- Cheap (linear) if Jacobian is (block-)diagonal or triangular

Computational considerations

- Compute log-determinant of Jacobian
- Cheap (linear) if Jacobian is (block-)diagonal or triangular
- Require partial derivatives

$$\frac{\partial z_k^{(d)}}{\partial z_{k-1}^{(>d)}} = 0 \quad \blacktriangleright\!\!\blacktriangleright\!\!\blacktriangleright \quad \frac{d\boldsymbol{z}_k}{d\boldsymbol{z}_{k-1}} = \begin{bmatrix} \frac{\partial z_k^{(1)}}{\partial z_{k-1}^{(1)}} & \mathbf{0} & \dots & \mathbf{0} \\ \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(1)}} & \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(2)}} & \dots & \vdots \\ \vdots & & \ddots & \mathbf{0} \\ \frac{\partial z_k^{(D)}}{\partial z_{k-1}^{(1)}} & \dots & \dots & \frac{\partial z_k^{(D)}}{\partial z_{k-1}^{(D)}} \end{bmatrix} \in \mathbb{R}^{D \times D}.$$

Autoregressive flows

- High-level idea:

$$z_k^{(d)} = \phi(z_{k-1}^{(\leq d)})$$

Autoregressive flows

- High-level idea:

$$z_k^{(d)} = \phi(z_{k-1}^{(\leq d)})$$

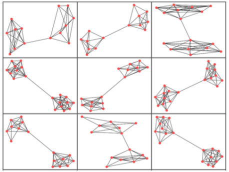
- NICE (Dinh et al., 2014)
- Inverse autoregressive flow (Kingma et al., 2016)
- Real NVP (Dinh et al., 2017)
- Masked autoregressive flow (Papamakarios et al., 2017)
- Glow (Kingma & Dhariwal, 2018)
- (Block) neural autoregressive flows, spline flows, ... (e.g., Huang et al., 2018; de Cao et al., 2019; Durkan et al., 2019)

Application areas

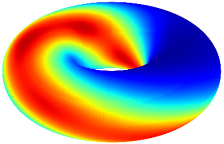
- Variational inference in deep generative models (e.g., Rezende & Mohamed, 2015)
- Graph neural networks (Liu et al., 2019)
- Parallel WaveNet (van den Oord et al., 2018)

Application areas

- Variational inference in deep generative models (e.g., Rezende & Mohamed, 2015)
- Graph neural networks (Liu et al., 2019)
- Parallel WaveNet (van den Oord et al., 2018)
- Continuous flows
  - Neural ODEs (e.g, E, 2017; Chen et al., 2018)
  - Flows on manifolds (e.g., Gemici et al., 2016; Rezende et al., 2020; Mathieu & Nickel, 2020)

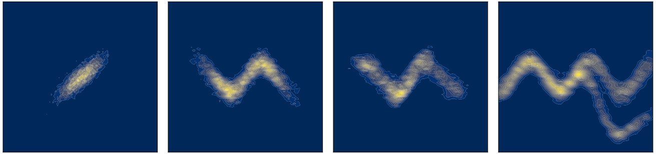


From Liu et al. (2019)



From Rezende et al. (2020)

Summary



- Normalizing flows provide a constructive way to generate rich distributions
- Key idea: Transform a simple distribution using a flow of successive (invertible) transformations
- Key ingredient: Change-of-variables trick
- Jacobians can be computed efficiently, if the transformations are defined appropriately
- Can be used as a generator and inference mechanism

References

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*.

De Cao, N., Aziz, W., and Titov, I. (2019). Block Neural Autoregressive Flow. In *Proceedings of the Conferene on Uncertainty in Artificial Intelligence*.

Dinh, L., Krueger, D., and Bengio, Y. (2014). NICE: Non-linear Independent Components Estimation. *arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density Estimation Using Real NVP. In *Proceedings of the International Conference on Learning Representation*.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural Spline Flows. In *Advances in Neural Information Processing Systems*.

E, W. (2017). A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11.

References (cont.)

Gemici, M. C., Rezende, D. J., and Mohamed, S. (2016). Normalizing Flows on Riemannian Manifolds. *arXiv:1611.02304*.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural Autoregressive Flows. In *Proceedings of the International Conference on Machine Learning*.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative Flow with Invertible 1×1 Convolutions. In *Advances in Neural Information Processing Systems*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Procesing Systems*.

Kobyzev, I., Prince, S., and Brubaker, M. (2020). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. (2019). Graph Normalizing Flows. In *Advances in Neural Information Processing Systems*.

References (cont.)

Mathieu, E. and Nickel, M. (2020). Riemannian Continuous Normalizing Flows. *arXiv:2006.10605*.

Mohamed, S. and Lakshminarayanan, B. (2016). Learning in Implicit Generative Models. *arXiv:1610.03483*.

Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2018). Parallel WaveNet: Fast High-Fidelity Speech Synthesis. In *Proceedings of the International Conference on Machine Learning*.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing Flows for Probabilistic Modeling and Inference. *arXiv:1912.02762*.

Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*.

Rezende, D. J. and Mohamed, S. (2015). Variational Inference with Normalizing Flows. In *Proceedings of the International Conference on Machine Learning*.

References (cont.)

Rezende, D. J., Papamakarios, G., Racanière, S., Albergo, M. S., Kanwar, G., Shanahan, P. E., and Cranmer, K. (2020). Normalizing Flows on Tori and Spheres. In *Proceedings of the International Conference on Machine Learning*.

Rippel, O. and Adams, R. P. (2013). High-Dimensional Probability Estimation with Deep Density Models. *arXiv:1302.5125*.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic Programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55.

Sisson, S. A., Fan, Y., and Tanaka, M. M. (2007). Sequential Monte Carlo Without Likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, 104(6):1760–1765.

Tabak, E. G. and Turner, C. V. (2013). A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.