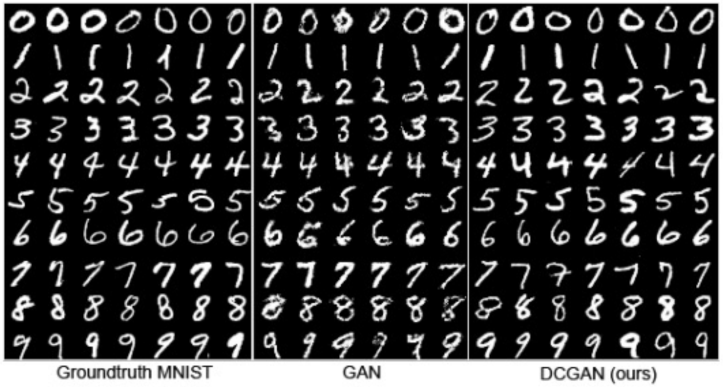
Invertible Normalizing Flows

ECE57000: Artificial Intelligence

David I. Inouye

GAN Limitation:
Cannot compute density values

- Evaluation of GANs is challenging
  - Explicit density models could use test log likelihood
  - "I think this looks better than that"
  - Inception-based scores

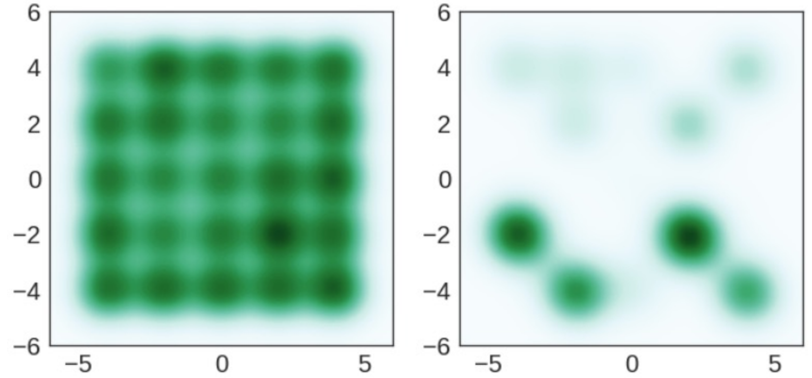


- Cannot use for classification or outlier detection
- Normalizing flows provide **exact** density values

Common problem with GANs: **Mode collapse** hinders diversity of samples

From: https://developers.google.com/machine-learning/gan/problems

- Normalizing flows do not suffer from mode collapse as MLE is used



(f) True Data (g) GAN

Metz, L., Poole, B., Pfau, D., & Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

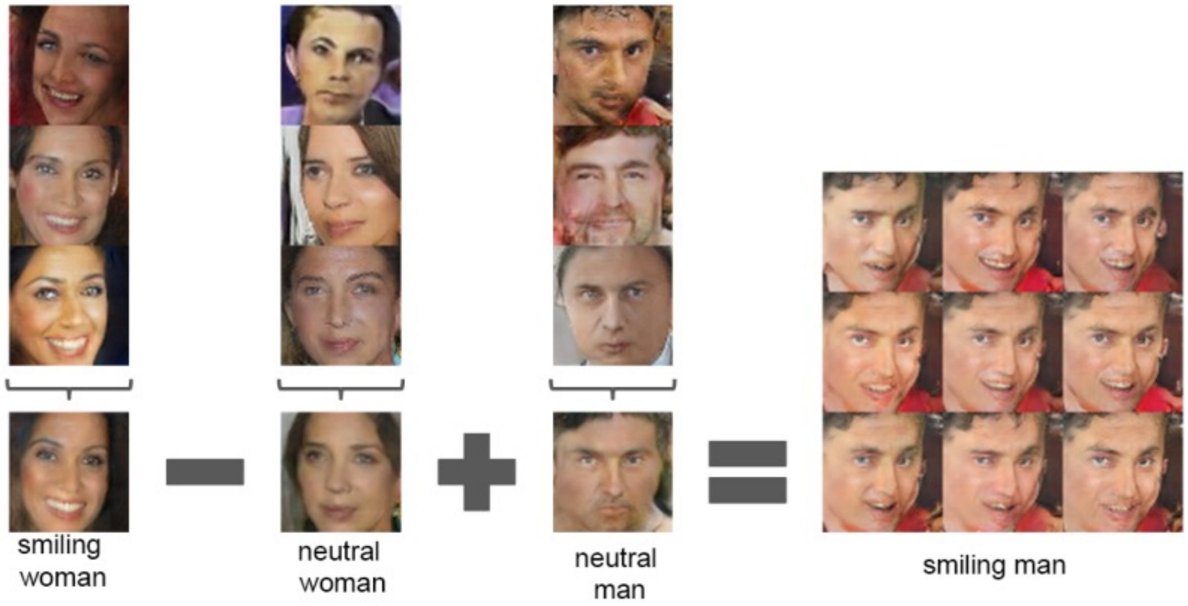
http://papers.nips.cc/paper/6923-veegan-reducing-mode-collapse-in-gans-using-implicit-variational-learning.pdf

https://software.intel.com/en-us/blogs/2017/08/21/mode-collapse-in-gans

GAN Limitation: Cannot go from observed to latent space, i.e. $x \to z$ not possible/easy

- Cannot manipulate an observed image in latent space
  - Cannot do the following, $x \to z,\ z' = z + 3,\ z' \to x'$
  - Rather, must start from fake image based on random z

All fake images->



Normalizing flows enable interpolation between **real images**

Figure 5: Linear interpolation in latent space between real images.

https://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf

Normalizing flows enable transformations of **real image** along various features

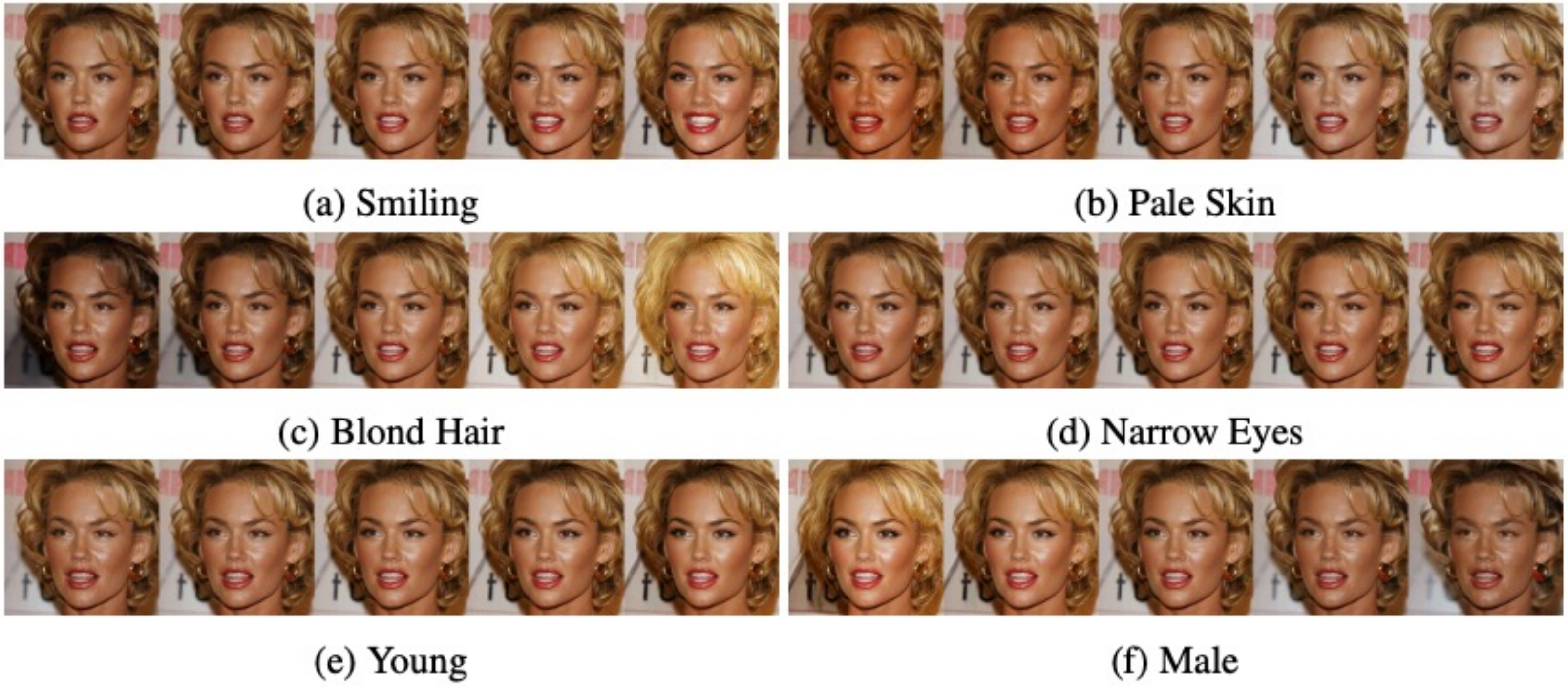
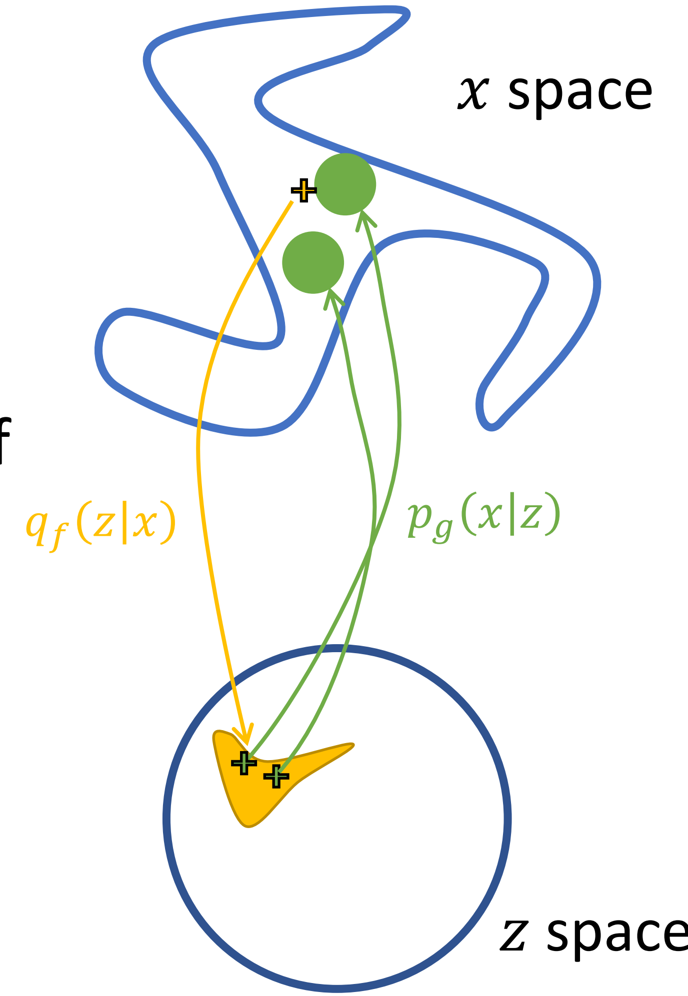


Figure 6: Manipulation of attributes of a face. Each row is made by interpolating the latent code of an image along a vector corresponding to the attribute, with the middle image being the original image

https://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf

Normalizing flows enable more powerful inference distributions $q_f(z|x)$ in VAEs

- The probabilistic encoder in VAEs requires 2 things:
  1. Ability to sample from $q_f(z|x)$ via reparameterization trick
  2. Ability to compute exact density of $q_f(z|x)$ for
  $KL\left(q_f(z|x), p_g(z)\right) = \mathbb{E}_{z \sim q_f(z|x)}\left[\log \frac{q_f(z|x)}{p_g(z)}\right]$
- Normalizing flows have these capabilities and thus significantly generalize the Gaussian $q_f(z|x)$



Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, *29*, 4743-4751.

Overview of Normalizing Flows

7

Motivation

Definition and comparison to GANs and VAEs

Objective function for flows

- Change of variables formula in 1D
- Generalization to higher dimensions via determinant of Jacobian
- Log likelihood of flows

Normalizing flow architectures

- Design requirements
- Autoregressive and inverse autoregressive
- RealNVP and Glow architecture ideas

<u>Normalizing flows</u> use **invertible** deep models for the generator which allow more capabilities

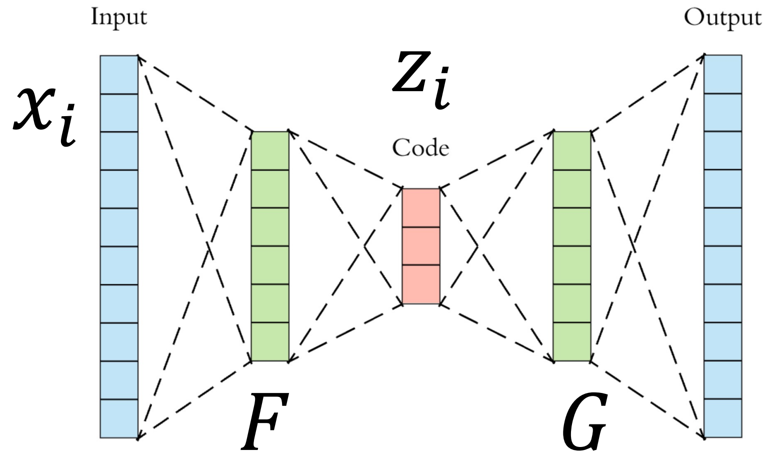
- Transforming between observed/input and latent space is easy
  - $x = G\ (z)$
  - $z = G^{-1}(x)$

- Simple sampling like GANs
  - $z \sim \text{SimpleDistribution}$
  - $x = G(z) \sim \hat{p}_g(x)$, which is estimated distribution

- **Exact density** is computable via change of variables
  - Standard maximum likelihood estimation can be used for training

Comparing VAEs and normalizing flows: Flows give zero reconstruction error

VAE

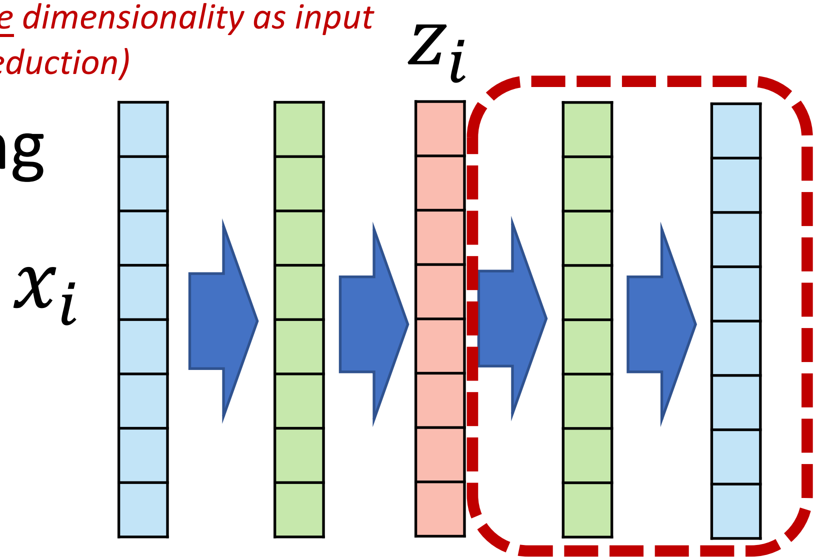


$\tilde{x}_i \sim p(x_i | G(z_i))$

$L(x_i, \tilde{x}_i)$

Latent code has <u>same</u> dimensionality as input (no dimensionality reduction)

Normalizing Flow



$F = G^{-1}$ $\quad G = F^{-1}$

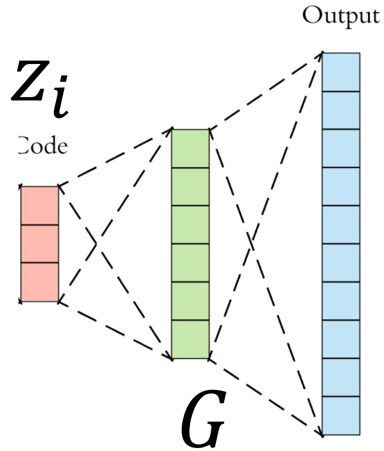
<u>*Implicit*</u> generator via $\mathrm{G} = F^{-1}$ (only need to train encoder F)

$$\tilde{x}_i = G(F(x_i)) = G(G^{-1}(x_i)) = x_i$$

$$\Rightarrow L(x_i, \tilde{x}_i) = L(x_i, x_i) = 0$$

Comparing GANs and normalizing flows: Normalizing flows can use MLE training

GAN

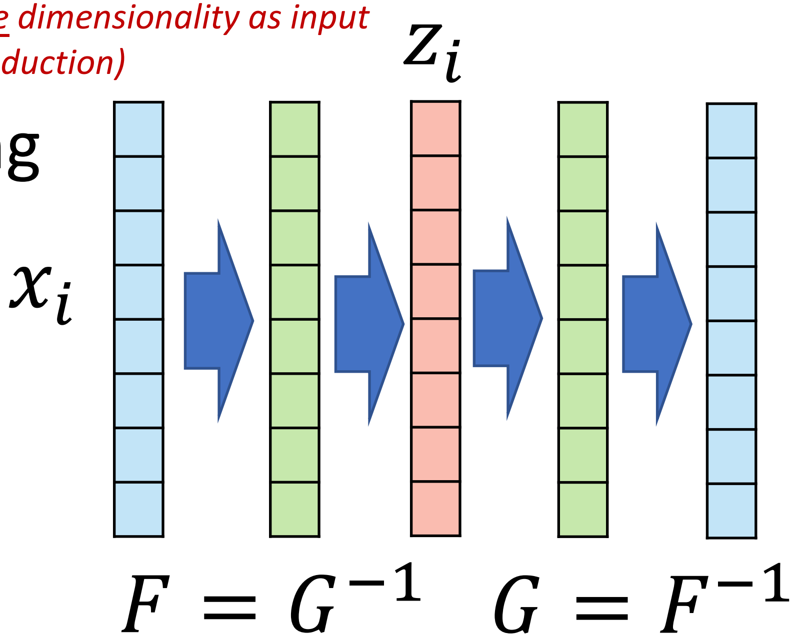


$\tilde{x}_i = G(z_i)$

Adversarial training to compare two sets of samples

Latent code has same dimensionality as input (no dimensionality reduction)

Normalizing Flow



$$\tilde{x}_i = G(z_i) = G\left(G^{-1}(x_i)\right) = x_i$$

MLE training since density function known

Back to maximum likelihood estimation (MLE): <u>How</u> can we compute the likelihood for normalizing flows?

- Suppose
  - $z \sim \text{Uniform}([0,1]), \text{i.e.}, p_z(z) = 1$ (latent space is uniform)
  - $G(\text{z}) = 2\text{z}$
  - Thus, $x = G(z) = 2z$.

- What is the density function of x, what is $p_x(x)$?

Change of variables formula gives p_x in terms of the p_z and the derivative of G^{-1}

- Key idea: Must conserve density **volume** (so that distribution sums to 1).
- $p_x(x)|dx| = p_z(z)|dz|$, this is like the preservation of volume/area/mass.
  - Intuition: We only have 1 unit of "dirt" to move around.
- Rearrange above equation to get formula

$$p_x(x) = \left|\frac{dz}{dx}\right| p_z(z) = \left|\frac{dG^{-1}(x)}{dx}\right| p_z\left(G^{-1}(x)\right)$$

Demo of change of variables

Derivation of change of variables using CDF function (Increasing)

- Assume $x = G(z)$, where $G(z)$ is an increasing function, i.e.,
$$z_1 \leq z_2 \Rightarrow G(z_1) \leq G(z_2)$$
$$z_1 \leq z_2 \Rightarrow G^{-1}(z_1) \leq G^{-1}(z_2)$$
- Remember CDF
$$F_x(a) = \Pr(x \leq a) = \int_{-\infty}^{a} p_x(t)dt$$
- Now $F_x(a) = \Pr(x \leq a)$
- $= \Pr(G(z) \leq a)$
- $= \Pr\left(G^{-1}(G(z)) \leq G^{-1}(a)\right)$
- $= \Pr\left(z \leq G^{-1}(a)\right)$
- $= F_z\left(G^{-1}(a)\right)$

Derivation of change of variables using CDF function (Increasing)

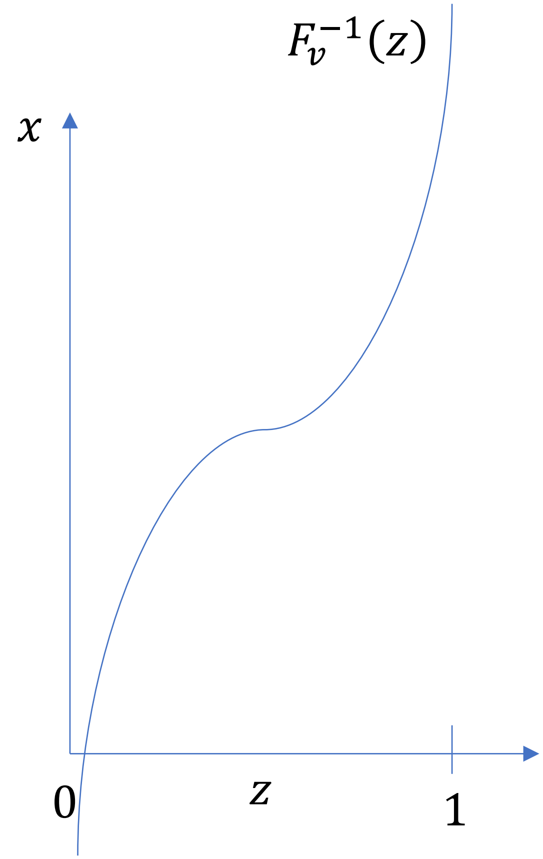
- From the previous slide, we have that
$$F_x(a) = F_z\left(G^{-1}(a)\right)$$
- Now take the derivative of both sides with respect to a
- $\frac{dF_x(a)}{da} = \frac{dF_z\left(G^{-1}(a)\right)}{da}$
- $p_x(a) = \frac{dF_z\left(G^{-1}(a)\right)}{d\left(G^{-1}(a)\right)}\left(\frac{dG^{-1}(a)}{da}\right)$
- $p_x(a) = p_z\left(G^{-1}(a)\right)\left(\frac{dG^{-1}(a)}{da}\right)$
- $p_x(a) = p_z\left(G^{-1}(a)\right)\left|\frac{dG^{-1}(a)}{da}\right|$
- What about decreasing functions?

Derivation of change of variables using CDF function (Decreasing)

- Assume $x = G(z)$, where $G(z)$ is a decreasing function, i.e.,
$$z_1 \le z_2 \Rightarrow G(z_1) \ge G(z_2)$$
$$z_1 \le z_2 \Rightarrow G^{-1}(z_1) \ge G^{-1}(z_2)$$
- $F_x(a) = \Pr(x \le a) = \Pr(G(z) \le a)$
- $= \Pr\left(G^{-1}(G(z)) \le G^{-1}(a)\right)$
- $= \Pr\left(z \ge G^{-1}(a)\right)$
- $= 1 - F_z\left(G^{-1}(a)\right)$
- Now take the derivative of both sides with respect to a
$$\frac{dF_x(a)}{da} = p_x(a)$$
$$-\frac{dF_z\left(G^{-1}(a)\right)}{da} = -\frac{dF_z\left(G^{-1}(a)\right)}{d\left(G^{-1}(a)\right)}\left(\frac{dG^{-1}(a)}{da}\right)$$
$$= -p_z\left(G^{-1}(a)\right)\left(\frac{dG^{-1}(a)}{da}\right) = p_z\left(G^{-1}(a)\right)\left|\frac{dG^{-1}(a)}{da}\right|$$

Inverse transform sampling is based on change of variables

- $z \sim \text{Uniform}([0,1])$
- $v \sim \text{AnotherDistribution}$
- $x = F_v^{-1}(z)$, where F_v^{-1} is the inverse CDF for v
- What is the distribution of x?
- $p_x(x) = p_z\big(F_v(x)\big)\left|\frac{dF_v(x)}{dx}\right|$
- $p_x(x) = (1)|p_v(x)| = p_v(x)$



What about change of variables in higher dimensions?

- Let's again build a little intuition (see demo)
- Again, conservation of volume: Consider infinitesimal expansion or shrinkage of volume

$$\mathrm{p}(x_1, x_2)|dx_1 dx_2| = p(z_1, z_2)|dz_1 dz_2|$$

- Given that Jacobian is all mixed derivatives we get generalization for vector to vector invertible functions:

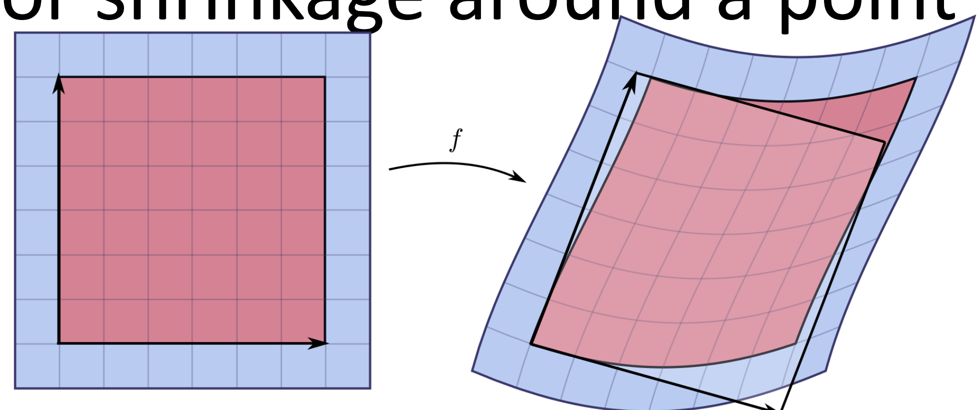
$$p_x(x) = |\det J_{G^{-1}}(x)|\, p_z\big(G^{-1}(x)\big)$$

Interpretation: What is the Jacobian again? The best linear approximation at a point

- The Jacobian definition:

$$\frac{dz}{dx} = J_z(x) = \begin{bmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_d}{\partial x_1} & \cdots & \frac{\partial z_d}{\partial x_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial G^{-1}(x)_1}{\partial x_1} & \cdots & \frac{\partial G^{-1}(x)_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial G^{-1}(x)_d}{\partial x_1} & \cdots & \frac{\partial G^{-1}(x)_d}{\partial x_d} \end{bmatrix}$$

- The determinant measures the local *linear* expansion or shrinkage around a point



Fact: The determinant Jacobian of compositions of functions is the product of determinant Jacobians

- Suppose $F(x) = F_2\big(F_1(x)\big)$
- The Jacobian expands like the chain rule
$$J_F(x) = J_{F_2}\big(F_1(x)\big)J_{F_1}(x) = J_{F_2}J_{F_1}$$
- If we take the determinant of the Jacobian, then it becomes a product of determinants
$$\det J_F = \det J_{F_2}J_{F_1} = \big(\det J_{F_2}\big)\big(\det J_{F_1}\big)$$
- This will be useful since each layer of our flows will be invertible

Okay, now back to learning flows: The log likelihood is the sum of determinant terms for each layer

- Simply optimize the minimize negative log likelihood where $F_\theta = G^{-1}$

$$\arg\min_{F_\theta} -\frac{1}{n}\sum_i \log p_x(x_i;\theta)$$

- $-\frac{1}{n}\sum_i \log p_z\left(F_\theta(x_i)\right)\left|\det J_{F_\theta}(x_i)\right|$
- $-\frac{1}{n}\sum_i\left[\log p_z\left(F_\theta(x_i)\right) + \log\left|\det J_{F_\theta}(x_i)\right|\right]$
- $-\frac{1}{n}\sum_i\left[\log p_z\left(F_\theta(x_i)\right) + \sum_\ell \log\left|\det J_{F_\theta^{(\ell)}}\left(z_i^{(\ell-1)}\right)\right|\right]$

$$\text{where } z_i^0 = x_i, \qquad \text{and} \quad z_i^\ell = F_\theta^{(\ell)}\left(z_i^{\ell-1}\right)$$

Overview of Normalizing Flows

22

Motivation

Definition and comparison to GANs and VAEs

Objective function for flows

- Change of variables formula in 1D
- Generalization to higher dimensions via determinant of Jacobian
- Log likelihood of flows

Normalizing flow architectures

- Design requirements
- Autoregressive and inverse autoregressive
- RealNVP and Glow architecture ideas

Normalizing flow architectures: How do we create these invertible layers?

- Consider arbitrary invertible transformation F_θ
  - How often would $\left|\det J_{F_\theta}\right|$ need to be computed?

- High computation costs
  - Determinant costs roughly $O(d^3)$ even if Jacobian is already computed!
  - Would need to be computed every stochastic gradient iteration

How do we create these invertible layers? Independent transformation on each dimension

- $z_1 = F_1(x_1)$
- $z_2 = F_2(x_2)$
- $z_3 = F_3(x_3)$
- What is the Jacobian?

$$J_F = \begin{bmatrix} \frac{dF_1(x_1)}{dx_1} & 0 & 0 \\ 0 & \frac{dF_2(x_2)}{dx_2} & 0 \\ 0 & 0 & \frac{dF_3(x_3)}{dx_3} \end{bmatrix}$$

How do we create these invertible layers? Autoregressive Flows based on chain rule

- Forward - Density estimation (in **parallel**)
  - $z_1 = F_1(x_1)$
  - $z_2 = F_2(x_2|x_1)$
  - $z_3 = F_3(x_3|x_1, x_2)$
- Inverse – Sampling (conditioned on x so must be **sequential**)
  - $x_1 = F_1^{-1}(z_1)$
  - $x_2 = F_2^{-1}(z_2|x_1)$
  - $x_3 = F_3^{-1}(z_3|x_1, x_2)$
- What is the Jacobian and determinant?
  - Product of diagonal!

$$J_F = \begin{bmatrix} \frac{dF_1}{dx_1} & 0 & 0 \\ \frac{dF_2}{dx_1} & \frac{dF_2}{dx_2} & 0 \\ \frac{dF_3}{dx_1} & \frac{dF_3}{dx_2} & \frac{dF_3}{dx_3} \end{bmatrix}$$

Rezende, D., & Mohamed, S. (2015, June). Variational Inference with Normalizing Flows. In *International Conference on Machine Learning* (pp. 1530-1538).

How do we create these invertible layers? Inverse Autoregressive Flows based on chain rule

- Forward - Density estimation (**sequential**)
  - $z_1 = F_1(x_1)$
  - $z_2 = F_2(x_2|z_1)$
  - $z_3 = F_3(x_3|z_1, z_2)$
- Inverse – Sampling (**parallel**)
  - $x_1 = F_1^{-1}(z_1)$
  - $x_2 = F_2^{-1}(z_2|z_1)$
  - $x_3 = F_3^{-1}(z_3|z_1, z_2)$
- What is the Jacobian and determinant?
  - Product of diagonal!

$$J_F = \begin{bmatrix} \frac{dF_1}{dx_1} & 0 & 0 \\ \frac{dF_2}{dx_1} & \frac{dF_2}{dx_2} & 0 \\ \frac{dF_3}{dx_1} & \frac{dF_3}{dx_2} & \frac{dF_3}{dx_3} \end{bmatrix}$$

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems* (pp. 4743-4751).

Scale-and-shift simple form of invertible functions (MAF https://arxiv.org/pdf/1705.07057.pdf)

- Forward – Density estimation (**parallel**)
  - $z_1 = \exp(\alpha_1) x_1 + \mu_1$
  - $z_2 = \exp(\alpha_2) x_2 + \mu_2,\ \alpha_2 = f_2(x_1),\ \mu_2 = g_2(x_1)$
  - $z_3 = \exp(\alpha_3) x_3 + \mu_3,\ \alpha_3 = f_3(x_1, x_2),\ \mu_3 = g_3(x_1, x_2)$
- What is the Jacobian and determinant?

$$J_F = \begin{bmatrix} \exp(\alpha_1) & 0 & 0 \\ \frac{dz_2}{dx_1} & \exp(\alpha_2) & 0 \\ \frac{dz_3}{dx_1} & \frac{dz_3}{dx_2} & \exp(\alpha_3) \end{bmatrix}$$

RealNVP and GLOW: Several key architecture ideas for **image-based** normalizing flows

1. Coupling layers
2. Invertible squeeze operation
3. Split dimensions along channel
4. Hierarchical structure
5. 1 x 1 convolutions

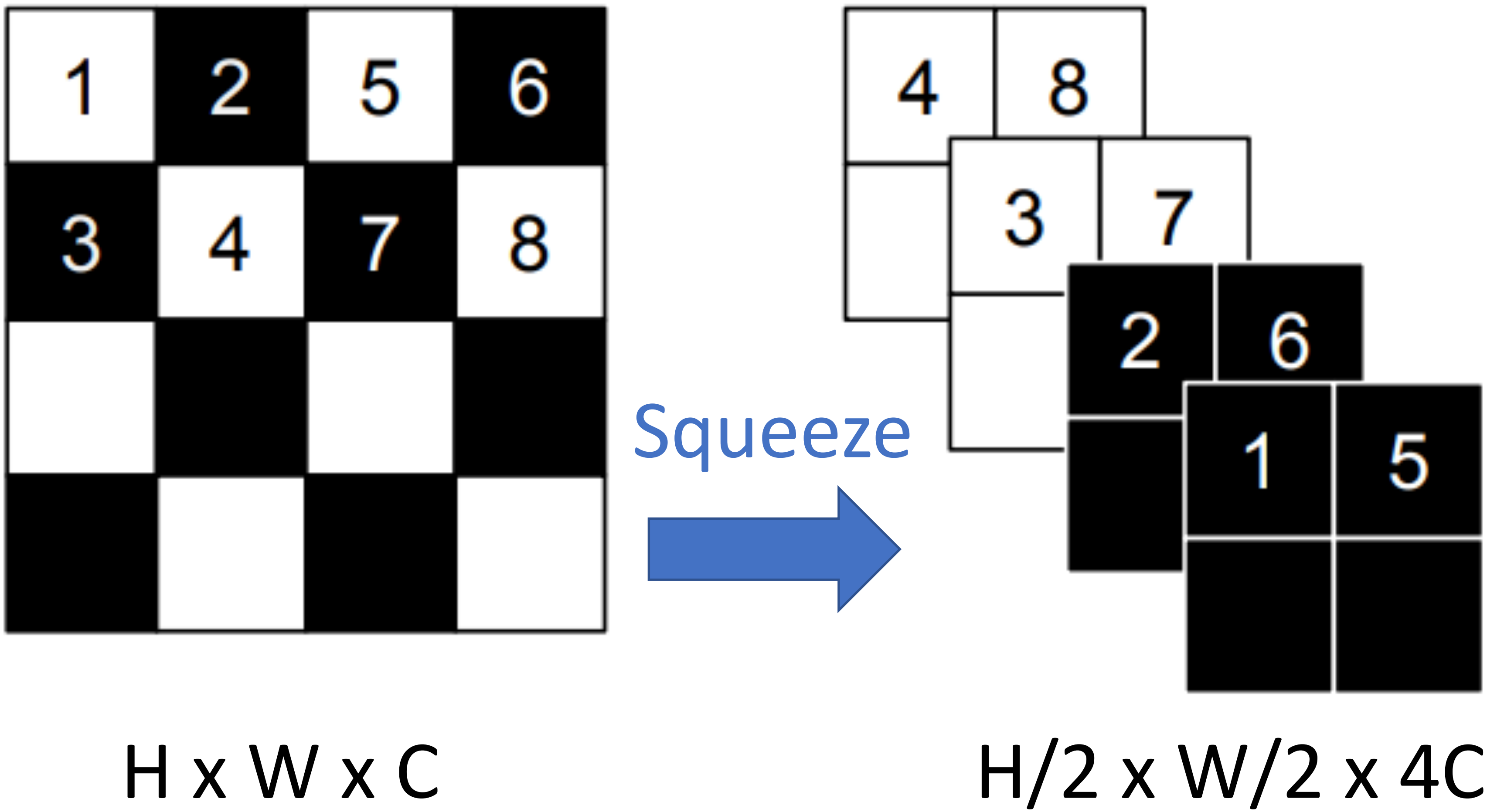
<u>Coupling layers</u> allow parallel density estimation **and** sampling

- Keep some set of features **fixed** and transform others
  - $z_{1:i-1} = x_{1:i-1}$
  - $z_{i:d} = \exp\big(f(x_{1:i-1})\big) \odot x_{i:d} + g(x_{1:i-1})$
- Reverse or shuffle coordinates and repeat
- What is Jacobian?

$$J_F = \begin{bmatrix} I & 0 \\ J_{cross} & \operatorname{diag}\Big(\exp\big(f(x_{1:i-1})\big)\Big) \end{bmatrix}$$

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

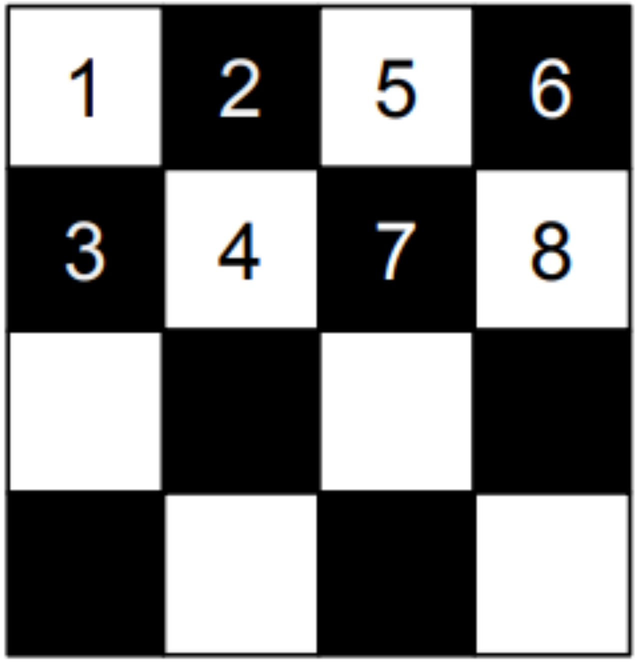
How to split dimensions for coupling layers?
The squeeze operation trades off between spatial and channel dimensions

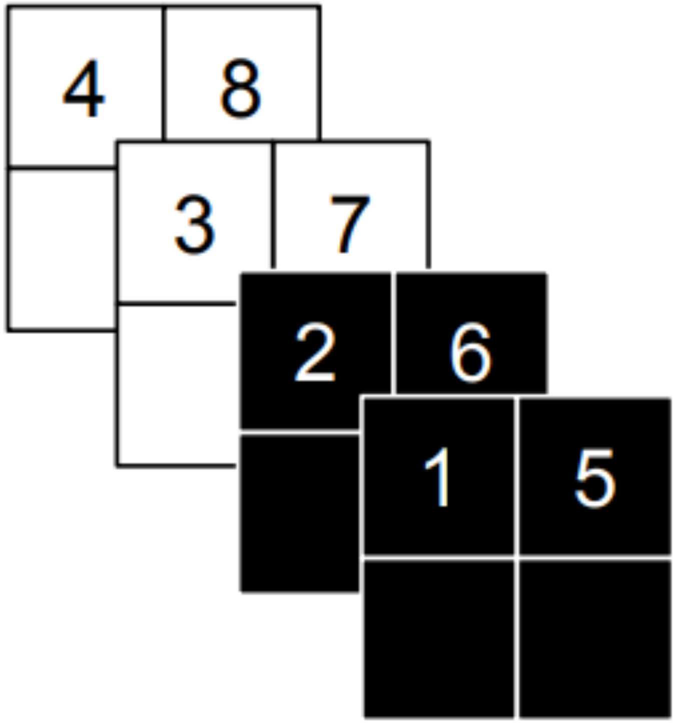


Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

How to split dimensions for coupling layers? Checkboard or channel-wise masking can be used to separate fixed and non-fixed set of variables

White are **fixed, i.e.,** $x_{1:i-1}$, and black are **transformed**, $x_{i:d}$.

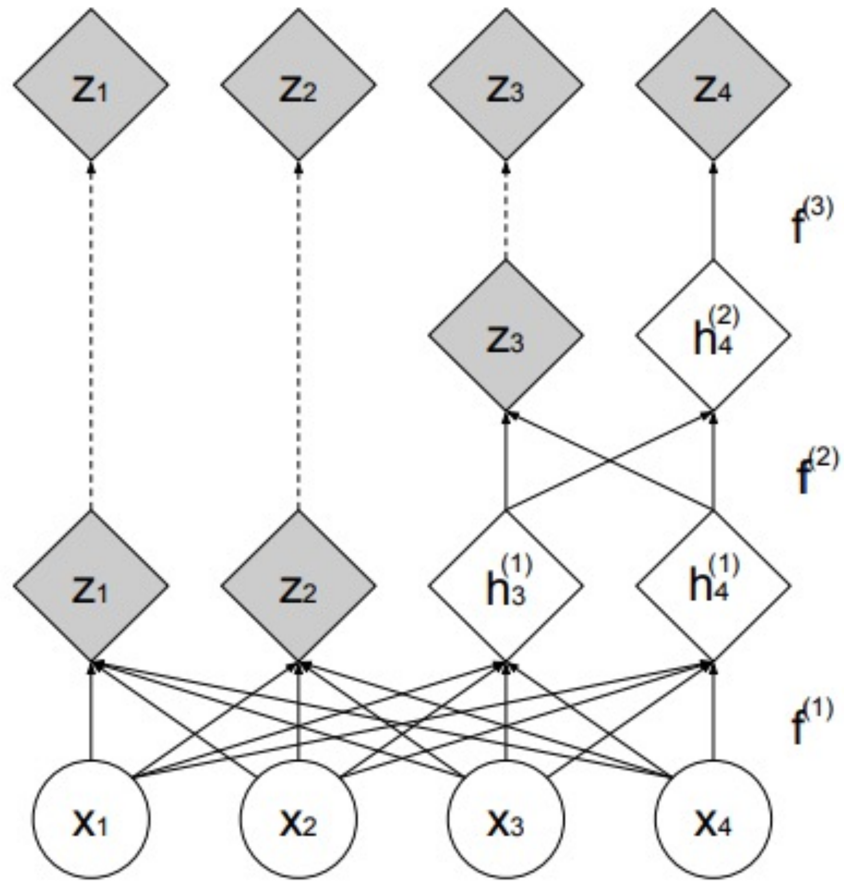




Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Hierarchical factorization is like an invertible dimensionality reduction method

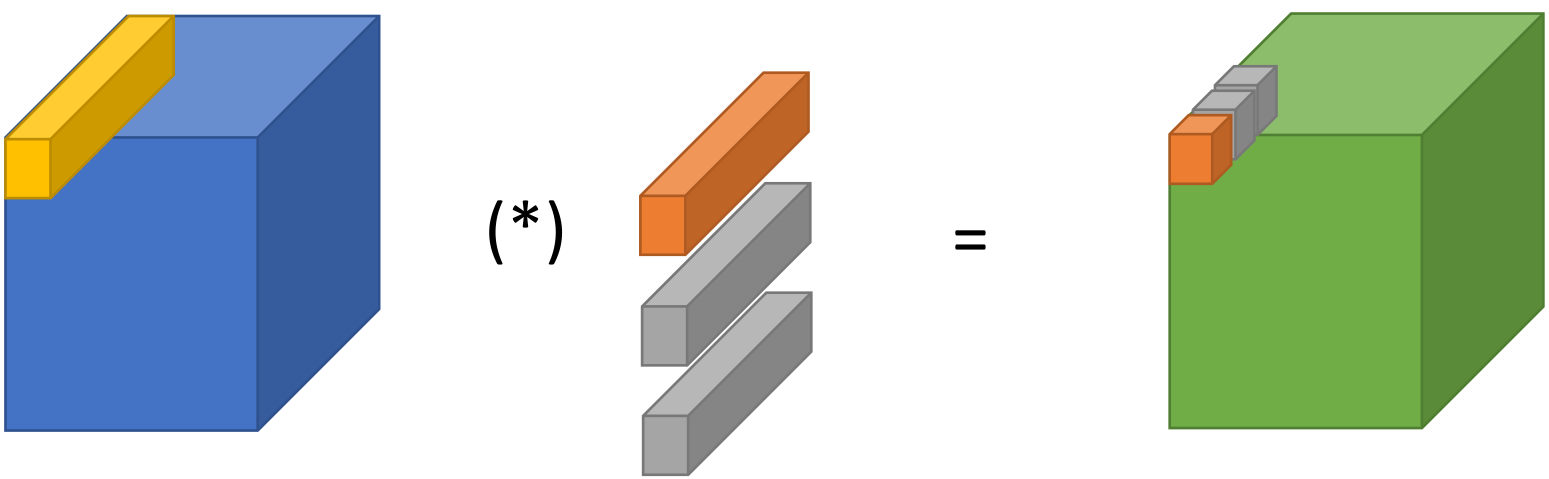
- After each block, half of the dimensions are fixed and the rest pass through more transformations
- Intuitively, the important part of the signal propagates deeper



Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

GLOW: Convolutional flows
1 x 1 invertible convolutions are like fully connected layers for each pixel

- Image tensor: $\mathrm{h} \times w \times c$
- If we use c filters than we map from a $\mathrm{h} \times w \times c$ to another $\mathrm{h} \times w \times c$ image
- The number of parameters is a matrix $W \in \mathbb{R}^{c \times c}$
- 1x1 convolutions can be seen as a linear transform along the channel dimension (mixes the channel dimensions)



Highly realistic random samples from powerful flow model (GLOW)

Figure 1: Synthetic celebrities sampled from our model; see Section 3 for architecture and method, and Section 5 for more results.

https://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf

More recent normalizing flows

- Neural Spline Flows and Neural Autoregressive Flows – Add more flexible 1D transforms
- Flow++ - Careful tweaks to some previous models
- FFJORD – Uses neural Ordinary Differential Equations (ODE) to **implicitly** define invertible functions
- Residual Flows – Careful construction of residual networks that are invertible (uses Lipschitz idea)
- MaCow – Masked Convolutional Generative Flow (carefully constructed masked convolutions to ensure invertibility)

Similar concepts can be used to generate realistic audio (WaveGlow)

- Listen to some examples
  https://nv-adlr.github.io/WaveGlow

- Very similar concepts for audio generation