Glow: Generative Flow with Invertible 1x1 Convolutions

By Diederik P. Kingma and Prafulla Dhariwal

Presented by Roderick Huang 11/17/2021

Log-likelihood-based methods

- Tractability of the exact log-likelihood, tractability of exact latent-variable inference, parallelizability of training and synthesis
- Three methods
  - Autoregressive Models
    - Disadvantage that synthesis has limited parallelizability
    - Lot of hidden layers with unknown marginal distributions, which makes it difficult to manipulate data
  - Variational Autoencoders
    - Optimizing a lower bound on the log-likelihood of data
  - Flow-based generative models
    - Glow builds off RealNVP

Basics of Normalizing Flow

$$f = f_4 \circ f_3 \circ f_2 \circ f_1$$

$$f^{-1} = f_1^{-1} \circ f_2^{-1} \circ f_3^{-1} \circ f_4^{-1}$$

- Let x be discrete data
  - ↳ Unknown 'true' distribution $x \sim p^*(x)$
- Let z be the latent variable
  - ↳ $z \sim p_\theta(z)$

    Ex: Spherical multivariate Gaussian distribution

    $$p_\theta(z) = N(z; 0, I)$$

- Generative Flow Process:

$$z \sim p_\theta(z)$$

$$x = f_\theta^{-1}(z)$$

$x = f_\theta(z)$

- Let $f = f_1 \circ f_2 \circ \cdots \circ f_K$

$$f^{-1} = f_K^{-1} \circ f_{K-1}^{-1} \circ \cdots \circ f_1^{-1}$$

$$x \xleftrightarrow{f_1} h_1 \xleftrightarrow{f_2} h_2 \cdots \xleftrightarrow{f_K} z$$

$$\det\left(\frac{\partial z}{\partial x}\right) = \det \prod_{i=1}^{K} \frac{\partial h_i}{\partial h_{i-1}}$$

$$\log\left|\det\left(\frac{\partial z}{\partial x}\right)\right| = \sum_{i=1}^{K} \log\left|\det \frac{\partial h_i}{\partial h_{i-1}}\right|$$

- Change of variables formula:

$$p_\theta(x) = p_\theta(z)\left|\det\left(\frac{\partial z}{\partial x}\right)\right|$$

Note: paper states $h_0 \hat{=} x$ & $h_K \hat{=} z$

$$\log p_\theta(x) = \log p_\theta(z) + \sum_{i=1}^{K} \log\left|\det\left(\frac{\partial h_i}{\partial h_{i-1}}\right)\right|$$

Jacobian Matrix

- No need to care about the Jacobian itself, we just care about the determinant of the Jacobian

Goal: Block triangular matrix

$$\begin{array}{cc} \frac{\partial z_A}{\partial x_A} & \frac{\partial z_A}{\partial x_B} \end{array}$$

$$Df(x) = \begin{bmatrix} I & 0 \\ \frac{\partial}{\partial x^A} \hat{f}(x^B \mid \theta(x^A)) & D\hat{f}(x^B \mid \theta(x^A)) \end{bmatrix}$$

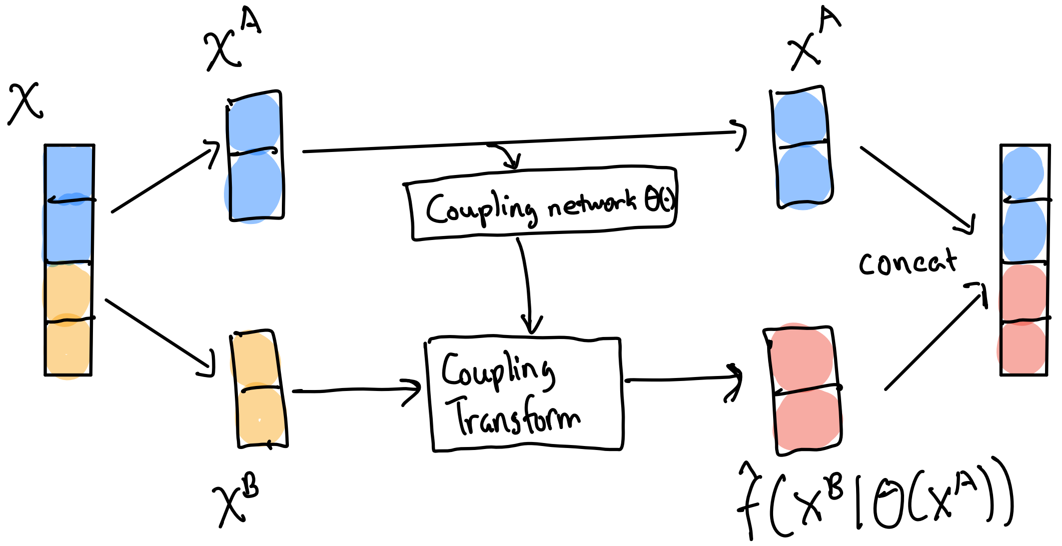
Coupling Flows

- General approach to construct non-linear flows

Partition the parameters into two disjoint subsets $x = (x^A, x^B)$. Then,

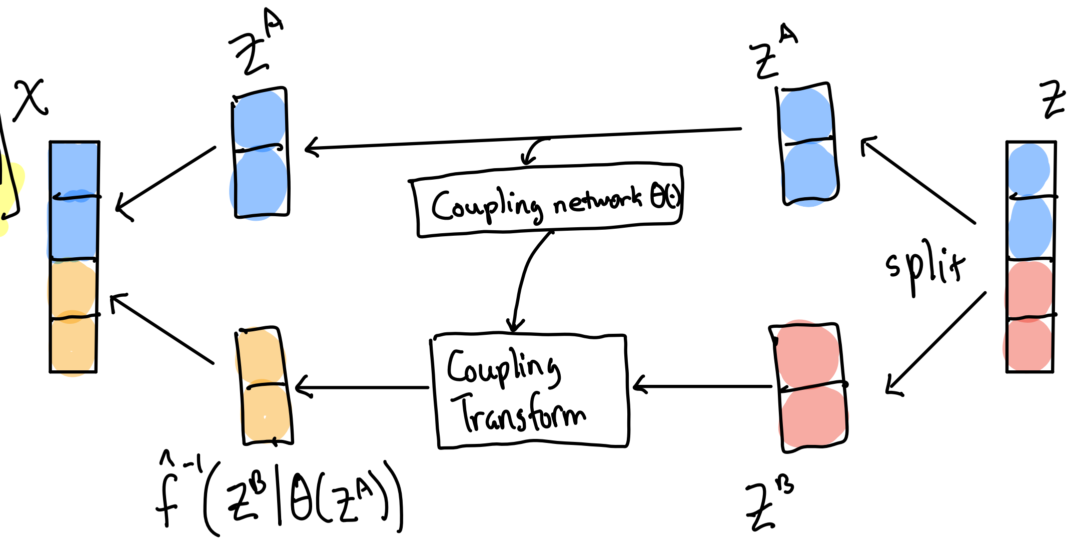
$$f(x) = (x^A, \hat{f}(x^B \mid \theta(x^A)))$$

where $\hat{f}(x^B \mid \theta(x^A))$ is another flow but whose parameters depend on x^A



$$f(x) = \begin{bmatrix} x^A \\ \hat{f}(x^B|\theta(x^A)) \end{bmatrix}$$

$$f^{-1}(z) = \begin{bmatrix} z^A \\ \hat{f}^{-1}(z|\theta(z^A)) \end{bmatrix}$$



Jacobian:

$$Df(x) = \begin{bmatrix} I & 0 \\ \frac{\partial}{\partial x^A}\hat{f}(x^B|\theta(x^A)) & \frac{\partial}{\partial x^B}\hat{f}(x^B|\theta(x^A)) \end{bmatrix}$$

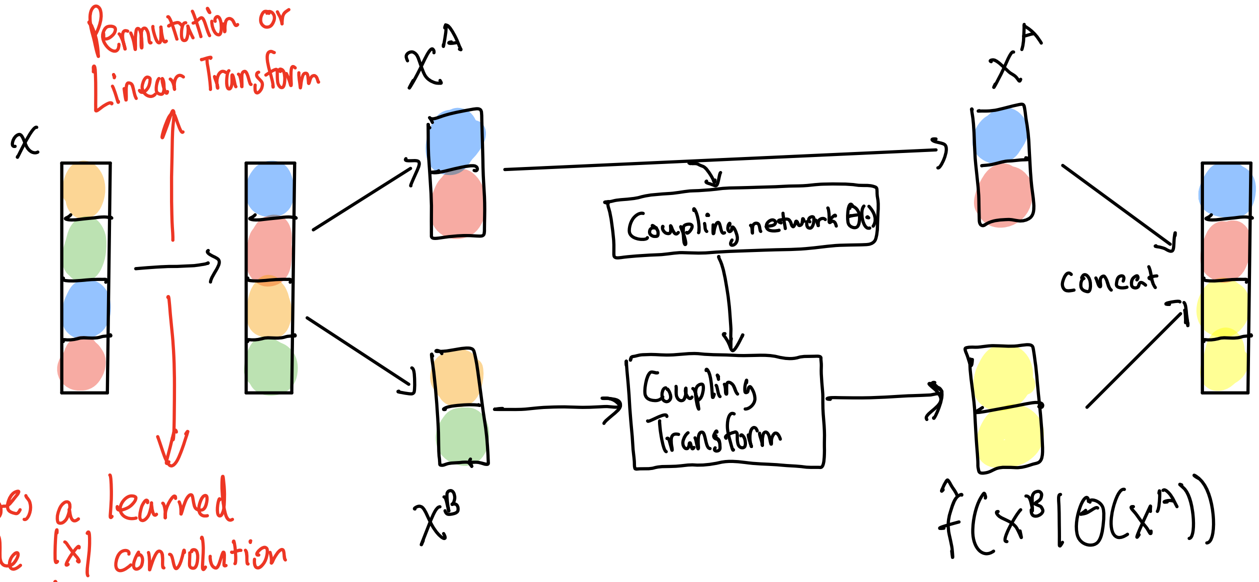
$$\det Df(x) = \det \frac{\partial}{\partial x^B}\hat{f}(x^B|\theta(x^A))$$

Using notation from the paper, $\det \frac{dh_i}{dh_{i-1}} = \prod_{i=1}^{K} diag\left(\frac{dh_i}{dh_{i-1}}\right)$

$$\Rightarrow \log\left|\det \frac{dh_i}{dh_{i-1}}\right| = sum\left(\log\left|diag\left(\frac{dh_i}{dh_{i-1}}\right)\right|\right)$$

- Can make $\theta(x^A)$ arbitrarily complex (MLP, CNN, RNN)

Multilayer Perceptron — Convolutional neural networks — Recurrent Neural Networks



Glow uses a learned invertible 1x1 convolution
↳ Block diagonal linear transformation

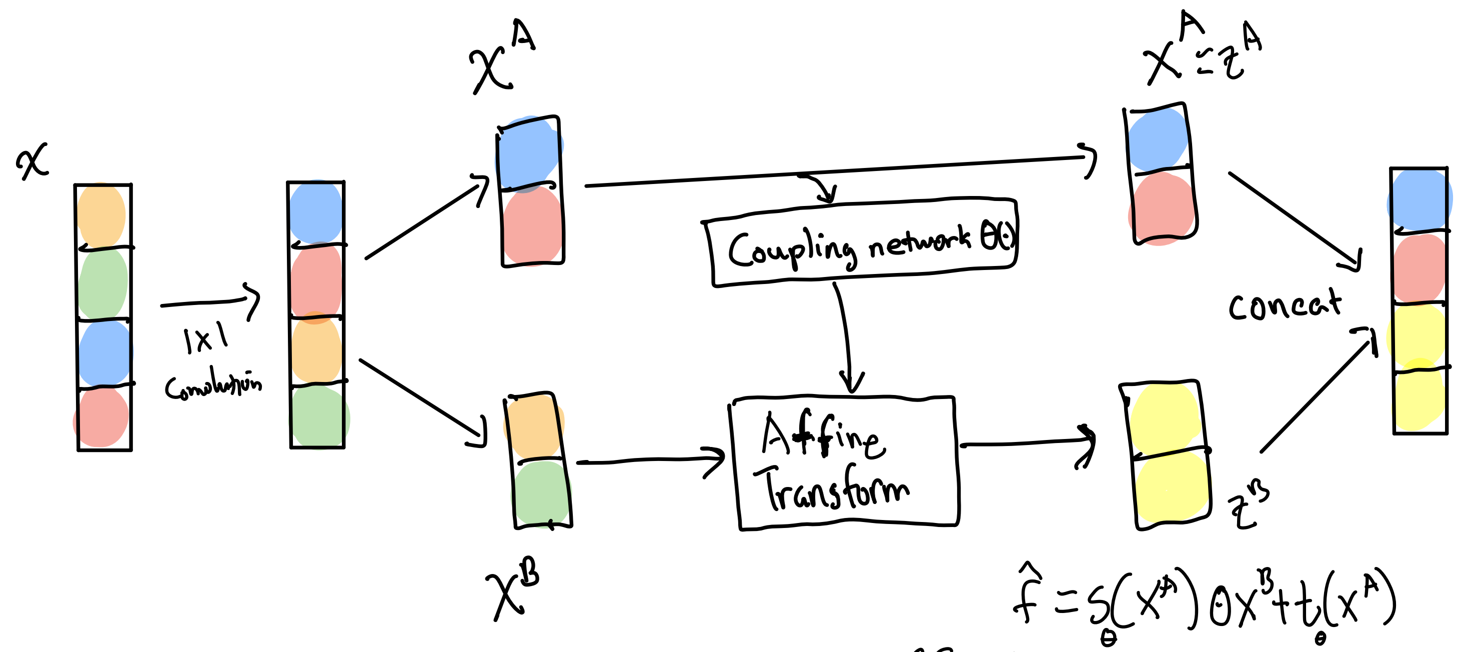
- RealNVP proposed a flow containing the equivalent of a permutation that reverses the ordering of channels
  - ↳ Benefits: ① Inverse of a permutation is its transpose
    ② Determinant of a permutation is 1 or -1
- Glow proposes to replace with a (learned) invertible 1x1 convolution where the weight matrix is initialized as a random rotation matrix
  - ↳ A 1x1 convolution w/ equal number of input and output channels is a generalization of a permutation operation.

Coupling Transform (What is $\hat{f}$)

- Additive $\hat{f}(x \mid t) = x + t$
- Affine (From RealNVP) $\hat{f}(x \mid s, t) = s \odot x + t$
  - ↳ commonly used coupling transform for flows

Hadamard product

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \odot \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} aj & bk & cl \\ dm & ne & fo \\ gp & hq & ir \end{bmatrix}$$



Deriving inverse of Affine transform:

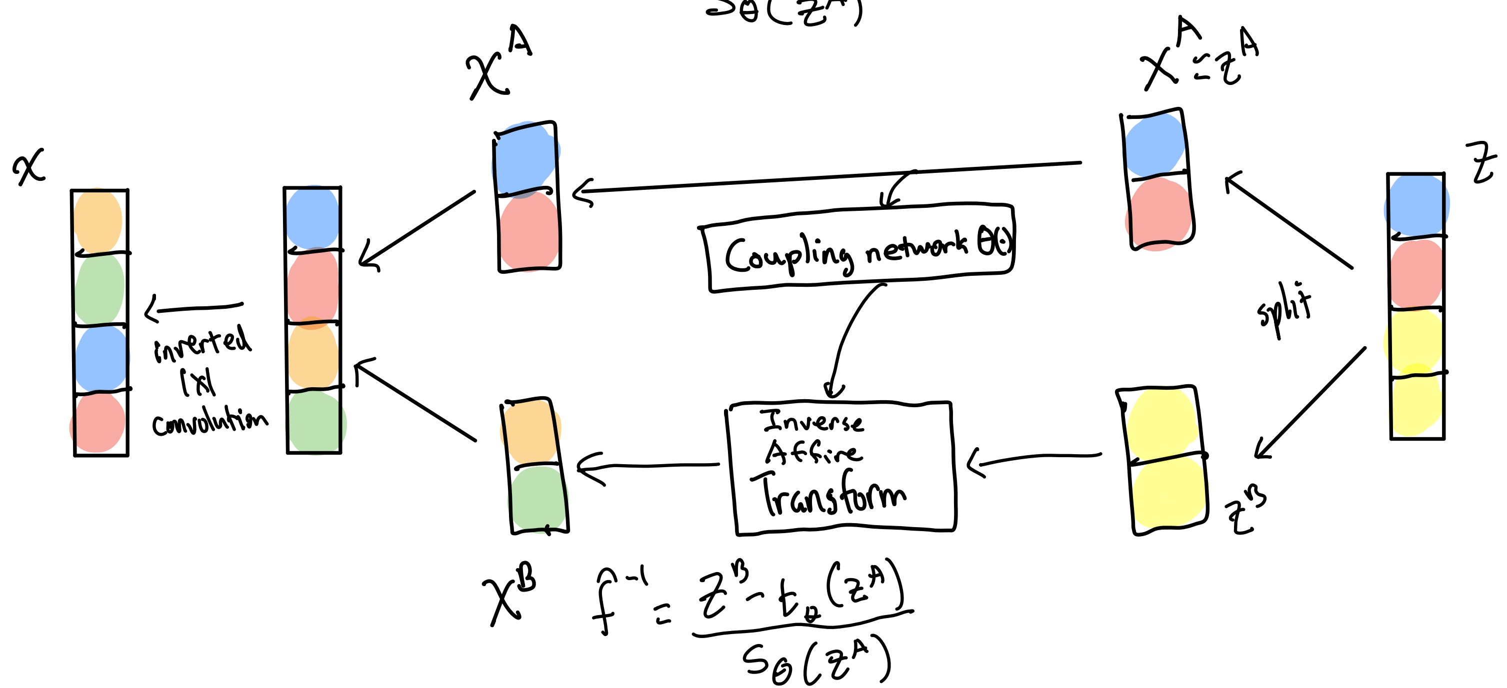
$$z^B = x^B \cdot s_\theta(x^A) + \overbrace{t_\theta(x^A)}^{\text{offset/constant}}$$

Arbitrary neural nets that must be differentiable

We know that $x^A = z^A$. So,

$$z^B = x^B \cdot s_\theta(x^A) + t_\theta(z^A)$$

$$x^B = \frac{z^B - t_\theta(z^A)}{s_\theta(z^A)}$$



$z_A = x_A$

$z_B = x_B \cdot s_\theta(x_A) + t_\theta(x_A)$

$$\frac{\partial z}{\partial x} = \begin{bmatrix} I & 0 \\ \frac{\partial z_B}{\partial x_A} & \mathrm{diag}(s_\theta(x_A)) \end{bmatrix}$$

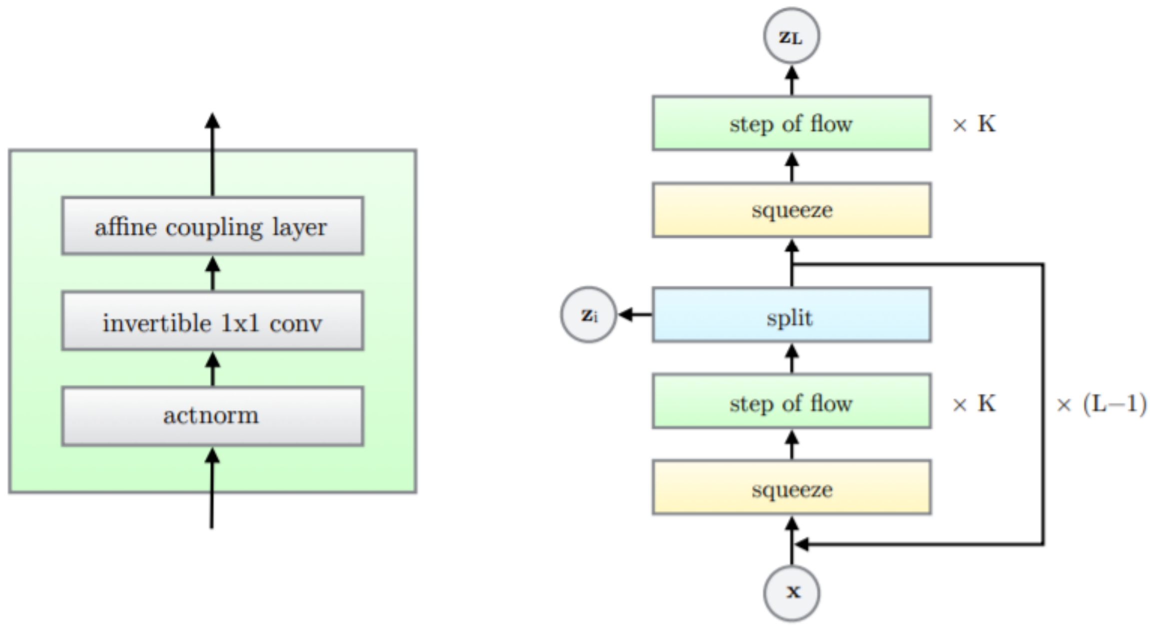
$$\det \frac{\partial z}{\partial x} = \prod_{k=1}^{d} s_\theta(x_A)_k$$

$$\text{Log-determinant} = \sum_{k=1}^{d} \log(s_\theta(x_A)_k)$$

Paper Notation: sum (log (|s|))

Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\mathtt{NN}()$ is a nonlinear mapping, such as a (shallow) convolutional neural network like in ResNets (He et al., 2016) and RealNVP (Dinh et al., 2016).

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)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t}) / \mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$



(a) One step of our flow.

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

① Actnorm: Hardware to test bits/dimension

Results

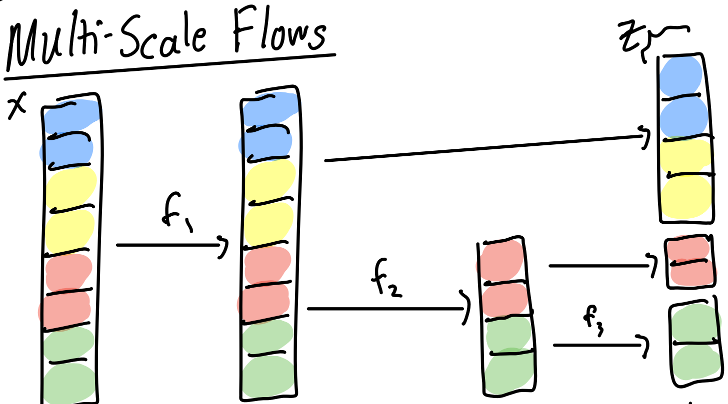
Using our techniques we achieve significant improvements on standard benchmarks compared to RealNVP, the previous best published result with flow-based generative models.

DATASET	REALNVP	GLOW
CIFAR-10	3.49	**3.35**
Imagenet 32x32	4.28	**4.09**
Imagenet 64x64	3.98	**3.81**
LSUN (bedroom)	2.72	**2.38**
LSUN (tower)	2.81	**2.46**
LSUN (church outdoor)	3.08	**2.67**

Quantitative performance in terms of bits per dimension evaluated on the test set of various datasets, for the RealNVP model versus our Glow model.*

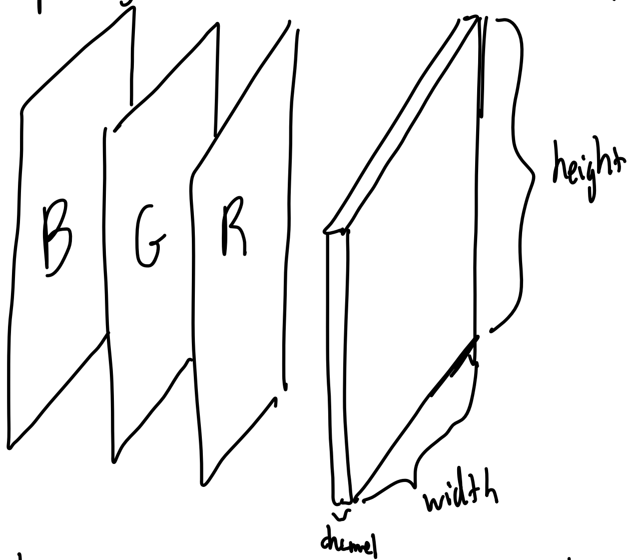
② Invertible learned 1x1 convolutions

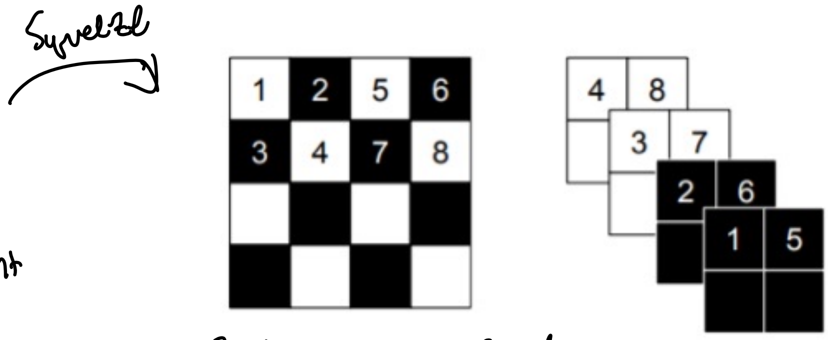
Multi-Scale Flows



Lower dimension representations → more channels

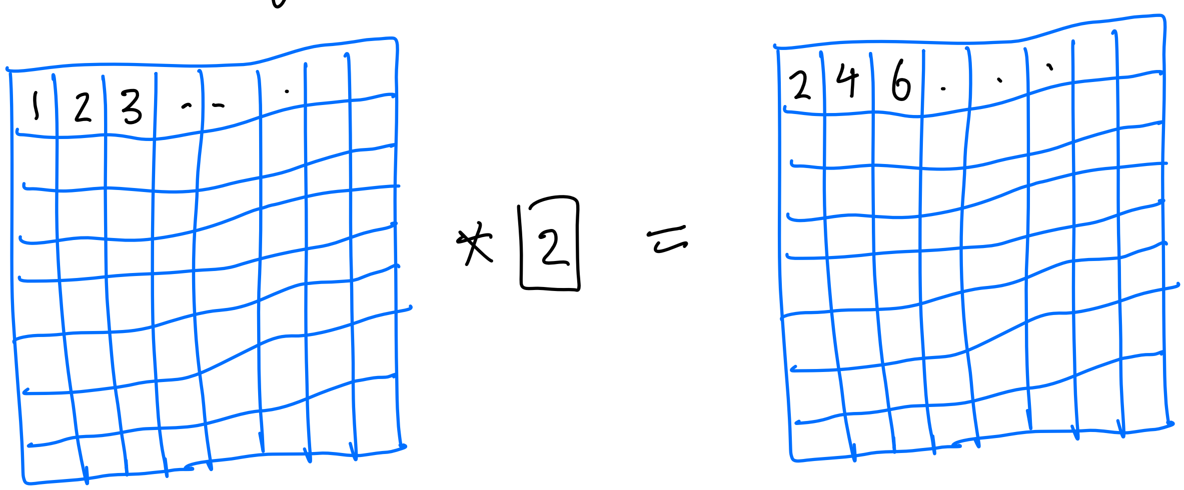
- Splitting dimensions for images

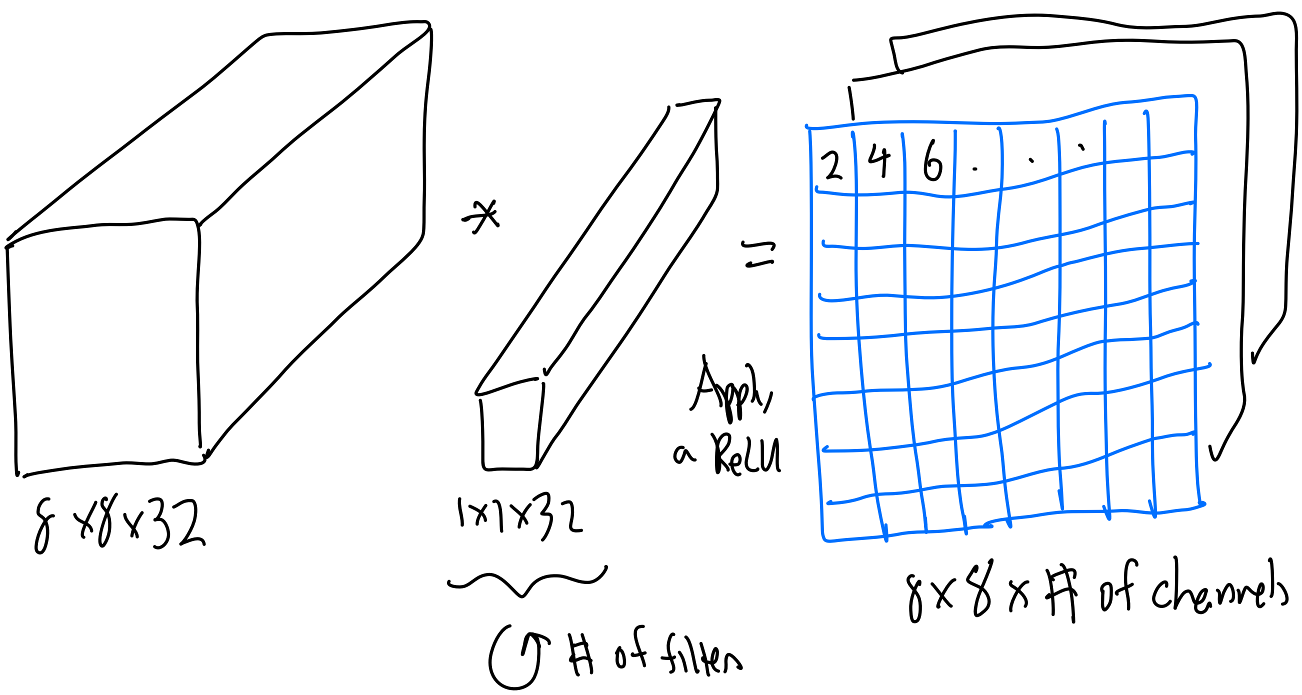




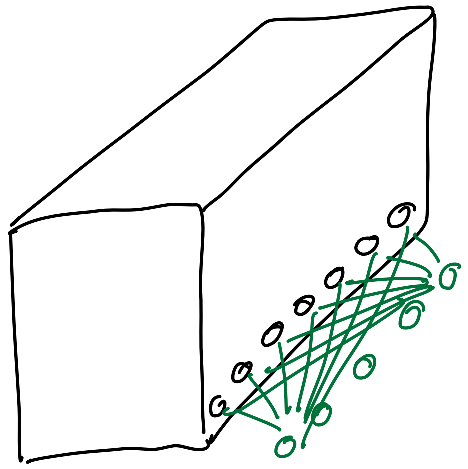
- RealNVP uses a fixed permutation

- Glow proposes to replace with a (learned) invertible 1x1 convolution where the weight matrix is initialized as a random rotation matrix





Each time you apply a convolution, it is like a NN.



Apply a 1×1 convolution of a $h\times w\times c$ tensor $\bar{h}$ with $c\times c$ weight matrix W

↳ W is initialized as random rotation matrix

↳ log determinant of 0

↳ value will diverge from 0 after one SGD

Log-determinant of a 1×1 convolution:

$$\log\left|\det\left(\frac{d\,\mathrm{conv2D}(\bar{h};W)}{d\bar{h}}\right)\right| = h\cdot w\cdot\log|\det(W)|$$

LU Decomposition: Reduce cost of computing $\det(W)$ from $O(c^3)$ to $O(c)$ by parametrizing W directly in its LU decomposition:

$$W = \underbrace{P}_{\text{fixed permutation matrix}}\;\underbrace{L}_{\text{lower triangular matrix}}\,(\underbrace{U}_{\text{upper triangular matrix}} + \mathrm{diag}(\underbrace{s}_{\text{vector}}))$$

$$\log|\det(W)| = \mathrm{sum}(\log|s|)$$

$\Rightarrow$ log-determinant is $h\cdot w\cdot \mathrm{sum}(\log|s|)$

Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\texttt{NN}()$ is a nonlinear mapping, such as a (shallow) convolutional neural network like in ResNets (He et al., 2016) and RealNVP (Dinh et al., 2016).

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i, j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
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 $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

LU decomposition

$$\begin{tikzcd} x \arrow[r, "1"'] & z \arrow[l, bend right, "2"'] \end{tikzcd}$$

Images

smile

$(x^{(1)} \cdots x^{(k)})$

$\downarrow$

$(z^{(1)} \cdots z^{(k)})$

no smile

$(x^{(k+1)} \cdots x^{(n)})$

$(z^{(k+1)} \cdots z^{(n)})$

$$x^{PERSON} \to z^{PERSON} + \alpha \left(z^{AVG}_{SMILE} - z^{AVG}_{NOSMILE} \right)$$

scaling

Average latent vector