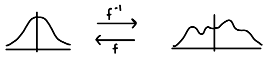
# Normalizing Flows from the Basics

Goal: Model a complex distribution by transforming a simple distribution



$Z \sim p_z = N(0,1)$   $X \sim \underbrace{p_x(x)}_{?}$

PDF

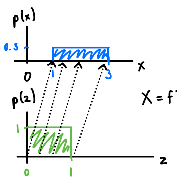
## Change of Variable Formula

Question: How can we model $p_x(x)$ using the known distribution $p_z(z)$?

- Use the change of variable formula

$$\boxed{p_x(x) = p_z(z)\left|\frac{dz}{dx}\right| = p_z(f(x))\left|\frac{d f(x)}{dx}\right|}$$

- Intuitive Breakdown
  - We are "transforming" the probability mass
  - $p_z(f(x))$ is the probability mass at $f(x)$ for $p_z$
  - $\left|\frac{df(x)}{dx}\right|$ is how much the probability mass needs to be squeezed or expanded to keep the volume the same




$X = f^{-1}(Z) = 2X+1$

- Expansion to Multivariate
  - Suppose we have two random vectors $\vec{X}$ and $\vec{Z}$ and invertible function $\vec{Z} = f(\vec{X})$
  - Change of Variable Formula

$$p_{\vec{x}}(\vec{x}) = p_{\vec{z}}(f(\vec{x}))\left|\underbrace{\det}_{\text{determinant}}\left(\underbrace{\frac{df(\vec{x})}{d\vec{x}}}_{\text{Jacobian}}\right)\right|$$

$$\frac{df(\vec{x})}{d\vec{x}} = \begin{bmatrix} \frac{\partial f_1(\vec{x})}{\partial x_1} & \cdots & \frac{\partial f_d(\vec{x})}{\partial x_d} \\ \vdots & \ddots & \\ \frac{\partial f_d(\vec{x})}{\partial x_1} & \cdots & \frac{\partial f_d(\vec{x})}{\partial x_d} \end{bmatrix}$$

  - $\left|\det\left(\frac{df(\vec{x})}{d\vec{x}}\right)\right|$ is the amount $f$ expands or shrinks a volume

## Deriving the Change of Variable Formula

- First, note if $f$ is invertible, it must be either monotonically increasing or decreasing (in order to be one-to-one)

**Monotonically Increasing**

$$\mathbb{P}(X \le x) = \mathbb{P}(f^{-1}(Z) \le x)$$
$$= \mathbb{P}(Z \le f(x))$$

Probability

$$\downarrow$$

CDF

$$P_x(x) = P_z(f(x))$$
$$\downarrow \tfrac{d}{dx}$$
$$p_x(x) = \frac{d}{dx}P_z(f(x))$$
$$= \frac{dP_z(f(x))}{dz}\frac{df(x)}{dx} \quad \text{(Chain Rule)}$$
$$= p_z(f(x))\underbrace{\frac{df(x)}{dx}}_{\ge 0 \text{ if mono} \uparrow}$$

**Monotonically Decreasing**

$$\mathbb{P}(X \le x) = \mathbb{P}(f^{-1}(Z) \le x)$$
$$= \mathbb{P}(Z \ge f(x))$$
$$\downarrow$$
$$P_x(x) = 1 - P_z(f(x))$$
$$\downarrow \tfrac{d}{dx}$$
$$p_x(x) = -\frac{d}{dx}P_z(f(x))$$
$$= -\frac{dP_z(f(x))}{dz}\frac{df(x)}{dx}$$
$$= -p_z(f(x))\underbrace{\frac{df(x)}{dx}}_{\le 0 \text{ if mono} \downarrow}$$

$$p_x(x) = p_z(f(x))\left|\frac{df(x)}{dx}\right|$$

## Maximum Likelihood

- Problem: we don't have function $f$ and we only have a finite number of samples from $p_{\vec{x}}$, $\{\vec{x}_i\}_{i=1}^n$
- Say our function $f_\theta$ is parameterized by $\theta$
- We can get a distribution with this function $f_\theta$

$$\underbrace{p_\theta(\vec{x})}_{= p(x|\theta)\ \text{likelihood}} = p_z(f_\theta(\vec{x}))\left|\det\left(\underbrace{\frac{\partial f_\theta(\vec{x})}{\partial \vec{x}}}_{\text{Jacobian}}\right)\right|$$

- Question: can we find parameters $\theta$ so $p_\theta \approx p_x$?
- One approach: maximum likelihood
  - Big idea: Try to maximize the likelihood $p_\theta$ for the known samples $\{\vec{x}_i\}_{i=1}^n = \mathcal{X}$

$$\max_\theta p_\theta(\mathcal{X}) \longleftrightarrow \max_\theta \log p_\theta(\mathcal{X}) = \max_\theta \sum_{i=1}^n \log p_\theta(\vec{x}_i)$$
$$= \boxed{\max_\theta \sum_{i=1}^n \log p_z(f_\theta(\vec{x}_i)) + \log\left|\det\left(\frac{df_\theta(\vec{x}_i)}{dx_i}\right)\right|}$$

Loss Function

## Components of a Flow

- Question: How should we construct $f_\theta$ ?
  - We want $f_\theta$ to be complex enough to get a good estimate of $p_X$
  - But if $f_\theta$ is too complex, $f_\theta^{-1}$ and $\log\left|\det\left(\frac{d f_\theta(\vec{x})}{\vec{x}}\right)\right|$ are hard to compute

- Idea: Compose many simple transformations (a flow of transformations)

  $f_\theta = f_{\theta_1} \circ f_{\theta_2} \circ \ldots \circ f_{\theta_K}$

  $$\det \frac{\partial f_\theta^{-1}(\vec{x}_i)}{\partial \vec{x}_i} = \prod_{k=1}^{K} \det \frac{\partial f_\theta^{-1}(\vec{x}_i^{(k)})}{\partial \vec{x}_i^{(k)}}$$

  intermediate value

  Using chain rule

- Requirements of each component
  1) Needs to be quickly invertible
  2) Needs to have an easily computable Jacobian

- Main Components

  1) Coupling Layer
     - Main type of transformation
     - Several different types

     **Affine Coupling**

     i) Split input into 2 parts, $\vec{x}_{1:j}$ and $\vec{x}_{j+1:d}$ (usually channel-wise)

     ii) One part remains unchanged

     $\vec{y}_{1:j} = \vec{x}_{1:j}$

     iii) Other part is scaled + translated by $\vec{s}_{\theta_K}, \vec{t}_{\theta_K} = \underbrace{NN_{\theta_K}}_{\text{Neural Network}}(\vec{x}_{1:j})$

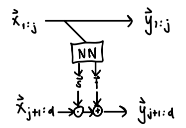
     Note: $\vec{s}_{\theta_K}$ is often modified to limit the amount of scaling for stability

     Ex: $\vec{s} = e^{c \tanh(\vec{s}_{\theta_K})}$

     constant

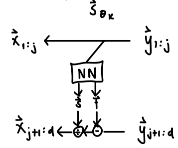
     Forward

     $\vec{y}_{j+1:d} = \vec{x}_{j+1:d} \odot \vec{s}_{\theta_K} + \vec{t}_{\theta_K}$




     Reverse

     $\vec{x}_{j+1:d} = \frac{\vec{y}_{j+1:d} - \vec{t}_{\theta_K}}{\vec{s}_{\theta_K}}$




     iv) The Jacobian is triangular so the determinant is just the product of the diagonal

     $$\det \frac{d\vec{y}}{d\vec{x}} = \prod_{i=1}^{d-(j+1)} s_{\theta_K, i}$$

  2) Activation Normalization (Glow)
     - Similar to Batch Norm but no batch dependent operations
     - Helps w/ training deep flows
     - Initialized so post actnorm activations per-channel have zero mean unit variance
     - Each spatial location is scaled + bias by learned parameters

     Forward

     $\underbrace{\forall i,j}_{\text{spatial indices}} \quad \vec{y}_{i,j} = \vec{s} \odot x_{i,j} + \vec{b}$

     learned parameters

     Reverse

     $x_{i,j} = \frac{(y_{i,j} - \vec{b})}{\vec{s}}$

     Log-det

     $h \cdot w \cdot \sum \log|\vec{s}|$

## 3) Permutation

- Permute the features so across flow each feature can affect another feature

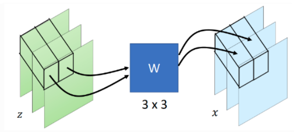
### 1x1 Convolution (Glow)

- Learns a 1x1 conv to do the permutation

Forward: $\forall i,j: y_{i,j} = W x_{i,j}$

Reverse: $x_{i,j} = W^{-1} y_{i,j}$

Log Det: $h \cdot w \cdot \log|\det(W)|$




- This can be slow to find det W so can speed up w/ LU decomposition

$$W = \underbrace{P}_{\text{Perm Matrix}}\,\underbrace{L}_{\text{Lower Triangular}}(\underbrace{U}_{\text{Upper Triangular}} + \text{diag}(\underbrace{s}_{\text{vector}})) \rightarrow \log|\det(W)| = \sum(\log|s|)$$

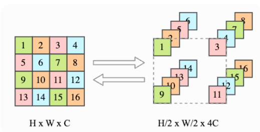
Ex: 2x2 matrix

$$\overset{W}{\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}} \overset{x}{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}} = \overset{y}{\begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{bmatrix}}$$

$$\underbrace{\frac{1}{w_{11}w_{22} - w_{12}w_{21}} \begin{bmatrix} w_{22} & -w_{12} \\ -w_{21} & w_{11} \end{bmatrix}}_{W^{-1}} \underbrace{\begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \end{bmatrix}}_{y} = \begin{bmatrix} \dfrac{w_{22}(w_{11}x_1 + \cancel{w_{12}x_2}) - w_{12}(w_{21}x_1 + \cancel{w_{22}x_2})}{w_{11}w_{22} - w_{12}w_{21}} \\ \cdots \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

## 4) Squeeze / Downsampling

- Trades spatial dimensions for channel dimensions
- Used w/ split operator + multi-scale architectures (explained next)




## 5) Split

- Splits half of the features to the end of the flow
- Other half is further processed
- Helps speed up the flow

### Overall Architecture