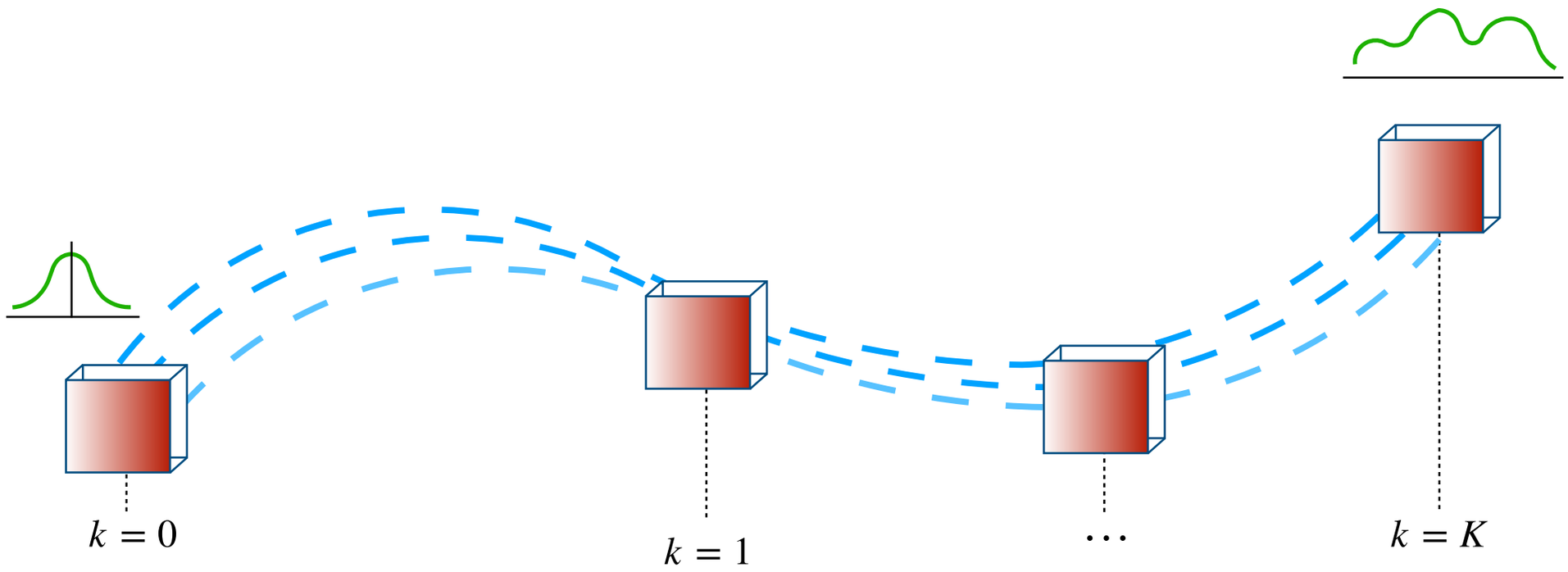
Normalizing Flows

CS6202

Abdul Fatir Ansari

Devamanyu Hazarika

Remmy Zen



Contents

Introduction

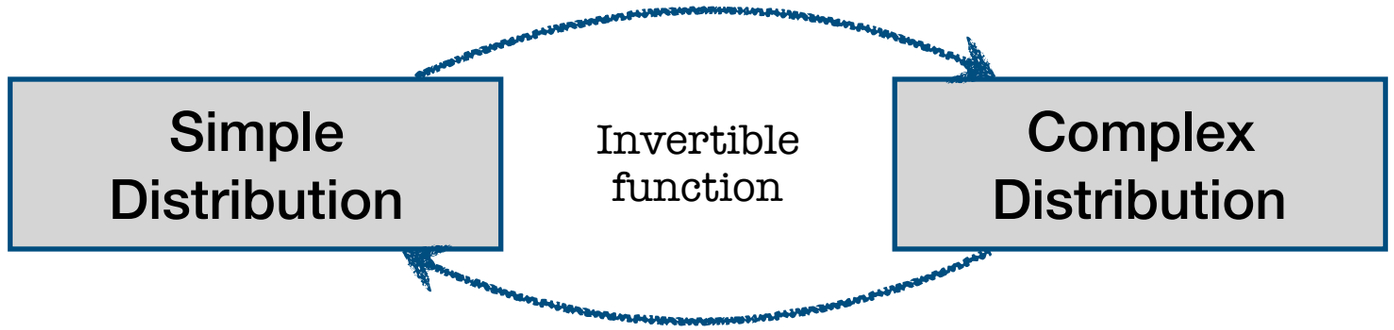
Challenges in representation learning

- Data of interest are high-dimensional and highly-structured.
- Powerful models required to capture data complexity.

Present models assume simple families for latent posterior or data likelihood ⟶ far off from true distributions.

Goal: Define flexible, arbitrarily complex, and scalable distributions.

Normalizing Flows!



Background

Autoencoder

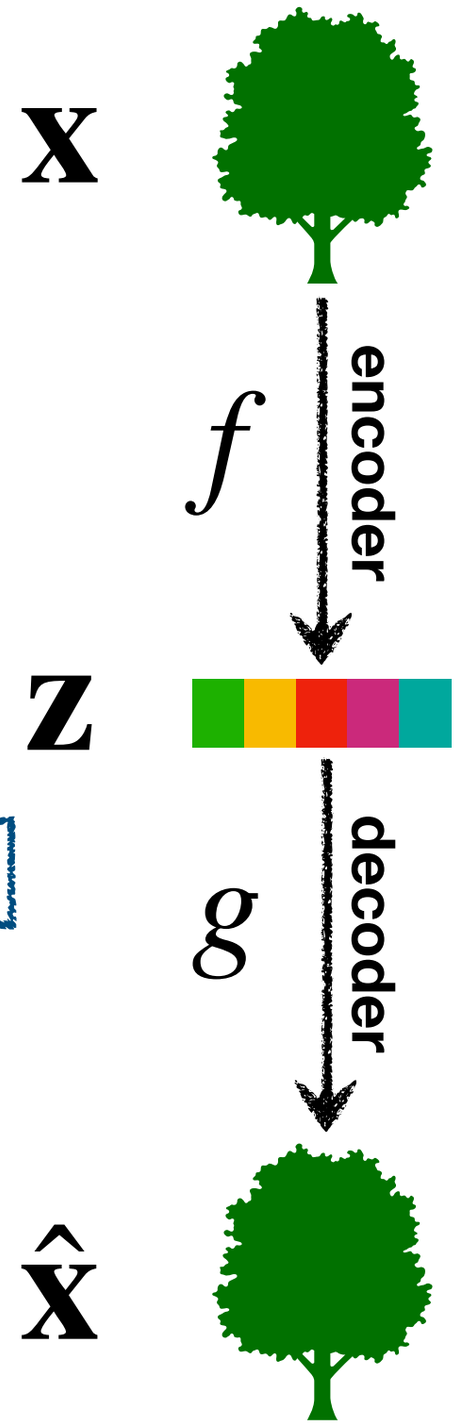
- Autoencoder is made of an **encoder** and a **decoder**.

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^p$$

$$g : \mathbb{R}^p \rightarrow \mathbb{R}^d$$

$$\mathbf{x} \in \mathbb{R}^d$$
$$\mathbf{z} \in \mathbb{R}^p$$
$$\hat{\mathbf{x}} \in \mathbb{R}^d$$

$$f, g = \underset{f,g}{\operatorname{argmin}} \mathscr{L}\left(\mathbf{x}, \hat{\mathbf{x}}\right)$$



- $\mathscr{L}$ is the reconstruction error which can be **squared-error** if we assume Gaussian likelihood or **cross-entropy** if we assume Bernoulli likelihood.

$$||\mathbf{x} - \hat{\mathbf{x}}||_2$$

$$-\sum_{j=1}^{d} x_j \log \hat{x}_j + (1 - x_j)\log(1 - \hat{x}_j)$$

Variational Autoencoder

- Suppose we want to learn a generative model.

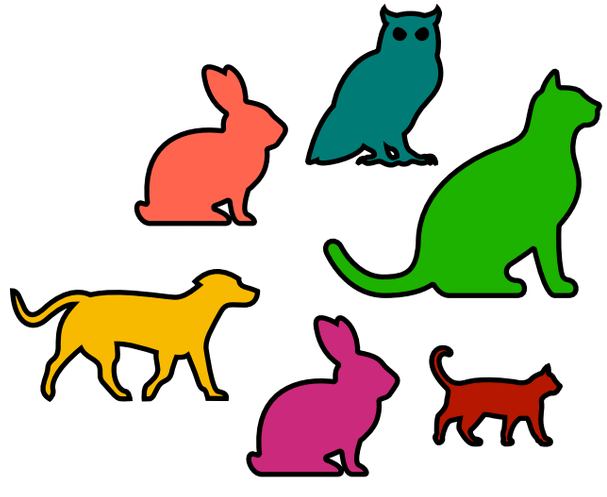
- Just do $\underset{\theta}{\text{argmax}}\, \mathbb{E}_{p_{\text{data}}} \left[\log p_\theta(\mathbf{x})\right]$ ✘

- Further specify the distribution

$$\log p_\theta(\mathbf{x}) = \log \int_{\mathbf{z}} p_\theta(\mathbf{x}\,|\,\mathbf{z}) p(\mathbf{z}) d\mathbf{z}$$ ✘

- We also want to to perform inference; i.e., $p(\mathbf{z}\,|\,\mathbf{x})$ ✘

- Specify a simpler family $q(\mathbf{z}\,|\,\mathbf{x})$ and perform variational inference. ✓



Complex!

$$\log p(\mathbf{x}) = \log \int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) \frac{q(\mathbf{z}\,|\,\mathbf{x})}{q(\mathbf{z}\,|\,\mathbf{x})} d\mathbf{z}$$

$$= \log \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{x})}\right]$$

$$\geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{x})}\right] = \mathscr{L}_{ELBO}$$

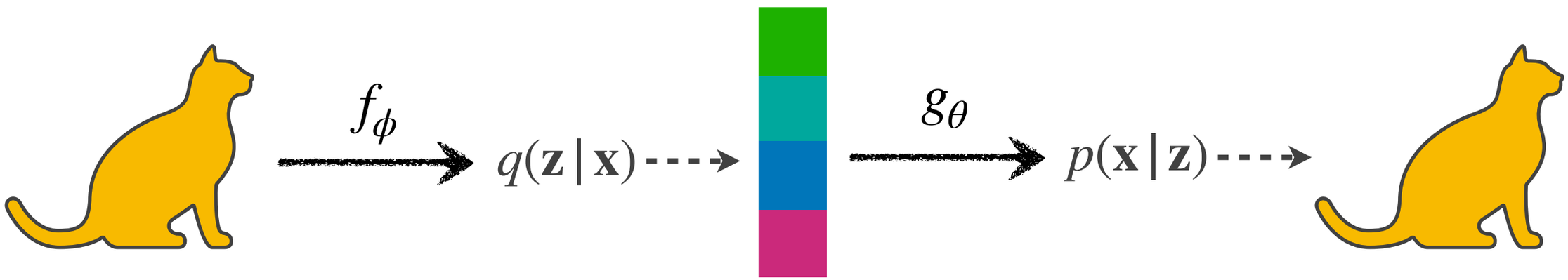
By Jensen's Inequality:

$\varphi(\mathbb{E}[X]) \geq \mathbb{E}[\varphi(X)]$ **if** φ **is a concave function**

Kingma, D.P. and Welling, M., Auto-encoding variational bayes. arXiv 2014.

Variational Autoencoder

- Neural networks f_ϕ and g_θ are used to model the variational posterior and the likelihood respectively.
- Training proceeds by minimizing $-\mathscr{L}_{ELBO}$ using SGD.



- SGD requires $\nabla \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\right]$ which can be computed efficiently using the **reparameterization-trick**.

$$-\mathscr{L}_{ELBO} = -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right] + \mathbb{D}_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

reconstruction error **distance from prior**

Normalizing Flows

Transformation of a Random Variable

- Let X be a random variable with the density function $p(x)$.
- Let $Y = f(X)$

★ **Question**: What is the density function of Y?

- It is possible to compute the analytic density of Y if f is **continuous**, **differentiable**, and **invertible** in the support of X.

✤ **Example**:

$x \in [0,1]$

$p(x) = 2x$

$$f(x) = e^x$$

continuous; differentiable; invertible

Example

- **Example**: $x \in [0,1] \quad p(x) = 2x \quad f(x) = e^x$

- Cumulative Distribution Function (CDF)

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} p(x)dx$$

$$\begin{aligned} F_Y(y) &= P(Y \leq y) \\ &= P(e^X \leq y) \\ &= P(X \leq \log y) \\ &= F_X(\log y) \end{aligned}$$

- Now, $p(y) = F'_Y(y) = \frac{d}{dy}\left(F_X\left(\log y\right)\right)$

$$\begin{aligned} F_X(\log y) &= \int_0^{\log y} 2x dx \\ &= \left[2\frac{x^2}{2}\right]_0^{\log y} \\ &= (\log y)^2 \end{aligned}$$

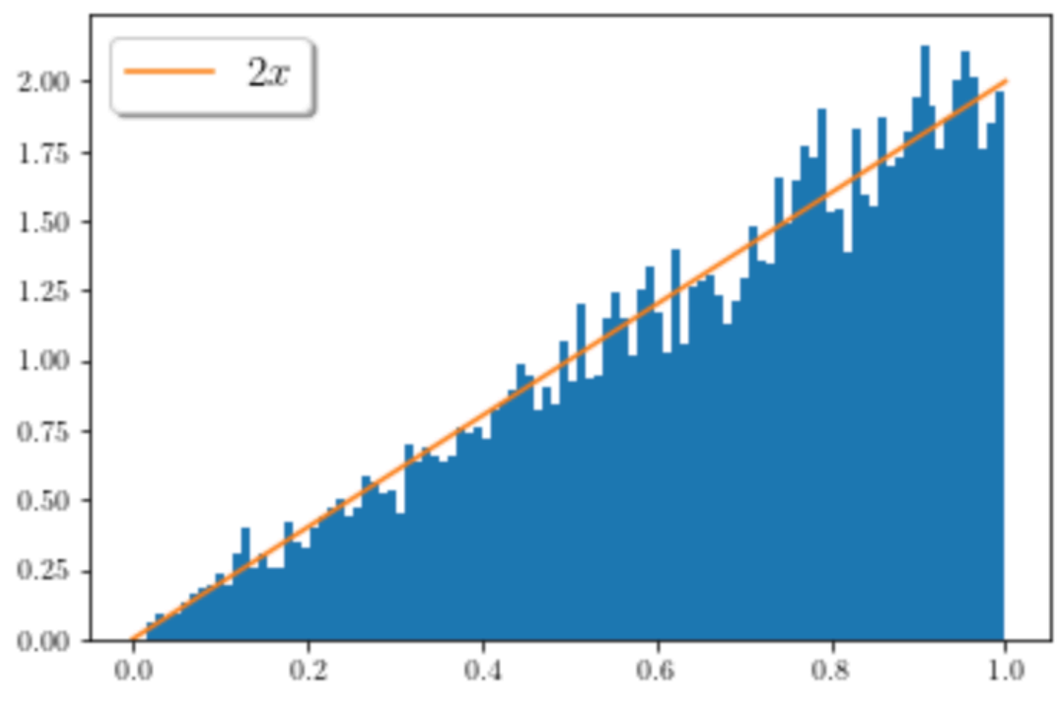
$$p(y) = 2\frac{\log(y)}{y}$$

Example

- Sample from $p(x) = 2x$ using any sampling method such as **inverse-transform sampling**.

- Plot the histogram of drawn samples.

```
y = np.random.uniform(0, 1, 10000)
x = np.sqrt(y)
plt.hist(x, density=1, bins=100)
```

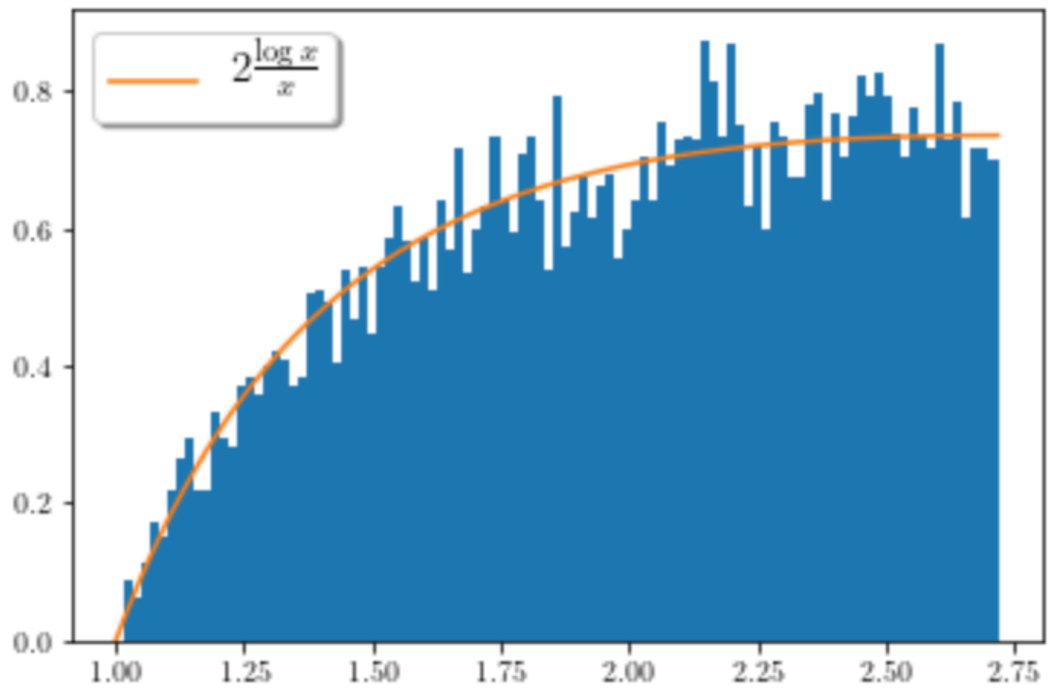


Example

- Plot the analytically obtained density function $p(y) = 2\frac{\log(y)}{y}$

- Apply $f(x) = e^x$ to samples from $p(x)$ and plot the histogram.

```
xvals = np.linspace(1, np.exp(1), 1000)
yvals = 2 * np.log(xvals)/xvals
plt.hist(np.exp(x), density=1, bins=100)
```



It Works!

Multivariate Distributions

- This idea can be extended to multivariate distributions.

- Let $q_0(\mathbf{z})$ be a multivariate density function and f an invertible function.

$$f : \mathbb{R}^d \to \mathbb{R}^d \qquad \mathbf{y} = f(\mathbf{z})$$

- The PDF of $\mathbf{y}$ is then given by

Jacobian determinant

$$q_1(\mathbf{y}) = q_0(\mathbf{z}) \left| \det \frac{\partial f^{-1}}{\partial \mathbf{y}} \right| = q_0(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1}$$

expensive to compute

- Such transformations are called **normalizing flows**.

Applications

Applications of Normalizing Flows

- **Variational Inference**
  - Planar and Radial Flow *[Rezende and Mohamed, 2015]*
  - Inverse Autoregressive Flow (IAF) *[Kingma et al., 2016]*

- **Density Estimation**
  - Non-linear Independent Components Estimation (NICE) *[Dinh et al., 2014]*
  - Real-valued Non-Volume Preserving (RealNVP) *[Dinh et al., 2017]*
  - Masked Autoregressive Flow (MAF) *[Papamakarios et al., 2017]*

Normalizing Flows for Variational Inference

- The variational posterior in VAEs is generally chosen to be a **normal** distribution with **diagonal covariance** matrix.

- This simplistic assumption may be far off from the true posterior. It cannot have multiple modes!

- Normalizing flows can be used to modify the simple normal distribution into a **richer posterior** that can better model the true posterior.

Flow-based Bound

- Let's apply a length **K** flow, i.e., $\mathbf{z}_K = f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{z}_0)$

- $\mathbf{z}_0 \sim q_0(\mathbf{z})$ which is generally a simpler distribution such as the multivariate normal distribution $\mathcal{N}(\mu, \sigma^2\mathbf{I})$

$$q_K(\mathbf{z}_K) = q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|^{-1}$$

Recall

$$q_1(\mathbf{z}_1) = q_0(\mathbf{z}_0) \left| \det \frac{\partial f}{\partial \mathbf{z}_0} \right|^{-1}$$

- Let the variational distribution in VAE now be $q(\mathbf{z}\,|\,\mathbf{x}) := q_K(\mathbf{z}_K)$

$$\begin{aligned}
\mathscr{L} &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}\,|\,\mathbf{x}) \right] \\
&= \mathbb{E}_{q_0(\mathbf{z_0})} \left[\log p(\mathbf{x}, \mathbf{z}_K) - \log q_K(\mathbf{z}_K) \right] \\
&= \mathbb{E}_{q_0(\mathbf{z_0})} \left[\log p(\mathbf{x}, \mathbf{z}_K) - \log q_0(\mathbf{z}_0) + \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right| \right]
\end{aligned}$$

Planar Flow

- Applies transformations of type $f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$ where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$ and h is an element-wise non-linearity.

- The Jacobian is then given by $\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = \mathbf{I} + \mathbf{u}h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}^\top$ and the Jacobian determinant by

$$\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = (1 + h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}^\top \mathbf{I}^{-1}\mathbf{u}) \det(\mathbf{I})$$
$$= (1 + h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}^\top \mathbf{u})$$

using Matrix determinant lemma

$$\det\left(\mathbf{A} + \mathbf{u}\mathbf{v}^\top\right) = \left(1 + \mathbf{v}^\top \mathbf{A}^{-1}\mathbf{u}\right) \det\left(\mathbf{A}\right)$$

Rezende, DJ, and Mohamed, S, "Variational inference with normalizing flows." ICML 2015.

Planar Flow: Example

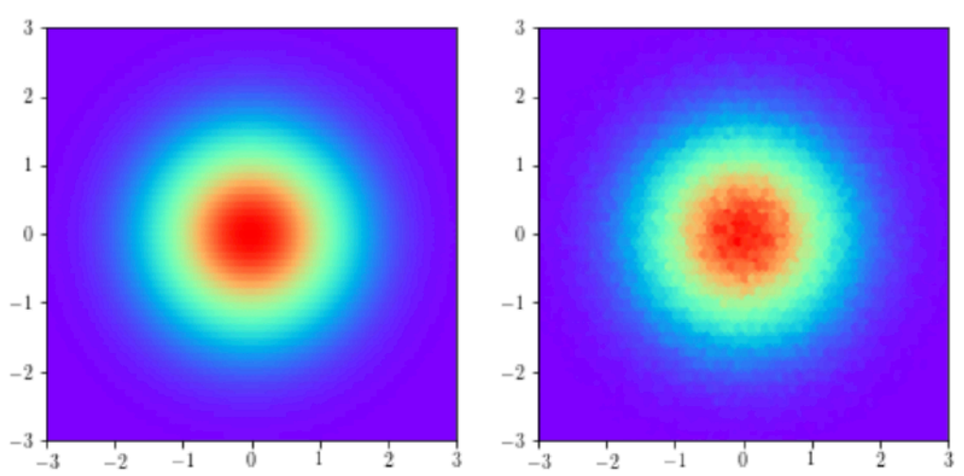
- For $\mathbf{z} \in \mathbb{R}^2$ and $q_0(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ let's apply the following Planar Flow

$$\mathbf{w} = [5, 0]^\top$$

$$\mathbf{u} = [1, 0]^\top$$

$$b = 0$$

$$h(\mathbf{x}) = \tanh(\mathbf{x})$$



Analytic and empirical densities $q_0(\mathbf{z})$

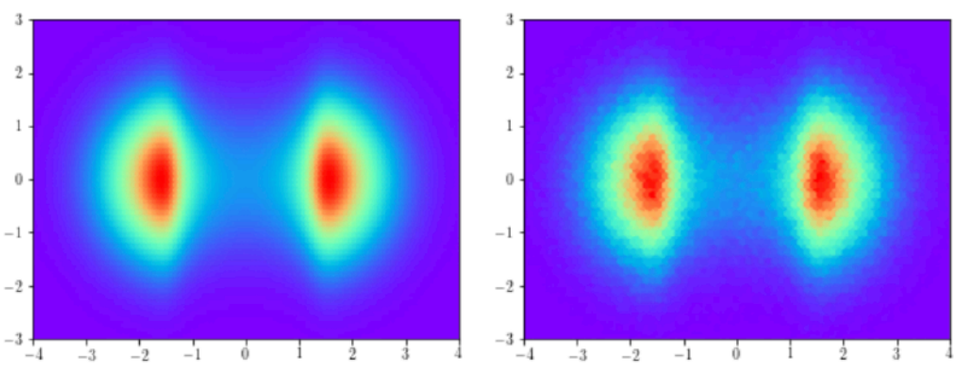
```
w = np.array([5., 0])
u = np.array([1., 0])
b = 0

def h(x):
    return np.tanh(x)

def h_prime(x):
    return 1 - np.tanh(x) ** 2

def f(z):
    y = z + np.dot(h(np.dot(z, w) + b).reshape(-1,1), u.reshape(1,-1))
    return y

def det_J(z):
    psi = h_prime(np.dot(z, w) + b).reshape(-1,1) * w
    det = np.abs(1 + np.dot(psi, u.reshape(-1,1)))
    return det
```



Analytic and empirical densities $q_1(\mathbf{y})$

Radial Flow

- Applies transformations of type $f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_0)$ where $\alpha \in \mathbb{R}^+, \beta \in \mathbb{R}, r = \|\mathbf{z} - \mathbf{z}_0\|, h(\alpha, r) = (\alpha + r)^{-1}$

- The Jacobian is then given by

$$\begin{aligned}\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} &= \mathbf{I} + \beta \left((\mathbf{z} - \mathbf{z}_0) h'(\alpha, r) \frac{\partial r}{\partial \mathbf{z}} + h(\alpha, r)\mathbf{I} \right) \\ &= (1 + \beta h(\alpha, r))\mathbf{I} + \beta h'(\alpha, r)(\mathbf{z} - \mathbf{z}_0) \frac{(\mathbf{z} - \mathbf{z}_0)^\top}{\|\mathbf{z} - \mathbf{z}_0\|}\end{aligned}$$

- And the Jacobian determinant by

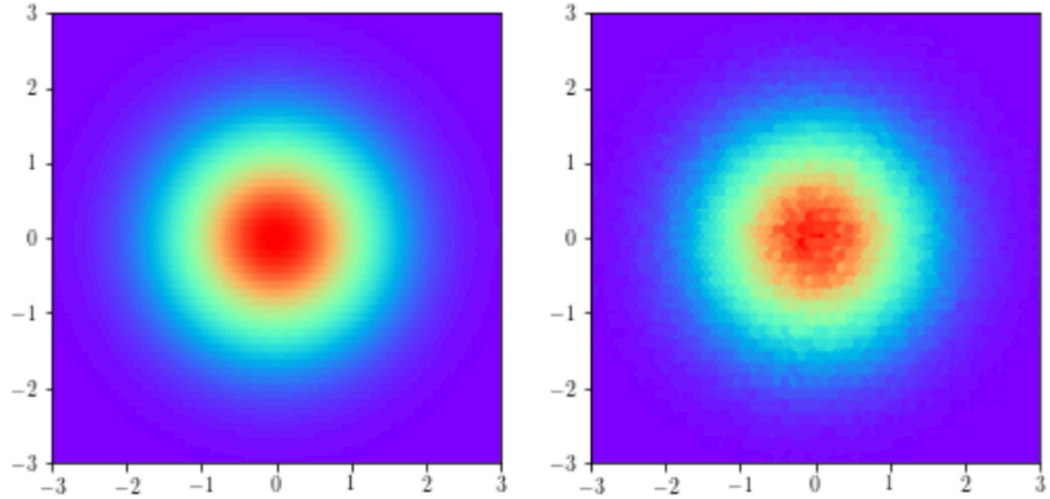
$$\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = \Big(1 + \beta h(\alpha, r) + \beta h'(\alpha, r) r\Big)(1 + \beta h(\alpha, r))^{d-1}$$

Rezende, DJ, and Mohamed, S, "Variational inference with normalizing flows." ICML 2015.

Radial Flow: Example

- For $\mathbf{z} \in \mathbb{R}^2$ and $q_0(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ let's apply the following Radial Flow

$$\mathbf{z}_0 = [1, 0]^\top$$
$$\alpha = 2$$
$$\beta = 5$$



Analytic and empirical densities $q_0(\mathbf{z})$

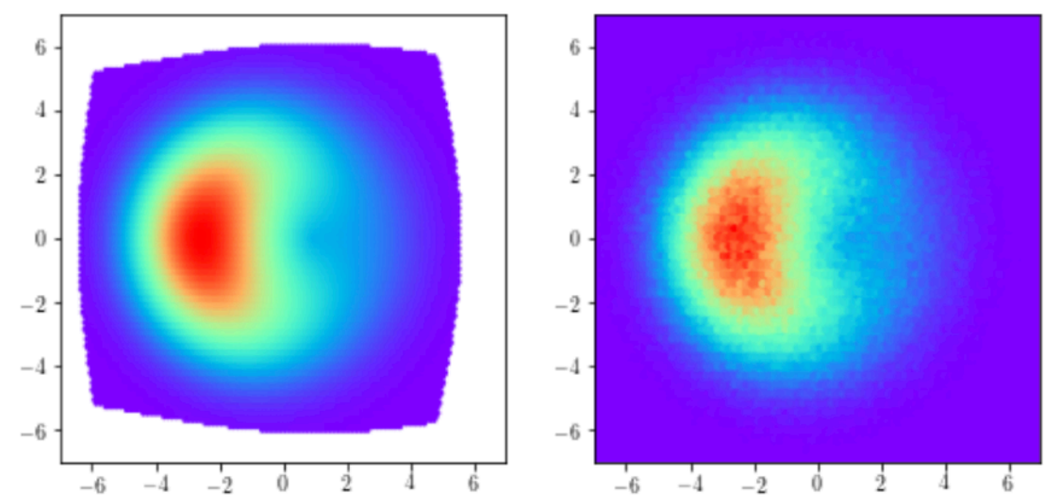
```
z0 = np.array([1, 0])
a = 2
b = 5

def h(r):
    return 1/(a+r)

def h_prime(r):
    return -1/(a+r) ** 2

def f(z):
    r = LA.norm(z - z0, axis=1).reshape(-1, 1)
    y = z + b * h(r) * (z - z0)
    return y

def det_J(z):
    n_dims = z.shape[1]
    r = LA.norm(z - z0, axis=1).reshape(-1, 1)
    tmp = 1 + b * h(r)
    det = (tmp + b * h_prime(r) * r) * tmp ** (n_dims - 1)
    return det
```



Analytic and empirical densities $q_1(\mathbf{y})$

Autoregressive Flow

- **Autoregressive Transform:** Given a sequence of variables, each variable is only dependent on the previously indexed variables.

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i \,|\, x_1, \ldots, x_{i-1}) = \prod_{i=1}^{D} p(x_i \,|\, x_{1:i-1})$$

- **(Gaussian) Autoregressive Flow:** sequentially apply an autoregressive transform on a noise vector $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

$$y_0 = \mu_0 + \sigma_0 \odot \epsilon_0$$

$$y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1}) \cdot \epsilon_i$$

- **Advantage:** Scales well to high-dimensional space than planar and radial flow.
- **Invertible? Jacobian?**

Germain, Mathieu, et al. "Made: Masked autoencoder for distribution estimation." International Conference on Machine Learning. 2015.
Kingma, Diederik P., et al. "Improved variational inference with inverse autoregressive flow." NIPS 2016.

Inverse Autoregressive Flow (IAF)

- Calculating Inverse Autoregressive Flow (individual elements of ϵ_i) **do not depend on each other** and **can be vectorized**.

$$\epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{1:i-1})}{\sigma_i(\mathbf{y}_{1:i-1})} \qquad \epsilon = (\mathbf{y} - \mu(\mathbf{y}))/\sigma(\mathbf{y})$$

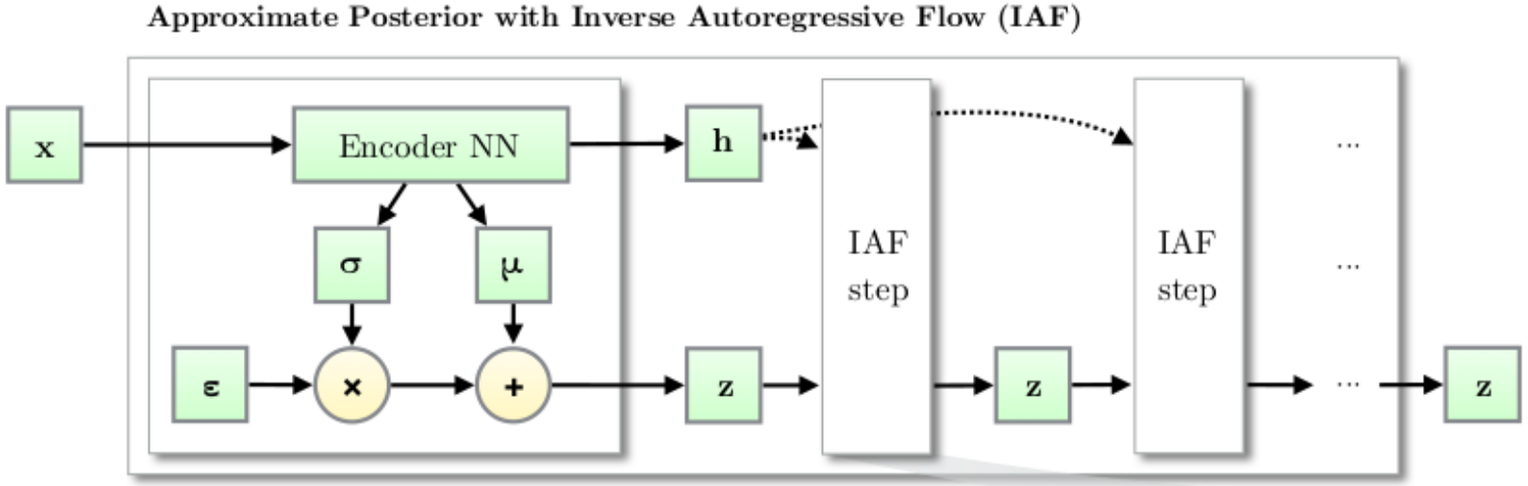
- Jacobian of Inverse Autoregressive Flow is a lower triangular matrix ⟶ **determinant easy to compute!**

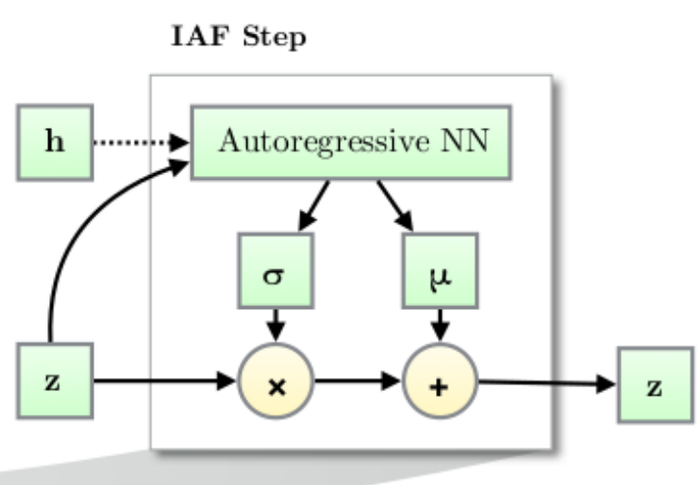
$$\log\det\left|\frac{d\epsilon}{d\mathbf{y}}\right| = \sum_{i=1}^{D} -\log\sigma_i(\mathbf{y})$$

Kingma, Diederik P., et al. "Improved variational inference with inverse autoregressive flow." NIPS 2016.

IAF for Variational Inference

- **Adding IAF to VAE**
  1. Adding IAF transforms after latent variables **z**
  2. Modify the likelihood to account for IAF transforms





$$\mathbf{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \odot \boldsymbol{\epsilon}$$

$$\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1}$$

$$[\mathbf{m}_t, \mathbf{s}_t] \leftarrow \texttt{AutoregressiveNN}[t](\mathbf{z}_t, \mathbf{h}; \boldsymbol{\theta})$$

$$\sigma_t = \text{sigmoid}(\mathbf{s}_t)$$

$$\mathbf{z}_t = \sigma_t \odot \mathbf{z}_{t-1} + (1 - \sigma_t) \odot \mathbf{m}_t$$

$$\mathbf{s}_t = \frac{1}{\sigma_t} \quad \text{and} \quad \mathbf{m}_t = -\frac{\mu_t}{\sigma_t}$$

Applications of Normalizing Flows

- **Variational Inference**
  - Planar and Radial Flow *[Rezende and Mohamed, 2015]*
  - Inverse Autoregressive Flow (IAF) *[Kingma et al., 2016]*

- **Density Estimation**
  - Non-linear Independent Components Estimation (NICE) *[Dinh et al., 2014]*
  - Real-valued Non-Volume Preserving (RealNVP) *[Dinh et al., 2017]*
  - Masked Autoregressive Flow (MAF) *[Papamakarios et al., 2017]*

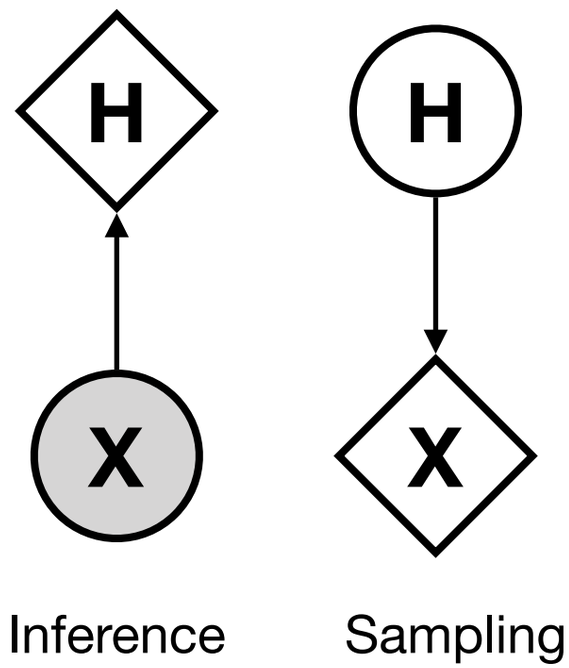
Normalizing Flows for Density Estimation

- A good representation is one in which the distribution of the data is easy to model.

- Transform data density into a simpler distribution, ideally factorized.

- Estimate *maximum likelihood* of data by learning a transformation function **h** = f(**x**) :

$$\log(p_X(\mathbf{x})) = \log p_H(\mathbf{f}(\mathbf{x})) + \log \left| \det \frac{\partial f(x)}{\partial \mathbf{x}} \right|$$



NICE

Non-linear Independent Components Estimation

Maximum likelihood estimation with **factorial** prior distribution

$$\log(\, p_X(\mathbf{x})\,) = \sum_{d=1}^{D} \log\, p_{H_d}(\mathbf{f_d}(\mathbf{x})) \;+ \log \left| \det \frac{\partial f(x)}{\partial \mathbf{x}} \right|$$

Family of functions with triangular Jacobians : **Coupling** layer

Dinh, L., Krueger, D. and Bengio, Y., NICE: Non-linear independent components estimation, ArXiv 2014.

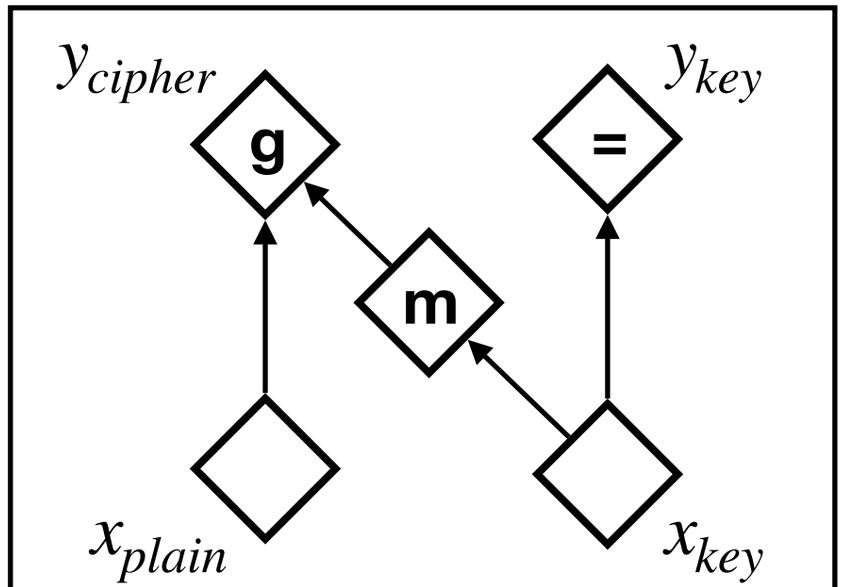
COUPLING Layer

Let

$x \in \mathcal{X}, I_1, I_2$: **partition of** $[1,D]$ **and** $d = |I_1|$

Define

$$y = (y_{I_1}, y_{I_2}): \quad y_{I_1} = x_{I_1}$$
$$y_{I_2} = g(x_{I_2}; m(x_{I_1}))$$



Computational graph of a coupling layer.

where $g : \mathbb{R}^{D-d} \times m(\mathbb{R}^d) \to \mathbb{R}^{D-d}$ is an invertible map

Jacobian: $$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{I_2}}{\partial x_{I_1}} & \frac{\partial y_{I_2}}{\partial x_{I_2}} \end{bmatrix}$$

Here, $$\det \frac{\partial y}{\partial x} = \det \frac{\partial y_{I_2}}{\partial x_{I_2}}$$

Additive COUPLING Layer

Choices of coupling:

- Additive : $g(a;b) = a + b$
- Multiplicative : $= a \circ b$
- Affine : $= a \circ b_1 + b_2$

NICE chooses additive coupling

Transformation	Inverse
$y_{I_1} = x_{I_1}$	$x_{I_1} = y_{I_1}$
$y_{I_2} = x_{I_2} + m(x_{I_1})$	$x_{I_2} = y_{I_2} - m(y_{I_1})$

Stack coupling layers by alternating roles for x_{key} and x_{plain}.

$m()$: reLU activation

Prior Distribution

- Factorial prior is chosen
- Gaussian:

$$\log(p_{H_d}) = -\frac{1}{2}(h_d^2 + \log(2\pi))$$

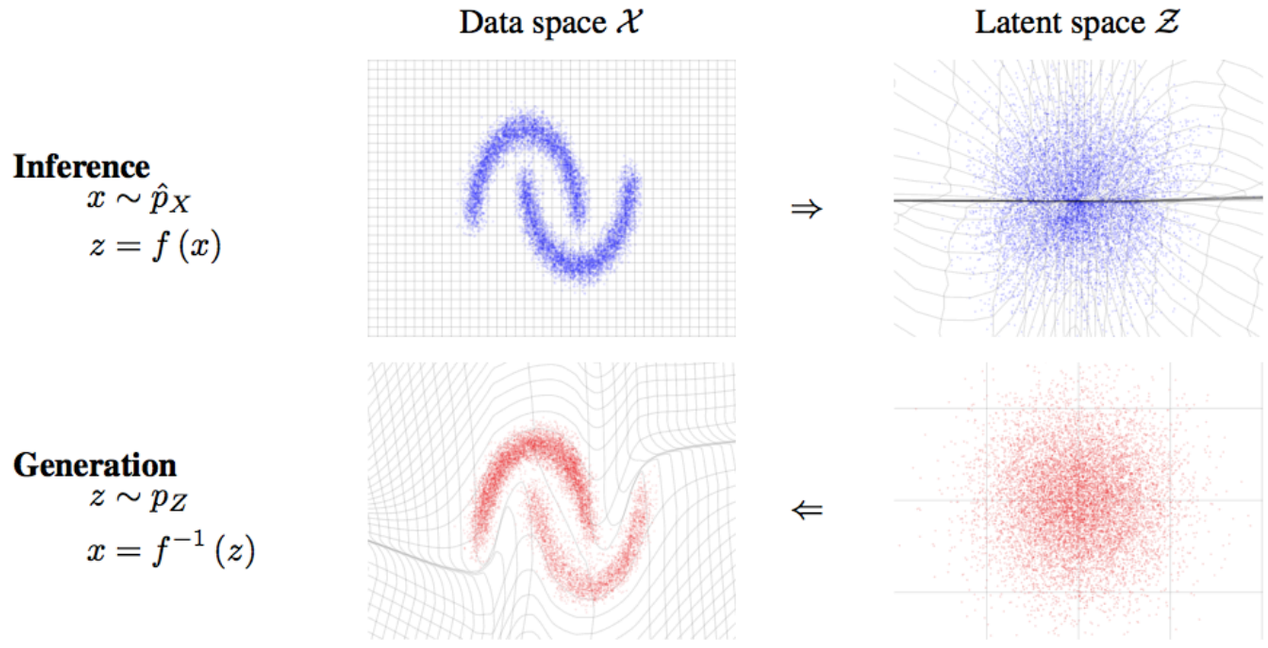
- Logistic:

$$\log(p_{H_d}) = -\log(1 + \exp(h_d)) - \log(1 - \exp(h_d))$$

RealNVP

Density Estimation using real-valued non-volume preserving transformations

- Extension of NICE
- Exact and tractable likelihood and inference
- No reconstruction error in cost function => sharper images.



Dinh, L., Sohl-Dickstein, J. and Bengio, Density estimation using Real NVP. ICLR 2017.

Affine Coupling Layers

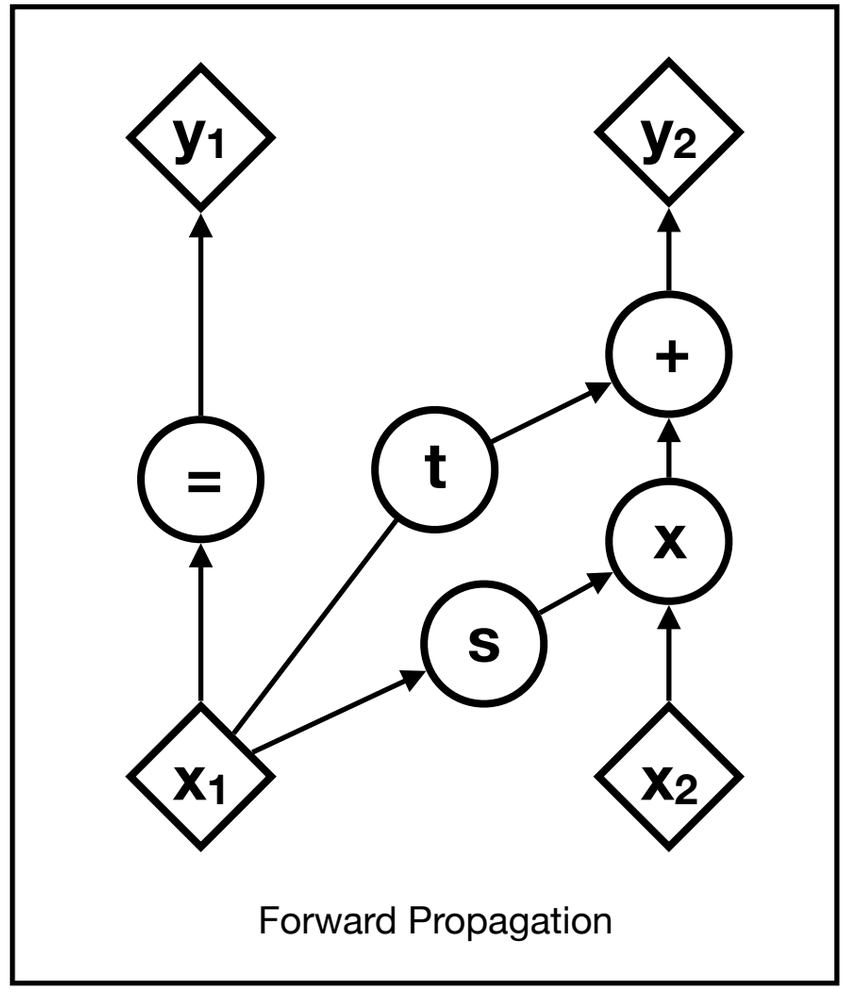
- Affine Transformation:

$$y_{1:d} = x_{1:d}$$

$$y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}))$$

Scale

Shift



- Inverse:

$$x_{1:d} = y_{1:d}$$

$$x_{d+1:D} = (y_{d+1:D} - t(y_{1:d})) \odot \exp(-s(y_{1:d})))$$

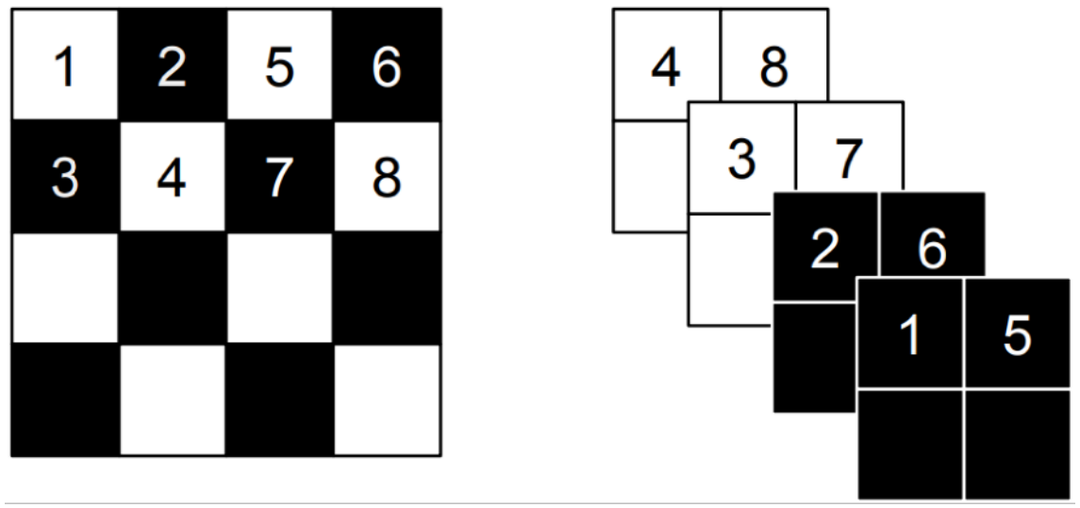
  - s(.) and t(.) can be arbitrarily complex, e.g. Neural Networks.
  - Inverse of s(.) and t(.) is not required !

Masked Convolution

- Partitioning through binary masks:

$$y = b \odot x + (1 - b) \odot \left(x \odot \exp(s(b \odot x)) + t(b \odot x)\right)$$

- Exploit local correlations of image:



- Also propose an invertible batch-normalization variant to facilitate deep networks.

Key Advantages

- **Variational Inference**
  - Richer posteriors — can better approximate the true posterior which may be multi-modal.
- **Density Estimation**
  - Tractable and exact log-likelihood evaluation like auto-regressive models — much more flexible.
  - Unlike VAEs, no explicit likelihood function is used — sharper images.

Normalizing Flows in PPLs

NF in TFP

- Both Pyro (based on PyTorch) and Tensorflow Probability (TFP) (based on Tensorflow) provide a construct called `Bijector` (from bijection, i.e., a one-to-one and onto function) for implementing Normalizing Flows.

- TFP also provides `TransformedDistribution` which takes in **a base distribution** and **a bijector** to represent the distribution obtained after the transformation.

- TFP implements a number of bijectors:

`Affine` `Exp` `Sigmoid` `RealNVP` `Gumbel` ...

The Bijector

- Implementing the `Bijector` class in TFP requires implementation of the following functions

  ★ `_forward`: To implement the forward function. $y = f(x)$

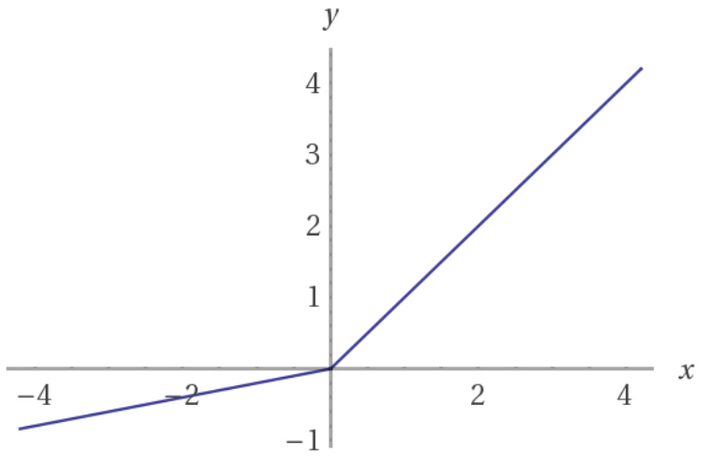
  ★ `_inverse`: To implement the inverse function. $x = f^{-1}(y)$

  ★ `_inverse_log_det_jacobian`: To implement the inverse log-determinant of the Jacobian. $\log\det\left|\frac{\partial f^{-1}}{\partial y}\right|$

The Bijector: Example

- Let's implement a `Bijector` for the Parameterized ReLU (PReLU) function

$$y = f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases} \quad \alpha \in [0,1]$$



- The inverse function is given by

$$x = f^{-1}(y) = \begin{cases} y & y \geq 0 \\ \frac{y}{\alpha} & y < 0 \end{cases}$$

- The Jacobian is diagonal and each element is given by

$$\left.\frac{\partial f}{\partial x}\right|_{ii} = \begin{cases} 1 & x_i \geq 0 \\ \alpha & x_i < 0 \end{cases} \quad \text{or} \quad \left.\frac{\partial f^{-1}}{\partial y}\right|_{ii} = \begin{cases} 1 & y_i \geq 0 \\ \frac{1}{\alpha} & y_i < 0 \end{cases}$$

The Bijector: Example

```
class PReLU(tfb.Bijector):
    def __init__(self, alpha=0.5, validate_args=False, name="p_relu"):
        super(PReLU, self).__init__(
            forward_min_event_ndims=0,
            validate_args=validate_args,
            name=name)
        self.alpha = alpha

    def _forward(self, x):
        return tf.where(tf.greater_equal(x, 0), x, self.alpha * x)

    def _inverse(self, y):
        return tf.where(tf.greater_equal(y, 0), y, 1. / self.alpha * y)

    def _inverse_log_det_jacobian(self, y):
        I = tf.ones_like(y)
        J_inv = tf.where(tf.greater_equal(y, 0), I, 1.0 / self.alpha * I)
        log_abs_det_J_inv = tf.log(tf.abs(J_inv))
        return log_abs_det_J_inv
```

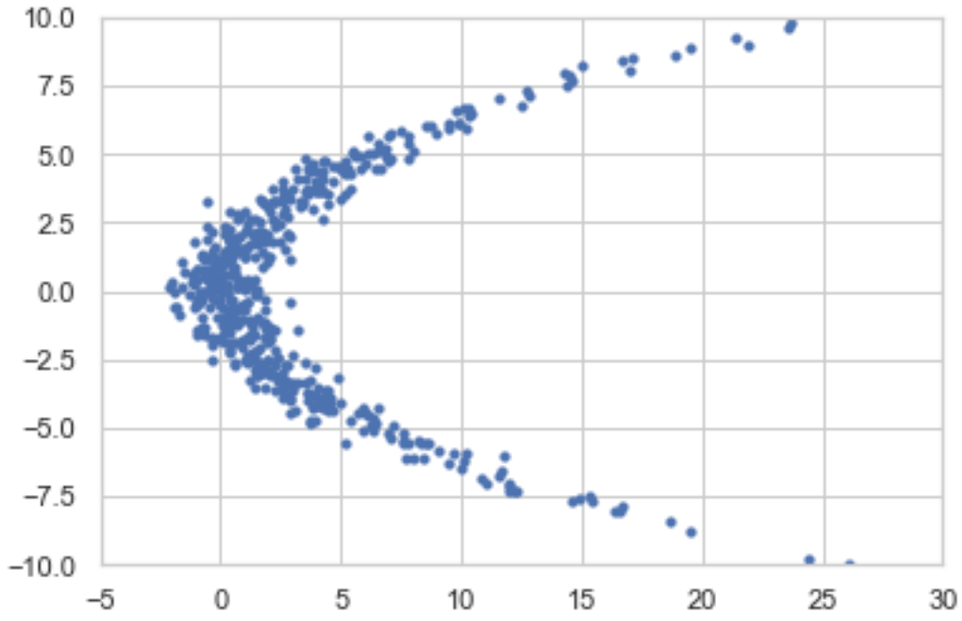
Example

Adapted from Eric Jang's blog post

- Let's try to estimate the following 2D probability distribution using TFP with affine transformation and PReLU function

$$x_2 \sim \mathcal{N}(0, 4^2)$$

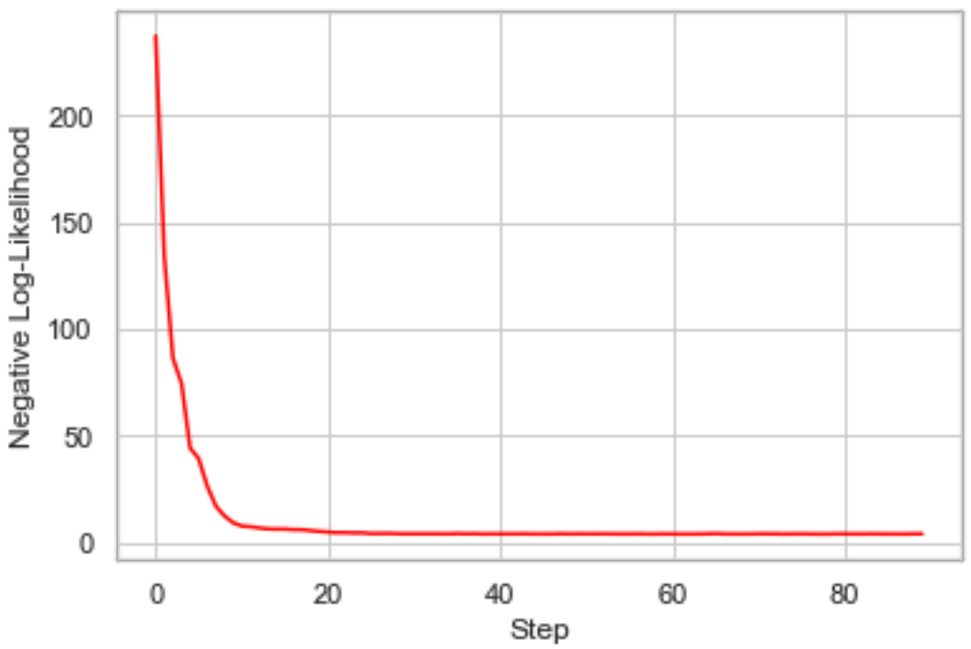
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad x_1 \sim \mathcal{N}\left(\frac{x_2^2}{4}, 1^2\right)$$



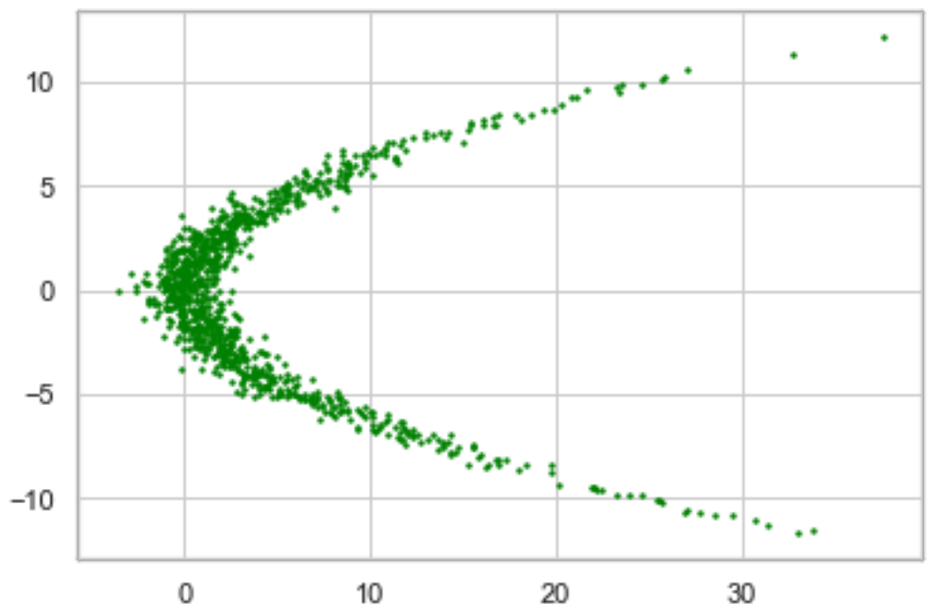

```
base_dist = tfd.MultivariateNormalDiag(loc=tf.zeros([2], DTYPE))
d, r = 2, 2
bijectors = []
num_layers = 6
for i in range(num_layers):
    with tf.variable_scope('bijector_%d' % i):
        V = tf.get_variable('V', [d, r], dtype=DTYPE)
        shift = tf.get_variable('shift', [d], dtype=DTYPE)
        L = tf.get_variable('L', [d*(d+1)/2], dtype=DTYPE)
        bijectors.append(tfb.Affine(
            scale_tril=tfd.fill_triangular(L),
            scale_perturb_factor=V,
            shift=shift,
        ))
        alpha = tf.abs(tf.get_variable('alpha', [], dtype=DTYPE))+.01
        bijectors.append(PReLU(alpha=alpha))
```

Example

```
mlp_bijector = tfb.Chain(list(reversed(bijectors[:-1])),
                         name='2d_mlp_bijector')
dist = tfd.TransformedDistribution(
    distribution=base_dist,
    bijector=mlp_bijector
)
loss = -tf.reduce_mean(dist.log_prob(x_samples))
train_op = tf.train.AdamOptimizer(1e-3).minimize(loss)
```



Training Curve

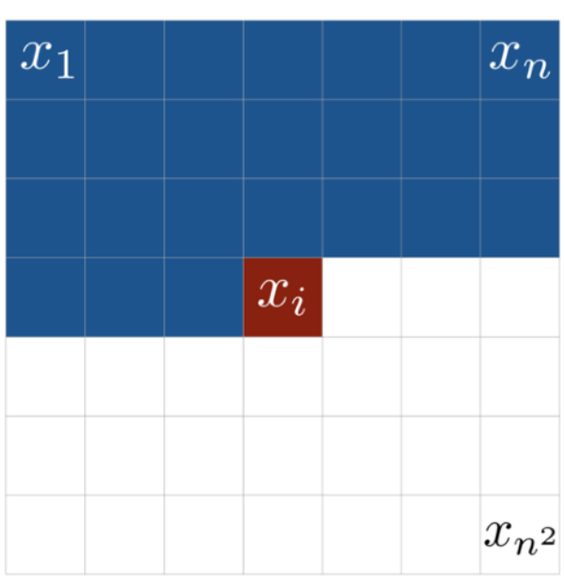


Random Samples

Other Recent Advances

PixelRNN

- **Idea:** Use RNN to generate images one pixel at a time.
- Each pixel is sampled conditioned on pixels that have been generated ⟶ **autoregressive flows!**



Context

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

likelihood of image **x**

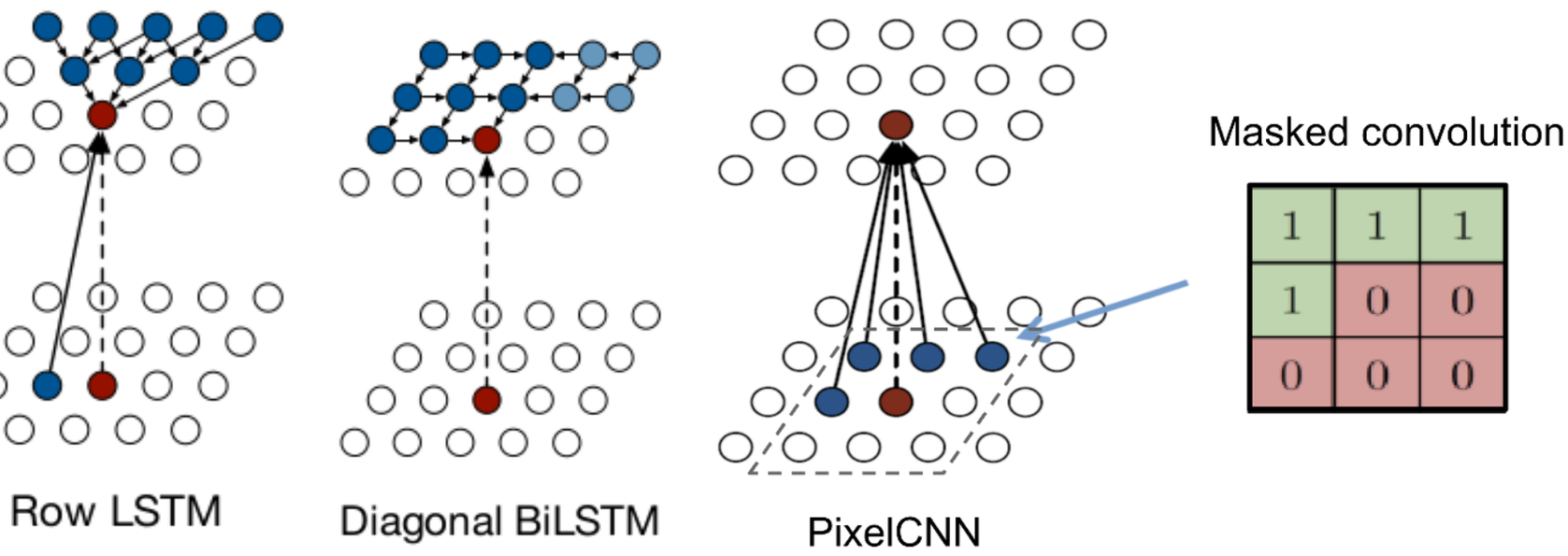
probability of i-th pixel given previous pixels

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

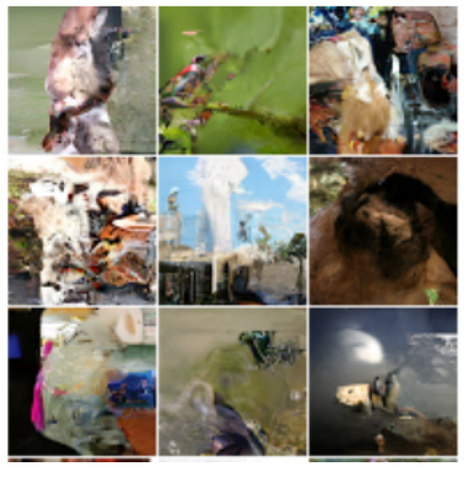
Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks." ICML 2016.

PixelRNN

- **Methods:**



- **Application:**



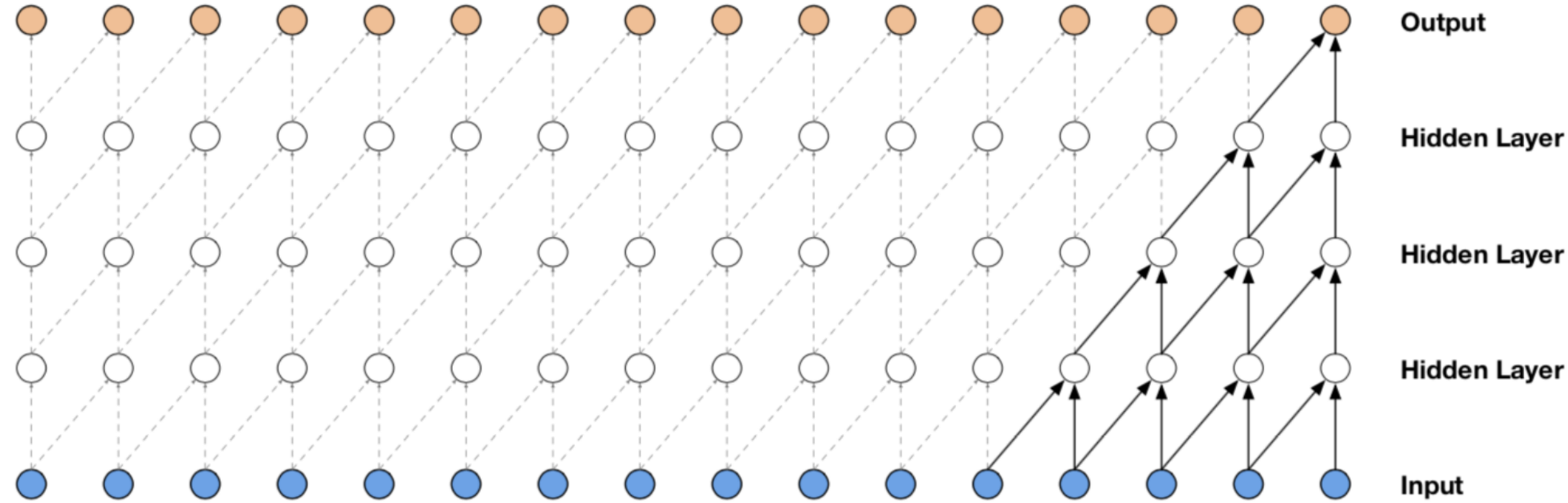
Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks." ICML 2016.

WaveNet

- **Idea:** Similar to PixelCNN but for 1D. Causal / masked convolution applied to 1D signal to generate audio.

- Convolution prediction at a certain timestamp can only **consume the data observed in the past** $\longrightarrow$ **autoregressive flows!**

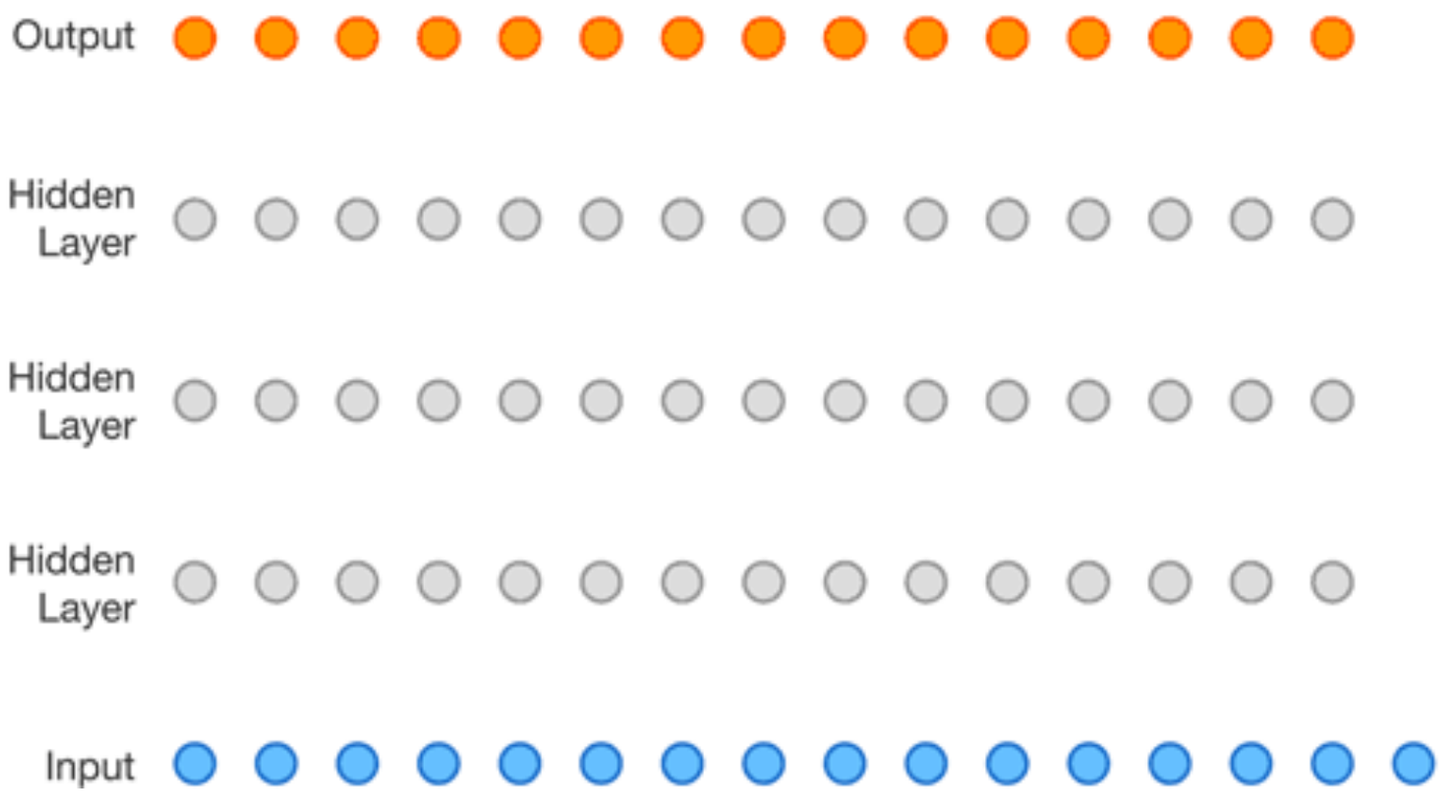
Oord, Aaron van den, et al. "Wavenet: A generative model for raw audio." arXiv 2016.



Oord, Aaron van den, et al. "Wavenet: A generative model for raw audio." arXiv 2016.

WaveNet

- **Causal Convolutional Layers Visualization**

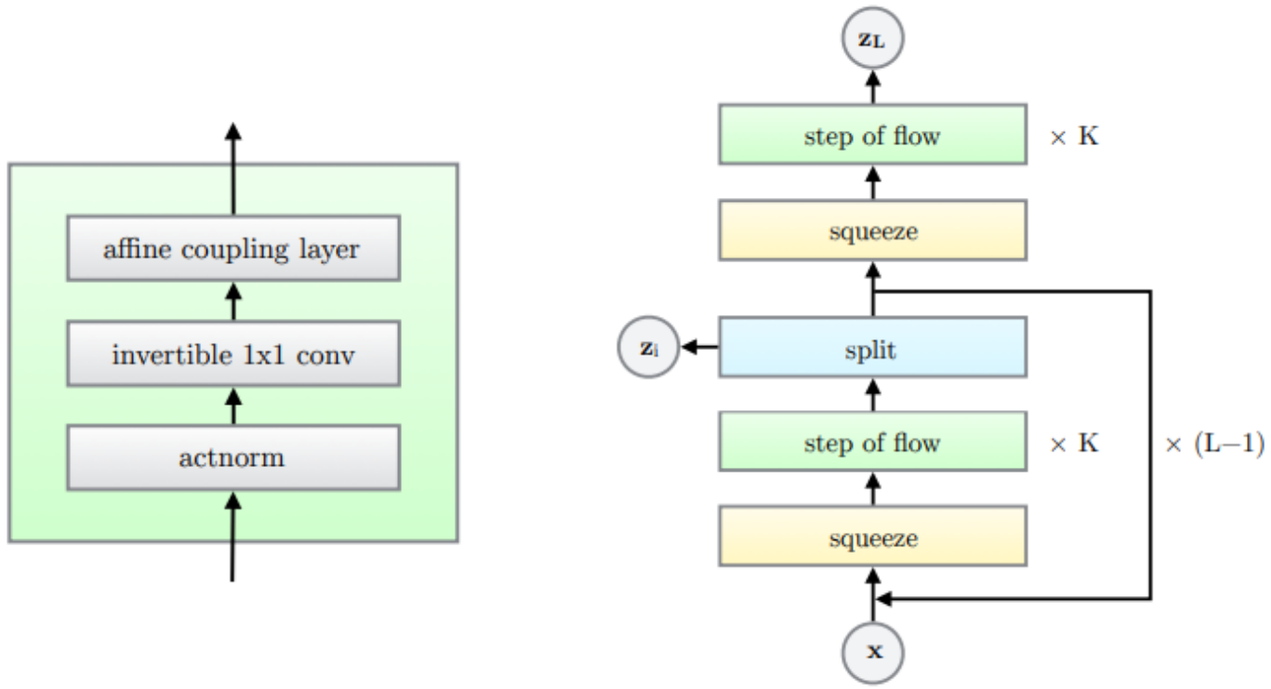


- **Application:** Speech synthesis, Google Text-to-Speech model

Oord, Aaron van den, et al. "Wavenet: A generative model for raw audio." arXiv 2016.

Glow

- **Idea: extends** and **simplifies** RealNVP.
- New in Glow:
  1. Invertible 1x1 convolution
  2. Act norm instead of batch norm in RealNVP



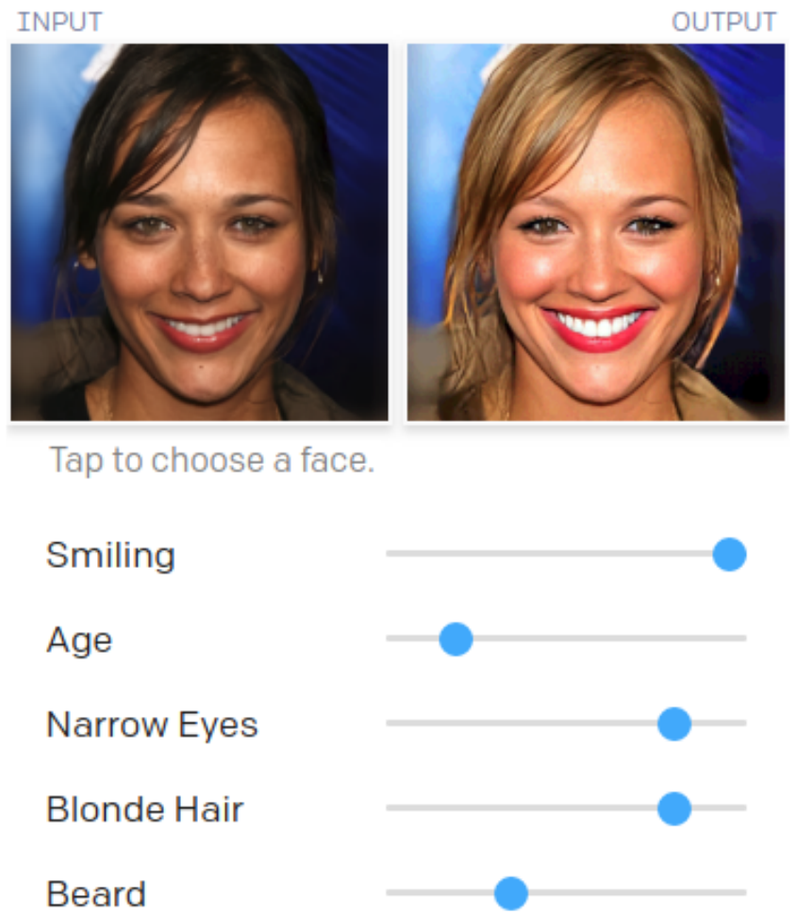
(a) One step of our flow.

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

Kingma, Diederik P., and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions." NIPS 2018.

Glow

- **Advantage:** Similar to RealNVP - sharper images and meaningful latent space.



- **Disadvantage:** Very deep and expensive model (600 conv layers). Still prefer to use GAN.

Kingma, Diederik P., and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions." NIPS 2018.

Conclusion

- Normalizing Flows (NF) provide an effective way to model complex distributions.
- NFs can be applied on various representation learning regimes, e.g.,
  - Variational Inference
    - Richer Latent-posterior proposals.
  - Density Estimation
    - Allows exact likelihood estimation.
- The invertible-maps required for NFs have been implemented as the `Bijector` in deep probabilistic programming languages **TFP** and **Pyro**.