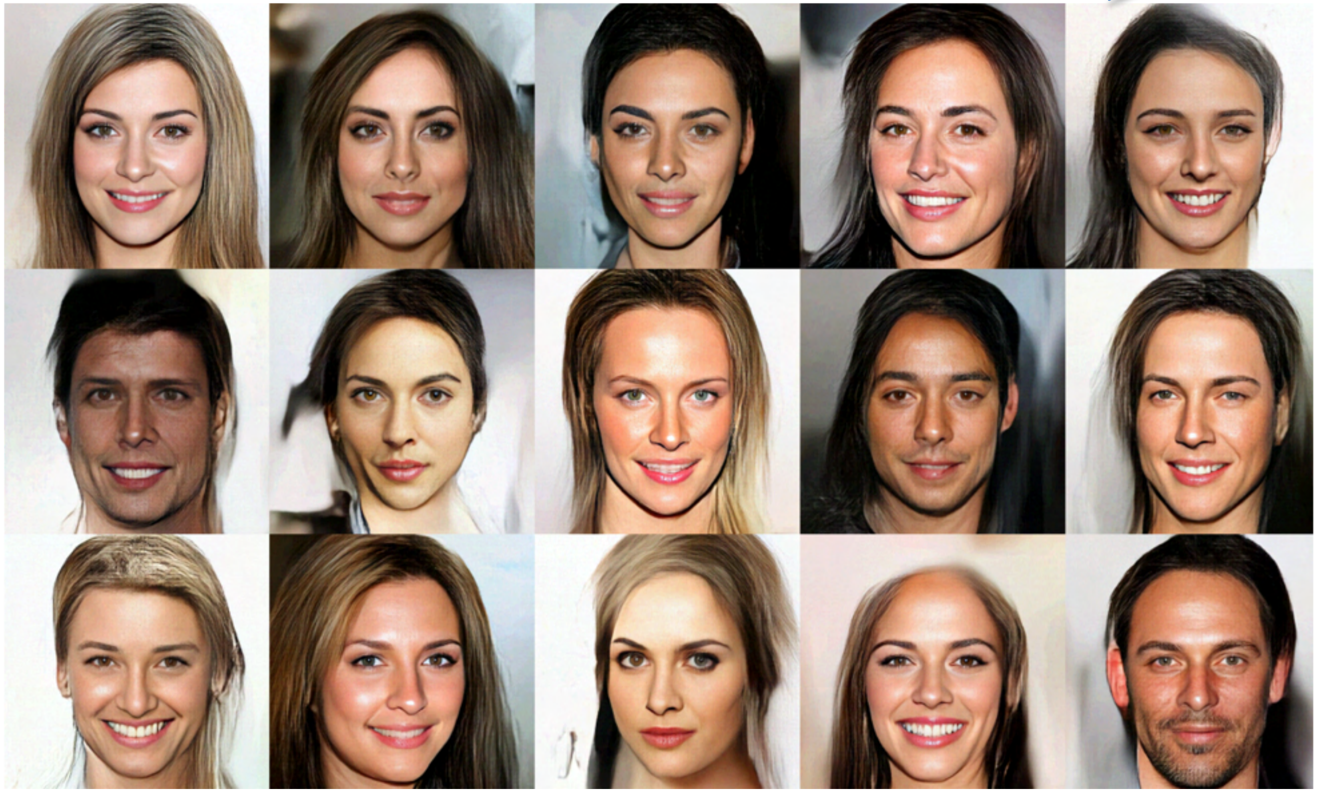
Go with the flow

An introduction to normalizing flows

These people are not real they are generated samples using NF



Oliver Dürr

Datalab BBS ZHAW 03/October/2019

A bit of Motivation

- A the End of the lecture, you can create and understand something like:

- Look at the intermediate pictures, they look real.
- Persons no celebrities (not part of celebA-HQ used for training)

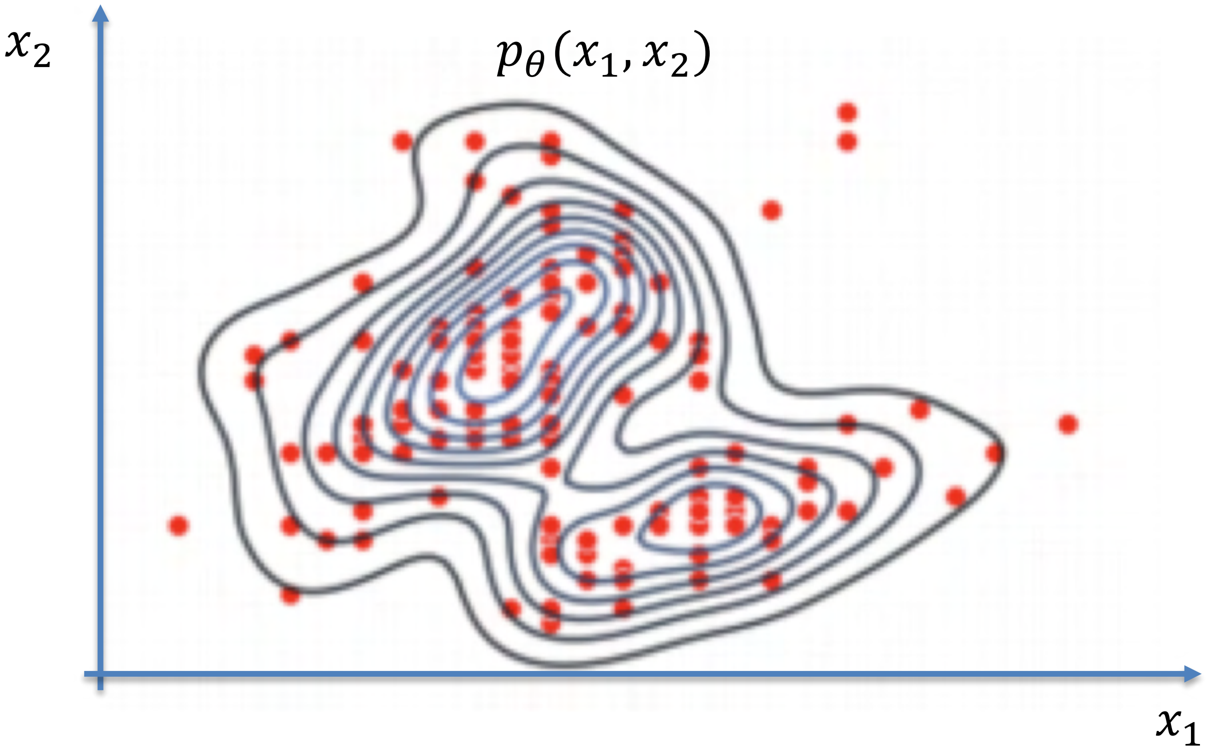
Outline

- Classification and motivate NF
  - Density Estimation
  - Generative Models
  - Need for flexible distributions
- Change of Variables
- Using networks to control flows
  - RealNVP
  - If time Autoregressive Flows
- Glow for image data

- Demo code is currently in
  - https://github.com/tensorchiefs/dl_book_playground/tree/master/flow

Normalizing Flows

- An novel method of parametric density estimation
  - Example of parametric density estimation 2-D Gaussians with μ and Σ



- Density Estimations are generative models…

Image from Priyank Jaini talk

Definition: Generative Model [cs231n]

Given training data, generate new samples from same distribution.

Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Several flavors:

- Explicit density estimation: explicitly define and learn $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly having a density

Why Generative Models?

- Generation of new data
  - For fun create persons that does not exists
  - Additional training data
  - Private Data (anonymization)
  - Image and Audio synthesis Wavenet / PixelCNN

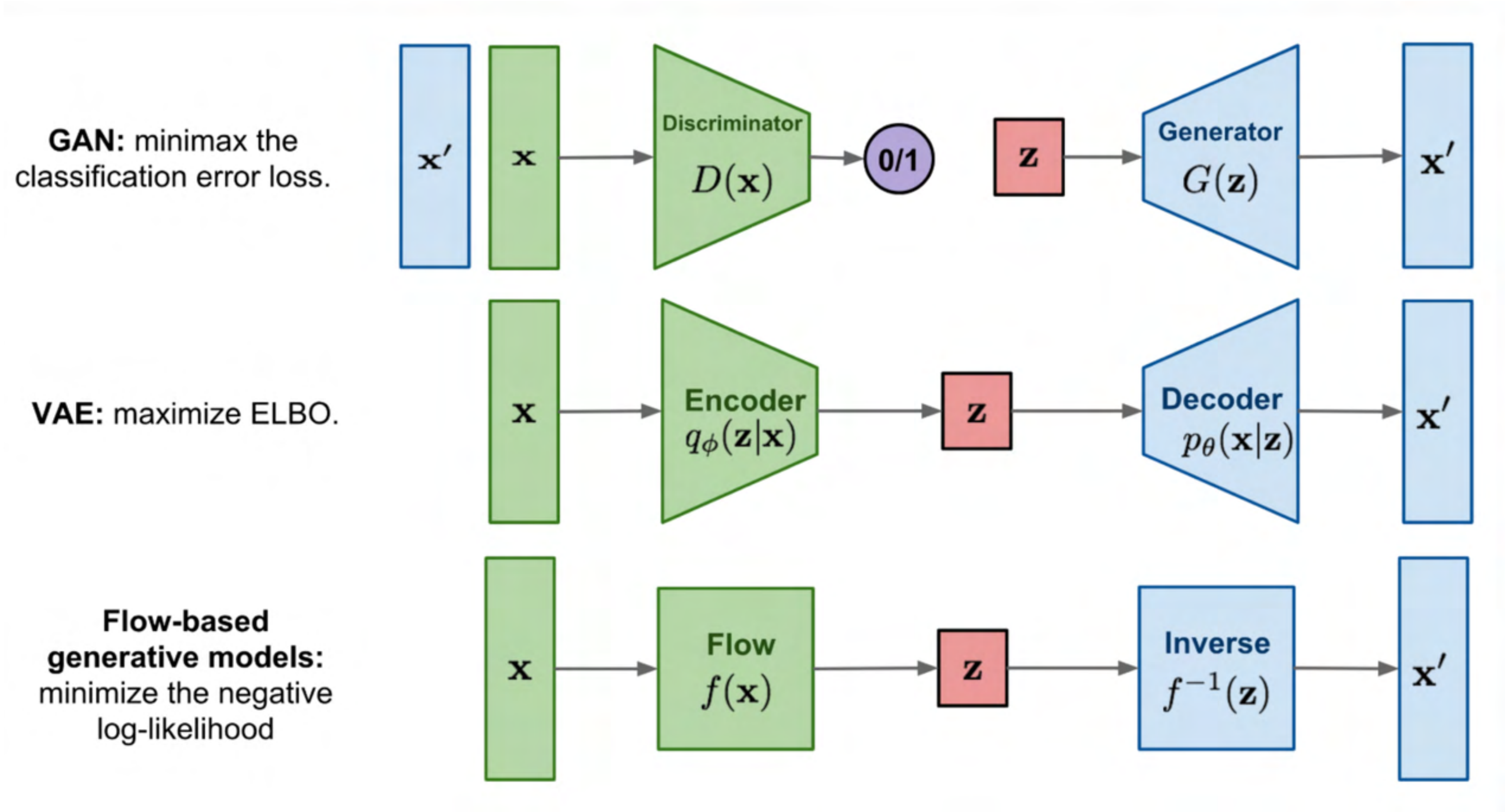
- Outlier detection $p_{ok}(x)$
  - Is image/vibration/… x from ok distribution?
    - Best with explicit models

- Semi-supervised Learning
  - Latent representation

- Flexible replacement for too simple functions
  - Pimp up prior of VAE

Image from Priyank Jaini talk

Generative models currently (2019) on vogue



Minmax

Approx. Likelihood (ELBO)

True, Likelihood

VAEs and GANs have been covered in Datalab BBS

Image (modified) from: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Generative models on vogue

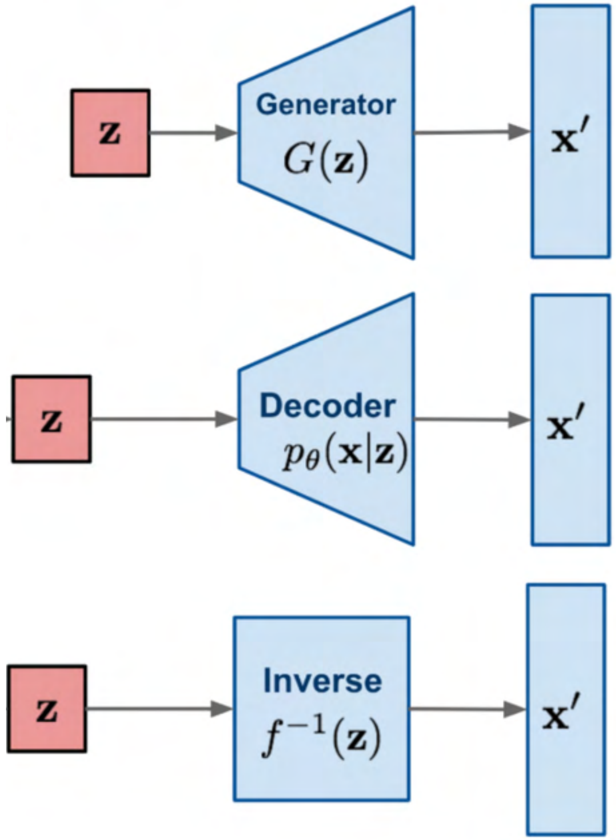
GAN: minimax the classification error loss.

VAE: maximize ELBO.

Flow-based generative models: minimize the negative log-likelihood

Different Training,
Same generative process

Z→X



Minmax

Approx. Likelihood (ELBO)

True, Likelihood

VAEs and GANs have been covered in Datalab BBS

Image (modified) from: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

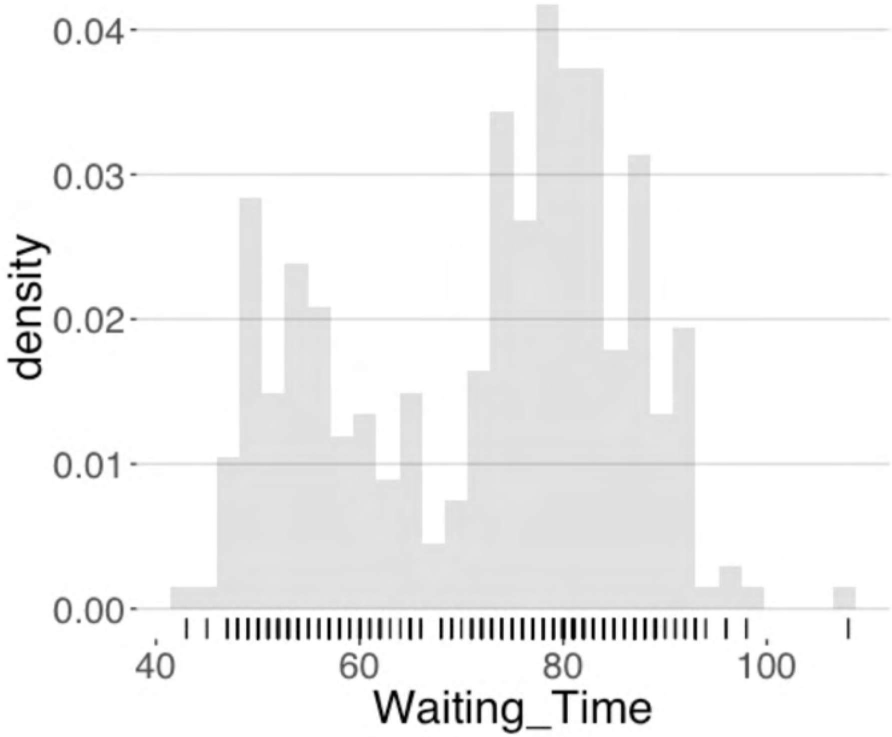
Theory: Name Some Distributions

- Gaussian
- Uniform
- Weibull
- Binomial
- Log-Normal

These are the distributions we have in our Toolbox.

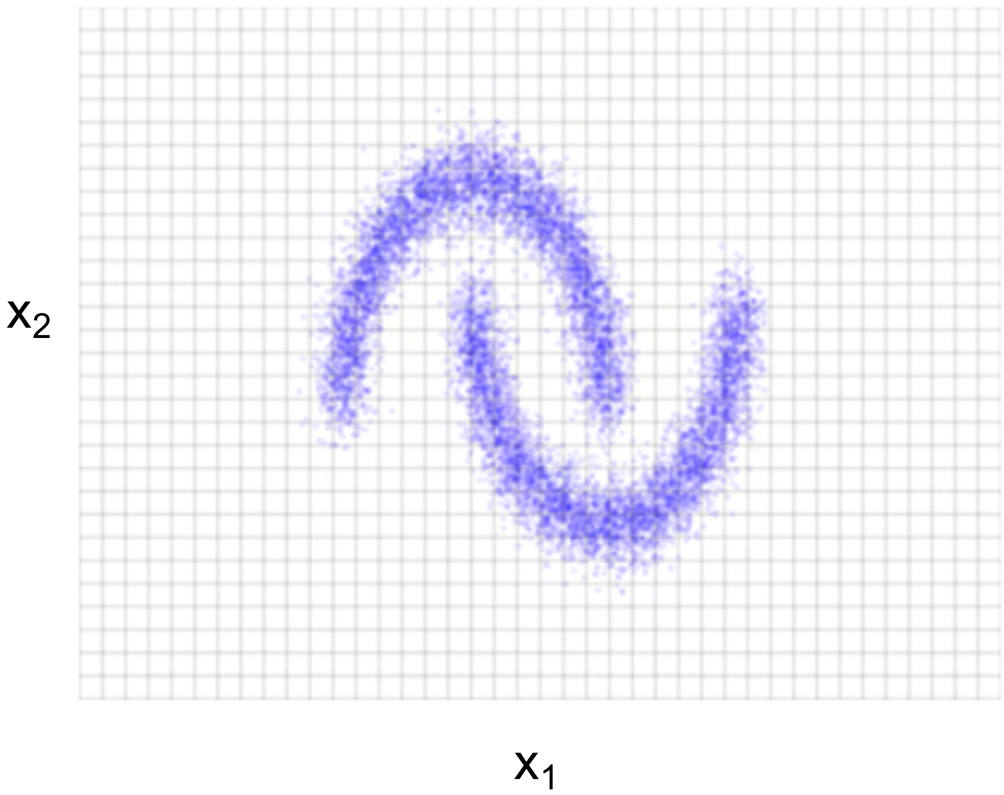
Is the reality like this?

Reality: Data (1-D)



What distribution can you use?

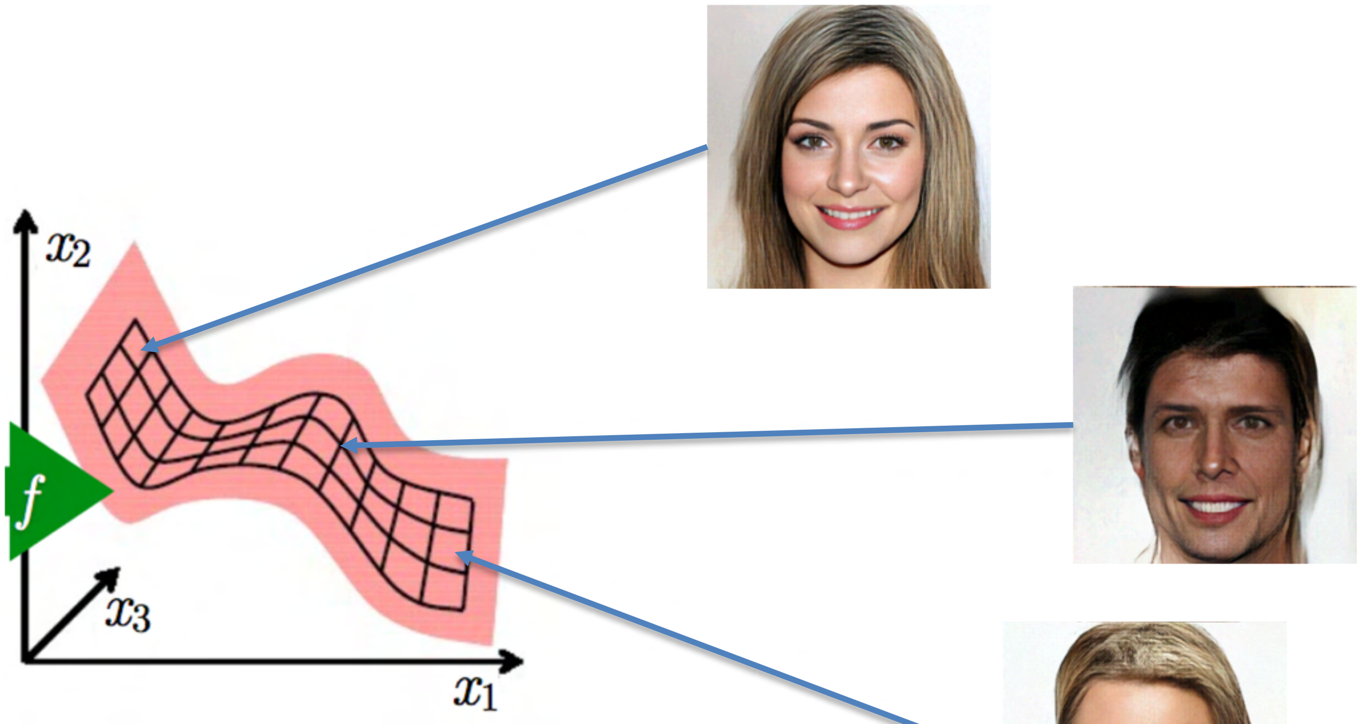
Reality: Data (2-D)



What distribution can you use?

Reality: Data (256x256x3=196'608 Dimensions)

3 data points sampled from the high dimensional distribution



This should be 196’608 dimensions
and not 3!

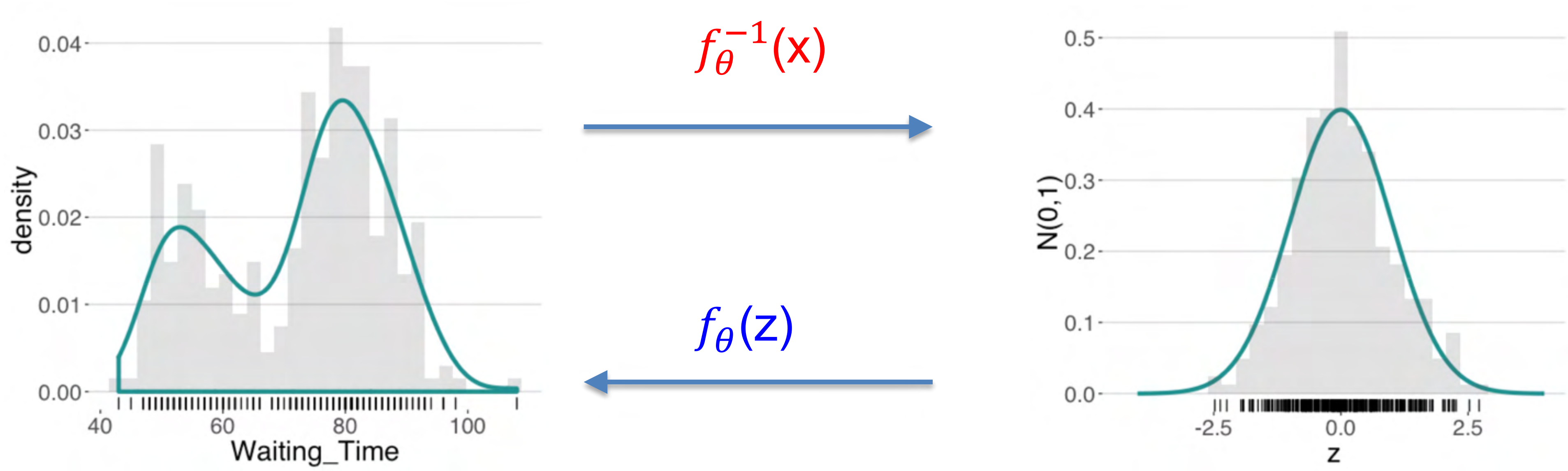
What distribution can you use?

Approches for Density Estimation task, we want $p_\theta(X)$:

- For easy cases fit normal “estimate mean and variance”
  - Limited to simple distributions
- Mixtures of simple Distributions such as Gaussian
  - Limited to fairly simple distribution
- Kernel Density estimation / Histograms
  - Non-Parametric, low dimensions (non-sparse)
- MCMC
  - Allows to sample from complicated distributions
  - Need pointwise p(X) up to constant
  - Typical $p(W|X)$ in Bayes
- GANs (only have an *implicit* estimation can sample from $p(X)$)
- VAE (only have an approximation to $p(X)$)
  - $\log\big(p(x)\big) = L^v + \cancel{D_{KL}(q(z|x)||p(z|x))}$ the KL-Term is disregarded
- **Normalizing Flows**

Main Idea of Normalizing Flows

Data $x \sim$strange_function in $\mathbb{R}^1$ Transformed function $z_1 \sim N(0,1)$



pdf p(x) pdf $\pi(z)$

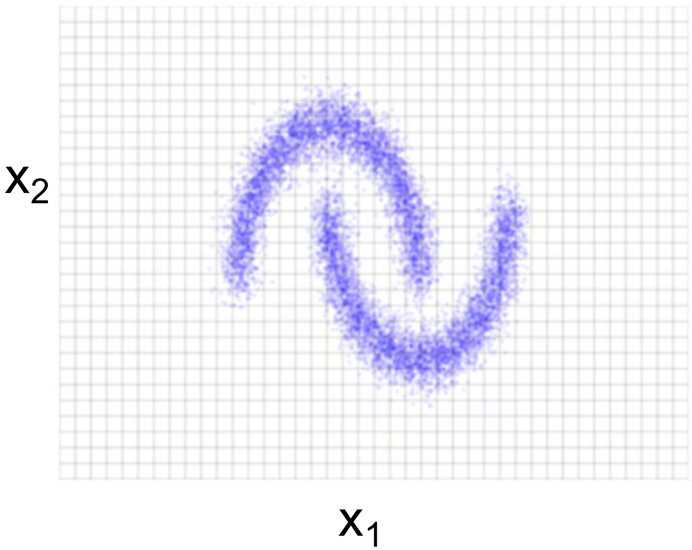
Idea: learn an *invertible* transformation to simple function usually Gaussian $N(0,1)$

- Sampling from p(x): sample $z^* \sim \pi(z)$ then transform it via f_θ(z*)
- Density of *x**: calculate z*= $f_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

Main Idea of Normaliuing Flows

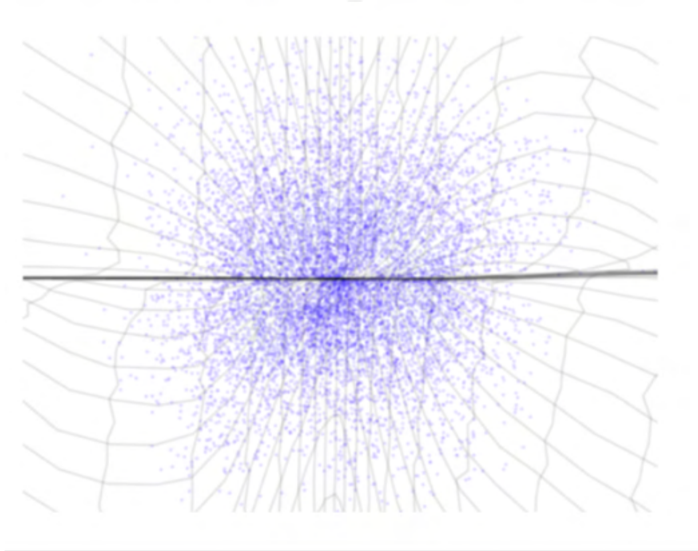
Data $x\sim$strange_function in $\mathbb{R}^2$

Transformed Data $z_1, z_2 \sim N(0,1)$



f_θ^{-1}(x) $\longrightarrow$

$\longleftarrow$ f_θ(u)



pdf p(x)

pdf $\pi(z)$

Idea: learn an *invertible* transformation to simple function usually Gaussian $N(0,1)$

- Sampling from p(x): sample $z^* \sim \pi(z)$ then transform it via f_θ(z*)
- Density of *x**: calculate z*= $f_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

Image Credit: RealNVP

Main Idea of Normalizing Flows

Data x~strange_function in $\mathbb{R}^{196608}$

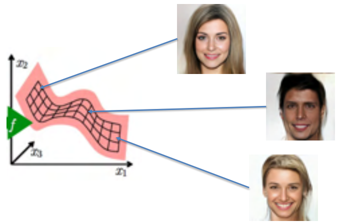
f_θ^{-1}(x)

x_1, x_2, x_{196608}~strange_function

$z_1, z_2, \dots, z_{196608} \sim N(0,1)$

With many correlations

f_θ(u)



pdf p(x)

pdf $\pi(z)$

Idea: learn an *invertible* transformation to simple function usually Gaussian $N(0,1)$

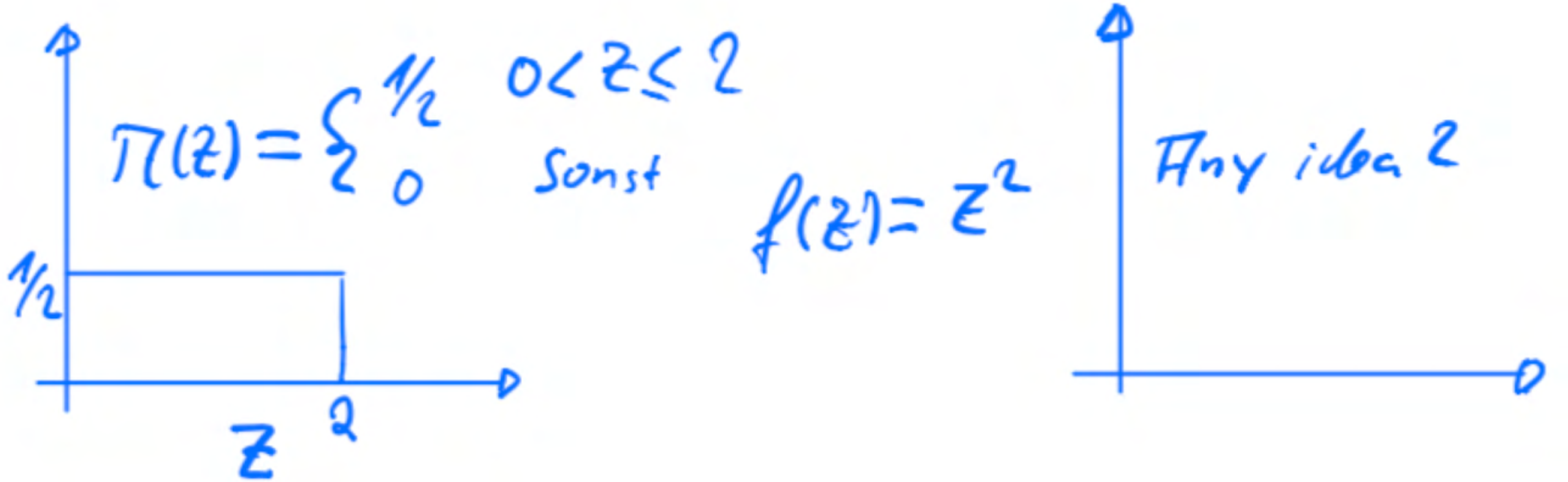
- Sampling from p(x): sample $z^* \sim \pi(z)$ then transform it via f_θ(z*)
- Density of *x**: calculate z*= $f_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

Transformation of Variables
-- Some math

Simple Transformation

- Say you have z~$Uniform(0,2)$
- $f(z) = z^2$

```
N = 10000
d = tfd.Uniform(low=0, high=2)
zs = d.sample(N)
x = zs**2
```



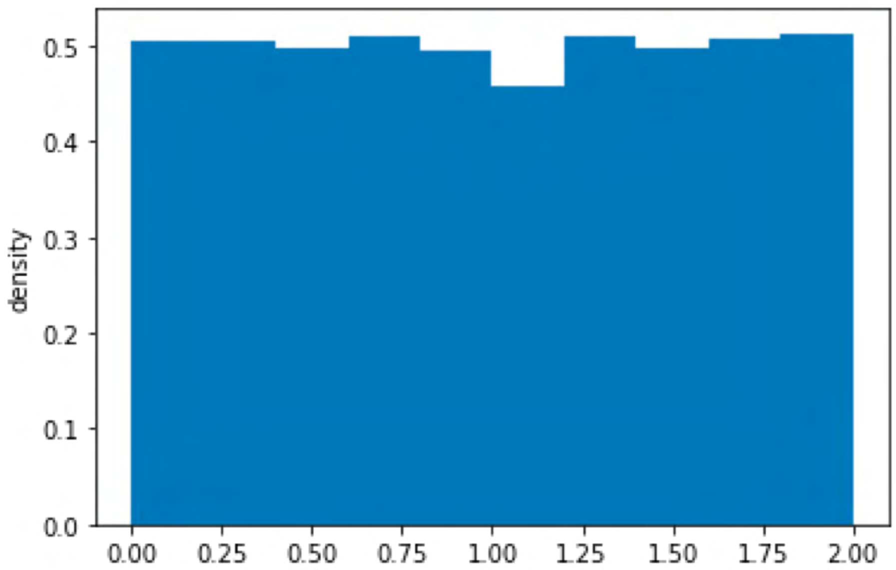
Try to come up with an answer, how is z distributed?

Try it

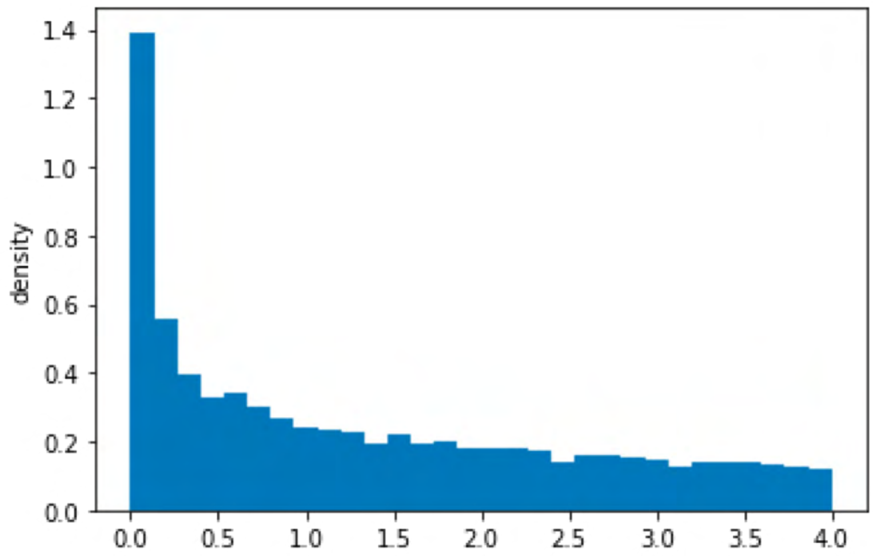
```
N = 10000
d = tfd.Uniform(low=0, high=2)
zs = d.sample(N)
```

```
x = zs**2
```

hist zs



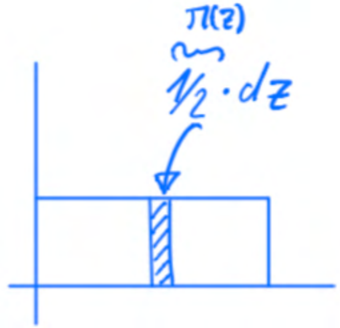
hist zs**2

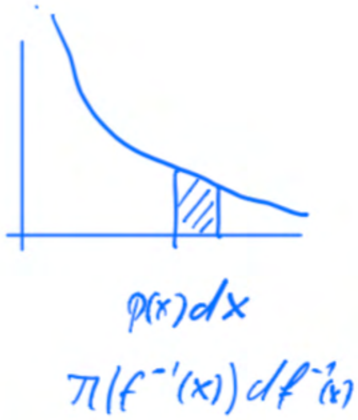


What happened?
Probability Mass needs to be conserved

Think of samples

Think of mass
needs to be conserved





$$\pi(z)\,dz = p(x)\,dx$$

1-D

$$\pi(z)\,dz = p(x)\,dx$$

$$\Rightarrow p(x) = \pi(z)\frac{dz}{dx}$$

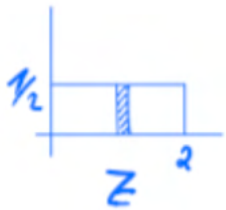
$$x = f(z) \Rightarrow z = f^{-1}(x)$$

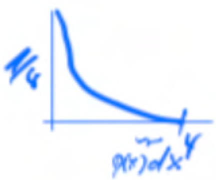
$$\Rightarrow \boxed{p(x) = \pi(f^{-1}(x))\left|\frac{df^{-1}(x)}{dx}\right|}$$

Bsp $\quad x = z^2 \Rightarrow z = f^{-1}(x) = \sqrt{x}$

$$p(x) = \pi(\sqrt{x})\frac{d\sqrt{x}}{dx}$$

$$p(x) = \begin{cases} \frac{1}{2}\,\frac{1}{2}\,\frac{1}{\sqrt{x}} & 0 < x \leq 4 \\ 0 & \end{cases}$$





Here $\left|\frac{df^{-1}(x)}{dx}\right|$ since $\frac{df^{-1}(x)}{dx}$ can be negative.

du and dx are positive by definition.

Transformation D>1

Generally $f: \mathbb{R}^d \to \mathbb{R}^d \quad x, z \in \mathbb{R}^d$

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$f^{-1}(\vec{x}) = \begin{pmatrix} f_1^{-1}(\vec{x}) \\ f_2^{-1}(\vec{x}) \\ f_3^{-1}(\vec{x}) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$$

$$\frac{df^{-1}(x)}{dx} \to \begin{pmatrix} \frac{\partial f_1^{-1}(\vec{x})}{\partial x_1} & \frac{\partial f_1^{-1}(\vec{x})}{\partial x_2} & \frac{\partial f_1^{-1}(\vec{x})}{\partial x_3} \\ \frac{\partial f_2^{-1}(\vec{x})}{\partial x_1} & \frac{\partial f_2^{-1}(\vec{x})}{\partial x_2} & \frac{\partial f_2^{-1}(\vec{x})}{\partial x_3} \\ \frac{\partial f_3^{-1}(\vec{x})}{\partial x_1} & \frac{\partial f_3^{-1}(\vec{x})}{\partial x_2} & \frac{\partial f_3^{-1}(\vec{x})}{\partial x_3} \end{pmatrix}$$

In higher dimensions

$f \colon \mathbb{R}^D \to \mathbb{R}^D$ from u (simple) to x (complicated)

$$p(x) = \pi\left(f^{-1}(x)\right) \cdot \left| \left(\frac{df^{-1}(x)}{\partial x} \right) \right| \longrightarrow p(x) = \pi\left(f^{-1}(x)\right) \cdot \left| \det \left(\frac{\partial f_i^{-1}(x)}{\partial x_j} \right) \right|$$

$$\log p(x) = \log \pi\left(f^{-1}(x)\right) + \log\left(\left| \det \left(\frac{\partial f_i^{-1}(x)}{\partial x_j} \right) \right| \right)$$

$c_{ij} = \dfrac{\partial f_i^{-1}(x)}{\partial x_j} =$ is the Jacobian of f^{-1}

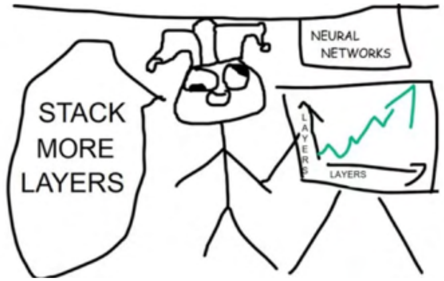
Intuition: The determinant of the Jacobian reflects the change of volume going from *x* to *u*. Going the other way, we get the reverse.

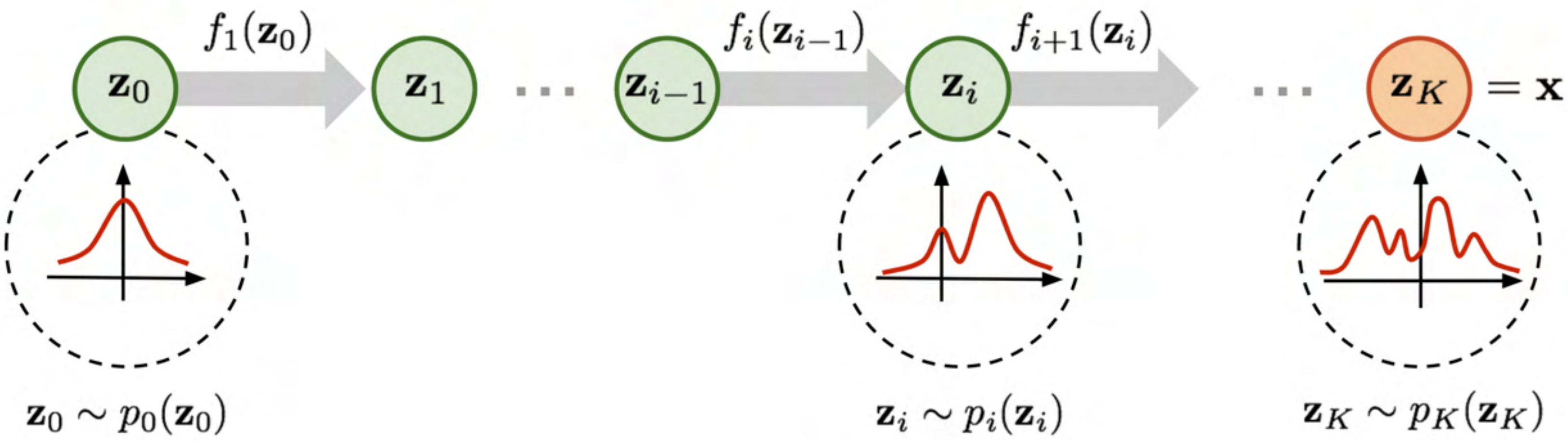
==> “inverse function theorem” (Not surprisingly)

$$\left| \det \left(\frac{\partial f_i^{-1}(x)}{\partial x_j} \right) \right| = \frac{1}{\left| \det\left(\frac{\partial f_i(z)}{\partial z_j} \right) \right|} = \left| \det \left(\frac{\partial f_i(z)}{\partial z_j} \right) \right|^{-1}$$

Normalizing Flows (Chaining Transformations)

- Start with a *simple distribution* for z_0
- Repeat change of variable K times to come to a complicated distribution z_k
- Chaining several bijectors as layers in neural networks
- This direction is sometimes referred to as “noise to data”





$$\log p(x) = \log p_0(z_0) - \sum \log\left(\det\left|\frac{\partial f_i(z_i)}{\partial z_{i-1}}\right|\right) \text{ with } z_i = f_i(z_{i-1})$$

The above equation needs a bit math (see blog post)

Image (modified) from: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Normalizing Flows in TFP (examples)

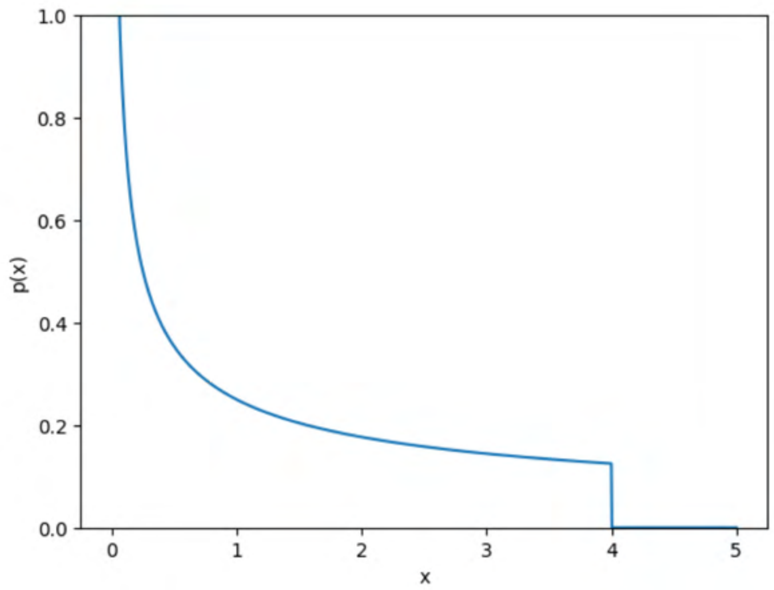
$f(z) \rightarrow$ `bijector` (the Square in our case)

```
In [35]: f = tfb.Square() # This is a bijector
         f.forward(2.0) #4
         f.inverse(4.0) #2

Out[35]: <tf.Tensor: id=974, shape=(), dtype=float32, numpy=2.0>
```

Doing the Transformation

```
In [27]: base_dist = tfd.Uniform(0.0,2.0)
         mydist = tfd.TransformedDistribution(distribution=base_dist, bijector=f)
```

```
In [36]: xs = np.linspace(0.001, 5,1000)
         ps = mydist.prob(xs)
         plt.plot(xs,ps)
         plt.xlabel('x')
         plt.ylabel('p(x)')
         plt.ylim(0,1)

Out[36]: (0, 1)
```



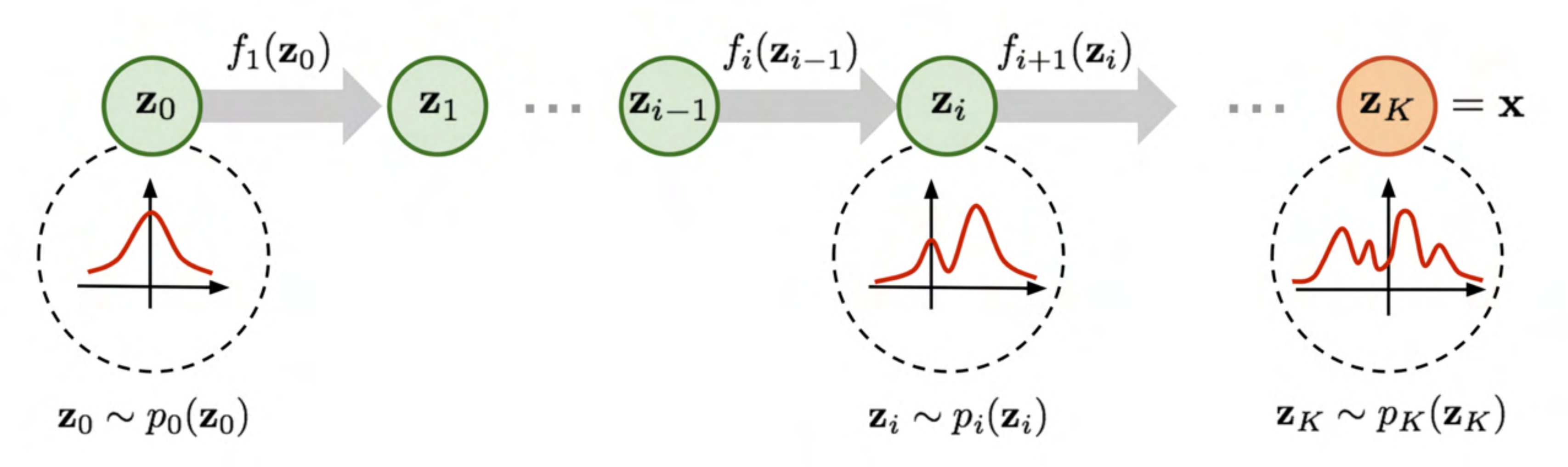
Chaining several Bijectors

Using several bijectors

```
In [39]: chain = tfb.Chain([tfb.Square(), tfb.Square()], name="x4")
         chain.forward(2.0)

Out[39]: <tf.Tensor: id=1174, shape=(), dtype=float32, numpy=16.0>
```

Notebook Flow_101.ipynb

Learning to flow



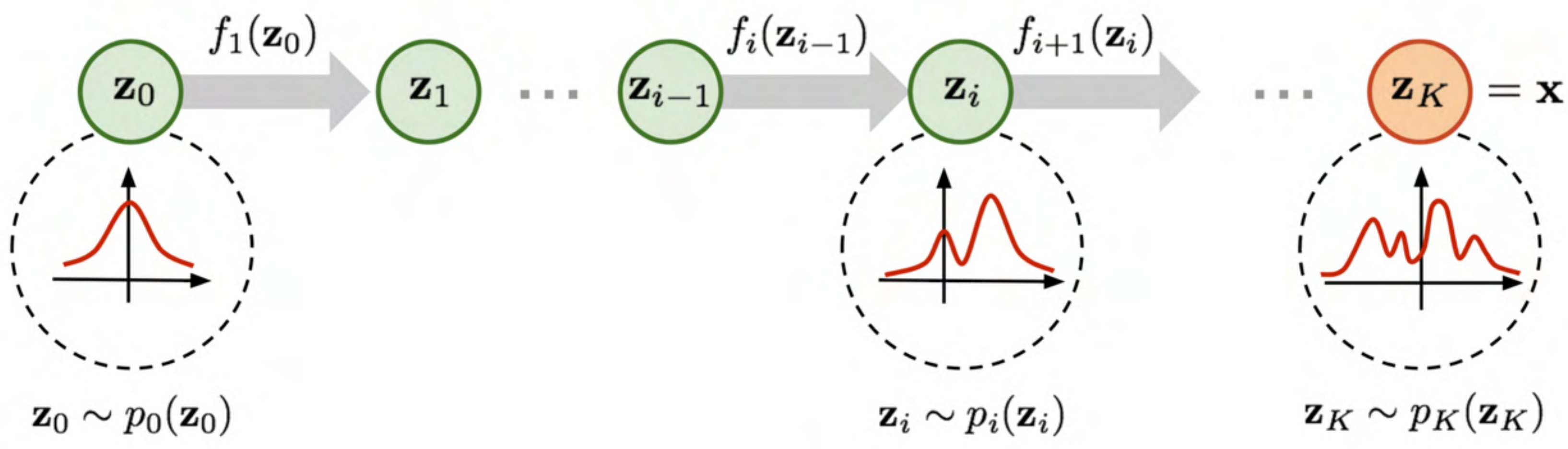
$$\log p(x) = \log p(z_k) = \log p_0(z_0) - \sum \log\left(\det\left|\frac{\partial f_i(z_i)}{\partial z_{i-1}}\right|\right) \text{ with } z_i = f_i(z_{i-1})$$

The log-probability $\log p(x)$ of a training sample x can be easily calculated from the Jacobian and the $\log p_0(z_0)$. You get z_0 by successively applying the reversed functions f_i^{-1}.

How to fit?

Tune the parameter(s) θ of the model M so that (observed) data is most likely!

Learning to flow



$$\log p(x) = \log p(z_k) = \log p_0(z_0) - \sum \log\left(\det\left|\frac{\partial f_i(z_i)}{\partial z_{i-1}}\right|\right) \text{ with } z_i = f_i(z_{i-1})$$

The log-probability $\log p(x)$ of a training sample x can be easily calculated from the Jacobian and the $\log p_0(z_0)$. You get u_0 by successively applying the reversed functions f_i^{-1}.

Maximum Likelihood: Minimize the Negative Log Likelihood $-\sum \log p(x^i)$ of all training data point x^i. There parameters of the model, are in the transformations.

Simple example in NB Flow_101.ipynb

Requirements for the bijectors

A flow is composed of serval possible different *f's,* the bijectors in TFP language. The following restrictions apply for them

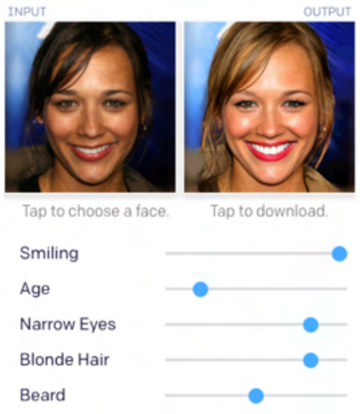
- *f* needs for be invertible (strict requirement)
- Training
  - Fast calculation of $f^{-1}(x)$
  - Fast calculation of Jacobi-Determinant
- Application:
  - Fast calculation of $f(z)$

Flows with networks

Flows using networks

2 Main lines of research

- Guided by autoregressive (AR) models
  - All AR models like Wavenet can be understood as normaliuing flows
    - Mask Autoregressive Flow (MAF)
    - Inverse Mask Autoregressive Flow (IMAF)
- Using 'handcrafted' network based flows
  - NICE (1410.8516 Dinh, Krueger, Bengio)
  - RealNVP ([1605.08803](1605.08803) Dinh, Dickstein, Bengio)
  - Glow (https://arxiv.org/abs/1807.03039 Kingma, Dahriwal)



- Unifying framework (Triangular Maps)
  - SOS paper ICML https://arxiv.org/abs/1905.02325

Requirement / Design considerations

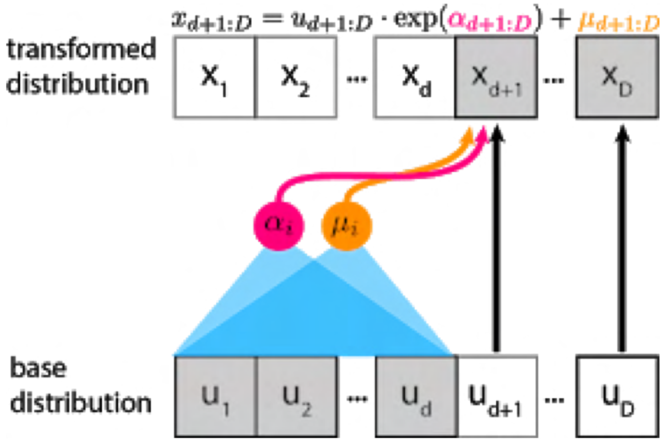
- Fast calculation of $f(z),\ f^{-1}(x)$
- Crucial: We need fast calculation of Jacobi Matrix
  - $\left|\det\left(\frac{\partial f_i(z)}{\partial z_j}\right)\right|^{-1}$

$$\begin{pmatrix} \frac{\partial f_1(z)}{\partial z_1} & \frac{\partial f_1(z)}{\partial z_2} & \frac{\partial f_1(z)}{\partial z_3} \\ \frac{\partial f_2(z)}{\partial z_1} & \frac{\partial f_2(z)}{\partial z_2} & \frac{\partial f_2(z)}{\partial z_3} \\ \frac{\partial f_3(z)}{\partial z_1} & \frac{\partial f_3(z)}{\partial z_2} & \frac{\partial f_3(z)}{\partial z_3} \end{pmatrix}$$

- Lin. Alg.: The determinant of triangular matrix is sum of diagonal terms (trace)
  - Want triangular matrix $\frac{\partial f_1(z)}{\partial z_2} \overset{!}{\triangleq} 0$
    - ➔ $f_1(z) = f_1(z_1, \cancel{z_2, z_3}),\ f_d(z) = f_1\left(z_1, \ldots, z_d, \cancel{z_{d+1}, z_{d+2,\ldots}}\right)$
  - Diagonal terms $\frac{\partial f_2(z)}{\partial z_2}$ easy to be calculated (no network!)
- $\frac{\partial f_2(z)}{\partial z_1}$ no restrictions, can be as complicated as hell (neural network)

Real-NVP (coupling layer)

- Main ingredient the coupling layer
- Consider (high) dimensional data with dimension *D*



z->u

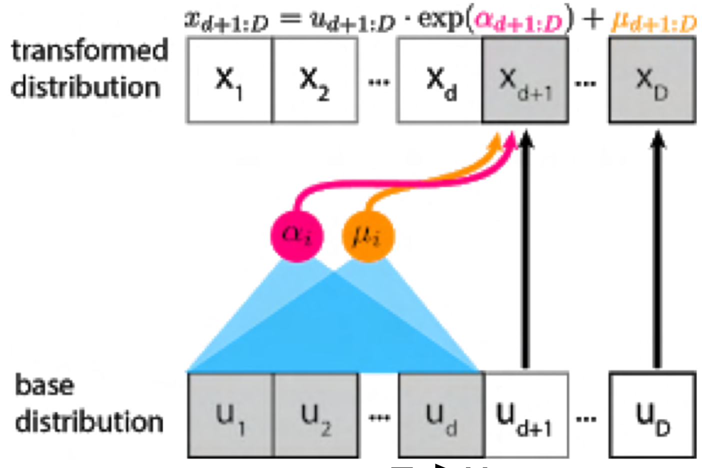
- Choose arbitrary d<D $\alpha_i(z_{1:d})$ and $\mu_i(z_{1:d})$ are NNs with inputs $z_1 \dots z_d$ and outputs for $d+1, \dots, D$.
- $x_1 = z_1 \quad x_2 = z_2 \dots \quad x_d = z_d$
- $x_{d+1} = \mu_i(z_{1:d}) + \exp\big(\alpha_i(z_{1:d})\big) \cdot z_{d+1}$ # shift and scale transformation
- # $x_{d+1} \sim N(\mu = \mu_i(z_{1:d}), \sigma = \exp\big(\alpha_i(z_{1:d})\big))$ renormalisation trick
- $x_{d+2} = \mu_{i+1}(z_{1:d}) + \exp(\alpha_{i+1}) \cdot z_{d+2},$
- $\dots$
- In short $x_{1:d} = z_{1:d}$ and $x_{d:D} = \mu_{i:i+d}(z_{1:d}) + \exp(\alpha_i) \cdot z_{d:D}$

Figure credit Eric Jang

Real-NVP (coupling layer, properties)

Inverse

- $z_{1:d} = x_{1:D}$ No network
- $z_{1:d} = x_{1:D}$
- $x_{d+1:D} = \mu\left(z_{1:d}\right) + \exp(\alpha_i(z_{1:d})) \cdot z_{d+1:D}$
- $z_{d+1:D} = \left(x_{d+1:D} - \mu\left(z_{1:d}\right)\right) / \exp(\alpha_i(z_{1:d})))$



z->u

Jacobian for D=5, d=2 note that $\frac{\partial f_i(z)}{\partial z_j} = \frac{\partial x_i}{\partial z_j}$

$$
\begin{array}{c}
\xrightarrow{\quad j \quad} \\
i \downarrow \begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
e & e & \exp(\alpha_1(z_{1:2})) & 0 & 0 \\
e & e & e & \exp(\alpha_2(z_{1:2})) & 0 \\
e & e & e & e & \exp(\alpha_3(z_{1:2}))
\end{pmatrix} \\
\begin{array}{ccccc}
\frac{\partial .}{\partial z_1} & \frac{\partial .}{\partial z_2} & \frac{\partial .}{\partial z_3} & \frac{\partial .}{\partial z_4} & \frac{\partial .}{\partial z_5}
\end{array}
\end{array}
$$

$x_1 = z_1$

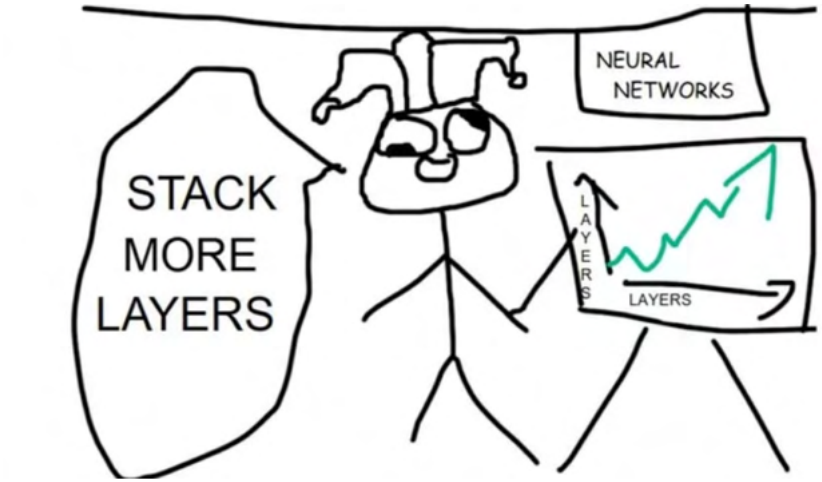
$x_2 = z_2$

$x_3 = \mu_i(z_{1:2}) + \exp\left(\alpha_i(z_{1:2})\right) \cdot z_3$

$x_4 = \mu_i(z_{1:2}) + \exp\left(\alpha_i(z_{1:2})\right) \cdot z_4$

$x_5 = \mu_i(z_{1:2}) + \exp\left(\alpha_i(z_{1:2})\right) \cdot z_5$

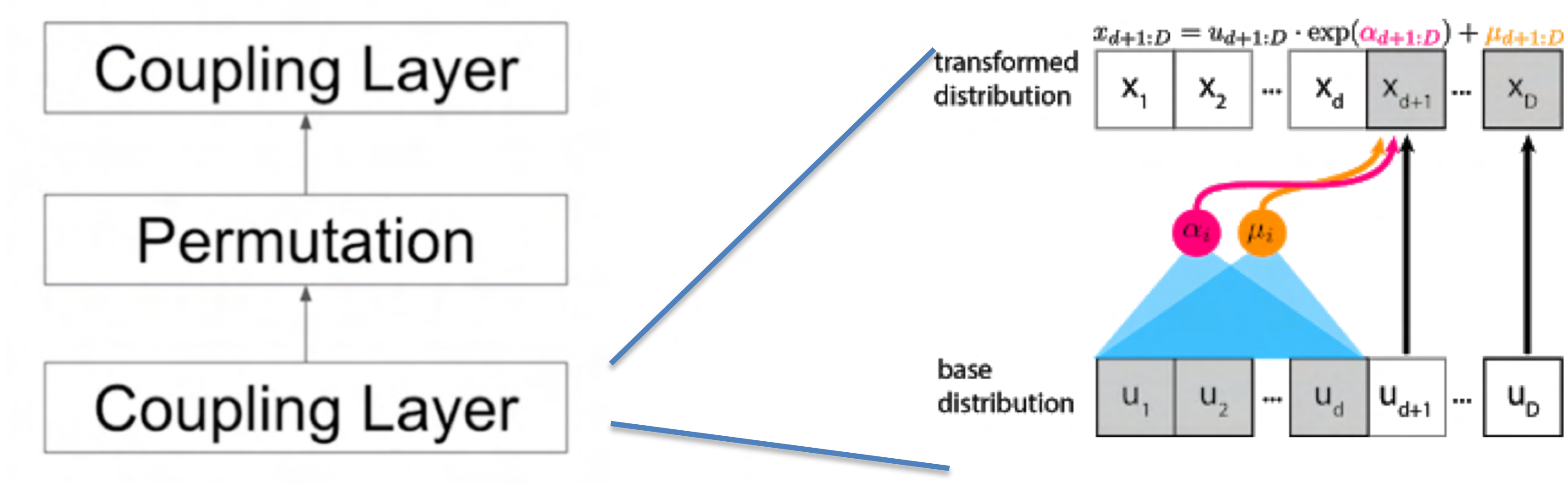
e=don't care



Do the DL-Trick

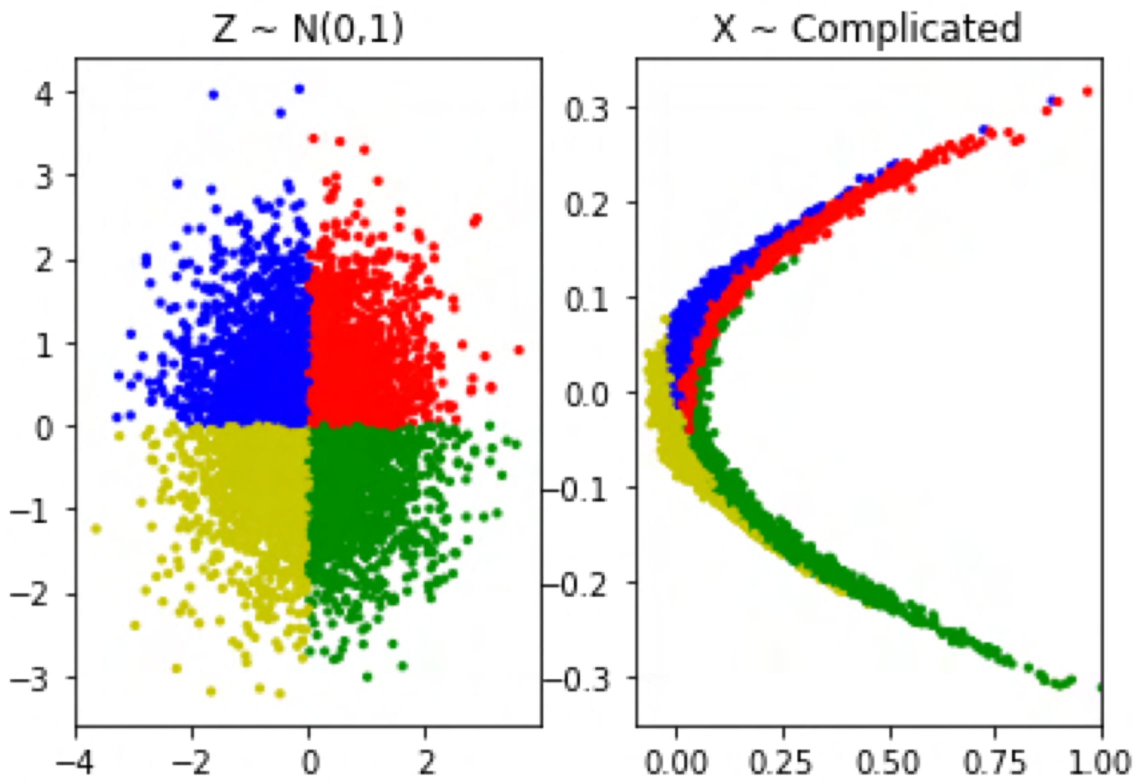
Stack more Layers (Permutation)

- In RealNVP
  - d is arbitrary and also the ordering
  - In AR-Flows ordering is arbitrary
- When stacking several coupling layers put fixed permutation of dimensions in between
- Fix permutation is invertible and det=1 (If a bijection)



Demo

- See Flow_101_learning_parameters_NVP



Glow for image data

--arXiv:1807.03039

Glow: Generative Flow with Invertible 1×1 Convolutions

Diederik P. Kingma*, **Prafulla Dhariwal***
OpenAI, San Francisco

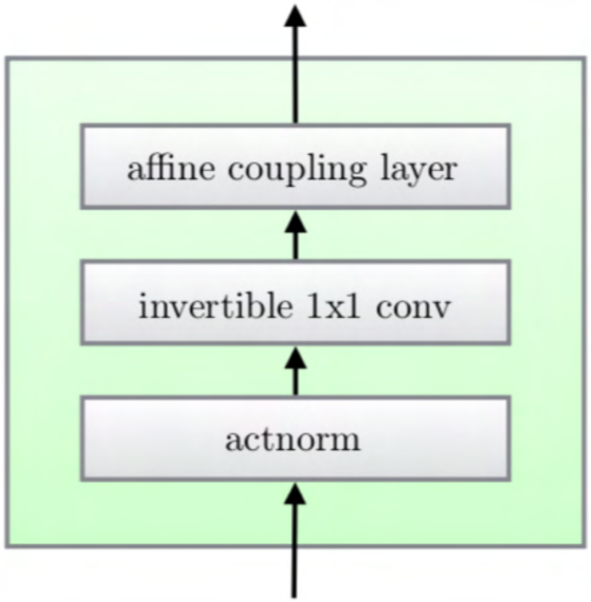
Specialties of glow

- Use 1x1convolutions instead of Permutation

- Image Data
  - Multiscale Architecture (also in RealNVP Paper)
  - X and Z are now tensors (3 dimensional, shape w,h,c)
  - Keep the w,h dimension work on the channel dimension
  - The channel dimension get’s larger by squeeze operation (see below)
  - As before (Affine coupling layer now with tensors)

Glow (new incredients)

- Additional actnorm (like a batchnorm for batch siue 1)
- Instead of a permutation 1x1 convolution is used (simple Matrix Multiplication)
- They stack 32 of those layers



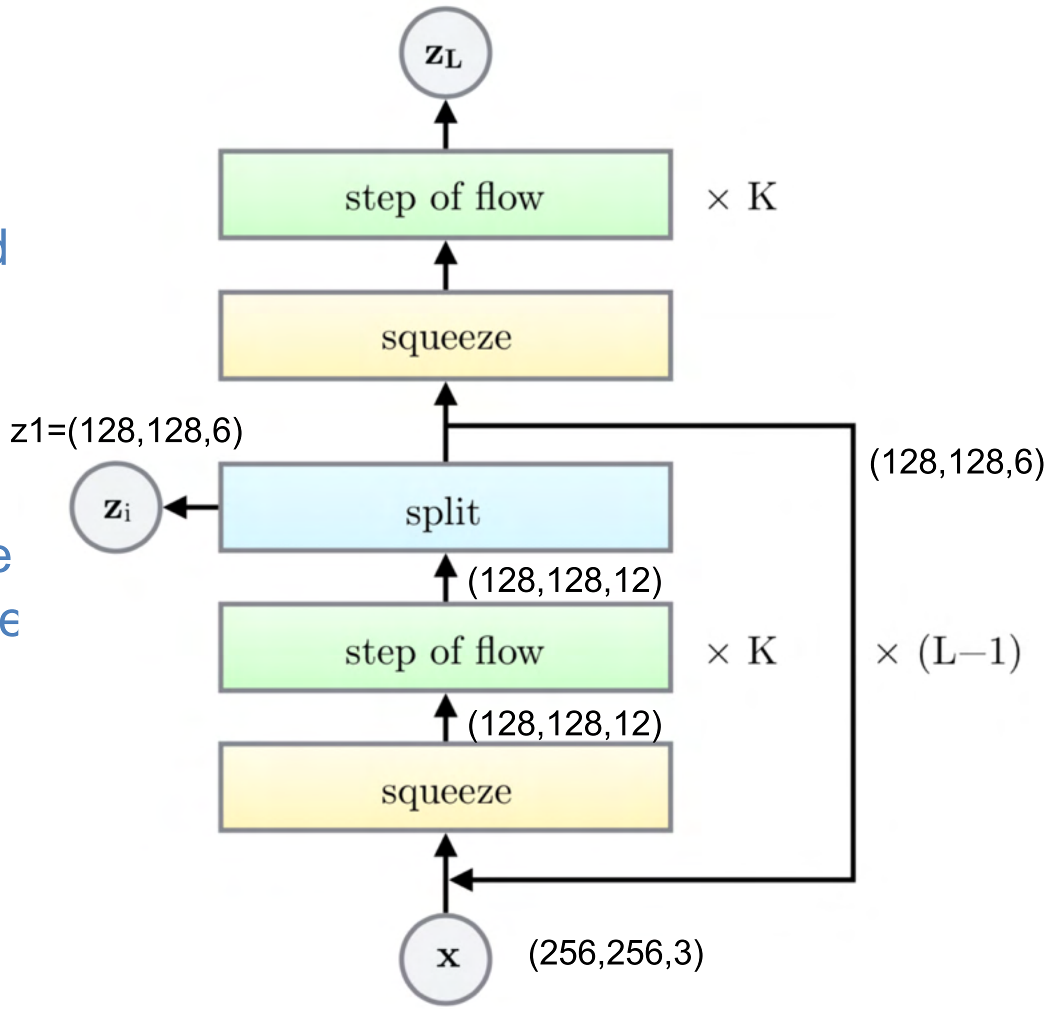
(a) One step of our flow.

Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$

Multiscale Architecture

- Squeeze operation:
  - s,s,c → s/2, s/2, 4*c
  - Reduces the spatial resolution
  - Keeps the number of entries fixed

- Split operation
  - Splits input tensor in two halves
  - 50% of the variables only observe one flow. These correspond to fine grade details.
  - The rest is squeezed and thus describes finer details
  - L = 6 in paper



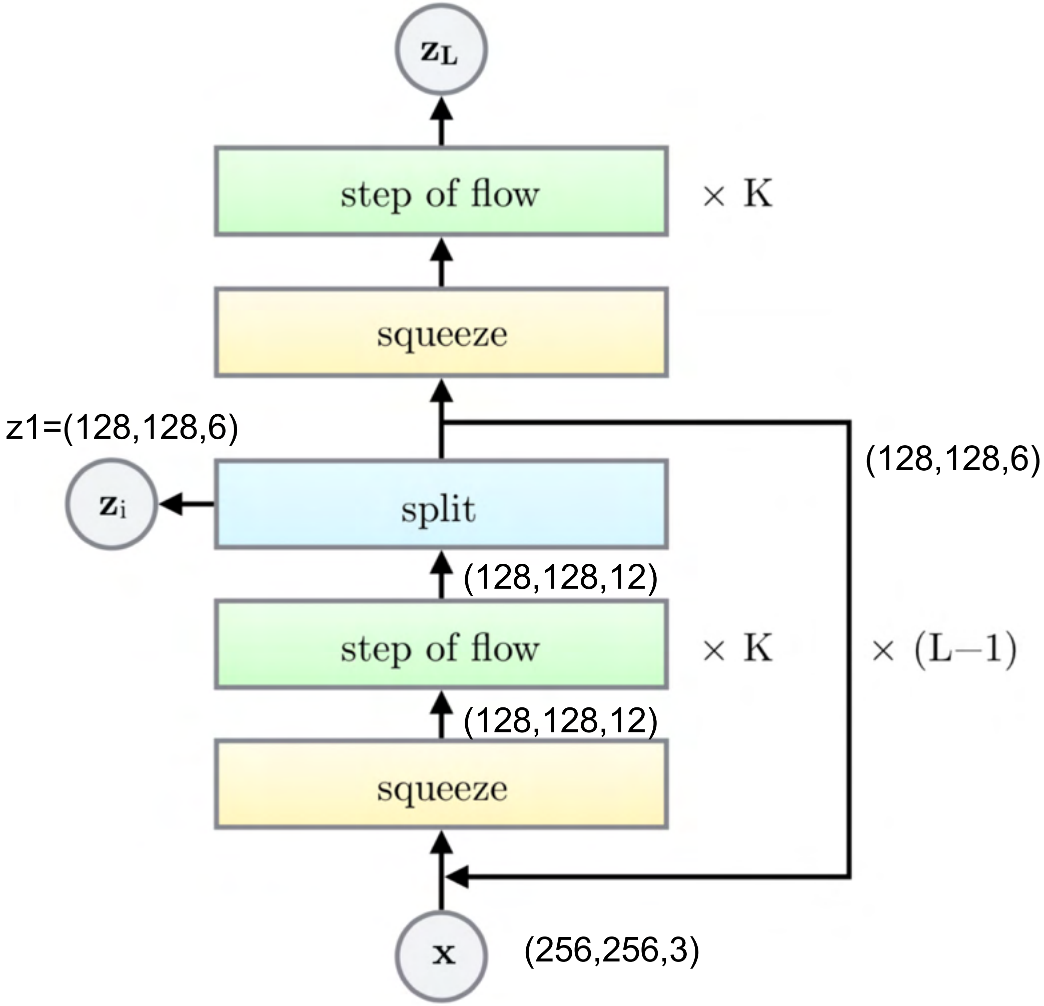
Multiscale Architecture

- Shapes of the Z

```
for i,e in enumerate(eps_shapes):
  print('z_{}'.format(i+1),e)
```

```
z_1 (128, 128, 6)
z_2 (64, 64, 12)
z_3 (32, 32, 24)
z_4 (16, 16, 48)
z_5 (8, 8, 96)
z_6 (4, 4, 384)
```

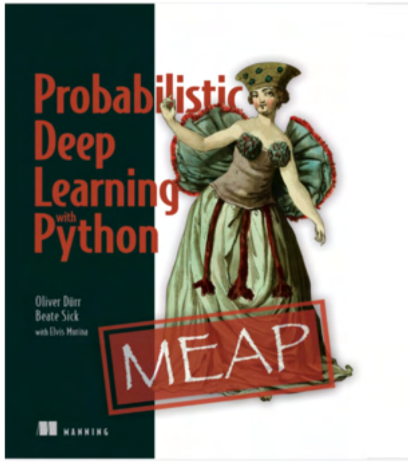
$$\sum = (256,256,3)$$



Demo

- Network has been trained on CelebA-HQ
  - 30000 (256x256x3) images of celebrities
  - Images have been aligned
- Sampling: draw 256*256*3 numbers from N(0,1)
  - Reduced Temperature draw from N(0,T*1)
- Interpolation
  - Blackboard
- Demo
  - Uses pretrained network
  - **fun_with_glow**

Further reading



Coming soon

Some interesting reads and talks

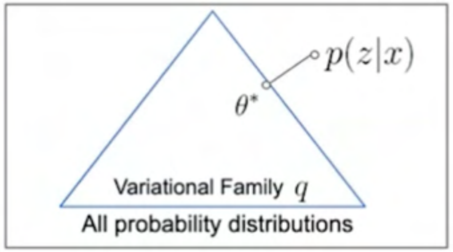
- Eric Jang
  - Blog: part1 (introduction) part2 (modern flows)
  - 2019 ICML Tutorial
- Priyank Jaini
  - Lecture Waterloo University CS 480_680 8/24/2019 lecture 23 (slides | youtube)
  - SOS paper ICML (https://arxiv.org/abs/1905.02325) Talk
- Arsenii Ashukha
  - Lecture at day 3 at deepbayes.ru summer school 2019 (slides | video)
- Papers (relevant to this talk)
  - Density estimation using Real NVP: https://arxiv.org/abs/1605.08803
  - Glow: Generative Flow with Invertible 1×1 Convolutions https://arxiv.org/abs/1807.03039

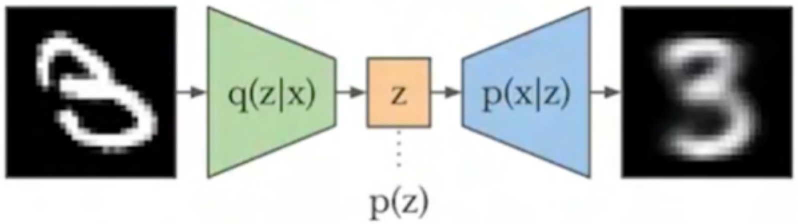
Thank you! Questions?

Further use of flows

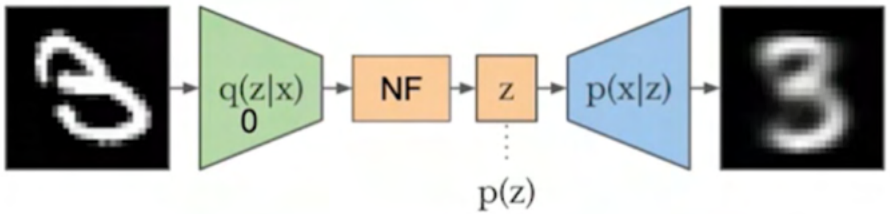
- It's possible to use normaliuing flow as a drop-in replacement for anywhere you would use a Gaussian, such as VAE priors [evjang]

$$\log p(X) = \mathcal{L}(X, \theta) + KL(q_\theta(z \,|\, x) \,||\, p(z \,|\, x))$$





q(u|x) is network parameteruing Gaussian



Use NF to make this more expressive

Material to check

- Tutorial on normaliuing flows, slideslive.com/38917907/tutorial-on-normaliuing-flows
- ● Tips for Training Likelihood Models, blog.evjang.com/2019/07/likelihood-model-tips.html
- ● FFJORD tutorial, https://youtu.be/_ALdCSSVYkw
- ● Must read papers:
  - ○ Variational Inference with Normaliuing Flows, https://arxiv.org/abs/1505.05770
  - ○ Density estimation using Real NVP, https://arxiv.org/abs/1605.08803
  - ○ Glow: Generative Flow with Invertible 1×1 Convolutions https://arxiv.org/abs/1807.03039
  - ○ Sylvester Normaliuing Flows for Variational Inference, https://arxiv.org/abs/1803.05649
  - ○ FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, https://arxiv.org/abs/1810.01367
  - ○ Do Deep Generative Models Know What They Don't Know?, https://arxiv.org/abs/1810.09136
  - ○ Classification Accuracy Score, https://arxiv.org/abs/1905.10887