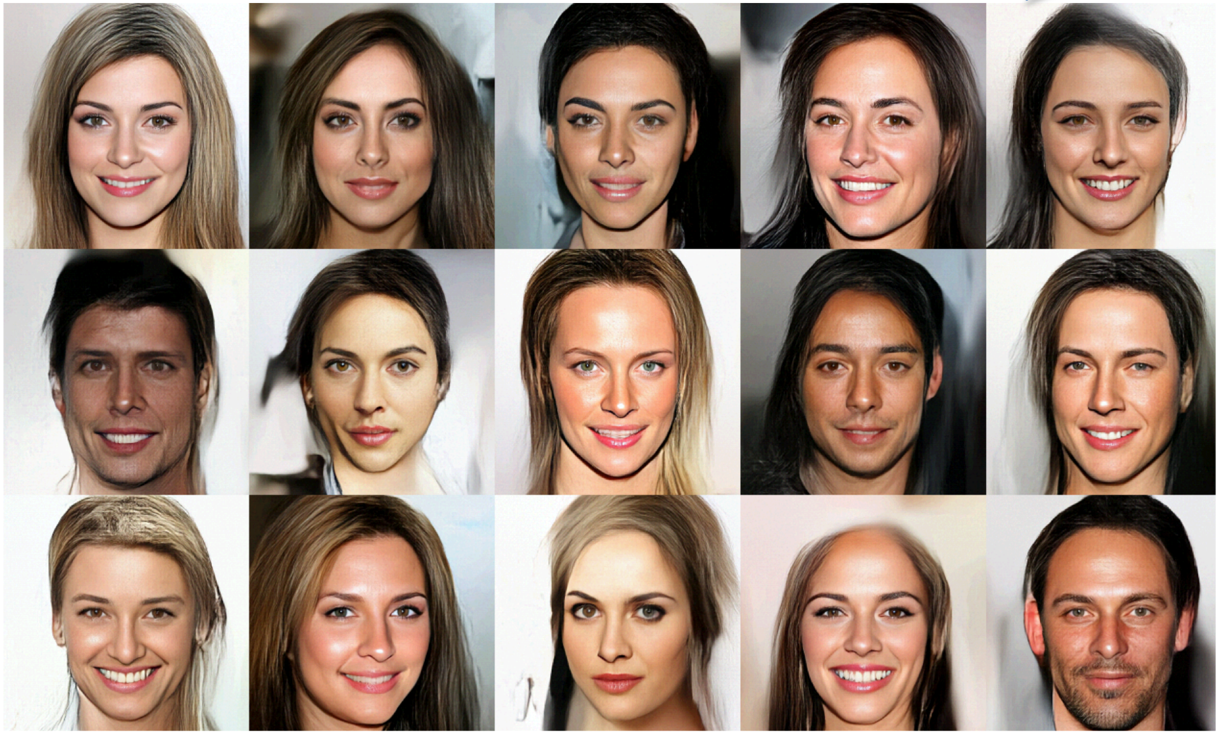
Go with the flow

An introduction to normalizing flows

These people are not real they are generated samples using NF



Oliver Dürr

Brown Bag Seminar HTWG 25/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)

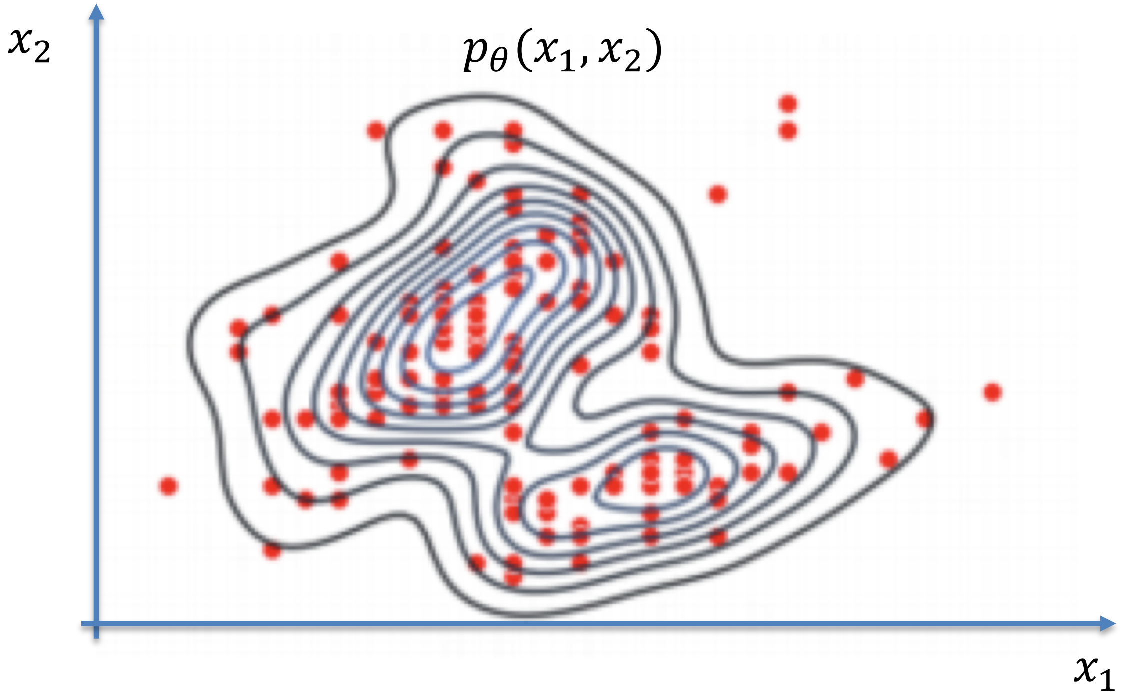
Outline

- 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/tree/master/chapter_06

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 ~ $p_{data}(x)$

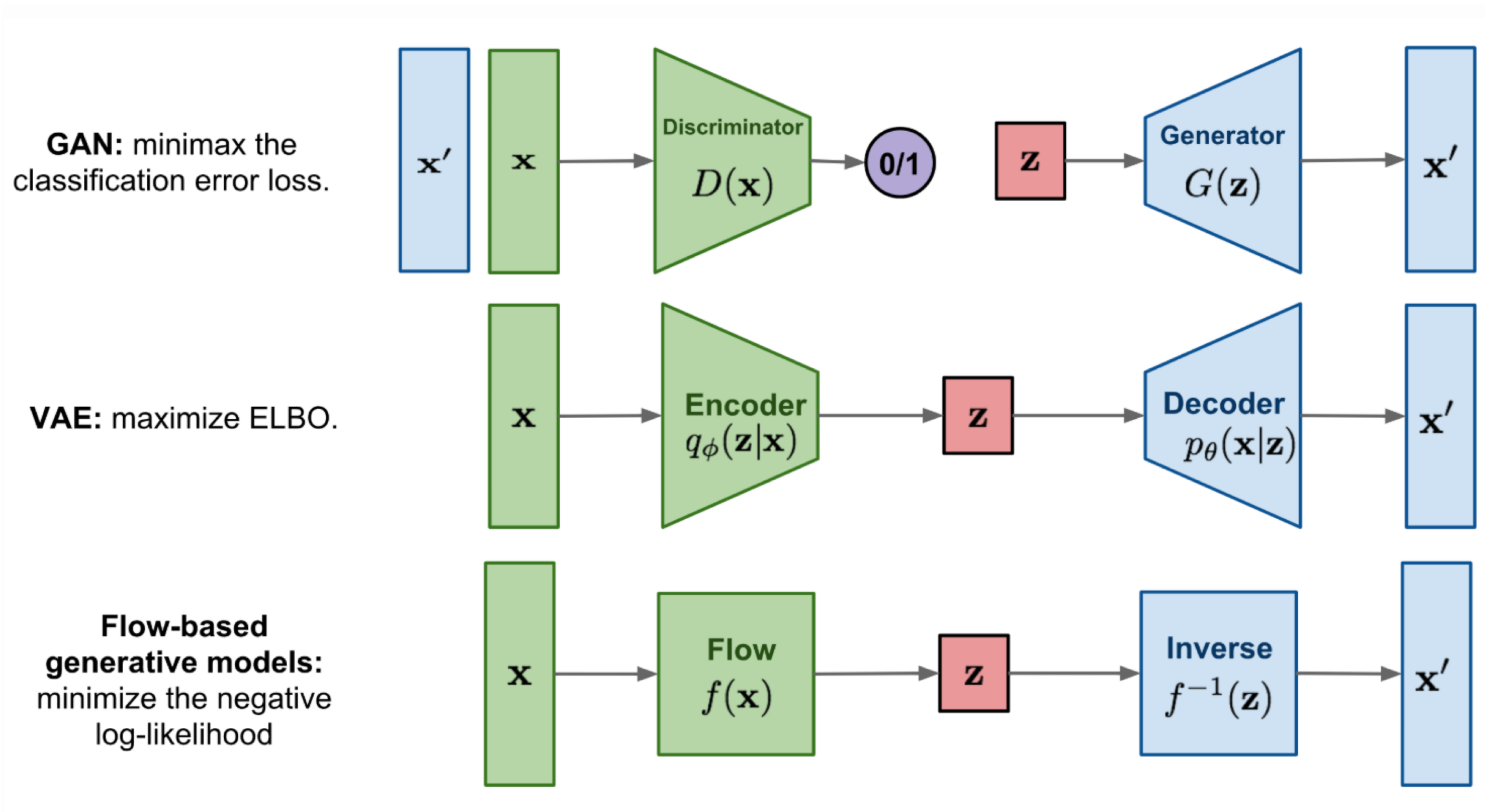
Generated samples ~ $p_{model}(x)$

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

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

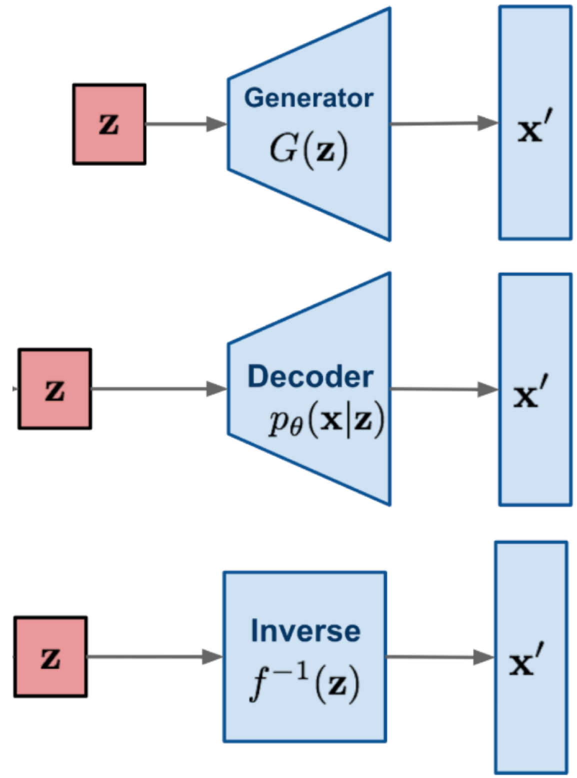
GAN: minimax the classification error loss.

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

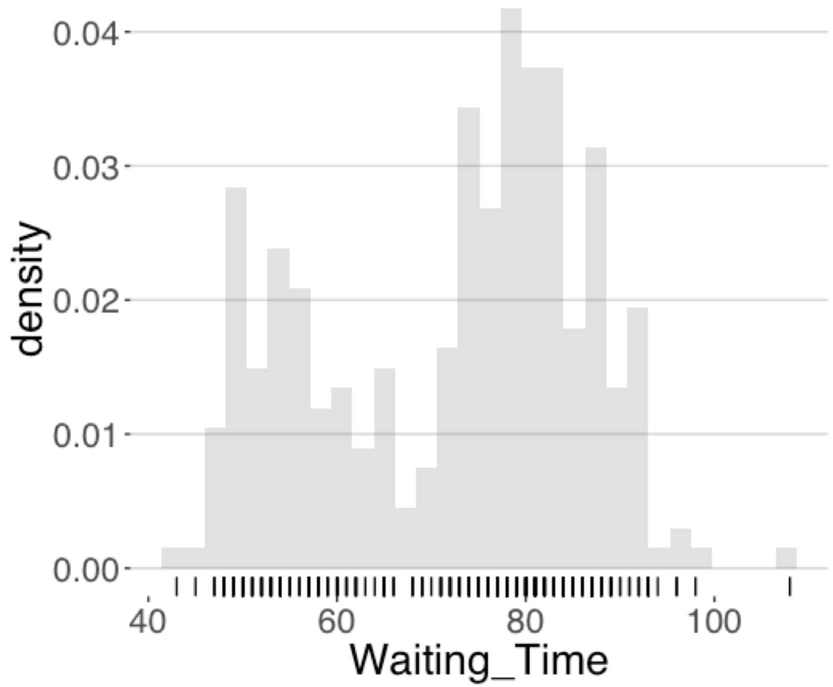
Theory: Name Some Distributions

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

These are the distributions we have in our Toolbox.

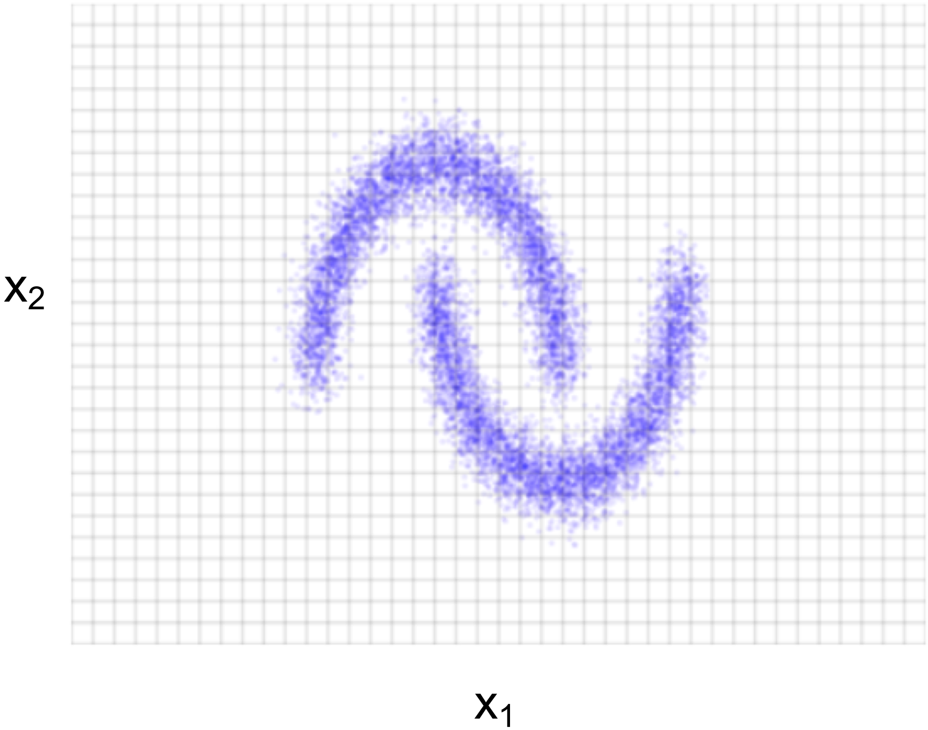
Is the reality like this?

Reality: Data (1-D)



What distribution can you use?

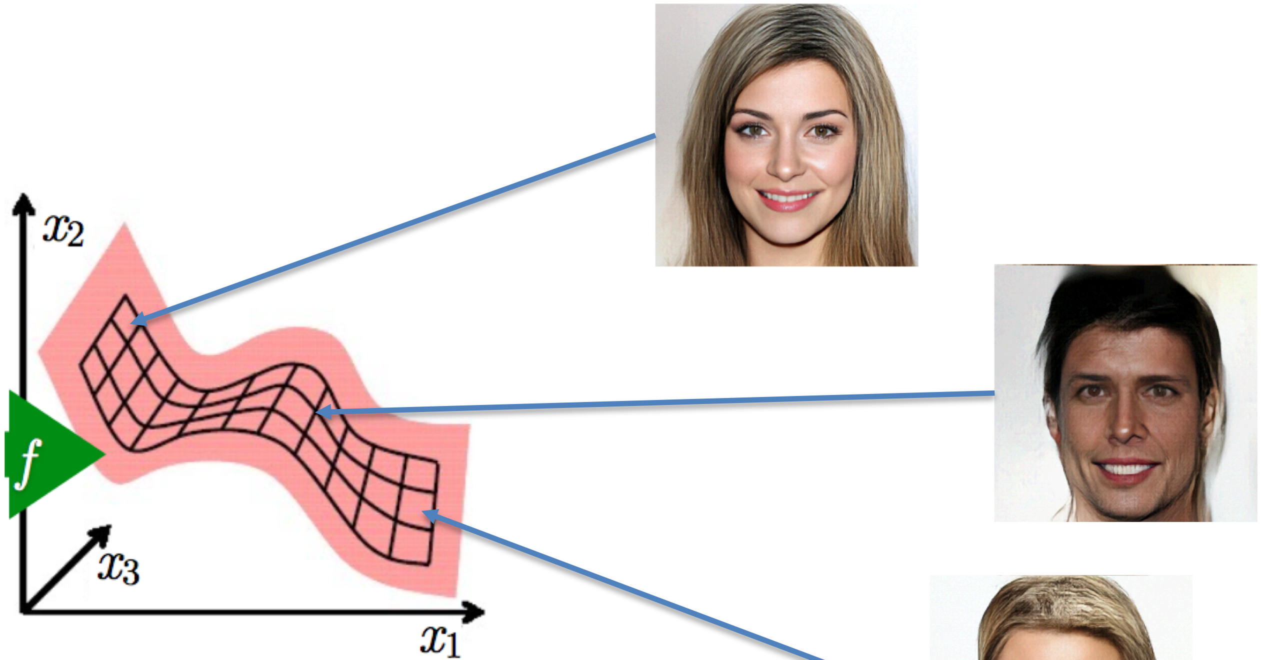
Reality: Data (2-D)

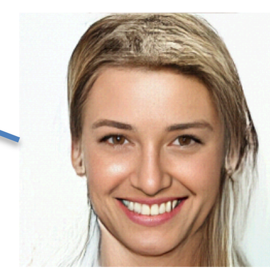


What distribution can you use?

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

3 data points **sampled** from the high dimensional distribution





This should be 196’608 dimensions
and not 3!

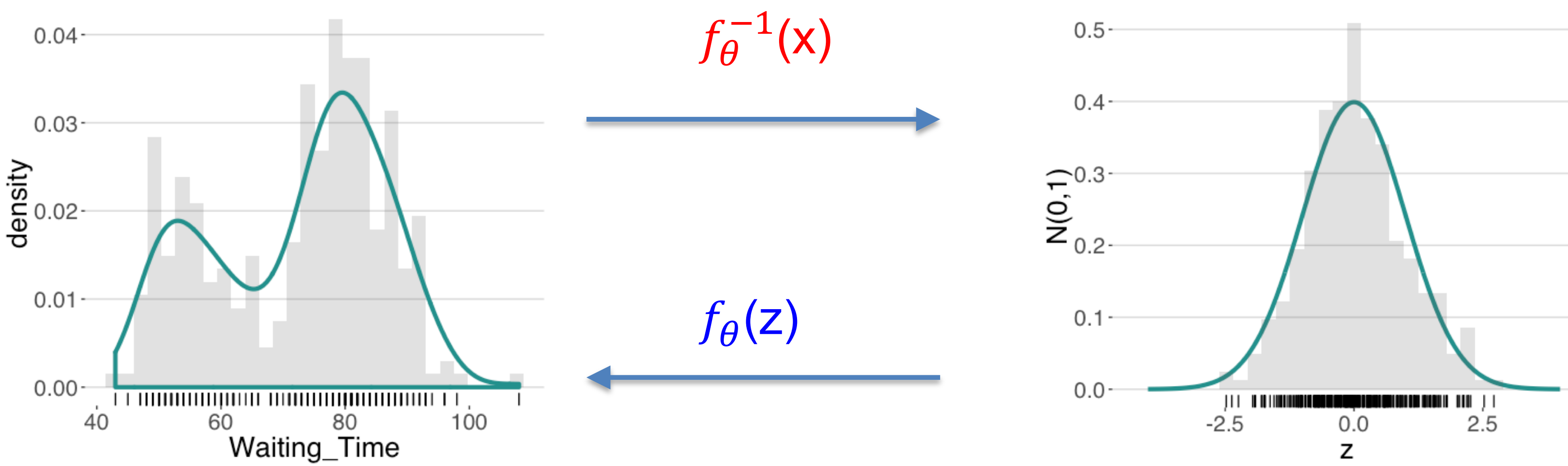
What distribution can you use?

Aproches 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)
- Copulas (since yesterday)
  - Limited to some 10 or 100 dimensions
- MCMC
  - Allows to **sample** from complicated distributions
- GANs (only have an *implicit* estimation can sample from $p(X)$)
- VAE (only have an approximation to $p(X)$)
  - $\log\big(p(x)\big) = L^v + \sout{D_{KL}(q(z|x)||p(z|x))}$ 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 $N(0,1)$

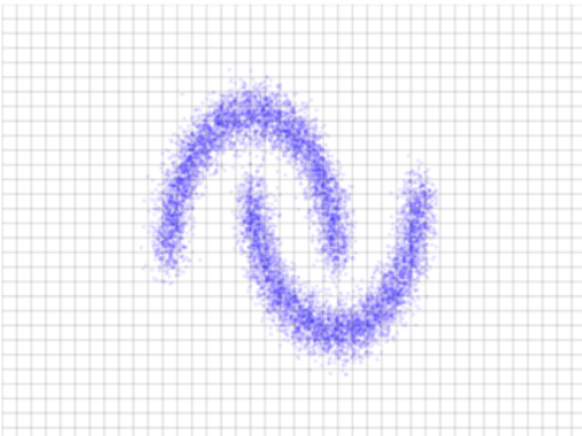
- Sampling from p(x): sample $z^* \sim \pi(z)$ then transform it via f_θ(z*)
- Density of *x**: calculate z*= $f_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

Main Idea of Normaliuing Flows

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

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

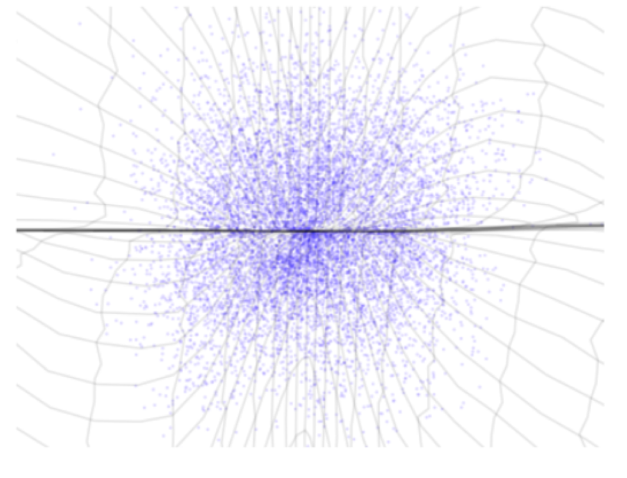
x_2



x_1

f_θ^{-1}(x) →

← 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 $\mathbb{R}^{196608}$

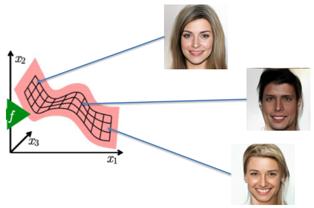
f_θ^{-1}(x)

x_1, x_2, x_{196608}~strange_function

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

With many correlations

f_θ(u)



pdf p(x)

pdf $\pi(z)$

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

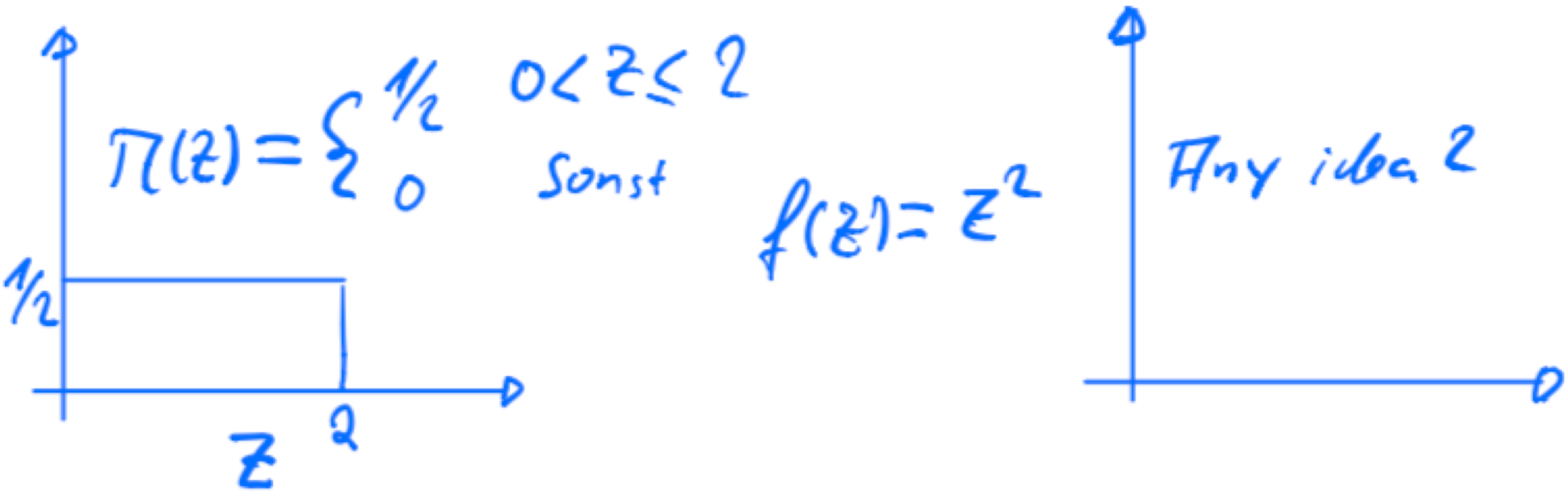
- Sampling from p(x): sample $z^* \sim \pi(z)$ then transform it via f_θ(z*)
- Density of *x**: calculate z*= $f_\theta^{-1}(x^*)$ and evaluate $N(z^*; 0,1)$

Transformation of Variables -- Some math

Simple Transformation

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

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

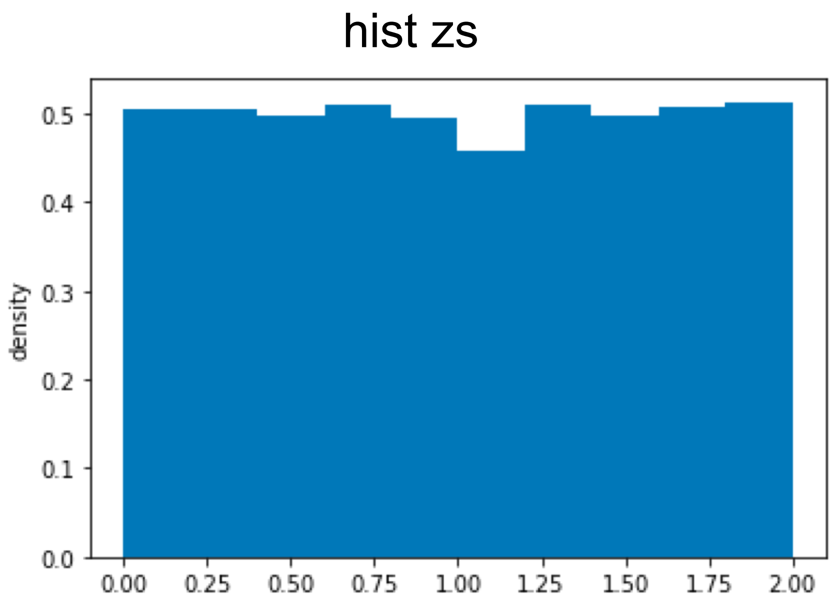


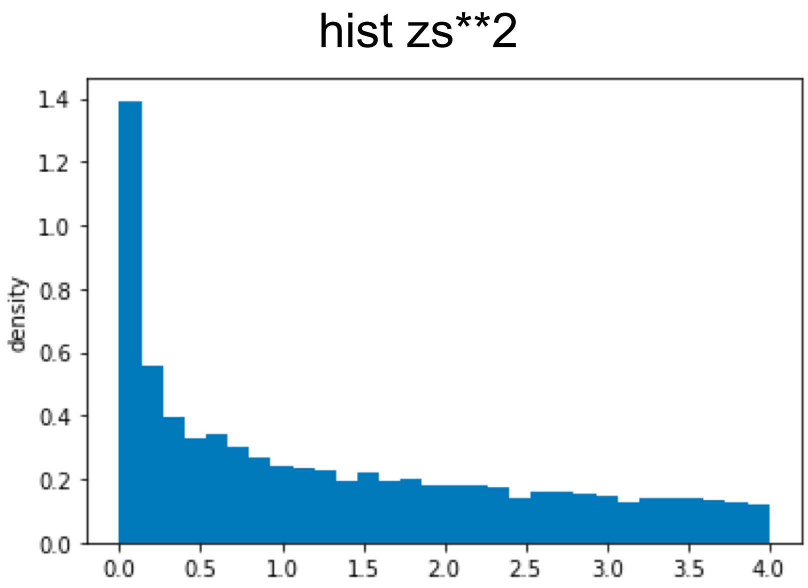
Try to come up with an answer, how is z distributed?

Try it

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

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

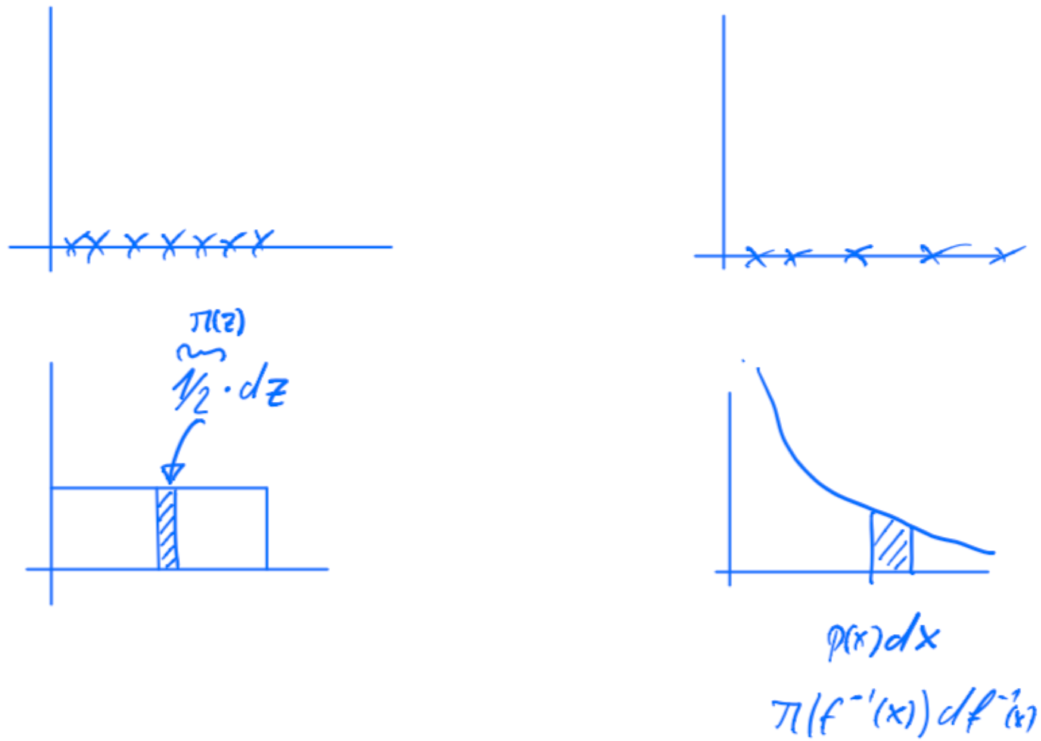




What happened?
Probability Mass needs to be conserved

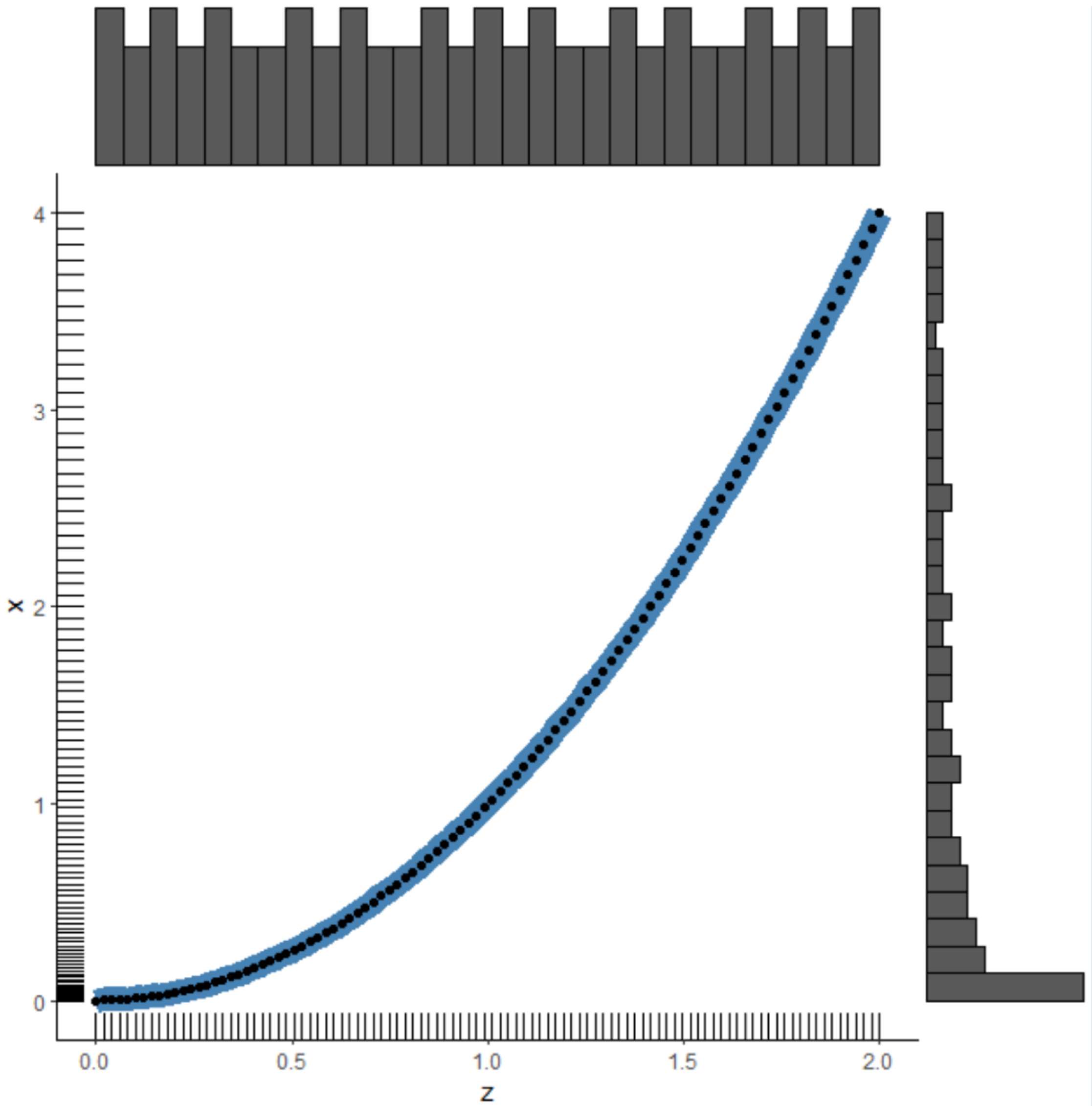
Think of samples

Think of mass
needs to be conserved



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

Annother View



1-D

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

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

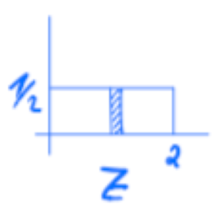
$$x = f(z) \Rightarrow z = f^{-1}(x)$$

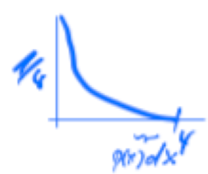
$$\Rightarrow \boxed{p(x) = \pi(f^{-1}(x))\left|\frac{df^{-1}(x)}{dx}\right|}$$

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

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

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





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

du and dx are positive by definition.

Definition in TFP

Listing 6.tfb2: The first bijector

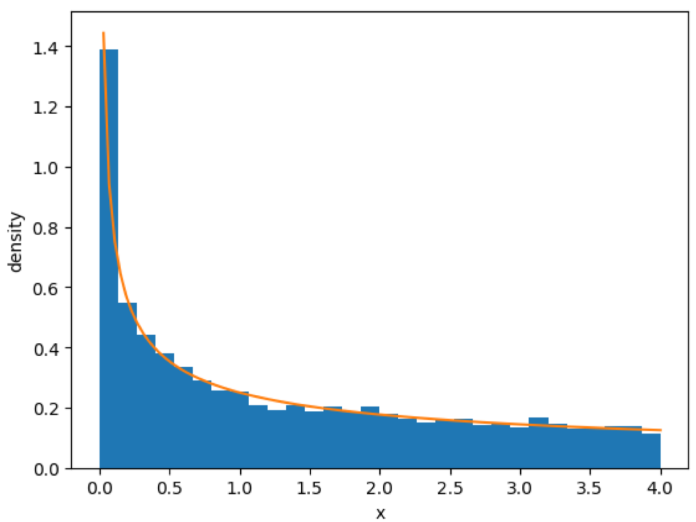
```
tfb = tfp.bijectors
g = tfb.Square() #A
g.forward(2.0)   #B
g.inverse(4.0)   #C

#A This is a simple bijector going from z → z**2
#B Yields 4
#C Yields 2
```

Listing 6.tfb3: The simple example in TFP

```
# 10           20        30          40        50          55
#1234567890123456789012345678901234567890123456789012341234
g = tfb.Square() #A
db = tfd.Uniform(0.0,2.0) #A2
mydist = tfd.TransformedDistribution( #B
    distribution=db, bijector=g)

xs = np.linspace(0.001, 5,1000)
px = mydist.prob(xs) #C
```



https://github.com/tensorchiefs/dl_book/blob/master/chapter_06/nb_ch06_03.ipynb

Learning to flow

- How probable (well density) is a data point xi

$$p_x(x_i) = p_z(g^{-1}(x_i))$$

- All Data points

$$\prod_{i=1}^{n} p_x(x_i)$$

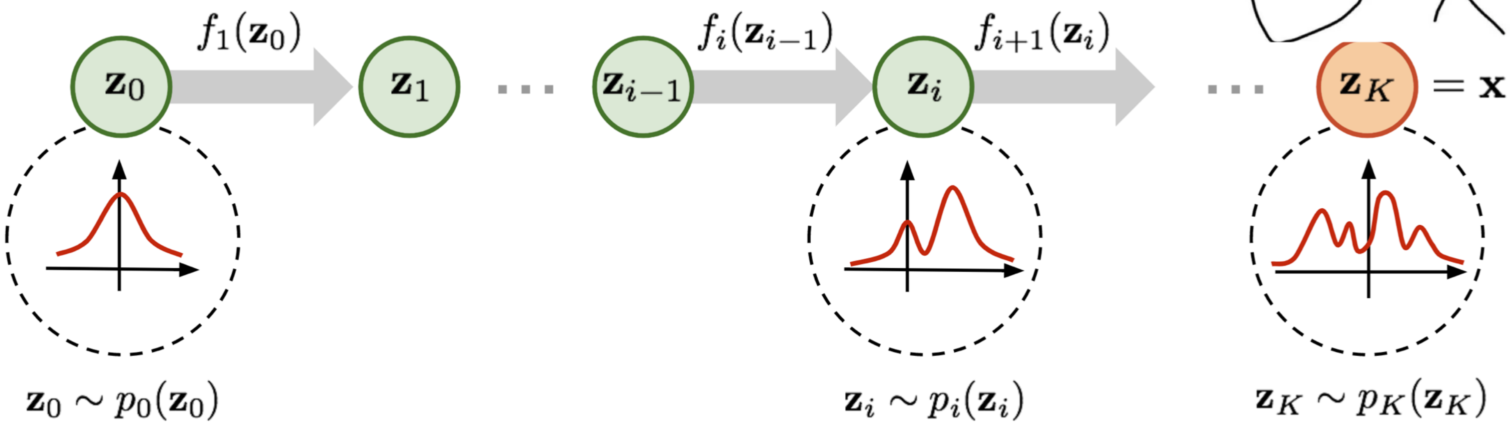
Tune the parameter(s) θ of the model M so that (observed) data is most likely!

- Affine linear

$$g(x) = a \cdot z + b$$

- https://github.com/tensorchiefs/dl_book/blob/master/chapter_06/nb_ch06_03.ipynb

Chaining



z0->z1->z2

$$p_{z_1}(z_1) = p_{z_0}(z_0) \cdot |g_1'(z_0)|^{-1}$$

$$p_{z_2}(z_2) = p_{z_1}(z_1) \cdot |g_2'(z_1)|^{-1}$$

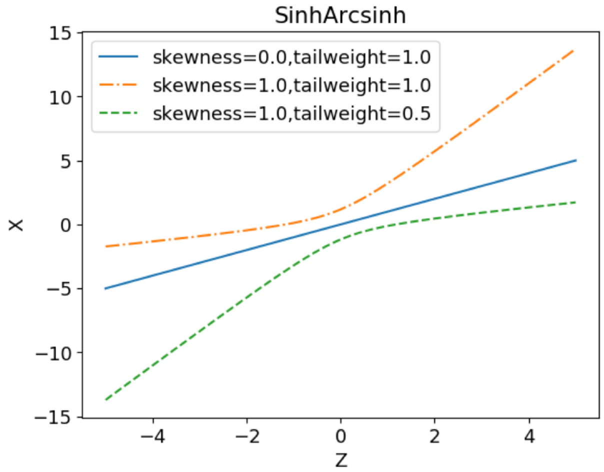
$$p_{z_2}(z_2) = p_{z_0}(z_0) \cdot |g_1'(z_0)|^{-1} \cdot |g_2'(z_1)|^{-1}$$

$$log(p_{z_2}(z_2)) = log(p_{z_0}(z_0)) - log(|g_1'(z_0)|) - log(|g_2'(z_1)|)$$

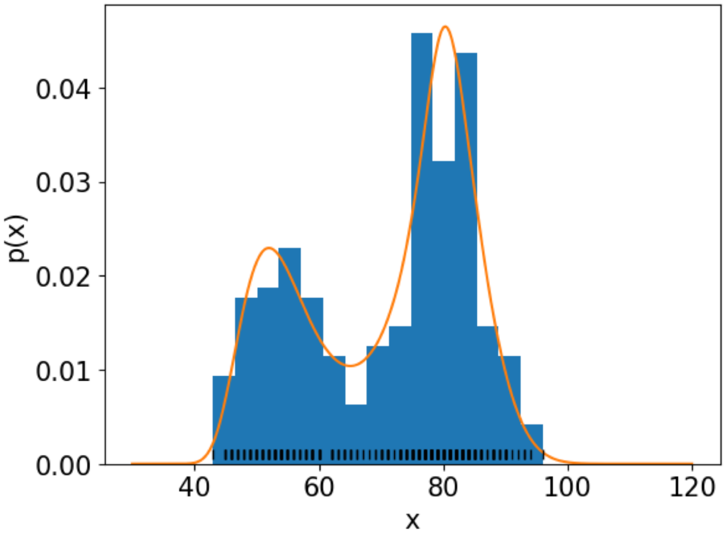
$$log(p_x(x)) = p_{z_0}(z_0) - \sum_{i=1}^{k} log(|\frac{dg_i(z_{i-1})}{dz_{i-1}}|)$$

Practical example

- Need non-linearity



- Geyser Data



Going to higher dimensions

Transformation in high dimensions

$$g(z) = \begin{pmatrix} g_1(z_1, z_2, z_3) \\ g_2(z_1, z_2, z_3) \\ g_3(z_1, z_2, z_3) \end{pmatrix}$$

$$\frac{\partial g(z)}{\partial z} = \begin{pmatrix} \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_1(z_1,z_2,z_3)}{\partial z_3} \\ \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_2(z_1,z_2,z_3)}{\partial z_3} \\ \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_1} & \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_2} & \frac{\partial g_3(z_1,z_2,z_3)}{\partial z_3} \end{pmatrix}$$

$$p_x(x) = p_z(z) \cdot \left|det\left(\frac{dg(z)}{dz}\right)\right|^{-1}$$

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 $f(z)$

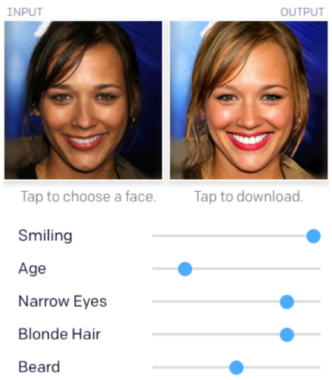
Flows with networks

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 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 $\frac{\partial f_1(z)}{\partial z_2} \overset{!}{\triangleq} 0$
    - ➔ $f_1(z) = f_1(z_1, \cancel{z_2, z_3}),\ f_d(z) = f_1\left(z_1, \dots, z_d, \cancel{z_{d+1}, z_{d+2,\dots}}\right)$
  - 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)

Simple Solution

- Blackboard
  - Netz
  - Invertierbarkeit (pice of cake)
  - Jacobi Determinante

Simple Solution

- Blackboard
  - Netz
  - Invertierbarkeit (pice of cake)
  - Jacobi Determinante

j

i

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ e & e & \exp\left(\alpha_1(z_1,z_2)\right) & 0 & 0 \\ e & e & e & \exp\left(\alpha_2(z_1,z_2)\right) & 0 \\ e & e & e & e & \exp\left(\alpha_3(z_1,z_2)\right) \end{pmatrix}$$

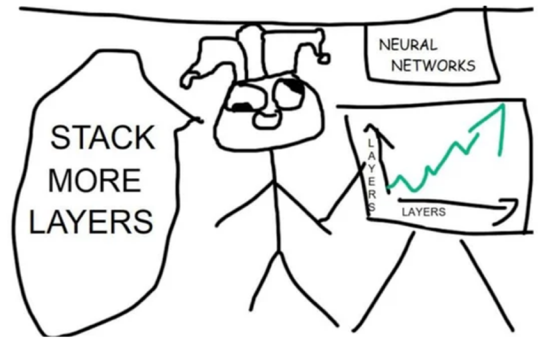
$\frac{\partial .}{\partial z_1}$ $\frac{\partial .}{\partial z_2}$ $\frac{\partial .}{\partial z_3}$ $\frac{\partial .}{\partial z_4}$ $\frac{\partial .}{\partial z_5}$

$x_1 = z_1$

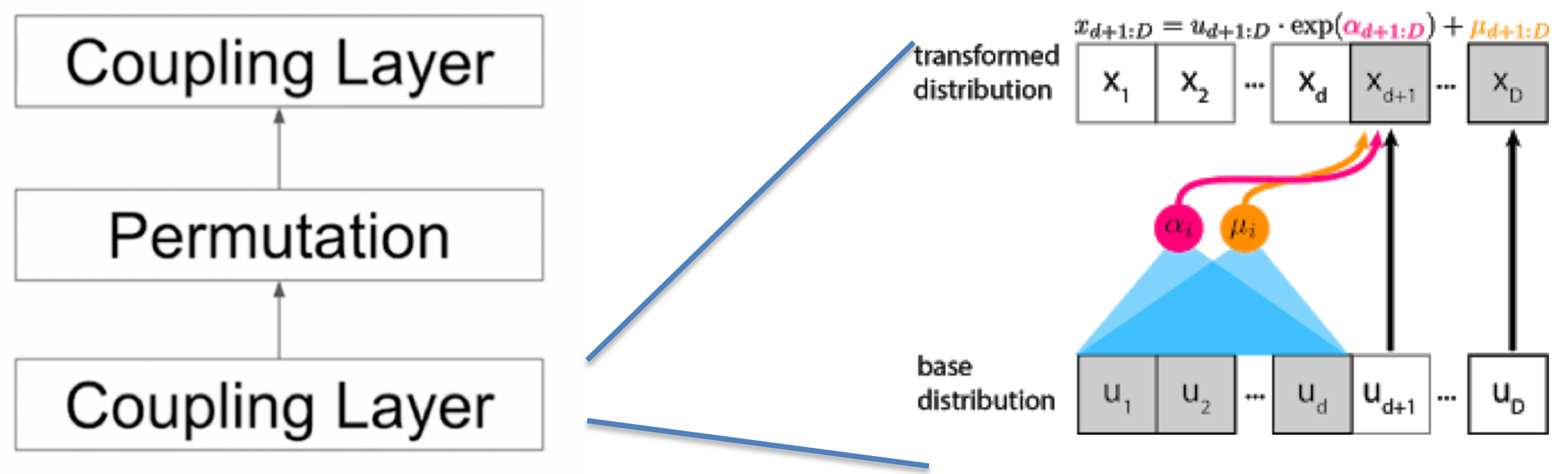
$x_2 = z_2$

e=don’t care

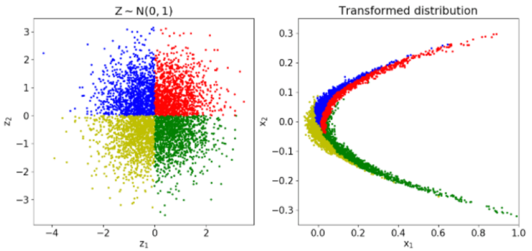
Stack more Layers (Permutation)



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



Example



Listing 6.realnvp: The simple example in TFP

```
#1234567890123456789012345678901234567890123456789012345678901234

bijectors=[] #A

h = 32
for i in range(5): #B
    net = tfb.real_nvp_default_template(hidden_layers=[h, h])#C
    bijectors.append(tfb.RealNVP(shift_and_log_scale_fn=net,num_masked=num_masked))#D
    bijectors.append(tfb.Permute([1,0])) #E
    self.nets.append(net)
bijector = tfb.Chain(list(reversed(bijectors[:-1])))

self.flow = tfd.TransformedDistribution(#F
    distribution=tfd.MultivariateNormalDiag(loc=[0., 0.]),
    bijector=bijector)
```

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 (Details of the affine coupling layer)

$$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$$
$$(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$$
$$\mathbf{s} = \exp(\log \mathbf{s})$$
$$\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$$
$$\mathbf{y}_b = \mathbf{x}_b$$
$$\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$$

x has dimensions e.g. (128x128x12)

x_a has dimensions e.g. (128x128x6)

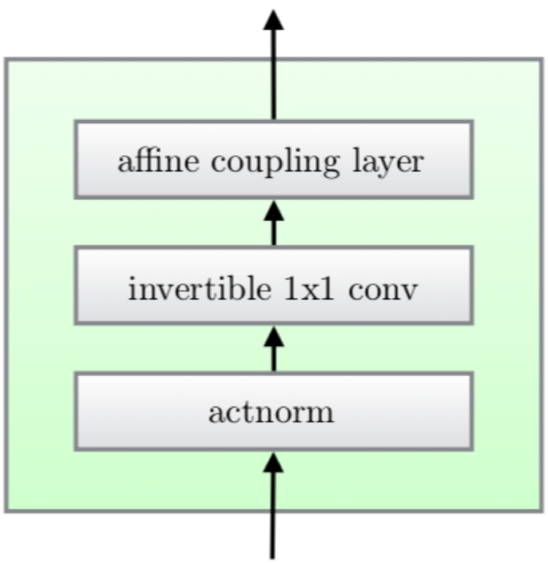
x_b has dimensions e.g. (128x128x6)

NN is CNN, **s** is vector with length = depth of x_a

See: Glow paper for further details

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



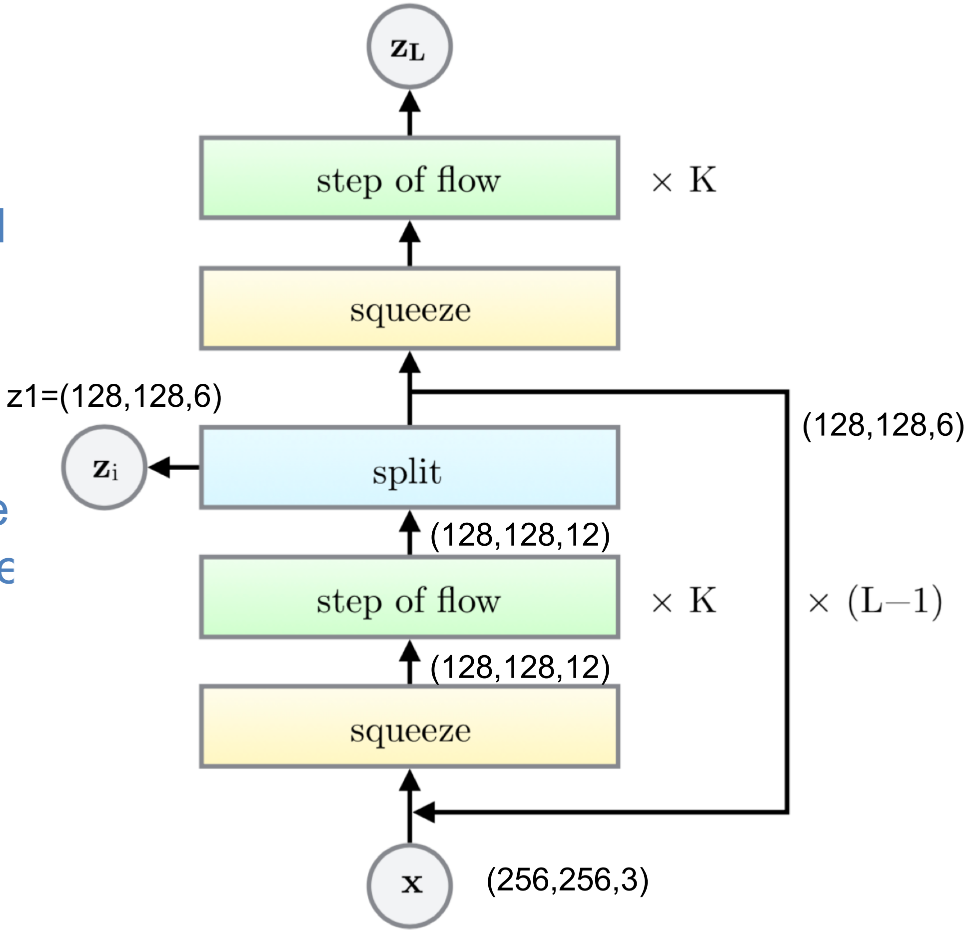
(a) One step of our flow.

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



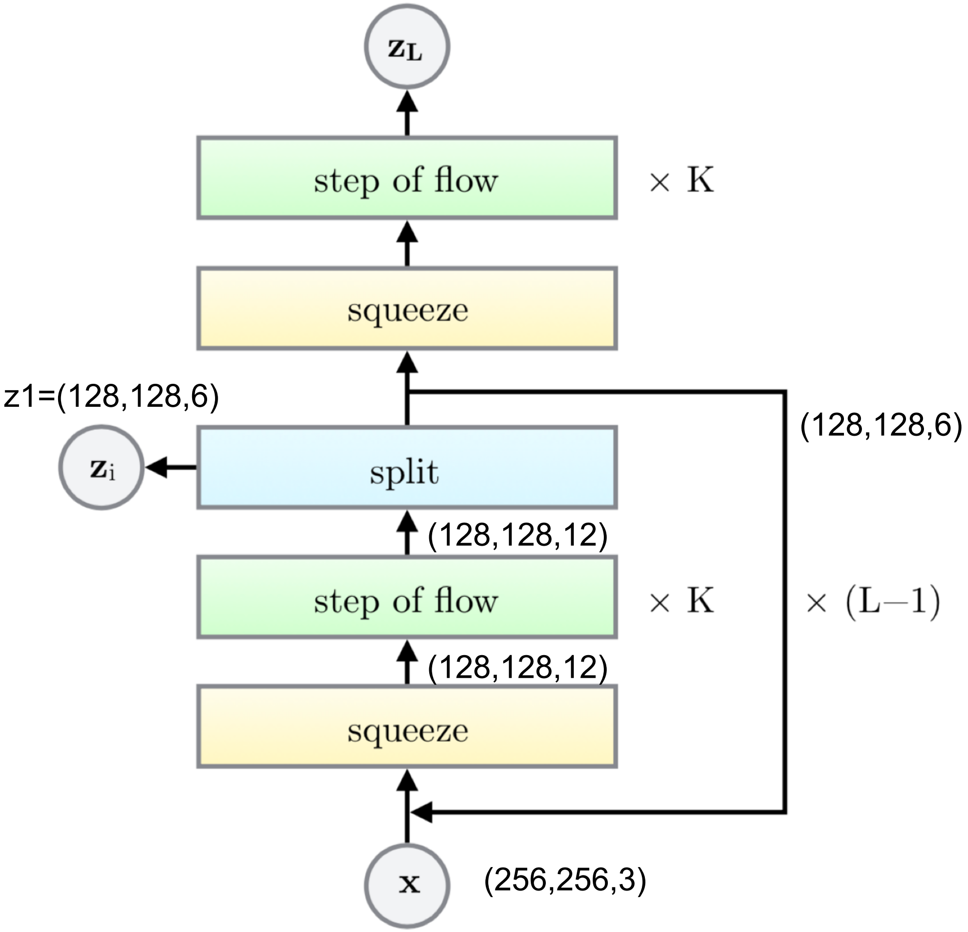
Multiscale Architecture

- 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)
```

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

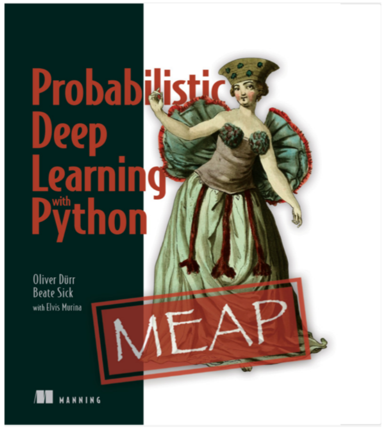


Demo

- 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

Some interesting reads and talks



Coming soon

- 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?