Simulation and inference in neuroscience

Lecture 8: Normalising flows

March 2025

Pedro Gonçalves
goncalveslab.sites.vib.be/en

Philipp Berens, Jonas Beck
https://hertie.ai/data-science/team

Simulation and inference in neuroscience — PJ Goncalves, P Berens, J Beck @CCC BY-NC-SA 3.0

Motivation

- Many tasks in Probabilistic Machine Learning require flexible models for (conditional) distributions, e.g., for SBI.
- Gaussian distributions are very convenient, but not flexible
- So: to make a flexible model, take a Gaussian, push it through a neural network.
- Gaussians + Change of Variables + Some Tricks = Normalising Flows
- Normalising Flows: Very useful for SBI (but also beyond)

Modelling complex distributions

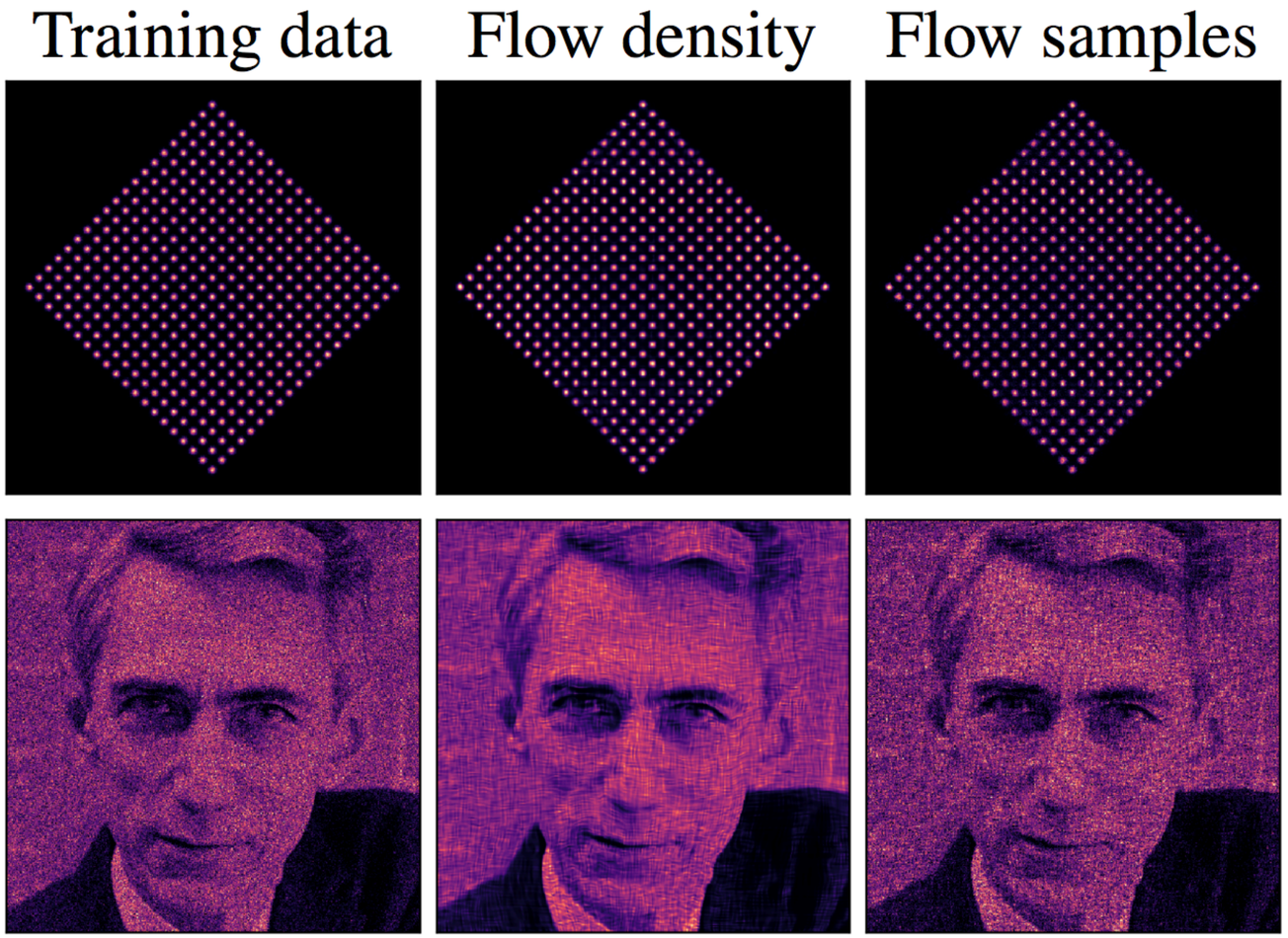


Figure taken from Durkan et al 2019

Modelling complex distributions

Figure taken from Murphy “Probabilistic Machine Learning: Advanced Topics”

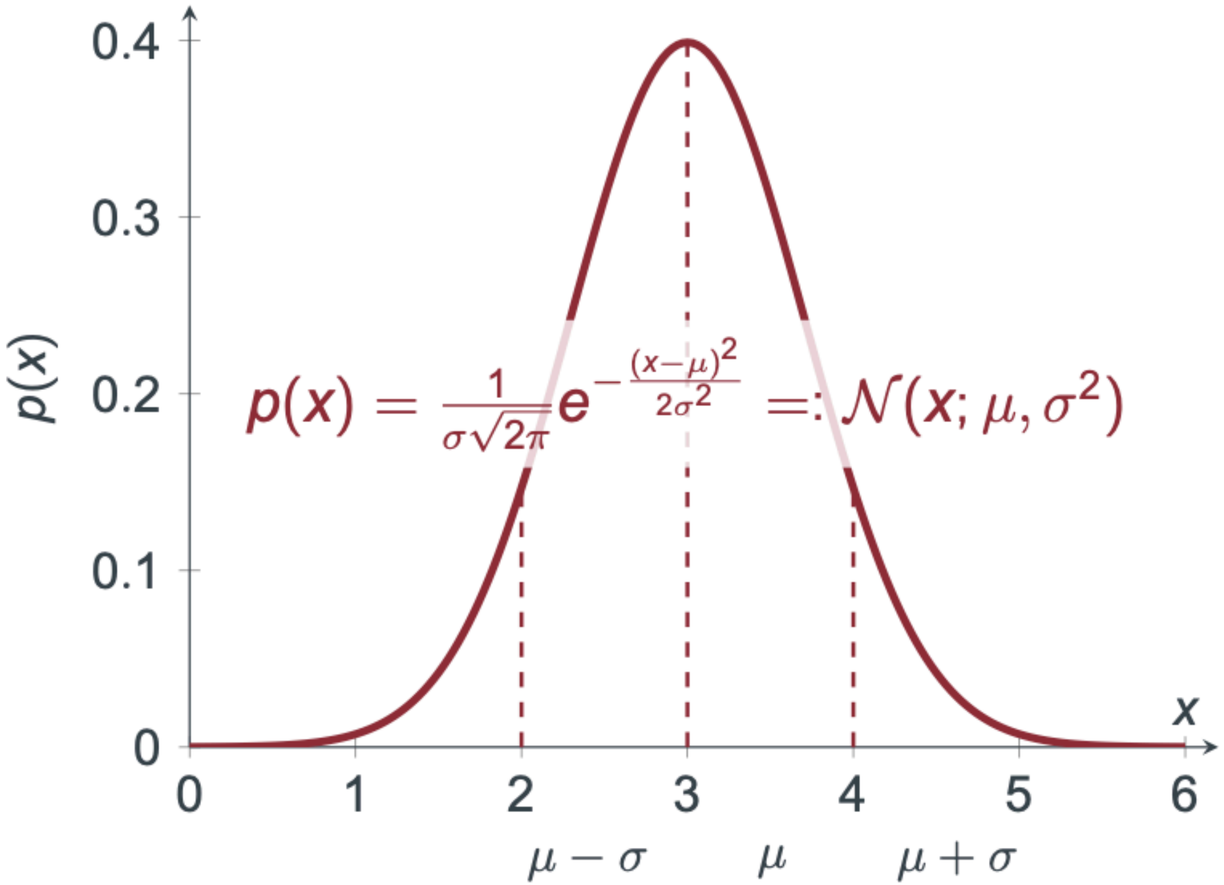
8.1 Normal distributions

Prob ML — JH. Macke, SoSe 2022 — Lecture 04: The Gaussian Distribution — © 2022 CC BY-NC-SA 3.0. Many ideas and slides originally by P Hennig

Why are Gaussians important?

- Gaussians are ubiquitous: Central Limit Theorem
- The Gaussian distribution is a central building block of many advanced models.
- The Gaussian distribution has many computationally attractive properties:
  - products of Gaussian densities are (proportional to) Gaussian densities
  - linear maps of Gaussian variables are Gaussian variables
  - marginals of Gaussians are Gaussians
  - linear conditionals of Gaussians are Gaussians
- The Gaussian distribution provides a link between Linear Algebra and Probability Theory. Computers are good at Linear Algebra!

First: the univariate Gaussian



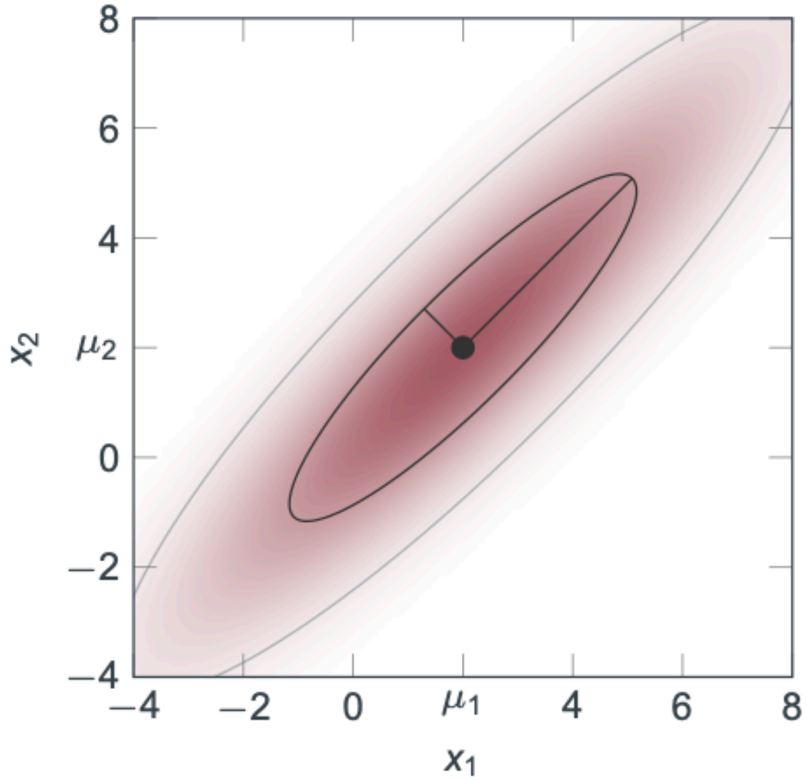
μ **mean** of x

σ^2 **variance** of x

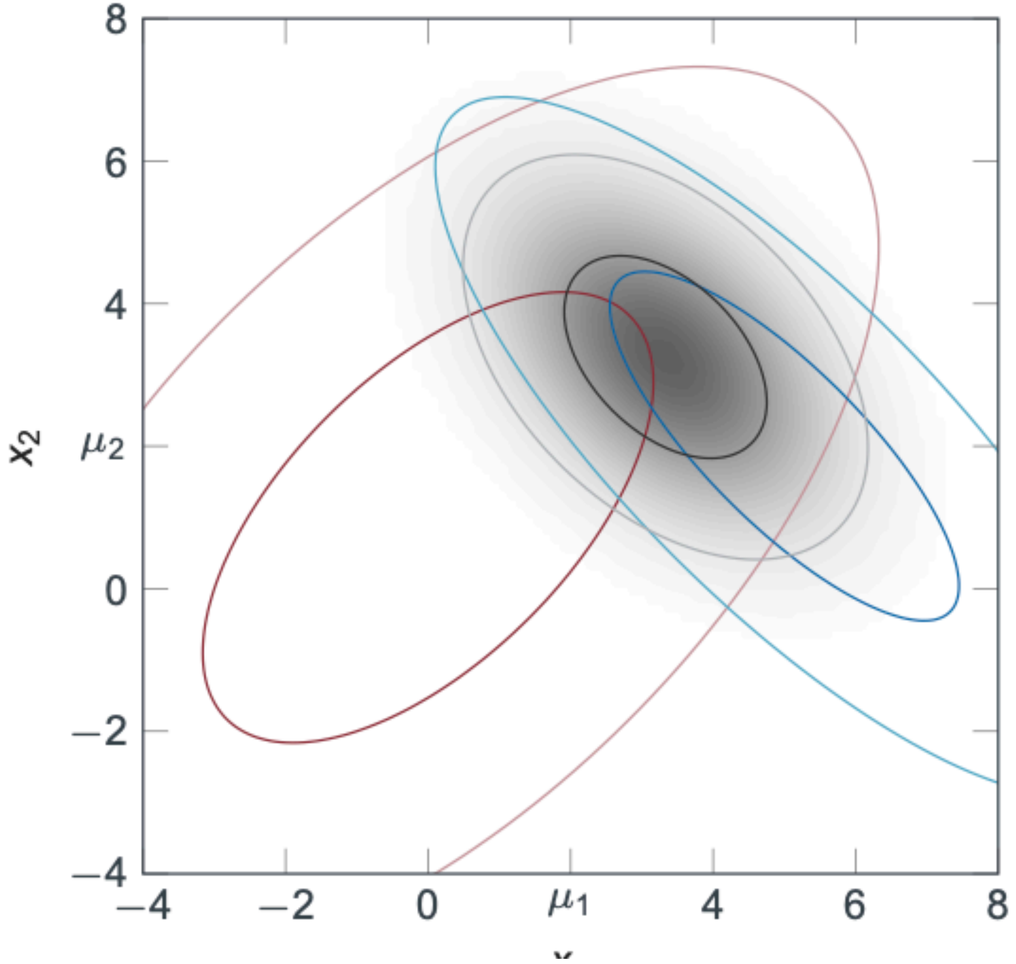
σ **standard deviation** of x

The multivariate Gaussian

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\intercal}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \quad \boldsymbol{x}, \boldsymbol{\mu} \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n\times n} \text{ spd.}$$



The product of two Gaussian density is an (unnormalised) Gaussian density: closure under multiplication



To multiply Gaussian densities, add the natural parameters

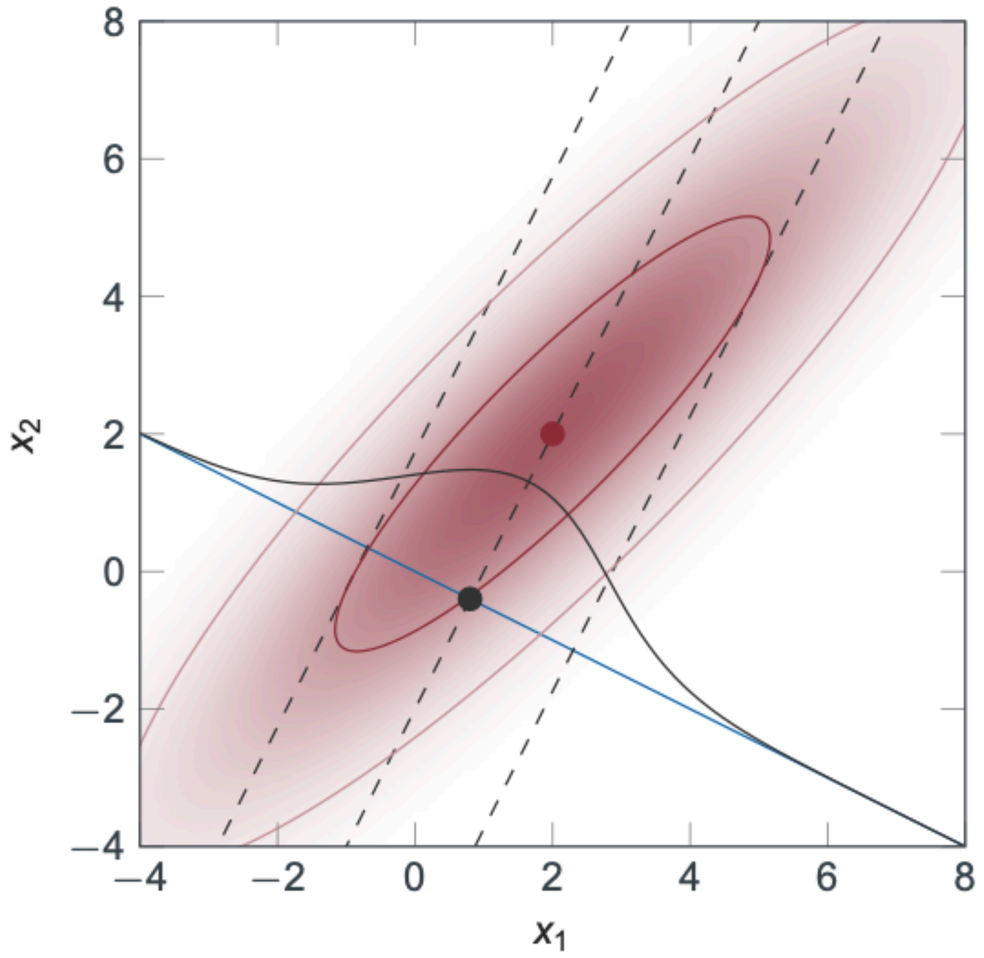
$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{a}, A)\mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, B) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{c}, C)Z$$

$$C = (A^{-1} + B^{-1})^{-1}$$

$$\boldsymbol{c} = C(A^{-1}\boldsymbol{a} + B^{-1}\boldsymbol{b})$$

$$Z = \mathcal{N}(\boldsymbol{a}; \boldsymbol{b}, A + B)$$

Linear Transformations of Gaussians are Gaussians: closure under linear maps



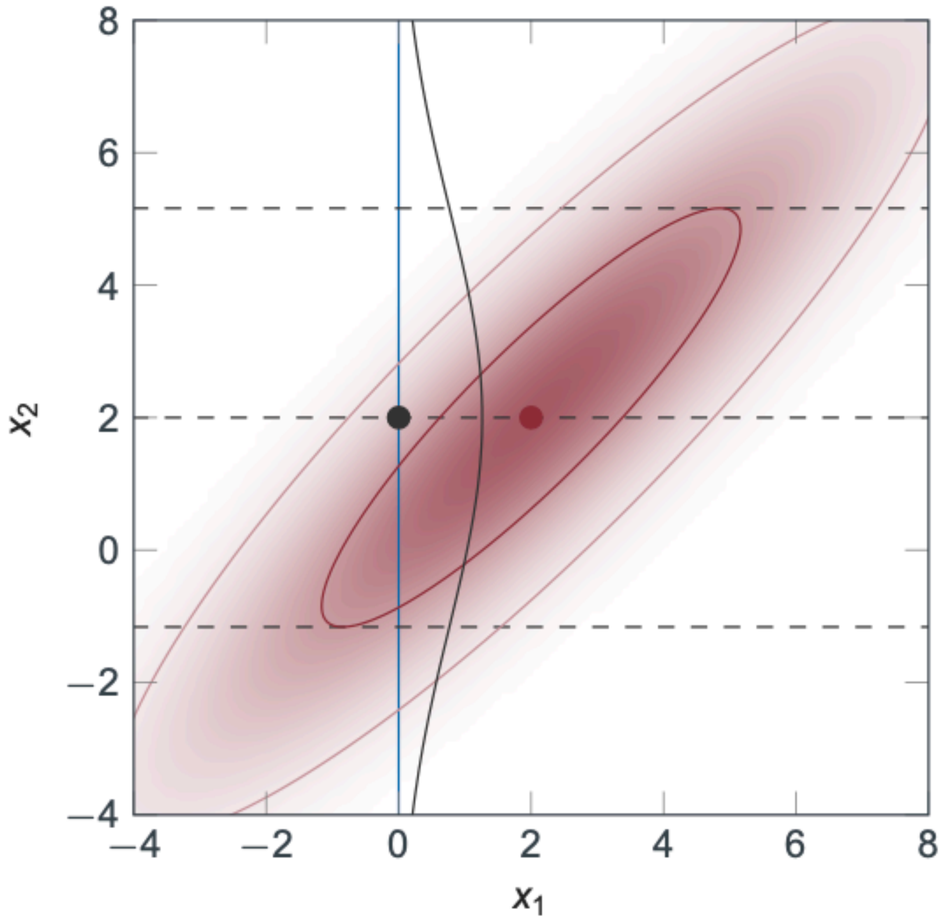
To linearly transform a Gaussian variable, transform the parameters

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \Sigma)$$
$$\Rightarrow \quad p(A\boldsymbol{z}) = \mathcal{N}(A\boldsymbol{z}, A\boldsymbol{\mu}, A\Sigma A^\intercal)$$

whenever $A\Sigma A^\intercal$ is full rank.

Also true for projections onto vectors!

Marginals of Gaussians are Gaussians: closure under marginalisation



$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \Sigma) \quad \Rightarrow \quad p(A\boldsymbol{z}) = \mathcal{N}(A\boldsymbol{z}, A\boldsymbol{\mu}, A\Sigma A^{\mathsf{T}})$$

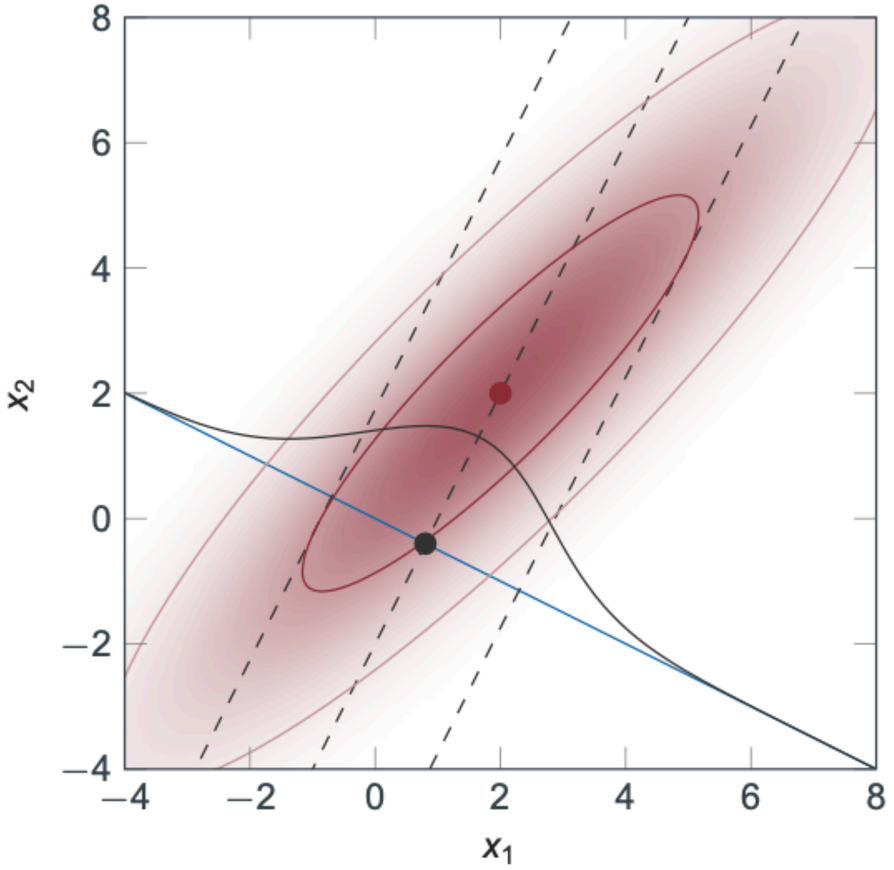
$$\text{choose } A = \begin{pmatrix} \mathbf{I} & 0 \end{pmatrix}$$

$$\int \mathcal{N}\left[\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix}; \begin{pmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{pmatrix}, \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix} \right] d\boldsymbol{y} = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_x, \Sigma_{xx})$$

- this is the sum rule

$$\int p(x, y)\, dy = \int p(y \mid x) p(x)\, dy = p(x)$$

Cuts through Gaussians are Gaussians: closure under linear conditioning



$$p(\boldsymbol{x} \mid A\boldsymbol{x} = \boldsymbol{y}) = \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{y})}$$
$$= \mathcal{N}\big(\boldsymbol{x}; \mu + \Sigma A^\intercal (A\Sigma A^\intercal)^{-1}(\boldsymbol{y} - A\mu),$$
$$\Sigma - \Sigma A^\intercal (A\Sigma A^\intercal)^{-1} A\Sigma\big)$$

- this is the product rule
- so Gaussians are closed under the rules of probability

The Gaussian has very useful closure properties

- products of Gaussian densities are proportional to Gaussian densities

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{a}, A)\mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, B)$$
$$= \mathcal{N}(\boldsymbol{x}; \boldsymbol{c}, C)\mathcal{N}(\boldsymbol{a}; \boldsymbol{b}, A + B)$$
$$C := (A^{-1} + B^{-1})^{-1} \quad \boldsymbol{c} := C(A^{-1}\boldsymbol{a} + B^{-1}\boldsymbol{b})$$

- linear transformations of Gaussians are Gaussians

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \Sigma)$$
$$\Rightarrow \quad p(A\boldsymbol{z}) = \mathcal{N}(A\boldsymbol{z}, A\boldsymbol{\mu}, A\Sigma A^{\intercal})$$

whenever $A\Sigma A^{\intercal}$ is full rank

- marginals of Gaussians are Gaussians

$$\int \mathcal{N}\left[\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{pmatrix}; \begin{pmatrix}\boldsymbol{\mu}_x\\ \boldsymbol{\mu}_y\end{pmatrix}, \begin{pmatrix}\Sigma_{xx} & \Sigma_{xy}\\ \Sigma_{yx} & \Sigma_{yy}\end{pmatrix}\right] dy = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_x, \Sigma_{xx})$$

- (linear) conditionals of Gaussians are Gaussians

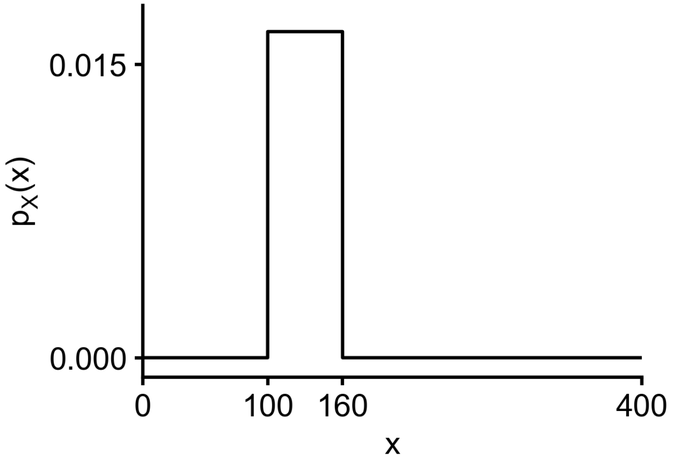
$$p(\boldsymbol{x} \mid \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_x + \Sigma_{xy}\Sigma_{yy}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y), \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}\right)$$

8.2 Change-of-variables on probability density functions

Example for change of variables: A simple transformation (example by Michael Deistler)

https://www.deutschland.de/en/topic/life/surviving-the-autobahn-ten-guidelines

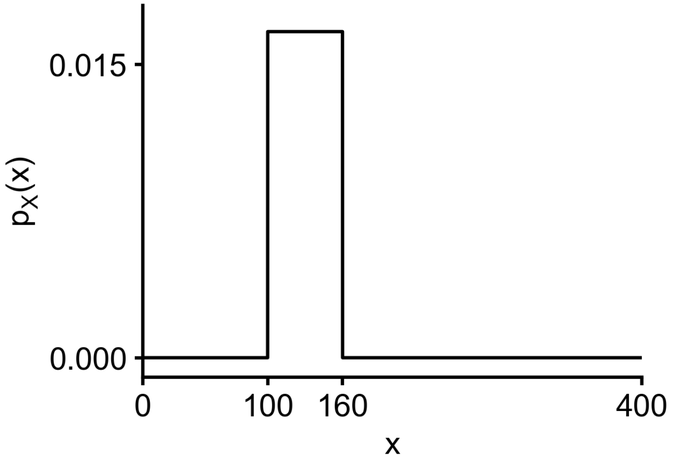
- Let the random variable X describe the velocity of cars on a German highway.
- Lets say the distribution of these velocities is uniform between 100 and 160 (km/h, but we will ignore units in this example).



Example for change of variables: A simple transformation (example by Michael Deistler)

https://www.deutschland.de/en/topic/life/surviving-the-autobahn-ten-guidelines

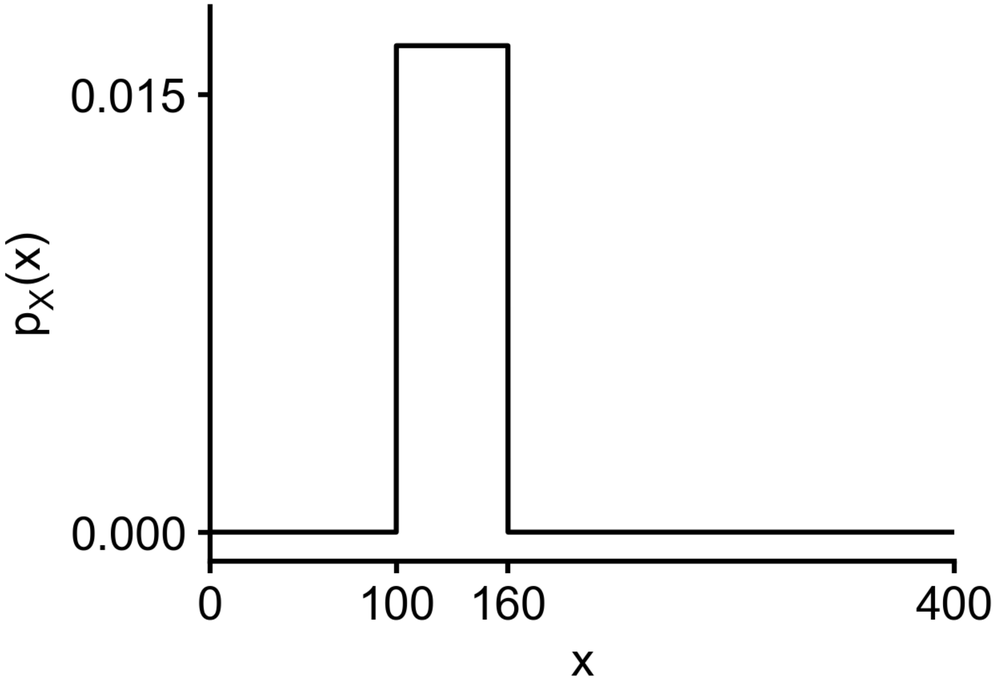
- Let the random variable X describe the velocity of cars on a German highway.
- Lets say the distribution of these velocities is uniform between 100 and 160 (km/h, but we will ignore units in this example).



- Let the random variable Y be the distance (in km) which cars travel within 2 hours. Then

$$\text{Distance} = \text{Velocity} \cdot \text{Time} \qquad Y = f(X) = 2X$$

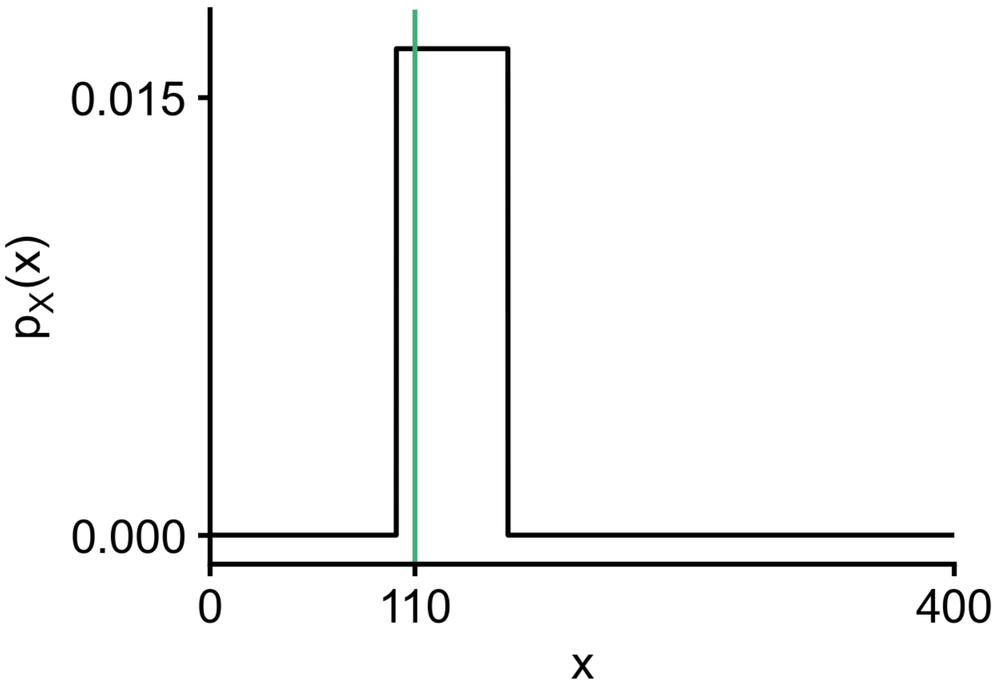
What does it mean to transform a random variable?



$$Y = f(X)$$

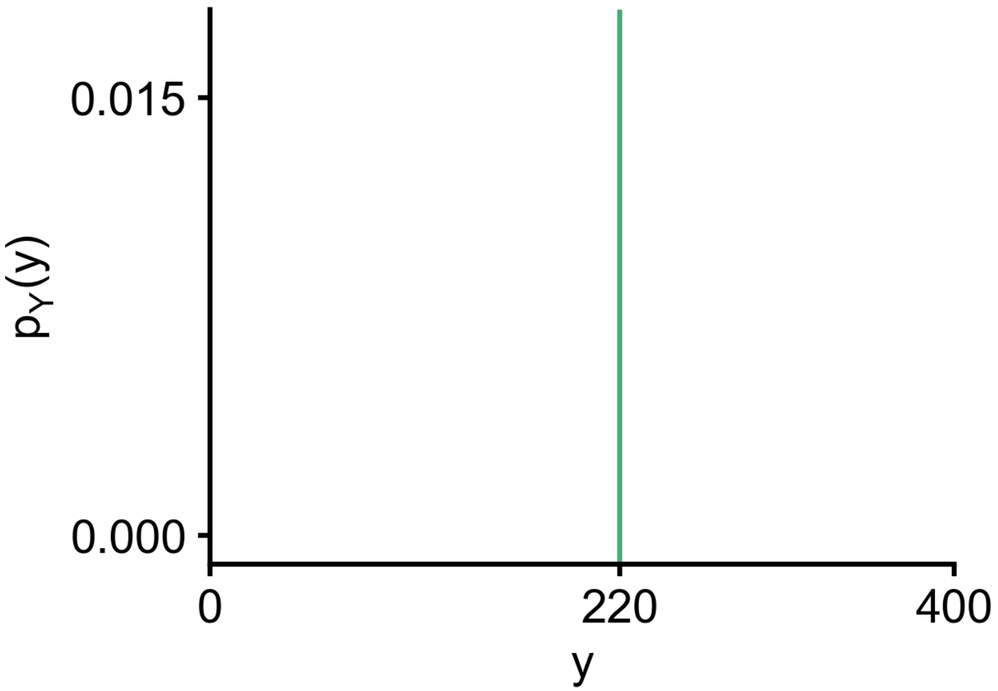
$$Y = 2X$$

What does it mean to transform a random variable?

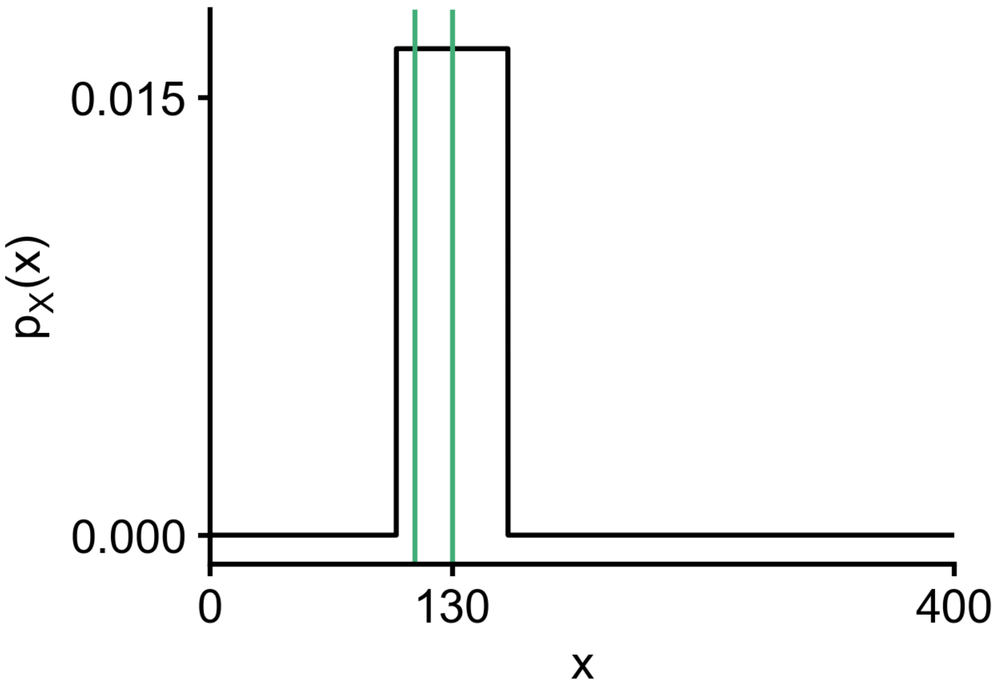


$$Y = f(X)$$

$$Y = 2X$$

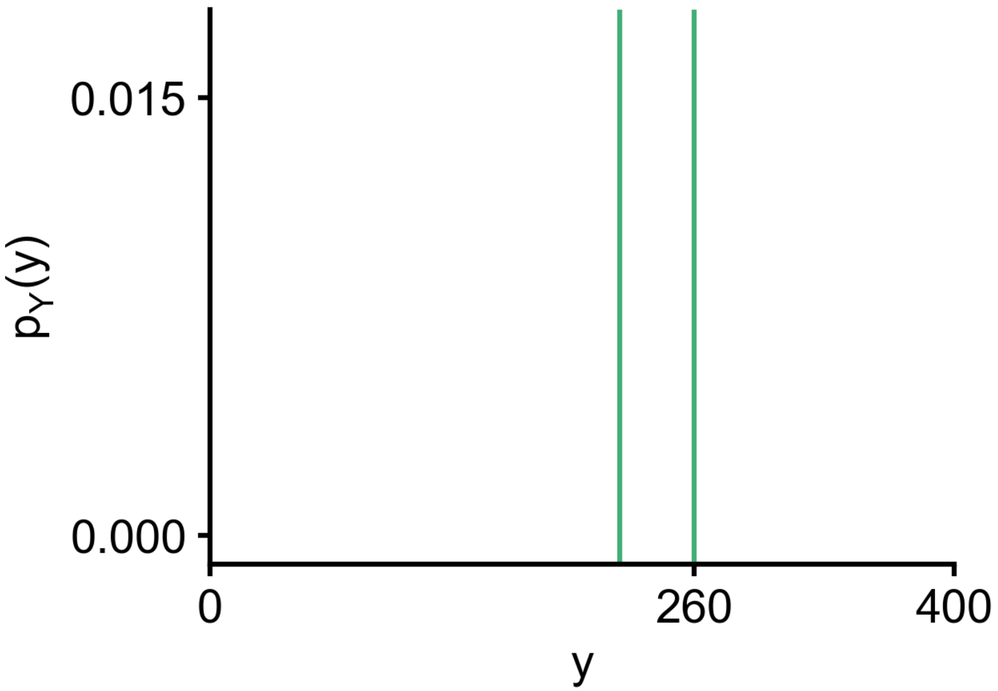


What does it mean to transform a random variable?

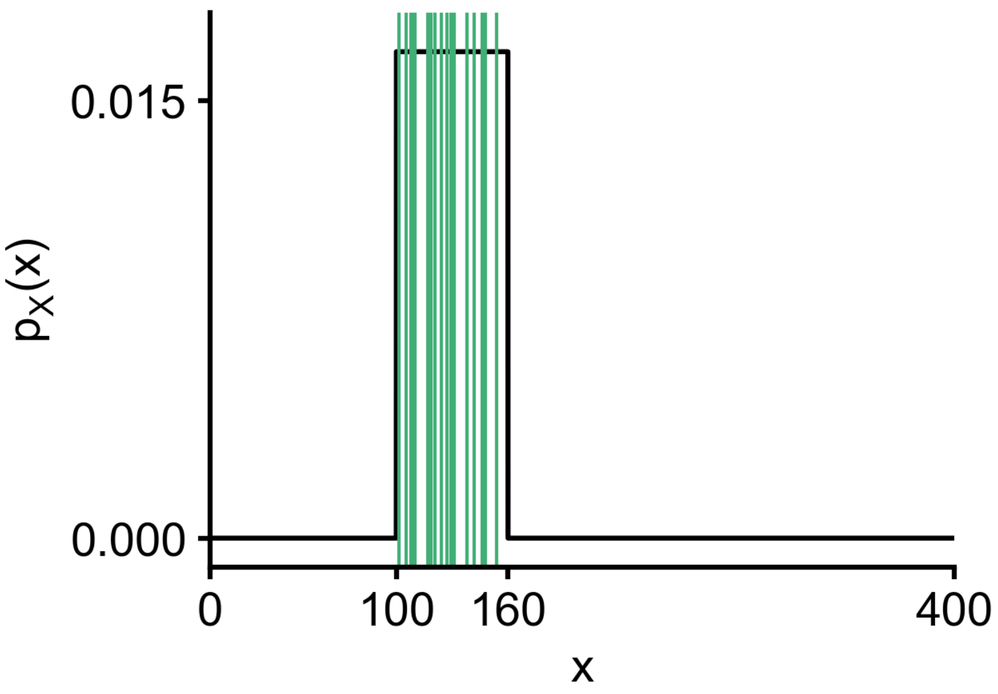


$$Y = f(X)$$

$$Y = 2X$$

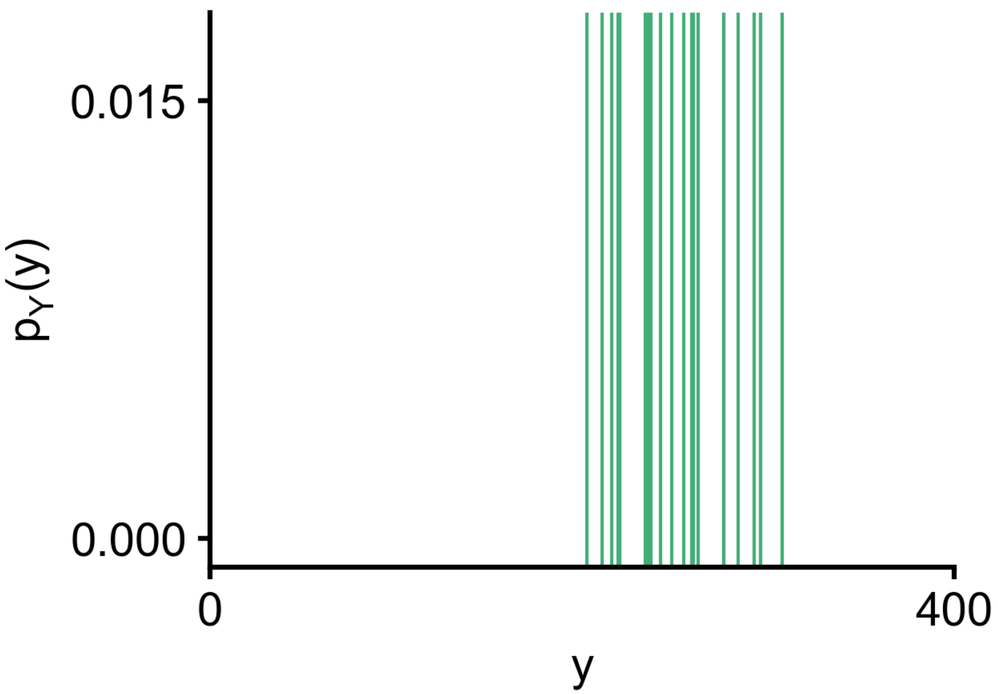


What does it mean to transform a random variable?

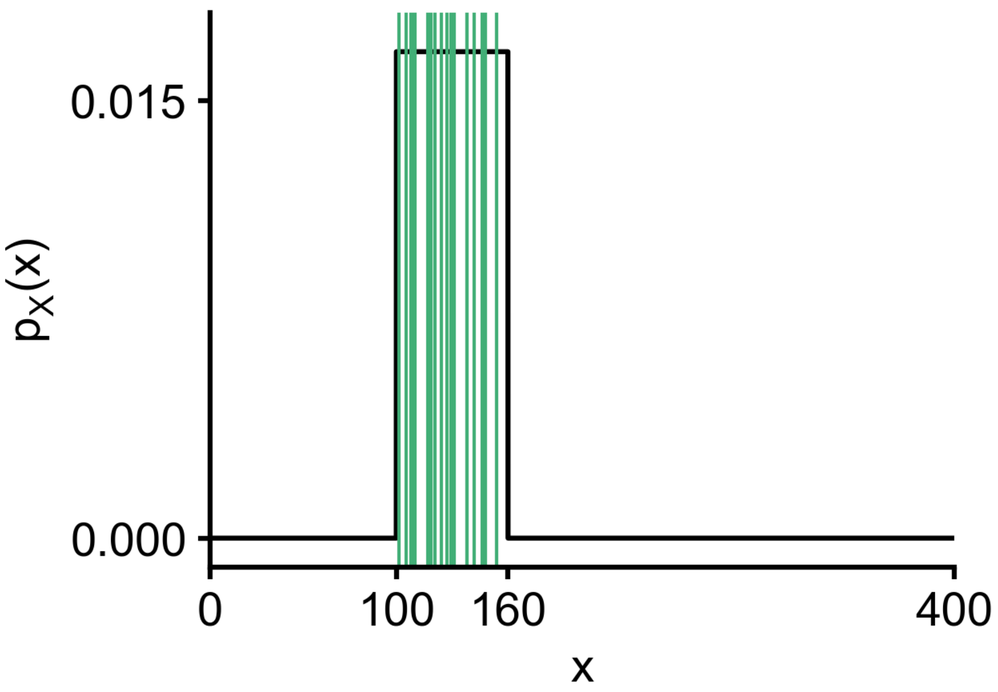


$$Y = f(X)$$

$$Y = 2X$$

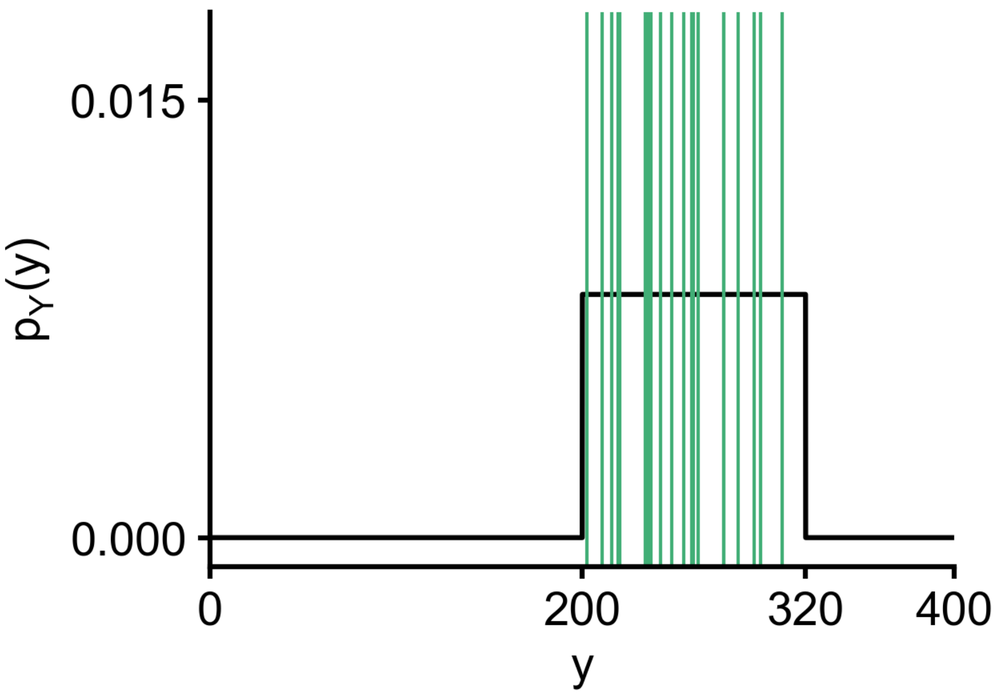


How can we compute the probability density function? Given p(x) and f, compute p(y)



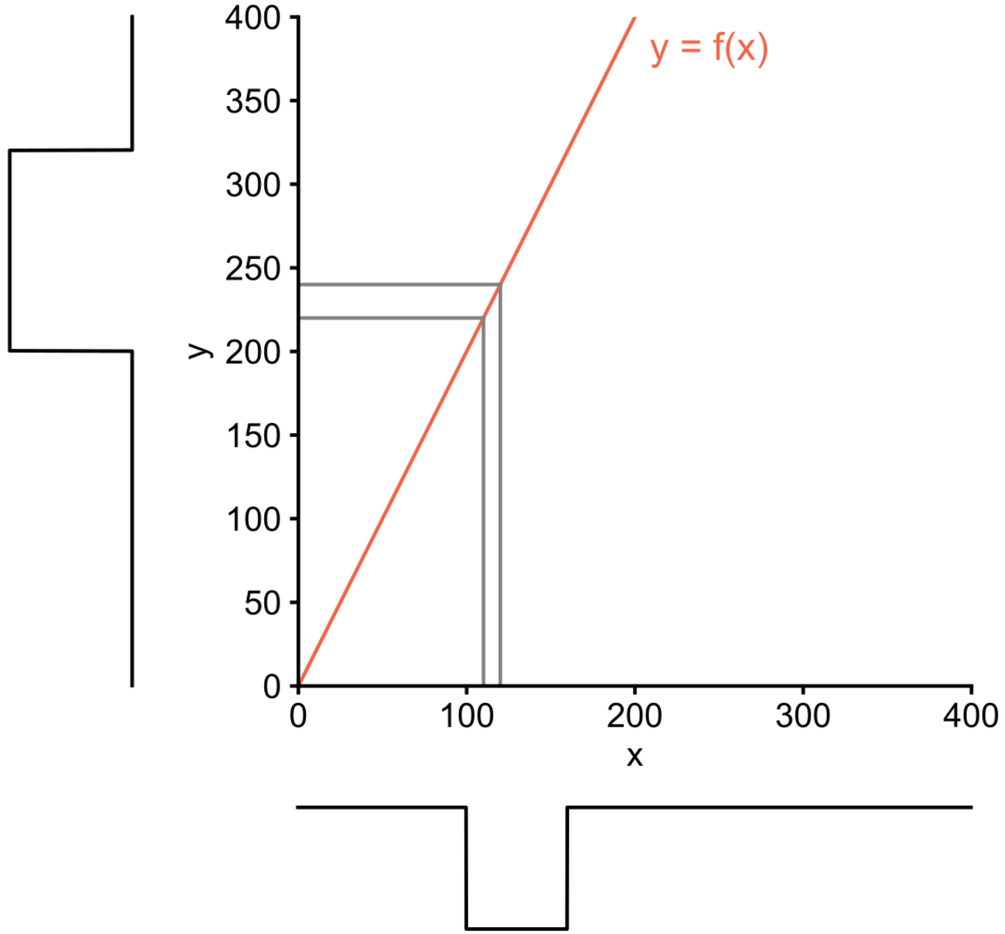
$$Y = f(X)$$

$$Y = 2X$$

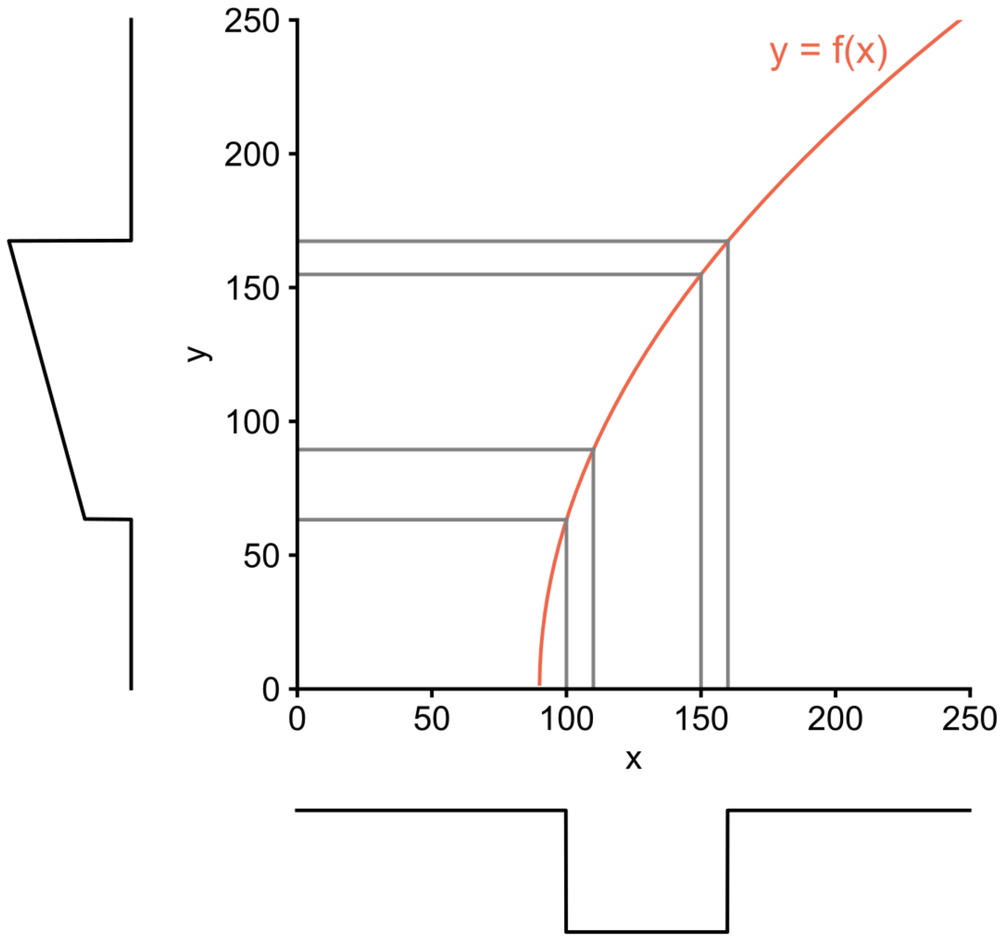


To sample from the transformed distribution, we draw samples from $p_X(x)$ and apply the transformation.
But: How can we evaluate the probability density function of the transformed random variable?

A different way to visualise this problem: Small volumes get stretched by the transformation, so the density decreases



What is the function is nonlinear? Different locations get stretched by the derivative of the function f.



$$p_Y(y) = \frac{1}{f'(x)} p_X(x)$$

with $x = f^{-1}(y)$

Change of variables for probability distributions. Given $p_X(x)$ and f, compute $p_Y(y)$

If $x \in \mathbb{R}^1$ with $x = f^{-1}(y)$

$$p_Y(y) = \frac{1}{|f'(x)|} p_X(x)$$

If $x \in \mathbb{R}^N$

$$p_Y(y) = \frac{1}{\det(J_f(x))} p_X(x) = \frac{1}{|J_f(x)|} p_X(x)$$

What do we need to compute to use this formula?

$$p_Y(y) = \frac{1}{\det(J_f(x))} p_X(x) \text{ with } x = f^{-1}(y)$$

Conditions for applying the change of variables formula (i.e to evaluate p_Y):

- identify all x for which $f(x) = y$, i.e. compute $x = f^{-1}(y)$
- evaluate Jacobian $J_f(x)$
- compute determinant of the Jacobian, which is $\mathcal{O}(N^3)$ with $N = \dim(x)$

In order to sample from p_Y:

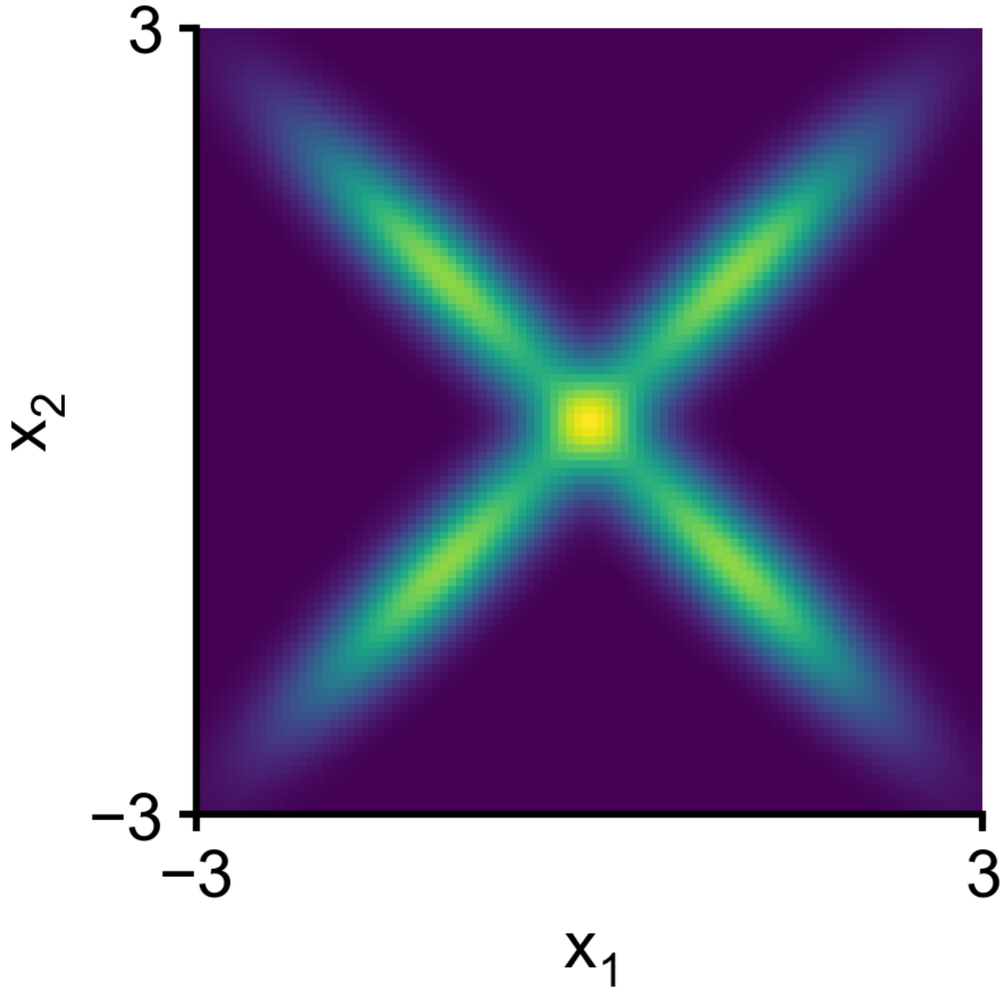
- evaluate $f(x)$

This is fulfilled if f is a diffeomorphism: a differentiable bijection

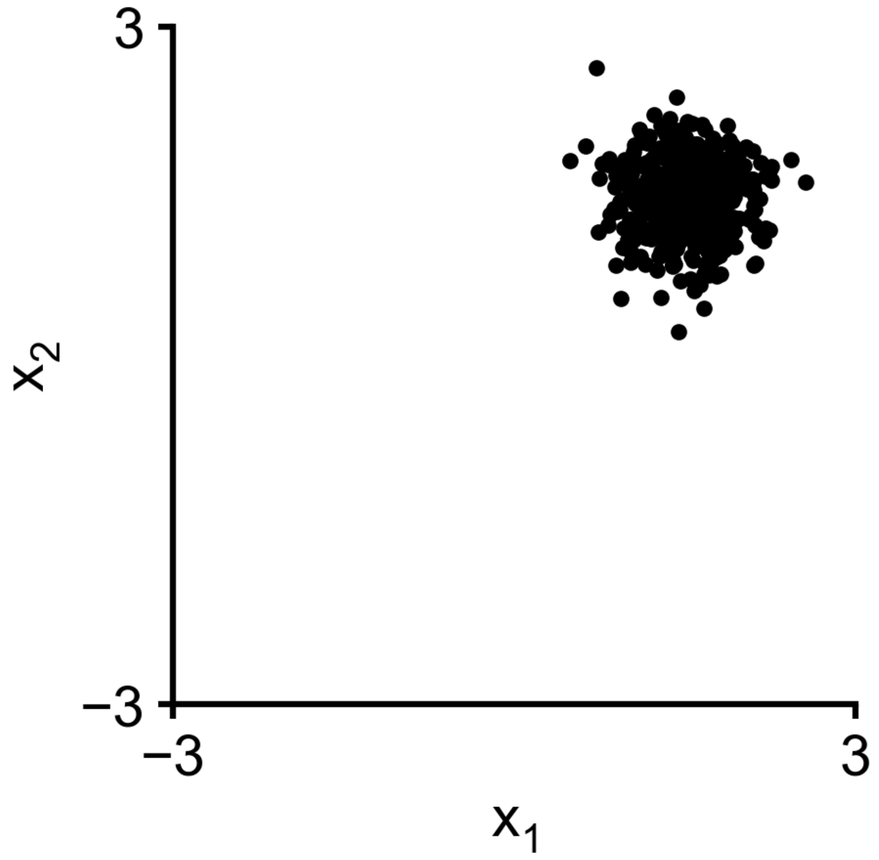
Summary: change of variables

- Change of variables describes the transformation of random variables with a function f
- Sampling from the transformed distribution is easy: simply apply f to samples from the the base distribution
- Evaluating p_Y requires using the change of variables formula
- This formula requires that the function can be inverted and that it is differentiable

8.3 Normalising flows



Density Estimation with Maximum Likelihood: Obtain a maximum-likelihood estimate of parameters of a distribution given data



$$\phi^* = \max_{\phi} \sum_{i}^{N} \log p(x^i; \phi)$$

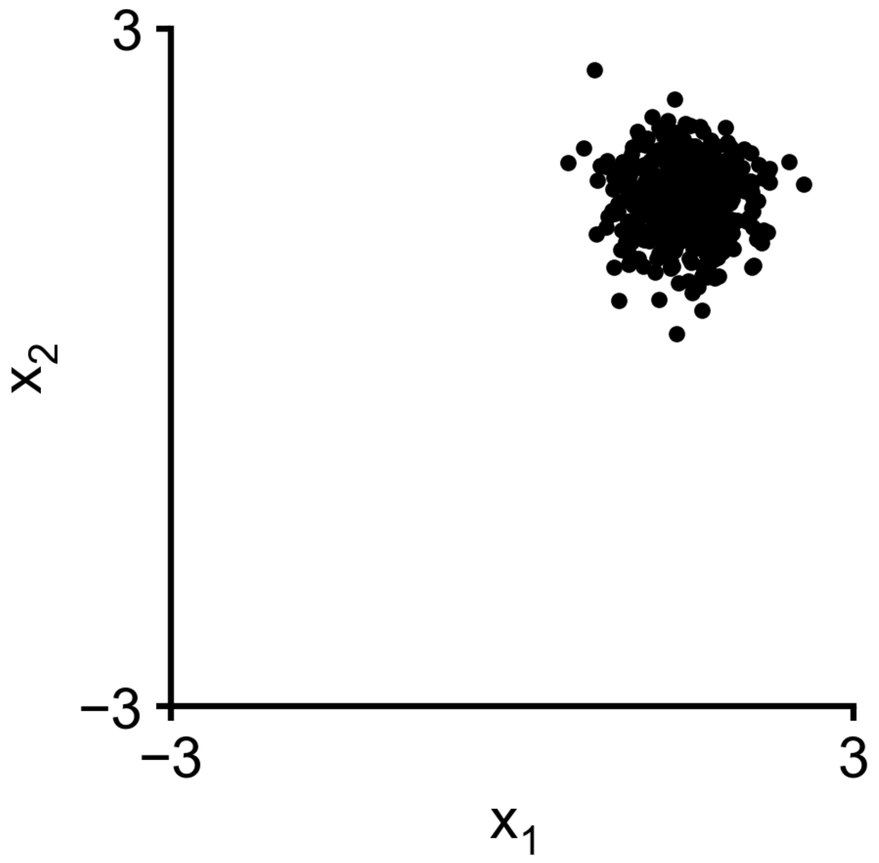
Example: Estimate the density of the data on the left

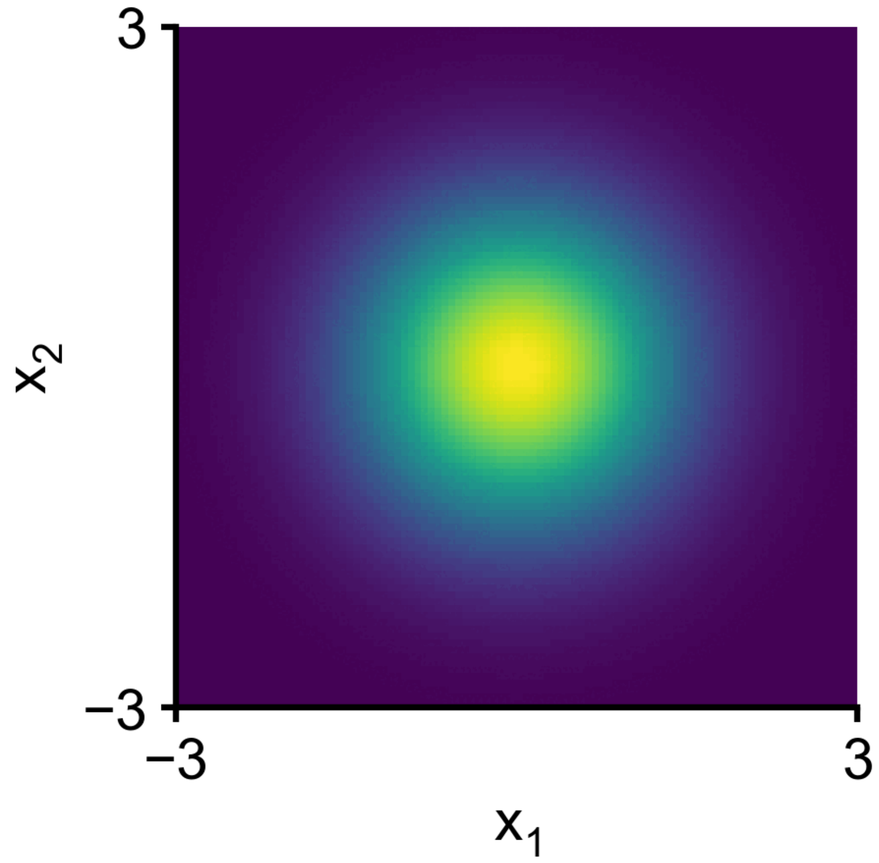
1) Assume a parametric family $p(x; \phi)$

E.g., $p(x; \phi) = \mathcal{N}(x; \mu, \sigma)$

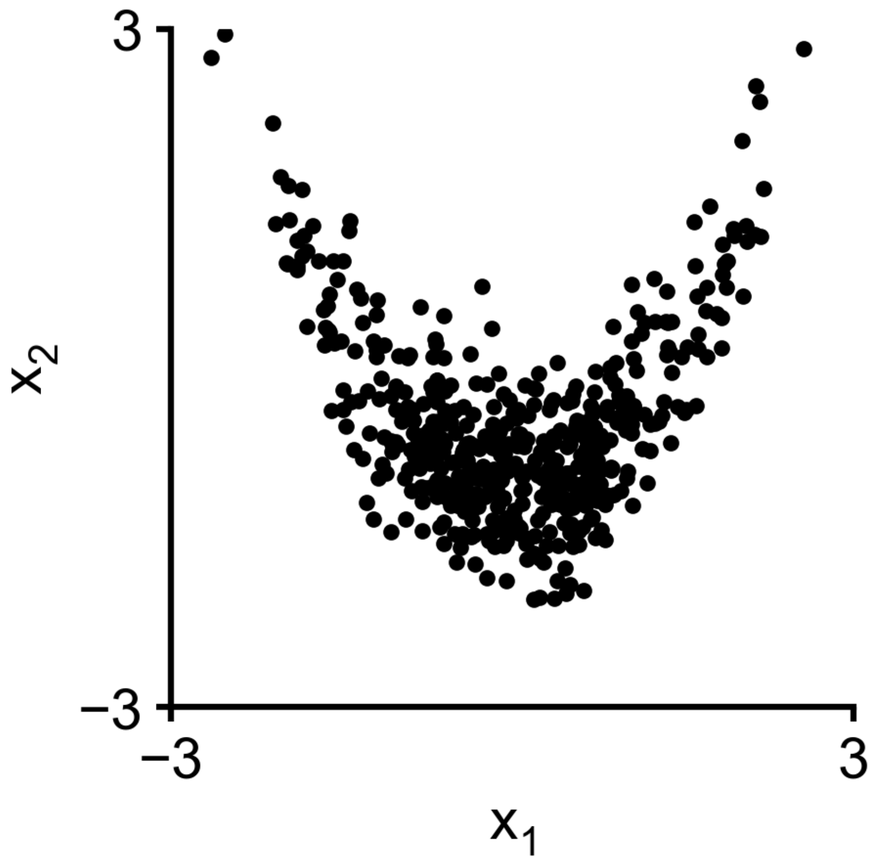
2) Optimize the parameters ϕ with maximum likelihood

Density Estimation with Maximum Likelihood: Obtain a maximum-likelihood estimate of parameters of a distribution given data





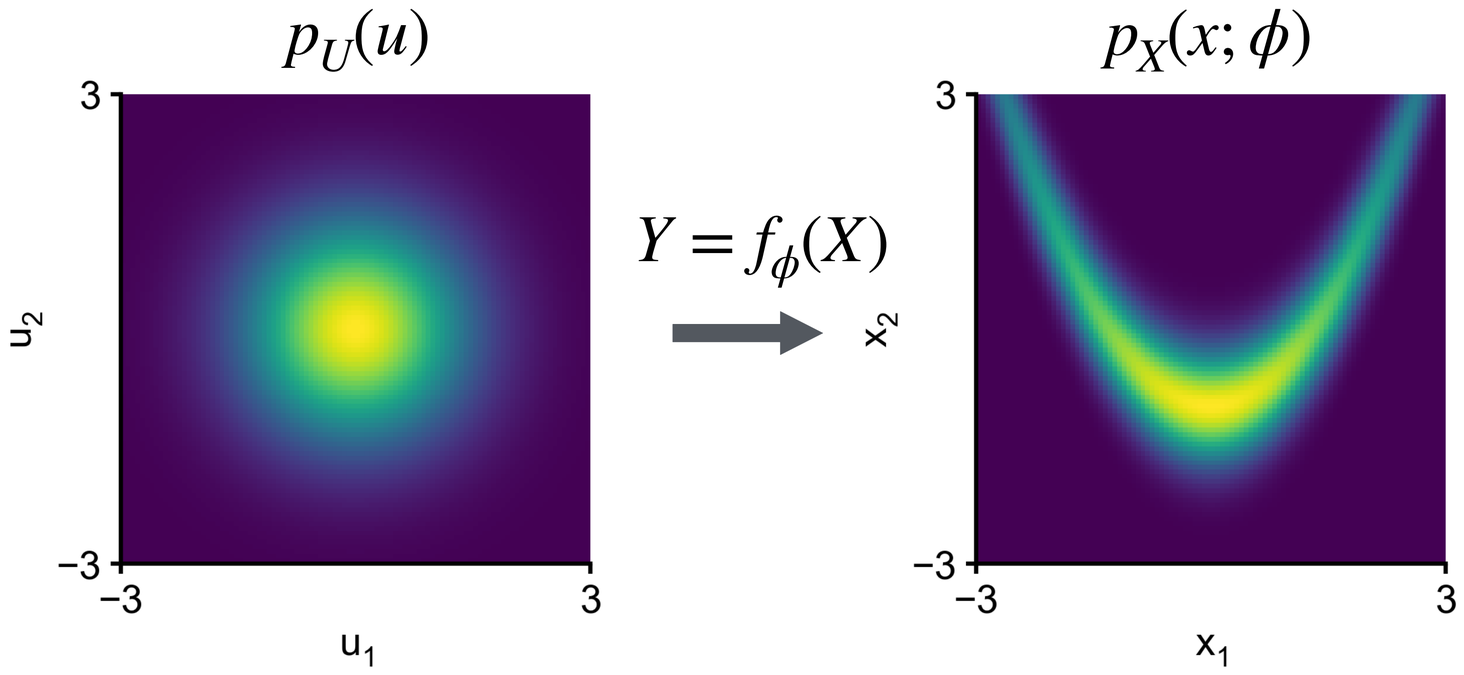
But: What if the data is not Gaussian? How to model nonlinear dependencies between Gaussians?



How can we build parametric densities that can capture any data?

- **Normalizing flows** allow doing that with the change of variables formula.

Key idea: learn transformation parameters dependent on values of another parameter



$$\phi^* = \max_\phi \sum_i^N \log p(x^i;\phi) = \max_\phi \sum_i^N \log \frac{1}{|J_{f_\phi}(u^i)|} p_U(u^i)$$

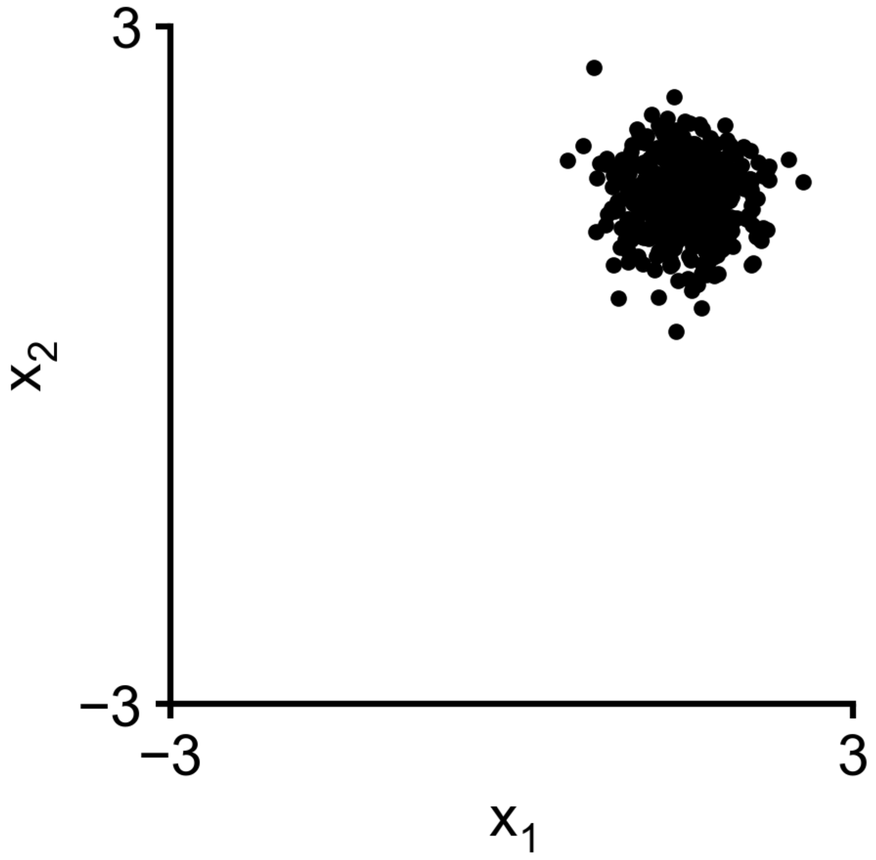
Parameterize the transformation f_ϕ and fit its parameters with maximum likelihood.

How do we parameterise f_ϕ such that we can still apply change of variables?

- invertible
- differentiable
- efficiently computable determinant

Different Normalizing flows differ in how they implement these criteria!

Our first normalising flow: A linear transformation in each dimension



$$f(u) = Au + b = \begin{bmatrix} \exp(a_1) & 0 \\ 0 & \exp(a_2) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$\phi = (a_1, a_2, b_1, b_2)$$

$$p_U(u) = \mathcal{N}(u; 0, I)$$

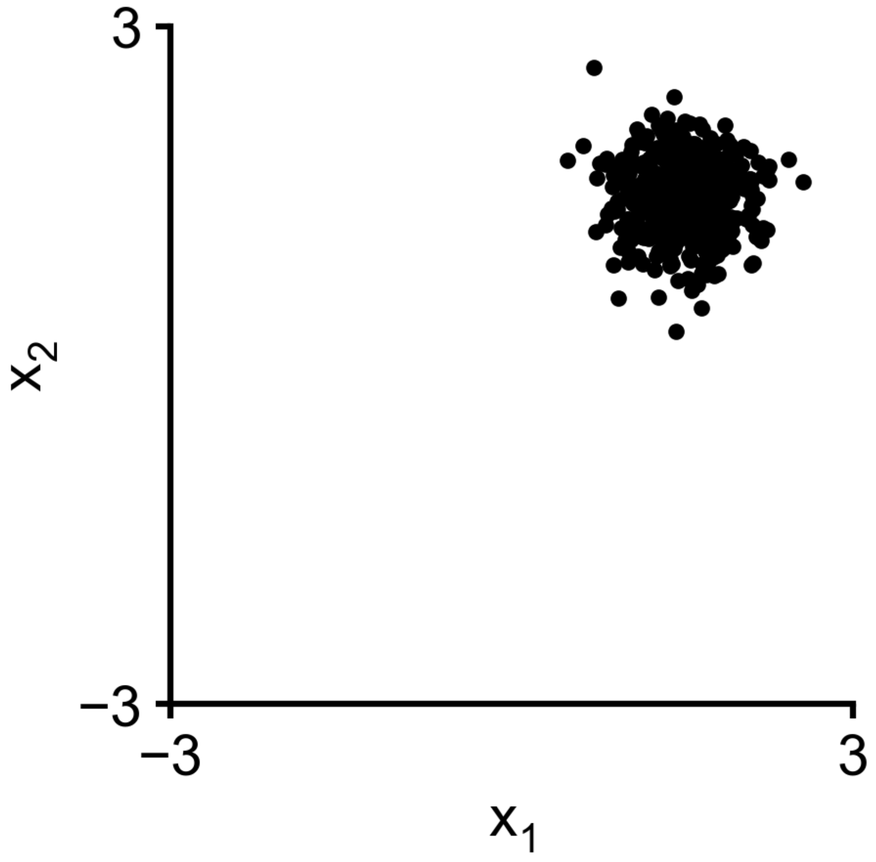
$$\phi^* = \max_\phi \sum_i^N \log p(x^i; \phi) = \max_\phi \sum_i^N \log \frac{1}{|J_{f_\phi}(u^i)|} p_U(u^i)$$

$$= \max_\phi \sum_i^N \log \frac{1}{|A|} p_U(u^i) \qquad \text{with } u_i = A^{-1}(x_i - b_i)$$

Note: exponentiation of in order to ensure invertibility.

Invertible Differentiable $O(N)$ determinant

Our first normalising flow: A linear transformation in each dimension



$$f(u) = Au + b = \begin{bmatrix} \exp(a_1) & 0 \\ 0 & \exp(a_2) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$\phi = (a_1, a_2, b_1, b_2)$$

$$p_U(u) = \mathcal{N}(u; 0, I)$$

$$\phi^* = \max_\phi \sum_i^N \log p(x^i; \phi) = \max_\phi \sum_i^N \log \frac{1}{|J_{f_\phi}(u^i)|} p_U(u^i)$$

$$= \max_\phi \sum_i^N \log \frac{1}{|A|} p_U(u^i) \qquad \text{with } u_i = A^{-1}(x_i - b_i)$$

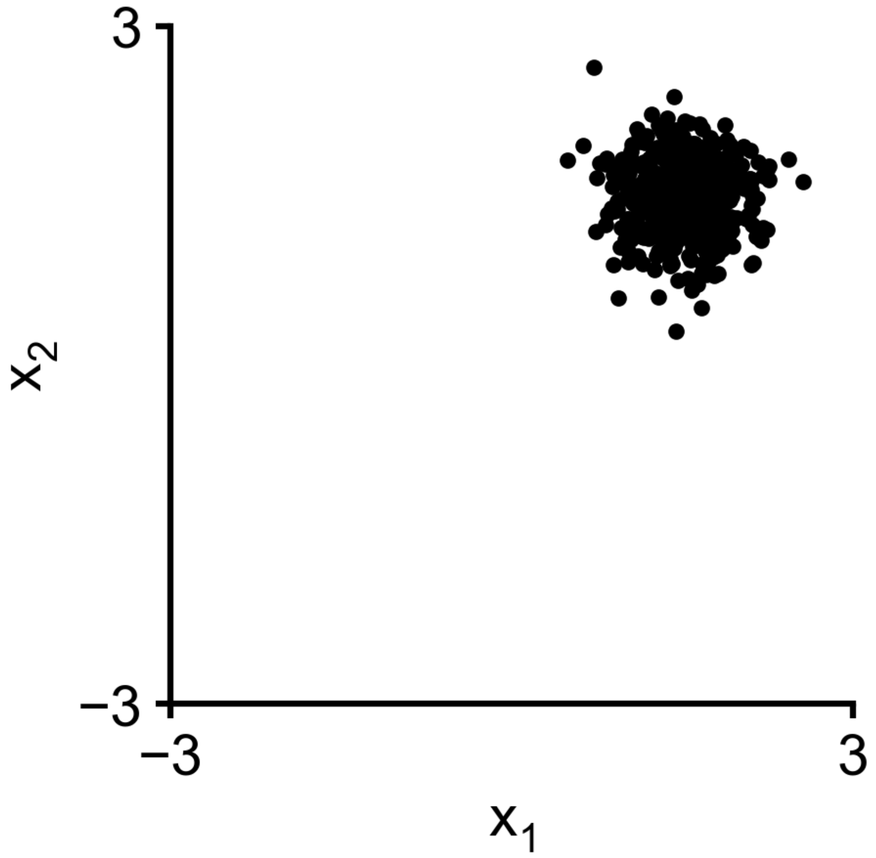
Note: exponentiation of in order to ensure invertibility.

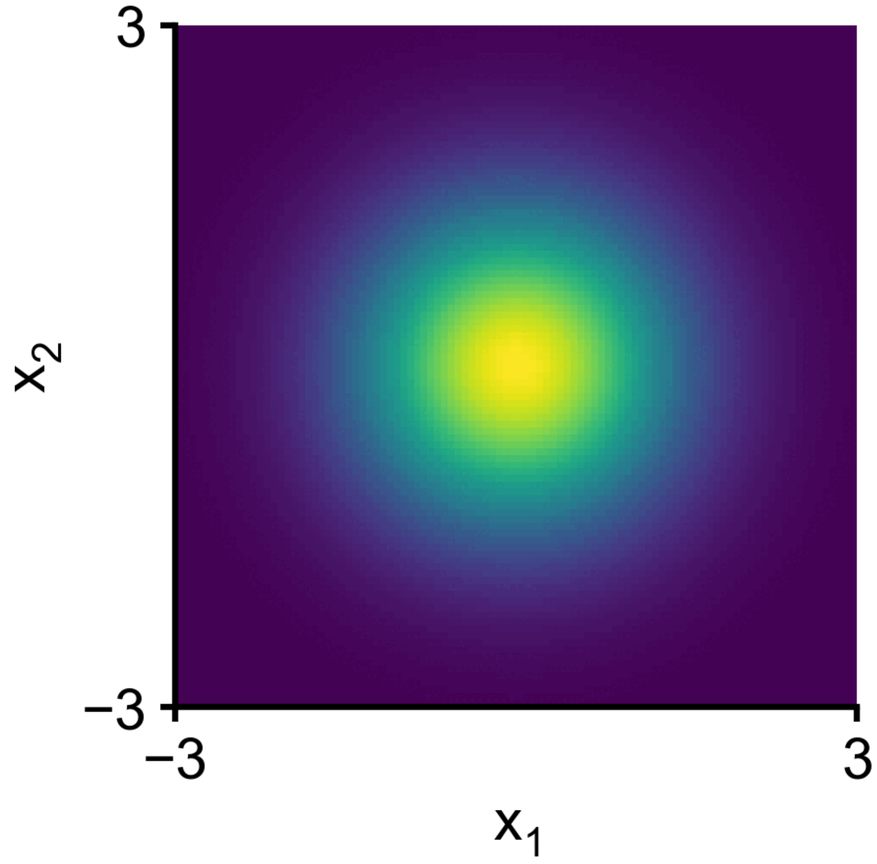
✓ Invertible
$u = A^{-1}(x - b)$

✓ Differentiable
$J_{f_\phi}(u) = A$

✓ $\mathcal{O}(N)$ determinant
$\det(A) = \prod_i \exp(a_i)$

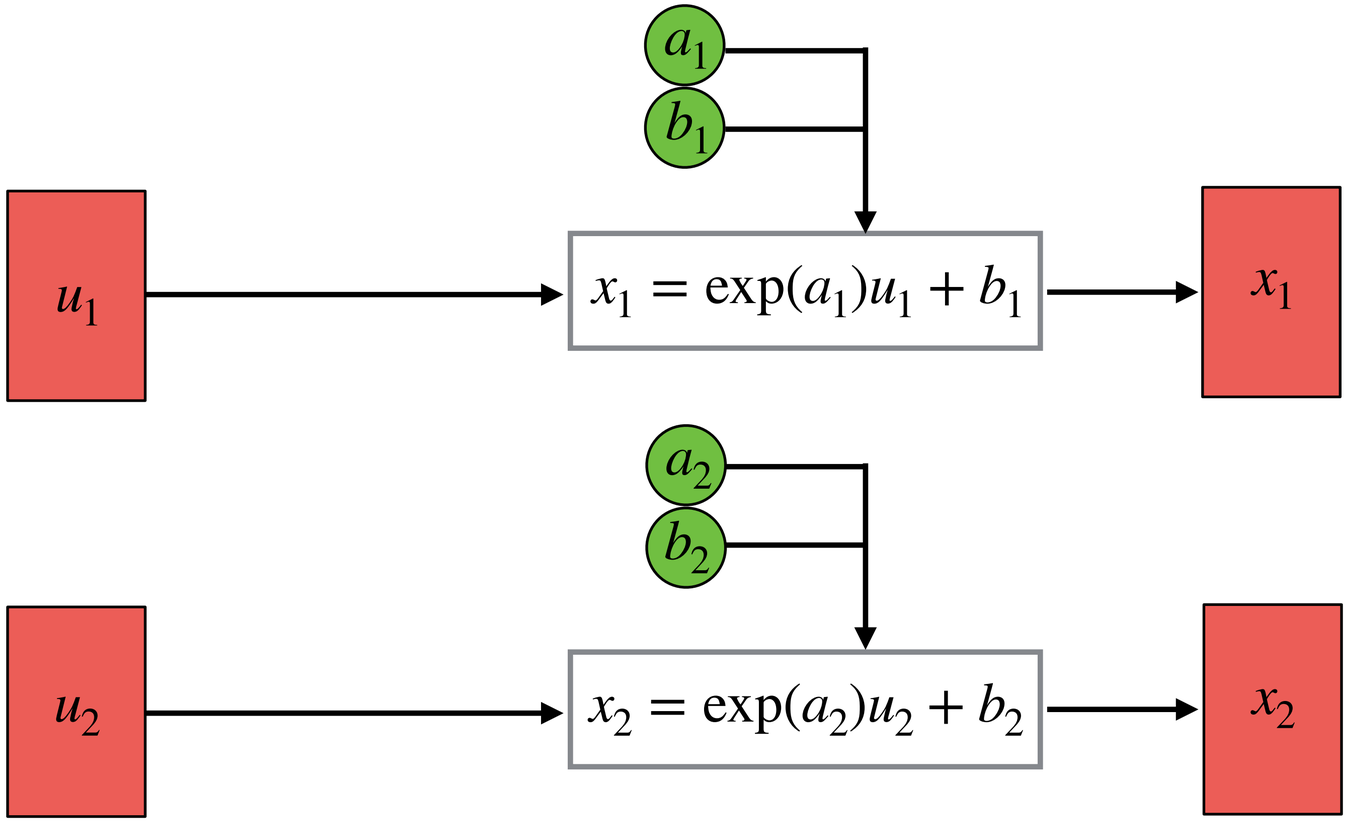
Works, but still only linear/Gaussian …



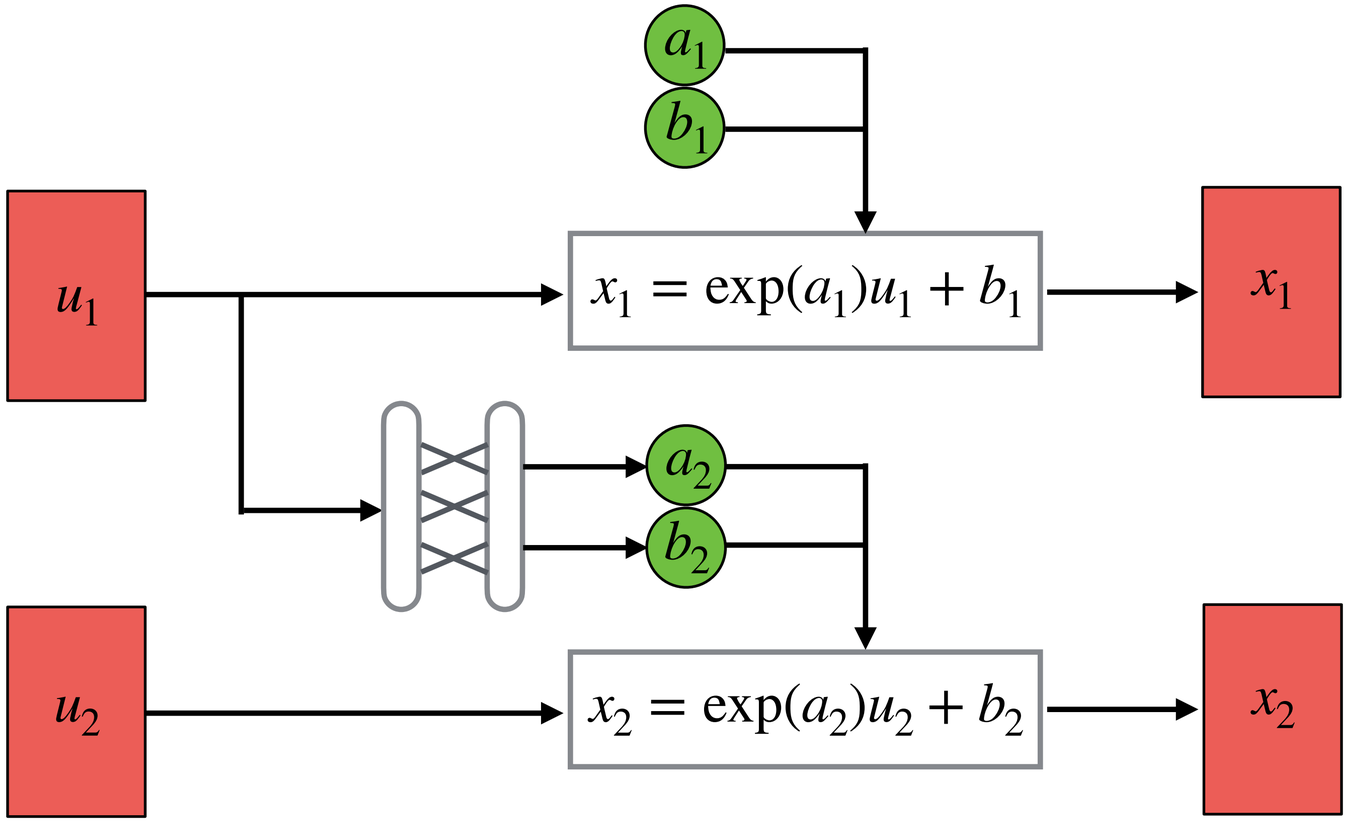


How can we build more expressive flows?

Autoregressive Normalising Flows I: Forward pass

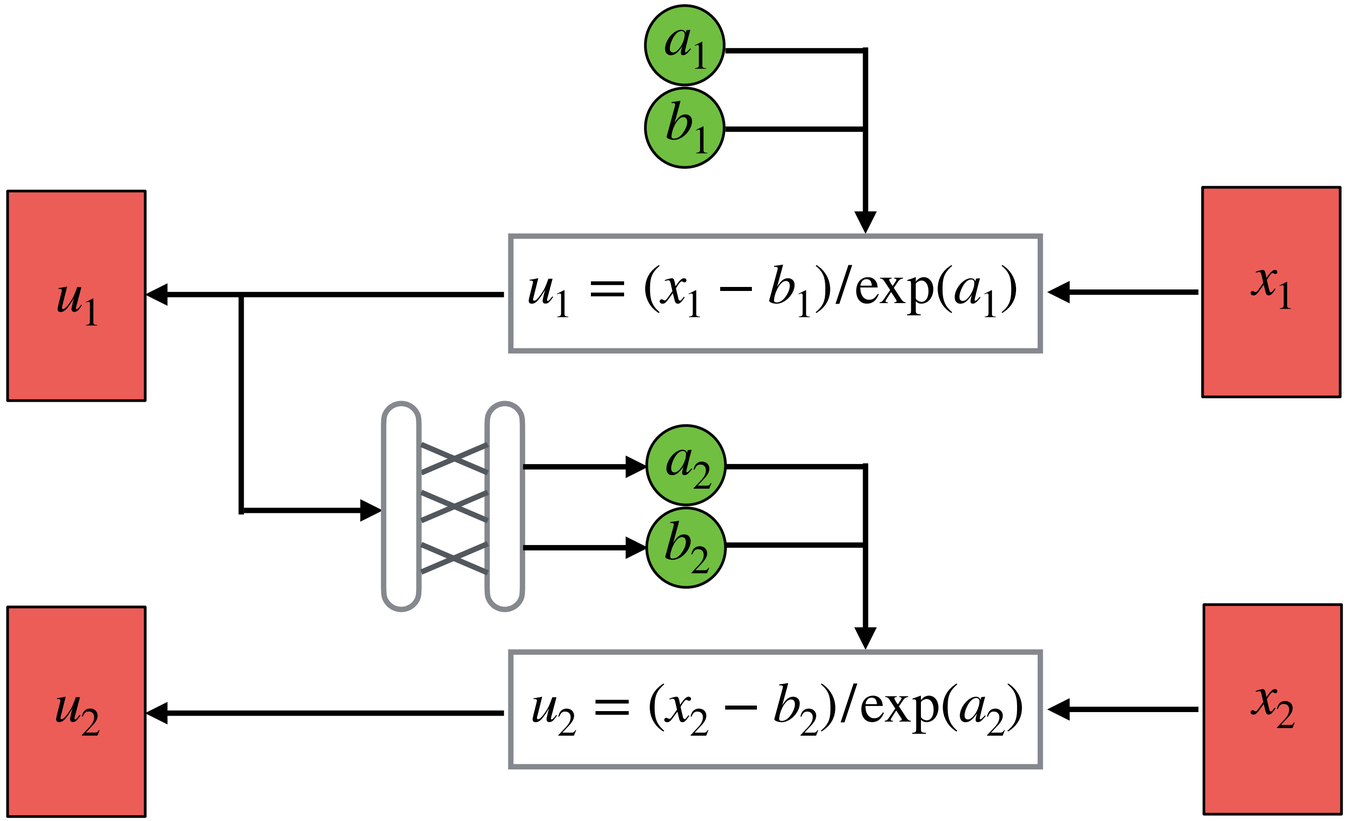


Autoregressive Normalising Flows I: Forward pass. Key idea: Make transformation of second variable dependent on first variable

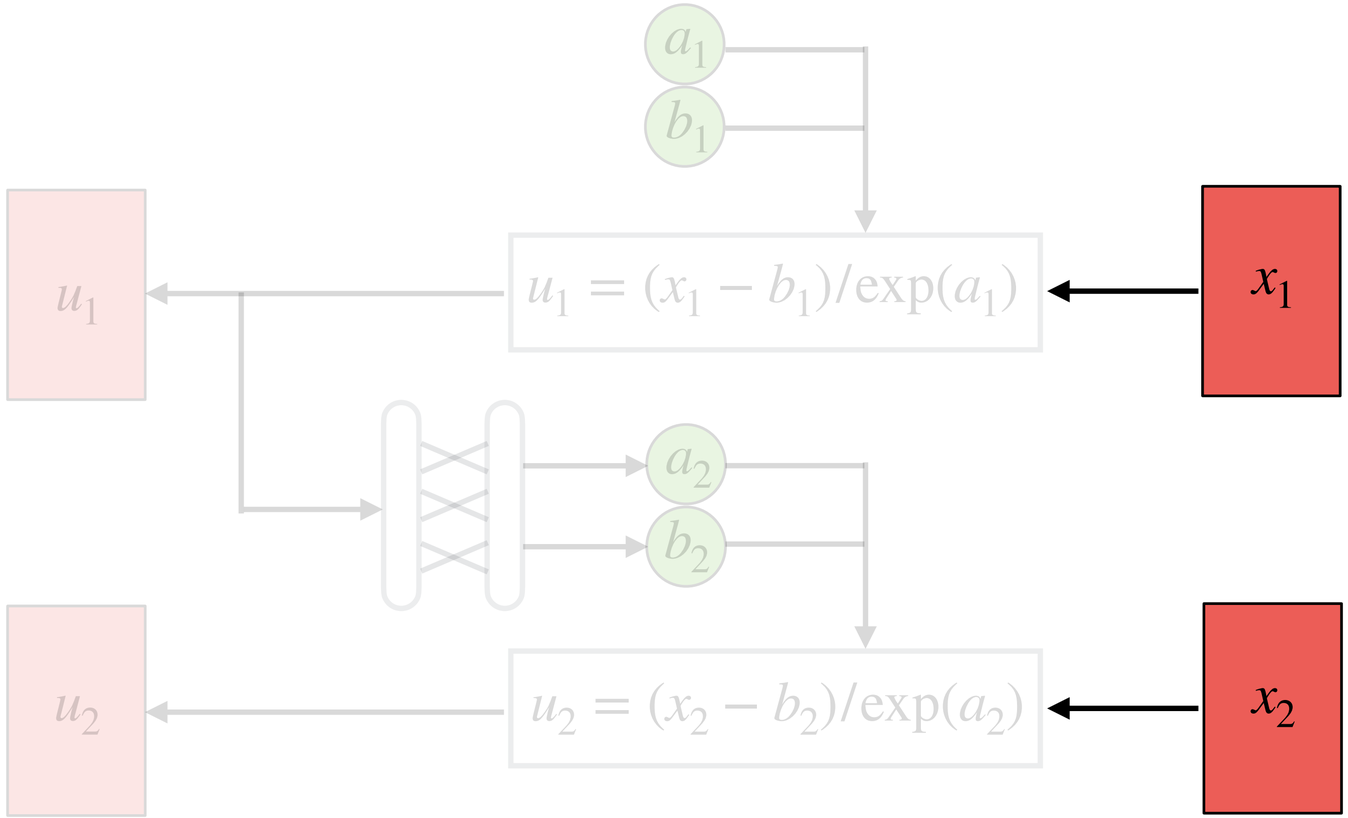


- Transform some parameters dependent on the value of other parameters.
- Dependency is modelled by, e.g., a neural net

Autoregressive Normalising Flows: Inverse. Applying the inverse transform

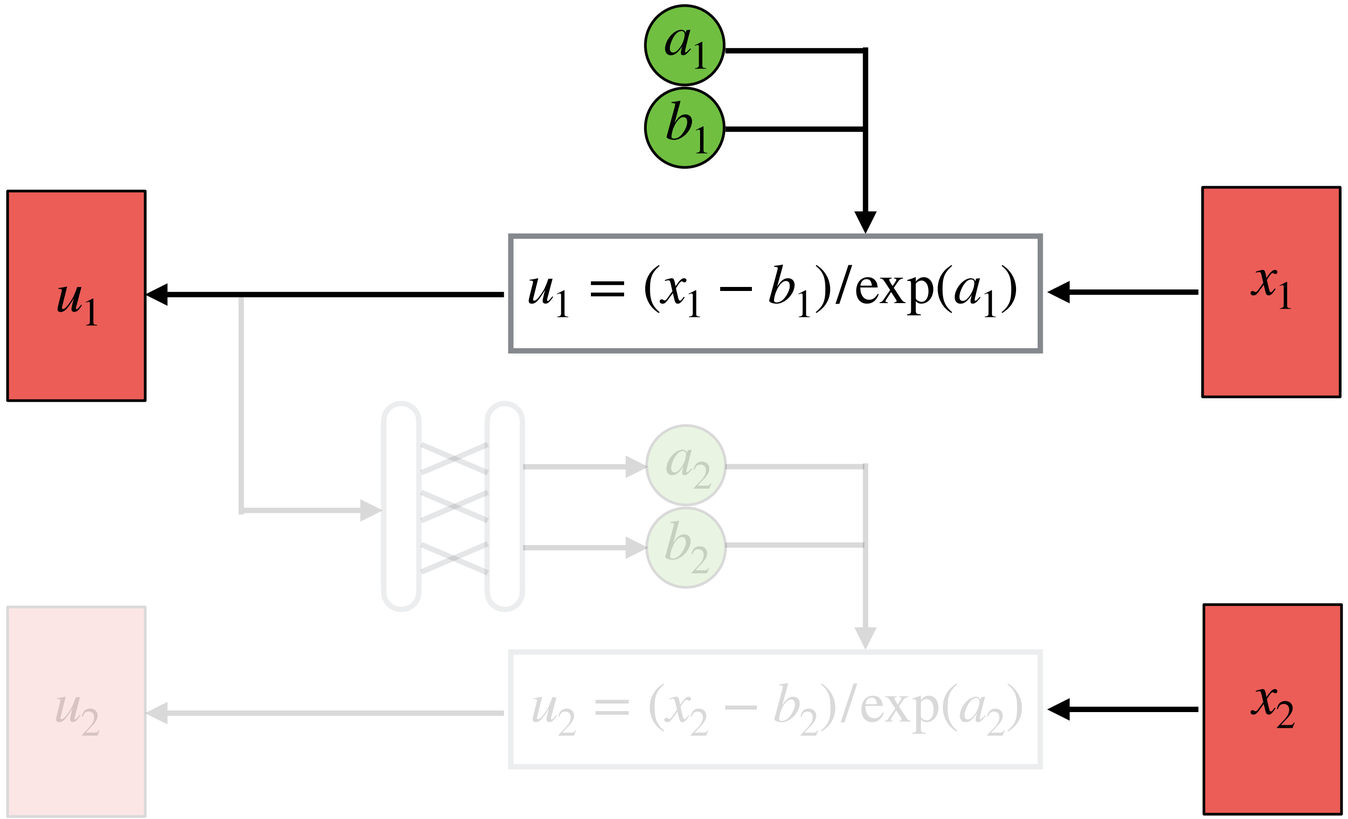


Autoregressive Normalising Flows: Inverse. But…



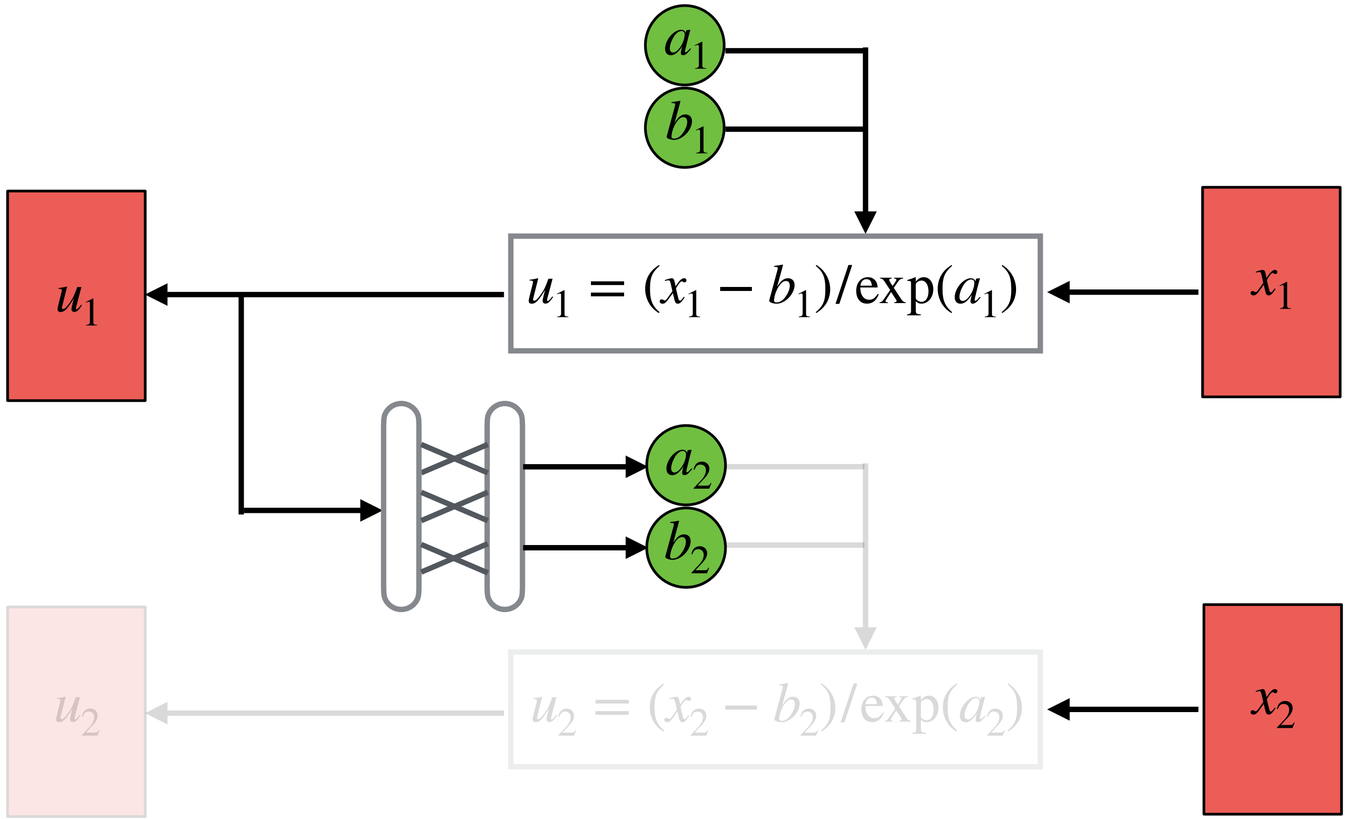
- We do not know the values of (a_2, b_2) because they depend on the value of u_1

Autoregressive Normalising Flows: Inverse. But…



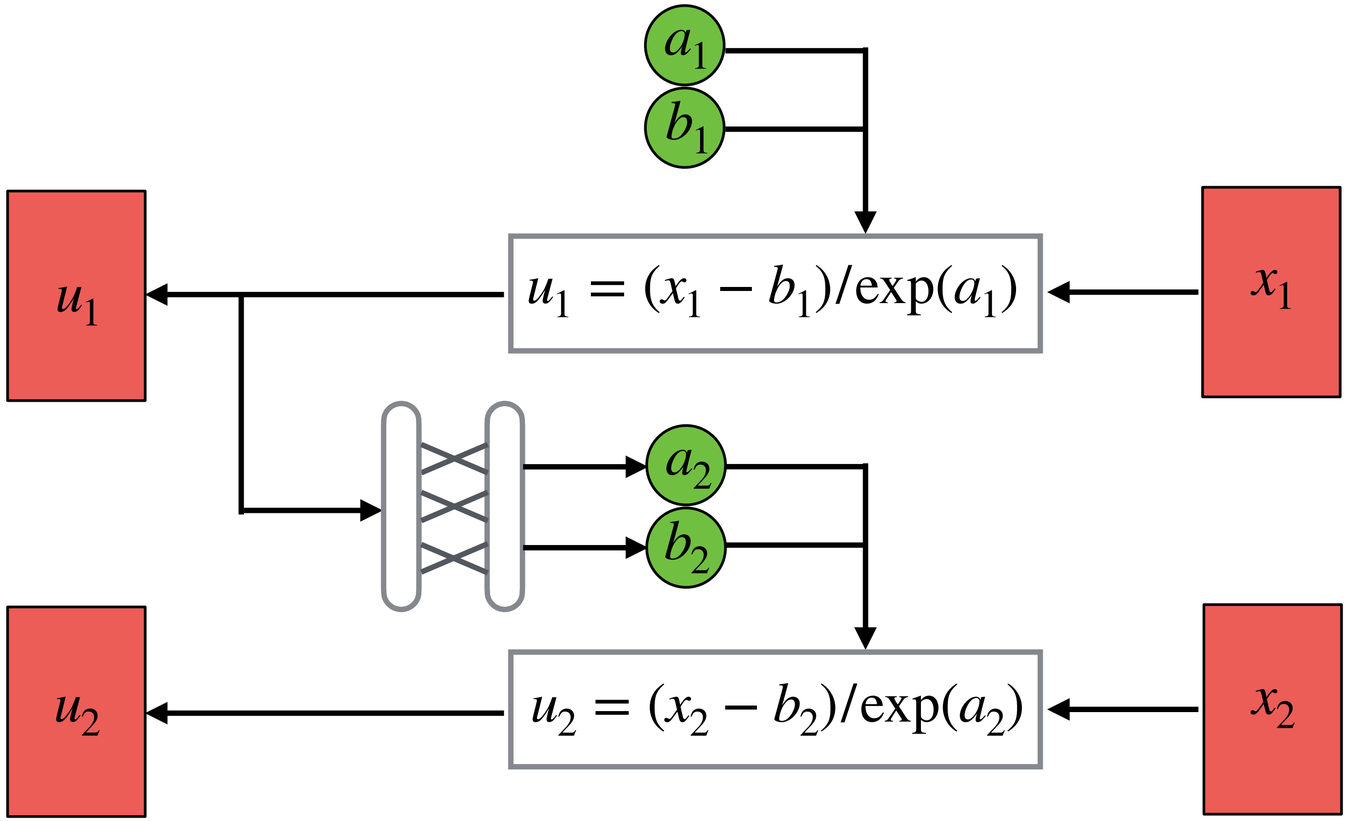
- We do not know the values of (a_2, b_2) because they depend on the value of u_1
- However, we can compute u_1 from x_1 because (a_1, b_1) do not depend on u

Autoregressive Normalising Flows: Inverse. But…



- We do not know the values of (a_2, b_2) because they depend on the value of u_1
- However, we can compute u_1 from x_1 because (a_1, b_1) do not depend on u
- We can then compute (a_2, b_2)

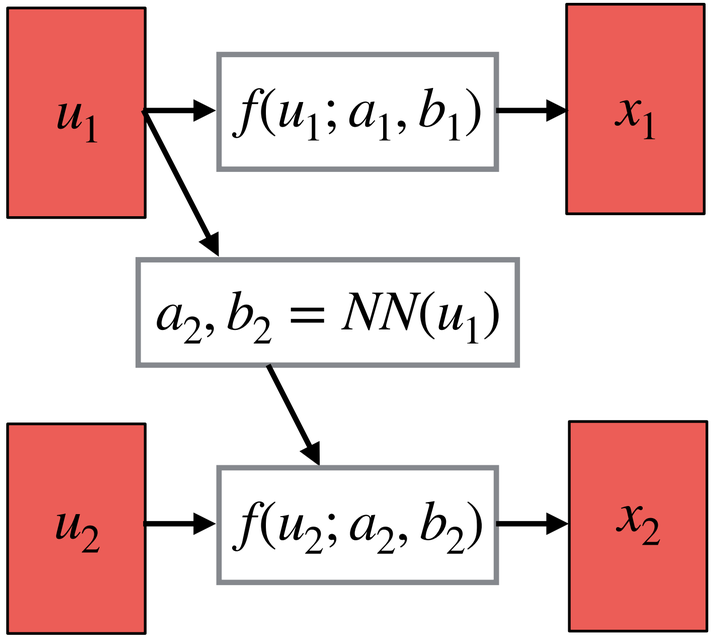
Autoregressive Normalising Flows II: Inverse



- We do not know the values of (a_2, b_2) because they depend on the value of u_1
- However, we can compute u_1 from x_1 because (a_1, b_1) do not depend on u
- We can then compute (a_2, b_2)
- With (a_2, b_2) we can compute u_2

✓ Invertible

Simplified version of this diagram. Can the determinant be efficiently computed?



$$\begin{bmatrix} \frac{dx_1}{du_1} & \frac{dx_1}{du_2} \\ \frac{dx_2}{du_1} & \frac{dx_2}{du_2} \end{bmatrix} = \begin{bmatrix} \frac{dx_1}{du_1} & 0 \\ \frac{dx_2}{d\phi_2}\frac{\phi_2}{du_1} & \frac{dx_2}{du_2} \end{bmatrix}$$

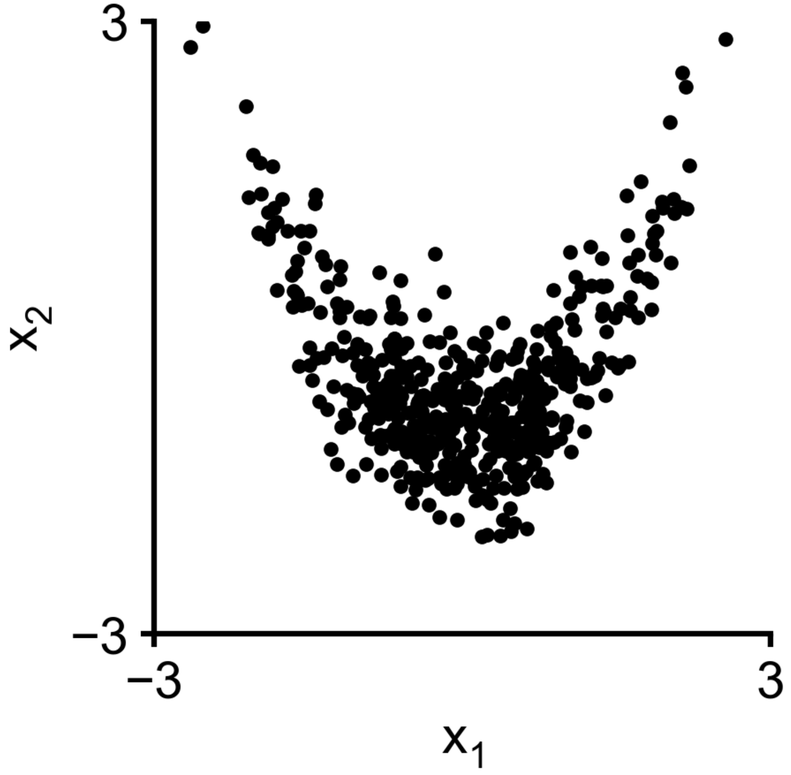
Jacobian is lower-triangular. Thus:

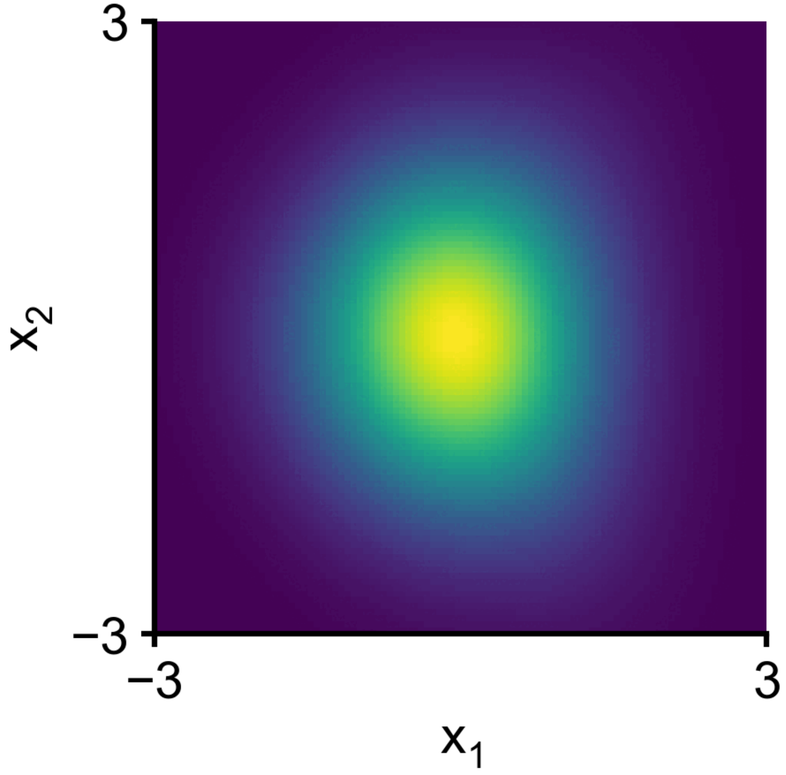
$$\det(J_f) = \prod_i \text{diag}(J_{f^i}) = \prod_i \exp(a_i)$$

This is $\mathcal{O}(N)$ instead of $\mathcal{O}(N^3)$ as usually for determinants.

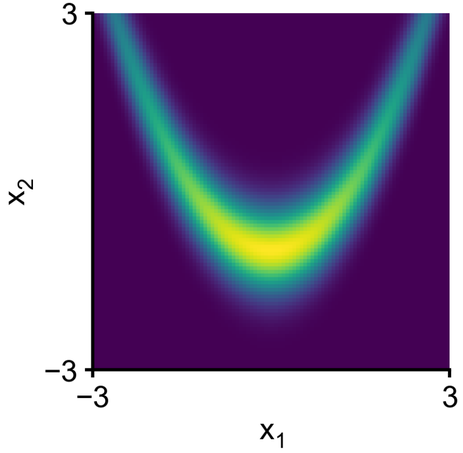
✓ Differentiable ✓$\mathcal{O}(N)$ determinant

So lets see this in action …



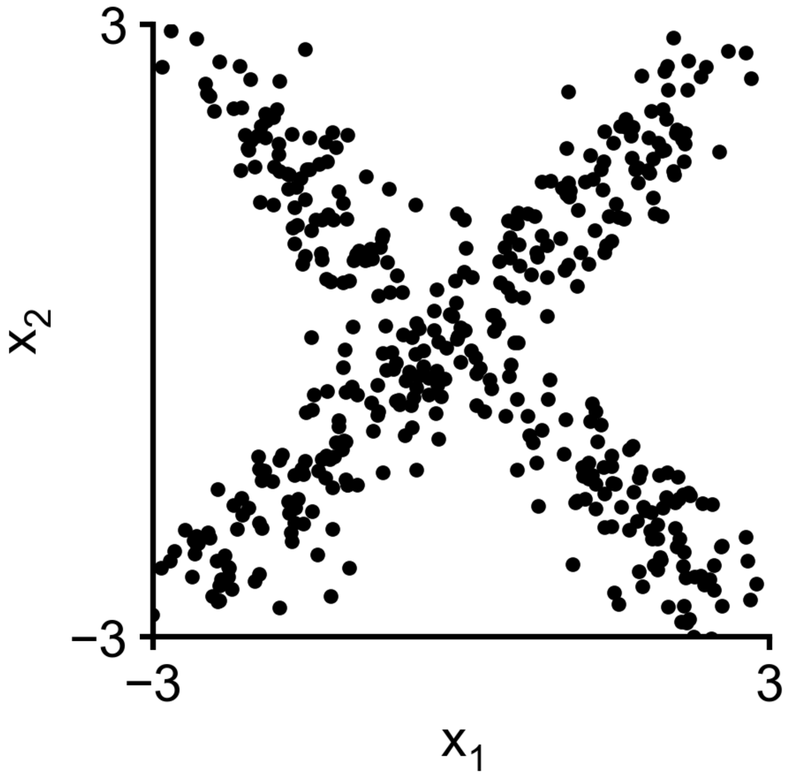


Ground truth:

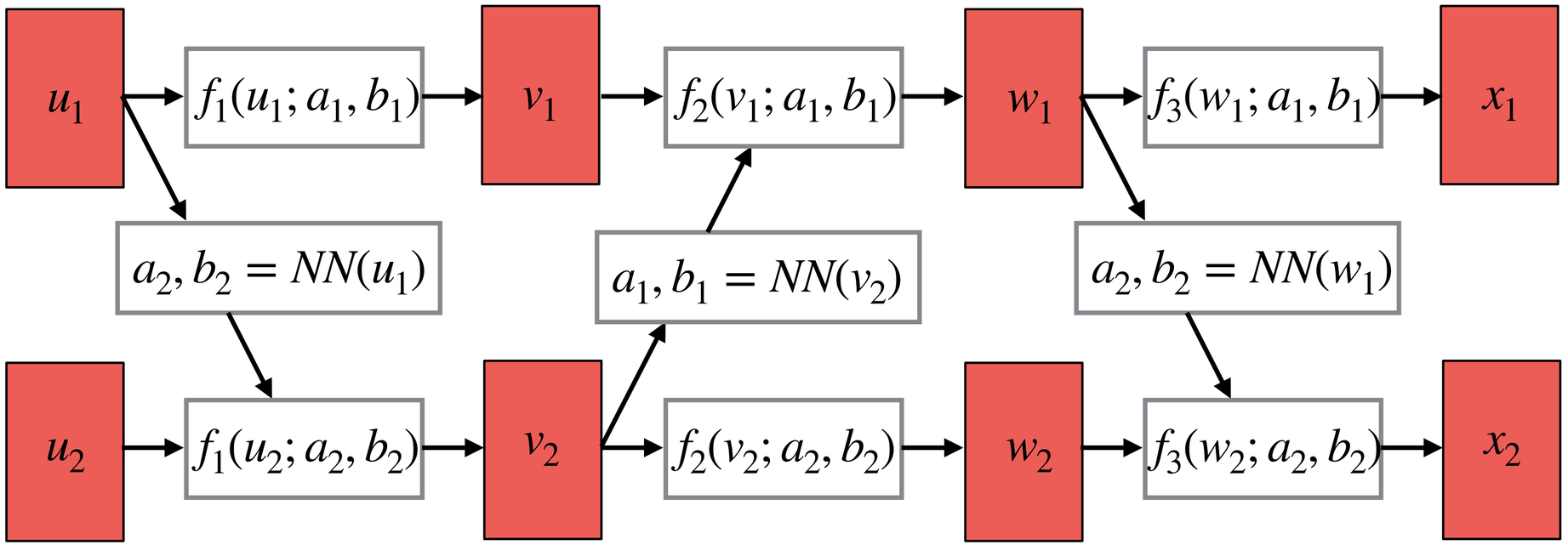


Simulation and inference in neuroscience — PJ Goncalves, P Berens, J Beck @CCC BY-NC-SA 3.0

But: What if our data has all non-Gaussian conditionals? Need even more expressive transformations!

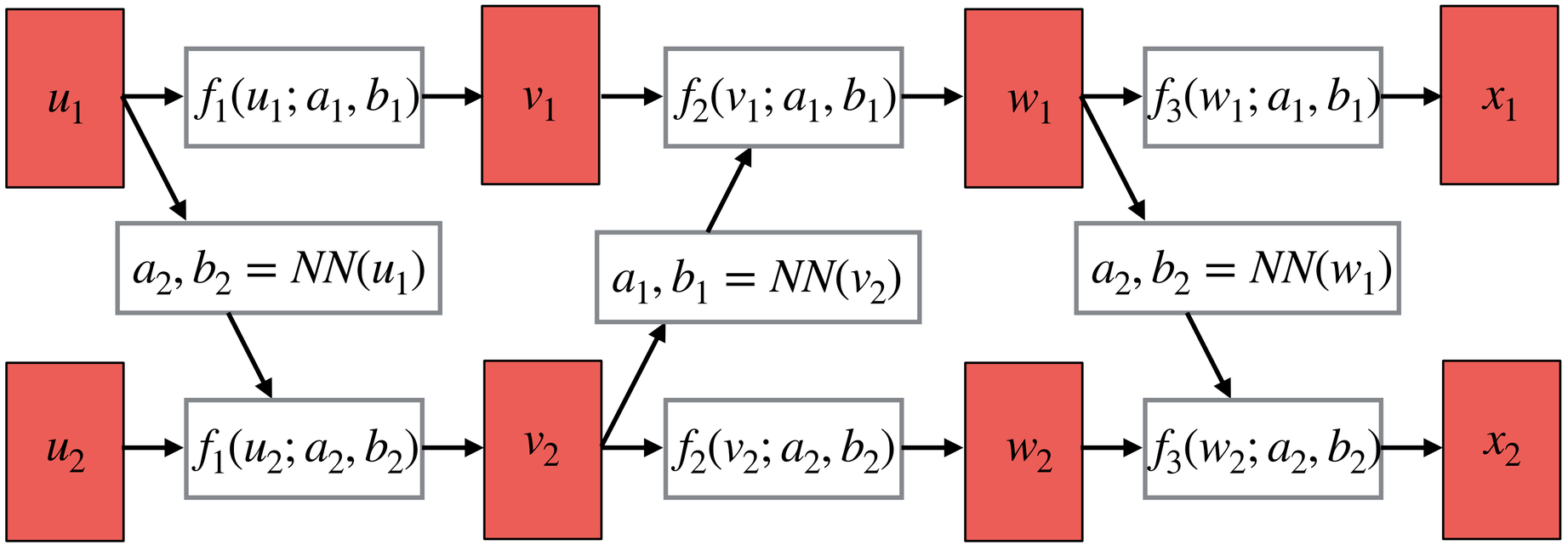


Stacking several layers



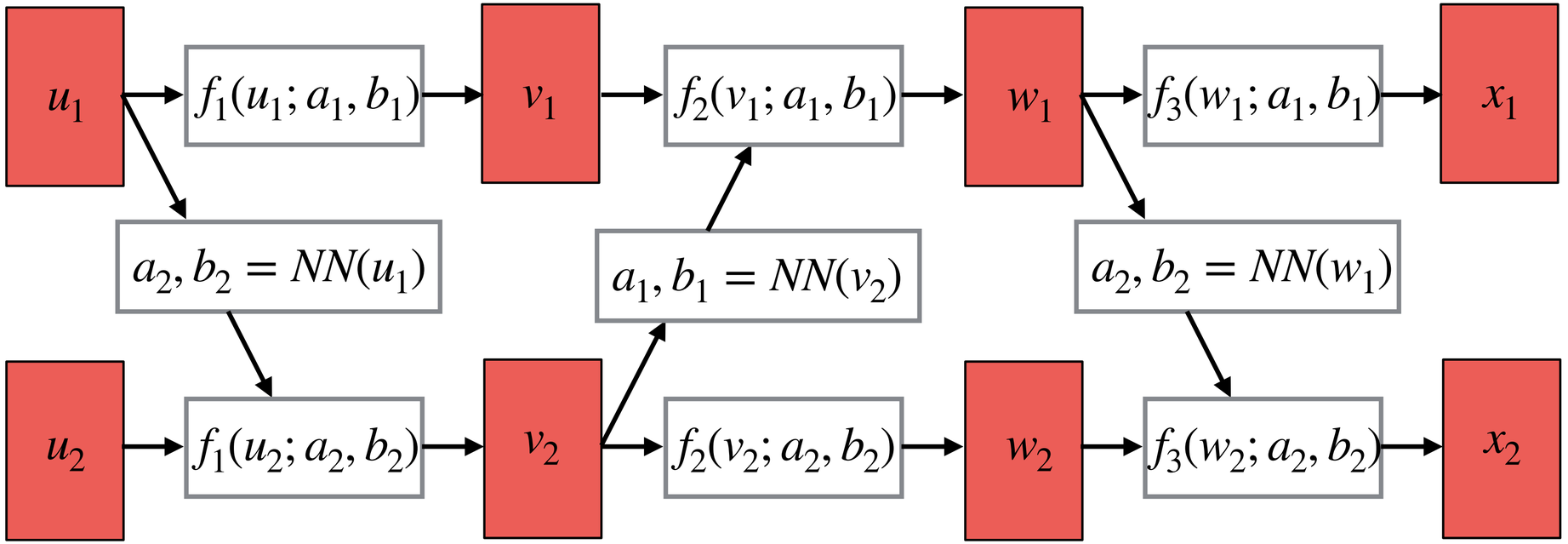
Note: the neural networks have different weights in each layer. The parameters (a_1, b_1, a_2, b_2) are also different in each layer. They have the same names here to avoid notational clutter.

Stacking several layers



✓ Invertible because each layer is invertible ✓ Differentiable because each layer is differentiable

Stacking several layers

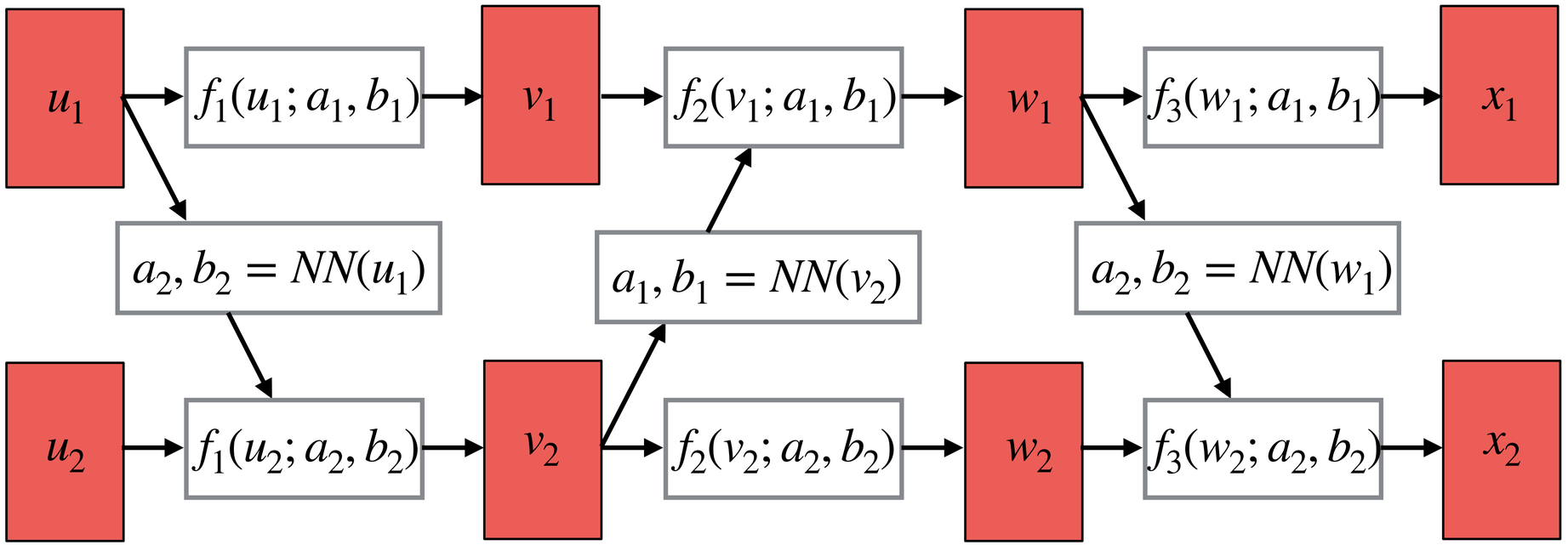


$$p_V(v) = \frac{1}{|J_{f_1}(u)|} p_U(u)$$

$$p_W(w) = \frac{1}{|J_{f_2}(v)|} p_V(v)$$

$$p_X(x) = \frac{1}{|J_{f_3}(w)|} p_W(w)$$

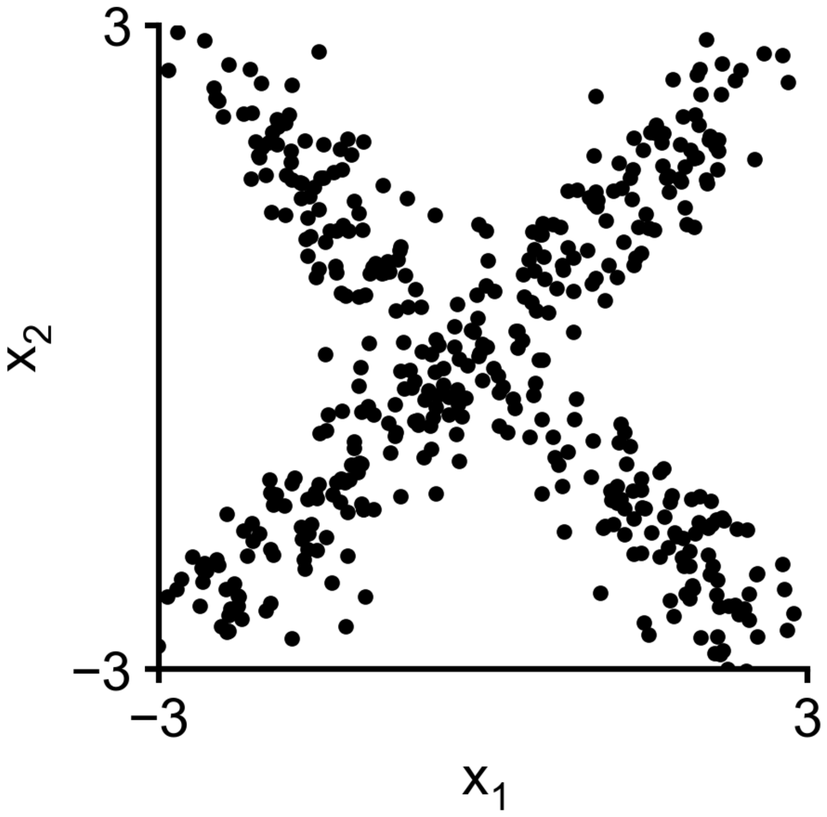
Stacking several layers

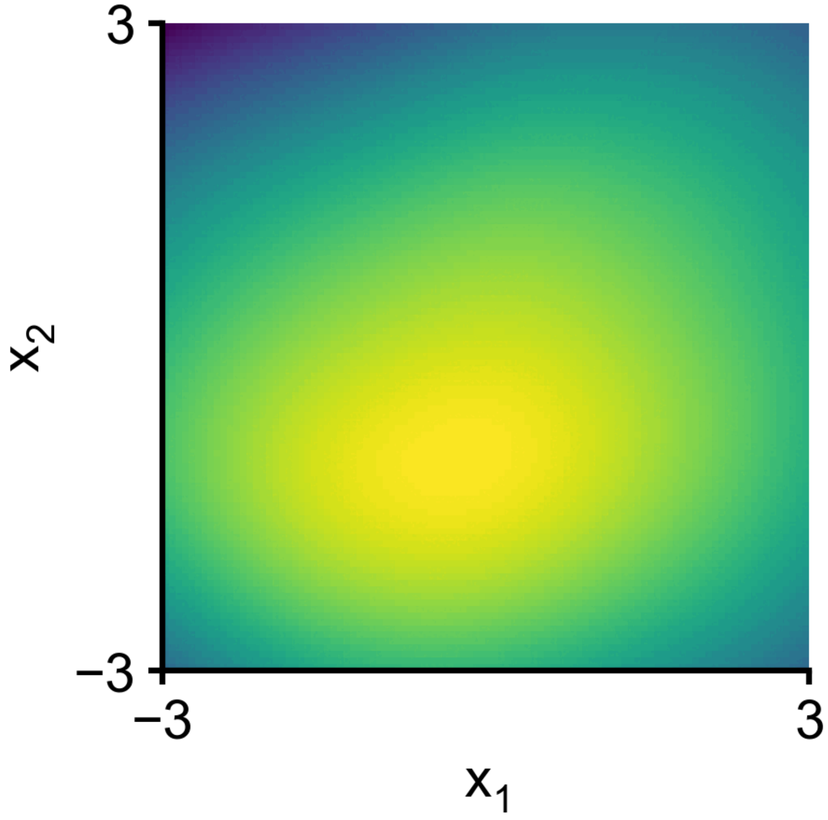


$$p_X(x) = \frac{1}{|J_{f_3}(w)| \cdot |J_{f_2}(v)| \cdot |J_{f_1}(u)|} p_U(u)$$

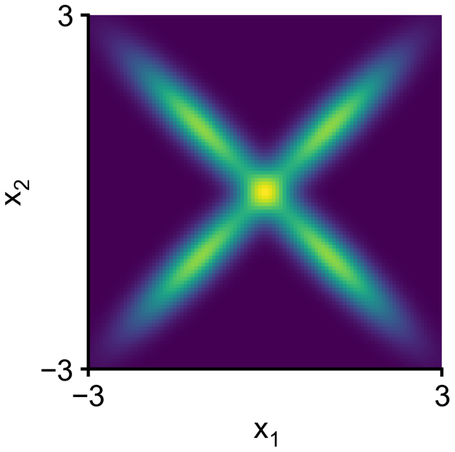
✓$\mathcal{O}(N)$ determinant because the determinant factorises into the product of determinant per layer.

Let us see this in action …





Ground truth:



Lecture 8: Normalising flows

- Normalizing flows are parametric densities which are highly flexible.
- They rely on change of variables: they learn a transformation such that a known base-distribution (usually a multivariate Gaussian) can be transformed into the target distribution.
- The transformation must be invertible, differentiable, and have an efficiently computable determinant of the Jacobian.
- Normalizing flows can be used for density estimation, but also for variational inference.
- Unlike VAEs or GANS, normalising flows allow for closed-form density evaluation and can, thus, be used as proposals in, e.g., rejection sampling or importance sampling.

Further reading on Normalising flows

- Papamakarios, G. et al. (2021). Normalizing flows for probabilistic modeling and inference. Journal of Machine Learning Research, 22(57), 1-64.