Deep Learning

Lecture 6: Flow models

Chris G. Willcocks

Durham University

Lecture overview

1 Generative models

- learning the data distribution
- definition
- probability examples
- maximum likelihood estimation
- cumulative distribution sampling
- generative networks

2 Flow models

- definition
- the determinant
- the change of variables theorem

3 Normalising flows

- definition
- triangular Jacobians
- normalising flow layers

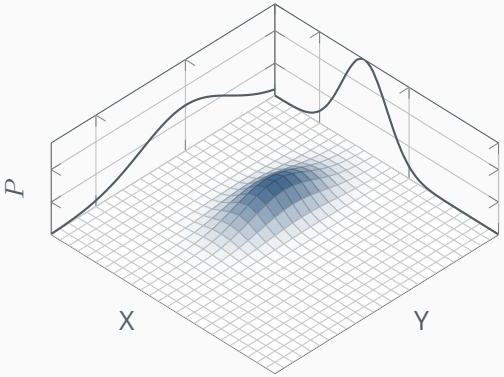
Learning the data distribution

So what is it we want exactly?

- $P(Y|X)$ discriminative model (classification)
- $P(X|Y)$ conditional generative model
- $P(X,Y)$ generative model

We want to learn the probability density function of our data (natures distribution)

The data distribution $P(X,Y)$

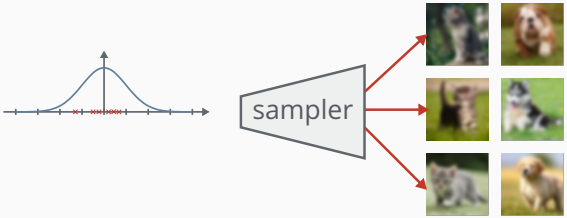


Generative models definition

Definition: Generative models learn a joint distribution over the entire dataset. They are mostly used for sampling applications or density estimation:

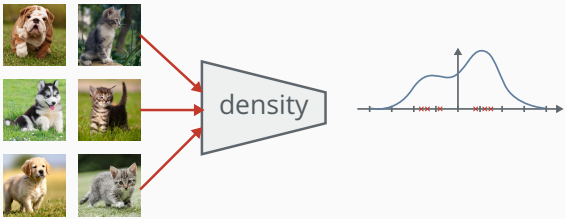
Sampling the model

A generative model learns to fit a model distribution over observations so we can sample novel data from the model distribution, $\mathbf{x}_{\text{new}} \sim p_{\text{model}}(\mathbf{x})$



Density estimation

Density estimation is estimating the probability of observations. Given a datapoint $\mathbf{x}$, what is the probability assigned by the model, $p_{\text{model}}(\mathbf{x})$?



Examples

Linguists

- What is the probability of a sentence? $P(\text{sentence})$
  - $P(\text{`the dog chased after the ball'})$
  - $P(\text{`printers eat avocados when sad'}) \approx 0$

Meteorologists

- What is the probability of whether it will rain? $P(\text{rain})$

Artists

- What is the probability of this image being a face? $P(\text{face})$

Musicians

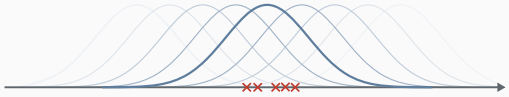
- What is the probability this sounds like Beethoven? $P(\text{Beethoven})$

Definition: maximum likelihood estimation

Maximum likelihood estimation (MLE) is a method for estimating the parameters of a probability distribution by maximizing a likelihood function, so that under the model the observed data is most probable

$$\begin{aligned}\theta^* &= \arg\max_\theta p_{\text{model}}(\mathbb{X};\theta) \\ &= \arg\max_\theta \prod_{i=1}^{n} p_{\text{model}}(\mathbf{x}^i;\theta) \\ &\approx \arg\max_\theta \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_{\text{model}}(\mathbf{x};\theta)],\end{aligned}$$

where $\mathbb{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^n\}$ are from $p_{\text{data}}(\mathbf{x})$

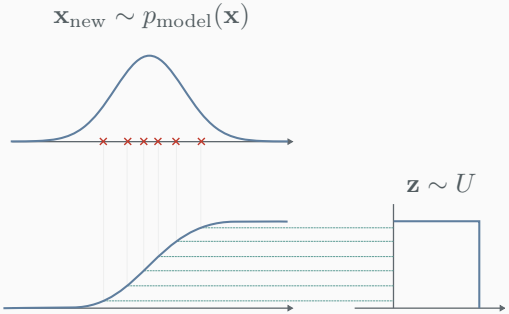


Example: cumulative distribution sampling

Given the CDF $F_X(\mathbf{x})$, the antiderivative of $f_X(\mathbf{x}) = p_{\text{model}}(\mathbf{x})$, e.g. where $F'(\mathbf{x}) = p_{\text{model}}(\mathbf{x})$

$$F_X(\mathbf{x}) = \int_{-\infty}^{\mathbf{x}} f_X(\mathbf{u})\, \mathrm{d}\mathbf{u}$$

we can sample new data by transforming random values $\mathbf{z}$ from the uniform distribution $\mathbf{z} \sim U$ via the inverse of the CDF $F_X^{-1}(\mathbf{z})$.

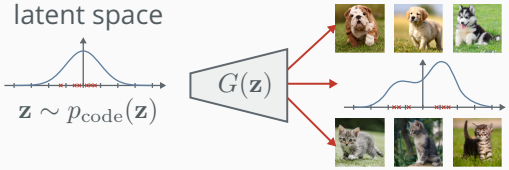


Generative networks definition

Definition: generative networks

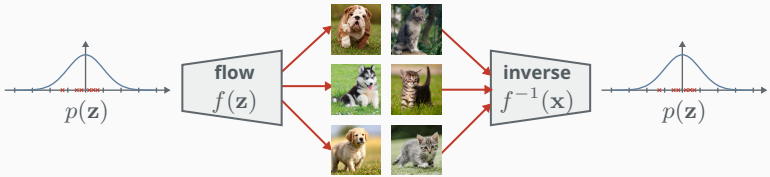
The goal of generative networks is to take some simple distribution, like a normal distribution or a uniform distribution, and apply a non-linear transformation (e.g. a deep neural network) to obtain samples from $p_{\text{data}}(\mathbf{x})$

In 1D, we can say $G = F^{-1}_{\text{data}}(\mathbf{x})$ and sample $\mathbf{z} \sim U$, and similarly in ND — but assuming the determinant of the Jacobian and the inverse of G are computable, which is a large restriction. Ideally we want $\mathbf{z}$ in low dimensions



Definition: flow models

Flow models restrict our function to be a chain of invertible functions, called a flow, therefore the whole function is invertible.



Definition: the Jacobian matrix

The collection of all first-order partial derivatives of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$

$$\boldsymbol{J}_f = \nabla_{\mathbf{x}} f = \frac{\mathrm{d}f}{\mathrm{d}\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix},$$

$$\boldsymbol{J}_f(i, j) = \frac{\partial f_i}{\partial x_j}$$

Definition: the determinant

The determinant of an $n \times n$ square matrix M is a scalar value that determines the factor of how much a given region of space increases or decreases by the linear transformation of M:

$$\det M = \det \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \sum_{j_1 j_2 \dots j_n} (-1)^{\tau(j_1 j_2 \dots j_n)} a_{1 j_1} a_{2 j_2} \dots a_{n j_n}$$

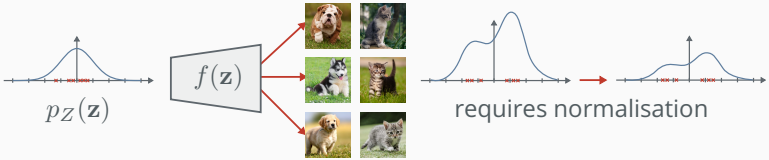
Watch a 3Blue1Brown's video here

PyTorch: `torch.det(M)`, for example: `torch.det(torch.eye(3,3))` returns `1.0` and `torch.det(torch.tensor([[3.,2.],[0.,2.]]))` returns `6.0`

Definition: the change of variables theorem

Given $p_Z(\mathbf{z})$ where $\mathbf{x} = f(\mathbf{z})$ and $\mathbf{z} = f^{-1}(\mathbf{x})$ we ask what is $p_X(\mathbf{x})$?

$$p_X(\mathbf{x}) = p_Z(f^{-1}(\mathbf{x})) \left| \det \left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$



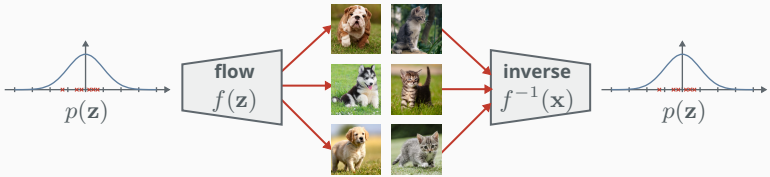
Normalising flows definition

Definition: normalising flows

Normalising flows $f : \mathbb{R}^n \to \mathbb{R}^n$ transform and renormalise a sample $\mathbf{z} \sim p_\theta(\mathbf{z})$ through a chain of bijective transformations f, where:

$$\mathbf{x} = f_\theta(\mathbf{z}) = f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z})$$

$$\log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{z}) + \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial f_i^{-1}}{\partial \mathbf{z}_i} \right) \right|$$



Easy to compute determinants

We have a sequence of high-dimensional bijective functions, where we need to compute the Jacobian determinants.

Computing the determinants can be expensive, so most of the literature focuses on restricting the function f^{-1} to those with easy-to-compute Jacobian determinants.

This is done by ensuring the Jacobian matrix of the functions is triangular.

Definition: triangular Jacobian

If the Jacobian is lower triangular:

$$J = \begin{bmatrix} a_{1,1} & & & & 0 \\ a_{2,1} & a_{2,2} & & & \\ a_{3,1} & a_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

then the determinant is simply the product of its diagonals.

Normalising flows normalising flow layers

Description	Function	Log-Determinant
Additive Coupling [1]	$\mathbf{y}^{(1:d)} = \mathbf{x}^{(1:d)}$ $\mathbf{y}^{(d+1:D)} = \mathbf{x}^{(d+1:D)} + f(\mathbf{x}^{(1:d)})$	0
Planar [2]	$\mathbf{y} = \mathbf{x} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$ With $\mathbf{w} \in \mathbb{R}^D$, $\mathbf{u} \in \mathbb{R}^D$, $\mathbf{b} \in \mathbb{R}$	$\ln\lvert 1 + \mathbf{u}^T h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}\rvert$
Affine Coupling [3]	$\mathbf{y}^{(1:d)} = \mathbf{x}^{(1:d)}$ $\mathbf{y}^{(d+1:D)} = \mathbf{x}^{(d+1:D)} \odot f_\sigma(\mathbf{x}^{(1:d)}) + f_\mu(\mathbf{x}^{(1:d)})$	$\sum_1^d \ln\lvert f_\sigma(x^{(i)})\rvert$
Batch Normalization [3]	$\mathbf{y} = \frac{\mathbf{x} - \tilde{\mu}}{\sqrt{\tilde{\sigma}^2 + \epsilon}}$	$-\frac{1}{2}\sum_i \ln\left(\tilde{\sigma}_i^2 + \epsilon\right)$
1x1 Convolution [4]	With $h \times w \times c$ tensor $\mathbf{x}$ & $c \times c$ tensor $\mathbf{W}$ $\forall i, j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$h \cdot w \cdot \ln\lvert\det \mathbf{W}\rvert$
i-ResNet [5]	$\mathbf{y} = \mathbf{x} + f(\mathbf{x})$ where $\lVert f\rVert_L < 1$	$\mathrm{tr}(\ln(\mathbf{I} + \nabla_{\mathbf{x}} f)) =$ $\sum_{k=1}^{\infty}(-1)^{k+1}\frac{\mathrm{tr}((\nabla_{\mathbf{x}} f)^k)}{k}$
Emerging Convolutions [6]	$\mathbf{k} = \mathbf{w}_1 \odot \mathbf{m}_1, \quad \mathbf{g} = \mathbf{w}_2 \odot \mathbf{m}_2$ $\mathbf{y} = \mathbf{k} \star_l (\mathbf{g} \star_l \mathbf{x})$	$\sum_c \ln\lvert\mathbf{k}_{c,c,m_y,m_x}\mathbf{g}_{c,c,m_y,m_x}\rvert$

References I

[1] Laurent Dinh, David Krueger, and Yoshua Bengio. “Nice: Non-linear independent components estimation”. In: arXiv preprint arXiv:1410.8516 (2014).

[2] Danilo Jimenez Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: arXiv preprint arXiv:1505.05770 (2015).

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using real nvp”. In: arXiv preprint arXiv:1605.08803 (2016).

[4] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: Advances in neural information processing systems. 2018, pp. 10215–10224.

[5] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. “Invertible residual networks”. In: International Conference on Machine Learning. 2019, pp. 573–582.

[6] Emiel Hoogeboom, Rianne van den Berg, and Max Welling. “Emerging convolutions for generative normalizing flows”. In: arXiv preprint arXiv:1901.11137 (2019).