Deep Learning

Lecture 6: Diffusion and flows

Chris G. Willcocks

Durham University

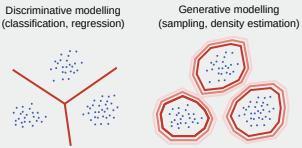
Part 1: Diffusion overview

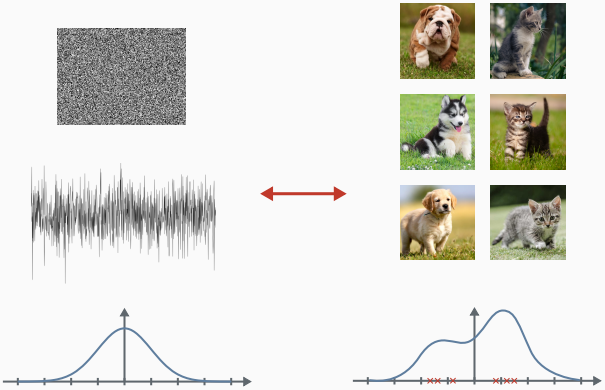
1. **High-level modelling trilemma**
2. **Manifolds**
3. **Energy-based models**
   - definition
   - GANs as energy-based models
   - clustering as an energy-based model
   - softmax and softmin
   - exact likelihood
   - Boltzmann machines definition
   - restricted and deep Boltzmann machines
4. **Probabilistic diffusion models**
   - approach and implementation
   - flow matching
   - diffusion-based anomaly detection
   - VQ-GAN-EBM hybrids

Recap: Generative models

Generative models

- Learn a distribution over a dataset.
- Map between noise and data.
- Noise (prior) can be different dimension to data.





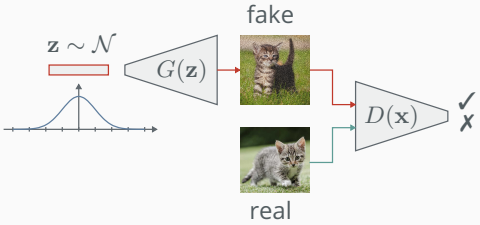
Recap: GANs

Definition: generative adversarial networks

A generative adversarial network (GAN) is a non-cooperative zero-sum game where two networks compete against each other [1].

One network $G(\mathbf{z})$ generates new samples, whereas D estimates the probability the sample was from the training data rather than G:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))].$$



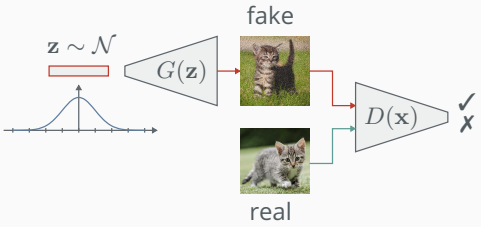
Recap: GANs

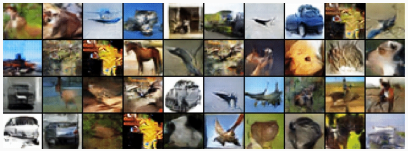
GAN **properties**

GANs benefit from differentiable data augmentation, but are otherwise have a variety of issues:

- Non-convergence
- Diminishing gradient
- Difficult to balance
- Mode collapse

Link to Colab example





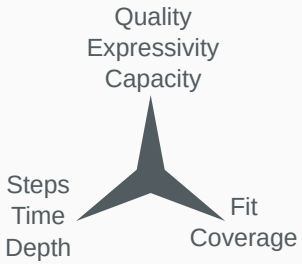
Reflection

Question: the trilemma?

- They are fast (quick to sample)
- They have very poor coverage
- They have excellent quality

...2/3 ain't bad.

The generative modelling trilemma
(empirical observation)



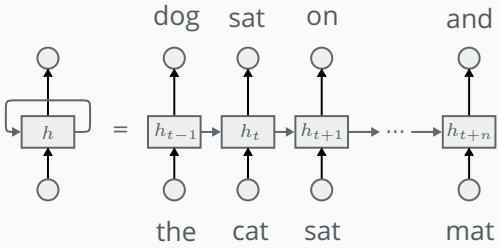
Autoregressive generative models

Definition: autoregressive (AR) generative models (e.g. chatGPT)

AR models maximise the likelihood of the training data (excellent mode coverage):

$$p_\theta(\mathbf{x}) = p_\theta(x_1, ..., x_N) = \prod_{i=1}^{N} p_\theta(x_i | x_1, ..., x_{i-1})$$

This is slow due to the sequential nature defined by the chain rule of probability.



Today: Diffusion probabilistic models (DPMs)

Definition: diffusion probabilistic models (DPMs)

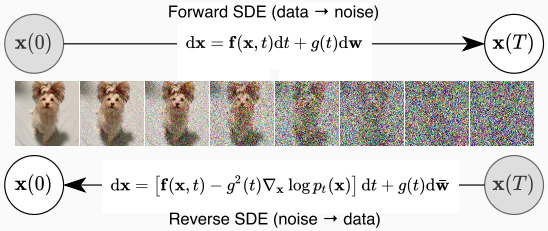
Instead of modelling a sequence of words, model a diffusion of noise in the data space. Define a forward (diffusion) equation:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t),$$

which can be reversed to sample the model:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}).$$

This also requires a long iterative **transformation process**.



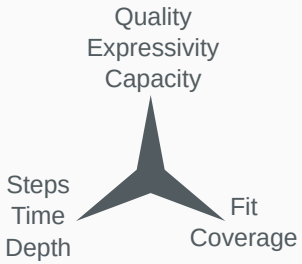
Reflection

Question: where are these in the trilemma?

- They are slow (lots of iterations)
- They have excellent coverage
- They have excellent quality

...I'd do anything for 3/3...

The generative modelling trilemma
(empirical observation)



Manifold definition

Definition: manifold

A manifold is a topological space that locally resembles Euclidean space

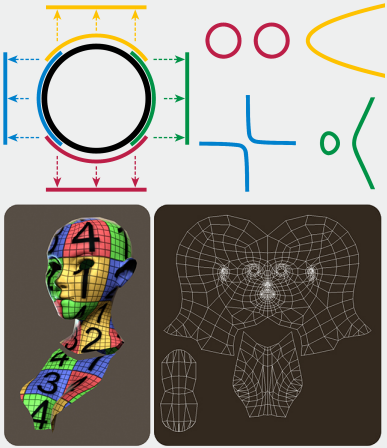
- topological manifold
- differentiable manifold
- Riemannian manifold

Definition: embedding

An embedding is a function ϕ that maps a manifold $\mathcal{M}$ to a new manifold $\mathcal{N}$ in an injective way that preserves its structure:

$$\phi : \mathcal{M} \to \mathcal{N}$$

Example: manifolds



Manifold charts by KSmrq, distributed under CC. UV mesh example by Antony Ward.

Energy-based models definition

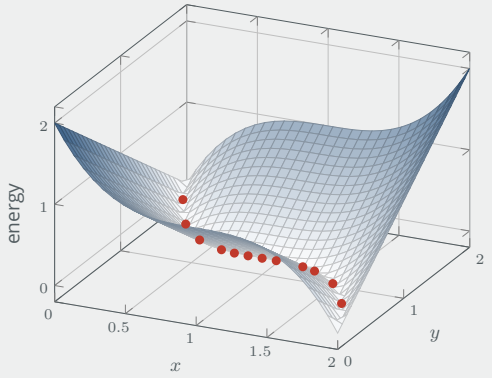
Definition: energy-based models

These are just any function that is happy when you input something that looks like data, and is not happy when you input something that doesn't look like data.

$$E(\mathbf{x}) = 0 \ \checkmark$$
$$E(\tilde{\mathbf{x}}) > 0 \ \times$$

This generic definition fits a large majority of machine learning models. For example $\mathcal{L}(E(\mathbf{x}), \mathbf{y})$ (a classifier)

Energy increases off manifold



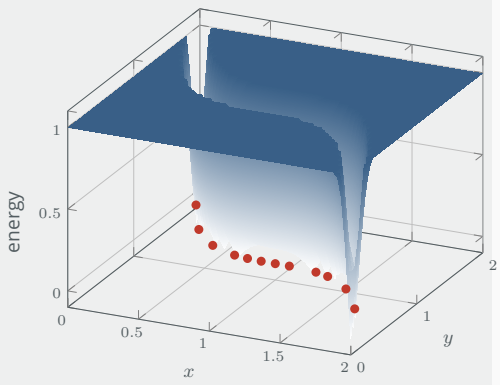
Definition: energy-based models

GANs are also energy models. The generator G generates samples off the manifold, then the descriminator D says these should be one everywhere, whereas it says real samples should be zero everywhere.

The generator also has to get good at sampling points on the data manifold. So it has to learn to generate points in the valley regions.

Is this smooth? What does a 1-Lipschitz discriminator do to the energy landscape?

GAN energy



Definition: clustering algorithm

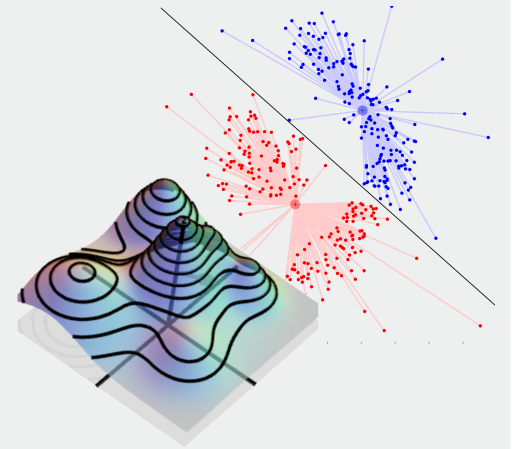
A cluster is a **connected-component** of a **level-set** of the **unknown PDF** over our data observations.

Traditionally:

- We don't know the PDF (the energy landscape)
- We don't necessarily know the level set
  - although 0.5 is appropriate for BCE
- This can be expensive (deep learning)

Click to watch a video that visually explains from the definition

Example: clustering by its definition



Definition: softmax and softmin

Softmax and softmin functions rescale elements to be in the range [0, 1] and such that they sum to 1. So they create a probability mass function, e.g.:

$$\begin{bmatrix} 1.3 \\ 7.2 \\ 2.4 \\ 0.5 \\ 1.1 \end{bmatrix} \rightarrow \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{K} e^{\mathbf{z}_j}} \rightarrow \begin{bmatrix} 0.0027 \\ 0.9858 \\ 0.0081 \\ 0.0012 \\ 0.0022 \end{bmatrix}$$

Softmax functions are widely used (not just for EBMs) where a distribution is needed, such as the last layer of a classifier.

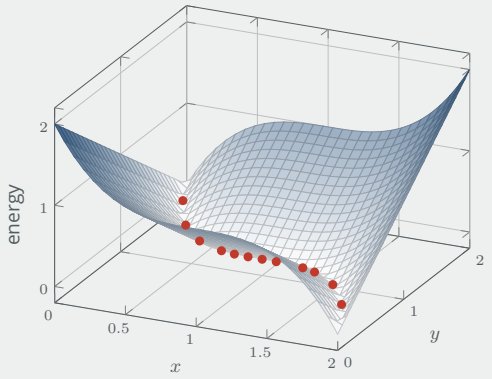
Challenges: energy-based models

EBMs are based on the observation that any probability density function $p(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^n$ can be expressed as:

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{\int_{\tilde{\mathbf{x}} \in \mathcal{X}} e^{-E(\tilde{\mathbf{x}})}},$$

where $E(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is the energy function. However computation of the integral is intractable [2] for most models.

Energy increases off manifold



Boltzmann machines definition

Definition: Boltzmann machine

Boltzmann machines [3] are one of the earliest neural networks for modeling binary data. They can associate the probability of the visible vectors $\mathbf{v}$ using finite summations:

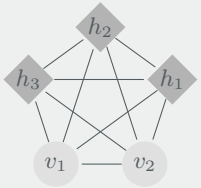
$$p_\theta(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-\beta E_\theta(\mathbf{v},\mathbf{h})}}{\sum_{\tilde{\mathbf{v}}} \sum_{\mathbf{h}} e^{-\beta E_\theta(\tilde{\mathbf{v}},\mathbf{h})}}$$

They are typically trained via negative log-likelihood through contrastive divergence, where the weights are updated:

$$\sum_{\mathbf{x}\in\mathcal{X}} \frac{\partial \ln p(\mathbf{x})}{\partial w_{i,j}} = \mathbb{E}_{p_\text{d}}[\mathbf{v}\mathbf{h}^T] - \mathbb{E}_{p_\text{model}}[\mathbf{v}\mathbf{h}^T]$$

Example: Boltzmann machine

They are an energy model which just have visisble layers $v_1, v_2, ..., v_n$ (inputs) and hidden layers $h_1, h_2, ..., h_n$ (no outputs):



This example Boltzmann Machine has 2 visible units and 3 hidden units.

Definition: RBMs and DBMs

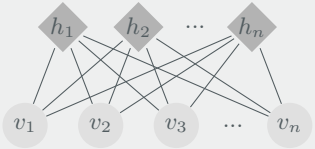
Restricted Boltzmann Machines (RBMs) and Deep Boltzmann Machines (DBMs) are Boltzmann machines with a more restricted (bipartite) graph structure [4]. DBMs have additional hidden layers.

That means that the visible units conditional on the hidden units become independant, which makes training these straightforward in practice.

Link to Colab **Good YouTube talk**

Example: RBM

RBMs have a restricted architecture architecture so that there are no connections between hidden units:



DBMs are like the above, but with multiple hidden layers between.

Diffusion probabilistic modelling

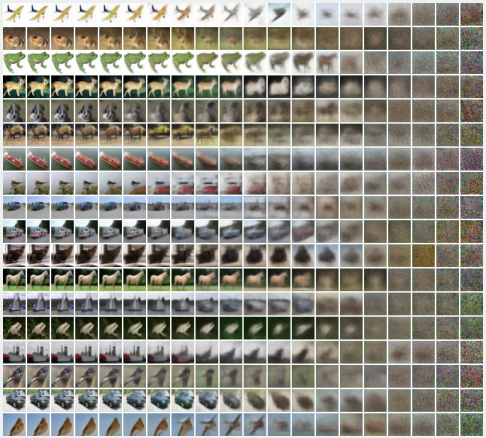
Diffusion probabilistic modelling approaches (such as DDPMs [5]) typically have a U-Net shaped architecture:

Data is gradually diffused in a forward process for T timesteps until it approximates the prior distribution.

The reverse process gradually removes noise, e.g. starting at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$ for T timesteps.

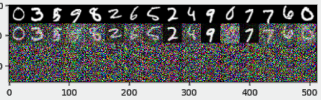
Like GANs, inject quality conditionals for better samples! (DALL-E 3, Stable Diffusion).

Example: CIFAR10 samples from [5]



DDPM implementation covered in the practicals

Predicting noise with U-Net




```
class DDPM(nn.Module):
    def init (self,):
        super(DDPM, self). init ()
        self.net = nn.Sequential(
                nn.Conv2d(in_channels=3, out_channels=64, 3,2,1),
                nn.BatchNorm2d(64),
                nn.ReLU(),
                nn.Conv2d(in_channels=64, out_channels=3, 3,2,1)
            )

    # algorithm 1 in DDPM paper (simplified)
    def forward(self, x):
        ts = torch.randint(1, T+1, (x.shape[0],)).to(x.device)
        eps = torch.randn like(x)
        x_t = nonlinear_blend(x, eps, ts)
    return F.mse loss(eps, self.net(x_t))

    # algorithm 2 (simplified)
    def sample(self, n_sample , size):
        x_i = torch.randn(n_sample, *size).to(device)
        for i in range(T, 0, -1):
                z = torch.randn(n_sample , *size).to(device) if i > 1 else 0
            eps = self.net(x_i)
            x_i = schedule_func(x_i, eps, z, i)
        return x_i
```

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Flow matching straight transition paths via optimal transport

Definition: Flow matching

Flow matching [6] defines a simpler **ODE** between prior and data distribution:

$$\frac{dx_t}{dt} = v_\theta(x_t, t), \quad t \in [0, 1]$$

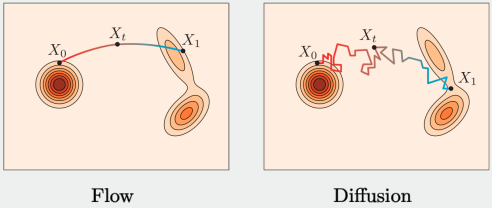
where $x_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (noise) and $x_1 \sim p_{\text{data}}$ (data) and x_t is e.g. a linear interpolant: $x_t = (1-t)x_0 + tx_1$

The model learns to predict velocity fields $(x_1 - x_0)$ by minimising:

$$\mathcal{L} = \mathbb{E}_{x_0, x_1, t}\left[\|v_\theta(x_t, t) - (x_1 - x_0)\|^2\right]$$

Theory: Straight (optimal transport) paths in the transition space $\Rightarrow$ simpler and faster.

Comparison: straight vs. curved paths



Flow matching learns direct paths through the probability landscape, while DDPMs follow noisy diffusion trajectories.

Link to Colab example

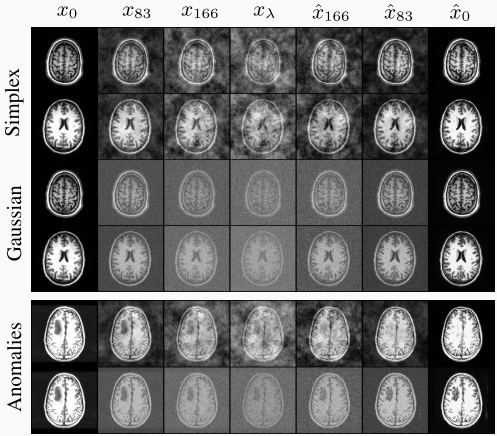
Diffusion-based anomaly detection

Diffusion-based anomaly detection

Like GANs, diffusion-based models work well for anomalies (great for small datasets).

- Do a partial diffusion
- Train only on healthy/normal data
- Abnormal denoising will only know how to make the data look normal
- Any error = surprise = anomalies

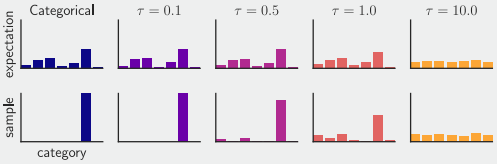
Our paper, AnoDDPM [7] (CVPR NTIRE), uses simplex noise to capture multi-scale anomalies. See also UNIT-DDPM [8] (unpaired translation).



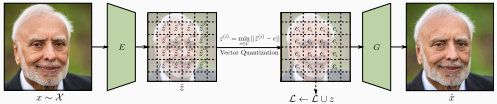
Link to project page

Vector quantization

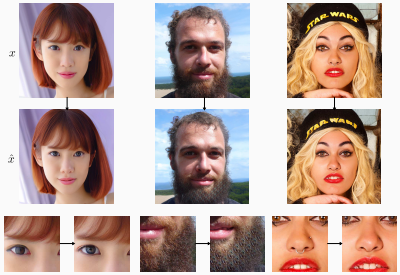
Imposing a discrete prior on the latents can be achieved with either variational or adversarial (non-blurry) approaches.



The Gumbel-Softmax distribution interpolates between discrete one-hot-encoded categorical distributions and continuous categorical densities.



Above: vector quantisation. **Below:** shift mode collapse to perceptually unimportant parts of the signal.



Our hybrids [9, 10] 2 seconds generation, 2 days training, single GTX 1080Ti

Part 2: Normalising flows overview

1 Generative model fundamentals

- 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

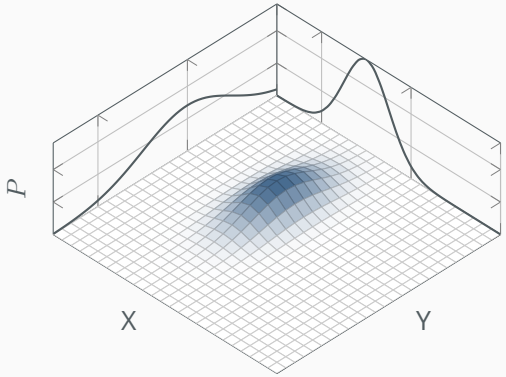
Learning the data distribution

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 $P(X,Y)$

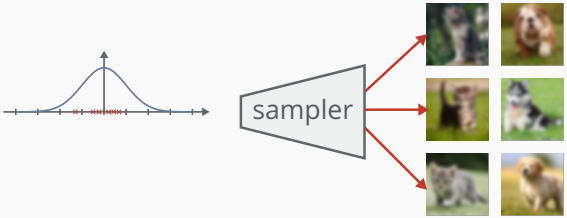


Generative models definition

Definition: Generative models learn a joint distribution over the entire dataset with some target variable(s). They are mostly used for sampling applications or density estimation:

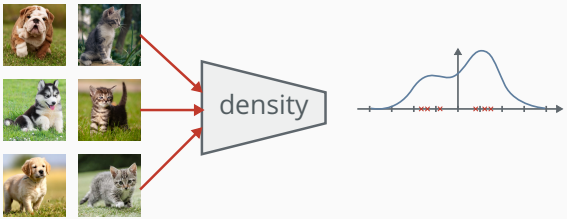
Sampling the model

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

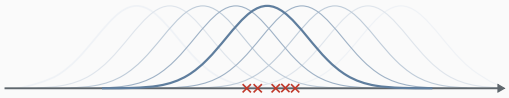
- What is the probability this sounds like Beethoven? $P(\text{Beethoven})$

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 $p_{\text{data}}(\mathbf{x})$

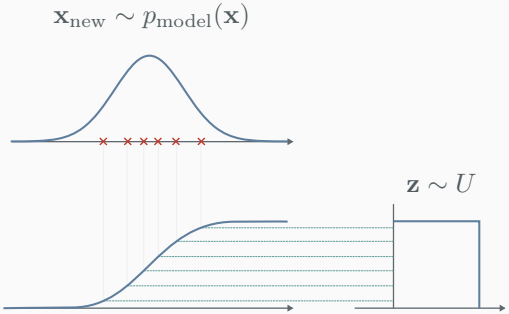


Example: cumulative distribution sampling

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 $F_X^{-1}(\mathbf{z})$.

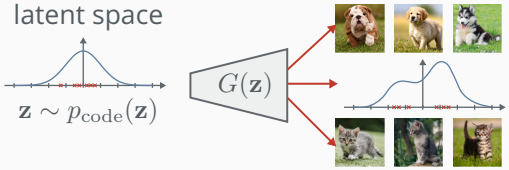


Generative networks definition

Definition: generative networks

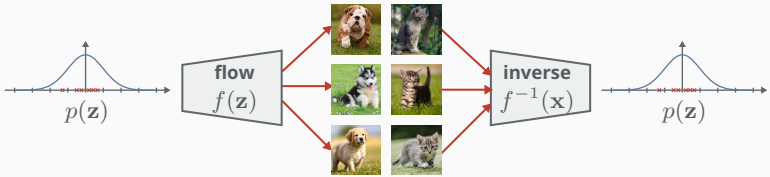
The goal of generative networks is to take some simple distribution, like a normal distribution or a uniform distribution, and apply a non-linear transformation (e.g. a deep neural network) to obtain samples from $p_{\text{data}}(\mathbf{x})$

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:

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

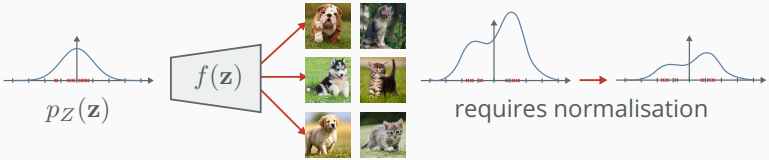
Watch a 3Blue1Brown's video here

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 $p_X(\mathbf{x})$?

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



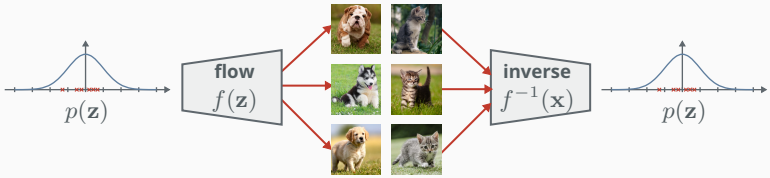
Normalising flows definition

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]	$\mathbf{k} = \mathbf{w}_1 \odot \mathbf{m}_1, \quad \mathbf{g} = \mathbf{w}_2 \odot \mathbf{m}_2$ $\mathbf{y} = \mathbf{k} \star_l (\mathbf{g} \star_l \mathbf{x})$	$\sum_c \ln\lvert\mathbf{k}_{c,c,m_y,m_x}\mathbf{g}_{c,c,m_y,m_x}\rvert$

Part 1: References I

[1] Ian Goodfellow et al. “Generative adversarial nets”. In: Advances in neural information processing systems. 2014, pp. 2672–2680.

[2] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. “A tutorial on energy-based learning”. In: Predicting structured data 1.0 (2006).

[3] Geoffrey E Hinton and Terrence J Sejnowski. “Optimal perceptual inference”. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. Vol. 448. Citeseer. 1983.

[4] Geoffrey E Hinton. “Training products of experts by minimizing contrastive divergence”. In: Neural computation 14.8 (2002), pp. 1771–1800.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: arXiv preprint arXiv:2006.11239 (2020).

[6] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. “Flow Matching for Generative Modeling”. In: The Eleventh International Conference on Learning Representations. 2023. URL: https://openreview.net/forum?id=PqvMRDCJT9t.

Part 1: References II

[7] Julian Wyatt, Adam Leach, Sebastian M Schmon, and Chris G Willcocks. "AnoDDPM: Anomaly Detection With Denoising Diffusion Probabilistic Models Using Simplex Noise". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022, pp. 650–656.

[8] Hiroshi Sasaki, Chris G Willcocks, and Toby P Breckon. "Unit-ddpm: Unpaired image translation with denoising diffusion probabilistic models". In: arXiv preprint arXiv:2104.05358 (2021).

[9] Alex F McKinney and Chris G Willcocks. "Megapixel Image Generation with Step-Unrolled Denoising Autoencoders". In: arXiv preprint arXiv:2206.12351 (2022).

[10] Sam Bond-Taylor, Peter Hessey, Hiroshi Sasaki, Toby P Breckon, and Chris G Willcocks. "Unleashing Transformers: Parallel Token Prediction with Discrete Absorbing Diffusion for Fast High-Resolution Image Generation from Vector-Quantized Codes". In: European Conference on Computer Vision (ECCV) (2022). DOI: 10.1007/978-3-031-20050-2_11.

Part 2: 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).