4M24 Coursework Lecture

Score Matching and Diffusion Models

Alex Glyn-Davies, 18th November

Score Matching and Diffusion Models

Overview

- 25% of overall grade
- ≈10 hours work
- Python code provided
- Deadline: 21st January 2026
- Coursework sheet with questions
- Coursework code on GitHub

Deliverables

- Maximum 10 page report (including any figures & appendix)
- Answers to questions in Part I, II, III of coursework
- No need to include code

Generative Modelling

Text & Video generation - OpenAI valued $500 billion

Anthropic Founded 2021 - valued $183 billion

ChatGPT

Stable Diffusion

Gemini

perplexity

ANTHROP\C

Drug discovery

Value $1 billion

Insilico Medicine

Valued $2 billion

RECURSION

Valued $1.2 billion

RELAY® THERAPEUTICS

This coursework: Score-based Generative Models

Generative models

- Aim to learn a data distribution, such that samples can be taken
- Only have access to samples from the data distribution $\mathscr{D} = \{x_i\}_{i=1}^{N}$
- Generated from an *unknown* distribution $x_i \sim p(x)$
- Generative models approximate with a parameterised distribution $p_\theta(x)$

Score-based generative models approximate the *score* $\nabla_x \log p(x)$

Part I - Score and Sampling

Build a complex distribution from simpler base distributions

Gaussian distribution

$$p(x) = \frac{1}{(2\pi)^{n/2} \det |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

Mixture distribution

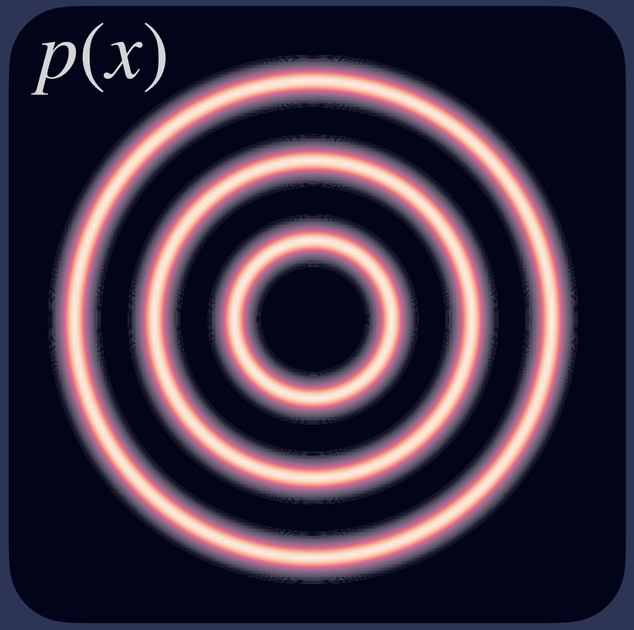
$$f(x) = \sum_{i=1}^{K} w_i p_i(x), \quad \text{with} \quad \sum_{i=1}^{K} w_i = 1$$

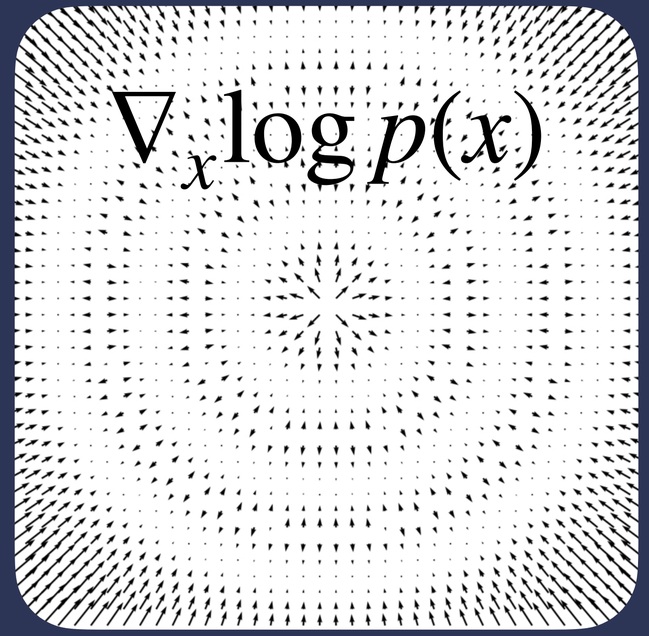
Derive and implement score functions $\nabla_x \log p(x)$

Sample from this distribution via Langevin dynamics

$$\mathrm{d}x_t = \nabla_x \log p(x_t)\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}B_t$$

(Lecture 12 spoiler) Langevin SDE

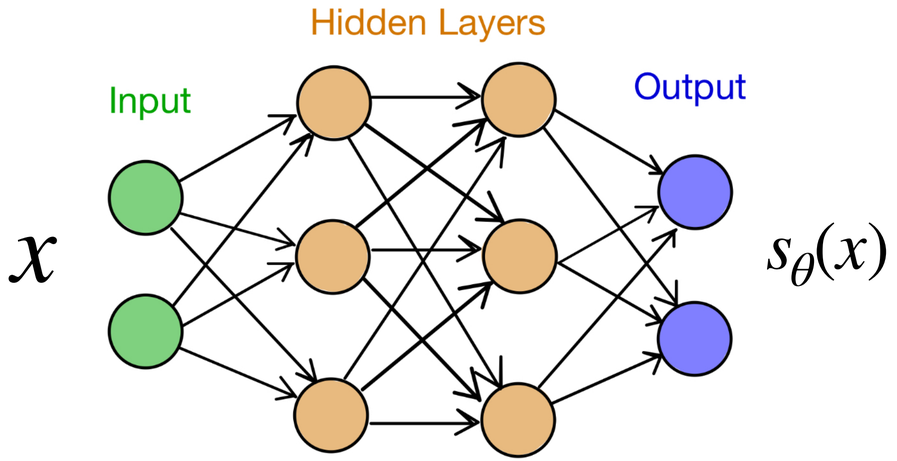




Part II - Sliced and Denoising Score Matching

Score matching: approximate true score with score-based model

$$s_\theta(x) = \nabla_x \log p_\theta(x)$$



s_θ **Neural network**

θ **Weights** $\{W_1, b_1, \ldots, W_L, b_L\}$

Using samples from the data distribution $x_i \sim p(x)$, you will estimate score $\nabla_x \log p(x)$

Score matching objective function

$$J_{\text{SM}}(\theta) = \frac{1}{2}\mathbb{E}_{x\sim p(x)}\left[\left\| \nabla_x \log p(x) - s_\theta(x) \right\|^2 \right]$$

You will implement and train **sliced-score matching**, and **denoising score matching** models

Part III - Diffusion Models

Target a sequence of perturbed data distributions

Perturbed distribution

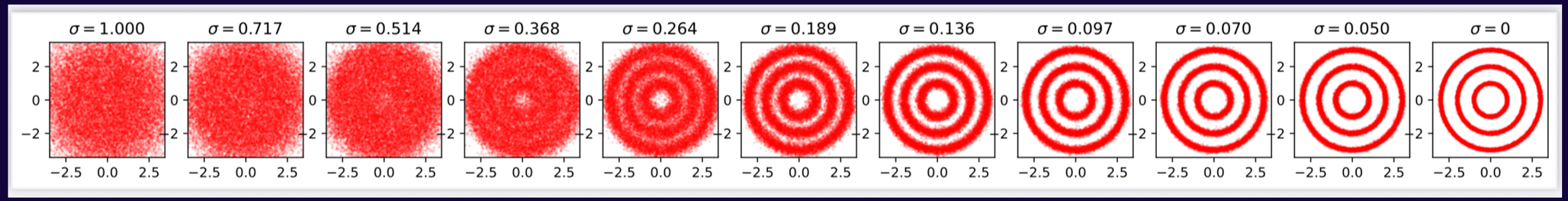
$$q_\sigma(\tilde{x}) = \int q_\sigma(\tilde{x} \mid x) p(x) \mathrm{d}x$$

Gaussian noise

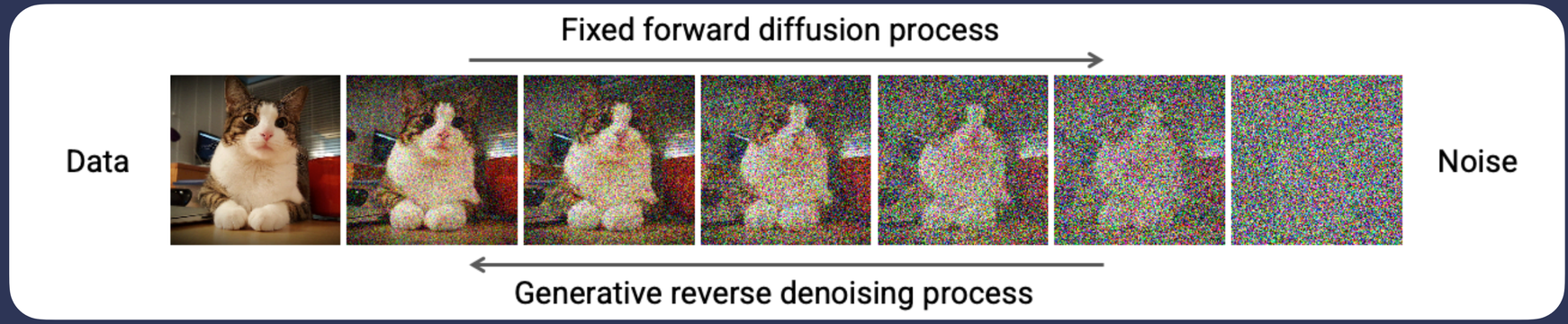
$$q_\sigma(\tilde{x} \mid x) = \mathcal{N}\left(x, \sigma^2 I\right)$$

Conditional NN

$$s_\theta(x, \sigma)$$



Foundation of the “diffusion model”



Coursework Sheet

Questions

- some theoretical
- some coding
- some written

Answer these in the report, with labels, e.g. II(a)

Include your derivations in the report.

Engineering Tripos Part IIB FOURTH YEAR

Coursework 4M24: Score Matching and Diffusion Models

Part I - Scores and Langevin Dynamics

In this section, we start by deriving and implementing the probability density, log-density and score of different distributions, where the score of a density $p(\mathbf{x})$ is defined as the gradient of the log-density, i.e. $s(\mathbf{x}) \coloneqq \nabla_{\mathbf{x}} \log p(\mathbf{x})$. We will compose these to form a complex distribution for testing out the Langevin sampling algorithms. It is recommended you read through the `JAX` tutorial notebook (`tutorials/jax.ipynb`) before attempting the implementations.

Probability Densities and Score Functions

Consider the multivariate Gaussian distribution: $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\mathbf{x} \in \mathbb{R}^n$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} \det |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

(a) Using the formula for the multivariate Gaussian as defined above, derive and implement the `Gaussian.pdf`, `Gaussian.logpdf` and `Gaussian.score` methods in `distributions.py`.

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is **spherically symmetric** provided it is constant on any sphere centered on the origin, i.e. for all $r \geq 0$, we have

$$f(\{\mathbf{x} \in \mathbb{R}^n \colon \|\mathbf{x}\| = r\}) = c(r),$$

where c is a constant depending on r. We say a distribution is spherically symmetric if its probability density function is spherically symmetric. Given a radial probability density $p(r)$, we can write the spherically symmetric density f as

$$f(\{\mathbf{x} \in \mathbb{R}^n \colon \|\mathbf{x}\| = r\}) = g(r)\, p(r),$$

where $g(\cdot)$ is a function that depends only on r.

(b) Derive this function $g(\cdot)$, and use this to complete the function `nSphere.g`, corresponding to the function $g(r)$.

Tutorials

4M24 Coursework – Score Matching and Diffusion Mo

This repository contains
Matching and Diffusion

We use `uv` , a fast Pytho
run the notebooks.

1. Install `uv`

Linux / macOS

```
curl -LsSf https://
```

If your system doesn't h

```
wget -qO- https://a
```

Windows (PowerShell)

```
powershell -c "irm
```

1 JAX Tutorial

This tutorial covers the key conc

Modules: - jax.numpy - jax.rand

Composable transforms: - jax.vm

Advanced Automatic Differentiat

[1]:
```
# Import the JAX library
import jax

# Import JAX's numpy librar
import jax.numpy as jnp

import matplotlib.pyplot as
```

2 jax.numpy Arrays

[2]:
```
# JAX arrays - initialised
m = jnp.array([0.,1.])
c = jnp.array([[1.,0.],[0.,

# Indexed the same:
print('first element of m:
print('first row of c: ', c
```

first element of m: 0.0

1 Equinox Tutorial

This tutorial covers the key concepts of the Equinox library, for neural network models. See the documentation for more info.

Other popular alternatives for JAX-based neural network libraries include Flax, and Haiku. Also, PyTorch and Tensorflow offer Python-based neural network libraries, as an alternative to JAX.

Here we cover:

Equinox: lightweight neural network library - eqx.Module - eqx.filter_jit, eqx.filter_grad

Optax: JAX-based optimisation library (used for first-order gradient descent)

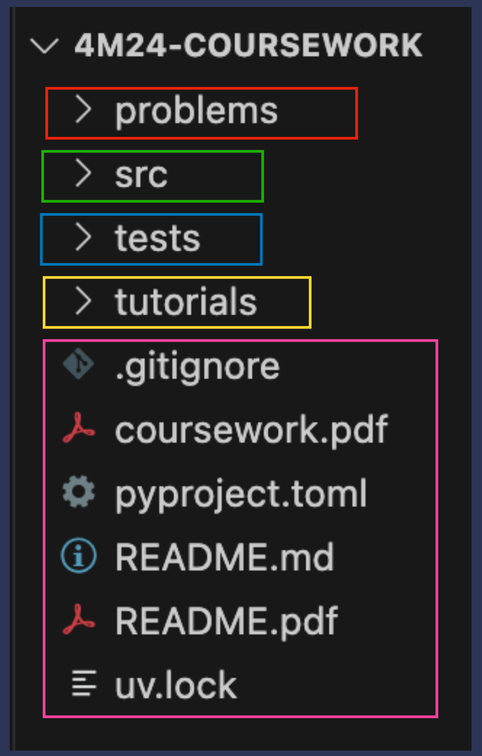
[1]:
```
import jax
import jax.numpy as jnp
import jax.tree_util as jtu

# Import the equinox package
import equinox as eqx

import matplotlib.pyplot as plt
```

First we will start with an example: learning a simple linear model.

Suppose we have real-valued data $y \in \mathbb{R}$, corresponsing to inputs $x \in \mathbb{R}^m$. We specify a general linear model is of the form:

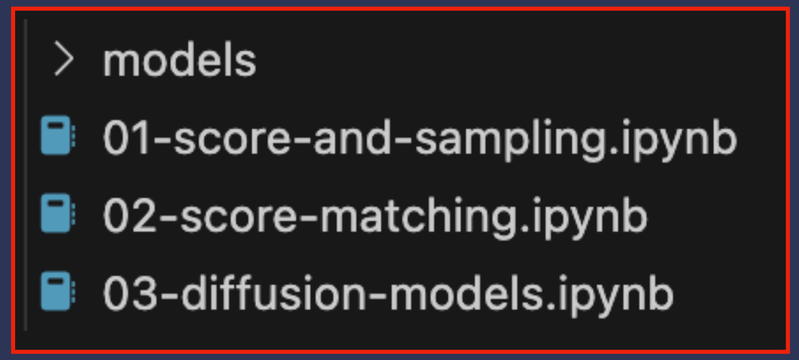
$$y = \beta_0 + \sum_{j=1}^{m} \beta_j x_j + \epsilon,$$

Folder structure



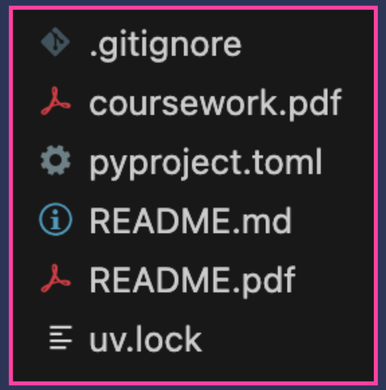
Code Structure

Jupyter notebooks
Where you will run the code.

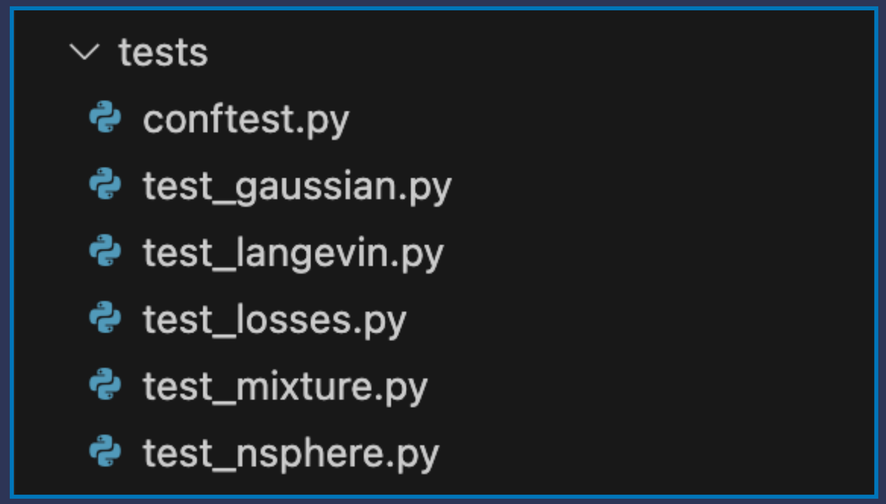


Source files - you will implement functions in these files

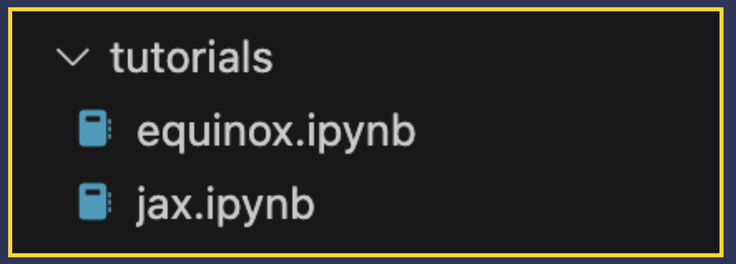
Package files - help for installing, questions.



Python tests for checking your implementations



Tutorial notebooks - read these first.



Example: You will implement code to sample random unit vectors

```python
@staticmethod
def log_sum_exp(a):
    """
    Compute log(sum(exp(a))) in a numerically stable way.
    Parameters:
        a: array
    Return:
        LogSumExp(a)
    """
    ###--- TODO ----------------------------------------
    return
```

```python
def sample(self, key, n_samples):
    r_key, u_key = jax.random.split(key)

    # Generate random unit vector in R^n
    u_vec = jnp.zeros((n_samples, self.n))
    ###--- TODO ----------------------------------------

    # Multiply unit vectors by sampled r~p(r)
    def scale(r, u):
        """ Scale unit vector u by factor r."""
        return r * u
    radii = self.r_dist.sample(r_key, n_samples)
    samples = jax.vmap(scale)(radii, u_vec)
    return samples
```

Anywhere with the **TODO** tag, you will need to add code.

Thanks for listening

Tomorrow (19th November) - Drop-in session
Before this:

- Install the code
- Run the tutorial notebooks

Any issues - come to the session.

Email: ag933@cam.ac.uk
Moodle: post on forum