Lecture overview

- What is a neural differential equation (NDE)?
- The link between NDEs and neural network architectures
- State of the art NDEs
  - Coupled oscillatory RNNs
  - Diffusion models

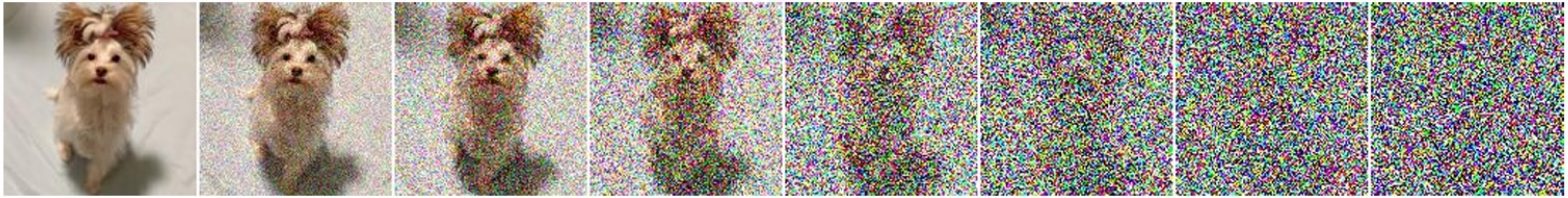
Learning objectives

- Be able to define an NDE
- Explain the connection between numerical PDE solvers and neural network architectures
- Be aware of state-of-the-art applications of NDEs

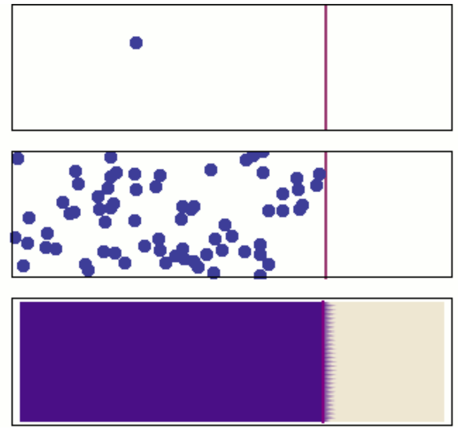
Diffusion models

Source: DALL·E 3, OpenAI

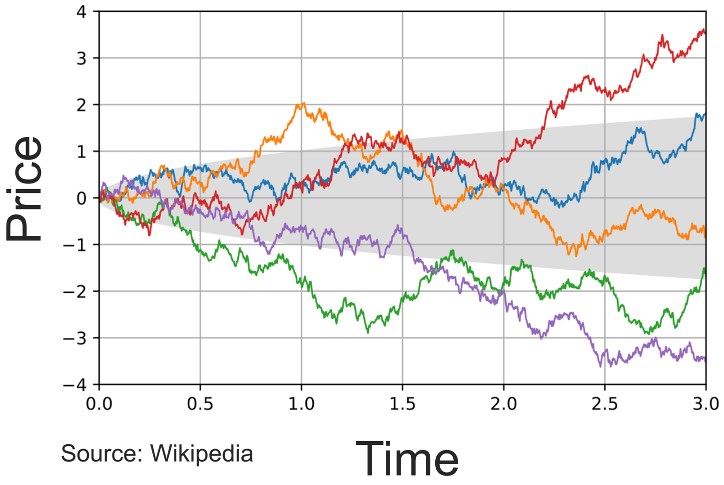
Diffusion



Diffusion = movement of (atoms, molecules, energy, prices, …) from a region of **higher** concentration to a region of **lower** concentration, driven by **random** motion



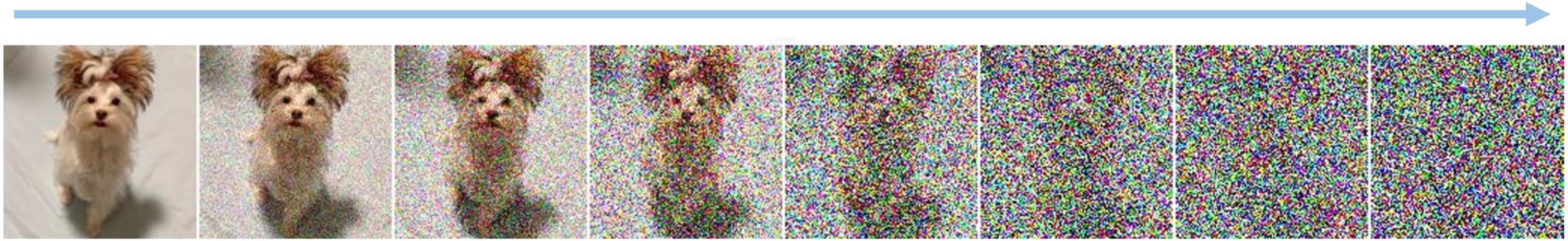
Source: Wikipedia



Source: Wikipedia

Image credit: Song et al, Score-Based Generative Modeling through Stochastic Differential Equations. ICLR, 2020

Diffusion – turning images into noise



$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

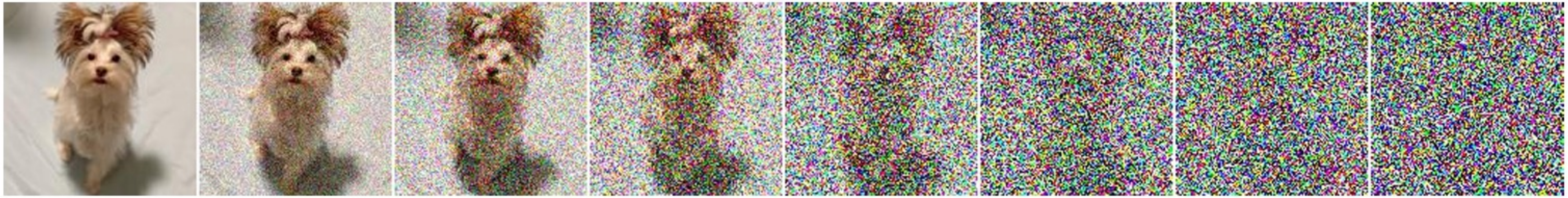
Forward model (stochastic differential equation):

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)dt + g(t)d\boldsymbol{w}$$

Drift function — Diffusion function — Wiener process (Add Gaussian noise at each step)

Image credit: Song et al, Score-Based Generative Modeling through Stochastic Differential Equations. ICLR, 2020

Diffusion – turning images into noise



$x_0 \sim p_0(x)$ $\qquad$ $x_T \sim p_T(x)$

Forward model (stochastic differential equation):

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)dt + g(t)d\boldsymbol{w}$$

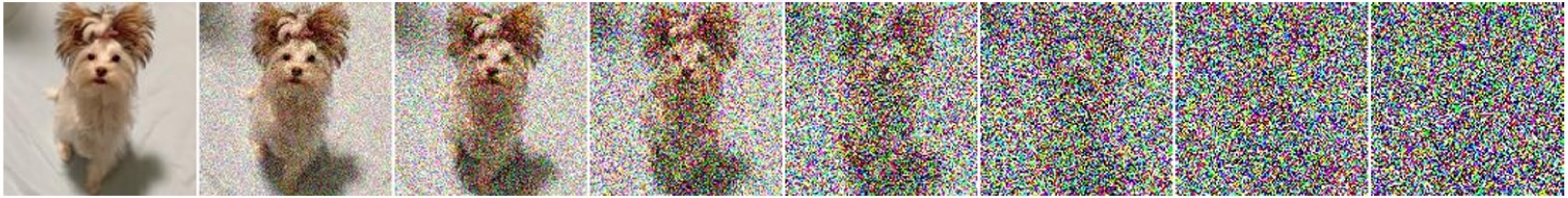
This SDE can be **reversed** by solving the following SDE **backwards in time** (starting at $\boldsymbol{x}_T$):

$$d\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t) - g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})]dt + g(t)d\bar{\boldsymbol{w}}$$

Anderson, Reverse-time diffusion equation models. Stochastic Processes and their Applications, 1982

Song et al, Score-Based Generative Modeling through Stochastic Differential Equations. ICLR, 2020

Diffusion – turning images into noise



$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

Forward model (stochastic differential equation):

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)dt + g(t)d\boldsymbol{w}$$

Moreover, it can be shown solving the following ODE backwards in time (starting at $\boldsymbol{x}_T$) results in trajectories drawn from the **same distribution** as the SDE:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$$

Anderson, Reverse-time diffusion equation models. Stochastic Processes and their Applications, 1982

Song et al, Score-Based Generative Modeling through Stochastic Differential Equations. ICLR, 2020

Diffusion models – turning noise into images

$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)

2. Solve the ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$$

Diffusion models – turning noise into images

$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)

2. Solve the ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$$

What's the problem here?

Diffusion models – turning noise into images

$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)

2. Solve the ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$$

Problem: we don't know what $p_t(\boldsymbol{x})$ is!

Diffusion models – turning noise into images

$\boldsymbol{x}_0 \sim p_0(x)$ $\qquad$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)

2. Solve the ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \underline{\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})}$$

Idea: use a neural network to **learn** the “score function”

Problem: we don’t know what $p_t(\boldsymbol{x})$ is!

$$\boldsymbol{s}(\boldsymbol{x}, t; \boldsymbol{\theta}) \approx \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$$

Diffusion models – turning noise into images

$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)

2. Solve the ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \underline{\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})}$$

Idea: use a neural network to **learn** the "score function"

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \boldsymbol{s}(\boldsymbol{x}, t; \boldsymbol{\theta})$$

And we now solve a **neural ODE** to generate an image

Diffusion models – learning the score function

We want the network to match the true score function, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_t \sim p_t} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) \right\|^2 \right]$$

Diffusion models – learning the score function

We want the network to match the true score function, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_t \sim p_t} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) \right\|^2 \right]$$

It can be shown that this is equivalent to

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_0 \sim p_0,\, \boldsymbol{x}_t \sim p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0)} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0) \right\|^2 \right] + C$$

Vincent, A connection between score matching and denoising autoencoders. Neural Computation, 2011

where $p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0)$ is the **transition** probability and C is a constant.

Diffusion models – learning the score function

We want the network to match the true score function, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_t \sim p_t} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) \right\|^2 \right]$$

It can be shown that this is equivalent to

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_0 \sim p_0,\, \boldsymbol{x}_t \sim p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) \right\|^2 \right] + C$$

Vincent, A connection between score matching and denoising autoencoders. Neural Computation, 2011

where $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is the **transition** probability and C is a constant.

Consider the simple forward SDE $d\boldsymbol{x} = d\boldsymbol{w}$ then

$$p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \frac{1}{\sqrt{2\pi t}} e^{-\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_0\|^2}{2t}} \Rightarrow \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) = -\frac{1}{t}(\boldsymbol{x}_t - \boldsymbol{x}_0)$$

Diffusion models – learning the score function

We want the network to match the true score function, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_t \sim p_t} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) \right\|^2 \right]$$

It can be shown that this is equivalent to

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_0 \sim p_0,\, \boldsymbol{x}_t \sim p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0)} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0) \right\|^2 \right] + C$$

Vincent, A connection between score matching and denoising autoencoders. Neural Computation, 2011

where $p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0)$ is the **transition** probability and C is a constant.

Consider the simple forward SDE $d\boldsymbol{x} = d\boldsymbol{w}$ then

$$p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0) = \frac{1}{\sqrt{2\pi t}} e^{-\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_0\|^2}{2t}} \Rightarrow \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0) = -\frac{1}{t}(\boldsymbol{x}_t - \boldsymbol{x}_0)$$

and

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_0 \sim p_0,\, \boldsymbol{x}_t \sim p_{0t}(\boldsymbol{x}_t | \boldsymbol{x}_0)} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \frac{1}{t}(\boldsymbol{x}_0 - \boldsymbol{x}_t) \right\|^2 \right] + C$$

Diffusion models – learning the score function

We want the network to match the true score function, i.e.

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_t \sim p_t} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_t(\boldsymbol{x}_t) \right\|^2 \right]$$

It can be shown that this is equivalent to

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_0 \sim p_0,\, \boldsymbol{x}_t \sim p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) \right\|^2 \right] + C$$

Vincent, A connection between score matching and denoising autoencoders. Neural Computation, 2011

where $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is the **transition** probability and C is a constant.

Consider the simple forward SDE $d\boldsymbol{x} = d\boldsymbol{w}$ then

$$p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \frac{1}{\sqrt{2\pi t}} e^{-\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_0\|^2}{2t}} \Rightarrow \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) = -\frac{1}{t}(\boldsymbol{x}_t - \boldsymbol{x}_0)$$

and

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, \boldsymbol{x}_0 \sim p_0,\, \boldsymbol{x}_t \sim p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[\left\| \boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \underline{\frac{1}{t}(\boldsymbol{x}_0 - \boldsymbol{x}_t)} \right\|^2 \right] + C$$

In this case – score function just predicts the noise added to image

Diffusion models – summary

$\boldsymbol{x}_0 \sim p_0(x)$ $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

1. Get lots of examples of $\boldsymbol{x}_0$

2. Assume some underlying SDE of the form $d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)dt + g(t)d\boldsymbol{w}$ with transition probability $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$

3. Train score function $\boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta})$ using

$$\mathcal{L}(\boldsymbol{\theta}) = E_{t,\, x_0 \sim p_0,\, x_t \sim p_{0t}(x_t|x_0)}\left[\left\|\boldsymbol{s}(\boldsymbol{x}_t, t; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}_t} \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)\right\|^2\right]$$

To generate an image:

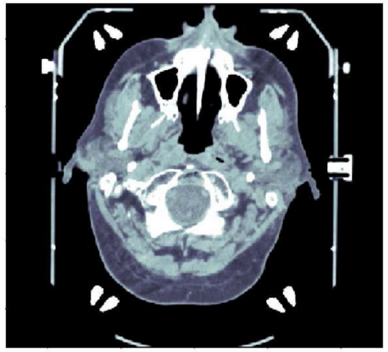
4. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)

5. Solve the neural ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \boldsymbol{s}(\boldsymbol{x}, t; \boldsymbol{\theta})$$

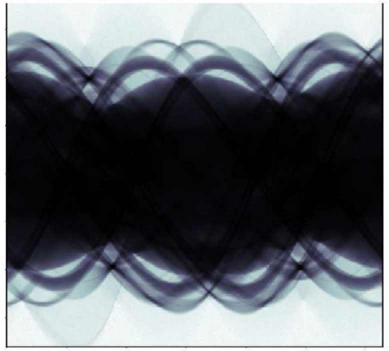
For more on diffusion models, see e.g.: Yang et al, Diffusion Models: A Comprehensive Survey of Methods and Applications, ArXiv 2024

Computed tomography – inverse problem



Ground truth computed tomography image

$a(x, y)$



Resulting tomographic data (sinogram)

$$b(\theta, \tau) = F(a) = I_0 e^{-\int_{l_{\theta,\tau}} a(x,y)\, ds}$$

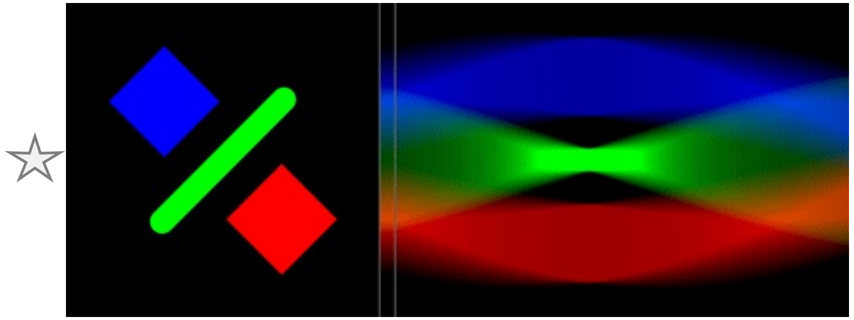
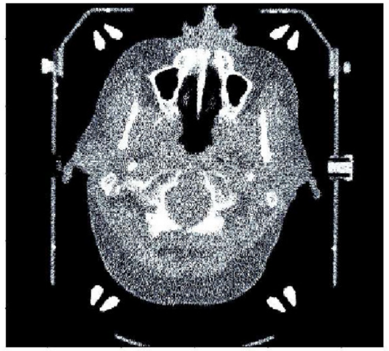
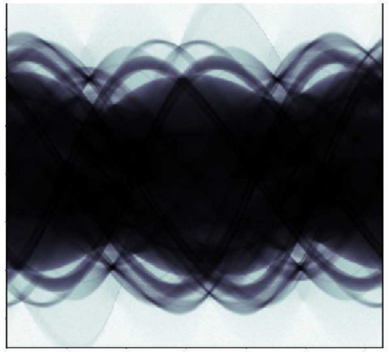


Image source: Wikipedia



Result of inverse algorithm

$\hat{a}$



Observed sinogram

b

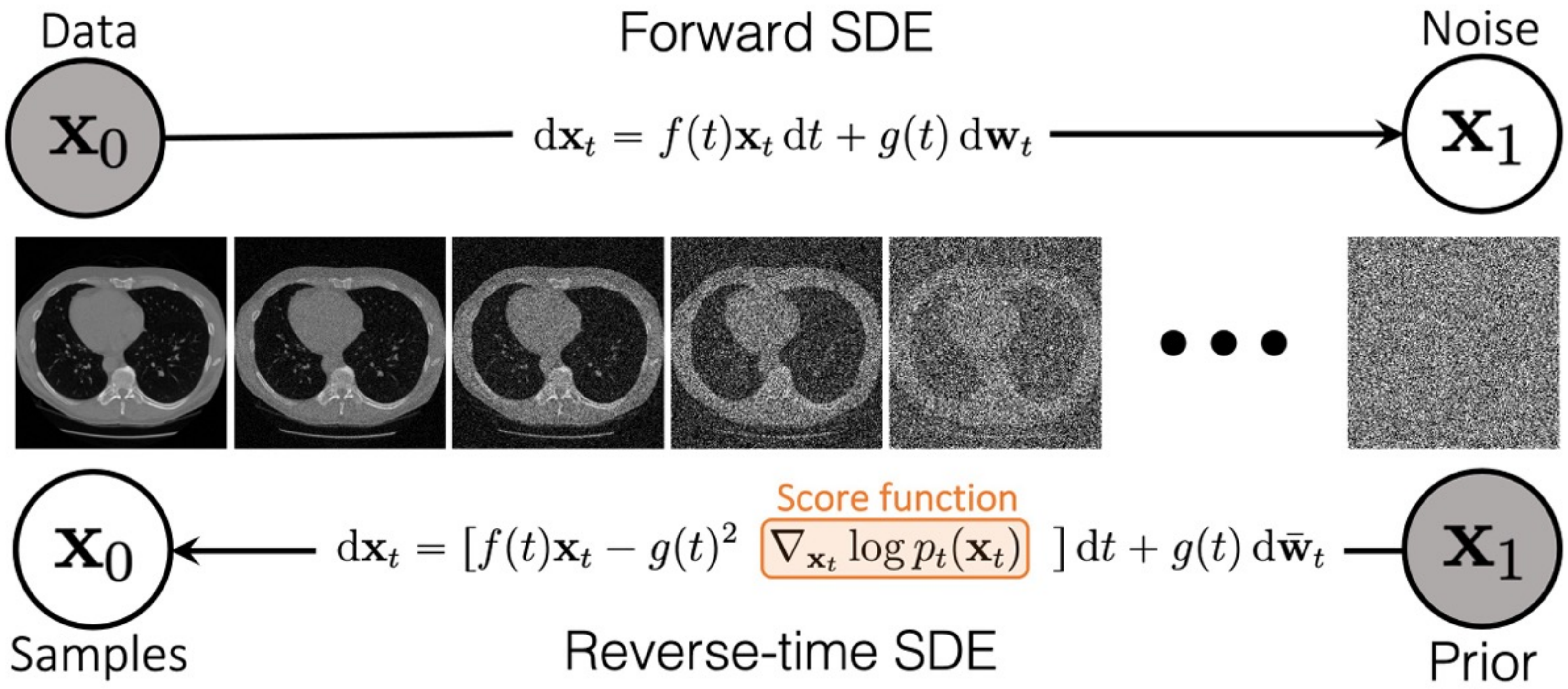
$$b = F(a)$$

a = set of input conditions

F = physical model of the system

b = resulting properties given F and a

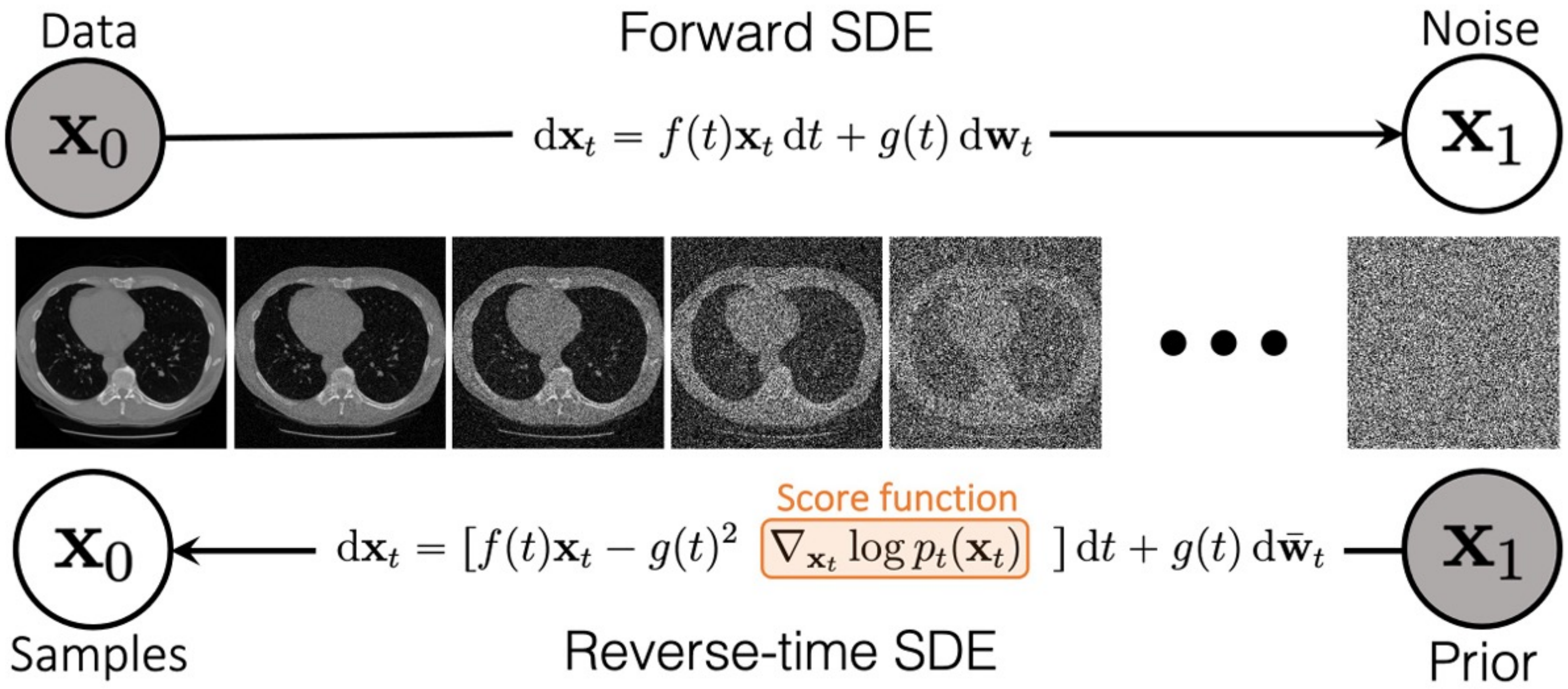
Diffusion models for medical imaging



- We can use a diffusion model to learn the prior distribution of images

Song et al, Solving Inverse Problems in Medical Imaging with Score-Based Generative Models, ICLR (2022)

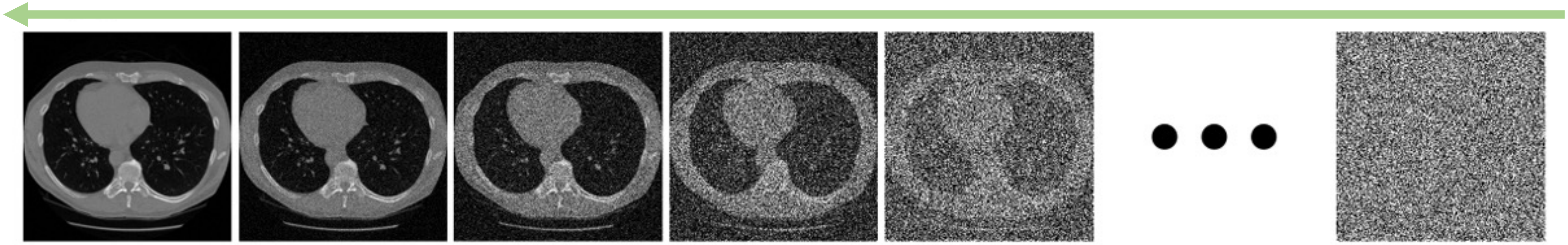
Diffusion models for medical imaging



- We can use a diffusion model to learn the prior distribution of images
- Q: how could we use this model to solve the CT inverse problem?

Song et al, Solving Inverse Problems in Medical Imaging with Score-Based Generative Models, ICLR (2022)

Diffusion models for medical imaging



$x_0 \sim p_0(x)$

$x_T \sim p_T(x)$

```
def sample_diffusion_model(theta):

    # sample prior
    x = torch.randn(128,128)

    # solve neural ODE in reverse
    for t in range(T,0,-1):
        x = x - dt*(f(x,t)-0.5*g(t)**2*NN(x,t,theta))
```

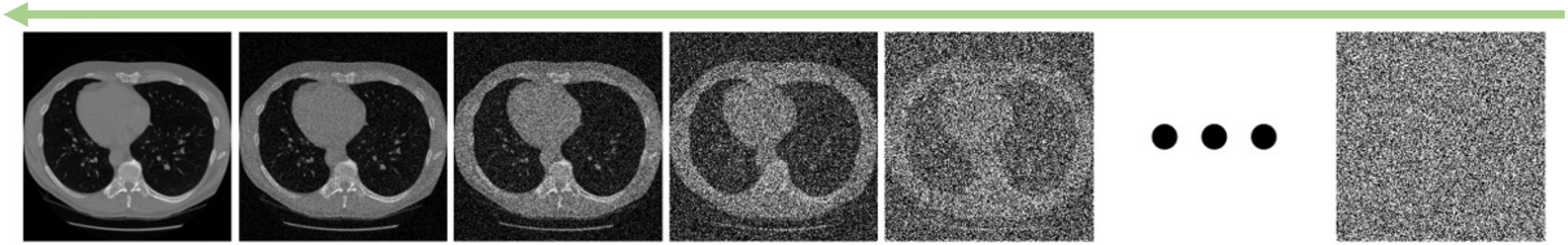
To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)
2. Solve the neural ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \boldsymbol{s}(\boldsymbol{x}, t; \boldsymbol{\theta})$$

Song et al, Solving Inverse Problems in Medical Imaging with Score-Based Generative Models, ICLR (2022)

Diffusion models for medical imaging



$\boldsymbol{x}_0 \sim p_0(x)$

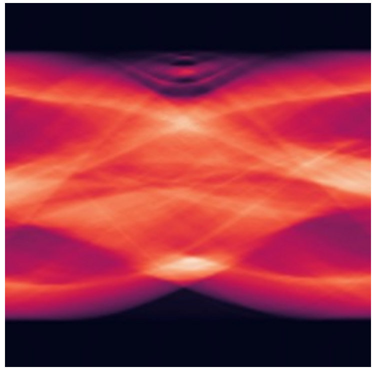
$\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$

To generate an image:

1. Sample $\boldsymbol{x}_T \sim p_T(\boldsymbol{x})$ (usually a Gaussian)
2. Solve the neural ODE (or reverse SDE) backwards from $t = T$ to $t = 0$:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \boldsymbol{s}(\boldsymbol{x}, t; \boldsymbol{\theta})$$

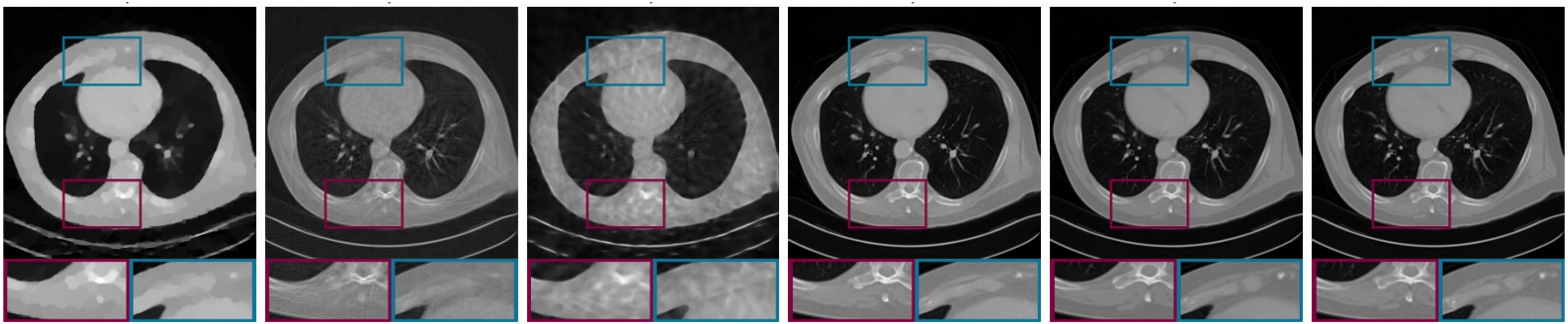
```
def sample_diffusion_model(theta):

    # sample prior
    x = torch.randn(128,128)

    # solve neural ODE in reverse
    for t in range(T,0,-1):
        x = x - dt*(f(x,t)-0.5*g(t)**2*NN(x,t,theta))

def sample_diffusion_model(theta, y_obs):

    # sample prior
    x = torch.randn(128,128)

    # solve neural ODE in reverse
    for t in range(T,0,-1):

        # "nudge" x so that it satisfies F(x)=y_obs
        # using a proximal optimisation step
        x = proximal_update(x, F(x)-y_obs)

        x = x - dt*(f(x,t)-0.5*g(t)**2*NN(x,t,theta))
```



$\boldsymbol{y}_{obs}$

Song et al, Solving Inverse Problems in Medical Imaging with Score-Based Generative Models, ICLR (2022)

Diffusion models for medical imaging



(a) FISTA-TV (b) cGAN (c) Neumann (d) SIN-4c-PRN (e) Ours (f) Ground truth

Song et al, Solving Inverse Problems in Medical Imaging with Score-Based Generative Models, ICLR (2022)

Lecture summary

- A neural differential equation uses neural networks to represent **learnable parts** of the equation
- A discretised NDE solver can be thought of as neural network architecture with **interpretable dynamics**
- State of the art ML models, e.g. diffusion models, solve NDEs