50.039 Theory and Practice of Deep Learning

W12-S2 About Diffusion Models

Matthieu De Mari

About this week (Week 12, an informative lecture about diffusion models)

1. What is a **Stochastic Differential Equation**?
2. What are **typical uses** of SDEs?
3. Can I **noise images progressively** using SDEs?
4. Can you train a denoising autoencoder model that removes SDE noise from images?
5. What is the **diffusion process** and how to use it to train **diffusion models**?
6. What are the architectures of DALL-E and Midjourney models?

Diffusion Models: a definition

Definition (Diffusion Models):

Diffusion models are a class of **stochastic mathematical models** that have typically been used to describe the random motion of particles, often referred to as “Brownian motion.“

They are widely used in various fields, such as physics, chemistry, biology, and finance, for instance to model the stochastic behaviour of particles or other quantities that exhibit random fluctuations over time.

The fundamental concept behind diffusion models is the idea of **Stochastic Differential Equations** (**SDEs**).

Stochastic Differential Equations (SDEs)

Definition (Stochastic Differential Equations):

Stochastic Differential Equations (SDEs) are **differential equations where one or more of the terms include a random component, usually in the form of a noise term.**

The randomness is often modelled using a Wiener process or Brownian motion, which is a continuous-time stochastic process characterized by a mean, a variance, and independent increments.

Diffusion models as SDEs

Mathematically, a **diffusion model can be represented as an SDE**:

$$dx(t) = a(x(t), t)dt + b(x(t), t)dW(t) \quad and \quad x(t = 0) = x_0$$

Where,

- $x(t)$ represents the **state of the system at time *t***,
- $a(x(t), t)$ is the **deterministic drift term**,
- $b(x(t), t)$ is the **stochastic diffusion term**,
- and $W(t)$ is the **Wiener process**.

The drift term governs the deterministic part of the system's dynamics, while the diffusion term introduces randomness into the system's evolution.

A first SDE: the Brownian motion

As a first example, let us consider one of the simplest SDEs:

$$dx(t) = dW(t) \quad and \quad x(0) = 0$$

This SDE typically defines a random walk starting from 0 and the noise process is simply zero mean and variance $\sqrt{timestep}$.

Implemented as shown in Notebook 1 and on the right.

```python
# Parameters
num_simulations = 5
total_time = 10.0
num_steps = 1000
dt = total_time/num_steps
```

```python
# Initialize the state variable as 0, at t = 0
x = np.zeros((num_simulations, num_steps))
x[:, 0] = 0.0

# Simulate Brownian motion
for t in range(1, num_steps):
    # Definition random motion using a Normal distribution with
    # Zero mean and variance sqrt(dt)
    dW = np.random.normal(0, np.sqrt(dt), size = num_simulations)
    x[:, t] = x[:, t-1] + dW
```

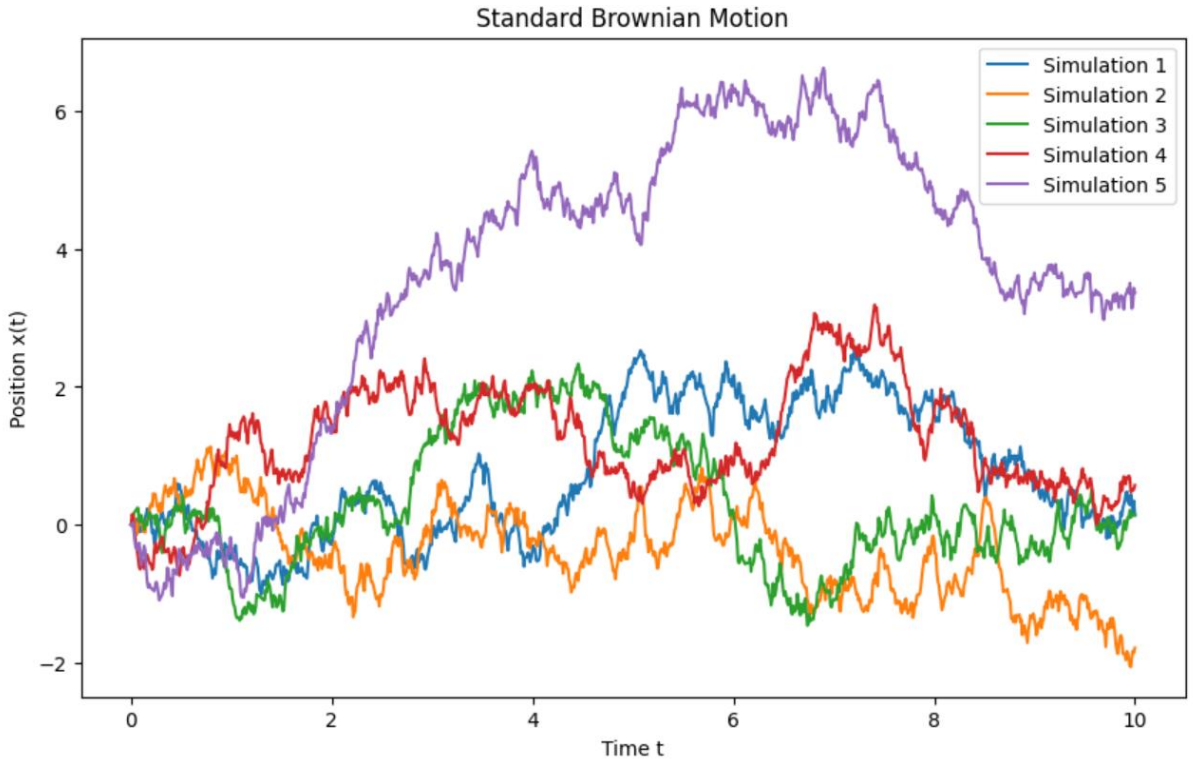
A first SDE: the Brownian motion

As a first example, let us consider one of the simplest SDEs:

$$dx(t) = dW(t) \quad and \quad x(0) = 0$$

This SDE typically defines a random walk starting from 0 and the noise process is simply zero mean and variance $\sqrt{timestep}$.

Implemented as shown in Notebook 1 and on the right.



A second SDE with drift

As a second example, let us consider an SDE wit drift:

$$dx(t) = 0.1x(t)dt + dW(t) \quad and \quad x(0) = 10$$

The drift function here is $0.1x(t)$, adding deterministically ~10% to the current value of $x(t)$ for each unit of time t.

A second SDE with drift

- This second SDE is typically implemented as shown.
- We also show as xr, the realisation of the SDE without any Wiener motion.

```python
# Parameters
num_simulations = 5
total_time = 10.0
num_steps = 1000
dt = total_time / num_steps
```

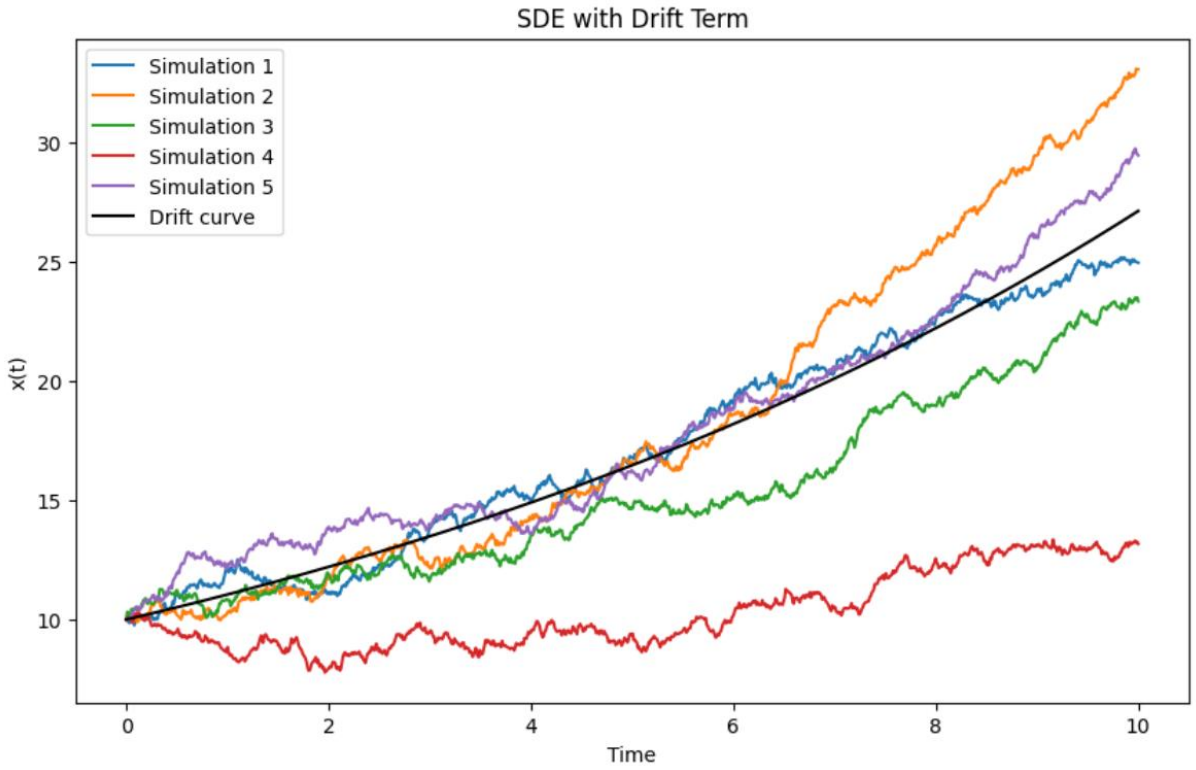
```python
# Define the drift term
def drift(x, t):
    return 0.1*x
```

```python
# Initialize the state variable
x = np.zeros((num_simulations, num_steps))
x[:, 0] = 10.0  # Initial condition

# Simulate the SDE
for t in range(1, num_steps):
    dW = np.random.normal(0, np.sqrt(dt), size = num_simulations)
    dx = drift(x[:, t-1], t*dt)*dt + dW
    x[:, t] = x[:, t-1] + dx
```

```python
xr = np.zeros((1, num_steps)).reshape(num_steps)
xr[0] = 10.0
for t in range(1, num_steps):
    dx = drift(xr[t-1], t*dt)*dt
    xr[t] = xr[t-1] + dx
```

A second SDE with drift

- This second SDE is typically implemented as shown.
- We also show as xr, the realisation of the SDE without any Wiener motion.
- Typically a 10% increase per unit of time (as is often described the US stock market), with noisy variations.
- Is the stock market an SDE of some sort then?



Typical examples of SDEs

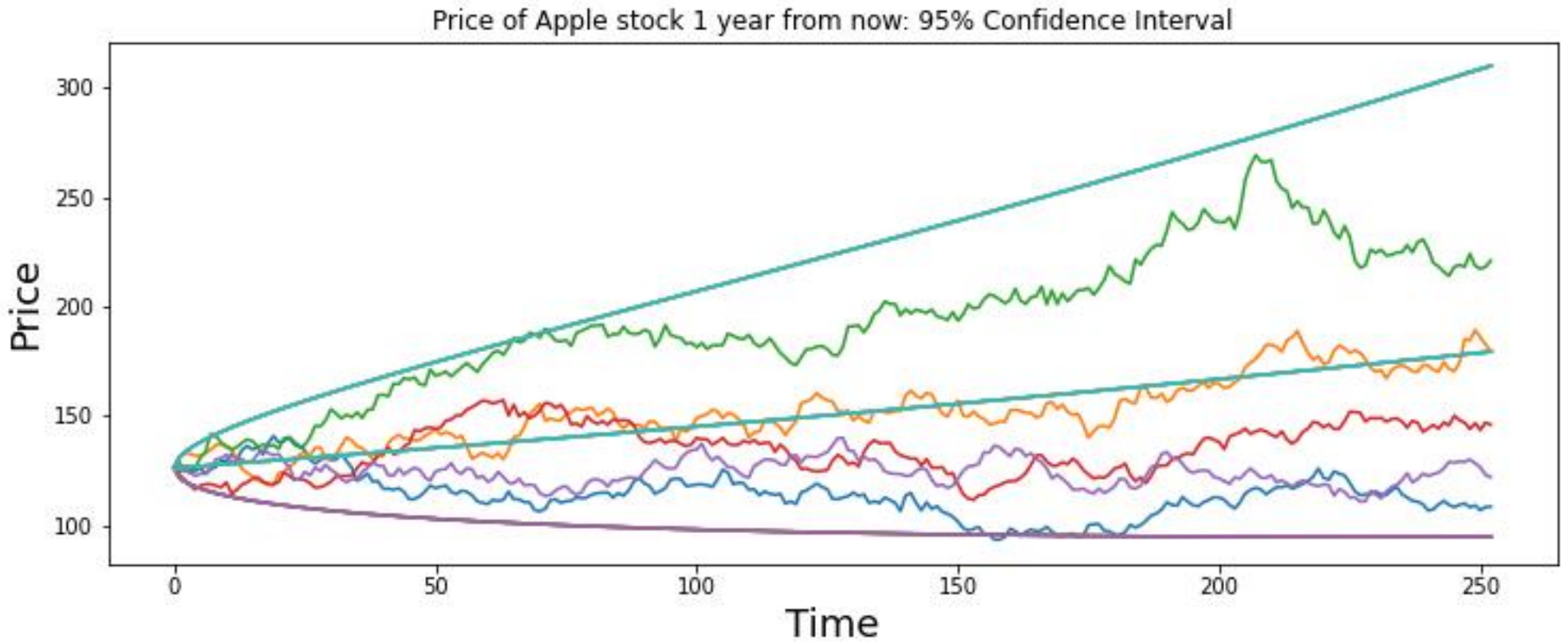
In fact, stochastic differential equations (SDEs) have been widely used, in finance, to model the dynamics of various financial instruments and quantities. For instance, the Geometric Brownian motion is a popular model for simulating the price of assets such as stocks. It is given by:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t)$$

Where $S(t)$ is the asset price, μ is the drift of the asset (i.e. expected return of the asset over time), σ is the volatility, and $W(t)$ is a Wiener process modelling the unpredictable behaviour of the asset.

Typical examples of SDEs

In fact, stochastic differential equations (SDEs) have been widely used, in finance, to model the dynamics of various financial instruments and



A first task, denoising images

- **Task:** take images from MNIST and noise them. The model should try its best to denoise them an get back to the original image.
- **Dataset:** MNIST images, noised (inputs) and original (output).
- **Model:** an encoder-decoder model of some sort, similar to an autoencoder (almost).
- **Loss and training procedure:** As in the case of autoencoders, a simple MSE loss and a basic training procedure with Adam optimizer will do.

A first task, denoising images

- For simplicity, we consider an encoder-decoder model, similar to our autoencoder in W10-NB1, running on linear layers.

```python
class Denoiser(nn.Module):
    def __init__(self):
        super(Denoiser, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, x):
        return self.decoder(self.encoder(x))
```

```python
# Define the encoder
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 400),
            nn.ReLU(),
            nn.Linear(400, 400),
            nn.ReLU()
        )

    def forward(self, x):
        return self.encoder(x)
```

```python
# Define the decoder
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.decoder = nn.Sequential(
            nn.Linear(400, 400),
            nn.ReLU(),
            nn.Linear(400, 28*28),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.decoder(x)
```

A first task, denoising images

- For simplicity, we consider an encoder-decoder model, similar to our autoencoder in W10-NB1, running on linear layers.
- And a basic training procedure…

```python
class Denoiser(nn.Module):
    def __init__(self):
        super(Denoiser, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, x):
        return self.decoder(self.encoder(x))
```

```python
# Define the encoder
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 400),
            nn.ReLU(),
            nn.Linear(400, 400),
            nn.ReLU()
        )

    def forward(self, x):
        return self.encoder(x)
```

```python
# Define the decoder
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.decoder = nn.Sequential(
            nn.Linear(400, 400),
            nn.ReLU(),
            nn.Linear(400, 28*28),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.decoder(x)
```

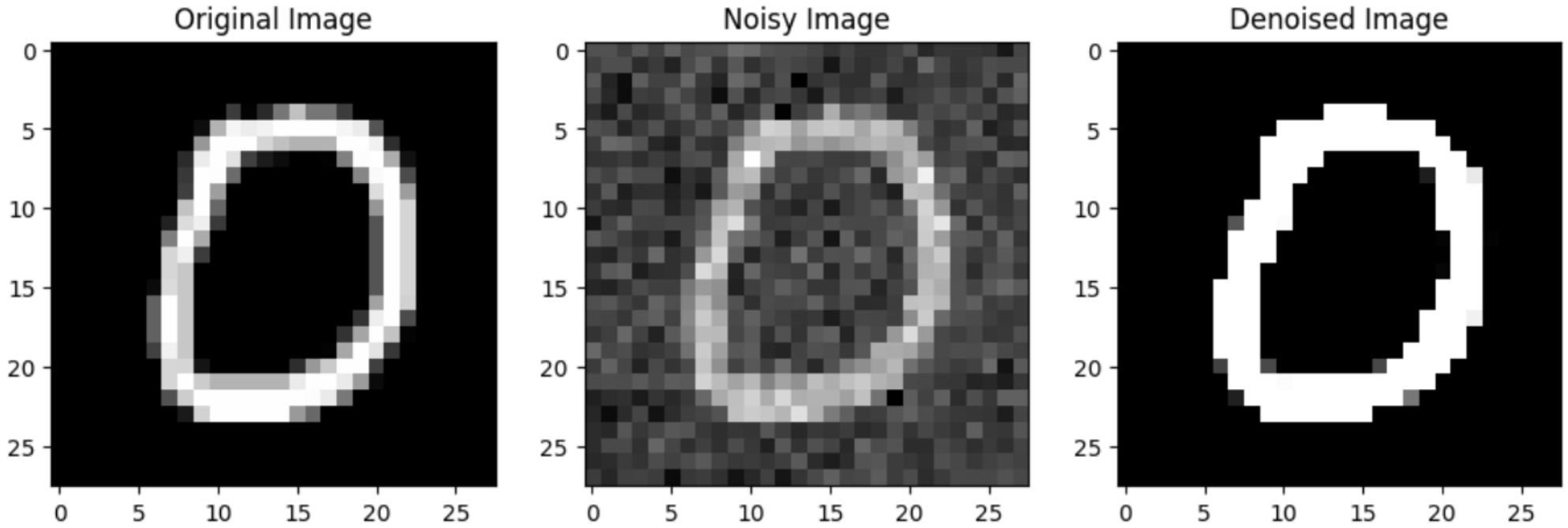
```python
# Hyperparameters
epochs = 10
learning_rate = 1e-3
noise_factor = 0.5
```

```python
# Initialize the model, loss function, and optimizer
model = Denoiser()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

```python
# Training loop
for epoch in range(epochs):
    for batch_idx, (data, _) in enumerate(train_loader):
        # Noise the data with one or many steps of SDEs
        # (Here a single step Brownian Motion)
        data = data.view(data.size(0), -1)
        noisy_data = data + noise_factor*torch.randn_like(data)

        # Forward pass
        optimizer.zero_grad()
        outputs = model(noisy_data)

        # Backprop
        loss = criterion(outputs, data)
        loss.backward()
        optimizer.step()

        # Display
        if batch_idx % 100 == 0:
            print(f'Epoch: {epoch+1}/{epochs}, Batch: {batch_idx}/{len(train_loader)}, Loss: {loss.item()}')
```

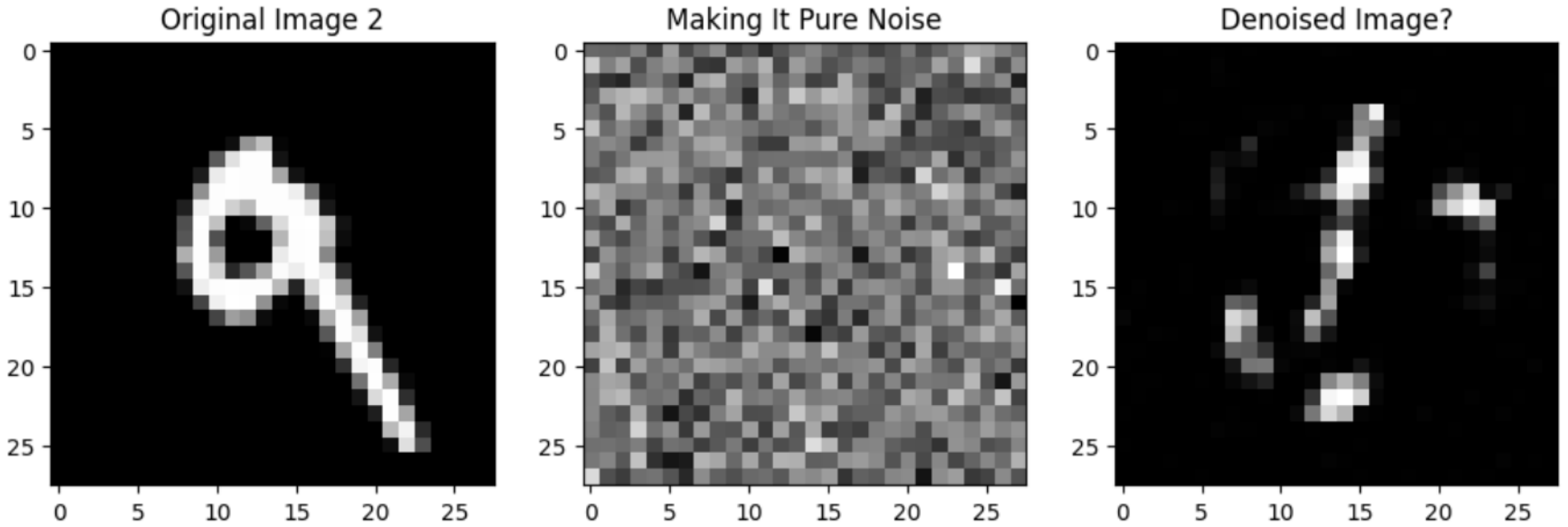
A first task, denoising images

- The model trains and seems able to denoise lightly noised images.



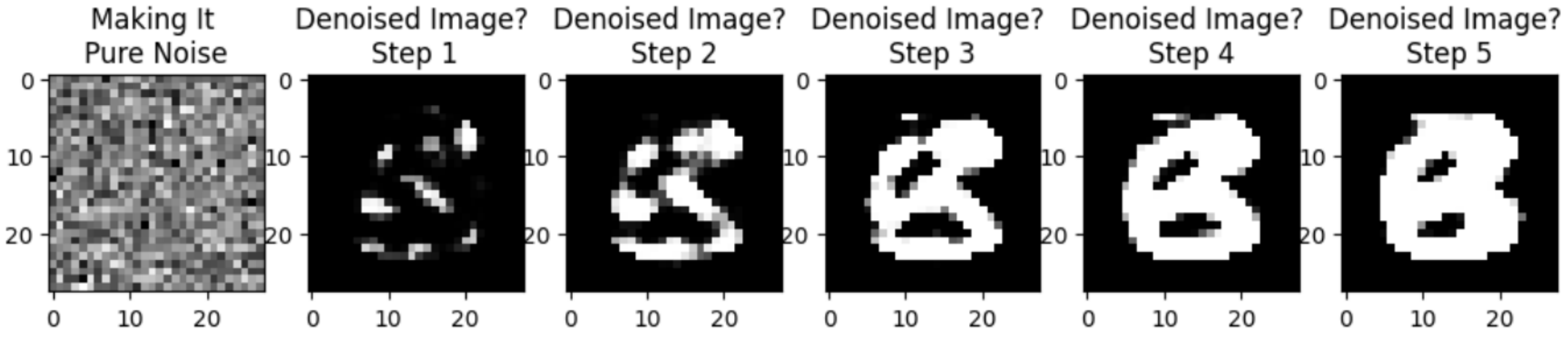
A first task, denoising images

- The model trains and seems able to denoise lightly noised images.
- **Could it denoise a pure noise image, though?** Not really.



A first task, denoising images

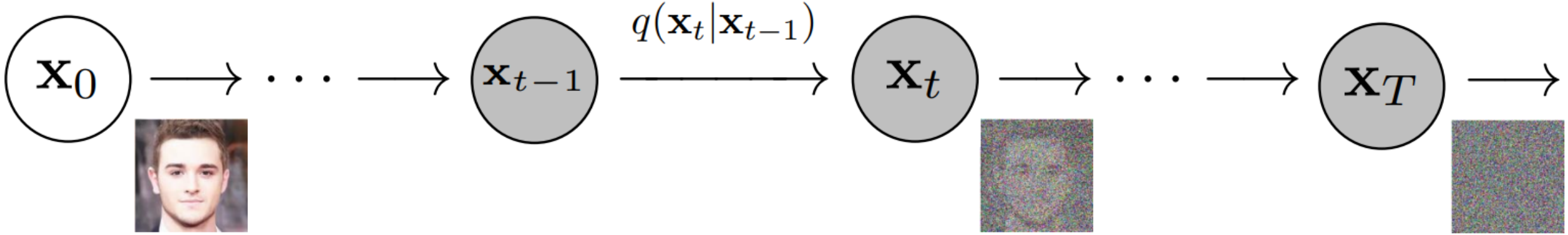
- The model trains and seems able to denoise lightly noised images.
- **Could it denoise a pure noise image, though?** Not really.
- **What if we used the model many (e.g. 5) times in a row on our full noise image?** Eventually would obtain something B&W in the likes of MNIST but somehow non-sensical in terms of shapes…?!



A (not so?) crazy idea

Idea: what if instead of noising the image in one step, and denoising it in one call of the model, we try iterative noising and denoising?

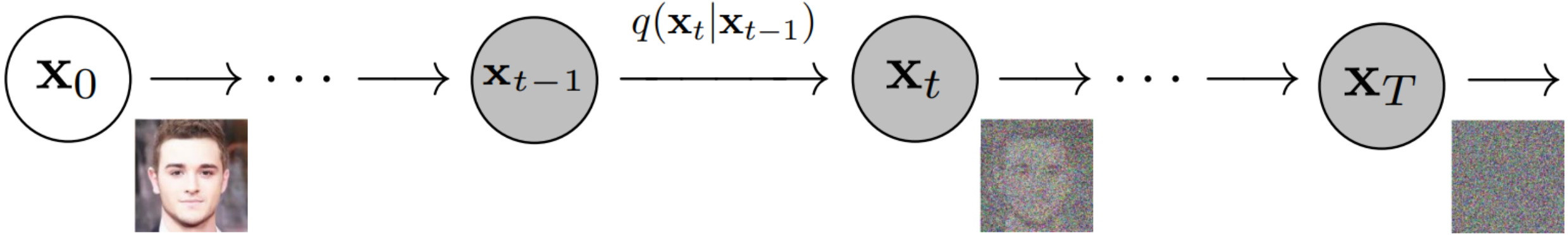
(Somehow similar to the idea of modifying the image iteratively during the iterated versions of our attacks in W5!)



A (not so?) crazy idea

Idea: what if instead of noising the image in one step, and denoising it in one call of the model, we try iterative noising and denoising?

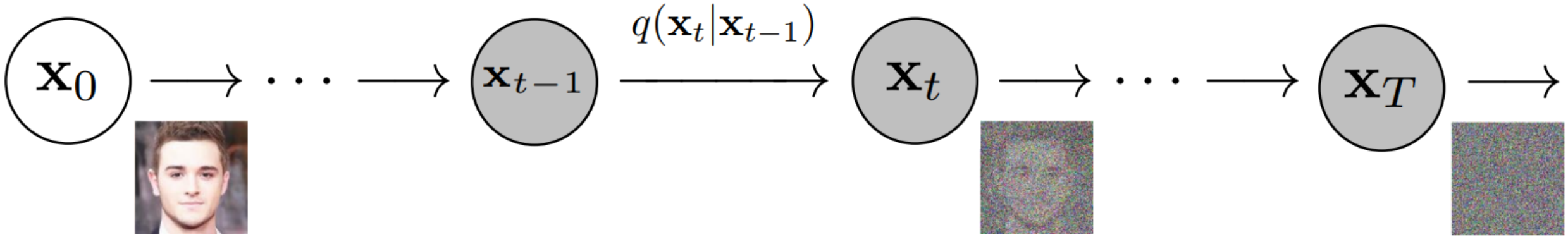
- **Step 1:** Add progressive noising T times in a row, until the image becomes pure noise (or so).
- The noising process follows some sort of an SDE equation describing the progressive noising of the original image. (our Forward here!)



A (not so?) crazy idea

Idea: what if instead of noising the image in one step, and denoising it in one call of the model, we try iterative noising and denoising?

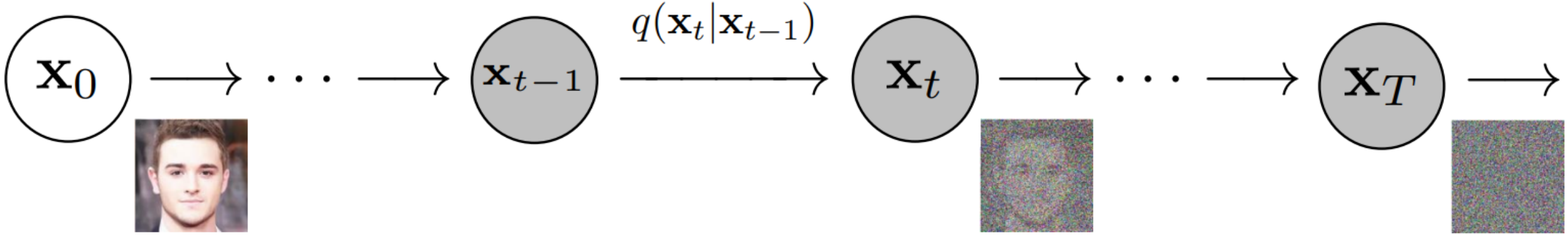
- **Step 2:** Train a single model to try and cancel each round of noising.
- It receives image x_t and tries to produce x_{t-1}, do for each t. (our Backward here!)



A (not so?) crazy idea

Idea: what if instead of noising the image in one step, and denoising it in one call of the model, we try iterative noising and denoising?

- **Training procedure:** Use original images from MNIST and noise them T times in a row, train model on denoising each step and pair of images (x_t, x_{t-1}).
- Use MSE loss and training procedure similar as before.



A basic Diffusion Model setup

- Roughly same encoder-decoder model as before.
- Adds an info about current time step t for noising as input parameter, along with x_t.

```
class DiffusionModel(nn.Module):
    def __init__(self):
        super(DiffusionModel, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(28 * 28 + 1, 400),
            nn.ReLU(),
            nn.Linear(400, 400),
            nn.ReLU(),
            nn.Linear(400, 28 * 28)
        )

    def forward(self, x, t):
        x = torch.cat([x, t.expand(x.size(0), 1)], dim = 1)
        return self.network(x)
```

A basic Diffusion Model setup

- Roughly same encoder-decoder model as before.
- Adds an info about current time step t for noising as input parameter, along with x_t.
- Same training parameters as before.
- Five steps of noising (T = 5).
- Training procedure somewhat similar to what we had before…

```python
class DiffusionModel(nn.Module):
    def __init__(self):
        super(DiffusionModel, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(28 * 28 + 1, 400),
            nn.ReLU(),
            nn.Linear(400, 400),
            nn.ReLU(),
            nn.Linear(400, 28 * 28)
        )

    def forward(self, x, t):
        x = torch.cat([x, t.expand(x.size(0), 1)], dim = 1)
        return self.network(x)
```

```python
# Hyperparameters
epochs = 10
batch_size = 64
learning_rate = 1e-3
num_timesteps = 5
noise_schedule = torch.linspace(0, 1, num_timesteps)
```

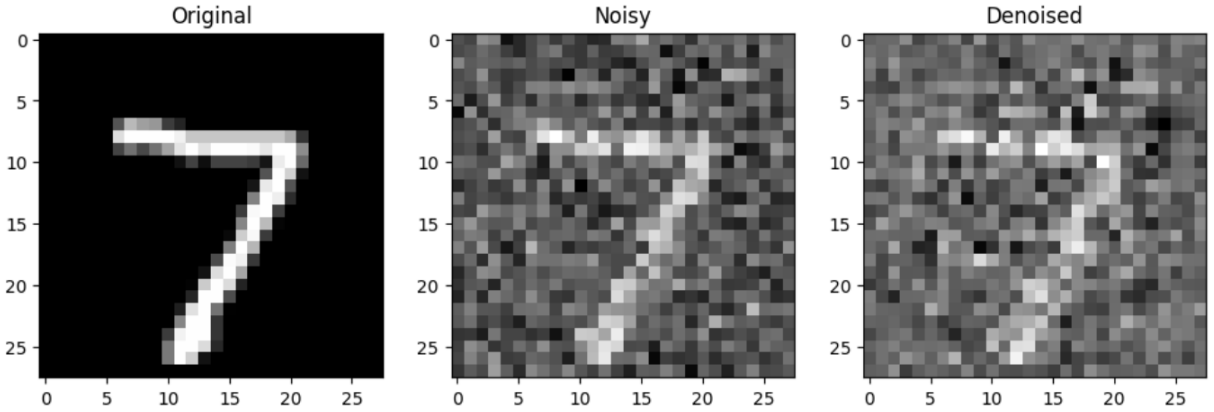
```python
# Initialize the model, loss function, and optimizer
model = DiffusionModel()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

```python
# Training loop
# (Very long, use at your own discretion!)
for epoch in range(epochs):
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.view(data.size(0), -1)
        noisy_data = data.clone()

        # Apply multiple noising steps
        for t in range(num_timesteps-1):
            noisy_data += torch.randn_like(noisy_data)*torch.sqrt(noise_schedule[t+1]-noise_schedule[t])

        optimizer.zero_grad()
        loss = 0

        # Apply multiple denoising steps
        for t in reversed(range(num_timesteps-1)):
            denoised_data = noisy_data-torch.randn_like(noisy_data)*torch.sqrt(noise_schedule[t+1]-noise_schedule[t])
            target_score = (data-denoised_data)/(noise_schedule[t+1]-noise_schedule[t])
            score_prediction = model(denoised_data, noise_schedule[t].view(-1, 1))
            loss += criterion(score_prediction, target_score)

        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            print(f'Epoch: {epoch+1}/{epochs}, Batch: {batch_idx}/{len(train_loader)}, Loss: {loss.item()}')
```

In practice, however…

Need more iterations and many more steps of noising and denoising (heavy to train), in the hundreds/thousands or so.

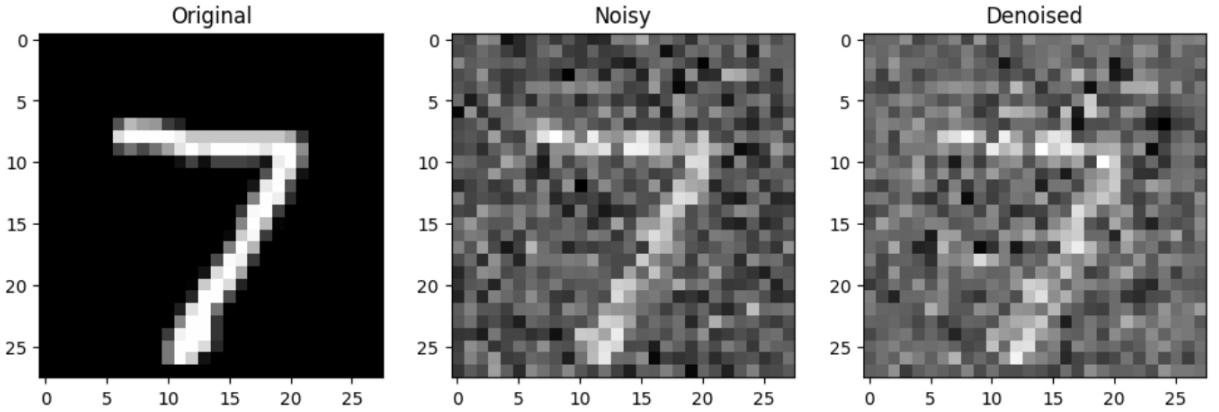
- Our Notebook 3 runs on only 5 steps of noising/denoising.
- But you get the idea…

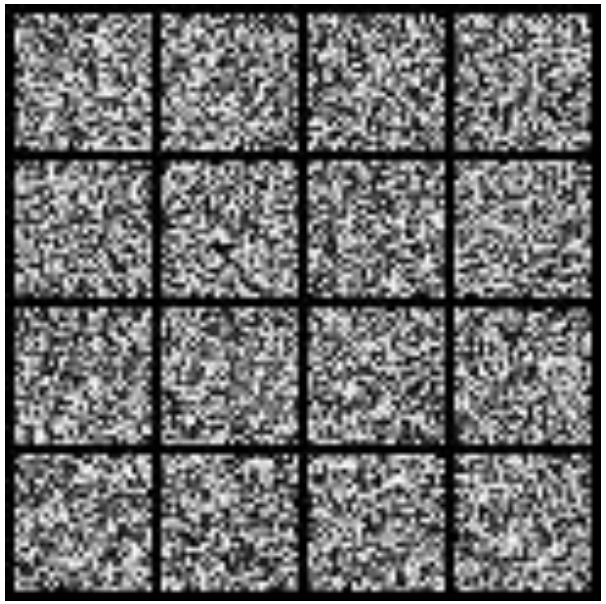


In practice, however…

With more steps and CNN layers, get this kind of behaviour:

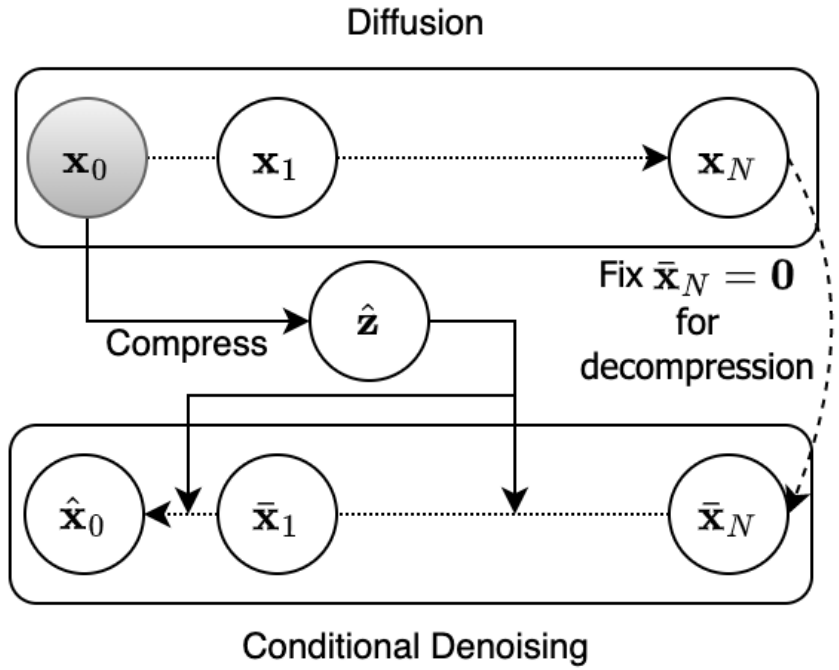
- Being able to train a model to denoise and eventually obtain a convincing MNIST picture from something that was originally pure noise! (as shown in GIF).

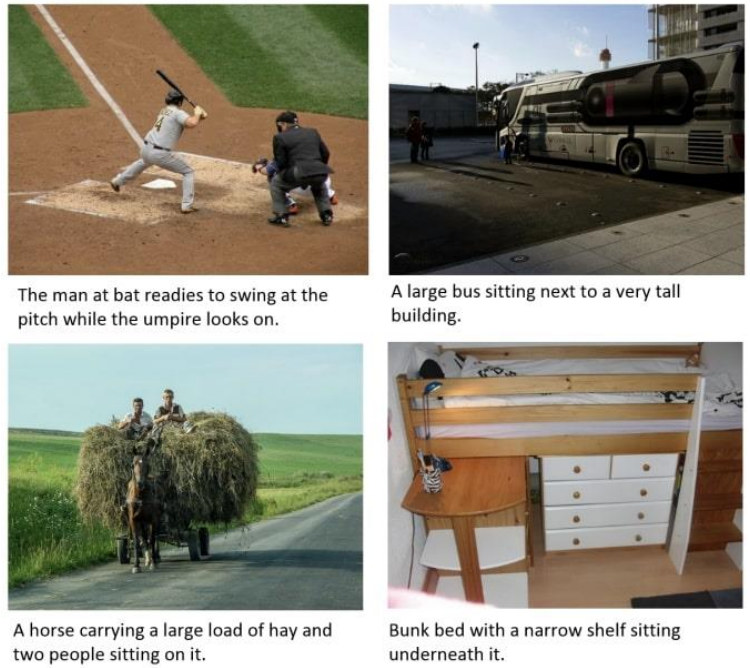




Making it conditional

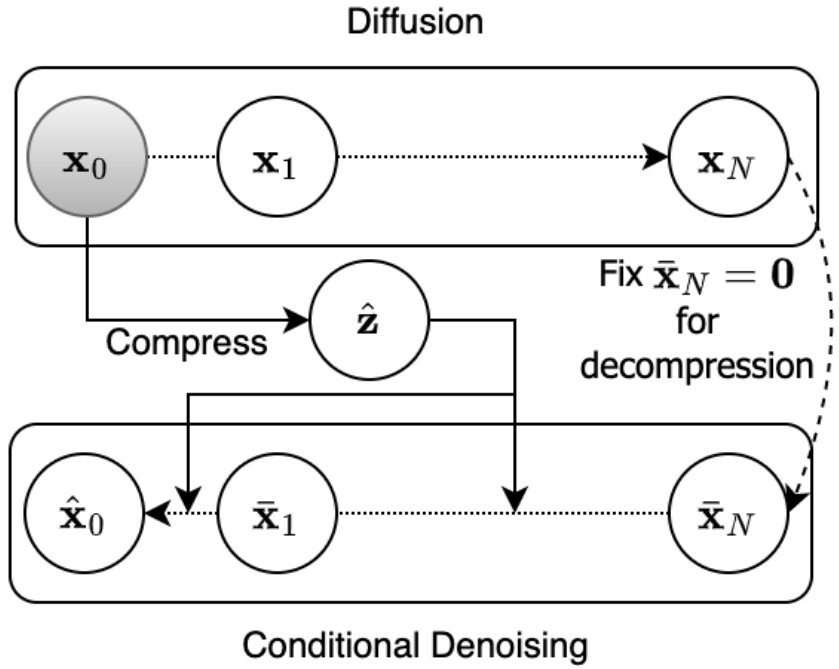
- If you train with a dataset of captioned images, the extra label could be used during the denoising part, to help the denoising model figure out what was the original image about.
- Encode these captions and use them during training of the denoising model in place of t.
- Helps to direct the denoising in right direction!

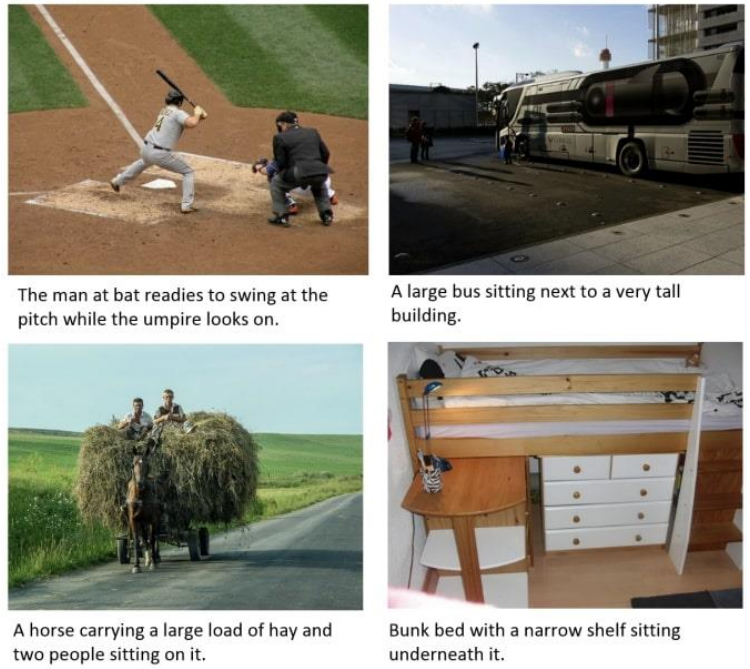




Making it conditional

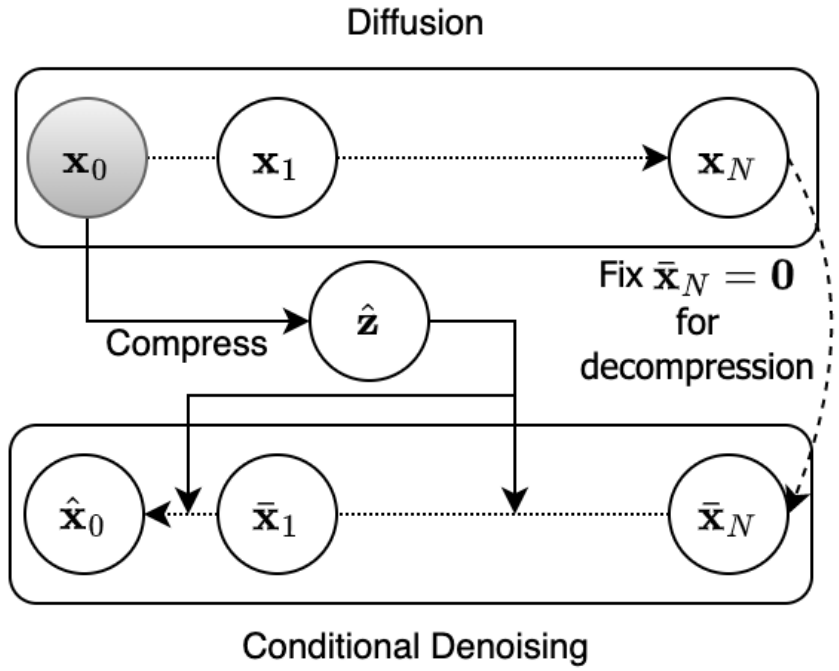
- During testing, use pure noise as x_N and ask the user for a caption of their choice that will be compressed as $\hat{z}$.
- Use your denoiser many times in a row and obtain an image matching the caption $\hat{z}$ starting from pure noise?!

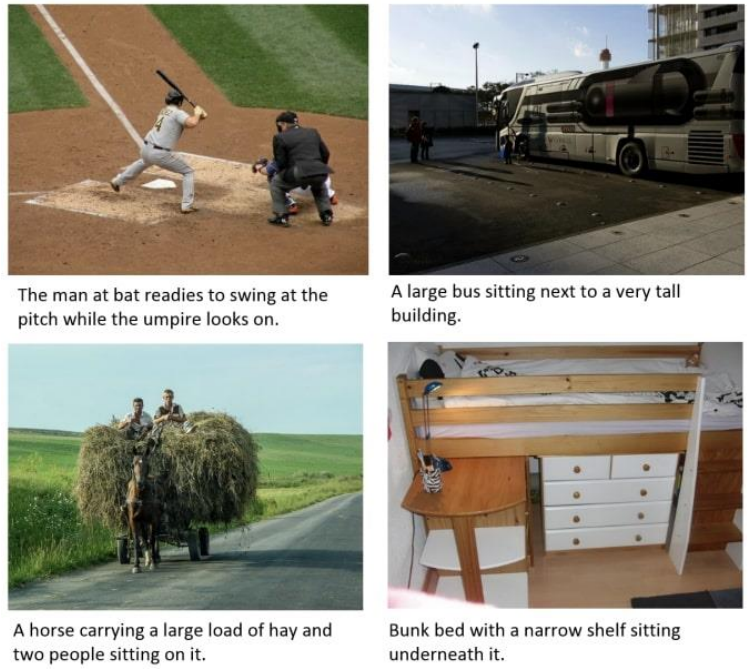




Making it conditional

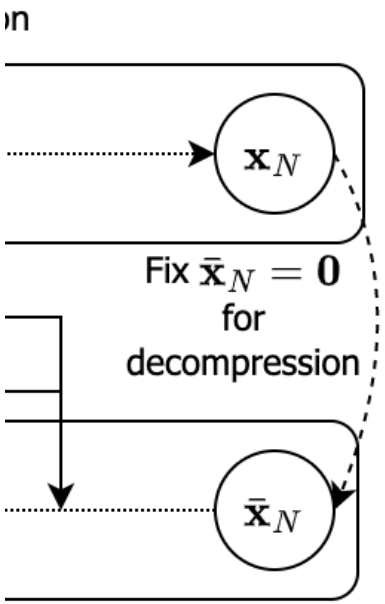
- During testing, use pure noise as x_N and ask the user for a caption of their choice that will be compressed as $\hat{z}$.
- Use your denoiser many times in a row and obtain an image matching the caption $\hat{z}$ starting from pure noise?!
- Yes, and better than most GANs these days!





Makin

- During t
  x_N and
  of their
  compres
- Use you
  a row ar
  matchin
  from pu
- Yes, and
  these da



enoising


large bus sitting next to a very tall
ilding.


nk bed with a narrow shelf sitting
underneath it.

two people sitting on it.

Diffusion: a good entry point to continue your study of generative models

- **Stable diffusion:** Stable diffusion is a term used to describe a certain property of stochastic differential equations (SDEs) or diffusion processes.
- In the context of diffusion models, it refers to the stability of the diffusion process, ensuring that the SDE does not explode or diverge over time.
- Stable diffusion processes are crucial in designing diffusion models because they ensure that the generated samples remain within a reasonable range and do not become arbitrarily large or small.

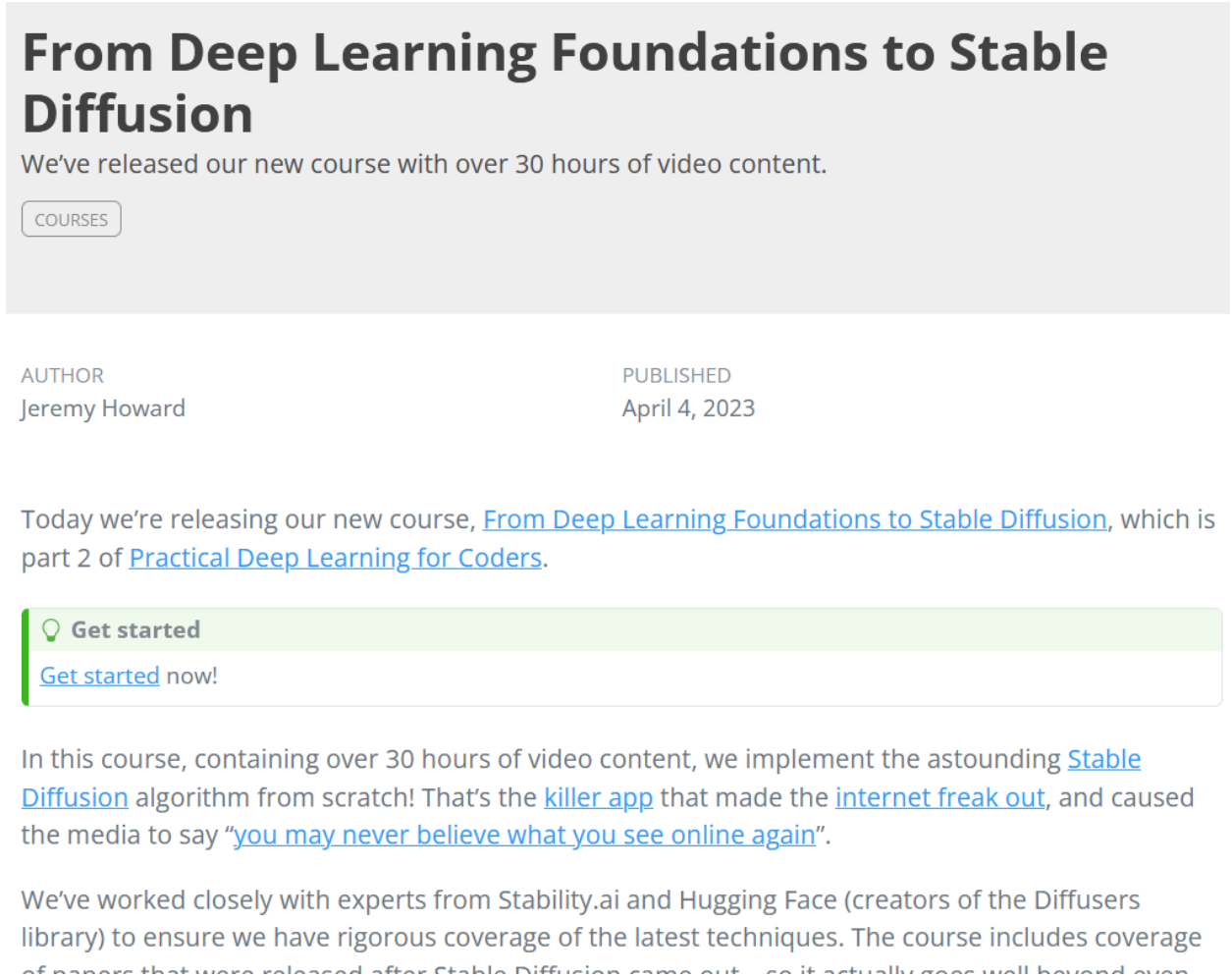
Diffusion: a good entry point to continue your study of generative models

- Diffusion models is a promising direction, but many more steps needed to get to a generative model like DALL-E or MidJourney!
- DALL-E and MidJourney models are not so to speak based on diffusion models, but have leveraged intuition from said models.
- They would be better described, for now, as generative models based on transformer architectures.
- Something for later in **Term 7 Computer Vision** course?

Diffusion: a good entry point to continue your study of generative models

- If you cannot wait for **Term 7 Computer Vision** course…
- I guess, this course from **Jeremy Howard (Fast.ai),** one of the most important AI teachers out there, is probably very nice to take…

https://www.fast.ai/posts/part2-2023.html



From Deep Learning Foundations to Stable Diffusion

We've released our new course with over 30 hours of video content.

COURSES

AUTHOR
Jeremy Howard

PUBLISHED
April 4, 2023

Today we're releasing our new course, From Deep Learning Foundations to Stable Diffusion, which is part 2 of Practical Deep Learning for Coders.

Get started

Get started now!

In this course, containing over 30 hours of video content, we implement the astounding Stable Diffusion algorithm from scratch! That's the killer app that made the internet freak out, and caused the media to say "you may never believe what you see online again".

We've worked closely with experts from Stability.ai and Hugging Face (creators of the Diffusers library) to ensure we have rigorous coverage of the latest techniques. The course includes coverage

Conclusion

1. What is a **Stochastic Differential Equation**?
2. What are **typical uses** of SDEs?
3. Can I **noise images progressively** using SDEs?
4. Can you train a denoising autoencoder model that removes SDE noise from images?
5. What is the **diffusion process** and how to use it to train **diffusion models**?
6. What are the architectures of DALL-E and Midjourney models?

Learn more about these topics

Out of class, for those of you who are curious

- The one paper, usually cited for Diffusion models
  [Ho2020] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems", 2020.
  https://arxiv.org/abs/2006.11239

- [Dhariwal2021] P. Dhariwal, and A. Nichol, "Diffusion models beat GANs on ImageNet.", 2021.
  https://arxiv.org/abs/2105.05233