Title: Tutorial - Scientific Machine Learning, PHYS 777

Speakers: Sehmimul Hoque

Collection/Series: Scientific Machine Learning (Elective), PHYS 777, February 23 - March 27, 2026

Subject: Condensed Matter, Cosmology, Other, Particle Physics

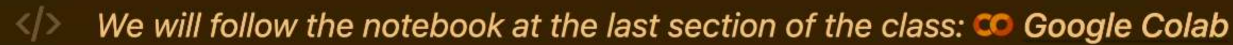
Date: March 27, 2026 - 6:00 PM

URL: https://pirsa.org/26030093

Content

Here we will cover:

1. Autoencoders and Variational Autoencoders
2. Diffusion Models
   a. Noise Conditional Score Network
   b. Score Based Diffusion models
3. Conditional Diffusion models



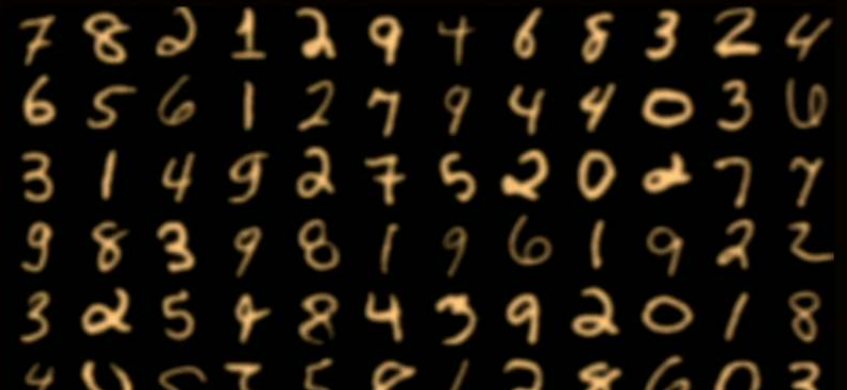
Generative Models essentials

- Generative models are used to generate new data.
- Consider a dataset $D = (X, y)$

Diffusion Models | Notion × Diffusion Models × Stable Diffusion Online × Diffusion-Model-Demo.i × Stable Diffusion Online × Calendar - Sehmimul Ho × 0905.2199v2.pdf × Fields Institute Personal ×
brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3
New Chrome available

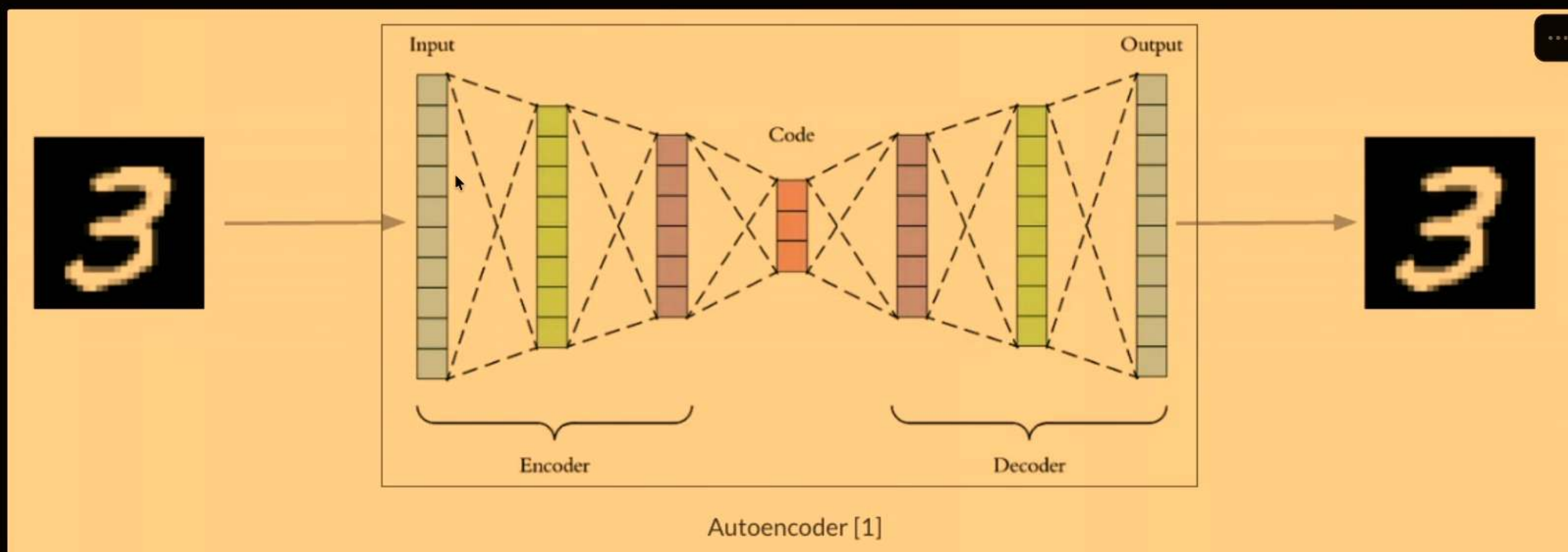
Generative Models essentials

- Generative models are used to generate new data.
- Consider a dataset $D = (X, y)$
- In general classification involves learning a neural network $f_\theta(x) \rightarrow y$.
- Generative Models can learn some probability distribution and can generate data from that learned distribution by sampling.



Autoencoders and Variational Autoencoders

- Autoencoders form encoder decoder models where you encode data in a smaller space (called latent space) and decode it to the data space.



Autoencoder for MNIST

- Now let's look at a two dimensional latent space of the autoencoder

Input

Output

Code

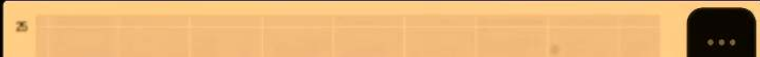
Encoder

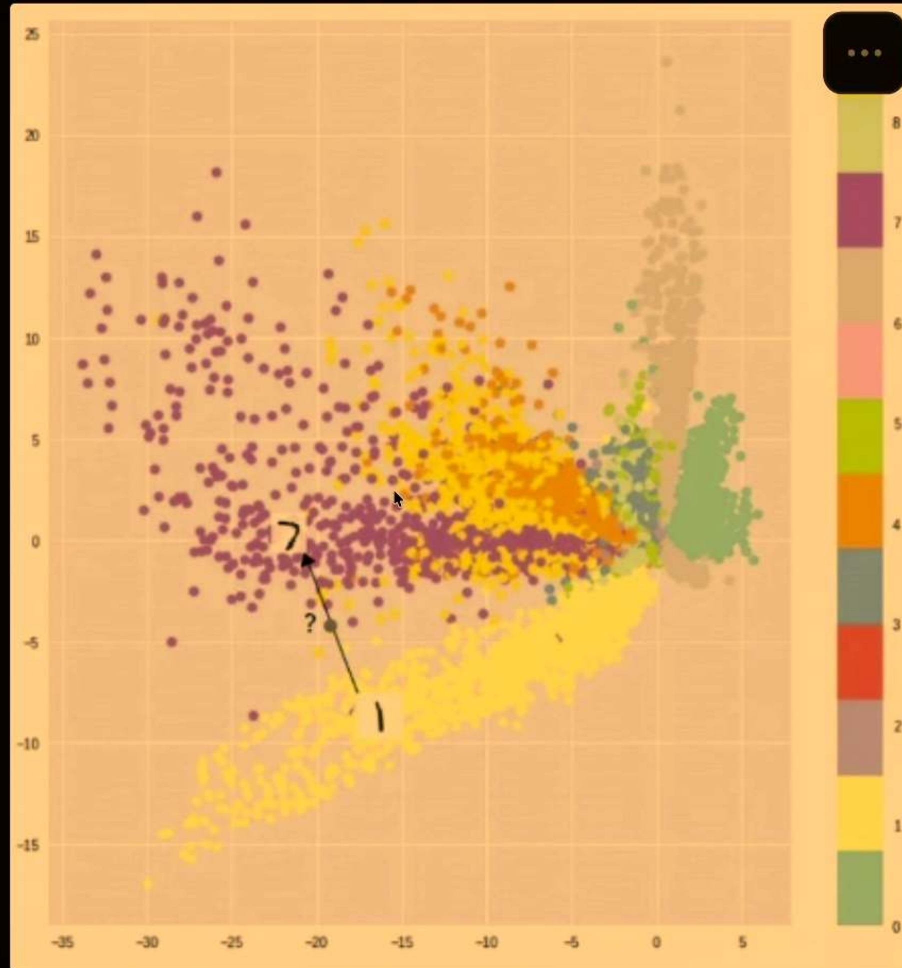
Decoder

Autoencoder [1]

coder for MNIST

- Now let's look at a two dimensional latent space of the autoencoder





7
?
1

- Downsides of VAE are:

Multi-Stage Variational Auto-Encoders for Coarse-to-Fine Image Generation

Lei Cai * Hongyang Gao † Shuiwang Ji ‡

Abstract

Variational auto-encoder (VAE) is a powerful unsupervised learning framework for image generation. One drawback of VAE is that it generates blurry images due to its Gaussianity assumption and thus ℓ_2 loss. To allow the generation of high quality images by VAE, we increase the capacity of decoder network by employing residual blocks and skip connections, which also enable efficient optimization. To overcome the limitation of ℓ_2 loss, we propose to generate images in a multi-stage manner from coarse to fine. In the simplest case, the proposed multi-stage VAE divides the decoder into two components in which the second component generates refined images based on the course images generated by the first component. Since the second component is independent of the VAE model, it can employ other loss functions beyond the ℓ_2 loss and different model architectures. The proposed framework can be easily generalized to contain more than two components. Experiment results on the MNIST and CelebA datasets demonstrate that the proposed multi-stage VAE can generate sharper images as compared to those from the original VAE.

Disentangling Variational Autoencoders

Rafael Pastrana[a,*]

[a]School of Architecture, Princeton University, United States of America

Abstract

A variational autoencoder (VAE) is a probabilistic machine learning framework for posterior inference that projects an input set of high-dimensional data to a lower-dimensional, latent space. The latent space learned with a VAE offers exciting opportunities to develop new data-driven design processes in creative disciplines, in particular, to automate the generation of multiple novel designs that are aesthetically reminiscent of the input data but that were unseen during training. However, the learned latent space is typically disorganized and entangled: traversing the latent space along a single dimension does not result in changes to single visual attributes of the data. The lack of latent structure impedes designers from deliberately controlling the visual attributes of new designs generated from the latent space. This paper presents an experimental study that investigates latent space disentanglement. We implement three different VAE models from the literature and train them on a publicly available dataset of 60,000 images of hand-written digits. We perform a sensitivity analysis to find a small number of latent dimensions necessary to maximize a lower bound to the log marginal likelihood of the data. Furthermore, we investigate the trade-offs between the quality of the reconstruction of the decoded images and the level of disentanglement of the latent space. We are able to automatically align three latent dimensions with three interpretable visual properties of the digits: line weight, tilt and width. Our experiments suggest that i) increasing the contribution of the Kullback-Leibler divergence between the prior over the latents and the variational distribution to the evidence lower bound, and ii) conditioning input image class enhances the learning of a disentangled latent space with a VAE.

- VAE Images are generally blurry as we constrain latent space to be standard Gaussian.
- VAE latent space can often be entangled.

VAE downsides

- This concludes the section of encoder decoder model introduction.

Diffusion Models | Notion × Diffusion Models × Stable Diffusion Online × Diffusion-Model-Demo.i × Stable Diffusion Online × Calendar - Sehmimul Ho × 0905.2199v2.pdf × Fields Institute Personal ×

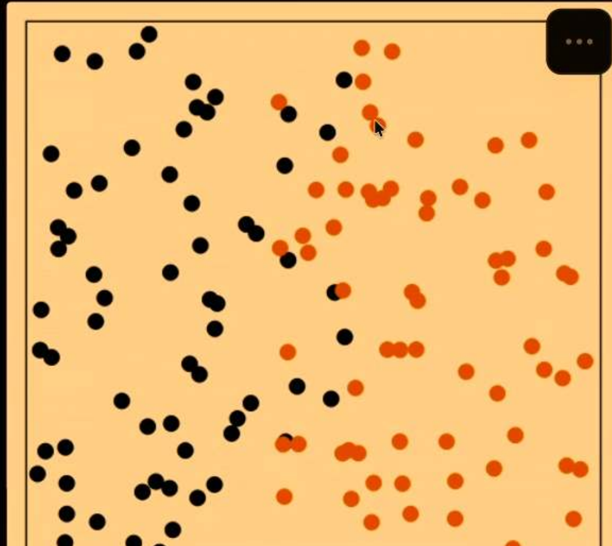
brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available

Diffusion Models

- First we talk about some of the motivations of diffusion model coming from physics.
- Diffusion is a phenomena of something spreading from an ordered distribution to some disordered distribution.

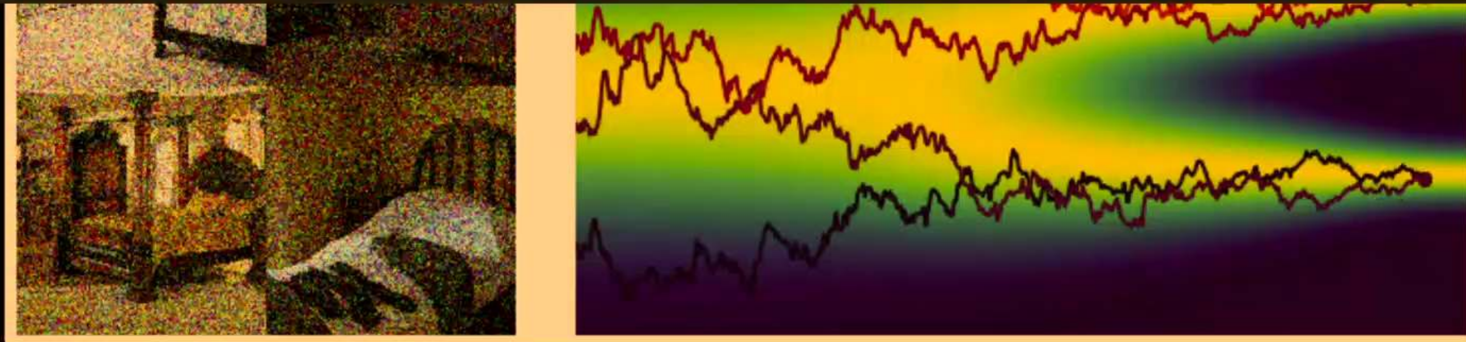
▼ GIF diffusion and diffusion model



Diffusion Models | Notion · Diffusion Models · Stable Diffusion Online · Diffusion-Model-Demo · Stable Diffusion Online · Calendar - Sehmimul Ho · 0905.2199v2.pdf · Fields Institute Personal

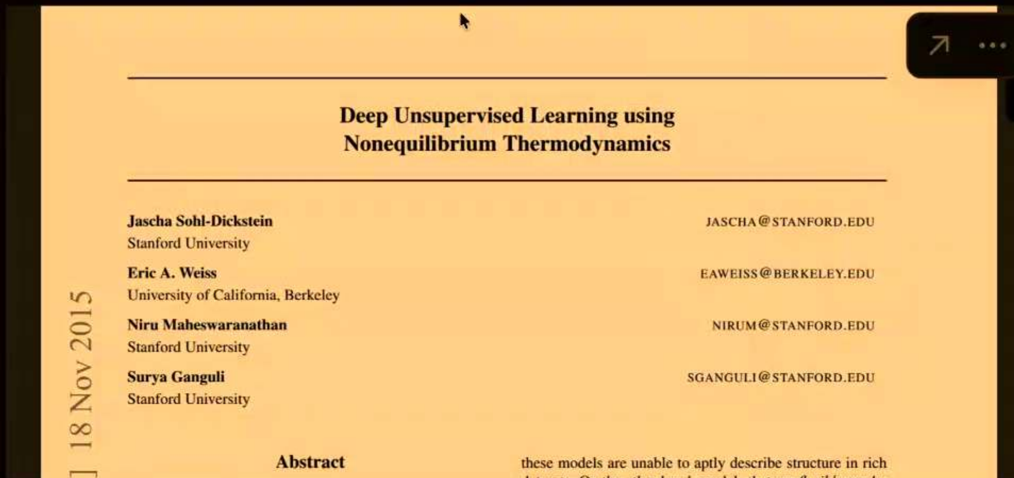
brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available



Diffusion model denoising process (source)

- Diffusion model comes from non equilibrium thermodynamics. In fact the first diffusion model paper was:



Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University

Eric A. Weiss EAWEISS@BERKELEY.EDU
University of California, Berkeley

Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University

Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

18 Nov 2015

Abstract

these models are unable to aptly describe structure in rich

Diffusion Models | Notion | Diffusion Models | Stable Diffusion Online | Diffusion-Model-Demo.ip | Stable Diffusion Online | Calendar - Sehmimul Hoc | 0905.2199v2.pdf | Fields Institute Personal
brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3
New Chrome available

- Concretely, for a system with configurations x and energy function $E(x)$ at temperature T, at equilibrium, the probability of the configurations follow the Gibbs Distribution:

$$p(x) = \frac{e^{-\frac{E(x)}{T}}}{Z}$$

  where Z is the partition function

- blackboard 1
- Gibbs sampling is done using Langevin Algorithm (for small constant ϵ):

$$x_{t+1} = x_t - \frac{\epsilon}{2}\nabla_x E(x_T) + \sqrt{\epsilon}\mathcal{N}(0, \mathcal{I})$$

- But this Langevin Algorithm is Equilibrium dynamics and so we know it will take a long time for it to converge.

Diffusion Models | Notion × Diffusion Models × Stable Diffusion Online × Diffusion-Model-Demo.ip × Stable Diffusion Online × Calendar - Sehmimul Ho × 0905.2199v2.pdf × Fields Institute Personal ×

brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available

- Concretely, for a system with configurations x and energy function $E(x)$ at temperature T, at equilibrium, the probability of the configurations follow the Gibbs Distribution:

$$p(x) = \frac{e^{-\frac{E(x)}{T}}}{Z}$$

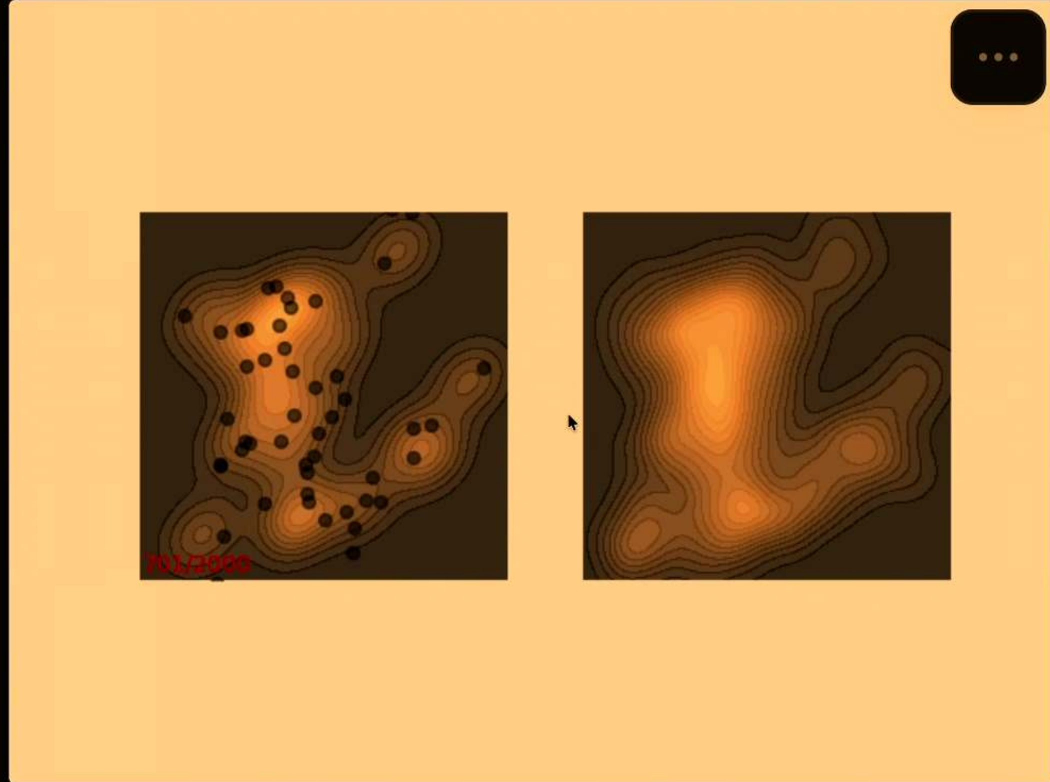
where Z is the partition function

- blackboard 1
- Gibbs sampling is done using Langevin Algorithm (for small constant ϵ):

$$x_{t+1} = x_t - \frac{\epsilon}{2}\nabla_x E(x_T) + \sqrt{\epsilon}\mathcal{N}(0, \mathcal{I})$$

- But this Langevin Algorithm is Equilibrium dynamics and so we know it will take a long time for it to converge.

this Langevin Algorithm is Equilibrium dynamics and so we know it will take a long time it to converge.



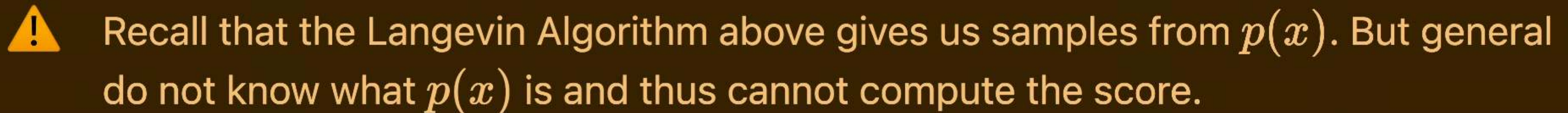
Langevin Simulation (source)

- Note that we can rewrite the the Langevin Algorithm in terms of the 'score':

$$x_{t+1} = x_t + \frac{\epsilon}{2} \nabla_x \log p(x_t) + \sqrt{\epsilon}\mathcal{N}(0, \mathcal{I})$$

where the score is $\nabla_x \log p(x)$

- blackboard 2



Diffusion Models | Notion × Diffusion Models × Stable Diffusion Online × Diffusion-Model-Demo.i × Stable Diffusion Online × Calendar - Sehmimul Ho × 0905.2199v2.pdf × Fields Institute Personal ×

brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available

Recall that the Langevin Algorithm above gives us samples from $p(x)$. But general do not know what $p(x)$ is and thus cannot compute the score.

- So we need to estimate the score by a neural network!
- Ideally we need to minimize:

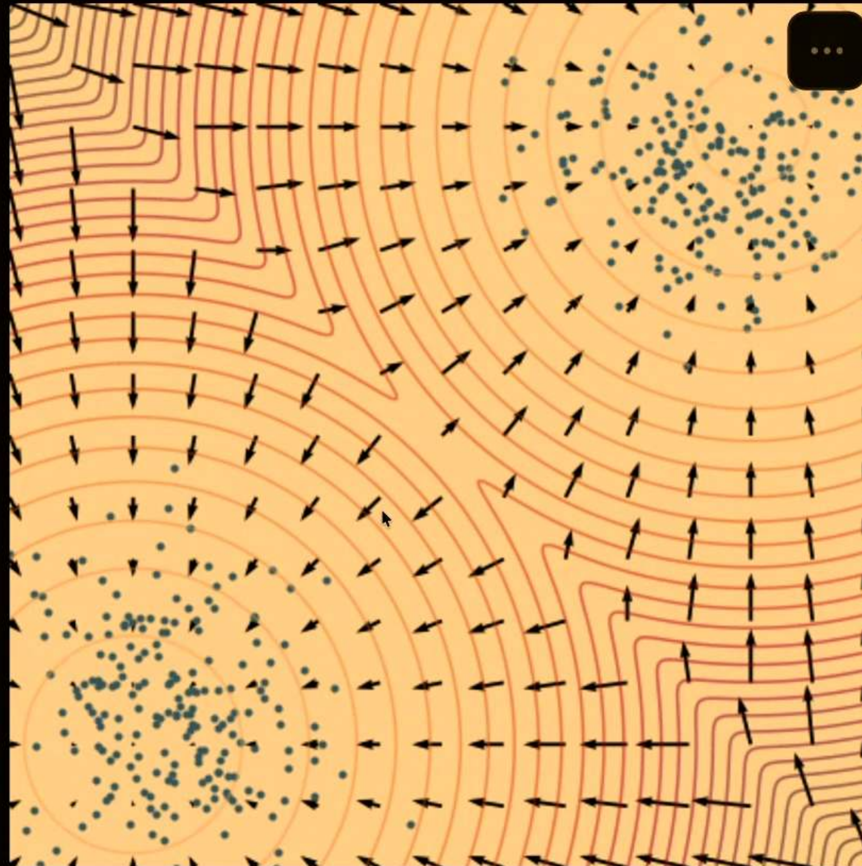
$$L(x) = \mathbb{E}_{x \sim p(x)} || s_\theta(x) - \nabla_x \log p(x) ||$$

- There is an *advantage* 😄 and a *disadvantage* 😞 of minimizing this instead of estimating directly. What are they?
- The loss can actually be written with respect to data samples: 🔗 www.iro.umontreal.c

▶ extra

Diffusion Models | Notion | Diffusion Models | Stable Diffusion Online | Diffusion-Model-Demo.i | Stable Diffusion Online | Calendar - Sehmimul Ho | 0905.2199v2.pdf | Fields Institute Personal
brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3
New Chrome available

samples from the data distribution $p(x)$!



Training to approximate the score and generating

- However, there is a major problem.
- Can anyone see what the problem is from the loss function?

$$L(x) = \mathbb{E}_{x \sim p(x)} || s_\theta(x) - \nabla_x \log p(x) ||$$

▶ *spoiler!*

- The fix to this problem is rather surprising! It's adding noise and we will cover this in the next section.

Noise Conditional Score Network

This section is based on arXiv.org Generative Modeling by Estimating Gradients of the Data Dist...

Perturbed density	Perturbed scores	Estimated scores

Diffusion Models | Notion × Diffusion Models × Stable Diffusion Online × Diffusion-Model-Demo.i × Stable Diffusion Online × Calendar - Sehmimul Ho × 0905.2199v2.pdf × Fields Institute Personal × +

brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available

- blackboard 3
- Extra
- The loss derived on the blackboard is:

$$\mathcal{L}_{NCSN} = \mathbb{E}_t \mathbb{E}_{x_0 \sim p_{data}} \mathbb{E}_{x_t \sim \mathcal{N}(x_0, \sigma_t^2 \mathcal{I})} \left[\left\| s_\theta(x_t, \sigma_t) + \frac{x_t - x_0}{\sigma_t^2} \right\|^2 \right]$$

- So now we know how training works. What about inference?
- blackboard 4

Score Based Diffusion Models

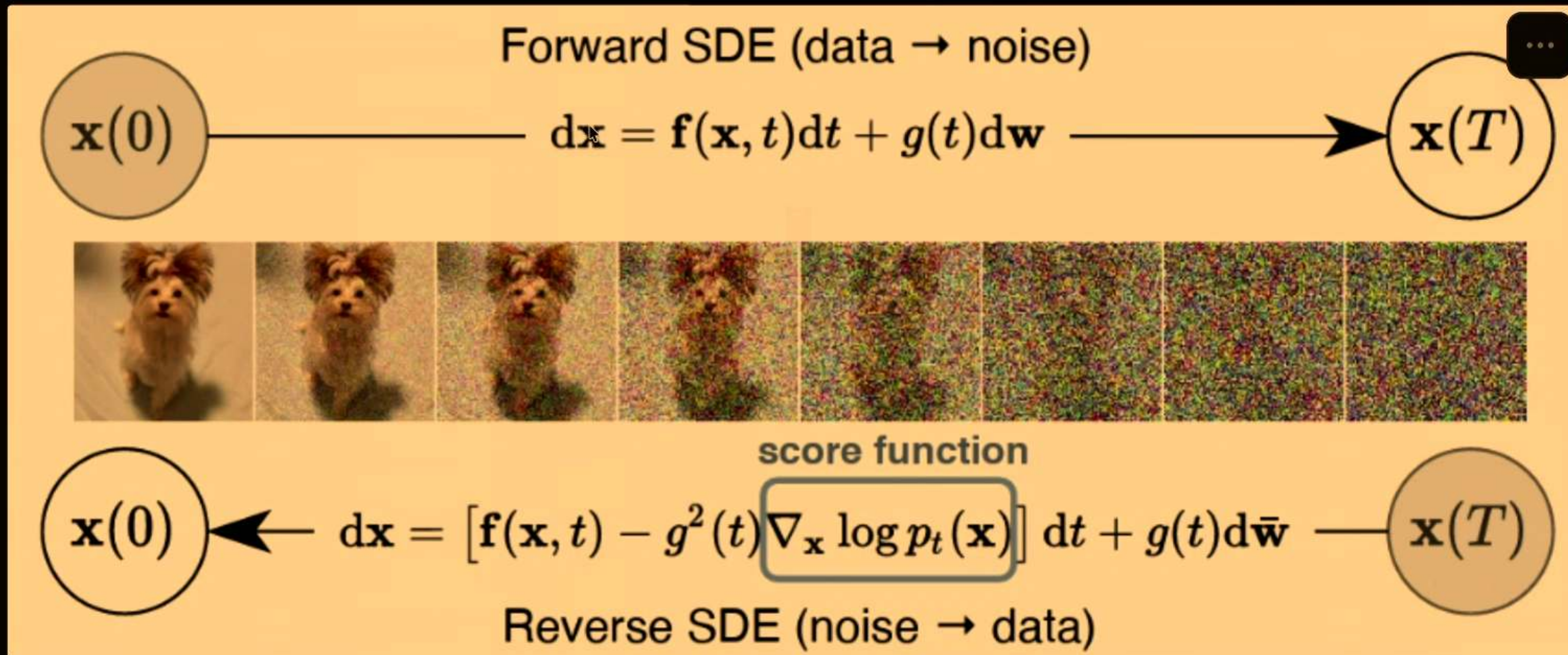
These models are more general than NCSN and are based on Stochastic Differential Equations

Diffusion Models | N × | Diffusion Models × | Diffusion-Model-Den × | Stable Diffusion Onli × | Diffusion-Model-Den × | Stable Diffusion Onli × | Calendar - Sehmimul × | 0905.2199v2.pdf × | Fields Institute Perso ×

brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available

- Hence we have a forward noising process and a reverse denoising process.



Diffusion process using SDEs (source)

- blackboard 5
- The loss is the same as before!

Diffusion Models | Diffusion Models | Diffusion-Model- | Diffusion-Model- | Stable Diffusion | Diffusion-Model- | Stable Diffusion | Calendar - Sehm | 0905.2199v2.pdf | Fields Institute

colab.research.google.com/drive/1k_rVSCZUQ_gSr2V3WgNLCwKc53C9fzWy?usp=sharing#scrollTo=5dHatk_gOHDf

New Chrome available

Demo of Score Based Diffusion model

Here we will see how to implement a simple diffusion model from scratch for MNIST

Setup and data

We start with the necessary imports and make sure you are connected to a T4 Iinstant provided by colab!

```
import os
import random
import numpy as np
import torch
from torch import nn, optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, utils
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
from torchvision.utils import make_grid


# device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
```

```
Device: cuda
```

Then we get the data and some data processing.

```
# Hyperparams you can adjust
batch_size = 128
root = "./data"

# MNIST transform - convert to tensor and scale to [-1,1] which is convenient for many diffusion examples
transform = transforms.Compose([
    transforms.ToTensor(),                    # -> [0,1]
    transforms.Lambda(lambda t: (t - 0.5) * 2) # -> [-1,1]
])
```

Labels: [9 9 4 0 4 5 1 0 5 8 7 7 2 9 2 5]

Unconditional Diffusion Model

Here we first define the diffusion process in the class called VESDE

Defining the SDE

```
class VESDE: #class used
  def __init__(self, sigma_min=0.01, sigma_max=50, N=1000): #sigma is a noise thing, sigma<simga2<...<sigmaT
    """Construct a Variance Exploding SDE.

    Args:
      sigma_min: smallest sigma.
      sigma_max: largest sigma.
      N: number of discretization steps
    """
    super().__init__()
    self.sigma_min = sigma_min
    self.sigma_max = sigma_max
    self.N = N
    self.probability_flow = False

  @property
```

Diffusion Models | N× Diffusion Models × Diffusion-Model-De× Stable Diffusion Onl× Diffusion-Model-De× Stable Diffusion Onl× Calendar - Sehmimul× 0905.2199v2.pdf × Fields Institute Pers×

colab.research.google.com/drive/1k_rVSCZUQ_gSr2V3WgNLCwKc53C9fzWy?usp=sharing&authuser=1#scrollTo=eZnjwFRcPdxb

New Chrome available

Defining the SDE

```
class VESDE: #class used
  def __init__(self, sigma_min=0.01, sigma_max=50, N=1000): #sigma is a noise thing, sigma<simga2<...<sigmaT
    """Construct a Variance Exploding SDE.

    Args:
      sigma_min: smallest sigma.
      sigma_max: largest sigma.
      N: number of discretization steps
    """
    super().__init__()
    self.sigma_min = sigma_min
    self.sigma_max = sigma_max
    self.N = N
    self.probability_flow = False

  @property
  def T(self):
    return 1

  def sde(self, x, t):
    sigma = self.sigma_min * (self.sigma_max / self.sigma_min) ** t
    drift = torch.zeros_like(x) #is the f in the formula (dx=fdt+gdW)
    diffusion = sigma * torch.sqrt(torch.tensor(2 * (np.log(self.sigma_max) - np.log(self.sigma_min)),
                                                device=t.device))
    return drift, diffusion

  def forward(self, x, t):
    std = self.sigma_min * (self.sigma_max / self.sigma_min) ** t
```

Diffusion-Model-Demo.ipynb ☆ Saving failed since 14:57

Edit View Insert Runtime Tools Help

nds + Code + Text ▷ Run all

```
    return mean, std

  def prior_sampling(self, shape):
    return torch.randn(*shape) * self.sigma_max


  def reverse_sde(self, x, t, score_fn, probability_flow = False, labels = None):
    """Create the drift and diffusion functions for the reverse SDE/ODE."""
    drift, diffusion = self.sde(x, t)
    if labels == None:
      score = score_fn(x, t)
    else:
      score = score_fn(x, t, labels)
    drift = drift - diffusion[:, None, None, None] ** 2 * score * (0.5 if probability_flow else 1.)

    diffusion = 0. if probability_flow else diffusion # Set the diffusion function to zero for ODEs. (not relevant fo
    return drift, diffusion
```

Exercise on forward noising

In the next section I want you to add noise to a small batch of images. Choose:

```
    ax.axis("off")

plt.show()
```

t = 0.00 | t = 0.25 | t = 0.50 | t = 0.75 | t = 1.00

Creating the Score Network

This score network is a UNET containing time embedding. Note that it is AI generated. :)

```
# ------------------------
# Helpers: Time embeddings
# ------------------------

class SinusoidalTimeEmbedding(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.dim = dim

    def forward(self, t):
        """
        t: shape [B]
        returns [B, dim]
        """
```

Diffusion Models | N × Diffusion Models × Diffusion-Model-Den × Stable Diffusion Onli × Diffusion-Model-Den × Stable Diffusion Onli × Calendar - Sehmimul × 0905.2199v2.pdf × Fields Institute Pers ×

colab.research.google.com/drive/1k_rVSCZUQ_gSr2V3WgNLCwKc53C9fzWy?usp=sharing&authuser=1#scrollTo=FVvOdM1vSL5b

New Chrome available

```
score_fn = get_score_fn(sde, model)

t = torch.rand(batch.shape[0], device=batch.device) * (sde.T - config.eps) + config.
z = torch.randn_like(batch)
mean, std = sde.forward(batch, t)
perturbed_data = mean + std[:, None, None, None] * z
score = score_fn(perturbed_data, t)

losses = torch.square(score * std[:, None, None, None] + z)

losses = torch.mean(losses.reshape(losses.shape[0], -1), dim=-1)
loss = torch.mean(losses)

if train == True:
    optim.zero_grad()
    loss.backward()
    optim.step()

return loss
```

Diffusion Models | N × Diffusion Models × Diffusion-Model-Der × Stable Diffusion Onli × Diffusion-Model-Der × Stable Diffusion Onli × Calendar - Sehmimul × 0905.2199v2.pdf × Fields Institute Perso ×

colab.research.google.com/drive/1k_rVSCZUQ_gSr2V3WgNLCwKc53C9fzWy?usp=sharing&authuser=1#scrollTo=N41JNjKI_Y6l

New Chrome available

```
        pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{config.epochs}")

        for batch, _ in pbar:
            batch = batch.to(config.device)

            loss = train_step(config, batch, model, sde, optimizer, train=True)
            epoch_losses.append(loss.item())

            pbar.set_postfix(loss=loss.item())

        mean_epoch_loss = sum(epoch_losses) / len(epoch_losses)
        train_losses.append(mean_epoch_loss)

        print(f"Epoch {epoch+1}: loss = {mean_epoch_loss:.6f}")

    # ======================
    # PLOT TRAINING CURVE
    # ======================
    plt.figure(figsize=(7,5))
    plt.plot(train_losses, marker='o')
    plt.title("Score-based Diffusion Training Loss (MNIST)")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.grid(True)
    plt.show()
```

```
Epoch 1/4: 100%  469/469 [01:06<00:00,  6.89it/s, loss=0.0926]
Epoch 1: loss = 0.143330
Epoch 2/4:  22%  101/469 [00:14<00:52,  6.95it/s, loss=0.0779]
```

Sampler

```python
    for epoch in range(config.epochs):
        epoch_losses = []
        pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{config.epochs}")

        for batch, _ in pbar:
            batch = batch.to(config.device)

            loss = train_step(config, batch, model, sde, optimizer, train=True)
            epoch_losses.append(loss.item())

            pbar.set_postfix(loss=loss.item())

        mean_epoch_loss = sum(epoch_losses) / len(epoch_losses)
        train_losses.append(mean_epoch_loss)

        print(f"Epoch {epoch+1}: loss = {mean_epoch_loss:.6f}")

    # =====================
    # PLOT TRAINING CURVE
    # =====================
    plt.figure(figsize=(7,5))
    plt.plot(train_losses, marker='o')
    plt.title("Score-based Diffusion Training Loss (MNIST)")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.grid(True)
    plt.show()
```

```
Epoch 1/4: 100% 469/469 [01:06<00:00,  6.89it/s, loss=0.0926]
Epoch 1: loss = 0.143330
Epoch 2/4: 100% 469/469 [01:07<00:00,  7.45it/s, loss=0.085]
Epoch 2: loss = 0.077144
Epoch 3/4: 100% 469/469 [01:07<00:00,  7.53it/s, loss=0.0481]
Epoch 3: loss = 0.070427
Epoch 4/4:  44% 205/469 [00:29<00:38,  6.93it/s, loss=0.0593]
```

Sampler

Epoch 1/4: 100% 469/469 [01:06<00:00, 6.89it/s, loss=0.0926]
```
Epoch 1: loss = 0.143330
```
Epoch 2/4: 100% 469/469 [01:07<00:00, 7.45it/s, loss=0.085]
```
Epoch 2: loss = 0.077144
```
Epoch 3/4: 100% 469/469 [01:07<00:00, 7.53it/s, loss=0.0481]
```
Epoch 3: loss = 0.070427
```
Epoch 4/4: 100% 469/469 [01:07<00:00, 7.45it/s, loss=0.0656]
```
Epoch 4: loss = 0.066621
```

Sampler

↳ 5 cells hidden

sion-Model-Demo.ipynb ☆ Saving failed since 14:57

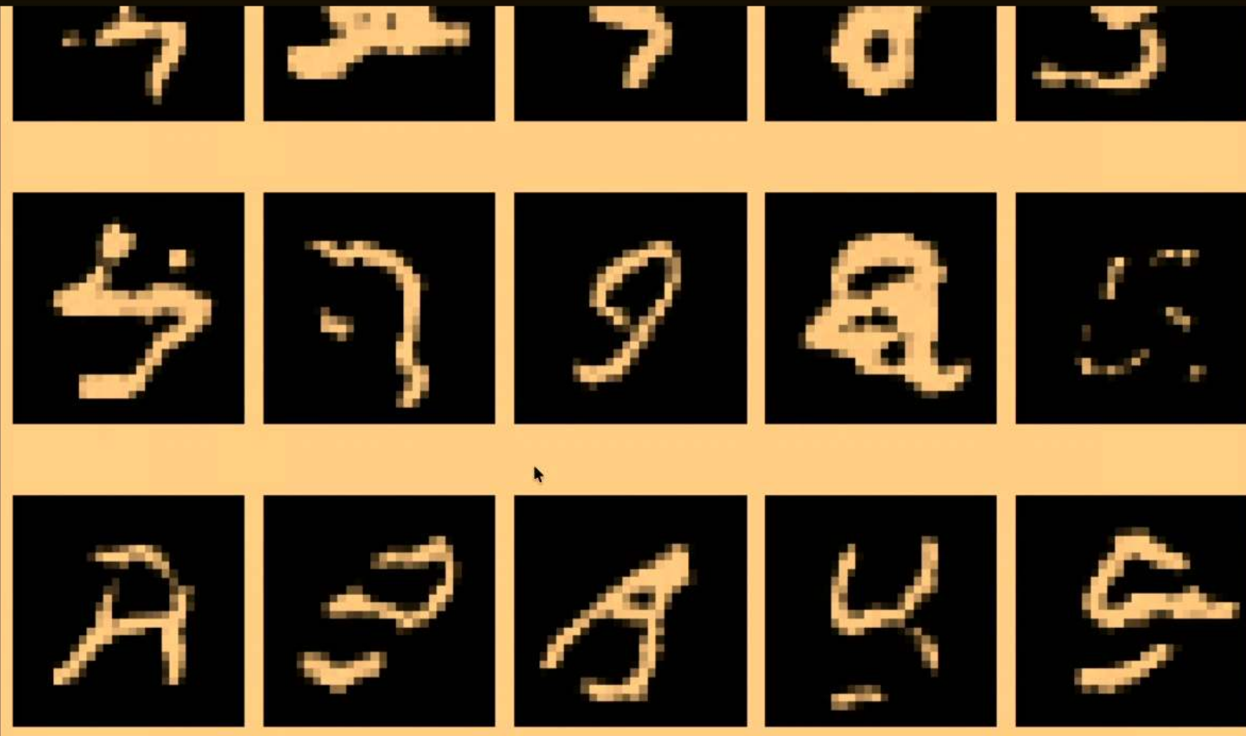
View Insert Runtime Tools Help

+ Code + Text ▷ Run all

```
def sample(sde, model, num_samples = 1000, eps=1e-5, device = torch.device("cuda" if torch.cuda.is_available
    model.eval()
    score_fn = get_score_fn(sde, model)
    sampler = Sampler(sde = sde, score_fn = score_fn)
    shape = [num_samples, 1, 28, 28] # the shape has been hardcoded!
    x = sde.prior_sampling(shape).to(device)
    timesteps = torch.linspace(sde.T, eps, sde.N, device=device)
    with torch.no_grad():
      for i in tqdm(range(sde.N)):
          t = torch.ones(shape[0], device=device) * timesteps[i]
          x, x_mean = sampler.update_fn(x, t)

    return x_mean
```

d now we generate samples!



Conditional Diffusion exercise

↳ 11 cells hidden

Conditional Diffusion (solution)

For conditional, we need a new neural network which has to take in the labels

w let's generate samples

```
conditioning = 1
num_samples = 20
labels = torch.ones(num_samples).to(device).long()*conditioning
samples = sample(sde, model, labels = labels, eps = config.eps, num_samples = num_samples)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
```

```
samples = samples.detach().cpu()   # ensure on CPU, no grad

fig, axes = plt.subplots(4, 5, figsize=(10, 10))

for i, ax in enumerate(axes.flatten()):
    img = samples[i, 0]            # [28, 28]
    ax.imshow(img, cmap="gray")
    ax.axis("off")

plt.tight_layout()
```

Diffusion-Model-Demo.ipynb

```
        pbar.set_postfix(loss=loss.item())

    mean_epoch_loss = sum(epoch_losses) / len(epoch_losses)
    train_losses.append(mean_epoch_loss)

    print(f"Epoch {epoch+1}: loss = {mean_epoch_loss:.6f}")
# =====================
# PLOT TRAINING CURVE
# =====================
plt.figure(figsize=(7,5))
plt.plot(train_losses, marker='o')
plt.title("Score-based Diffusion Training Loss (MNIST)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

```
Epoch 1/4: 100% 469/469 [00:43<00:00, 10.42it/s, loss=0.105]
Epoch 1: loss = 0.209856
Epoch 2/4: 100% 469/469 [00:44<00:00, 10.83it/s, loss=0.0728]
Epoch 2: loss = 0.098499
Epoch 3/4: 100% 469/469 [00:43<00:00, 10.99it/s, loss=0.0697]
Epoch 3: loss = 0.086069
Epoch 4/4: 100% 469/469 [00:44<00:00, 11.09it/s, loss=0.0794]
Epoch 4: loss = 0.080977
```

Score-based Diffusion Training Loss (MNIST)

Loss: 0.20, 0.18, 0.16, 0.14, 0.12, 0.10

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```python
    def __init__(self, sde, score_fn):
      super().__init__()
      self.sde = sde
      self.score_fn = score_fn

    def update_fn(self, x, t, labels):
      dt = -1. / self.sde.N
      z = torch.randn_like(x)
      drift, diffusion = self.sde.reverse_sde(x, t, self.score_fn, labels = labels)
      x_mean = x + drift * dt
      x = x_mean + diffusion[:, None, None, None] * np.sqrt(-dt) * z
      return x, x_mean


  def sample(sde, model, labels = None, num_samples = 1000, eps=1e-5, device = torch.device("cuda" if torch.cuda.is_available() else "cpu")):
      model.eval()
      score_fn = get_score_fn(sde, model)
      sampler = ConditionalSampler(sde = sde, score_fn = score_fn)
      shape = [num_samples, 1, 28, 28] # the shape has been hardcoded!
      x = sde.prior_sampling(shape).to(device)
      timesteps = torch.linspace(sde.T, eps, sde.N, device=device)
      with torch.no_grad():
        for i in tqdm(range(sde.N)):
            t = torch.ones(shape[0], device=device) * timesteps[i]
            x, x_mean = sampler.update_fn(x, t, labels)

      return x_mean
```

Now let's generate samples

```python
conditioning = 8
num_samples = 20
labels = torch.ones(num_samples).to(device).long()*conditioning
samples = sample(sde, model, labels = labels, eps = config.eps, num_samples = num_samples)
```

100% 1000/1000 [00:07<00:00, 148.00it/s]

```python
samples = samples.detach().cpu()   # ensure on CPU, no grad

fig, axes = plt.subplots(4, 5, figsize=(10, 10))

for i, ax in enumerate(axes.flatten()):
    img = samples[i, 0]            # [28, 28]
    ax.imshow(img, cmap="gray")
    ax.axis("off")
```

Diffusion Models | N × Diffusion Models × Diffusion-Model-Den × Stable Diffusion Onli × Diffusion-Model-Den × Stable Diffusion Onli × Calendar - Sehmimul × 0905.2199v2.pdf × Fields Institute Pers ×

brief-mango-0be.notion.site/Diffusion-Models-2afd31660db3806ba3dbe81c2d5be6f3

New Chrome available

$$dx = [f(x,t) - g^2(t)s_\theta(x(t),t)]dt + g(t)dW_t$$

for finite time N to 0

Conditional Diffusion models

- There are 3 ways to do conditional diffusion models
  - Simple conditioning
  - Classifier Guidance: arXiv.org **Score-Based Generative Modeling through Stochastic Different...**
  - ...lassifier-Free Diffusion Guidance

Clas... ...of the art

- blac...

Score-Based Generative Modeling through Stochastic Differential...

Creating noise from data is easy; creating data from noise is generative modeling. We present a stochastic differential equation...

arXiv.org

▼ blackboard 1 and 2

https://arxiv.org/abs/2011.13456