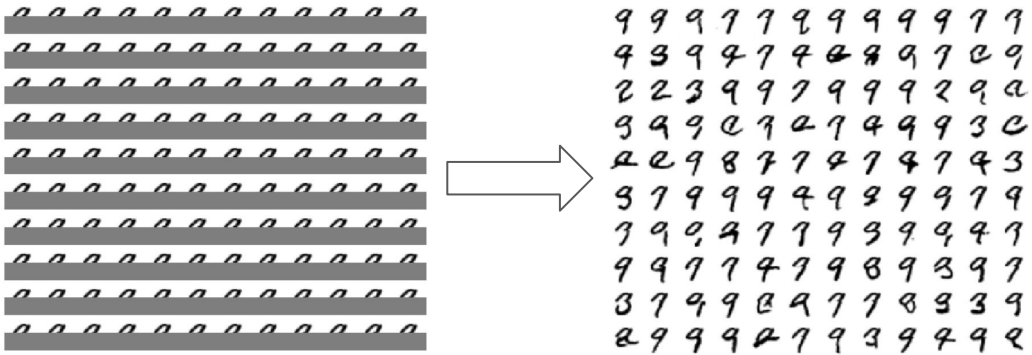
CS-E4890: Deep Learning
Q&A Session Assignment 8 - Diffusion

Nicola Dainese, 5/05/2023

Denoising Diffusion Probabilistic Models (DDPM)

In this assignment:

- We follow the paper Denoising Diffusion Probabilistic Models (DDPM) from Ho et al. 2020.
- We implement:
  - the **forward and reverse diffusion processes** with Gaussian noise,
  - a **U-net** based architecture as a **denoising model**,
  - a **training loop** for MNIST dataset,
  - an **in-painting** function, that completes an image of a digit given only a part of it.



Why diffusion models?

We evaluate image generative models along three axes:

1. **Image quality:** how good do the images look?
   - depends on the generative process, training objective and neural architecture.
2. **Diversity:** how much variety of samples we get?
   - depends on the training objective; likelihood based models (e.g. VAEs) are the best in this, while GANs have problems.
3. **Generation speed:** how fast is to generate an image?
   - how many times do we need to call the model to generate an image? VAEs and GANs both require a single call, so they're quite fast.

Diffusion:
multiple network iterations
during sampling
+
likelihood-based method
=
good quality
good diversity
slow generation

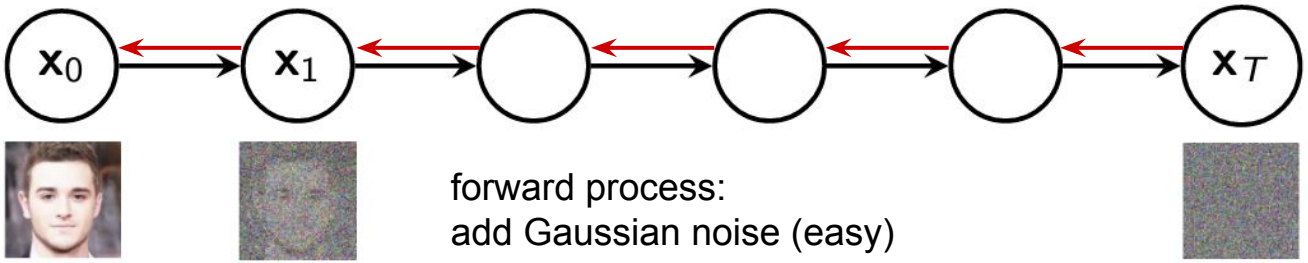
Paradigm	Quality	Diversity	Speed
VAE	✗	✓	✓
GAN	✓	✗	✓
Diffusion	✓	✓	✗

What are diffusion models?

Class of latent variable models trained to denoise corrupted data by reversing the corruption process.

Corruption scheme, a.k.a. forward process, is a fixed Markov chain that gradually adds Gaussian noise to the data.

reverse process:
remove Gaussian noise (dark magic?)



forward process:
add Gaussian noise (easy)

training data distribution

learned distribution

$$q(x_0) \approx p_\theta(x_0)$$

Gaussian noise

$$q(x_T) \approx \mathcal{N}(0, I)$$

Why diffusion works?

We want to learn how to approximate $q(x_0)$, but we know only how to sample from it.

Therefore we need to learn an approximation to $q(x_{t-1}|x_t)$, so that we can apply it for all t=T,...,1 starting from Gaussian noise and get an approximation of $q(x_0)$.

Assume Gaussian distribution of p_θ

$$q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t)$$

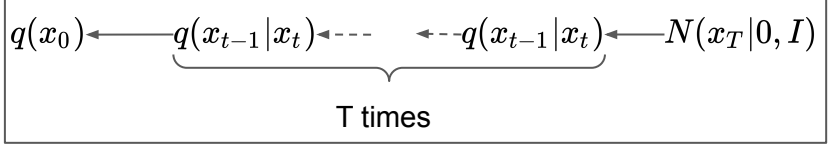
Note that if we condition on x_0 we have an exact expression* for the expected value

$$\mathbb{E}[x_{t-1}|x_t(x_0), x_0] = \tilde{\mu}_t(x_t(x_0), x_0)$$

Learn a marginalization of the expected denoised value by training a denoising model

$$\min_\theta \mathop{\mathbb{E}}_{x_0 \approx q(x_0)} [||\mu_\theta(x_t, t) - \tilde{\mu}(x_t(x_0), x_0)||^2]$$

Sampling Gaussian noise
+
T steps of denoising
=
Generative DDPM model



*see Eq. 7 of the DDPM paper

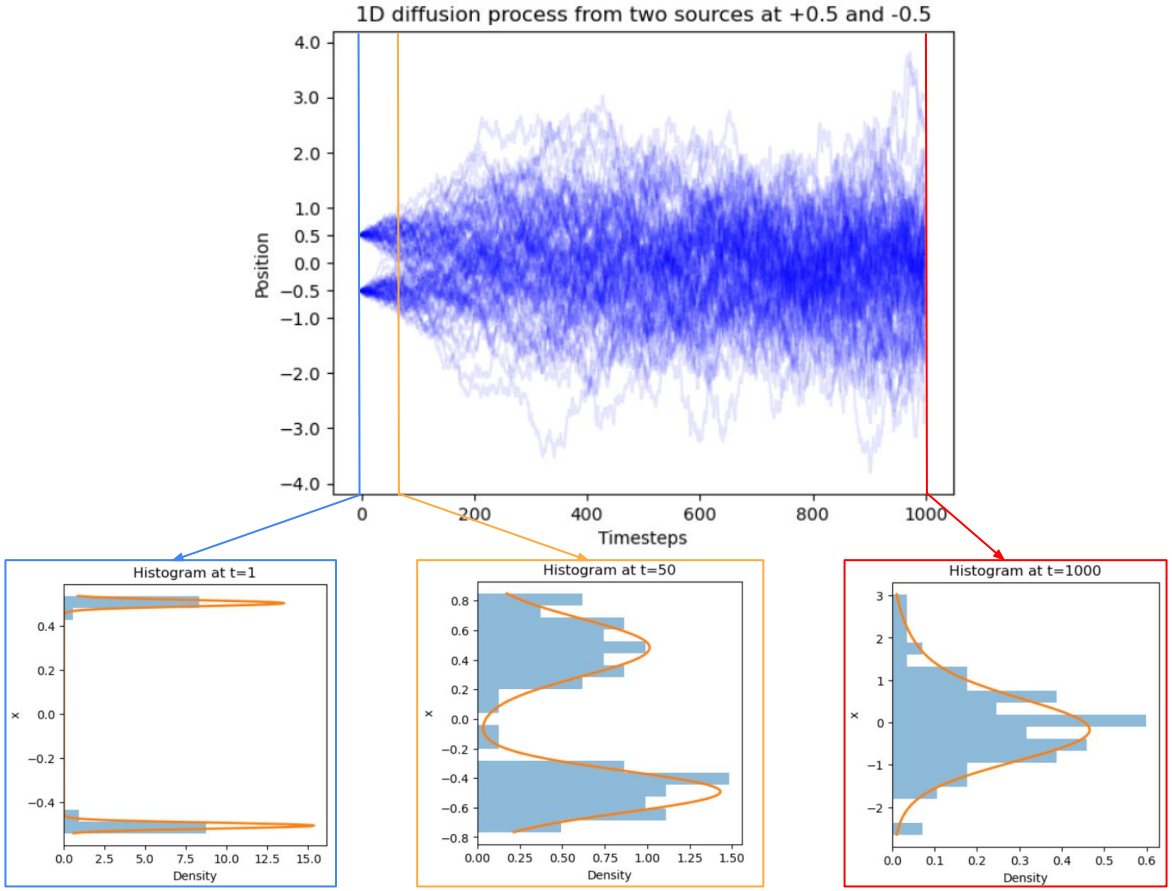
Why diffusion works? A simple example

Let's consider a 1D toy example

Dataset consisting of two points:

$\{x_A=-0.5, x_B=+0.5\}$

with equal probability of being sampled.



Why diffusion works? A simple example (pt. 2)

The forward process reduces the signal and adds noise to the data step-by-step:

$$q(x_t|x_{t-1}) := N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

If the noise is completely random, what is there to learn?

x_t carries information about all possible x_{t-1} that can lead to it and the respective transition probabilities.

For instance in the example $x_t^{(A)}$ is much more likely to come from $x_{t-1}^{(A)}$ then from $x_{t-1}^{(B)}$ and the denoising model will learn that through training.

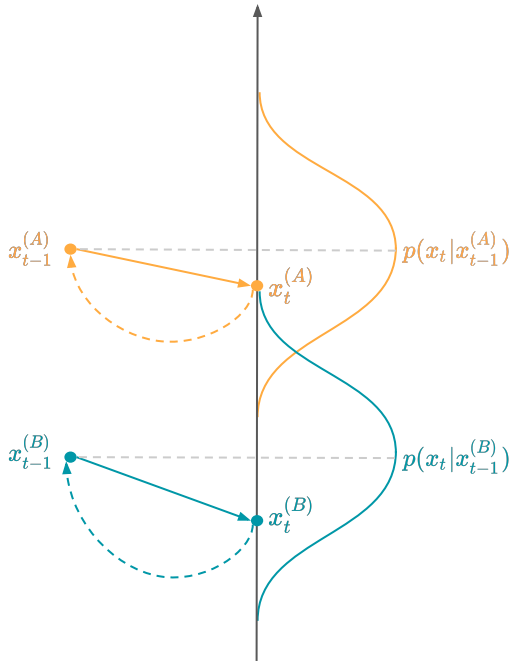


Illustration of the forward (solid lines) and reverse (dashed lines) diffusion process in the case of two points.

How diffusion works?

Simplest form

1. Sample x_0 and t,
2. Obtain x_t through the forward process,
3. Compute the expected value of x_{t-1} given x_t and x_0 (analytical formula),
4. Learn to predict that value given only x_t and t.

$$\min_{\theta} \mathbb{E}_{x_0 \approx q(x_0)} [||\mu_\theta(x_t, t) - \tilde{\mu}(x_t(x_0), x_0)||^2]$$

Reparametrization trick

Make explicit the noise ε that from x_0 brings us to x_t .
Analytic formula for x_t given x_0 and ε :

$$x_t(x_0, \epsilon) = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

Plug this into the formula above, do some math and it turns out that it's equivalent to train the model to predict the noise ε that produces x_t:

$$\mathbb{E}_{x_0, \epsilon} \left[\frac{\beta_t}{2\alpha_t(1-\bar{\alpha}_t)} ||\epsilon - \epsilon_\theta(x_t(x_0, \epsilon), t)||^2 \right]$$

Objective	IS	FID
$\tilde{\mu}$ prediction (baseline)		
L, learned diagonal Σ	7.28 ± 0.10	23.69
L, fixed isotropic Σ	8.06 ± 0.09	13.22
$\|\|\tilde{\mu} - \tilde{\mu}_\theta\|\|^2$	–	–
ϵ prediction (ours)		
L, learned diagonal Σ	–	–
L, fixed isotropic Σ	7.67 ± 0.13	13.51
$\|\|\tilde{\epsilon} - \epsilon_\theta\|\|^2$ (L_{simple})	$\mathbf{9.46 \pm 0.11}$	**3.17**

How diffusion works?

Final simplification

from

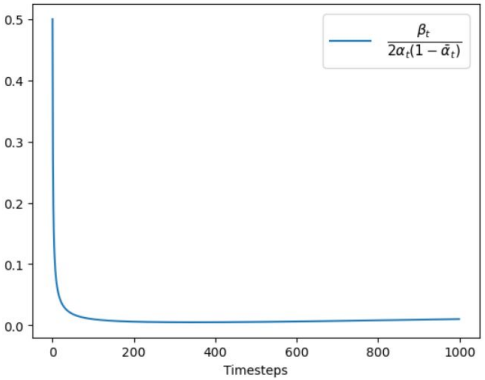
$$\mathbb{E}_{x_0,\epsilon}\left[\frac{\beta_t}{2\alpha_t(1-\bar{\alpha}_t)}||\epsilon-\epsilon_\theta(x_t(x_0,\epsilon),t)||^2\right]$$

to

$$\mathbb{E}_{x_0,\epsilon}\left[||\epsilon-\epsilon_\theta(x_t(x_0,\epsilon),t)||^2\right]$$

The weighting term (see on the right) gives more weight to terms with small t.
Those terms, which have relatively little noise, are quite easy to denoise, thus dropping the weight gives dramatic improvements (last hop on the table).

Objective	IS	FID
$\tilde{\mu}$ prediction (baseline)		
L, learned diagonal Σ	7.28±0.10	23.69
L, fixed isotropic Σ	8.06±0.09	13.22
$\|\|\tilde{\mu}-\tilde{\mu}_\theta\|\|^2$	–	–
ϵ prediction (ours)		
L, learned diagonal Σ	–	–
L, fixed isotropic Σ	7.67±0.13	13.51
$\|\|\tilde{\epsilon}-\epsilon_\theta\|\|^2$ (L_{simple})	**9.46±0.11**	**3.17**



Practical tips for the assignment

Check carefully the timestep convention, t takes values in [0,num_timesteps-1] in the assignment, while in Algorithm 1 and 2 from the paper t takes values in [1, num_timesteps].

Pay attention and have patience in implementing the U-Net architecture; in particular the skip connections are concatenated and not added to the signal. This also affects the amount of input channels of the decoder residual blocks.

During training, the same noise ε is used to get x_t from Diffusion.forward and as target for the MSE loss.

Training on GPU should take roughly 20 minutes for 20 epochs.

The in-painting part is a bit tricky, but notice that you can do the usual denoising step and then change the known part with a more accurate (but still noisy) estimate of x_{t-1} given x_t and x_0.

Good luck!