Advanced Course on Deep Generative Models **Spring 2024**

Exercise 7: Denoising Diffusion Probabilistic Models

Dan Rosenbaum

Due: 31 July 2024

In this exercise we will use a modified version of the notebook from previous exercises.

`https://colab.research.google.com/drive/14PUswAVnvOZ4QOg3kR6RfemjUf27fiNX?usp=sharing`

The goal of this exercise is to understand and implement the DDPM model, as presented in class.

In order to do that, use the same training process used in previous exercises. Follow the instructions below:

- **UNet:** You will need to implement a UNet architecture to be used to estimate the score function (which is equivalent to estimating the noise) in each step of the model. You are given a partially implemented model. You need to understand it and implement the *forward* function. Note that the time parameter $t \in [0, T)$ given to the model is an integer and needs to be converted to float in the range $[0, 1]$.
- **DDPM:** Implement the functions used for DDPM training and sampling. These include the q function which computes $q(x_t|x_0; t)$, the *reverse_q* function that computes $q(x_{t-1}|x_t; t, \varepsilon_t)$, the function that computes the loss for each training step, and a function that generates samples. The computation of the diffusion variance schedule and the corresponding parameters are already implemented (β_t, α_t, $\bar{\alpha}_t$).

The images are normalized to the range $[-1, 1]$ which is standard when using diffusion based models.

Assignments

1. We have seen in the lectures that $q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t)$. Use Bayes theorem and our assumptions on the prior to prove it, and find $\tilde{u}_t, \tilde{\beta}_t$. You can use claims from hw1 without proving them.

2. Train the implemented model on the MNIST data for at least 20 epochs. You should see that during the training, samples start to look like valid digits. After training, report the loss curve for training and generate 16 samples. Are they recognizable?

3. Now, we will use the model to perform probabilistic inference. The task is to generate samples of half the image, given the other half. Do this for both the bottom half given the top half and the top half given the bottom half.

In order to perform that, you will need to implement:

complete_image(y=None, ymask=None, scale=1.)

This should be similar to the *sample* function, but after each step of sampling from $q(x_{t-1}|x_t; t; \epsilon_t)$, the following correction term is added:

$$-s\nabla_{x_t}\mathrm{MSE}(y_m \cdot (\hat{x}_0 - y))$$

where y is an image where half of the pixels are masked out (multiplied by zero), y_m is a binary mask indicating which pixels of y are given, and s is some scaling measurement.

The function that computes $\hat{x}_0 = \mathbb{E}[x_0|x_t]$ at each step t is already implemented. The function also clips the values to be between -1 and 1, which can stabilize the results. You can use this function also to compute $q(x_{t-1}|x_t; t; \epsilon_t)$ both in the *complete_image* function and in the *sample* function. Sometimes this leads to more stable samples because of the clipping.

You can compute the gradient with respect to either x_t before using the UNet (this will compute backpropogation through the UNet), or the mean of x_{t-1} (before adding the noise and correction term). The second option should be faster.

Choose 6 images and hide the bottom half of the first 3, and the top half for the second 3. Do this by multiplying the images by a binary mask. For each image generate 5 different conditional samples. Make sure to present the images (6), the visible half of the images (6) and the completed samples (6×5).

You should submit a colab notebook that contains the text of the answers, figures and code for all the above questions.

Bonus (10 pts)

If you choose to do the bonus, please add a note in your submission stating that you did so. Also, download clip-data.csv

So far we have only seen some results on MNIST, but now we will use another dataset and gain control over the sampled images. We will use CLIP, a pretrained model aiming encode an image into a representation that is as similar to the encoding of possible text describing that image. This will allow us to add context to the model.

For this part we will use a dataset of images of faces called *CelebA*.

Before you start, we recommend to look through the 'intro to clip' section and make sure you understand how it works.

Now, follow the instructions below:

- Implement the forward process of a modified UNet according to the specified architecture, you will need to add the conditioning of a context vector.
- Re-implement the DDPM model. You can copy the model you implemented and update it. In training time, each image is given along with a context vector that is computed from the image using CLIP. At sampling time, the model is given a context vector computed from a text prompt using CLIP. The sampling function should get a list of text prompts, and return the sampled images, one for each text. To implement this use classifier-free guidance as described in class. At training time use *conditioning dropout* by multiplying the given context by a random binary mask using the *get_context_mask* function. At sampling time, test the performance with different guidance scales.

Train the model for CelebA data, for at least 50 epochs. You can use a list of text prompts to generate samples during training in order to monitor the performance.

After training, generate a batch of unconditional samples, and a batch of samples conditioned in different text prompts.