Data Science Project A2 - IASD 2023
A Diffusion Approach to Image Generation

Group nico_luc_hipp: Nicolas Dunou, Hippolyte Gisserot, Lucas Marandat

1 Introduction

In the recent past, GANs and VAEs have found a lot of success and recognition. GANs work great for multiple applications, but they are difficult to train and their output lacks diversity. Although VAEs have a very solid theoretical foundation, the design of a relevant loss function has remained a critical and unsolved challenge.

There exists other techniques which originate from probabilistic likelihood estimation methods and take inspiration from a physical phenomenon, called diffusion models. The key concept in diffusion modelling is the following: if one can build a model able to learn a systematic information decay due to a noising process, then it should be possible to reverse it and and recover the information back from the generated noise.

In this report, we try to use this approach to generate new samples inspired from the well-known MNIST data set (70,000 handwritten digits of size 28x28, along with their respective labels), relying on the work of Ho and al. [HJA20] and Song and al. [SME20] on diffusion models.

2 Mathematical theory

Our goal is to find the probability distribution P_θ from which the MNIST samples are drawn.

2.1 Noising process

First of all, we need to define a noising process $(\beta_t)_{t=1}^T$, where T is the number of time steps used in the whole noising task.

$$P(x_t|x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}x_{t-1},\ \beta_t I) \iff x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t$$

Where x_t denotes a flatten image of size (784,) after t noising steps and
$(\epsilon_t)_{t=1}^T \sim \mathcal{N}(0, I)$ are i.i.d. random (784,) vectors.

From the above expression and by defining $\forall 1 \leq t \leq T,\ \alpha_t = (1-\beta_t),\ \bar{\alpha_t} = \prod_{s=1}^t \alpha_s$ we can derive that:

$$P(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha_t}}x_0,\ (1-\bar{\alpha_t})I)$$

Now, it becomes much clearer that we want to define the sequence $(\beta_t)_{t=1}^T$ such that $\bar{\alpha_t}$ is smoothly decreasing to 0, starting from $\bar{\alpha_0} = 1$. It is then up to us to decide whether we want to parameterize our noising process on the mean coefficient $\sqrt{\bar{\alpha_t}}$ or the noising coefficient $1 - \bar{\alpha_t}$. Either way, $(\beta_t)_{t=1}^T$ can easily be recovered: $\beta_t = 1 - \frac{\bar{\alpha_t}}{\bar{\alpha}_{t-1}}$.

Figure 1: Example of a noising process linear in $\sqrt{\bar{\alpha_t}}$, $T = 10$

2.2 Denoising task

Now that we have defined the noising process, we need to find a way to recover the x_0 corresponding to a given $x_T \sim \mathcal{N}(0, I)$, that is, we have to find the parameter θ such that:

$$x_0 \sim P_\theta(x_0|x_1)P_\theta(x_1|x_2)...P_\theta(x_{T-1}|x_T)\mathcal{N}(0, I)$$
$$\text{Where } P_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t, t),\ \Sigma_\theta(x_t, t)),\ \forall 1 \leq t \leq T$$

Using Bayes' rule, we get that:

$$P(x_{t-1}|x_t, x_0) = \mathcal{N}(\tilde{\mu}_t(x_t, x_0),\ \tilde{\beta}_t I)$$
$$\text{Where } \tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t \text{ and } \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

We finally use the previous result to approximate the unknown mean of each distribution $P_\theta(x_{t-1}|x_t)$. This task is performed by a neural network fed with two inputs: a t-noised image as well as its corresponding time step t.

$$\mu_\theta(x_t, t) \approx \tilde{\mu}_t(x_t, x_0)$$

2.3 Image generation

Once the denoiser is trained, it gets very straightforward to generate new images. One only needs to sample 28x28 images drawn from a $\mathcal{N}(0, I)$ distribution and then pass them through the whole denoising pipeline:

1. At $t = T$: generate 28x28 random noise $x_T \sim \mathcal{N}(0, I)$.
2. At $1 \leq t \leq T - 1$: predict $\mu_\theta(x_{t+1}, t + 1)$ using the denoiser and then sample around the mean according to the relevant variance scheduler, $x_t = \mu_\theta(x_{t+1}, t + 1) + \sqrt{\beta_t}\epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, I)$.
3. At t=0: predict $\mu_\theta(x_1, 1)$ and use it as the final denoised image x_0.

3 Performance assessment

In order to evaluate the performance of a generative model, we rely on the Frechet Inception Distance (FID), that provides some information about the distance between two distributions.

For two multivariate normal distributions $\mathcal{D}_1 = \mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{D}_2 = \mathcal{N}(\mu_2, \Sigma_2)$, the Frechet distance can be computed as follows:

$$dist(\mathcal{D}_1, \mathcal{D}_2)^2 = \|\mu_1 - \mu_2\|_2^2 + Tr(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{\frac{1}{2}})$$

Since the raw set of generated images is not normally distributed, we use the pre-trained InceptionV3 model to map the images to a Gaussian distribution before computing the Frechet distance.

4 Model experimentation

4.1 Dense neural network

As a benchmark, let's first implement a very basic approach using a dense neural network.

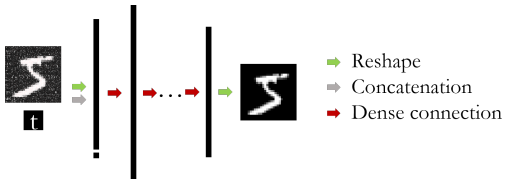

Figure 2: Dense neural network architecture

The main parameters used are the following: {*batch size*: 256; *number of epochs*: 2000; *loss function*: x-prev-l2; *noising process*: linear-noise; *normalization range*: [-1,1]} (see more details in sections 4.3.1 to 4.3.7).

We get an FID of 34.2.

Figure 3: 100 images generated by the dense model

4.2 Traditional convolutional network

The main issue when dealing with images as simple vectors is that we completely loose contextual information due to arbitrary flattening (there exists multiple ways of flattening an image).

In this section, we thus implement a more relevant architecture, based on 2D convolutional layers.

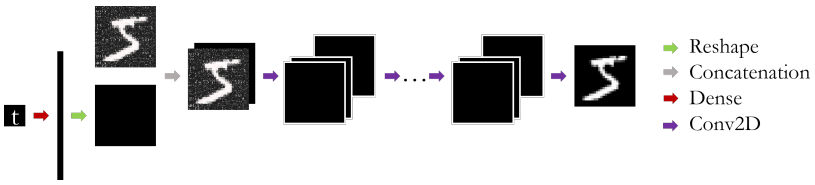

Figure 4: 2D-Convolutional neural network architecture

One can notice on the above diagram that we embed the time step as a 28x28 image, in order to add it as a second channel to our network input.

The main hyperparameters used in this section are the following: {*batch size*: 256; *number of epochs*: 2000; *loss function*: x-prev-l2; *number of convolutions*: 4; *number of channels*: 4; *noising process*: linear-noise; *normalization range*: [-1,1]} (see more details in sections 4.3.1 to 4.3.7)

We get an FID of 14.1, which is as expected, significantly better than the one obtained using the dense network.

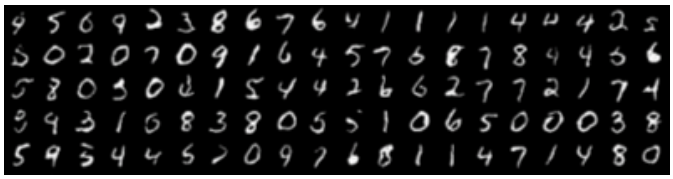

Figure 5: 100 images generated by the traditional convolutional model

4.3 U-Net

A U-Net is composed of an encoder and a decoder. The encoder learns the context of an image by decreasing its size and increasing the number of filters. The decoder performs the reverse path in order to reconstruct an image with the same size as the input's. It is helped in its decoding task by concatenation operations which are used to complete the information flow.

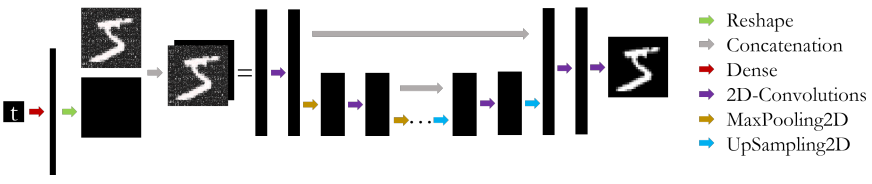

Figure 6: Dense neural network architecture

In this section, we perform sequential hyperparameter tuning using FID as an evaluation metric. That is, we start from the baseline model described just below, and then sequentially update the hyperparameters by choosing the ones that minimize the FID. This method is less accurate but much quicker than grid search.

The parameters of the baseline model are the following: {*batch size*: 256; *number of epochs*: 50; *loss function*: x-prev-l2; *number of convolutions*: 10; *number of channels*: (32,64,128); *noising process*: linear-noise; *normalization range*: [-1,1]} (further details in sections 4.3.1 to 4.3.7).

Figure 7: 100 images generated by the baseline U-Net model

4.3.1 Batch size

We first tune the batch size of our model. We select 256.

Batchsize	32	64	128	**256**	512	1024	2048
FID	26.4	27.3	27.5	**21.4**	42.2	41.5	54.7

4.3.2 Number of epochs

Then, we tune the number of epochs. 50 seems to be the best trade-off between performance and speed of execution.

Epochs	1	5	10	25	**50**	100	200	1000
FID	197.6	78.1	31.2	17.6	**12.6**	13.3	15.1	10.9

4.3.3 Loss function

Next, we focus on the loss function. We use different loss types:

- mu-tilde-l2: $\|\hat{x}_{t-1} - \frac{1}{\sqrt{\alpha}}(x_t - \epsilon_t \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}}})\|_2^2$
- mu-tilde-l1: $\|\hat{x}_{t-1} - \frac{1}{\sqrt{\alpha}}(x_t - \epsilon_t \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}}})\|_1$
- x-prev-l2: $\|\hat{x}_{t-1} - \frac{1}{\sqrt{\alpha}}(x_t - \epsilon_t \frac{1-\alpha_t}{\sqrt{1-\alpha}})\|_2^2$
- x-prev-l1: $\|\hat{x}_{t-1} - \frac{1}{\sqrt{\alpha}}(x_t - \epsilon_t \frac{1-\alpha_t}{\sqrt{1-\alpha}})\|_1$
- epsilon-l2: $\|\hat{\epsilon_t} - \epsilon_t\|_2^2$
- epsilon-l1: $\|\hat{\epsilon_t} - \epsilon_t\|_1$

The best loss appears to be x-prev-l2.

Loss function	mu-tilde-l2	mu-tilde-l1	**x-prev-l2**	x-prev-l1	epsilon-l2	epsilon-l1
FID	24.2	20.0	**13.3**	13.3	339.3	338.5

4.3.4 Sequence of channels

The sequence of channels is also an interesting hyperparameter. It can be symbolized by a tuple that indicates the number filters used at each down/up-scaling stage of the U-Net. It basically controls the depth and the curvature of the network. Here, the best choice appears to be (32,64,128).

Channels	(8,16,32)	(16,32,64)	**(32,64,128)**	(62,128,256)
FID	30.9	13.5	**13.3**	14.9

4.3.5 Number of noising steps

Then, we focus on the number of noising steps T. The greater T the easier the denoising task but the higher the cost of the learning phase. We select $T = 15$.

Noising steps	1	5	10	**15**	20	30	50	75	100	200	500	1000
FID	117.1	14.5	16.6	**11.9**	15.1	20.8	18	27.1	35.8	118	171.4	447.9

4.3.6 Noising process

Another critical hyperparameter is the noising process since it highly conditions the capacity of our network to denoise images. Below are the processes we test:

- cosine: $x_t = \sqrt{cos\left(\frac{\frac{t}{T}+s}{1+s}\frac{\pi}{2}\right)}x_0 + \sqrt{1-cos\left(\frac{\frac{t}{T}+s}{1+s}\frac{\pi}{2}\right)}\epsilon_t$, where s is positive number close to 0
- linear-alpha-bar: $x_t = \sqrt{\frac{T-t}{T}}x_0 + \sqrt{\frac{t}{T}}\epsilon_t$
- linear-x: $x_t = \frac{T-t}{T}x_0 + \sqrt{1-\left(\frac{T-t}{T}\right)^2}\epsilon_t$
- linear-noise: $x_t = \sqrt{1-\left(\frac{t}{T}\right)^2}x_0 + \frac{t}{T}\epsilon_t$

As evidenced in the table below, we select the linear-noise process.

Noising process	cosine	linear-alpha-bar	linear-x	**linear-noise**
FID	14.7	18.6	18	**13.5**

4.3.7 Normalization range

Finally, we try several data normalization techniques. The best performance is obtained when scaling pixel values between -1 and 1.

Normalization range	[0,255]	[0,1]	**[-1,1]**
FID	220.3	74.8	**10.9**

Let's finally display some images generated by the tuned U-Net model.

Figure 8: 100 images generated by the tuned U-Net model

5 Conclusion

Unsurprisingly, basic models such as the dense neural network, already yield decent results due to the simplicity of the data set of interest. Nevertheless, by tuning the architecture and some key hyperparameters, we manage to significantly increase performance compared to the baseline approaches.

Some avenues remain to be explored, however: keeping increasing the complexity of our model (overall depth, number of channels, curvature...) – while taking care of overfitting, adding new features (attention layers, image labels as inputs...) or even testing them on brand new data sets.

References

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[SME20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.