LEARNING THE NOISE - DECODING DIFFUSION MODELS

Zeljko Antunovic (s233025), Alex Belai (s233423),
Lukas Samuel Czekalla (s233561), Nandor Takacs (s232458)

Technical University of Denmark

ABSTRACT

In image generation the model family of denoising diffusion probabilistic models (DDPM) has shown great potential in generating images from pure Gaussian noise. This report shall showcase the theoretical foundations of DDPMs and outline an implementation that is capable of achieving good results on the MNIST [1] and CIFAR-10 [2] datasets.

1. BASICS OF DDPMS

Diffusion models are a class of generative models which draw inspiration from nonequilibrium thermodynamics, learning the reverse process of a slow and systematic deconstruction of input data. The learned reverse process provides a flexible and tractable model which can generate new samples that are similar to the input data. The main building blocks of diffusion models are two Markov chains: the forward process and the reverse process. [3]

1.1. Forward process

The forward process is a Markov chain that gradually adds noise to the original image $\mathbf{x}_0$, following a distribution:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{1}$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2}$$

where t represents a time step of the noise addition and β_t is fixed by a scheduler. Following the original paper [3], we can derive a one step equation for the forward process by sampling from the distribution q using the reparametrization trick:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon} \tag{3}$$

where $\boldsymbol{\epsilon}$ is sampled from a normalized Gaussian distribution $\mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$. By continuously inserting the same reparameterization trick for the $\mathbf{x}_{t-1}$ in equation (3), and defining $\alpha_t = 1 - \beta_t$, we can derive the one-step equation for the forward process:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \tag{4}$$

where $\bar{\alpha_t} = \prod_{i=1}^{t} \alpha_i$. Thus, we can write the final form of our forward process distribution as:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{5}$$

1.2. Reverse process

The reverse process is a Markov chain that gradually removes noise from an initial input $\mathbf{x}_T$, which is just Gaussian noise, in the end revealing an image $\mathbf{x}_0$ resembling data from the distribution the model was trained on. The process is defined by a joint distribution:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \tag{6}$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{7}$$

where, in our case, the variance $\boldsymbol{\Sigma}_\theta$ is actually fixed by a scheduler $\boldsymbol{\Sigma}_\theta = \beta_t\mathbf{I}$, while the mean $\boldsymbol{\mu}_\theta$ is a free parameter. This process cannot be simplified in one step, therefore during sampling the reverse process has to run T times to sample one image, as we will discuss in section 2.4.

1.3. Learning objective

Our learning objective is to minimize the negative log-likelihood of our reverse process, $-\log p_\theta(\mathbf{x}_0)$. To do that, we have to minimize the variational lower bound of our negative log likelihood:

$$-\log p_\theta(\mathbf{x}_0) \leq -\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \tag{8}$$

By deriving the lower bound further we get to the final equation:

$$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)} \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2\right] \tag{9}$$

where $\boldsymbol{\epsilon}_\theta$ is the noise predicted by our neural network. However, experiments have shown that working with a more simplified version of a lower bound works just as well[3]:

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2\right] \tag{10}$$

1.4. Training & Inference

We have reimplemented the training and sampling algorithms from the original paper. Our training algorithm is described in Algorithm 1 of [3], which optimizes the simplified variational lower bound.

Our sampling algorithm, uses the forward process posterior mean combined with noise prediction from our model, to calculate $\mathbf{x}_{t-1}$ for each timestep t as described in Algorithm 2 in [3]. The variance in our reverse process is fixed by a scheduler as was explained in section 1.2, meaning that $\sigma_t = \beta_t$.

1.5. Classifier Guidance

Classifier guidance is a method to generate conditional samples using a DDPM. A classifier $p_c(y|x)$ is trained on the same dataset as the DDPM. The gradient of the probability of the desired class w.r.t. the classifiers input is then used in each backward step to alter the predicted noise towards the desired class according to the following expression: [4]

$$\boldsymbol{\epsilon}_\theta = \boldsymbol{\epsilon}_{\theta'} - \lambda \nabla ln(p_c(y|\mathbf{x_t})) \tag{11}$$

1.6. Classifier-free Guidance

Additionally, we have implemented a simple form of classifier free guidance as described in [5]. We jointly train a conditional and unconditional model, not using the data label y as condition to our input with probability p_{uncond}. This is a hyperparameter. Equation 13 is the key part of taking the gradient descent step as described in Algorithm 1 from [5].

$$y \leftarrow \emptyset \text{ with probability } p_{\text{uncond}} \tag{12}$$

$$\nabla_\theta \|\boldsymbol{\epsilon}_\theta(z_\lambda, y) - \boldsymbol{\epsilon}\|^2 \tag{13}$$

Instead of using the linear combination of a conditional and an unconditional model during sampling as described by [5] we simplified the sampling procedure by solely using the conditional model.

$$\tilde{\boldsymbol{\epsilon}}_t = \boldsymbol{\epsilon}_0(z_t, y) \tag{14}$$

2. IMPLEMENTATION

2.1. UNet Model

Our model is inspired by [3]'s diffusion model implementation (GitHub Repository). We have reimplemented the TensorFlow model with a couple of modifications in PyTorch.

The model follows a UNet architecture, with encoder & decoder blocks and skip connections. It has a dynamic nature regarding the number of layers, the number of ResNet blocks in each layer, the use of attention blocks and the number of channels, all set via constructor arguments. The overall architecture of the model is depicted in Figure 1.

The ResNet blocks use GroupNorm as a form of normalization. The temporal embeddings (see 2.3) are projected into the feature maps in each ResNet block. This *'reminds'* the model of the schedule in each step of the forward process. The attention blocks perform self-attention between the *'pixels'* of the feature map. The advantage of using attention blocks is that their receptive field is the whole feature map.

On top of the reimplementation of the TensorFlow model in Pytorch, our modifications to the original model include changes in the temporal embedding. We have made the use of attention blocks optional in the bottleneck, and we have created a wrapper class to provide a more simple interface for the model.

2.2. Noise Schedules

The increments in which the noise variance is changed during the forward process are defined by noise schedules. We implement a linear noise schedule, similar to the original paper, scaling noise linearly from $\beta_0 = 10^{-4}$ to $\beta_T = 0.02$.

As an interesting experiment, we decided to also implement the cosine schedule from [6] to compare the results. This schedule destroys the information in the image more gradually, giving the model a chance to learn more in each iteration. The jitter parameter s was set to 0.008, according to the original paper.

2.3. Temporal Embeddings

Sinusoidal temporal embeddings according to the implementation in [7] are projected into the ResNet blocks of the model via linear layers. The embeddings are 64-dimensional and calculated in log space to preserve numerical stability.

2.4. Classifier guidance

To implement the concept of classifier guidance for DDPMs we trained simple convolution networks on MNIST and CIFAR-10 respectively to a decent accuracy. The classifiers are just trained on unnoised images and do not incorporate the temporal embedding, which are slight deviations from the approach taken in [4]. The classifiers are then used in a dedicated reverse process as outlined in section 1.5.

2.5. Classifier-free guidance

We have implemented classfier-free guidance on MNIST by utilizing label embeddings. During training, the label embeddings were added to the temporal embeddings with probability $(1 - p_{uncond})$ following Equation 13. The combination of those embeddings is then projected into the feature maps in the ResNet blocks as in the non-guided implementation. During sampling, we simply set $p_{uncond} = 0$.

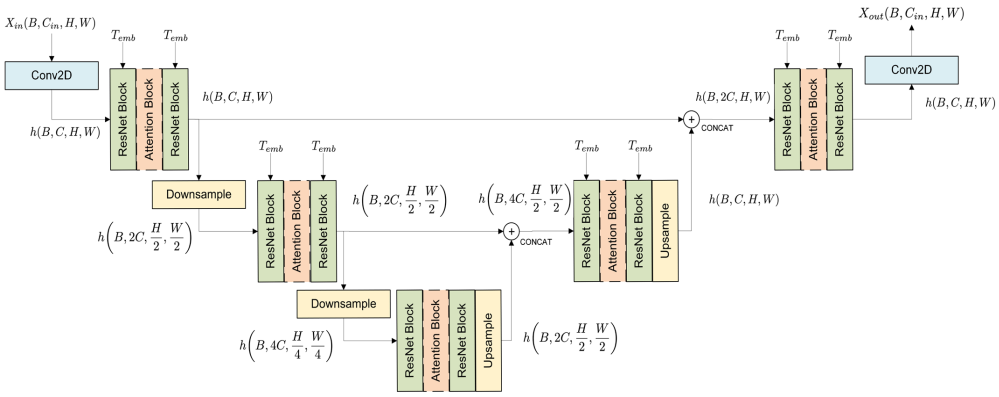

Fig. 1: UNet architecture using ResNet and Attention blocks in each encoder & decoder.

2.6. Code implementation

Our code implementation can be found in this GitHub repository. We provide a `.ipynb`-notebook that reproduces our main results.

3. EXPERIMENTS & RESULTS

We have structured our experiments in three parts. First, we evaluate the performance of our DDPM implementation on the MNIST dataset [1]. Second, we do the same for the CIFAR-10 dataset [2]. Last, we visualize our results for classifier & classifier-free guidance.

We train on both MNIST and CIFAR-10 using different ablations. We compare UNets with or without attention blocks and the usage of either a linear or a cosine noise schedule.

In order to have comparable results we utilize a common training framework:

- All input images are normalized to the range $[-1, 1]$.
- Each DDPM has $T = 1000$ diffusion steps.
- The learning rate is set to $l_r = 10^{-4}$ with *Adam* as optimizer.

Training progress has been observed by monitoring the loss on the training and validation set as well as the FID-score on the first 5 minibatches of the validation set. Figure 2 visualizes the training progress for one ablation of the MNIST dataset.

Evaluation of the DPPM results is done using 8192 samples for which the FID-score is calculated w.r.t. the whole training and test datasets. An MNIST-classifier is utilized for the activation map calculation.

3.1. MNIST

Training on MNIST has been done for 30 epochs using the previously described ablations. The FID-scores displayed in table 1 show the performance of all trained models.

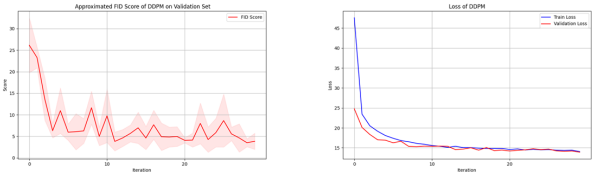

Fig. 2: Approximated FID-score, training & validation loss for DDPM using linear schedule and architecture without attention blocks.

	Linear schedule		Cosine schedule	
	no att.	att.	no att.	att.
Train-FID	2.286	2.849	2.105	1.766
Test-FID	2.803	3.490	2.724	2.385

Table 1: FID-scores for DDPMs trained on MNIST.

All ablations achieve decent results. However, it can be seen that using a linear schedule seems to work better for models without attention blocks, while for the cosine schedule this relationship is inversed. Overall, models trained with a cosine schedule outperform models trained with a linear schedule. Generated samples from the best performing model as well as a comparison of the reverse process for both schedules can be found in figures 3 and 4.

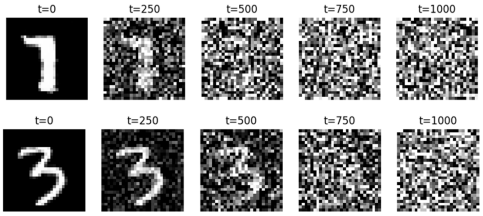

Fig. 3: Comparison of reverse process for a DDPM using linear schedule (top) and cosine schedule (bottom).

3.2. CIFAR-10

Training on CIFAR-10 has been done for 60 epochs using the previously described ablations. The FID-scores displayed in

Fig. 4: Generated MNIST samples using DDPM with cosine schedule and attention blocks.

table 2 show the performance of all trained models. Please note that the obtained FID-scores have been calculated using the activations from a MNIST-classifier, due to implementation issues. Therefore, they can just be used to compare the performance between the ablations.

	Linear schedule		Cosine schedule	
	no att.	att.	no att.	att.
Train-FID	0.114	0.165	49.393	14.565
Test-FID	0.128	0.153	49.058	14.508

Table 2: FID-scores for DDPMs trained on CIFAR-10.

Only the models trained using a linear schedule achieve decent results, where again the model using no attention blocks performs slightly better. The results obtained by using a cosine schedule are significantly worse leading to the assumption that longer training or a different model architecture could have improved performance. Sampling results for both the linear and cosine schedules are visible in figure 5.

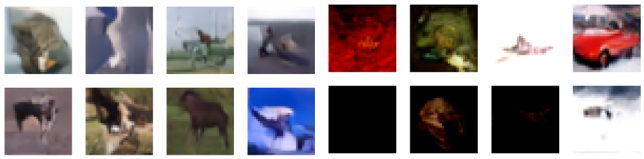

Fig. 5: Generated CIFAR-10 samples using DDPM with linear schedule & no attention (left) and with cosine schedule & attention (right).

3.3. Guidance

On top of training unconditional DDPMs on MNIST and CIFAR-10 we have also obtained results using classifier and classifier-free guidance.

3.3.1. Classifier guidance

Using the procedures defined in section 1.5 and 2.4 we managed to obtain the conditional sampling results visible in figures 6 and 7 using DDPMs with a linear noise schedule and no attention blocks with the sampling hyperparameter set to $\lambda = 200$. The generated samples are of the desired class, leading to the conclusion that the taken approach is successful.

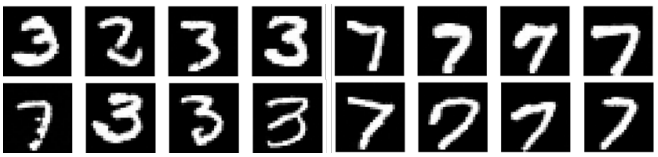

Fig. 6: Classifier guided sampling results for DDPM trained on MNIST. Sampling results for classes *3* and *7*.

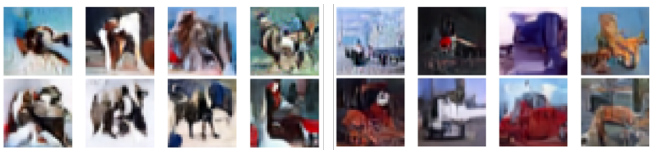

Fig. 7: Classifier guided sampling results for DDPM trained on CIFAR-10. Sampling results for classes *dog* and *truck*.

3.3.2. Classifier-free guidance

Based on 1.6 and 2.4 we managed to obtain the classifier-free conditional sampling results visible in Figure 8. We have trained for 30 epochs with $p_{uncond} = 0.01$.

Fig. 8: Classifier-free sampling results for DDPM trained on MNIST. Sampling results for class 3.

4. DISCUSSION & CONCLUSION

There are a few points we would like to improve or do further work on in the future:

- FID-score for CIFAR-10 should ideally be calculated using activations from an inception model trained on CIFAR-10 or ImageNet to obtain comparable results.
- The classifiers of classifer guidance should be trained using temporal embeddings as inputs and on noisy images. This may improve the results of the conditional sampling.
- Using a learned variance in combination with the cosine schedule could be an addition that enhances performance in comparison to the current results.
- As the model architecture is highly flexible, training DDPMs on other datasets would be a logical next step.

In general we have managed to implement DDPMs for MNIST and CIFAR-10 and achieved decent generated samples. Additionally, we have expanded our work by experimenting with a cosine noise schedule as well as classifier & classifier-free guidance.

5. REFERENCES

[1] Yann LeCun, Corinna Cortes, and CJ Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[2] Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., University of Toronto, 2009.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel, "Denoising diffusion probabilistic models," *CoRR*, vol. abs/2006.11239, 2020.

[4] Prafulla Dhariwal and Alex Nichol, "Diffusion models beat gans on image synthesis," 2021.

[5] Jonathan Ho and Tim Salimans, "Classifier-free diffusion guidance," 2022.

[6] Alex Nichol and Prafulla Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," 2023.