IMAGE SYNTHESIS USING DENOISING DIFFUSION MODELS

Olgeir Ingi Árnason - s212564

DTU Compute

ABSTRACT

Denoising diffusion probabilistic models (DDPM) are generative models that have gained traction in deep learning recently due to their performance compared to other state of the art generative models and efficient training scheme. In this project the inner workings of diffusion models were explored and an unconditional and a conditional model were implemented to create learnt samples from the MNIST dataset.

1. INTRODUCTION

The aim of diffusion models and generative models in general is to learn a good posterior distribution of the data being modeled and using the learned distribution to create samples that ideally are indistinguishable from the original data samples. This can be used for many applications such as image and sound synthesis for example. Until recently GANs have been considered one of the best architectures for generative modeling but with the excellent sample quality and efficient training of diffusion models, that might change. Diffusion models differ from traditional generative models in the way that they learn where diffusion models learn the posterior distribution by predicting incrementally added gaussian noise conditioned on the original data sample.
This project was mostly viewed as a learning opportunity and the focus is therefore on the inner workings of diffusion models in general. The code used in this project can be found under the following GitHub link:
https://github.com/olgeiringi095/Deep-Learning

2. BACKGROUND

Diffusion models are originally inspired by thermodynamics. Diffusion models define a Markov chain where gaussian noise is added to the data in each step of the chain and then learn to reverse the Markov chain by incrementally removing the added noise from the sample.

2.1. Forward Process

The aforementioned Markov chain is often called the forward process of diffusion models which incrementally adds gaussian noise to the data parameterized by

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

This leads to the importance of the parameters β_t which decide the amount of noise added in each step of the Markov chain. Two methods of noise scheduling have been proposed in the litarature the first one being the linearly increasing β_t and the cosine schedule. Another important property of the forward process is that it is possible to sample an arbitrary timestep t in the forward process without creating the samples that came before it in the chain. By defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod^t \alpha_t$ the arbitrary timestep sample becomes

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2}$$

This property allows for very efficient training of diffusion models.

2.2. Reverse Process

Being able to reverse the forward process would allow for sampling the learned distribution starting from a completely randomized gaussian input. Note that the reverse conditional probability $q(x_{t-1}|x_t, x_0)$ is only tractable when conditioned on the original data sample.

2.3. Training

Diffusion models are typically trained using variational inference which is equivalent to using the variational lower bound to optimize the log-likelihood

$$-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0)||p_\theta(x_{1:T}|x_0)) \tag{3}$$

Which leads to

$$
\begin{aligned}
L_{LB} &= E_{q(x_0:T)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}] \\
&= E_q[D_{KL}(q(x_T|x_0)||p_\theta(x_T)) \\
&+ \sum_{t=2}^{T} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) \\
&- \log p_\theta(x_0|x_1)] \\
&= L_T + L_{T-1} + \cdots + L_0
\end{aligned}
$$

Because of the forward process property (2), we can optimize random terms of L_{LB}. This means that during training, t is drawn from a uniform distribution and the model is trained to predict x_{t-1}.

According to [1] training the model with a slightly simplified training objective yielded empirically better results, the final training objective is then

$$
L_t = E_{t\in[1,T],x_0,\epsilon}[||\epsilon_t - \epsilon_\theta(\sqrt{\bar{\alpha}}x_0 + \sqrt{1-\bar{\alpha}}\epsilon_t, t)||^2] \quad (4)
$$

This simplified objective allows us to not learn the variance of the reverse process Σ_θ and instead have it fixed. In [1] they found that fixing the variance to $\beta_t\mathbf{I}$ was beneficial to sample quality when sampling from a randomized initial state.

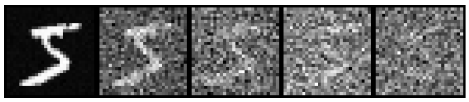
3. METHODS

As already mentioned, there are some important choices of hyperparameters that need to be defined for diffusion models such as noise scheduling parameters, the amount of steps in the forward process and all the other hyperparameters that typically are associated with deep learning models such as learning rate and activation functions etc.

3.1. Noise schedule

For this project we used the linearly increasing noise schedule as proposed in [1]. This means that the β_t parameters were linearly increasing from $1e-4$ to 0.02. The reasoning for this choice was purely due to the fact that we discovered the cosine schedule proposed in [2] too late in the process and could not implement it in time. Note that the cosine schedule has been shown to be more effective since it creates fewer samples that are completely contaminated by noise. It has also been shown in [2] that the noisier samples in the forward process contribute less to the optimization of the NLL of the model.

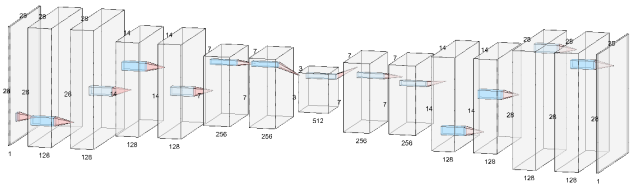
The total number of steps in the forward process was chosen to be 200 which is fairly small compared to what others have done in their work. It does however have the benefit of faster sampling due to fewer passes through the model. The number 200 was not the result of some calculations or testing since due to time constraints we did not manage to implement an objective measurement of model performance such as the NLL. This makes it difficult to justify any choice of hyperparameters and model architectures beyond what creates visually good samples.



Truncated forward process in 50 step increments (0,50,100,150,200).

3.2. Model architecture

The model architecture chosen for this project was a fairly simple convolutional neural net with depth 3 as can be seen in the figure below. While the figure is not completely accurate, it does illustrate the general dimensions of the model. The activation function used was the ReLU function and in each block in the figure below, a batch normalization was applied to help with regularization.



U-net architecture with skip connections and embeddings omitted.

Note that since the model was trained on the MNIST dataset which is low dimensional, we did not think it was needed to create a latent diffusion model where the data is first encoded to a lower dimensional space and diffused in that space.

3.3. Embeddings

As mentioned before, the noise added in each step of the forward process is linearly increasing and therefore the timestep that is being predicted needs to be embedded into the model so it knows how much noise it is trying to remove. There are many ways to embed the timestep in the model but the way we did it was to create a single hidden layer fully connected neural net that takes the timestep as input and outputs a long vector that is then rearranged and concatenated to the bottom block of the U-net.

3.4. Unconditional U-net

In this project we implemented two types of the U-net described earlier. The first one being the unconditional U-net which generates unconditional samples and does therefore not

know during training nor sampling what the label of the current data sample is. This U-net has been fully described so far and looks like the U-net in the figure above with the exception that it has skip connections between layers of the same depth in the U-net and the bottom block of the U-net has 384 channels and the first 256 of those are the result of the downward convolutions of the U-net while the next 128 channels are the timestep embedding generated by the single hidden layer fully connected neural net.

3.5. Conditional U-net

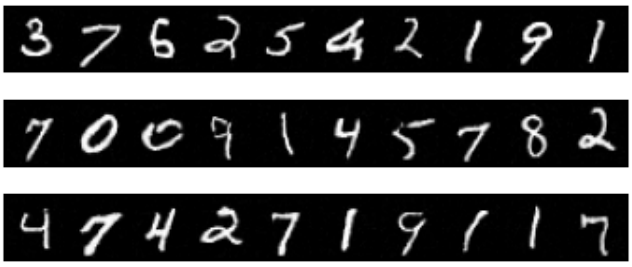
The conditional U-net has the advantage over the unconditional U-net in that it knows what the label of the data sample it is trying to predict is which allows for sampling the label that is wanted. This creates the need for embedding not only the timestep into the network but also the label of the data. This is done similarly to the timestep embedding where a single hidden layer fully connected neural net takes as input a one hot label vector and outputs a long vector that is also rearranged and concatenated to the bottom block of the U-net.

The conditional U-net is similar to the unconditional U-net with the exception that the bottom block of the conditional U-net has 512 channels instead of the 384 channels of the unconditional U-net. Similarly to the unconditional U-net, the first 256 channels in the bottom block of the conditional U-net are the result of the downward convolutions of the U-net while the next 128 channels are the result of the timestep embedding and the last 128 channels are the result of the label embedding.

4. RESULTS

As mentioned earlier, we did not manage to create an objective measurement of model performance which makes it difficult to justify any choice of hyperparameters or model architecture beyond what creates visually good samples. The only way of evaluating the model is then by visual inspection of the samples which will then be further discussed in the next chapter.

4.1. Unconditional U-net



Unconditional samples.

Eyeballing the unconditional samples we see that they look reasonable and could fool some people into thinking they were actually created by humans. It is however interesting to note that 11 of the 30 samples above are either a 1 or a 7. This could of course be a coincidence but as will be discussed in the next chapter, this did not seem to be the case.

4.2. Conditional U-net

Conditional samples.

Again, by eyeballing the conditional samples, we see that they look reasonably good and could fool many into thinking they were created by humans. In the conditional case, it is not possible to make the same argument of too many samples belonging to a certain class for obvious reasons.

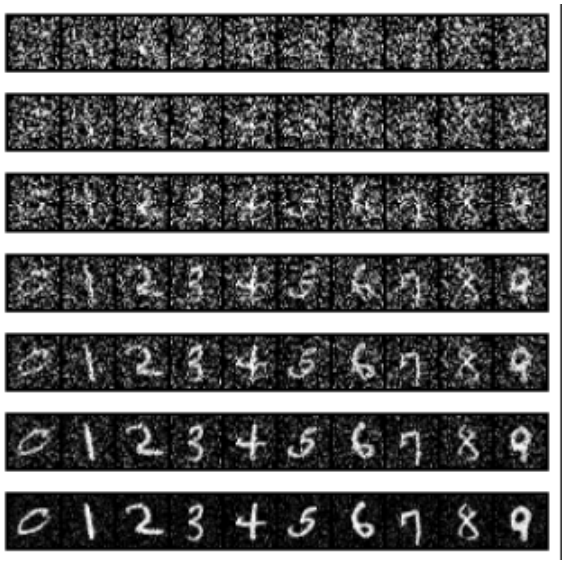
5. DISCUSSION

As mentioned before, it is interesting to see that the unconditional samples created by the model are so imbalanced classwise. According to [1], from which this project was heavily inspired, their model did not achieve a competetive log likelihood. In [2], it is mentioned that it has been found that improving the log likelihood of generative models forces the model to learn all modes of the underlying data distribution and small improvements in the NLL can lead to significant increase in sample quality and learnt feature representations. It is therefore not completely unreasonable to think that since the paper that we largely followed in this project did not achieve competetive NLL, therefore our model might also not have very good NLL. This would then lead us to thinking that if our model does not have a good NLL and it is sampling more from certain classes, then there might be some mode collapse happening in our learnt data distribution. This is however completely speculatory at this point and would require the NLL which as mentioned before, was not implemented.

6. FUTURE WORK

Given more time, we would have liked to implement an objective measurement of model performance such as the NLL. This would then allow for justifying different choices of hyperparameters, model architectures and noise schedules. It

would also be interesting to see if the improvement in the NLL would lead to a more balanced class sampling in the unconditional model.



Truncated reverse process.

7. REFERENCES

[1], [2], [3] heavily inspired the choices and arguments made in this project.

8. REFERENCES

[1] Pieter Abbeel Jonathan Ho, Ajay Jain, "Denoising diffusion probabilistic models," 2020.

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

[3] Lilian Weng, "What are diffusion models?," *lilianweng.github.io*, Jul 2021.