BMI / CS 771 Homework Assignment 4
DDPM : Denoising Diffusion Probabilistic Models

Siddharth Baskar sbaskar2@wisc.edu
Karan Vikyath Veeranna Rupashree veerannarupa@wisc.edu
Anudeep Kumar kumar256@wisc.edu

1 Introduction

This assignment was focused on learning object detection by implementing DDPM - Denoising Diffusion Probabilistic Models. Diffusion models are a class of generative models inspired by an idea in Non-Equilibrium Statistical Physics, which states:

"We can gradually convert one distribution into another using a Markov chain"

Diffusion generative models are composed of two opposite processes i.e., Forward & Reverse Diffusion Process.

We train the model on MNIST dataset after writing the forward and reverse function along with the loss function.

2 Dataset

The dataset used for this is the classic MNIST dataset.The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.The MNIST database contains 60,000 training images and 10,000 testing images.

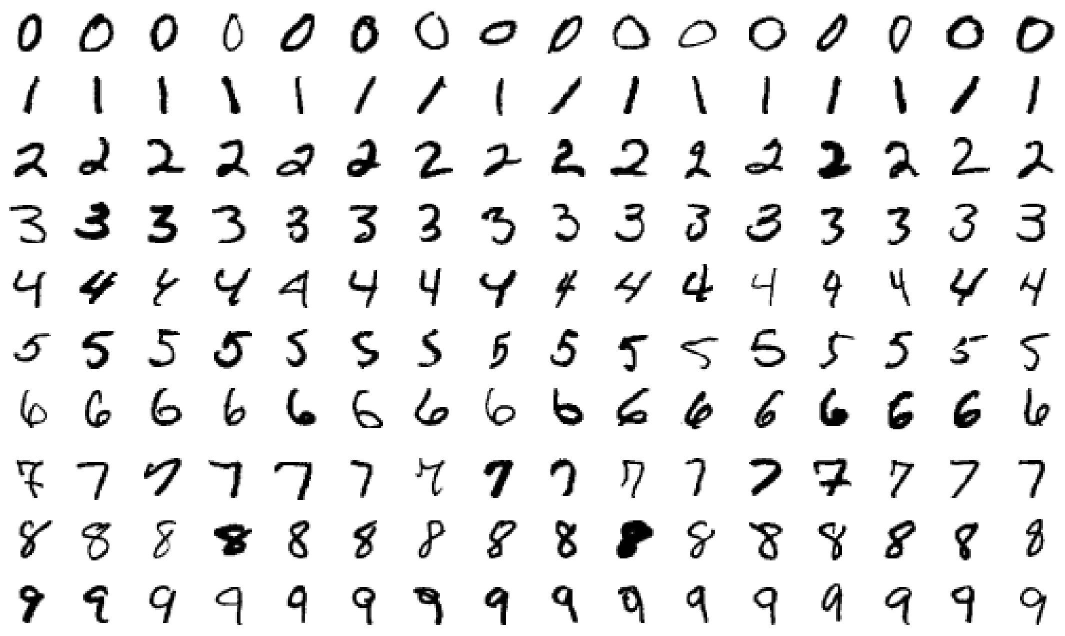


Figure 1: Sample from Dataset

3 Understanding DDPMs

3.1 Model Design

a. What is the latent space of DDPMs? How can you draw samples from a DDPM?

As mentioned above DDPM consists of Forward and Reverse diffusion process. *Latent space refers to this space in which diffusion steps takes place*. Diffusion steps refer to adding noise in steps to the current datapoint to transform it to a sample from the target distribution. In our case adding noise at each step to our initial image till it becomes pure noise. This is the forward diffusion process. At the end of the forward process, the images become entirely unrecognizable. The complex data distribution is wholly transformed into a (chosen) simple distribution. Each image gets mapped to a space outside the data subspace.

In the reverse diffusion process we reverse the corruption done on image in the forward process.The benefit of starting from a simple space is that we know how to sample a point from this simple distribution.The problem is that we can take infinite paths starting from a point in this "simple" space, but only a fraction of them will take us to the "data" subspace. In DDPM , they use a deep-learning model at each step to predict the PDF parameters of the forward process. Once we train the model, we can start from any point in the simple space and use the model to iteratively take steps to lead us back to the data subspace. *This is how we can draw samples from a DDPM.* [2]

b. How does our helper code inject time and condition (image labels) into the denoising function (UNet)?

The huggingfaces implementation was the great starting point. It had the time injection function in Unet but for label embedding it was not present.But the helper code has label embedding included in it.In the unet.py line 36 and 44 the declaration of the time embedding and label embedding is present.[3][2] Here's a breakdown of the code:

- **Time Embedding (`self.time_embd`):**
  - The network begins with an embedding for time steps using sinusoidal positional encoding (`SinusoidalPE`). This technique is commonly used in sequence-to-sequence models to provide the model with information about the relative positions of elements in the input sequence.
  - The output of the sinusoidal positional encoding is then passed through a linear layer (`nn.Linear(dim, time_dim)`) with a subsequent SiLU (Sigmoid Linear Unit) activation function (`nn.SiLU()`).
  - Another linear layer is applied (`nn.Linear(time_dim, time_dim)`), which is followed by the final SiLU activation.
- **Label Embedding (`self.label_embd`):**
  - The network also includes an embedding for labels, which is similar to the time embedding.
  - It uses a custom `LabelEmbedding` module, which is defined in blocks.py. This module uses nn.Embedding to create the embedding
  - The output of the label embedding is processed similarly to the time embedding, with a linear layer (`nn.Linear(dim, context_dim)`), a SiLU activation, and another linear layer (`nn.Linear(context_dim, context_dim)`).

The purpose of these embeddings is to capture temporal and contextual information related to time steps and labels.

Now using these in the forward pass the embedding are used. The input image x undergoes an initial convolution operation. The time step time is passed through the time embedding module , producing an embedding tensor t. The label is passed through the label embedding module. The result is unsqueezed along the first dimension, to match dimensions with the later concatenation operation. The resulting tensor is denoted as c.

The processed image, time embedding (t), and label embedding (c) are sequentially passed through the encoder, which consists of a series of residual blocks , transformers, and downsample operations . The intermediate results are stored in a list h.

The processed image from the encoder is further processed by a middle block , followed by attention, and another middle block. This section captures the middle part of the network.

The decoder is essentially the reverse of the encoder. It consists of a series of residual blocks , transformers , and upsample operations . The decoder takes the concatenated tensor of the current processed image and the tensor popped from the list h.

The final convolution is applied to the output of the decoder to produce the final result.

4 Model Inference

4.1 Implementing Forward Diffusion Process

For this exercise we were required to implement the Forward diffusion process based on equation 4 on the paper [1]

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

With the help of reference code and understanding the equation we calculate return x_t as :
x_t `= sqrt_alphas_cumprod_t * x_start + sqrt_one_minus_alphas_cumprod_t * noise`

Where `sqrt_alphas_cumprod_t` and `sqrt_one_minus_alphas_cumprod_t` are calculated in function `compute_alpha_vars()`

4.2 Implementing Reverse Denoising Process

As mentioned in `p_sample` and `p_sample_loop` are responsible for the reverse denoising process. As per equation 11 in the paper [1]

$$\mu_\theta(x_t, t) = \tilde{\mu}_t(x_t, \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\theta(x_t))) = \frac{1}{\sqrt{\alpha_t}}(x_t - \beta_t\sqrt{1-\bar{\alpha}_t}\theta(x_t, t))$$

With inspiration from the reference and taking care of the labels we implement the function as :
`model_mean = sqrt_recip_alphas_t * (x - betas_t * self.model(x,label,t) / sqrt_one_minus_alphas_cumprod_t)`
Where we incorporate label in the function call `self.model(x,label,t)`

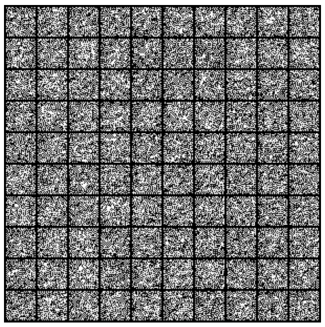
4.3 Implementing DDPM Loss

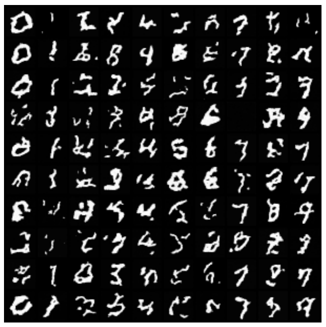
For `compute_loss` we refer Algorithm 1 [1], noise is calculated and forward diffusion is done. We obtain the predicted noise by passing it through the Unet model and then we calculate the MSE loss.

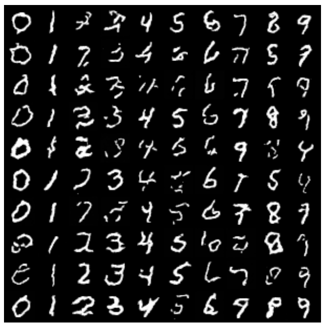
```
x_noisy = self.q_sample(x_start=x_start, t=t, noise=noise)
predicted_noise = self.model(x_noisy,label, t)
loss = F.mse_loss(noise, predicted_noise)
```

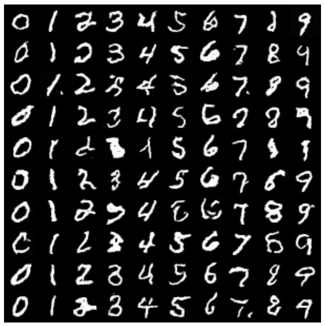
4.4 Experimenting with MNIST dataset

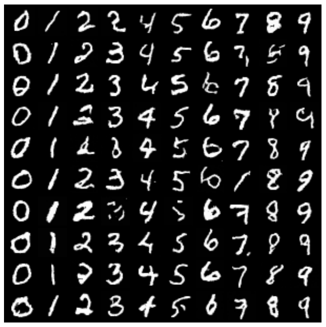
We ran the code for 30 epochs as mentioned and it took 1.71 hours. The digits started at 10th epoch and by 15th epoch we can see good results and at the end of training we can see perfect results.

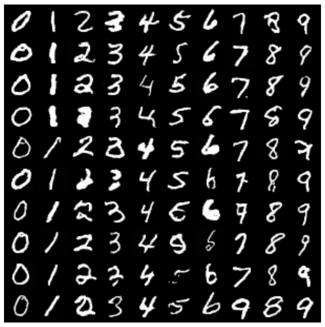

(a) Sample at 0


(b) Sample at 5


(c) Sample at 10


(d) Sample at 15


(e) Sample at 20


(f) Sample at 30

The training curve can be seen below

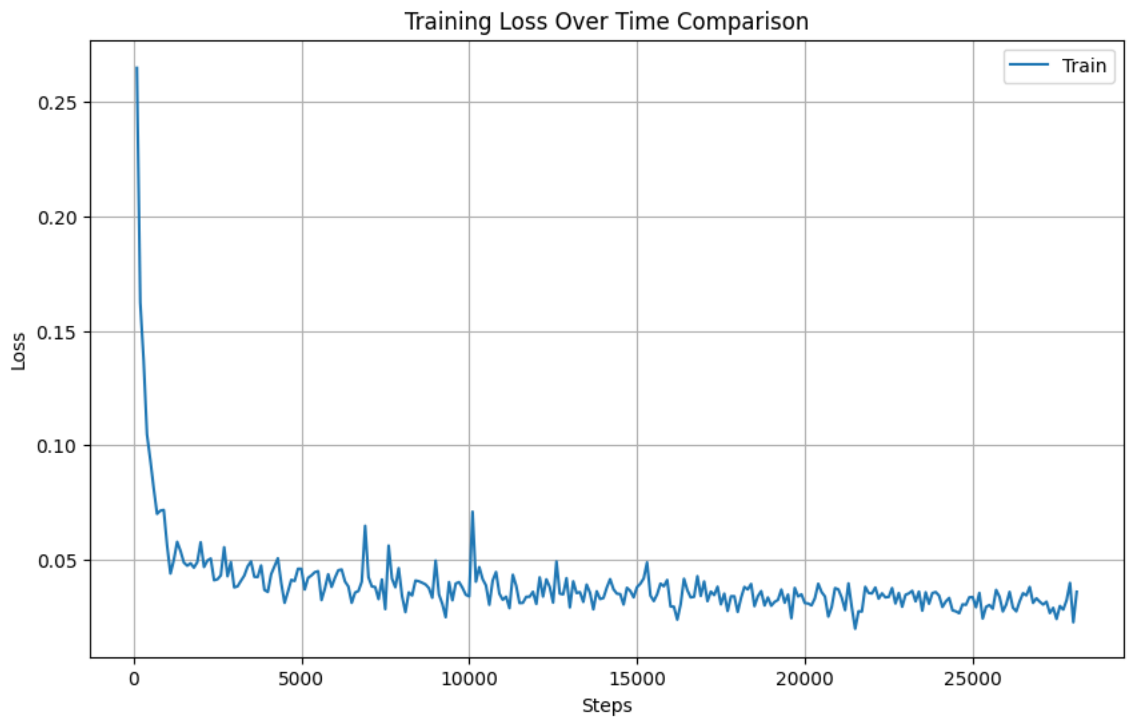


Figure 3: Training Curves

5 Results and Conclusion

Hence we can observe that training the model with MNIST we were able to generate good results in 30 epochs. The way the authors used non equilibrium thermodynamics to make model learn to generate images is very interesting. The usage of DDPM are increasing and its being used in various fields as well. One of the interesting approach inspired from DDPM is DDM^2 [4]which denoises MRI using Diffusion Models.

6 Teammate Contributions

Anudeep Kumar	**Karan Vikyath Veeranna Rupashree**	**Siddharth Baskar**
Understanding DDMP	Understanding DDPM	Understanding DDPM
Implementing Forward Diffusion Process	Implementing Reverse Denoising Process	Implementing DDPM Loss
Experimenting with MNIST dataset	Experimenting with MNIST dataset	Experimenting with MNIST

References

[1] Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." Advances in neural information processing systems 33 (2020): 6840-6851.

[2] DDPM Article LearnOpenCV

[3] Hugging Face Reference

[4] Xiang, Tiange, Mahmut Yurt, Ali B. Syed, Kawin Setsompop, and Akshay Chaudhari. "DDM ^{2}: Self-Supervised Diffusion MRI Denoising with Generative Diffusion Models." arXiv preprint arXiv:2302.03018 (2023).