DDPM VS VAE On Image Synthesis

Maha Mapara
maha.mapara@tufts.edu

Tina Wu
yiyun.wu@tufts.edu

Abstract

Purpose: The goal of our project is to understand how Denoising Diffusion Probabilistic Models (DDPM) Ho et al. (2020) work and explore its advantages and limitations by comparing its performance against VAE's Kingma and Welling (2013) on image generation tasks across a variety of datasets.
Hypothesis: Given the same training input, DDPM will outperform VAE on image synthesis, because DDPM not only learns the noises through the network, but its network, U-net Ronneberger et al. (2015), also resembles very much like an autoencoder, which allows the model to keep the embeddings with only the important features as well. Performance will be evaluated via model accuracy and sampled image qualities.
Evaluation plan: We will use MNIST Deng (2012) and CIFAR-10 Krizhevsky (2009): the first dataset produces visually analyzable results, which are especially useful when implementing a new algorithm, and the latter one assesses the diversity of output samples. We will measure model accuracy using variational bound and quantify image qualities with FID Heusel et al. (2017).

A Changelog

- Moved to using GPU to reduce training time.
- Modified datasets tested to MNIST and CIFAR10 only.
- Adopted pre-defined U-net from github repository. Rogge and Rasul (2022)
- Removed runtime costs from properties to measure; we realized that these are dependent on many factors outside of algorithms, such as computing power.
- Adjusted timeline to accommodate implementation period.

B Challenges

Architecture One challenge in implementing DDPM is building the neural network. Based on the DDPM paper Ho et al. (2020), we decided to adopt a predefined U-net from Rogge and Rasul (2022), since it is not the main focus of the project. We modified it, so that the same architecture can be applied to train datasets with different numbers of channels, namely 1 and 3.

Parallelization Based on the two algorithms defined in the paper Ho et al. (2020), training and sampling represent a full forward and reverse process, respectively. Considering the number of hours it takes to train even with minimum convergence, sampling images after a full, converged, forward training process poses an issue: hours are wasted if the sampled images do not resemble any inputs. Instead, we combined the two algorithms to allow parallelization between training and sampling. Apart from printing loss, at every 10^{th} epoch during training, the model samples images for visual analysis, to ensure that the code is bug-free and the training is effective.

C Timeline

- Calculate FID scores for sampled DDPM images - Tina (by 11/30)
- Finish VAE for CIFAR10 and calculate FID - Tina & Maha (by 12/10)
- Analyze qualitative results and prepare final presentation - Tina & Maha (by 12/11)
- Write up the final report - Tina & Maha (by 12/19)

1 Goals

The goal of the project is to learn a relatively new technique in the literature, DDPM Ho et al. (2020), and explore especially its advantages and weaknesses. To achieve this goal, we will implement a DDPM model and compare it against an existing generative model, VAE Kingma and Welling (2013), on image generation, a task that takes in an image dataset as input and produces sampled images as output. Performances of both models will be evaluated via model accuracy, or ELBO, and sampled image qualities, quantified by FID score Heusel et al. (2017).

We hypothesize that DDPM will outperform VAE slightly on image synthesis, because apart from its special noise diffusion process, DDPM with U-net also employs the key component of VAE: exploring a low dimensional embedding of important features. Although the original diffusion model Sohl-Dickstein et al. (2015) does not alter the dimensions of the input, which is typically not true with VAE, the authors of Ho et al. (2020) proposed using U-net Ronneberger et al. (2015) as the neural network to train the parameters, which, same as under-complete autoencoders, maps input data to lower dimensions for keeping the key features and up-scales it back to the original dimension for output Weng (2018).

This is an interesting project because diffusion models are an exciting area of research in generative models, especially for image synthesis tasks; many popular architectures use diffusion models for this purpose, like DALLE-2 and Imagen. This is a good opportunity to gain in-depth exposure about the architecture and modeling approach used for diffusion models. In particular, we were inspired by the paper 'Diffusion Models Beat GAN on Image Synthesis' Dhariwal and Nichol (2021), where the authors compare DDPM performance with GANs on image synthesis using ImageNet and LSUN image data. Instead of comparing DDPM to GAN, we will compare diffusion models with VAEs. This is because:

1. We have not seen a comparison between VAE and diffusion models in the literature on our choice of data sets.
2. The models have some similarities:
    - Forward process that turns data into latent representations
    - Reverse process that involves sampling some latent variable
    - Training objective which is a lower bound on data likelihood
3. It will be interesting to see how different the model performances are, and evaluate the expected underperformance of VAEs based on our understanding of the key differences between DDPMs and VAEs.

2 Methods

2.1 DDPM

The key probabilistic model we are focusing on is the DDPM. There are two main steps involved: the forward process and the reverse process. The q distribution is for the forward process and the p_θ distribution is for the reverse process. The key random variable is x. The distributions and random variable are defined below.

There are many variations to the original diffusion model, and we decided to implement mainly the algorithms described in Ho et al. (2020). Diffusion models are composed of two Markov chains, namely forward training process and reverse sampling process Sohl-Dickstein et al. (2015)

2.1.1 Forward Process

In the forward step, the input data, x_0, is converted into a standard Gaussian, x_T, by adding a Gaussian noise at each intermediate step, x_t, in a Markov chain. Therefore, the estimated posterior can be defined as a joint distribution of all intermediate steps, given x_0 Sohl-Dickstein et al. (2015):

$$q(x_{1:t}|x_0) = \sum_{t=1}^{T} q(x_t|x_{t-1})$$

Each intermediate distribution is defined as a Gaussian distribution

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

where β_t is the variance schedule or noise added at each step. To simplify it, instead of conditioning on x_{t-1} and calculate every single step in the chain, we will condition $q(x_t)$ on the input data, x_0, and derive $\overline{\alpha}_t$ from β using the reparameterization trick:

$$q(x_t|x_0) = N(\sqrt{\overline{\alpha}_t} x_0, (1-\overline{\alpha}_t)I), \tag{1}$$

with $\overline{\alpha}_t = \sum_{s=1}^{t} 1 - \beta_s$. This allows us to compute $q(x_t)$ at any arbitrary t Ho et al. (2020). There are a number of choices for the noise schedule, β, such as linear, cosine, quadratic, constant, etc.

2.1.2 Model

We adopt U-net from Rogge and Rasul (2022) as the backbone structure to our neural network. U-net, itself, is composed of an ‘encoder’ and a ‘decoder’, same as autoencoders. Similar to CNN, the input is down-scaled in dimensions and up-scaled in channels during encoding; reducing the dimensionality of input allows the model to interpret only the features that matter the most. The process is reversed in the decoder, so that the output has the same dimensions as the input Ronneberger et al. (2015).

We train the model with pytorch library and Adam optimizer Paszke et al. (2019). The model, $\epsilon_\theta(x_t, t)$, is fed with two parameters, where t is uniformly chosen from $\mathcal{U}(1, T)$, and x_t is a batch of noised images in the shape [C, H, W]; the output is a noise of the same dimension. The loss function is evaluated at variational bound, or ELBO. For DDPM, it is simplified to the l2 loss of the difference between a standard Gaussian, ϵ, and the output noise of the model, $\epsilon_\theta(x_t, t)$, Ho et al. (2020):

$$||\epsilon - \epsilon_\theta(x_t, t)||^2,$$

where x_t can be sampled from equation 1. From the loss function, we can see that the objective of the network is to learn a noise, or Gaussian transition, that makes the input data, x_0, approaching standard Gaussian, $N(0, I)$, after applying such noise.

2.1.3 Sampling

The sampling process, on the other hand, is a full Markov chain, starting with the prior being a standard Gaussian distribution, $x_T \sim N(0, I)$, and our goal is to sample x_0 at the end of the process. Same as the forward process, the reverse process can be defined as a joint distribution of intermediate distributions in the Markov chain:

$$p(x_{0:T}) = p(x_T) \sum_{t=1}^{T} p(x_{t-1}|x_t)$$

Intermediate image, x_{t-1}, can be sampled from a Gaussian distribution with

$$\mu = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}} \epsilon_\theta(x_t, t))$$

and

$$\sigma = \sqrt{\frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t} \beta_t I},$$

where ϵ_θ is the Gaussian transition learned by the model during training Ho et al. (2020).

Table 1: Hyperparameters in DDPM

Hyperparameter	Justification
Timesteps=300	Fixed to a large value for simplicity
num_epochs=200	Fixed to a large value for simplicity
batch_size	Fixed for simplicity
variance schedule β	Fit by loop though different β
Learning rate	Fit by looping through a list of values linearly in $[0.0001, 0.001]$

Table 2: Hyperparameters in VAE

Hyperparameter	Justification
hidden_layer_size	Fixed to a large value for simplicity
Variance	Fixed to 0.2 for simplicity
batch_size	Fixed for simplicity
num_epochs	Fixed to a large value rather than fitting or until converged
num_mc_samples	Fixed to a large value rather than fitting
Learning rate	Fit by looping through a list of values linearly in $[0.0001, 0.001]$

2.2 VAE

Same as the HW 4, we will use the pytorch neural network with Adam optimizer and no regularization to train VAE. Specifically, we will use under-complete VAE, which maps the input, (28x28), to (2x2) encoded space with 512 hidden layers.

3 Experiments

3.1 Data Description

Our two data sets are MNIST and CIFAR-10.

MNIST Deng (2012) MNIST is an image dataset, composed of handwritten digits of 0-9, with 60,000 examples in the training split. It is 28x28 pixels in grayscale, each pixel ranging 0 to 255, with digits centered in the image. We will heavily use MNIST for implementation, because MNIST is clear and easy to interpret. Among all image datasets, MNIST is very simple (gray-scale) and takes a shorter time to train, which is appropriate for implementing the algorithm and working through coding issues. Also, MNIST can comprehensively be used to analyze image quality. Therefore, we can compare the results of the diffusion model on MNIST against VAE on MNIST easily.

CIFAR10 Krizhevsky (2009) CIFAR10 is composed of 10 different classes of natural images, including cats, dogs, cars, etc. It is a bit more complicated than MNIST, considering that it has 3 channels instead of 1, and each image is 32x32 pixels. We will rely more on CIFAR10 to evaluate the true performance of each method, mainly because it resembles true real-world data more than MNIST.

3.2 Hyperparameters

DDPM See Table 1.

VAE See Table 2.

3.3 Experimental protocol

We will only be using the ‘train’ split of the dataset to train the model; the dataloader is shuffled. There is no test or evaluation set, because the sampling process for DDPM starts with standard Gaussian, and no other images are involved. For VAE, same operations on the training set, but for the

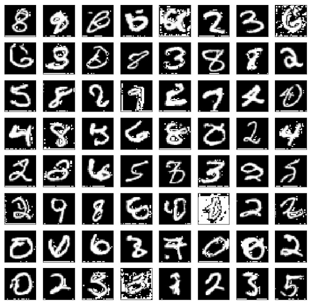
Figure 1: Sampled images from MNIST.

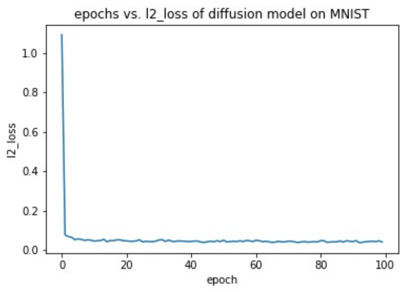

Figure 2: MNIST loss.

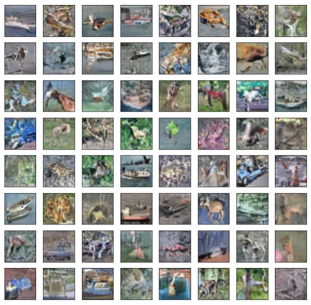
Figure 3: Sampled images from CIFAR10.

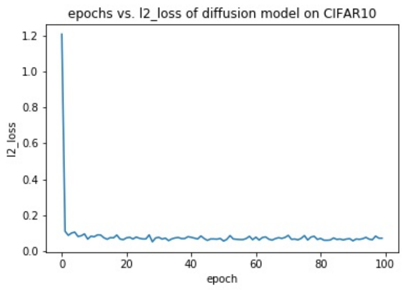

Figure 4: CIFAR10 loss.

testing set, we will get the batches from the other split with shuffle=false. We will run at most 200 epochs or until convergence, that is, the change in loss value is smaller than a defined threshold. The Markov chain will run for 300 iterations for DDPM, and 50 MC samples for VAE. Similarly, convergence is assessed by changes in loss values.

We will be using log likelihood to measure the accuray of each model. This is also our loss function, which tells how good the model fits the dataset and whether there are signs of overfitting or underfitting. Secondly, we will be using FID scores Heusel et al. (2017), which meausure how good the qualities of the generated samples are, by calculating the difference between feature vectors of real images and the feature vectors of generated images.The lower the FID score, the fewer distortions in the image and better the image quality. Therefore, the model that produces lower FID will have a better performance. If both variational bound and FID for DDPM are lower than VAE, our hypothesis is confirmed, showing that DDPM outperforms VAE in both accuracy and have better qualities for sampled images. If both metrics in DDPM are higher than those in VAE, our hypothesis is disproved; otherwise, it is a ’tie’, or we will need to perform further measurements, if possible, to evaluate the performance.

3.4 Figures

MNIST See Figure 1 & 2.

CIFAR10 See Figure 3 & 4.

3.5 Implementation

The U-net architecture (Ronneberger et al. (2015)) has been used for the neural network architecture. Model is trained using the pytorch library and an Adam optimizer (Paszke et al. (2019)). Other libraries used were numpy (Harris et al. (2020)), math (Van Rossum (2020)), and matplotlib (Hunter (2007)).

References

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rogge, N. and Rasul, K. (2022). The annotated diffusion model.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585.

Van Rossum, G. (2020). *The Python Library Reference, release 3.8.2*. Python Software Foundation.

Weng, L. (2018). From autoencoder to beta-vae. *lilianweng.github.io*.