NTNU
Kunnskap for en bedre verden

Department of Computer Science

DT8122 - Probabilistic Artificial Intelligence

Project Assignment
Diffusion and BFN

Author:
Grzegorz Malczyk

14.09.2024

1 Introduction

The goal of the project assignment is to get the hands-on experience with Diffusion and Bayesian Flow Networks. Results from training and evaluating the models on MNIST are presented in this report. It also covers discussion of results and methods as specified in the specific tasks from the assignment.

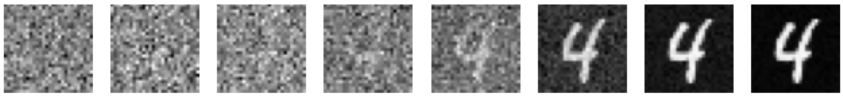
2 Task 1

The task is to implement Diffusion model. The presented implementation of the diffusion is based on the Ho, Jain et al. 2020; Ho and Salimans 2022. The implementation is trained on the MNIST dataset, readily available in most deep learning frameworks. The code can be found in the following repository www.github.com/grzemal/diffusion_model. The code is developed based on the minDiffusion, which provides a self-contained, minimalistic implementation of diffusion models using Pytorch with the modification and adjustments for better performance as discussed in Section 2.2.

2.1 Plots

- Plot of nine generated digits after training the diffusion model

- Plot of one generated digits across 8 diffusion steps, starting with an initial sample from the prior and ending with the "noise-less" generated image.



2.2 Discussion

This section briefly covers the network architecture, what it models, and how the diffusion step is incorporated. The network models the reverse diffusion process. Specifically, it's trained to predict the noise added to an image at a particular time step. This method allows the model to gradually denoise an image, starting from pure noise and ending with a clean sample, as shown in the above figures.

2.2.1 Architecture

The network uses a U-Net architecture, which is common in image-to-image tasks. The U-Net consists of:

1. A down-sampling path (encoder):
   (a) Initial convolutional layer

(b) Two down-sampling blocks – *UnetDown*

2. A bottleneck:
   (a) Average pooling layer
3. An up-sampling path (decoder):
   (a) Three up-sampling blocks – *UnetUp*
   (b) Final convolutional layers

DDPM Implementation: The DDPM class encapsulates the entire diffusion process. It uses pre-computed schedules for the forward and reverse processes followed by Ho, Jain et al. 2020. The forward method adds noise to images and trains the model to predict this noise. The sampling method generates new images by iteratively denoising random noise.

2.2.2 Encoding/incorporating the diffusion step t

The diffusion step t is crucial for the model to know how much denoising to perform. It's incorporated in several ways:

- Time embedding: Two embedding layers (*timeembed1* and *timeembed2*) convert the scalar time-step into feature maps
- Injection into the network: These time embedding layers are added to feature maps in the up-sampling path. This addition allows the time information to influence the denoising process at different scales.
- Input to the network: The time-step is also provided as an input to the forward pass, normalized by the total number of steps.

By incorporating the time-step in these ways, the network can adapt its behavior based on how far along the diffusion process is, allowing it to perform the appropriate denoising level at each step. This architecture and time-step encoding enable the model to learn a flexible denoising process that can generate high-quality samples when run in reverse, starting from pure noise. The network is trained to minimize the mean squared error between the predicted noise and the true noise at each time step. It is trained for 40 epochs, and the model and generated samples are saved at the end of training. The network is trained using the Adam optimizer with a learning rate of 1e-4 with batch size of 256 and trained on a 16 GB GPU, which allows for faster training within approximately 1.5 hours.

2.2.3 Challenges

During the development phase, it was quite challenging to obtain a network architecture which learns in a reasonable amount of time and delivers good results. First, I started with a small number of epochs and a smaller batch size to decrease the time while sacrificing the network outcome. After finalising the network architecture and fine-tuning the hyperparameters, I ended up with 40 epochs and a batch size of 256, which enables not too long a training time as well as provides good results based on the generated samples.

3 Task 2

The goal of this task is to read and discuss the Bayesian Flow Networks(BFN) paper Graves et al. 2023. Additionally, one is asked to train the model on the MNIST dataset to generate the same plots in Task 1.

3.1 Plots

- Plot of nine generated digits after training, sample from Output distribution

Figure 1: Caption

- Plot of one generated digits across 8 diffusion steps, starting with an initial sample from the prior and ending with the "noise-less" generated image.

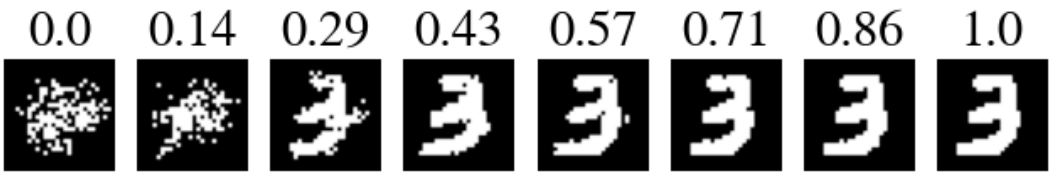


Figure 2: Caption

3.2 Discussion

In this section, we discuss the similarities and differences between Diffusion models and Bayesian Flow Networks (BFNs), particularly focusing on their approaches to learning discrete distributions.

3.2.1 Similarities

- Iterative process: Both methods use an iterative process to generate data, starting from a simple distribution and progressively refining it towards the target distribution.
- Noise reduction: Both methods can be viewed as gradually reducing noise in the data representation, though they approach this differently.
- Continuous-time formulation: Both can be formulated in continuous time, allowing for flexible choice of the number of steps during inference.
- Loss function: Both use loss functions derived from variational bounds, which are related to data compression.
- Neural network backbone: Both rely on neural networks to model the complex relationships in the data.

3.2.2 Differences

- Nature of the process: Diffusion models typically start with pure noise and gradually denoise it. BFNs start with a simple prior distribution and refine its parameters using Bayesian updates.
- Input to the neural network: In diffusion models, the network typically operates on noisy versions of the data. In BFNs, the network operates on the parameters of a distribution.
- Forward process: Diffusion models require defining and inverting a forward (noising) process. BFNs don't need a forward process, which simplifies their conceptual framework.

- Parameter space: Diffusion models often work directly in the data space. Bayesian Flow Networks work in the parameter space of distributions, which is always continuous.
- Flexibility: BFNs may be more easily adaptable to different data types due to their parameter-based approach.

3.2.3 Comparison for Discrete Distributions

When learning discrete distributions, BFNs offer several potential advantages. First, BFNs naturally work with continuous representations (distribution parameters) even for discrete data. This allows for gradient-based sample guidance and few-step generation, which can be challenging for traditional discrete diffusion models. Next, BFNs don't require carefully defined transition matrices or complex embedding schemes that some discrete diffusion models use. Additionally, Bayesian Flow Networks directly optimize the negative log-likelihood of discrete data, unlike some continuous diffusion methods for discrete data that may require simplified loss functions or auxiliary terms. The inputs to the network in BFNs (probabilities on the simplex) are natively differentiable, avoiding the need for relaxations or continuous embeddings. It is worth it to mention that the paper hypothesizes that BFNs might lead to faster learning on large datasets where the model under-fits, due to less noisy network inputs compared to diffusion models. The authors report that BFNs outperform all known discrete diffusion models on the text8 character-level language modelling task. However, it's important to note that discrete diffusion models are an active area of research, and new methods are continually being developed. The relative performance of BFNs and discrete diffusion models may vary depending on the specific task and implementation details. In conclusion, while both methods can be applied to discrete distributions, BFNs offer a more natural framework for handling discrete data due to their parameter-based approach. This leads to potential advantages in terms of simplicity, optimization, and performance. However, more extensive comparisons across a wider range of tasks would be needed to definitively establish the superiority of one method over the other. Therefore, a fair comparison cannot be made based on my work in this project since I focused mostly on the implementation of the diffusion model. Finally, the script for the BFN on the MNIST dataset was given to us. But after interacting with it (training + plotting), it turns to be more complicated in the network architecture and delivers similar results for the MNIST dataset. Thus, it is hard to conclude which of the models, diffusion or BFN, is better in general.

Bibliography

Graves, Alex et al. (2023). 'Bayesian flow networks'. In: *arXiv preprint arXiv:2308.07037*.

Ho, Jonathan, Ajay Jain and Pieter Abbeel (2020). 'Denoising diffusion probabilistic models'. In: *Advances in neural information processing systems* 33, pp. 6840–6851.

Ho, Jonathan and Tim Salimans (2022). 'Classifier-free diffusion guidance'. In: *arXiv preprint arXiv:2207.12598*.