ProbAI Project assignment

Martin Lund Trinhammer

August 12, 2024

1 Diffusion

1.1 Choice of architecture

This exercise is an implementation of a diffusion model on the classic MNIST dataset [4]. For my implementation, I will rely on classifier-free guidance (cfg) as described in [5]. This class of models lies within conditional diffusion models as opposed to unconditional ones. The main objective conditional models achieves is to gain some explicit control over the denoising process, allowing developers to guide the generative process towards some specific quality. This is what has made modern text-to-image generation models, such as Stable Diffusion, so popular and relevant in many different use-cases [6]. Cfg is an improvement to classifier guided (cg) diffusion models [2], which suffers a number of setbacks. In cg, a separate classifier is trained to guide the diffusion process towards the desired label. This is cumbersome to implement as one cannot easily use a pre-trained one, given that random, gradual noise is added. One suggested architecture, which also outperforms cg, is cfg which does not rely on training a separate classifier.

1.2 Implementation details

I have chosen to rely on cfg in this exercise given its state-of-the-art performance, excellent architecture and ease of implementation. I am using the algorithm outlined in Figure 1

Algorithm 1 Joint training a diffusion model with classifier-free guidance

Require: p_{uncond}: probability of unconditional training

1: **repeat**
2: $(\mathbf{x}, \mathbf{c}) \sim p(\mathbf{x}, \mathbf{c})$ ▷ Sample data with conditioning from the dataset
3: $\mathbf{c} \leftarrow \varnothing$ with probability p_{uncond} ▷ Randomly discard conditioning to train unconditionally
4: $\lambda \sim p(\lambda)$ ▷ Sample log SNR value
5: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: $\mathbf{z}_\lambda = \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon}$ ▷ Corrupt data to the sampled log SNR value
7: Take gradient step on $\nabla_\theta \left\| \boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon} \right\|^2$ ▷ Optimization of denoising model
8: **until** converged

Figure 1: Algorithm 1 from [5]

While Figure 1 is the core of my implementation using a UNet backbone, I would like to stress the following set of modifications or custom implementation choices. In developing my implementation, I drew inspiration from this codebase. I chose to one-hot encode the labels of the MNIST dataset to facilitate their use in conditioning aspects of my model. For managing noise during training, I opted for a linear noise scheduler, following the approach outlined in [4]. While a cosine schedule can sometimes yield slightly better performance, I found the linear scheduler to be sufficiently effective for this particular use case.

One thing I struggled a bit with, was how to embed the class information into the timesteps. I solved this with a simple Multi Layer Perception, using a GELU activation function as seen here

```
self.label_emb = nn.Sequential(
    nn.Linear(num_classes, time_dim),
```

```
    nn.GELU(),
    nn.Linear(time_dim, time_dim),
)
```

I then also added the positional encoding to the label embedding in the forward pass of my UNet model.

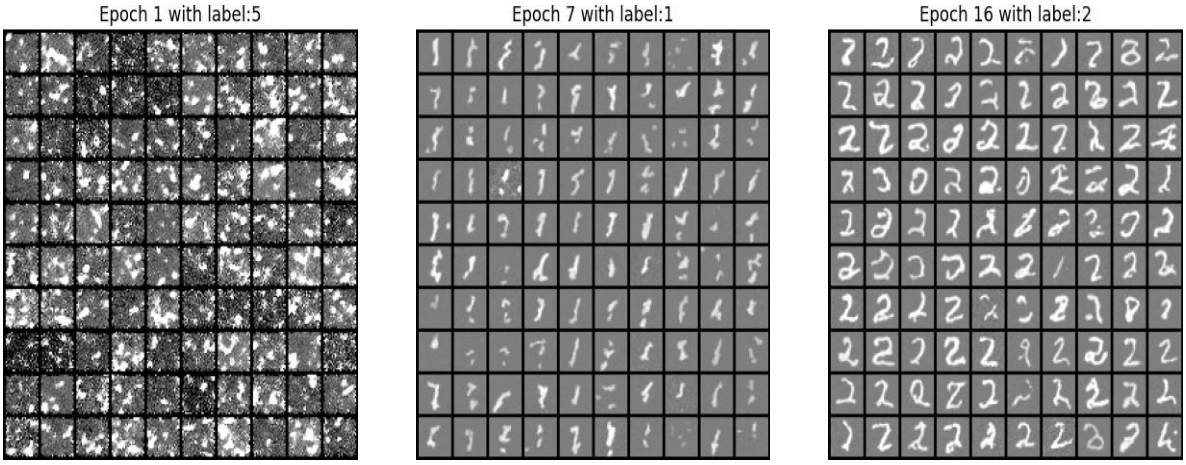
```
t = pos_encoding(t, self.time_dim, self.device)
t += self.label_emb(y)
```

Additionally, my training function incorporates a learning rate scheduler to dynamically adjust the learning rate throughout the training process.

1.3 Results

To continuously gauge the quality of the training procedure, I also had the train loop create a plot after each epoch showing 100 generations of the same digit. Some examples of these are shown in Figure 2

Here it is observed how, already after 16 epochs, the model is able to generate convincingly looking



(a) First epoch, 100 generations of digit 5

(b) Seventh epoch, 100 generations of digit 1

(c) Sixteenth epoch, 100 generations of digit 2

Figure 2: Comparison of different epochs and generations

2s. We also observe the gradual increase in quality from epoch 1 over 7 to the 16th.

With training for 30 epochs, we see how satisfactory results are achieved for most digits in Figure 3. For especially the digits "4" and "8", we observe how the model is failing to capture some of

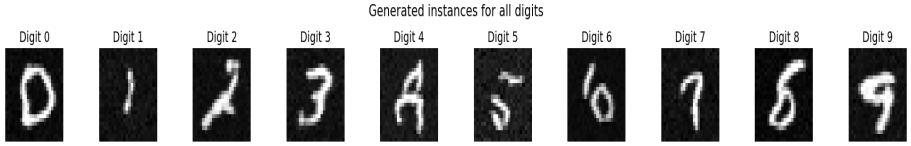


Figure 3: Digits

the nuances required for these digits. the "4" looks mostly like an "A", which suggests that the model is over reliant on straight lines. The same appears to be the case for digit "8". Efforts to alleviate this sub optimal performance could be to prolong training. This issue suggests that the model has not been able to learn enough details of the round artefacts, relying on straight lines instead. It could also be due to high variability within the dataset for digit "4", which combined with only training for 30 epochs results in the model not learning enough details.

We can also observe how the digit is more or less completely noisy up until its final timestep as seen in Figure 4. This could possibly be alleviated by the cosine noise scheduler as suggested by other authors [1], however, as the final digit is satisfactory it does not appear to be a critical issue in the current use case.

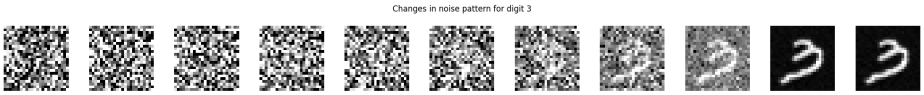


Figure 4: Noise level over timesteps for diffusion model

2 Bayesian flow networks (BFN)

The second part of this assignment will introduce Bayesian Flow Networks as described in [3], with the goal of comparing and discussions its differences and similarities to diffusion models in the context of MNIST image generation. To enable such analyses, I will start with a brief introduction to the model proposed in [3].

2.1 Introduction to BFN

In order to grasp the workings of BFN, it is fruitful to begin by considering the problem the proposed model attempts to solve. In my interpretation of the architecture, the objective can be boiled down to altering diffusion models to perform as well on discrete data types, as diffusion models perform on continuous data types. One of the main advantages of the diffusion model architecture is its ability to gradually learn the representation of continuous high-dimensional data through its sequence of denoising steps. By updating the variance of the Gaussian distribution only slightly between timesteps, the reverse process evades the complexities associate with modelling very high-dimensional data. This is one of the main advantages which the authors wish to build upon. The problem lies when the data type to be modelled is of discrete nature, such as text sequence, which renders a continuous transmission process unfeasible. The authors posit to overcome this challenge by letting the sequences transmit continuous probability distributions (the sender and receiver distributions) regardless of the input data type, rather than noisy version of the data itself, as in diffusion models.

2.2 Differences and similarities to Diffusion models

Having previously laid the groundwork for the main features defining BFN, we can now begin a comparison of the main similarities and differences to diffusion models. To start with the similarities, both classes of models are in the probabilistic generative realm, which both architectures posit to solve through iterative transformations (albeit different transformations).

As mentioned in [3], one of the most notable differences is how BFNs and diffusion models works with discrete data. As BFNs solely operates on probability distributions, they dont suffer the same limitations as diffusion models when modelling discrete data. This relates to the second major difference of how BFNs transmits probability distributions, and updates them according to bayesian inference, rather than progressively denoising the real data input. Also, BFNs does not rely on a forward process as there is no need to noise the input data.

2.3 Results: MNIST performance for BFN

To enable the comparison of how the two model architectures performs MNIST digit generation, let us now turn to the digits generated by BFN. For these plots, I rely on the codebase from Algomancer, with no changes to implementation other than plotting. As I had GPU available, I trained for the suggested 100 epochs.

As shown in Figure 5, the BFNs create digits of satisfactory quality. The really interesting pattern, however, arises when considering the gradual changes to the digit over the transitions, as shown in Figure 6

Figure 5: Digits for BFN

Figure 6: Noise level over timesteps for Bayesian Flow Network

Here we observe, how the generation process converges after roughly 1/3 of the timesteps.

2.4 Discussion of results

Considering the generated digits from Figure 5 and Figure 6, it is firstly observed how the generation process converges considerably quicker than in for the diffusion models (as compared to Figure 4. The authors of BFN are alluding to why this might be the case in [3]. The rationale is that the BFN starts with a fixed prior, rather than just random Gaussian noise as for the diffusion models.

Considering whether a fair comparison is possible between the two architectures it should be mentioned how the MNIST digits in this BFN section is generated using a discretized version of the digits, which essentially means that the pixel values are binarized prior to training. This is also why there are no greyish pixels in the BFN samples, and also no gradual change in color intensity level from black to white. This arguably makes the visual inspection process of the digits in especially Figure 6 more prone to biases. Principles of gestalt psychology posits how the brain processes inputs in terms of the general patterns and contours. The digits in Figure 6 at timestep 0.2 and 0.3, does not convincingly look like the correct 2, however, as there are only binary pixels and as the viewer will know to look for a 2 given the other correct digits in the sequence, it is possible to interpret the dots as representing a 2 when in fact such conclusion is still premature.

References

[1] Ting Chen. "On the importance of noise scheduling for diffusion models". In: *arXiv preprint arXiv:2301.10972* (2023).

[2] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis". In: *Advances in neural information processing systems* 34 (2021), pp. 8780–8794.

[3] Alex Graves et al. "Bayesian flow networks". In: *arXiv preprint arXiv:2308.07037* (2023).

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

[5] Jonathan Ho and Tim Salimans. "Classifier-free diffusion guidance". In: *arXiv preprint arXiv:2207.12598* (2022).

[6] Robin Rombach et al. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.