Exercise 1 - Generative Modelling

67912 - Advanced Course in Machine Learning

Last Updated: 4.5.2023

Due Date: 30.5.2023

1 Overview

Generative modeling has been one of the focal points for the machine learning community in the past few years. In this exercise you will experiment with 2 state-of-the-art methods for generative modelling. First, you will train a diffusion model which you will build from scratch by yourself. This part of the exercise is based on the formulation you saw in class, taken from the works of Song et al. [1] & Karras et al. [2]. Second, you will experiment with an auto-regressive model for text generation, based on GPT-2 [3] by OpenAI. Specifically, you will work on the educational version made by Andrej Karpathy.

You are expected to submit a PDF report of your work, in which you answer the questions, and present the requested findings described in Sec. 2.2 & 3.2. In the report please explain the difficulties you encountered, what worked and what didn't. If you have additional interesting findings you are welcome to describe them as well.

2 Training a 2D Diffusion Model

One recent breakthrough in generative modelling has been text-to-image models such as DALL-E 2 [4], Imagen [5], Stable Diffusion 2 (Re-implementation of Rombach et al. [6]) and many more. Most of these methods are based on diffusion models. In this exercise you will experiment with training a diffusion model from scratch yourself according to the relatively simple formulation you saw in class.

2.1 Background

As we saw in class, a diffusion model is composed from a forward process (diffusion process) and a reverse process. The forward process slowly converts inputs from their natural population distribution to a Gaussian distribution. The reverse process trains a model to slowly convert a sample from a Gaussian distribution back to the input distribution. We will elaborate on each process individually.

NOTE: In this exercise, we will formulate a diffusion model as a Stochastic Differential Equation (SDE), assuming

a Variance Exploding (VE) process. There are other existing formulations, e.g. DDPM [7] which we are based on VAEs, that we ignore here.

2.1.1 The Forward Process

We look at the forward process as a SDE which gradually changes the input in each time step. Specifically we shrink the input, $\mathbf{x}_0$, and add Gaussian noise to it, according to known shrinking and noising schedules: $f(t)$, $g(t)$. We start from $t = 0$ with a clean input $\mathbf{x}_0$, and gradually (step size dt) move towards $t = 1$. In each step, we shrink and add noise to the output of the previous step. Formally, a forward process step performs the following operation:

$$\mathbf{x}_{t+dt} = \mathbf{x}_t \cdot (1 - f(t) \cdot dt) + g(t) \cdot d\mathbf{W} \tag{1}$$

In this exercise, we will only discuss about the marginal time and noise schedulers: $S(t), \sigma(t)$. Therefore, in practice we can perform all the infinitesimal forward steps at once:

$$\mathbf{x}_t = \mathcal{N}(S(t) \cdot \mathbf{x}_0, S^2(t) \cdot \sigma^2(t) \cdot \mathbf{I}) \tag{2}$$

Assuming a VE process,

$$S(t) = 1 \tag{3}$$

We get:

$$\mathbf{x}_t = \mathbf{x}_0 + \sigma^2(t) \cdot \epsilon \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{4}$$

2.1.2 The Reverse Process - Training

We wish to sample more points from our input distribution, therefore we need to reverse the diffusion process. The reverse process aims to denoise an initial Gaussian noise in small steps, each inverting the process described in Sec. 2.1.1. We first train a denoiser which receives $\mathbf{x}_t$ and t, and aims to reconstruct the original $\mathbf{x}_0$.

Don't forget the next 3 implementation details:

1. The denoiser is conditional on the timestep t.
2. The denoiser predicts the added noise, not the original point. We obtain the denoised point from the relation described in Eq. 4.
3. The loss is on the prediction of the noise ϵ, rather than $\epsilon \cdot \sigma^2(t)$

Concretely, we train a diffusion denoiser D (Neural network), where each diffusion training step is as follows:

1. Sample Gaussian noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$

2. Sample a random time $t \in [0, 1]$

3. Obtain $\mathbf{x}_t$ using Eq. 4

4. Backpropagate this objective: $\mathcal{L}_{Diffusion} = \ell_2(D(\mathbf{x}_t, t), \epsilon)$

NOTE: Our denoiser is trained to predict the noise ϵ. This will affect our sampling algorithm.

2.1.3 The Reverse Process - DDIM Sampling

Sampling can be defined both in terms of the instantaneous scaling and noise addition schedules f(t) and g(t):

$$dx = [xf(t) - \frac{1}{2}g^2(t)\nabla_x log(p(x,t))]dt \tag{5}$$

We will find it more convenient to express sampling in terms of the marginal scale and noise functions $S(t)$ and $\sigma(t)$:

$$dx = [\frac{\dot{S}(t)}{S(t)}x - S^2(t)\dot{\sigma}(t)\sigma(t)\nabla_x log(p(x,t))]dt \tag{6}$$

where $\dot{\sigma}(t)$ is the time derivative of $\sigma(t)$.

This looks pretty complex at first, but as our process is VE (Eq. 3) we obtain,

$$dx = -\dot{\sigma}(t)\sigma(t) \cdot \nabla_x log(p(x,t)) \cdot dt \tag{7}$$

And we compute $\dot{\sigma(t)}$ analytically.

Important Note: We are going from $t = 1$ to $t = 0$ in dt step sizes. This means dt **is negative!**

Another minor change from what you saw in class, is that our denoiser predicts the noise. Keeping taht in mind, the reverse sampling step shall follow:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_t - \sigma(t)D(\mathbf{x}_t, t) \tag{8}$$

Alg. 1 describes our revised DDIM sampling process.

2.1.4 The Reverse Process - Probability Estimation

We wish to approximate the probability of a given input $\mathbf{x}$. As we saw in class, we can approximate the negative ELBO on the log-probability of $\mathbf{x}$ is given by:

$$\mathcal{L}_T(\mathbf{x}) = \frac{T}{2}\mathbf{E}_{\epsilon\sim\mathcal{N}(0,\mathbf{I}),t\sim\mathbf{U}(0,1)}[(SNR(t - |dt|) - SNR(t)) \cdot ||\mathbf{x} - \hat{\mathbf{x}}_0(\mathbf{z}_t; t)||_2^2] \tag{9}$$

where $SNR(t) = \frac{S^2(t)}{\sigma^2(t)}$. In our VE process $SNR(t) = \frac{1}{\sigma^2(t)}$

We won't go into much more details here, as this is out of scope. Saying that, in order to estimate $p(\mathbf{x})$ we will:

Algorithm 1 Reverse Process - DDIM Sampling

Input: Step Size dt, Noise Scheduler $\sigma(t)$

Sample $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$

for $t = 1, 1 + dt, ..., 0$ **do** ▷ $(dt < 0)$

$\hat{\epsilon} = D(\mathbf{z}, t)$

$\hat{\mathbf{x}}_0 = estimate\_denoised(\mathbf{z}, t, \hat{\epsilon})$ ▷ (Eq. 8)

$score(z) = \frac{\hat{\mathbf{x}}_0 - \mathbf{z}}{\sigma^2(t)}$

$dz = reverse\_process(\mathbf{z}, t)$ ▷ (Eq. 7)

$\mathbf{z} = \mathbf{z} + dz$

end for

return $\mathbf{z}$

- Randomize many possible noise and time combinations.
- Perform a forward process for all the combinations starting from $\mathbf{x}$ as the input (Eq. 4).
- Estimate $\mathbf{x}_0$ for all combinations (Eq. 8).
- Compute SNR differences for the sampled t values.
- Average the results and multiply by $\frac{T}{2}$.
- $-\mathcal{L}_T(\mathbf{x})$ is the lower bound for $\log p(x)$

2.2 The Exercise - Part 1 (65 pts)

You are required to train a diffusion model over 2D points, and answer the following questions / assignments. Sample between $1000 - 3000$ (your choice) points uniformly, inside a square, ranging from $x = -1, y = -1$ to $x = 1, y = 1$. This is your data, which you will train a diffusion process on. Meaning, we want to create a model that samples points in a 2D space. Don't hesitate the visualize the data during training or sampling, for understanding your errors.

The main challenge of this exercise is truly understanding the equations, then putting them together to create a diffusion process from scratch. Therefore, we do not supply you with any code. You are free to choose your implementation, as long as it follows the equations correctly. Do not use any external code or library other than *numpy*, *pytorch*, *matplotlib*, *tensorboard*, *wandb*, *pandas*, *plotly* (Optional, a prettier replacement for *matlpotlib*), and *tqdm*.

You are free to choose the Architecture, Optimizer, Learning Rate, Scheduler, Number of Sampling Steps (T), Batch Size, Number of Epochs, and etc. Saying that, we recommend you start from the followings:

- Arch.: a simple network, with $2 - 3$ Fully-Connected layers, with Leaky-Relu layers between them
- Optimizer: Adam
- Learning Rate: $1e - 3$
- T: 1000 (therefore $dt = 1./T = 1e - 3$)

- Scheduler: $\sigma(t) = e^{5(t-1)}$
- Batch Size: all samples (essentially performing GD, and not SGD).
- Number of Epochs: 3000

2.2.1 Resources

We designed this part of the exercise to require very little resources. You should be a able to run this part of the exercise on your private computer, without any problems. If you run into resources problems, Google Colab is more than enough for this part.

2.2.2 Question & Assignments

You are required to train a diffusion model over 2D points. Additionally, during the building of your model, we ask you to present the followings:

Unconditional Diffusion Model:

1. For a point of your choice, please present the forward process of it as a trajectory in a 2D space. Start from a point outside the square. Color the points according to their time t.
2. Present the loss function over the training batches of the denoiser.
3. Present a figure with 9 ($3x3$ table) different samplings of 1000 points, using 9 different seeds.
4. Show sampling results for different numbers of sampling steps, T. How does the sampling length affect the results?
5. Slightly modify the given sampler, or create a sampling schedule of your own. Plot the σ coefficients over time (sample them using some dt step size), in the original sampler, and your modified version. Describe your choice for ablation / sampler. In your viewpoint, what are the rules of thumb you find for an effective sampler?
6. Insert the input noise to the reverse sampling process 10 times (do not seed the model), and plot the outputs. Are the outputs the same?
   - If yes, offer a way to modify the sampling so they will be different. Implement the sampling process according to your modification.
   - If not describe the source of variation.

   Using the modified / original (depending on your answer) sampler, plot the denoising trajectories of 4 of the points on the same figure.

Conditional Diffusion Model:

Train a conditional version of your model. Condition the input by splitting your square to sub-spaces (however you wish). Have at least 5 classes. Insert the class input to the network with an embedding layer (look at *torch.nn.Embedding*), followed by a Fully-Connected that mixes the input with the class vector. The subspace of the point shall represent its class.

1. Plot your input coloring the points by their classes.
2. How did you insert the conditioning? Which equations have to be modified?
3. Sample 1 point from each class. Plot the trajectory of the points, coloring each trajectory with its class's color. Validate the points reach their class region.
4. Plot a sampling of at least 1000 points from your trained conditional model. Plot the sampled points coloring them by their classes.
5. Analyze the sampling results. Are certain classes more difficult than other? Do you see the same spatial distribution as the input? Do you have other interesting findings?
6. Estimate the probability of at least 5 sampled points. Plot the points on top of the data. Choose the set of points so some of them are within the input distribution and others are out of it. Also, choose a pair of points with the same location but different classes, one that matches the input distribution and one that isn't. Describe the results, explaining what happens when the location / class of a point do not match the initial distribution.

(Optional) Bonus (10 pts): Make one of your trained models (conditional or unconditional), output the point (4, 5.5) (which is far away from the square), as its sampled output. How did you do it? Describe your attempts (including unsuccessful ones). You may select the noise that is added in each step of you wish. Saying that, trivial solutions will receive less points than more advanced ones. Plot the trajectory of the sampled point over time.

3 Training an Auto-Regressive Text Model

As shown in class, additionally to diffusion models, auto-regressive models have also shown impressive generative results recently. The most popular (by a landslide), is of course ChatGPT. In this part of the exercise you will train a GPT-2 model of your own (in much smaller scale), and evaluate its results. We expect you to understand the mechanics behind auto-regressive models, as well as the transformer architecture, which you will be requested to modify.

3.1 Background

Auto-regressive models, are relatively simple. Given an input series of $\mathbf{x}_0, ..., \mathbf{x}_{n-1}$, we request the model to predict the next token $\mathbf{x}_n$.

We are dealing with text in our setting. Meaning, our input and output are composed of discrete tokens: $\mathbf{x}_0, ..., \mathbf{x}_{n-1} \in \mathcal{Y}$, where we denote the corpus of tokens we assume in our language as $\mathcal{Y}$. E.g., $\mathcal{Y}$ can be the set of words in the English languange. In our experiments, $\mathcal{Y}$ will be a set of common character combinations in English (you are given an encoder to extract them). We therefore treat the problem as a classification problem, requesting our model to predict the probability of each token $y \in \mathcal{Y}$ to be the following token in the sentence.

3.2 The Exercise - Part 2 (35 pts)

Your are required to train a auto-regressive model on the "Alice in Wonderland" text, which you can download from here. We refer you to the code of Andrej Karpathy as a starting point. This code uses the Cross-Entropy loss term in your experiments.

There are 2 main challenges to this part of the exercise: (i) Understanding the mechanics of the transformer architecture. (ii) Understanding the limitations of auto-regressive text generator.

3.2.1 Splitting the Text to Blocks

In order to train our auto-regressive generator we will need to feed the network with small parts of the long text in each batch. To do so, first encode the entire text using the supplied Byte Pair Encoder. In the file *bpe.py*, you can find an example for how to use the encoder. Save the tokenized version of the text as your data. Then, the i^{th} example will be the next $block\_size$ tokens up to i, Meaning, $tokens[i : i + block\_size]$. During training, target variable will be $tokens[i + 1 : i + block\_size + 1]$, meaning we train the model to predict the next token based on the given prefix.

3.2.2 Inversion

In Question 2 of Sec. 3.2 you will be asked to invert the AR model. Meaning, given a known sentence s, we want to find an input vector inp_vec, that satisfies: $AR(inp\_vec) = s$. The way we do that, is by optimizing the input vector of the AR model. In other words we:

1. Freeze the AR model
2. Initialize a learnt input vector
3. Loop k times, with the loss: $\mathcal{L}_{inversion} = Cross\_Entropy(AR(inp\_vec)[i], s[i])$, For all $i \in range(len(s))$.

After that we should get an input vector which satisfies: $AR(inp\_vec) = s$.

NOTE: inp_vec should be inserted instead of of the token embedding layer in the *forward* method of the AR model. This is because the token embedding layer expects *int* type vectors which we cannot optimize by infinitesimal steps.

3.2.3 Resources

We do not recommend running this part of your exercise on your private computer. Instead, run it with Google Colab's using a GPU. We also recommend adding some prints during the training so you'll be able to monitor your experiments while they are running. We also recommend to use a block size of no more than 64 tokens, due to resource limitations.

3.2.4 Questions & Requirements

Please perform the described experiments and discuss their results:

1. Train the model as standard GPT-2. Present the loss term over training time.

2. Perform an inversion process (Sec. 3.2.2) to the following sentence: "I am a little squirrel holding a walnut". Explain any special choice in your experiments. We won't deduct points if you were not able to reach the sentence, but in this case we want you to try and explain why the model was not able to do so.

3. Generate a sentence with at least 11 words (the 11^{th} word must be generated by the model). In the 11^{th} word prediction (find the tokens responsible for it), extract the attention scores ($Q * K$) for the last transformer block. Average the scores over the different attention heads. Paint the tokens by their relation to the 11^{th} word. Explain why did the model choose the next word based on this analysis. If you find painting a sentence too complex, you may present the scores below each word instead (which can be done with paint as well). What is the sum of all attention scores?

4. Repeat the last question, this time using the attention scores from the first transformer block. What difference do you see in the results?

5. Sample any sentence from the model. Compute the log-probability score of the sentence by multiplying the model's probability prediction for each word in the sentence. Don't forget to work in log-scale, working with the actual probabilities will lead to unstable and extremely low values.

4 Grading & Requirements

4.1 Ethics & Limitations

This is an advanced course, therefore we will have 0 tolerance for any kind of cheating. We are stating it very clearly that you are forbidden from doing the followings:

- Using any external github repository other minGPT.
- Sharing any part of your code with other students.
- Using any external library other then *numpy*, *pytorch*, *matplotlib*, *tensorboard*, *wandb*, *pandas*, *plotly*, *tqdm*, and additionally the ones imported by minGPT only for Part 2.

4.1.1 Github Copilot & ChatGPT

We **allow** (and even encourage) using automatic tools, such as github copilot and even the infamous ChatGPT, for the exercise. Saying that, we do have a few restrictions for these tools: We require to note and explain (in the submitted code itself) every part of the code that was written by these tools. Also, in your PDF (as a separate section) explain how did you use these tools, and for which aspects of the exercise you found them useful.

4.2 Submission Guidelines

4.2.1 PDF Report

The PDF report should be 10 pages at max. Answer the questions from Sec. 2.2 & 3.2 in it, marking clearly where are the answers to each questions. We are note requiring any format, but 10 paged stories that lack any structure will not be accepted.

4.2.2 Code & Weights

For each of Sec. 2.2 & 3.2, prepare a main file which trains your model and samples points from it. We will run this file as is to evaluate your results. Please name the files: "example_diffusion.py" & "example_gpt.py".

For us to be able to run your code, please add the trained weights for **one** of the trained models you created, for each method (one for the diffusion model, and one for the auto-regressive model). Additional weight files will be simply ignored.

4.2.3 Submission

In your submission should be a zip file including the following:

- A README file with your name and and cse username.
- Your code for the diffusion model training, ablations and evaluation.
- Your code for the auto-regressive model training, ablations and evaluation.
- Trained weights for your diffusion model and autoregressive model.
- A PDF report answering the questions / requests from Sec. 2.2 & 3.2.

References

[1] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[2] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *arXiv preprint arXiv:2206.00364*, 2022.

[3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[4] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[5] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

[6] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.