Statistical Methods in Image Processing EE-048954

Homework 2: Langevin Dynamics and Energy Based Models

Due Date: May 31, 2022

Submission Guidelines

- Submit a zip file with the name `hw2_id1_id2.zip` where `id1` and `id2` are your student IDs.
- Working environment:
  - For the coding questions we support **Nvidia** hardware, either using Google Colab or on there not require any installation.
  - You should submit a **jupyter** `.ipynb` file.
  - Your code should be in a **clear** and **organized** manner. Use markdown cells to explain what you are implementing.
  - If, *I.e.*, you want to use `DATASETS_ROOT_DIR = '/content/drive/MyDrive/datasets'` and it is your nested computing environ, and resources also.
  - **Do hyperparameters** search on the data, not on the `test` data (`DEV` - `DEV`) and report the results.

Mounting your drive for saving/loading stuff

```
!pip install -q -U google-drive-ocamlfuse
!apt-get install -qq -w 1 libfuse-dev libglib2.0-dev libcurl4-openssl-dev libjsoncpp-dev
!google-drive-ocamlfuse -cc
!mkdir -p drive
!google-drive-ocamlfuse drive
!ln -s /content/drive/MyDrive/ /content/drive/MyDrive/.shortcut-targets-by-id
```

Importing relevant libraries for Part 1

```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os
import torch
import torchvision
import torchvision.transforms as T
import torchvision.datasets as dset
from torchvision.utils import make_grid
from torch.utils.data import DataLoader
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import HTML, display, clear_output
import ipywidgets as widgets, io, PIL, time, copy, zipfile, datetime
from datetime import datetime
from google.colab.patches import cv2_imshow
DATASETS_ROOT_DIR = '/content/drive/MyDrive/datasets'
!ln -s /content/drive/MyDrive/datasets /content/drive/MyDrive/.shortcut-targets-by-id
```

Part 1. Top 20 Dataset (10 points)

In this part we will be using a small dataset of natural image patches that could be used for simple from energy-based models of images. But in one thing: all are of the same size and are very low-dimensional (27 x 27 pixels). For more information about this dataset, please read the following paper. Unfortunately we can familiar with such DEM with eight dimensions.

Since formally, the probability density distribution corresponds given by:

$$p_{top20}(x) = \frac{1}{20}\sum_{i=1}^{20}\mathcal{N}(x|\mu_i, \sigma_i^2 I)$$

with $\sigma_i = 1, \mu_i \sim \mathcal{U}[-1, 1]$.

and $\mu_i = \left[\mu_i^1, \mu_i^2\right] \sim \mathcal{U}[-1, 1]$.

Task 1: Write down the analytical gradient of $\log p_{top20}(x)$ with respect to x. (1 A-page)

Answer 1:

$$\nabla_x \log p_{top20}(x) = \frac{\sum_{i=1}^{20} \mathcal{N}(x|\mu_i, \sigma_i^2 I)\left(\frac{\mu_i - x}{\sigma_i^2}\right)}{\sum_{i=1}^{20} \mathcal{N}(x|\mu_i, \sigma_i^2 I)}$$

While this distribution can be sampled from using standard techniques, here we will sample from it using Langevin Dynamics.

Task 2: Implement Langevin Dynamics for sampling from p_{top20}.

- Initialize 1000 samples of pixels $x_0 \sim \mathcal{U}[-1, 1]$ and denote the corresponding $x_0 \in [-1, 1]$.
- Update the samples according to the Langevin Dynamics update step:

$$\forall x_i \in \left[x_0^1, \ldots, x_0^{1000}\right] \rightarrow x_i \leftarrow x_i + \frac{\epsilon}{2}\nabla_x \log p_{top20}(x_i) + \sqrt{\epsilon}\,\mathcal{N}(0, I)$$

- Repeat the process step for $T = 1000$ iterations.

```
num_samples, DIMENSIONS = 1000, 2
mu = torch.tensor([[-0.5, -0.5], [-0.5, 0.5], [0.5, -0.5], [0.5, 0.5], [-0.5, 0.0], [0.0, -0.5], [0.5, 0.0], [0.0, 0.5], [-0.25, -0.25], [-0.25, 0.25], [0.25, -0.25], [0.25, 0.25], [-0.75, -0.75], [-0.75, 0.75], [0.75, -0.75], [0.75, 0.75], [-0.75, 0.0], [0.0, -0.75], [0.75, 0.0], [0.0, 0.75]])
sigma = torch.ones(20)
def log_gaussian_pdf(x, mu, sigma):
    # Assumes x: (N, D), mu: (K, D), sigma: (K)
    N, D = x.shape[0], x.shape[1]
    K = mu.shape[0]
    # Compute log N(x | mu_k, sigma_k^2 * I) for each component.
    diff = x[:, None, :] - mu[None, :, :]  # (N, K, D)
    log_prefactor = -D * torch.log(sigma)  # (K,)
    log_exponent = -0.5 * (diff**2).sum(dim=2) / (sigma**2)  # (N, K)
    return log_prefactor + log_exponent  # (N, K)
def log_gaussian_mixture_pdf(x, mu, sigma):
    # Assumes x: (N, D), mu: (K, D), sigma: (K)
    log_components = log_gaussian_pdf(x, mu, sigma)  # (N, K)
    return torch.logsumexp(log_components, dim=1)  # (N,)
def langevin_dynamics(num_samples, num_steps, step_size, log_pdf_fn):
    # Initialize samples from uniform distribution [-1, 1]
    samples = torch.rand(num_samples, DIMENSIONS) * 2 - 1
    for i in range(num_steps):
        if i % 100 == 0:
            print(f"Step {i}/{num_steps}")
        # Compute gradient of log_pdf with respect to samples
        samples.requires_grad_(True)
        log_prob = log_pdf_fn(samples, mu, sigma)
        grad = torch.autograd.grad(log_prob.sum(), samples)[0]
        samples = samples.detach()
        # Langevin dynamics update: x_t = x_t + (step_size/2) * grad + sqrt(step_size) * noise
        noise = torch.randn_like(samples)
        samples = samples + (step_size/2) * grad + (step_size**0.5) * noise
        # Optional: clamp to [-1, 1] if we want to keep samples in the valid range
        samples = torch.clamp(samples, -1, 1)
    return samples
num_steps = 1000
step_size = 0.01
final_samples = langevin_dynamics(num_samples, num_steps, step_size, log_gaussian_pdf)
print("Sampling complete.")
print("Shape:", final_samples.shape)
print("Mean:", final_samples.mean(dim=0))
print("Std:", final_samples.std(dim=0))
```

Task 3: Every $T = 100$ iterations, from you using the provided code lines below and compare them visually to the samples drawn using ground-truth Gaussian mixture model parameters.

```
log_pdf_fn = lambda samples, mu, sigma: log_gaussian_mixture_pdf(samples, mu, sigma)
```

Verifies Task 3:

Since the exact the top-given square dis-shapes and their means are clearly. If the results match we also plotted the same scatters using kernel density estimation (KDE) to visualize the smooth density.

```
plot_comparison_kde(final_samples, mu, sigma, title="Langevin Dynamics KDE", save_path="langevin_kde.pdf")
print("Saved to langevin_kde.pdf")
```

Task 4: Repeat Steps 2 and 3 this time with $\frac{\epsilon}{2}\nabla_x \left[\log p(x) - \frac{\|x\|^2}{2}\right]$. Compare the results to Task 3 and explain the effect on the sampling process. The Langevin Dynamics update step now becomes $x_i \leftarrow x_i + \frac{\epsilon}{2}\nabla_x \left[\log p(x) - \frac{\|x\|^2}{2}\right] + \sqrt{\epsilon}\,\mathcal{N}(0, I)$. Compare the results to Task 3 and explain the effect on the sampling process.

```
# Modified Langevin dynamics with gradient of log_pdf - x/2
def langevin_dynamics_modified(num_samples, num_steps, step_size, log_pdf_fn):
    samples = torch.rand(num_samples, DIMENSIONS) * 2 - 1
    for i in range(num_steps):
        if i % 100 == 0:
            print(f"Step {i}/{num_steps}")
        samples.requires_grad_(True)
        log_prob = log_pdf_fn(samples, mu, sigma)
        grad = torch.autograd.grad(log_prob.sum(), samples)[0]
        samples = samples.detach()
        noise = torch.randn_like(samples)
        samples = samples + (step_size/2) * (grad - samples/2) + (step_size**0.5) * noise
        samples = torch.clamp(samples, -1, 1)
    return samples
final_samples = langevin_dynamics_modified(num_samples, num_steps, step_size, log_gaussian_pdf)
print("Sampling complete.")
print("Shape:", final_samples.shape)
print("Mean:", final_samples.mean(dim=0))
print("Std:", final_samples.std(dim=0))
```

Discussion 3-4:

$\nabla_x \left[\frac{\|x\|^2}{2}\right]$ - we see that the drifts with linear regime to are a more distinct step the centrality of the mass. For $\nabla_x \left[\frac{\|x\|^2}{2}\right]$ the results are more spread out.

$\nabla_x \left[\frac{\|x\|^2}{2}\right]$ - the gaussian drifts seem to be more centered around the mean and the samples are more concentrated. For $\nabla_x \left[\frac{\|x\|^2}{2}\right]$ the gaussian drifts seem to be more spread out and less concentrated.

Importing additional relevant libraries for Parts 2-IV

```
%matplotlib inline
%load_ext autoreload
%autoreload 2
import os
import torch
import torchvision
import torchvision.transforms as T
import torchvision.datasets as dset
from torchvision.utils import make_grid
from torch.utils.data import DataLoader
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import HTML, display, clear_output
import ipywidgets as widgets, io, PIL, time, copy, zipfile, datetime
from datetime import datetime
from google.colab.patches import cv2_imshow
DATASETS_ROOT_DIR = '/content/drive/MyDrive/datasets'
!ln -s /content/drive/MyDrive/datasets /content/drive/MyDrive/.shortcut-targets-by-id
```

Part II: Langevin Dynamics (30 points)

General introduction

In the remainder of this exercise we will focus on sampling from an Energy Based Model (EBM) that was trained on To the distribution, you will be using a pre-trained EBM to generate Mnist images. The EBM is given to you from a pre-trained Noise-Conditional Score Network (NCSN) and defined by:

$$p_\theta(x) = \frac{e^{-f_\theta(x)}}{Z_\theta}$$

Specifically, the model gets an image x and returns a scalar. Intuitively, the lower the energy, the more likely the model assigns to the given image.

Model Architecture

The provided code defines `DenseNet` implements the neural network approximating $f_\theta(x)$ with parameters θ where x is an input and $f_\theta(x)$ is a scalar. The network uses `DenseLayer` blocks with `Swish` activations and `GroupNorm` normalization, and a final `AvgPool` and `Linear` layer to produce the scalar output. The network is wrapped in `MnistNCSN` and expects inputs of shape `(B, C, H, W)` where `C = 1` and `H = W = 28` for MNIST.

In the following questions, we will use this model to generate MNIST samples. For more information about NCSN, see Yang and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", NeurIPS 2019.

```
# Download the pre-trained NCSN model checkpoint
!gdown --fuzzy "https://drive.google.com/file/d/1-0-Tf-C-Zo7rI_Y0Ki5_vfW_Cgo-Rn-j/view?usp=sharing"
# Load the checkpoint and inspect what's inside
import torch
ckpt = torch.load('ncsn_mnist.pt', map_segment_ref_1, map_segment_ref_2, map_segment_ref_3, map_segment_ref_4, map_segment_ref_5, map_segment_ref_6, map_segment_ref_7, map_segment_ref_8, map_segment_ref_9, map_segment_ref_10, map_segment_ref_11, map_segment_ref_12, map_segment_ref_13, map_segment_ref_14, map_segment_ref_15, map_segment_ref_16, map_segment_ref_17, map_segment_ref_18, map_segment_ref_19, map_segment_ref_20, map_segment_ref_21, map_segment_ref_22, map_segment_ref_23, map_segment_ref_24, map_segment_ref_25, map_segment_ref_26, map_segment_ref_27, map_segment_ref_28, map_segment_ref_29, map_segment_ref_30, map_segment_ref_31, map_segment_ref_32, map_segment_ref_33, map_segment_ref_34, map_segment_ref_35, map_segment_ref_36, map_segment_ref_37, map_segment_ref_38, map_segment_ref_39, map_segment_ref_40,
```

Gen Imgs e=2/256, k=500

```
Gen Imgs e=2/256, k=500, Adv Init
```

Gen Imgs e=3/256, k=20,000, MALA Adv Init

```
Gen Imgs e=3/256, k=20,000, Adv Init
```

Gen Imgs e=3/256, k=20,000, Adv Init

```
Gen Imgs e=3/256, k=20,000, Adv Init
```

Discussion 10.

The results are very good (for high ϵ and even k and b). Only artifacts when looks like the MALA algorithm suffers from mode collapse.

We tried to understand whether or not this is thanks to use of the VAE's or but results choose first both output looks clean and only minor for MALA.

Again we could not see any added value to the usage of MALA over Langevin. Although there is a slight improvement in the MALA results, anyway the cost to use it per sample is x10 with other hyper, so those were not the huge.

Part IV: Perceptual and MMSE Denoising (30 points)

Now we will use our trained models to denoise images. We will first consider the case of denoising using Langevin Dynamics and the VAE model trained in exercise 1a. We will perform denoising by finding a sample from $p(x|y)$.

Part IV. Write an expect expression for $\nabla_x \log p(x|y)$ in terms of $\nabla_x \log p(x)$.

Answer 11.

$$\nabla_x \log p(x|y) = \nabla_x \log \left(\frac{p(y|x)p(x)}{p(y)} \right)$$

$$\nabla_x = \text{linearity} \; \log p(y|x) + \nabla_x \log p(x) - \nabla_x \log p(y)$$

$$\nabla_x \log p(y|x) = \frac{1}{\sigma^2}(y - x) + \nabla_x \log p(x) - 0$$

$$\nabla_x \log p(x|y) = \frac{1}{\sigma^2}(y - x) + \nabla_x \log p(x)$$

Part IV. Using the update rule in Langevin Dynamics (Part II) but changed in order to draw samples from $p(x|y)$ instead of $p(x)$. We will such perform perceptual denoising results.

Answer 12.

In this case we enter the prior knowledge about the process of receiving Y in the component:

$$\frac{1}{\sigma^2}(y - x)$$

Every optimization step will also take into account the prior information.

Part IV. The implement of the sampling formula (Eq. Langevin dynamics) $\left[\frac{\epsilon}{2}\nabla_x \log p(x) + \epsilon\right]$ and read that images using the command

```
python main.py --denoise --model vae --ckpt ./ckpt/mnist_vae.pt --n_steps 100 --n_samples 64 --eps 0.02 --sigma 0.3 --save_path ./results/denoise_langevin_vae.pt
```

Exercise 13. Perform perceptual denoising with Langevin Dynamics with the parameters $\epsilon = 100$ and $\frac{\epsilon}{2\sigma^2} = \frac{1}{2}$. Present and discuss the results.

```
python main.py --denoise --model vae --ckpt ./ckpt/mnist_vae.pt --n_steps 100 --n_samples 64 --eps 0.02 --sigma 0.3 --save_path ./results/denoise_langevin_vae.pt
```

```
Data Imgs with sigma=50/256
```

Data Imgs with sigma=50/256

```
Data Imgs with sigma=100/256
```

Data Imgs with sigma=100/256

```
Data Imgs with sigma=100/256
```

Discussion 13.

The results are clean and have the original shape of the digits. Every digit can be classified in the correct way.

So the denoiser is working correctly.

Having said that, there is some recession of the shape of the digits, some men look, and other deformations, mostly on round digits such as 0 and 9. This is probably caused by the nature of the noise introduced by the denoiser in part of the original digit.

Part IV. Minimum MSE denoising can be obtained by averaging over the perceptual denoising results, since the conditional expectation is the minimum mean squared error estimator. Implement MMSE denoising using the command `python main.py --denoise --model vae --ckpt ./ckpt/mnist_vae.pt --n_steps 100 --n_samples 64 --eps 0.02 --sigma 0.3 --save_path ./results/denoise_langevin_vae.pt --n_avg_samples 256`. Report the MMSE results averaged over 256 perceptual samples. Why do you think the use of perceptual denoising is called that?

```
python main.py --denoise --model vae --ckpt ./ckpt/mnist_vae.pt --n_steps 100 --n_samples 64 --eps 0.02 --sigma 0.3 --save_path ./results/denoise_langevin_vae.pt --n_avg_samples 256
```



```
Data Imgs with sigma=100/256, Denoised
```

Data Imgs with sigma=100/256, Denoised


```
Data Imgs with sigma=100/256, Denoised
```

Discussion 14.

As mentioned in the previous task, the denoiser will same deformations to the original digit. By averaging, these deformed images into effect is mitigated.

So our estimation for a called perceptual denoising since the process is eliminating a negative effect of the denoiser that the negative effects cost more than either a deformation of the original shape, or not. Denoising. A perceptual expert of the process that not a mean