Due Date: May 2, 23:00

Instructions

- For all questions that are not graded only on the answer, show your work! Any problem without work shown will get no marks regardless of the correctness of the final answer.
- Please try to use a document preparation system such as LaTeX. If you write your answers by hand, note that you risk losing marks if your writing is illegible without any possibility of regrade, at the discretion of the grader.
- Submit your answers electronically via the course GradeScope. Incorrectly assigned answers can be given 0 automatically at the discretion of the grader. To assign answers properly, please:
  - Make sure that the top of the first assigned page is the question being graded.
  - Do not include any part of answer to any other questions within the assigned pages.
  - Assigned pages need to be placed in order.
  - For questions with multiple parts, the answers should be written in order of the parts within the question.
- In the code, each part to fill is referenced by a TODO and 'Not Implemented Error'
- Questions requiring written responses should be short and concise when necessary.Unnecessary wordiness will be penalized at the grader's discretion.
- Please sign the agreement below.
- It is your responsibility to follow updates to the assignment after release. All changes will be visible on Overleaf and Piazza.
- Any questions should be directed towards the TAs for this assignment: *Vitória Barin Pacela, Philippe Martin.*

For this assignment, the GitHub link is the following: `https://github.com/philmar1/Teaching_IFT6135---Assignment-3---H25`

I acknowledge I have read the above instructions and will abide by them throughout this assignment. I further acknowledge that any assignment submitted without the following form completed will result in no marks being given for this portion of the assignment..

Signature: _____

Name: _____

UdeM Student ID: _____

1 VAE (68 points)

Variational Autoencoders (VAEs) are probabilistic generative models to model data distribution $p(\boldsymbol{x})$. In this question, you will be asked to train a VAE on the MNIST dataset, using the negative ELBO loss as shown in class. Each image has size 28×28. You should model the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, i.e. the decoder, as a product of Bernoulli distributions.

In this question, you should attach your code for the qualitative evaluation (questions marked with `report`) to the report.

1. **(`unittest`, 6 pts)** Implement the function '`log_likelihood_bernoulli`' in '`q1_vae.py`' to compute the log-likelihood $\log p(\boldsymbol{x})$ for a given binary sample $\boldsymbol{x}$ and Bernoulli distribution $p(\boldsymbol{x})$. $p(\boldsymbol{x})$ will be parameterized by the mean of the distribution $p(\boldsymbol{x} = 1)$, and this will be given as input for the function.

2. **(`unittest`, 6 pts)** Implement the function '`log_likelihood_normal`' in '`q1_vae.py`' to compute the log-likelihood $\log p(\boldsymbol{x})$ for a given float vector $\boldsymbol{x}$ and isotropic Normal distribution $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$. Note that $\boldsymbol{\mu}$ and $\log(\boldsymbol{\sigma}^2)$ will be given for Normal distributions.

3. **(`unittest`, 8 pts)** Implement the function '`log_mean_exp`' in '`q1_vae.py`' to compute the following equation[1] for each $\boldsymbol{y}_i$ in a given $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_i, \ldots \boldsymbol{y}_M\}$;

$$\log \frac{1}{K} \sum_{k=1}^{K} \exp\left(y_i^{(k)} - a_i\right) + a_i,$$

   where $a_i = \max_k y_i^{(k)}$. Note that $\boldsymbol{y}_i = [y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(k)}, \ldots, y_i^{(K)}]$s.

4. **(`unittest`, 6 pts)** Compute the analytical solution of the KL divergence $D_{\text{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right)$ for given p and q, where p and q are multivariate normal distributions with diagonal covariance. Then, implement the function '`kl_gaussian_gaussian_analytic`' in '`q1_vae.py`'.

5. **(`unittest`, 6 pts)** Implement the function '`kl_gaussian_gaussian_mc`' in '`q1_vae.py`' to compute KL diveregence $D_{\text{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right)$ by using Monte Carlo estimate for given p and q. Note that p and q are multivariate normal distributions with diagonal covariance.

6. **(`report`, 8 pts)** Train a VAE using the provided network architecture and hyperparameters from '`q1_train_vae.py`'.

   Fill in the function '`loss_function.py`' from '`q1_train_vae.py`' by reusing your code from '`q1_vae.py`'.

   Optimize it with Adam, with a learning rate of 10^{-3}, and train for 20 epochs.

   Then, evaluate the model on the validation set.

   The model should achieve an average loss ≤ 104 on the validation set.

   Report the final validation loss of your model and plot the training and validation losses.

[1] This is a type of log-sum-exp trick to deal with numerical underflow issues: the generation of a number that is too small to be represented in the device meant to store it. For example, probabilities of pixels of image can get really small. For more details of numerical underflow in computing log-probability, see `http://blog.smola.org/post/987977550/log-probabilities-semirings-and-floating-point`.

7. **(`report`, 6 pts) Provide visual samples generated by the model.** Comment on the quality of the samples (e.g. blurriness, diversity, "realisticness").

8. **(`report`, 12 pts)** We want to see if the model has learned a disentangled representation in the latent space. The VAE provided yields 20 latent factors. Plot the images from the latent traversals with 5 samples per latent factor. What can you comment about the disentanglement of the latent factors?

   To make the traversals, sample z from your the distribution (which, in this case, is a standard Gaussian). Make small perturbations to your sample z for each dimension (e.g. for a dimension i, $z'_i = z_i + \epsilon$). ϵ has to be large enough to see some visual difference. For each dimension, observe if the changes result in visual variations (that means variations in $g(z)$ – where g is the decoder).

9. **(`report`, 10 pts) Compare between interpolating in the data space and in the latent space.** Pick two random points z_0 and z_1 in the latent space sampled from the prior.

   (a) For $\alpha = 0, 0.1, 0.2 \ldots 1$ compute $z'_\alpha = \alpha z_0 + (1-\alpha) z_1$ and plot the resulting samples $x'_\alpha = g(z'_\alpha)$.

   (b) Using the data samples $x_0 = g(z_0)$ and $x_1 = g(z_1)$ and for $\alpha = 0, 0.1, 0.2 \ldots 1$ plot the samples $\hat{x}_\alpha = \alpha x_0 + (1-\alpha) x_1$.

   Explain the difference between the two schemes to interpolate between images.

2 Diffusion models

Implementing Denoising Diffusion Probabilistic Models (DDPM) and Classifier Free Guidance (CFG) models to generate MNIST style images (100 pts) .

1. Unconditional generation (45 pts.)

In this problem, you will implement a DDPM (`https://arxiv.org/abs/2006.11239`) class on MNIST dataset using PyTorch according to the guidence. The goal is to minimize the loss function and train the model to generate MNIST images.

The UNet classes is already implemented for you. You need to implement the DDPM class (see details below) and finish the Trainer class. The images generated by the model will be automatically shown according to the Trainer class implementation. Make sure the generated images are shown in the output, it will be graded.

If you need to create your own virtual environment, please create a python==3.11 venv and pip install dependencies from requirements.txt

We recommend that you follow the notebook DDPM.ipynb on your own computer. Once you filled the methods for the DenoiseDiffusion and Trainer classes, you can copy past them in ddpm.py and trainer.py, this will be used by Gradescope. Then, when you are ready to do a training, you can import this notebook to Google Colab if you need free access to a GPU.

Gradescope: The files you need to complete and submit via Gradescope for auto-grading are `q2_trainer_ddpm.py`, `q2_ddpm.py`. You must also submit your report (PDF) on Gradescope. Your report must contain answers to the problem. You do not need to submit code for these questions; the report will suffice.

Grade:

- Implement the DenoiseDiffusion class (15 points).
- Complete *sample* and *generate_intermediate_samples* methods from the Trainer class (10 points).
- Train the model to generate reasonable MNIST images within 20 epochs (10 points).
- Write a report to describe
  - The sampled images generated by each epochs and give recommendations about what could be done to improve the sampled images (5 points).
  - The images generated at different steps of the diffusion reverse pass by the trained model using the function *generate_intermediate_samples*() (5 points).

  Please note that the function to generate the image is already provided.

2. Conditional generation (55 pts.)

In the next part of the problem, you will modify the code to allow user to specify the model the class to generate. To that end, the model must be trained by taking into account the labels of the training data. We will use the Classifier Free Guidance approach (`https://arxiv.org/pdf/2207.12598`). The UNet class has been updated to take into account labels as input. In this tutorial, algorithm 1 will sample time t uniformly, just like in previous DDPM implementation.

Once again, we recommend you to follow the notebook ClassifierFreeGuidance.ipynb on your own computer. Once you filled the methods for the CFGDiffusion and Trainer classes, you can copy past them in cfg_diffusion.py and trainer.py, this will be used by Gradescope. Then, when you are ready to do a training, you can import this notebook to Google Colab if you need free access to a GPU.

Gradescope: The files you need to complete and submit via Gradescope for auto-grading are `q3_trainer_cfg.py`, `q3_cfg_diffusion.py`. You must also submit your report (PDF) on Gradescope. Your report must contain answers to the problem. You do not need to submit code for these questions; the report will suffice.

Grade:

- Explain why is the model called Classifier Free and why Guidance (10 points).

- According to the paper, what would be an alternative of classifier free ? Explain how would the loss change in this alternative compared to the original DDPM loss ? (10 points)
- Implement CFGDiffusion class (20 points)
- Complete the Trainer.sample() method (10 points)
- Write a report to describe the sampled images generated by each epochs (5 points).

For more details, please see the instructions provided in the Colab notebook.

Please DO NOT change the code provided, only add your own code where indicated. It is recommended that you use CPU session (for instance from VSCode) to debug when GPU is not necessary since Colab only gives 12 hrs of free GPU access at a time. If you use up the GPU resource, you may consider using Kaggle GPU resource. Thank you and good luck!