Due Date: April 12, 2020
Instructions

- *Read all instructions and questions carefully before you begin.*
- *For all questions, show your work!*
- *Submit your report (PDF) and your code electronically via the course Gradescope page.*
- *Training takes time and patience. Please start well ahead of time!*
- *TAs for this assignment are* ***Christos Tsirigotis*** *and* ***Philippe Brouillard****.*

Problem 1

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 *Binarised MNIST* dataset, using the negative ELBO loss as shown in class. Note that each pixel in this image dataset is binary: The pixel is either black or white, which means each datapoint (image) a collection of binary values. You have to model the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, i.e. the decoder, as a product of bernoulli distributions. [1]

1. **(`unittest`, 2 pts)** Implement the function '`log_likelihood_bernoulli`' in '`q1_solution.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`, 2 pts)** Implement the function '`log_likelihood_normal`' in '`q1_solution.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`, 2 pts)** Implement the function '`log_mean_exp`' in '`q1_solution.py`' to compute the following equation[2] 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.

[1] The binarized MNIST is not interchangeable with the MNIST dataset available on `torchvision`. So the data loader as well as dataset will be provided.

[2] 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`.

4. **(`unittest`, 2 pts)** Implement the function '`kl_gaussian_gaussian_analytic`' in '`q1_solution.py`' to compute KL divergence $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\right)$ via analytic solution for given p and q. Note that p and q are multivariate normal distributions with diagonal covariance.

5. **(`unittest`, 2 pts)** Implement the function '`kl_gaussian_gaussian_mc`' in '`q1_solution.py`' to compute KL divergence $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\right)$ by using Monte Carlo estimate for given p and q. Note that p and q are multivariate normal distributions with diagonal covariance.

6. **(`report`, 15 pts)** Consider a latent variable model $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})dz$. The prior is define as $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_L)$ and $\boldsymbol{z} \in \mathbb{R}^L$. Train a VAE with a latent variable of 100-dimensions ($L = 100$). Use the provided network architecture and hyperparameters described in '`vae.ipynb`'[3]. Use ADAM with a learning rate of 3×10^{-4}, and train for 20 epochs. Evaluate the model on the validation set using the **ELBO**. Marks will neither be deducted nor awarded if you do not use the given architecture. Note that for this question you have to:

   (a) Train a model to achieve an average per-instance ELBO of ≥ -102 on the validation set, and report the ELBO of your model. The ELBO on validation is written as:

$$\frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{\boldsymbol{x}_i \in \mathcal{D}_{\text{valid}}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i) \geq -102$$

   Feel free to modify the above hyperparameters (except the latent variable size) to ensure it works.

7. **(`report`, 15 pts)** Evaluate *log-likelihood* of the trained VAE models by using importance sampling, which was covered during the lecture. Use the codes described in '`vae.ipynb`'. The formula is reproduced here with additional details:

$$\log p(\boldsymbol{x} = \boldsymbol{x}_i) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(\boldsymbol{x} = \boldsymbol{x}_i|\boldsymbol{z}_i^{(k)})\; p(\boldsymbol{z} = \boldsymbol{z}_i^{(k)})}{q_\phi(\boldsymbol{z} = \boldsymbol{z}_i^{(k)}|\boldsymbol{x}_i)}; \quad \text{for all } k\text{: } \boldsymbol{z}_i^{(k)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)$$

   and $\boldsymbol{x}_i \in \mathcal{D}$.

   (a) Report your evaluations of the trained model on the test set using the log-likelihood estimate ($\frac{1}{N}\sum_{i=1}^{N} \log p(\boldsymbol{x}_i)$), where N is the size of the test dataset. Use $K = 200$ as the number of importance samples, D as the dimension of the input ($D = 784$ in the case of MNIST), and $L = 100$ as the dimension of the latent variable.

Problem 2

For this problem you are requested to complete code in order to train a Generative Adversarial Network (GAN). The code you need to complete is concerned with the implementation of GAN

[3]This file is executable in Google Colab. You can also convert `vae.ipynb` to `vae.py` using the Colab.

losses and regularization schemes, as well as their training procedure. You can find the code to be implemented in '`gan.py`'. The models for the tasks at hand are already determined at '`model.py`'. You are strongly encouraged to attempt solving **Problems 4 and 5** of the theoretical assignment before the practical GAN assignment. The material in those problems is going to help you in proceeding with this practical. For further instructions on how to work with the code, please consult the module docstrings of '`run_exp.py`'. In any case, your implementation must match with the description of the functions in their docstrings.

The problem can be split into three parts: First, you are asked to implement code and pass the unittests. Then, verify the validity of your implementation by examining empirically the theoretical predictions of the dirac-matching game, which was described in **Problem 5** of the theoretical assignment. In this part, you are asked to report your observations. Once you have the theoretical prediction align with your empirical observation, you are ready to attempt the last part. For the final part, you are asked to train models with predetermined hyperparameter configurations at the image generation task, report various aspects of training and interpret your results.

1. **(`unittest`, 3.5 pts)** Implement the functions '`jsd`' and '`w1`', the Jensen-Shannon divergence (see **Problem 4** of theoretical assignment) and the Wasserstein distance, given a batch of critic outputs for real and generated data respectively.

2. **(`unittest`, 3 pts)** Implement the functions '`ogp`' and '`r1gp`', the original one-centered gradient penalty and the modified zero-centered gradient penalty, which was examined in **Problem 5** of the theoretical assignment.

3. **(`unittest`, 3 pts)** Having all potential metric approximations and critic regularization coded, you are now asked to complete the functional '`make_losses`' which returns the metric (loss for the generator) and critic's loss function, given a critic function and batches of real and generated data.

4. **(`unittest`, 1 pts)** In GANs we often keep a second version of the generator model, whose parameters are formed by computing an exponential moving average (EMA) of the parameters of the first generator model over the course of training. It has been shown that utilizing EMA parameters in evaluation is superior in performance and more stable. We ask you to complete '`apply_exponential_moving_average`' by implementing the update rule: $\tilde{\boldsymbol{\theta}}_{k+1} = \alpha\tilde{\boldsymbol{\theta}}_k + (1-\alpha)\boldsymbol{\theta}_{k+1}$, where $\tilde{\boldsymbol{\theta}}$ are the eval-time generator's parameters and $\boldsymbol{\theta}$ the train-time generator's; α is a hyperparameter. In practice, we update the parameters of the eval-time generator each time after train-time generator has been updated by an optimization step.

5. **(`unittest`, 4.5 pts)** Implement functional '`make_train_step`' with which we make a train step routine for both the critic and the generator. Make sure to implement the inner optimization loop for the critic appropriately.

6. **(`report`, 15 pts)** Execute the jupyter notebook to train a dirac generator. Substitute the random seed with your matricule number and use the predetermined configurations denoted by the prefix '`dirac-`'. In the report, provide with the final plot of the parameter trajectories for each case and interpret the results.

If theory and practice are aligned by this step, you are ready to proceed in training an image generator.

7. **(`report`, 25 pts)** We will use CIFAR10 dataset with predetermined generator and critic architectures, as well as the hyperparameter configurations denoted by the prefix '`cifar10-`'. In the report, you are requested to provide with final samples, Inception Score and Fréchet Inception Distance (FID) and compare between the 5 configurations. Notice that in this section, we have included spectral normalization as a way to constrain critic's Lipschitz constant. Also, provide with a plot over training steps for the approximated metric (JSD or W1) and FID for each run. Contrast the two quantities into the same plot. What do you observe? Which metric is more indicative of training's progress?[4]

Problem 3

Normalizing flows are expressive invertible transformations of probability distributions. In this exercise, you will implement one instance of a normalizing flow: a planar flow. Most of the code is already given, you only need to complete some functions. Likewise, to train the model, you should use the default hyperparameters that are given. For questions noted as `unittest`, no public unit tests are given, but you have to make sure to follow the instructions in the docstrings of each function. A planar flow is a composition of planar transformation, each defined as:

$$f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u}h(\boldsymbol{w}^\top \boldsymbol{z} + b) \tag{1}$$

where $\boldsymbol{w} \in \mathbb{R}^d$, $\boldsymbol{u} \in \mathbb{R}^d$, $b \in \mathbb{R}$, and h is a smooth non-linear function.

For the question 3, you will make use of the matrix determinant lemma that states:

$$\det(\boldsymbol{A} + \boldsymbol{V}\boldsymbol{W}^T) = \det(I + \boldsymbol{W}^T\boldsymbol{A}^{-1}\boldsymbol{V})\det(\boldsymbol{A}) \tag{2}$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is an invertible matrix and $\boldsymbol{V}, \boldsymbol{W} \in \mathbb{R}^{d\times m}$.

1. **(`unittest`, 1 pts)** Implement the function '`planar_transformation`' in '`solution.py`' by adding the formula for a planar transformation where $h(x) = \tanh(x)$.
2. **(`report`, 2 pts)** Test your function by applying it to samples from a $N(\mathbf{0}, \boldsymbol{I})$ for different value of $\boldsymbol{u}$, $\boldsymbol{w}$, b. Describe what the transformation correspond to and add plots to show the effect of different transformations.
3. **(`report`, 4 pts)** Calculate the log determinant of the Jacobian of a planar transformation using the matrix determinant lemma. What is the complexity (using the big $\mathcal{O}$ notation) of evaluating it?

[4]Extra: Why? Find the answer at: Towards Principled Methods for Training Generative Adversarial Networks

4. **(`unittest`, 2 pts)** Implement the function '`get_logdet`' in '`solution.py`' which calculate the log determinant of the Jacobian of a planar transform.

5. **(`report`, 2 pts)** Let $U_0 \sim N(\mathbf{0}, \boldsymbol{I})$ be the base distribution. Express the induced log density $\ln q_k(\boldsymbol{u}_k)$ obtained by applying a flow constituted of k planar transformation, where $\boldsymbol{u}_k = f_k \circ f_{k-1} \circ \cdots \circ f_1(\boldsymbol{u}_0)$. What is the complexity of evaluating the log determinant of the normalizing flow?

6. **(`unittest`, 4 pts)** Now let's consider the flow $g : \mathcal{X} \to \mathcal{U}$ (Note that this is in the inverse direction compared to the previous question). We have examples $\boldsymbol{x}$ that are sampled from the data distribution P_X and $U \sim N(\mathbf{0}, \boldsymbol{I})$ is the base distribution. Lastly, $P_{g^{-1}(U)}$ is the distribution induced by applying the inverse of the flow to P_U. Complete the function '`loss1`' in '`solution.py`' which calculate the negative log likelihood, i.e. $-\ln P_{g^{-1}(U)}(\boldsymbol{x})$.

7. **(`report`, 4 pts)** Now that the flow is completed, train the model on the dataset '`data_arcs`' and '`data_sine`' with $k = \{2, 8, 32\}$ for 20000 iterations using the function '`launch_experiments_from_samples`' from '`train.py`'. For each setting of k and dataset, plot the learned density (these plots are automatically generated by the code) and report the negative log likelihood. Briefly comment the effect of k.

8. **(`unittest`, 2 pts)** While normalizing flows have to be invertible, planar flows are not easily invertible in practice. Suppose that we now want to generate samples from a given density. Modify the loss function by completing the function '`loss2`' in '`solution.py`'

9. **(`report`, 2 pts)** Train the model on the density '`density_arcs`' and '`density_sine`' with $k = \{2, 8, 32\}$ for 20000 iterations using the function '`launch_experiments_from_density`' from '`train.py`'. For each setting of k, report the plots of the samples generated by the normalizing flow (these plots are automatically generated by the code).

10. **(`report`, 4 pts)** Normalizing flows can also be used as a posterior of variational autoencoders. Write the loss of a VAE using a planar flow. What is the advantage of using a normalizing flow as the posterior of a VAE?