Due Date: April 19th 23:59, 2019
Instructions

- *Read all instructions and questions carefully before you begin.*
- *For all questions, show your work!*
- *Submit your code and your report (pdf) electronically via the course Gradescope page. If you use the Jupyter notebook, please export it as pdf and submit via Gradescope.*
- *You should push your code to a Github repository, and include the link to the repository in your report!*

Problem 1

Generative Adversarial Network (GAN) discriminators estimate a metric between two distributions. In this question, you will implement a discriminator that estimates the Jensen Shannon divergence (JSD) and a critic that estimates the Wasserstein Distance (WD). Then, using your estimators, you will compare the properties the JSD and the WD.

We provide samplers [1] to generate the different distributions you will need for this question. You can use any architecture as long as your function have enough capacity to correctly estimate the distribution. A 3-layers MLP should suffice. You may use SGD with a learning rate of $1e-3$ and a mini batch size of 512.

1. Implement a function to estimate the Jensen Shannon Divergence. Rembember, the JSD is defined as $D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||r) + \frac{1}{2}D_{KL}(q||r)$ where $r(x) = \dfrac{p+q}{2}$. Hence, the objective function that your neural network should optimize is:

$$\arg\max_{\theta} \left\{ \log 2 + \frac{1}{2}\mathbf{E}_{x\sim p}[\log(D_\theta(x))] + \frac{1}{2}\mathbf{E}_{y\sim q}\log(1 - D_\theta(y))] \right\}$$

2. Implement a function to estimate the Wasserstein Distance. Rembember, that the dual form of the WD is defined as $W(p,q) = \sup_{||T||_L \leq 1} \mathbf{E}_{x\sim p}[T(x)] - \mathbf{E}_{y\sim q}[T(y)]$. In order to constrain your function to be 1-lipschitz, we recommend that you use Gradient Penalty. Hence, the objective that your network should optimize is

$$\arg\max_{\theta} \mathbf{E}_{x\sim p}[T_\theta(x)] - \mathbf{E}_{y\sim q}[T_\theta(y)] - \lambda \mathbf{E}_{z\sim r}(||\nabla_z T_\theta(z)||_2 - 1)^2.$$

   r is the distribution over $z = ax + (1-a)y$, where $x \sim p$, $y \sim q$ and $a \sim U[0,1]$. We recommend $\lambda \geq 10$.

[1] See the assignment repository `IFT6135H19_assignment/assignment3/samplers.py`

3. Let $Z \sim U[0,1]$ be a random variable with a uniform distribution. Let p be the distribution of $(0, Z)$ and q_θ be the distribution of (ϕ, Z), where ϕ is a parameter. Plot the estimated JSD and the WD for $\phi \in [-1, 1]$ with intervale of 0.1 (i.e. 21 points). The x-axis should be the value of ϕ and the y-axis should be your estimate.

4. Let f_0 be the density of a 1-dimensional standard Gaussian, and f_1 be the unknown density function of `distribution4`. Train a discriminator D_θ by maximizing the value function

$$\mathbb{E}_{x\sim f_1}[\log D_\theta(x)] + \mathbb{E}_{y\sim f_0}[\log(1 - D_\theta(y))]$$

Estimate the density f_1 using the procedure of Question 5 in the theory part. Plot the discriminator output and the estimated density (using the script `density_estimation.py`).

Problem 2

Variational Autoencoders (VAEs) are probabilistic generative models. This means they can be used to estimate $p(x)$.

Train a VAE on the *Binarised MNIST* dataset, using the negative ELBO loss as shown in class. Each pixel in this dataset is binary: The pixel is either black or white, which means you have to model the likelihood $p(x|z)$, i.e. the decoder, as a bernoulli distribution (this means you should use the binary cross entropy loss for reconstruction). This is not interchangeable with the standard MNIST dataset available on `torchvision`, so either (a) Modify the code provided with this assignment or, (b) download the dataset here: `https://github.com/yburda/iwae/tree/master/datasets/BinaryMNIST`, and implement your own data loader.

Train a VAE (10pts) Train a VAE with a latent variable of 100-dimensions. The following are suggested hyperparameters for training the VAE on Binarized MNIST. You may use the following architecture:

Encoder

Conv2d(1, 32, kernel size=(3, 3))
ELU()
AvgPool2d(kernel size=2, stride=2)
Conv2d(32, 64, kernel size=(3, 3))
ELU()
AvgPool2d(kernel size=2, stride=2)
Conv2d(64, 256, kernel size=(5, 5))
ELU()
Linear(in features=256, out features=(100, 100))
(This final layer should output mean and log-variance)

Decoder

Linear(in features=100, out features=256)
ELU()
Conv2d(256, 64, kernel size=(5, 5), padding=(4, 4))
ELU()
UpsamplingBilinear2d(scale factor=2, mode=bilinear)
Conv2d(64, 32, kernel size=(3, 3), padding=(2, 2))
ELU()
UpsamplingBilinear2d(scale factor=2, mode=bilinear)
Conv2d(32, 16, kernel size=(3, 3), padding=(2, 2))
ELU()
Conv2d(16, 1, kernel size=(3, 3), padding=(2, 2))

This is not meant to be code. Find out the relevant API for the same behaviour in the framework you are familiar with to implement this.

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. Just note that you have to achieve an ELBO of ≥ -96.

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

Evaluating log-likelihood with Variational Autoencoders (20 pts) VAE models are evaluated with log-likelihood, which can be approximated by importance sampling, which was covered during the lecture:

$$\log p(x_i) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p(x_i|z_{ik})p(z_{ik})}{q(z_{ik}|x_i)}; \quad z_{ik} \sim q(z|x_i)$$

1. With M as the size of the *minibatch* evaluated, $K = 200$ as the number of importance samples being used, D as the dimension of the input ($D = 784$ in the case of MNIST), and $L = 100$ as the dimension of the latent variable, implement this importance sampling procedure as a function that:

   - **Given:**
     - Your model.
     - An (M, D) array of x_i's.
     - An (M, K, L). array of z_{ik}'s.
   - **Returns:**
     - $(\log p(x_1), \ldots, \log p(x_M))$ estimates of size $(M,)$

   Display this snippet of code in your report.

   Tip: Watch out for numerical underflow[2] issues (probabilities of images can get really small). Use the LogSumExp[3] trick to deal with this.

2. Report your evaluations of the trained model on the validation and test set using, (a) the ELBO, and (b) the log-likelihood estimate ($\frac{1}{N} \sum_{i=1}^{N} \log p(x_i)$), where N is the size of the dataset.

Problem 3

Recent years have shown an explosion of research into using deep learning and computer vision algorithms to generate images. An ongoing challenge in the image generation community is the question of evaluation. In computer science we often want objective metrics by which to compare our algorithms, but for aesthetic tasks it is not obvious that such a metric even exists. Thus, assessing

[2] the generation of a number that is too small to be represented in the device meant to store it.

[3] `http://blog.smola.org/post/987977550/log-probabilities-semirings-and-floating-point`

the quality of a generative model often involves a combination of qualitative and quantitative evaluations, and evaluation of generative models is an active area of research.

In this final question, we investigate the advantages and disadvantages of two popular generative models in deep learning: GANs and VAEs, using some popular evaluation methods from the literature.

Street View House Numbers (SVHN) The SVHN[4] dataset is made up of clipped images of house numbers from Google Street View. Like MNIST, the goal is to classify digits (0-9), but it is a larger dataset, with color images.

The data is available here: `http://ufldl.stanford.edu/housenumbers/train_32x32.mat` (train set), and `http://ufldl.stanford.edu/housenumbers/test_32x32.mat` (test set). Perform your own split on the training set for a validation set[5].

Alternatively, you may use the download, split, and iterator code provided with this assignment.

Generative models For the assignment, you will have to implement **both** of the following models:

1. Variational Auto-Encoders (VAE):
   Consists of an encoder, $f(x)$, and a decoder $g_{\text{VAE}}(z)$ that returns the mean of the Gaussian likelihood function $p(x|z) = \mathcal{N}(x; g_{\text{VAE}}(z), \mathbf{I})$.
2. Generative Adversarial Networks (GANs):
   Consists of a generator $g_{\text{GAN}}(z)$, and a discriminator $d(x)$.

Note that both models have a $g(z)$, which takes a sample from the prior distribution (in this case the standard Gaussian, $\mathcal{N}(\mathbf{0}, \mathbf{I})$), and produces an image. This is primarily what the qualitative evaluations will be centred around.

Hyperparameters & Training Pointers Train both models with a latent variable dimension of 100. Both models have a generator that takes in a latent code and produces an image, so use the same architecture in both cases.

Beyond those restrictions, you are free to experiment with different architectures and to use one that works. One way to verify that it does is to look at samples from $g(z)$. The following are some pointers to deal with the training of these models:

[4] `http://ufldl.stanford.edu/housenumbers/`

[5] You may use the additional `extra` set that's available if you like, but note that marks will neither be awarded nor deducted if you do.

- For GANS, use the WGAN-GP objective with gradient penalty. you might need to tune the number of times you update the parameters of the discriminator before updating the generator once.
- For VAEs, be careful with the activation you choose to ensure the standard deviation is positive (exponentiation works but might not be optimal for stability).
- Most importantly, there are some metrics that you can monitor throughout training, such as the discriminator's cross entropy loss for GAN and the KL divergence between the prior and approximate posterior for VAE. Make good use of these metrics!

Qualitative Evaluation (25pts) We want to evaluate the "goodness" of the generative models and if the latent space "makes sense". However, these concepts are subjective and hard to quantify. The following are some of the signals that authors include in their paper to demonstrate their model does something "sensible".

For **both** models:

1. **Provide visual samples.** Comment, compare, and contrast the quality of the samples from each model (e.g. blurriness, diversity).
2. **We want to see if the model has learned a disentangled representation in the latent space.** Sample a random z from your prior distribution. 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 $g(z)$). You do not have to show all dimensions, just a couple that result in interesting changes.
3. **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 $\alpha = 0, 0.1, 0.2 \dots 1$ 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 $\alpha = 0, 0.1, 0.2 \dots 1$ plot the samples $\hat{x}_\alpha = \alpha x_0 + (1-\alpha) x_1$.

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

Quantitative Evaluations (25pts) Some of the best generative models don't have a suitable tractable quantitative evaluation metric (they just provide samples). There are many metrics used in the research community to evaluate generated samples.

The Frechet Inception Distance (FID) is a score commonly used in the literature to evaluate GAN generation. In essence, it computes the l_2 distance of the first and the second order moment between the generated and the target representation. In practice, the representation is the coding layer of a neural network pre-trained on a classification task.

More formally, let p be the target distribution and q be the generator's/decoder's distribution. (μ_p, Σ_p) and (μ_q, Σ_q) are the mean and covariance of the "representations" of p and q, respectively. The FID is defined as

$$d^2((\mu_p, \Sigma_p), (\mu_q, \Sigma_q)) = ||\mu_p - \mu_q||_2^2 + \mathrm{Tr}(\Sigma_p + \Sigma_q - 2(\Sigma_p \Sigma_q)^{1/2}) \tag{1}$$

Here, we want to evaluate the samples given from both the GAN and the VAE using FID.

In the `assignment3` directory of the homework repository, find the script `score_fid.py`. Our representations are extracted using a custom classifier trained on SVHN images and their respective labels. The TAs have provided the trained model (`svhn_classifier.pt`), and the relevant functions for loading images (our data loaders load the same image formats that torchvision loads).

1. Using Equation 1, implement the function `calculate_fid_score(.,.)`, which accepts two iterators as arguments: an iterator of the extracted features from the samples, and an iterator of the extracted features from the test set.

   The function should estimate $\mu_p, \Sigma_p, \mu_q, \Sigma_q$ and compute $d^2((\mu_p, \Sigma_p), (\mu_q, \Sigma_q))$.

2. Sample 1000 images from $g(z)$ of both the GAN and the VAE and put them in the a directory e.g. `path/to/sample_directory/samples/`. Ensure that the `samples` directory is the only directory in `path/to/sample_directory/`.

   Then run `python score_fid.py path/to/sample_directory/`. Report the score this script returns.