Due Date: April 15th, 2022, 11 p.m. EST

Instructions

- *Read all instructions and questions carefully before you begin.*
- *For all questions, show your work!*
- *The repository for this homework is* here
- *Submit your answers electronically via Gradescope.*
- *TAs for this assignment are* ***Mohammad-Javad Bayazi*** *and* ***Naga Karthik****.*

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, 4 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, 4 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 $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. Note that $\boldsymbol{\mu}$ and $\log(\boldsymbol{\sigma}^2)$ will be given for Normal distributions.
3. **(unittest, 4 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, 4 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, 4 pts)** Implement the function '`kl_gaussian_gaussian_mc`' in '`q1_solution.py`' to compute KL diveregence $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\right)$ 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

Recent years have shown an explosion of research into using deep learning and computer vision algorithms to generate images. To train a GAN we need to estimate the distance between distributions of real and generated data. In this question, we employ Wasserstein distance and to ensure

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

the Lipschitz-constraint, we will penalize the violation of it as suggested by Petzka et. al. [4]. In this question, you will first implement a function to estimate the Wasserstein distance between two points $x \sim \mu$ and $y \sim \nu$ (real and generated samples respectively):

$$\mathbb{E}_{y\sim\nu}[f(y)] - \mathbb{E}_{x\sim\mu}[f(x)] \tag{1}$$

and with an added regularization term

$$\mathbb{E}_{\hat{x}\sim\tau}[(\max\{0, \|\nabla f(\hat{x})\| - 1\}]^2 \tag{2}$$

then in the network training process minimize:

$$\mathbb{E}_{y\sim\nu}[f(y)] - \mathbb{E}_{x\sim\mu}[f(x)] + \lambda\mathbb{E}_{\hat{x}\sim\tau}[(\max\{0, \|\nabla f(\hat{x})\| - 1\}]^2 \tag{3}$$

where $\hat{x} = tx + (1-t)y$ for $t \sim U[0,1]$.

Dataset & dataloader In this question, you will use the GAN framework train a generator to generate a distribution of images $\mathcal{X} \subset \mathbb{R}^{32\times32\times3}$, namely the **Street View House Numbers** dataset (SVHN) [5]. We will consider the prior distribution $p(z) = \mathcal{N}(\mathbf{0}, I)$ the isotropic gaussian distribution. We provide a function for sampling from the SVHN datasets in ‘q2_samplers.py‘.

Hyperparameters & Training Pointers We provide code for the GANs network as well as the hyperparameters you should be using. We ask you to code the Wasserstein distance and training procedure to train the GANs as well as the qualitative exploration that you will include in your report.

1. **(`unittest`, 4 pts)** Implement the functions ‘`vf_wasserstein_distance`’ and ‘`lp_reg`’ in ‘`q2_solution.py`’ to compute the objective function of the Wasserstein distance and compute the “*Lipschitz Penalty*”. Consider that the norm used in the regularizer is the $L2$-norm.

   Qualitative Evaluation In your report,

2. **(`report`, 8 pts) Provide visual samples.** Comment the quality of the samples from each model (e.g. blurriness, diversity).

3. **(`report`, 8 pts) 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.

[4] See Section 5 of https://arxiv.org/pdf/1709.08894.pdf

[5] The SVHN dataset can be downloaded at http://ufldl.stanford.edu/housenumbers/. Please note that the provided sampler can download the dataset so you do not need to download it separately.

4. (`report`, **8 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 with the two schemes to interpolate between images.

Problem 3

Self-supervised methods learn a representation of data by solving pretext tasks to alleviate expensive supervised labelling. Contrastive learning methods, a sub-category of self-supervised learning (SSL), learn an embedding by minimizing the distance between the embedding of two different views of the same sample while maximizing the distance between the embedding of the view of two different samples. However, non-contrastive SSL methods showed comparable results without using a large number of negative samples. In this question, you will be investigating why these networks do not collapse to a trivial solution by implementing the Simsiam algorithm and experimenting on key factors of its performance.

You will first implement a function to estimate the negative cosine similarity. Then you will implement Simsiam loss and forward functions. In the end, you will run experiments to analyze the performance of the model in different conditions.

Dataset and dataloader for Siamese network [6] In this question, you will work on an object classification task for the **CIFAR10** dataset [7]. This dataset consist of images $\mathcal{X} \subset \mathbb{R}^{32\times32\times3}$ of 10 classes. We provide samplers to generate the different distributions that you will need for this question. In the same repository, we also provide the architecture of a neural network functions f : $\mathcal{X} \rightarrow \mathcal{Z}$ and h : $\mathcal{Z} \rightarrow \mathcal{P}$ s.t. $\mathcal{X} \subset \mathbb{R}^{32\times32\times3}, \mathcal{Z} \subset \mathbb{R}^d, \mathcal{P} \subset \mathbb{R}^d$ (d is the feature dimension, Simsiam default is 2048)).

Hyperparameters & Training Pointers We provide code for the SSL network as well as the hyperparameters you should be using. We ask you to code the training procedure to train the Simsiam as well as the qualitative exploration that you will include in your report.

[6] A Siamese Neural Network is a class of neural network architectures that contain two or more identical subnetworks. (From here)

[7] The CIFAR10 dataset can be downloaded at https://www.cs.toronto.edu/ kriz/cifar.html. Please note that Pytorch CIFAR10 Dataset can download the dataset so you do not need to download it separately.

1. **(`unittest`, 4 pts)** Implement the forward functions of the Simsiam in '`q3_solution.py`'. This function receive two randomly augmented view x_1 and x_2 from an input sample x and compute the outputs of the network, which are as below:

$$z_1 \triangleq f(x_1), \quad p_1 \triangleq h(f(x_1)) \tag{4}$$

$$z_2 \triangleq f(x_2), \quad p_2 \triangleq h(f(x_2)) \tag{5}$$

   Note that you need to perform the grading stopping in this step as well.

2. **(`unittest`, 4 pts)** Implement the function '`cosine similarity`' in '`q3_solution.py`' to compute the similarity between two inputs. The cosine similarity between two variables A and B can be defined as

$$S_c(A, B) = \frac{A}{\|A\|_2} \cdot \frac{B}{\|B\|_2} \tag{6}$$

3. **(`unittest`, 4 pts)** Implement the Simsiam loss functions in '`q3_solution.py`' to compute the objective function of the Simsiam, using the cosine similarity above. Simsiam objective function is defined as:

$$L = 0.5 * \mathcal{D}(p_1, stop - grad(z_2)) + 0.5 * \mathcal{D}(p_2, stop - grad(z_1)) \tag{7}$$

   Where $\mathcal{D}$ is negative cosine similarity.

4. **(`report`, 10 pts)** Train the model for 100 epochs with and without gradient stopping. Plot training loss and Knn accuracy against training epochs.

5. **(`report`, 10 pts)** Investigate the effect of the predictor network (MLP) by experimenting the below setting(s). Plot training loss and KNN accuracy against training epochs.

   (a) Remove the predictor by replacing it with an identity mapping.