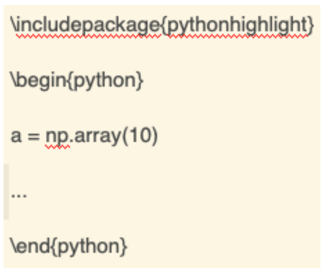
UNIVERSITY OF PENNSYLVANIA

ESE 546: PRINCIPLES OF DEEP LEARNING

HOMEWORK 4

Read the following instructions carefully before beginning to work on the homework.

- You will submit solutions typeset in LaTeX on Gradescope (strongly encouraged). You can use hw_template.tex on Canvas in the "Homeworks" folder to do so. If your handwriting is unambiguously legible, you can submit PDF scans/tablet-created PDFs.
- Clearly indicate the name and Penn email ID of all your collaborators on your submitted solutions.
- Start a new problem on a fresh page and mark all the pages corresponding to each problem. Failure to do so may result in your work not graded completely.
- For each problem in the homework, you should mention the total amount of time you spent on it. This helps us keep track of which problems most students are finding difficult.
- You can be informal while typesetting the solutions, e.g., if you want to draw a picture feel free to draw it on paper clearly, click a picture and include it in your solution. Do not spend undue time on typesetting solutions.
- You will see an entry of the form "HW 1 PDF" where you will upload the PDF of your solutions. You will also see entries like "HW 1 Problem 1 Code" and "HW 1 Problem 3 Code" where you will upload your solution for the respective problems.
- **For each programming problem/sub-problem, you should create a fresh .py file**. This file should contain **all** the code to reproduce the results of the problem/sub-problem, e.g., it should save the plot that is required (correctly with all the axes, title and legend) as a PDF in the same directory. You will upload the .py file as your solution for "HW 1 Problem 3 Code" or "HW 1 Problem 3 Code". Name your file as pennkey_hw1_problem3.py, e.g., I will name my code as pratikac_hw1_problem3.py. Note, we will not accept .ipynb files (i.e., Jupyter notebooks), you should only upload .py files. If you are using Google Colab to do your homework (and I suggest that you don't...), you can export the notebook to a .py file.
- **In addition to submitting the code, you should append the entire Python code for the particular problem to the solution in the PDF. If you are using Latex, you can do something like the screenshot below. The instructors will execute the code to check it. Your code should run without any errors and should create all output/plots required in the problem.**



```
\includepackage{pythonhighlight}

\begin{python}

a = np.array(10)

...

\end{python}
```

Problem 1 (35 points. Variational Auto-Encoder for MNIST. This can also be done on your laptop. Or at least debug on your laptop before running it on Colab.). In this problem we will train a variational auto-encoder for generating MNIST digits. The setup is exactly the same as that of Section 3 in the assigned reading "Auto-Encoding Variational Bayes" (https://arxiv.org/abs/1312.6114) by Kingma & Welling.

(i) **(0 points)** Create the dataset. Use 1000 images of each MNIST digit; this gives a total of 10,000 images. Sub-sample the MNIST images to 14×14 size. You will also binarize the MNIST dataset for this problem: if a pixel is greater than 128 set it to 1, else set it to zero.

(ii) **(15 points)** The encoder will consist of two fully-connected layers, the first has $14 \times 14 = 196$ inputs and 128 outputs (and tanh nonlinearity) and the second layer has 128 inputs and 16 outputs (and no nonlinearity). Therefore, the latent factor $z \in \mathbb{R}^8$ (8 output neurons for the mean and 8 more for the standard deviation). The decoder takes in as input $z \in \mathbb{R}^8$, pushes it through one layer with 128 outputs (and tanh nonlinearity) and then another layer with 196 output neurons (with sigmoid nonlinearity). Note that the final nonlinearity of the decoder should be sigmoid because we want to reconstruct binarized MNIST images. See Sections C1 & C2 in Appendix of the above paper for more details.

Let the parameters of the encoder be u and the parameters of the decoder be v. We will maximize the objective

$$\frac{1}{n} \sum_{i=1}^{n} \ell(u, v; x^i)$$

for images $\{x^1, \dots, x^n\}$ with respect to its parameters u, v where

$$\ell(u, v; x) = \mathop{\mathrm{E}}_{z \sim p_u(z|x)} [\log p_v(x \mid z)] - \mathrm{KL}\left(p_u(z \mid x) \mid\mid N(0, I)\right). \tag{1}$$

and x is one particular MNIST datum. We will set

$$p_u(z \mid x) = N\left(\mu_u(x); \sigma_u(x)^2 I\right)$$

where $\left[\mu_u(x), \sigma_u(x)\right] \in \mathbb{R}^{16}$ with each of them $\mu_u(x), \sigma_u(x) \in \mathbb{R}^8$ is the output of encoder upon feeding the image x. We are modeling $p_u(z \mid x)$ as a Gaussian distribution and the neural network predicts the mean $\mu_u(x)$ of the distribution and the diagonal of the covariance $\sigma_u(x)$. The KL-divergence is

$$\mathrm{KL}\left(p_u(z \mid x) \mid\mid N(0, I)\right) = -\frac{1}{2} \sum_{i=1}^{8} \left(1 + \log\left(\sigma_u(x)_i^2\right) - \mu_u(x)_i^2 - \sigma_u(x)_i^2\right).$$

We will use a Bernoulli distribution to model $p_v(x \mid z)$ because we are using the binarized MNIST dataset:

$$\log p_v(x \mid z) = \sum_{i=1}^{196} x_i \log y_i + (1 - x_i) \log(1 - y_i)$$

where $\mathbb{R}^{196} \ni y =$ output of the decoder.

Train the auto-encoder using the re-parametrization trick for computing the gradient of the u and standard back-prop for computing the gradient of the $\log p_v(x \mid z)$. You should use 2 samples to compute the expectation over $z \sim p_u(z \mid x)$.

(iii) **(5 points)** Plot the first and second term of ELBO in Eq. (1) separately as a function of the number of weight updates.

(iv) **(5 points)** Pick 8 MNIST images, run them through the encoder and the decoder to plot the output of the decoder side-by-side with the original images.

(v) **(5 points)** Sample from the generative model directly: sample $z \in \mathbb{R}^8$ from a standard Gaussian distribution and run the decoder network for this z to synthesize an MNIST image. Plot the synthesized images for a few different samples of z; remember that you can also use a mini-batch of latent variables z to synthesize a few images in one forward-prop of the decoder.

(vi) **(5 points)** We will next compute the validation performance of a generative model. Compute the average validation log-likelihood $\log p(x \mid z)$ for 100 images from the validation set after every 100 weight updates during training. Draw a plot of the reconstruction log-likelihood term in ELBO on the training mini-batches (one point after every weight update) and compare it with the validation log-likelihood as training progresses (one point after every 100 weight updates).