Homework 7 – Deep Neural Networks (CS/DS 541, Whitehill, Spring 2020)

1 Variational Auto-Encoders [50 points]

You may complete this task either individually or in teams up to 2 people.

In this problem you will code (in either TensorFlow or PyTorch) and train a variational auto-encoder (VAE) to synthesize novel data examples. In particular, you will construct a VAE whose encoder and decoders both consist of fully-connected neural networks; then, you will train this VAE on MNIST and use it to to generate novel images of handwritten digits.

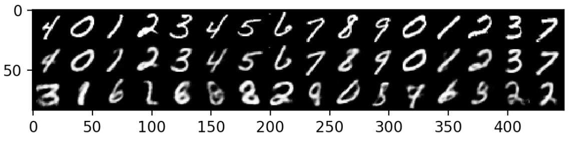
Details: The encoder Q_ϕ should take a vector in $\mathbb{R}^{784}$ as input and output the parameters of a p-dimensional normal (Gaussian) distribution with a mean μ and diagonal covariance matrix specified by vector $[\sigma_1^2, \dots, \sigma_p^2]$. In other words, Q_ϕ should produce $2p$ total outputs. Of course, each of the elements of the covariance matrix must be non-negative. To enforce positivity on the output of Q_ϕ, you can either square the output units or exponentiate them. The decoder P_θ should take a p-dimensional vector as input and output a vector in $(0,1)^{784}$ using a final layer of 784 independent logistic sigmoid units (*not* a softmax over 784 outputs!).

Implementation and training: Implement Algorithm 1 from the paper `https://arxiv.org/pdf/1312.6114.pdf`; the loss function is given in Equation (10) in their paper. Note that the $\frac{1}{L}$ constant attached to the second term of the loss function is really just another hyperparameter; it weights how much to encourage the encoder to output a Gaussian distribution versus accurately reconstruct the training image. **This hyperparameter is very important** and you will likely need to experiment with different values to obtain satisfactory generated examples. For training data, use the `mnist_train_images.npy` file from previous assignments.

Hints:

- My encoder consisted of two fully-connected layers (the first projected 784 down to 200, and the second projected from 200 to 32 dimensions for the latent code) with a tanh activation function and dropout. My decoder implemented the reverse process but without dropout. I trained with Adam and a mini-batch size of 100 for 150 epochs, using a sum over 784 independent log-loss terms for the reconstruction loss.
- The hyperparameter that weights how much to favor the reconstruction loss versus the KL-divergence loss is critical. Try playing with it to get a feel of how it works, e.g., set the weight for the KL-divergence loss to 0 and make sure you can reconstruct the images almost perfectly. At this point, your *new* images will probably look terrible, but at least you have part of the system working. Then, gradually increase it (relative to the reconstruction loss) to see if you can generate more realistic new images.
- Work on a very small training set (e.g., 100 images) and a very simple architecture (small number of hidden units and small dimensionality p) at first to make sure you can at least get the reconstructions to be accurate. Then increase the training set size, along with the capacity of the network, to obtain more realistic new images.

Deliverables: Beside your Python code, you should also include an image in your PDF showing at least 16 examples of each of the following: (1) Some **existing** MNIST images $\{\mathbf{x}^{(i)}\}_{i=1}^{16}$; (2) The corresponding **reconstructions** of the existing images, i.e., $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^{16}$, where each $\hat{\mathbf{x}}^{(i)} = P_\theta(Q_\phi(\mathbf{x}^{(i)}))$; and (3) Some **new** images obtained by sampling a Gaussian noise vector $\mathbf{z}$ and passing it through just the decoder P_θ. Here is an example:



In the example above, the first row shows some existing MNIST samples; the second row shows their reconstructions with a trained VAE; and the last row shows some novel examples. Your images do *not* have to be beautiful to obtain full credit – just make sure your VAE can generate images that clearly capture most of the 10 different digit classes. (We will be liberal with grading in this regard.)

Important note: You should implement a VAE from scratch using only PyTorch or TensorFlow. You may **not** look at anyone's full implementation of a VAE or borrow any of its code. However, you are permitted to read through online tutorials about VAEs if you wish (provided they do not spell out the entire solution line by line).

Submission: In addition to your Python code (`homework7_WPIUSERNAME1.py` or `homework7_WPIUSERNAME1_WPIUSERNAME2.py` for teams), create a PDF file (`homework7_WPIUSERNAME1.pdf` or `homework7_WPIUSERNAME1_WPIUSERNAME2.pdf` for teams) containing the screenshots described above. **Please submit both the PDF and Python files in a single Zip file.**