

# Homework 7 – Deep Neural Networks (CS/DS 541, Whitehill, Spring 2021)

You may complete this homework either individually or in teams up to 3 people.

## 1 Variational Auto-Encoders [40 points]

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 generate novel images of handwritten digits.

**Details:** The encoder  $Q_\phi$  should take a vector in  $\mathbb{R}^{784}$  as input and output the parameters of a  $p$ -dimensional normal (Gaussian) distribution with a mean  $\mu$  and diagonal covariance matrix specified by vector  $[\sigma_1^2, \dots, \sigma_p^2]$ . In other words,  $Q_\phi$  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  $Q_\phi$ , you can either square the output units or exponentiate them. The decoder  $P_\theta$  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:** Download the MNIST training set from [https://s3.amazonaws.com/jrwprojects/mnist\\_train\\_images.npy](https://s3.amazonaws.com/jrwprojects/mnist_train_images.npy). 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 important** and you will likely need to experiment with different values to obtain satisfactory generated examples.

### 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 important. 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  $P_\theta$ . 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.)

## 2 Generative Adversarial Networks [40 points]

In this problem you will code (in either TensorFlow or PyTorch) and train a generative-adversarial network (GAN) to synthesize new images of the MNIST dataset, and show some example images that your trained network produces.

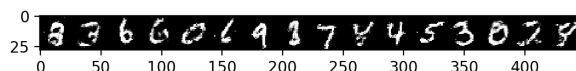
**Details:** The generator  $G$  should take a Gaussian noise vector as input and produce a vector in  $[0, 1]^{784}$  as output. The discriminator  $D$  should take a vector in  $\mathbb{R}^{784}$  as input and produce the probability  $\hat{y} \in [0, 1]$  of the image being real. Hence, both  $G$  and  $D$  should have sigmoid units in their output layers.

**Implementation and training:** Implement Algorithm 1 from the paper <https://arxiv.org/pdf/1406.2661.pdf>.

**Hints:**

- My own  $G$  consisted of 4 fully-connected layers, and between each pair of neighboring fully-connected layers was a ReLU non-linearity and batch normalization. The output layer of  $G$  consisted of 784 logistic sigmoids.
- In my  $D$ , the first layer was a dropout layer (as described in class) followed by three fully-connected layers. There was also a ReLU non-linearity and batch normalization between each pair of neighboring fully-connected layers.  $D$  ended with a single logistic sigmoid unit. The number of neurons in the hidden layers of both  $G$  and  $D$  was 256. Both  $G$  and  $D$  were optimized using Adam.

**Deliverables:** Beside your Python code, you should also include an image in your PDF showing at least 16 examples of images generated by your trained GAN. Here is an example:



Your images do *not* have to be beautiful to obtain full credit – just make sure your GAN can generate images that clearly capture most of the 10 different digit classes. (We will be liberal with grading in this regard.)

## 3 Rules

**Important note:** You should implement a VAE and a GAN from scratch using only PyTorch or TensorFlow. You may **not** look at anyone's full implementation of a VAE/GAN 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.