Exercise III

AMTH/CPSC 663b - Spring semester 2022

Published: Thursday, March 17, 2022
Due: Friday, April 8, 2022 - 11:59 PM

Compress your solutions into a single zip file titled `<lastname and initials>_assignment3.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment3.zip`. Include a single PDF titled `<lastname and initials>_assignment3.pdf` and any Python scripts specified. If you complete your assignment in a Jupyter notebook, please submit the notebook along with a PDF version of the notebook. Any requested plots should be sufficiently labeled for full points.

Programming assignments should use built-in functions in Python and PyTorch; In general, you may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Problem 1

In problem set 1, we found solutions to least-squares regression using the closed form solution to

$$\|Xw - y\|_2^2$$

Where X is the data, y is the targets, and w are the weights/coefficients. Sometimes when developing a model, it may be a good idea to check that the gradients determined by PyTorch are what we expect. We will now do this in a linear regression setting.

1. First, show that
$$\nabla_w \|Xw - y\|_2^2 = 2X^T Xw - 2X^T y$$
2. Using PyTorch's autograd functionality, check that the gradient (at the initial weights) with respect to w is the same as what we expect from the closed form solution of the gradient above. To do this, complete `p1.py` and include the finished version with your submission.

Problem 2

1. It's tempting to use gradient descent to try to learn good values for hyper-parameters such as λ and η. Can you think of an obstacle to using gradient descent to determine λ? Can you think of an obstacle to using gradient descent to determine η?

2. L2 regularization sometimes automatically gives us something similar to the new approach to weight initialization (i.e., we initialize the weights as Gaussian random variables with mean 0 and standard deviation $1/\sqrt{n_{in}}$ where n_{in} is the number inputs to a neuron). Suppose we are using the old approach to weight initialization (i.e., we initialize the weights as Gaussian random variables with mean 0 and standard deviation 1). Sketch a heuristic argument that:

   (a) Supposing λ is not too small, the first epochs of training will be dominated almost entirely by weight decay.

   (b) Provided $\eta\lambda \ll n$ the weights will decay by a factor of $\exp(-\eta\lambda/m)$ per epoch, where n is the is the total number of training examples, and m is the mini-batch size.

   (c) Supposing λ is not too large, the weight decay will tail off when the weights are down to a size around $1/\sqrt{n}$, where n is the total number of weights in the network.

Problem 3

1. Autoencoders learn to recreate their input after passing it through a sequence of layers in a neural network. This goal can be satisfied by simply learning the identity function. What design choices can be made (e.g. about the network architecture) to prevent them from learning the identity function? Pick at least two and discuss why an autoencoder with your design choice would be more useful than one without it.

2. In class we discussed denoising autoencoders that learn to correct a corruption process that we model with $C(\hat{x}|x)$. Describe an example of such a corruption process, express it in a formula, and provide a plot of an original two-dimensional dataset (e.g. samples from $y = x^2$ or anything of your choosing) and the same dataset after you've corrupted it with your function C.

3. Build an autoencoder that embeds the MNIST dataset into two dimensions (tip: start by modifying your previous feedforward MNIST classifier). You can experiment with the details of the architecture, but a good baseline is an encoder that uses three layers to compress the images into two dimensions, followed by a three-layer decoder that undoes this compression.

   After you've trained your model and have obtained reasonable reconstruction accuracy, obtain the 2-dimensional embeddings of a batch of images, and plot with colors indicating the label of each image. Describe what you can learn about this dataset from looking at the points embedded into the latent space of the autoencoder.

Problem 4

1. Describe the differences between Kullback-Leibler divergence, Jensen-Shannon divergence, MMD, and Wasserstein distance.

2. Using the skeleton code in vae.py, fill in the TODOs to build a VAE for the MNIST dataset. The code returns a sample of 8 different inputs and their reconstructions for each epoch. Include the reconstruction sample for the first and last epoch in your report. Submit your filled-in version of vae.py along with the assignment.

3. Retrain your VAE on the *Fashion MNIST* dataset by un-commenting lines 37-43 in vae.py. Fashion MNIST was designed as a more complex replacement for MNIST. Like MNIST, it contains 70,000 grayscale images of 28 by 28 pixels, but these are divided between ten classes of fashion accessories - sandals, handbags, pullovers – whose variety poses a greater challenge than handwritten digits. Play with the hyperparameters of your VAE to try to generate the best images possible. Include these in your report; they will be needed for comparison in Problem 5.

Problem 5

1. Use the skeleton code in gan.py to build a functional GAN. The Generator and Discriminator classes have been provided, but you'll need to implement a training routine for these classes by filling in the methods `train_generator` and `train_discriminator`.

2. Run your GAN on the Fashion MNIST dataset. Note: this will take 10-30 minutes or longer if your computer lacks an NVIDIA GPU. Google COLAB provides free GPU enabled runtimes, and could be a good resource.

3. Experiment with the hyperparameters of your GAN to try to produce the best-quality images possible. What happens if you use different learning rates for the discriminator and generator, or if you train the generator multiple times for every iteration of the discriminator? Describe the best training scheme, and any problems you encountered during training. Include sample generations in your report.

4. The standard GAN is an unsupervised model, but the Conditional GAN provides the generator and discriminator with image labels. This allows a Conditional GAN to generate images from a specific class on demand, and enables the discriminator to penalize generated images which deviate from their labels. Follow the TODOs in the Generator and Discriminator classes to turn your GAN into a Conditional GAN.

5. Run your Conditional GAN on Fashion MNIST. How do these generated images compare to your initial GAN, both in quality and in the relative abundance of each class?

6. Describe the theoretical differences between a generative adversarial network (GAN) and a variational autoencoder (VAE). What differences did you notice in practice? What tasks would be more suited for a VAE, and what tasks require a GAN?

7. Why do we often choose the input to a GAN (z) to be samples from a Gaussian? Can you think of any potential problems with this?

8. In class we talked about using GANs for the problem of mapping from one domain to another (e.g. faces with black hair to faces with blond hair). A simple model for this would learn two generators: one that takes the first domain as input and produces output in the second domain as judged by a discriminator, and vice versa for the other domain. What are some of the reasons the DiscoGAN/CycleGAN perform better at this task than the simpler model?

References

[1] "The scipy stack specification¶." [Online]. Available: https://www.scipy.org/stackspec.html