CSCI 5525: Advanced Machine Learning (Fall 2023)

Homework 4

(Due Tue, Nov. 21, 11:59 PM CST)

1. **(20 points)** In this problem, we consider principal component analysis (PCA) for dimensionality reduction. We will apply PCA to the MNIST dataset to reduce the dimensionality of each image from 784 to 2 and visualize the images in the lower-dimensional subspace.

   Write Python code to implement PCA in the class `MyPCA`. Apply your PCA to all the training data to learn the projection vector. Then, project each image in the training dataset to $\mathbb{R}^2$. Plot[1] each projected image as a point in a 2-dimensional figure where the x-axis is principal component 1 and the y-axis is principal component 2. Color each point in the figure according to the label of each image so we can visualize how well PCA reduced the dimensionality according to how closely the images are grouped by label. (Note, only use the labels to visualize how well the autoencoder did, do not use the labels anywhere else.) What are your observations from the figure? Where did PCA reduce the dimensionality well and where did it not do well? Note, you may get some complex numbers when computing the eigenvalues and eigenvectors. You can use `np.real()` to only keep the real part.

   In addition to `MyPCA.py` (details below), add your code to `hw4_q1.py` and use it to test. Please write up your results (2-d plot) and submit them in a PDF document.

   `MyPCA.py` is a class which must have the following:

```
class MyPCA:

    def __init__(self, num_reduced_dims):
        ...
    def fit(self, X):
        ...
    def project(self, x):
        ...
```

   For the class, the `__init__(self, num_reduced_dims)` function takes as input `num_reduced_dims` which is the number of dimensions in the lower-dimensional space to project the data points (i.e., m from the lecture notes).

   For `fit(self, X)`, the input `X` are the training dataset images. The fit function learns the projection matrix consisting of the principal components.

   For `project(self, x)`, the input `x` is an image from the training dataset. Your `predict` method should project the images to the lower-dimensional space and return the projected image.

[1] We provide a plotting function in `hw4_utils.py`, you can use that if you'd like.

2. **(35 points)** In this problem, we consider autoencoders for dimensionality reduction. We will apply an autoencoder to the MNIST dataset to reduce the dimensionality of each image from 784 to 2 and visualize the images in the lower-dimensional subspace.

   Write Python code to implement an autoencoder using PyTorch in the class `MyAutoencoder`. The autoencoder must have the following architecture:

   Encoder:

   - Input: flattened image of size 784
   - Fully-connected linear layer 1: input with bias; number of output nodes: 400 nodes
   - Tanh activation function
   - Fully-connected linear layer 2: number of input nodes: 400 and bias; number of output nodes: 2
   - Tanh activation function

   Decoder:

   - Fully-connected linear layer 3: number of input nodes: 2 and bias; number of output nodes: 400
   - Tanh activation function
   - Fully-connected linear layer 4: number of input nodes: 400 and bias; number of output nodes: 784
   - Sigmoid activation function
   - Use mean squared error (MSE) as loss function, i.e., $\frac{1}{n}\sum_i \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$ where n is the number of images in the mini batch, $\mathbf{x}_i$ is the ith flattened image vector and $\hat{\mathbf{x}}_i$ is your network's output for the ith image. Do not use the labels.

   Train your autoencoder on all the training data using the Adam optimizer with a learning rate of 0.001 and mini batches of size 10 images until convergence (you must decide when this happens). During training, print out the cumulative training loss at the end of each epoch. Also, create a plot of the cumulative training loss at the end of each epoch. (Your plot should have the number of epochs on the x-axis and loss on the y-axis.) After training, pass a random set of 1000 images from the training set through the encoder and store the output (this should be a 2-dimensional vector for each image). Plot each reduced image as a point in a 2-dimensional figure where the x-axis is dimension 1 (encoder hidden node 1) and the y-axis is dimension 2 (encoder hidden node 2). Color each point in the figure according to the label of each image so we can visualize how well the autoencoder reduced the dimensionality according to how closely the images are grouped by label. (Note, only use the labels to visualize how well the autoencoder did, do not use the labels anywhere else.) What are your observations from the figure? Where did the autoencoder reduce the dimensionality well and where did it not do well?

   In addition to `MyAutoencoder.py`, add your code to `hw4_q2.py` and use it to test. Please write up your results and plots for the Autoencoder (training loss and 2-d plot) and submit them in a PDF document. Note, you are free to design your `MyAutoencoder.py` class however you want.

3. **(45 points)** In this problem, we will consider generative adversarial networks (GANs) to learn the MNIST data distribution and generate realistic-looking images of handwritten digits.

   Write Python code to implement a GAN using PyTorch in two classes `MyGenerator` and `MyDiscriminator`. The GAN must have the following architecture.

   MyGenerator:

   - Input: 128-dimensional noise vector $\mathbf{z}$ sampled from a Gaussian distribution with mean $\mathbf{0}$ and identity covariance matrix
   - Fully-connected linear layer 1: input with bias; number of output nodes: 256 nodes
   - ReLU activation function
   - Fully-connected linear layer 2: number of input nodes: 256 and bias; number of output nodes: 512
   - ReLU activation function
   - Fully-connected linear layer 3: number of input nodes: 512 and bias; number of output nodes: 1024
   - ReLU activation function
   - Fully-connected linear layer 4: number of input nodes: 1024 and bias; number of output nodes: 784
   - Tanh activation function

   MyDiscriminator:

   - Input: 784-dimensional vector (real or fake flattened image)
   - Fully-connected linear layer 1: input with bias; number of output nodes: 1024 nodes
   - ReLU activation function
   - Dropout layer with probability $p = 0.30$
   - Fully-connected linear layer 2: number of input nodes: 1024 and bias; number of output nodes: 512
   - ReLU activation function
   - Dropout layer with probability $p = 0.30$
   - Fully-connected linear layer 3: number of input nodes: 512 and bias; number of output nodes: 256
   - ReLU activation function
   - Dropout layer with probability $p = 0.30$
   - Fully-connected linear layer 4: number of input nodes: 256 and bias; number of output nodes: 1
   - Sigmoid activation function

Following the algorithm described in lecture, train your generator and discriminator on all the training data using the Adam optimizer with a learning rate of 0.0002, mini batches of size $m = 128$ images, and $k = 1$ steps to update the discriminator (before updating the generator once) until training has convergence for both the generator and discriminator (you must decide when this happens). Note, we want to train the generator and discriminator separately so initialize two Adam optimizers, one for the generator and one for the discriminator, and use the respective optimizer to update the networks.

Since the discriminator is solving a binary classification problem (real vs fake images), we can use the cross entropy loss function to train the discriminator. Similarly, since the generator is trying to fool the discriminator, e.g. have the discriminator predict 1 for fake images (where 1 means the discriminator is predicting the image is real and 0 means the discriminator is predicting the image is fake), we can flip the labels when computing the loss for the generator and use cross entropy loss to train the generator. In other words, use the generator to generate a fake image, feed the fake image into the discriminator to get the discriminator's prediction, then input the discriminator's prediction and a label of 1 (even though the image is fake we want to flip the label since we're training the generator) to the cross entropy loss for the generator and use the loss to update the generator network.

During training, print out the cumulative training loss of both the generator and discriminator at the end of each epoch. Also, create a plot of the cumulative training loss at the end of each epoch for both the generator and discriminator. (Your plot should have the number of epochs on the x-axis and loss on the y-axis with two curves one for each the generator and discriminator.) Also, at the end of each epoch, generate 5 handwritten digits from scratch using the generator and plot them. Once training has converged, generate a final 5 handwritten digits from scratch using the generator and plot them. What digits did your generator plot? How do the generate images change from epoch 1 until convergence? Are the results better, worse, or the same?

In addition to `MyGenerator.py` and `MyDiscriminator.py`, add your code to `hw4_q3.py` and use it to test. Please write up your results and include all plots for your GAN (training loss and all generated digits for each epoch and final generated digits) and submit them in a PDF document. Note, you are free to design your `MyGenerator.py` and `MyDiscriminator.py` classes however you want.

Instructions

You must complete this homework assignment individually. You may discuss the homework at a high-level with other students but make sure to include the names of the students in your README file. You may not use any AI tools (like GPT-3, ChatGPT, etc.) to complete the homework. Code can only be written in Python 3.6+; no other programming languages will be accepted. One should be able to execute all programs from the Python command prompt or terminal. Make sure to include a requirements.txt, yaml, or other files necessary to set up your environment. Please specify instructions on how to run your program in the README file.

Each function must take the inputs in the order specified in the problem and display the textual output via the terminal and plots/figures, if any, should be included in the PDF report.

In your code, you can only use machine learning libraries such as those available from scikitlearn as specified in the problem description. You may use libraries for basic matrix computations and plotting such as numpy, pandas, and matplotlib. Put comments in your code so that one can follow the key parts and steps in your code.

To use PyTorch, you must install both PyTorch and Torchvision. Follow the installation instruction at `https://pytorch.org/get-started/locally/` and `https://pypi.org/project/torchvision/`. We recommend using Anaconda but you can also use the following commands to install PyTorch and Tortchvision in your environment: `pip install pytorch` and `pip install torchvision`. The implementation must be for the CPU version only (no GPUs or MPI parallel programming is required for this assignment).

You can use Google Colab[2] to write and test your code but make sure to submit only `.py` files but code must not require a GPU.

Follow the rules strictly. If we cannot run your code, you will not get any credit.

- **Things to submit**
  1. `[YOUR_NAME]_hw4_solution.pdf`: A document which contains solutions to all problems.
  2. `hw4_q1.py` and `MyPCA.py`: Code for Problem 1.
  3. `hw4_q2.py` and `MyAutoencoder.py`: Code for Problem 2.
  4. `hw4_q3.py`, `MyGenerator.py`, and `MyDiscriminator.py`: Code for Problem 3.
  5. README.txt: README file that contains your name, student ID, email, instructions on how to run your code, any assumptions you are making, and any other necessary details.
  6. Any other files, except the data, which are necessary for your code.

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems at a high level only. Each person must write up the final solutions individually. You need to list in the README.txt which problems were a collaborative effort and with whom. Please refer to the syllabus for more details. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help online,
- Look up things/post on sites like Quora, StackExchange, etc.

[2]`https://colab.research.google.com/`