CSCI 5525: Advanced Machine Learning (Fall 2023)

Homework 3

(Due Thu, Nov. 2, 11:59 PM CDT)

1. **(15 points)** Consider the convolutional neural network architecture given in Figure 1 for classifying MNIST digits. Assume that the input images are reduced to size 10×10 with only 1 channel (represented as a matrix in $\mathbb{R}^{10\times 10}$). In this architecture, the convolutional layer uses a 3×3 filter, $\mathbf{W}_{conv}$, with stride 3 and zero padding of size 1. The dimensions of the outputs of each layer are shown below.

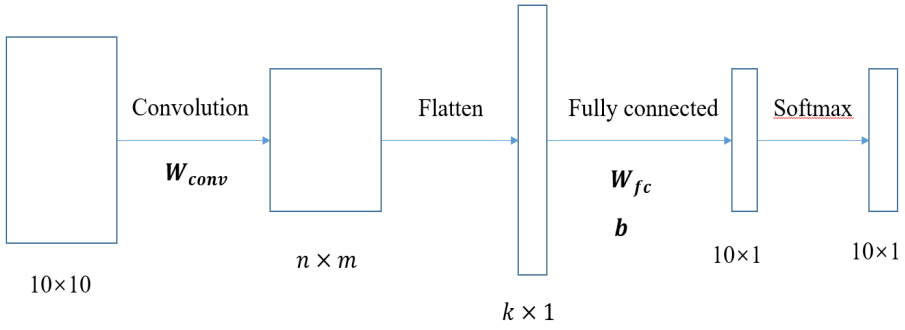

Figure 1: A toy CNN

   (a) **(7 points)** What are the values of n, m, and k in the graph?

   (b) **(8 points)** What are the sizes of $\mathbf{W}_{conv}$, $\mathbf{W}_{fc}$, and $\mathbf{b}$?

2. **(40 points)** Here we will consider multiclass classification using a multilayer perceptron (MLP). We will classify the `MNIST`[1] dataset where each data point is an image of a handwritten digit and its label indicates the value of the digit written $(0, 1, \ldots, 9)$. We will be using PyTorch[2] to implement the MLP.

   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 Torchvision 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).

   Write Python code to implement a multilayer perceptron (MLP) from scratch using PyTorch in the class `MyMLP`. The multilayer perceptron must have the following architecture:

[1] `https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html`
[2] `https://pytorch.org/`

- Input: 1-channel image of size 28x28 pixels
- Fully-connected linear layer 1: input image with bias; output with 128 nodes
- ReLU activation function
- Fully-connected linear layer 2: input from layer 1 with bias; output with 10 nodes
- Softmax activation function on output layer (note, you do not need to specify this anywhere, it is included when using cross entropy loss `torch.nn.CrossEntropyLoss()`).

We will experiment with the following 4 gradient descent optimizers: vanilla stochastic gradient descent, Adagrad, RMSprop, and Adam. These correspond to the Pytorch functions `torch.optim.{SGD, Adagrad, RMSprop, Adam}`. We will also experiment with learning rates $\eta \in \{1e-5, 1e-4, 1e-3, 0.01, 0.1\}$. Make sure to set the learning rate when initializing the gradient descent optimizer object and use default values for all other parameters in the optimizer. For each gradient descent optimizer and learning rate, train an MLP for N epochs (you must decide N) on all the training data using mini batches of size 32 images and cross entropy loss. During training, print out the cumulative training loss and training error rate at the end of each epoch. Once training has converged (you must decide when this happens), apply your learned model to the test set. Print out the cumulative test loss and test error rate on the test set. Which gradient descent optimizer and learning rate are the best? In addition to `MyMLP.py` (details below), add your code to `hw3_q2.py` and use it to test.

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

```
class MyMLP:

   def __init__(self, input_size, hidden_size, output_size, learning_rate, max_epochs):
       ...
   def fit(self, train_loader, criterion, optimizer):
       ...
   def predict(self, test_loader, criterion):
       ...
   def forward(self, X):
       ...
```

For the class, the `__init__(self, input_size, hidden_size, output_size, learning_rate, max_epochs)` function takes as input `input_size` which is the number of pixels in an image (set to 28*28), `hidden_size` which is the number of hidden nodes (set to 128), `output_size` which is the number of output nodes (i.e., number of classes in the dataset), `learning_rate` which is the gradient descent learning rate η, and `max_epochs` which is the maximum number of epochs to run during training (i.e., number of passes through the dataset).

For `fit(self, train_loader, criterion, optimizer)`, the input `train_loader` contains both the feature matrix and class labels, `criterion` is the cross entropy loss (e.g., `criterion = torch.nn.CrossEntropyLoss()`), and `optimizer` is the gradient descent algorithm variant (e.g., `torch.optim.SGD()`, `torch.optim.Adagrad()`, etc.) used to minimize the error. You must also train your algorithm for `max_epochs` epochs where an epoch is a single pass through

all the training data. Your fit method must return two arrays one for the training loss and one for the training error rate for each epoch.

For `predict(self, test_loader, criterion)`, the input `test_loader` includes the feature matrix and class labels corresponding to the test set, and `criterion` is as described above. Your `predict` method must return the test set loss and error rate.

For `forward(self, X)`, the input `X` are images from `train_loader` or `test_loader`. The forward function implements the forward pass of the MLP and should pass the data through the network and return the output of the network (i.e., the value of each of the 10 output nodes for each image).

You will likely need to use the following functions (among others):

- `nn.Linear()`,
- `nn.ReLU()`,
- `loss = criterion()`,
- `optimizer.zero_grad()`,
- `loss.backward()`,
- `optimizer.step()`,

see the provided code for additional details.

Please write up your results for the MLP (only test loss and error rates) and submit them in a PDF document. Make a table to present the results, for example:

Test Set Loss/Error Rate for SGD					
	$\eta = 1e-5$	$\eta = 1e-4$	$\eta = 1e-3$	$\eta = 0.01$	$\eta = 0.1$
Loss	#	#	#	#	#
Error Rate	#	#	#	#	#

3. **(45 points)** Here we will use a convolutional neural network (CNN) to classify the `MNIST` dataset.

   Write Python code to implement a CNN using PyTorch in the class `MyCNN`. The CNN must have the following architecture:

   - Input: 1-channel input image of size 28x28 pixels
   - Convolution layer: Convolution kernel of size (3, 3) with a stride of 1, dilation of size 1, and bias. Number of input channels: 1; Number of output channels: 20; Do not use any padding
   - ReLU activation function
   - Max pool: 2x2 kernel size with stride of 2
   - Dropout layer with probability $p = 0.50$
   - Flatten input for fully-connected layers

- Fully-connected layer 1. Input is flattened output from conv layer with bias; Number of output nodes: 128
- ReLU activation function
- Dropout layer with probability $p = 0.50$
- Fully-connected layer 2: Number of input nodes: 128 and bias; Number of output nodes: 10
- Softmax activation function on output layer (note, you do not need to specify this anywhere, it is included when using cross entropy loss `torch.nn.CrossEntropyLoss()`).

Train your CNN for N epochs (you must decide N) on all the training data using vanilla SGD (e.g., `torch.optim.SGD()`) as the optimizer with mini batches of size 32 images and cross entropy loss (e.g., `torch.nn.CrossEntropyLoss()`). During training, print out the cumulative training loss and training error rate at the end of each epoch. Once training has converged (you must decide when this happens), apply your learned model to the test set. Print out the test loss and test error rate on the test set. Also, randomly select 5 images which your model incorrectly predicted and plot the images. What is the correct label and what is the label your network predicted for each of the 5 images? Why do you think your model misclassified these images?

In addition to `MyCNN.py` (details below), add your code to `hw3_q3.py` and use it to test.

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

```
class MyCNN:

   def __init__(self, input_size, output_size, kernel_size, stride_size, max_pool_size,
learning_rate, max_epochs):
       ...
   def fit(self, train_loader, criterion, optimizer):
       ...
   def predict(self, test_loader, criterion):
       ...
   def forward(self, X):
       ...
```

For the `__init__` function, the parameter `kernel_size` is the size of the kernel in both directions (e.g., if the kernel is 2×2, then the size is set to 2). The parameters `stride_size` and `max_pool_size` are similar. The other functions inputs are similar to those for your MLP class in Problem 2. You will likely need to use the following functions (among others):

- `nn.Conv2d()`,
- `nn.MaxPool2d()`,
- `nn.Dropout()`,

see the provided code for additional details. Please write up your results and plots for the CNN (training/test loss and error rates and misclassified images) and submit them in a PDF document.

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 scikit-learn 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.

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

- **Things to submit**

  1. `[YOUR_NAME]_hw3_solution.pdf`: A document which contains solutions to all problems.
  2. `hw3_q2.py` and `MyMLP.py`: Code for Problem 2.
  3. `hw3_q3.py` and `MyCNN.py`: Code for Problem 3.
  4. 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.
  5. 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 on online,
- Look up things/post on sites like Quora, StackExchange, etc.