CS5785 / ORIE5750 / ECE5414 Homework 5

This homework is due on **Monday, November 20th, 2023 at 11:59PM ET**. Upload your homework to Gradescope (Canvas->Gradescope). The homework is split into programming exercises and written exercises. Upload your homework to Gradescope. Your submission will have two parts:

1. A write-up as a single `.pdf` file. Submit this under the `hw5-report` assignment in Gradescope.
2. Source code and data files for all of your experiments (AND figures) in `.ipynb` files (file format for IPython Jupyter Notebook). These files should be placed in a folder titled `hw5` and uploaded to the `hw5-code` assignment in Gradescope.

The write-up should contain a general summary of what you did, how well your solution works, any insights you found, etc. On the cover page, include the class name, and homework number. You are responsible for submitting clear, organized answers to the questions. You could use online LaTeX templates from Overleaf, under “Homework Assignment” and “Project / Lab Report”.

Please include all relevant information for a question, including text response, equations, figures, graphs, output, etc. If you include graphs, be sure to include the source code that generated them. Please pay attention to Canvas for announcements, policy changes, etc. and Piazza for homework related questions.

If you need help

There are several strategies available to you.

- If you get stuck, we encourage you to post a question on Piazza. That way, your solutions will be available to other students in the class.
- The professor and TAs offer office hours, which are a great way to get some one-on-one help.
- You are allowed to use well known libraries such as `scikit-learn`, `scikit-image`, `numpy`, `scipy`, etc. for this assignment (including implementations of machine learning algorithms), unless we explicitly say that you cannot in a particular question. Any reference or copy of public code repositories should be properly cited in your submission (examples include Github, Wikipedia, Blogs).

PROGRAMMING EXERCISES

Multilayer Perceptrons and Convolutional Neural Networks

In this programming exercise, we will work with the Fashion MNIST dataset. This dataset contains 70,000 28x28 grayscale images of fashion products from 10 categories from a dataset of Zalando article images, with 7,000 images per category. The training set consists of 60,000 images and the test set consists of 10,000 images. You need to recognize the category of these fashion products using MLP and CNN.

1. **Loading Dataset** For using this dataset, you will need to import FashionMNIST from Pytorch and load it as follows.

```
import torch
from torchvision import datasets, transforms

transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5, ),
   (0.5,))])

trainset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/',
   download=True, train=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size
   =32, shuffle=True)
testset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/',
   download=True, train=False, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=32,
   shuffle=False)
```

   Here, we are using Pytorch transforms to preprocess the dataset such that all images have a mean and stddiv of 0.5. The *trainloader* and *testloader* are two iterators for you to iterate through the training and test sets. Note that here we are setting a *batch_size* of 32, but you can change it to other values.

   To verify that you have loaded the dataset correctly, try printing out the shape of one batch in your training and test dataset matrices. For example, you can print out the batch dimension for the training dataset:

```
batch = next(iter(trainloader))
print(batch[0].shape, batch[1].shape)
```

   You will see "torch.Size([32, 1, 28, 28])" and "torch.Size([32])". The "batch" variable is a list of two items – 32 input images and their corresponding 32 labels.

You may also visualize some images:

```
import matplotlib.pyplot as plt
f, ax = plt.subplots(2, 5)
plt.subplots_adjust(bottom=0.3, top=0.7, hspace=0)
for i in range(2):
    for j in range(5):
        image, label = next(iter(trainloader))
        ax[i][j].set_axis_off()
        ax[i][j].imshow(image[0,0,:], cmap='gray')
```

Here we are selecting the first image of 10 batches. The "next (iter(trainloader))" will return the next batch of the training dataset.

2. **Multilayer Perceptrons** The data has images with 28 x 28 pixel values, and you can visualize some images using imshow() function. To implement multilayer perceptrons, you need to flatten the data: instead of representing each sampling using a 28x28 matrix, represent it using an array of length 784.

   IMPLEMENTATION Follow the implementation in class, try at least 5 different network structures, and report their the training and testing accuracies. For the network structure with the best performance, try 3 different types activation functions, and report the training and testing accuracies. What do you observe? Summarize your insights.

3. **CNN**

   IMPLEMENTATION Following the implementation in class, create a baseline CNN that is specified by the following:

   (a) A single convolutional layer with 3 x 3 sized window for computing the convolution, with 32 filters

   (b) Maxpooling layer with 2 x 2 window size.

   (c) Flatten resulting features to reshape your output appropriately.

   (d) Dense layer on top of this (100 neurons) with ReLU activation

   (e) Dense layer with 10 neurons for calculating softmax output (Our classification result will output one of the ten possible classes, corresponding to our digits)

After defining this model, we use Stochastic Gradient Descent (SGD) as the optimizer and the cross-entropy loss. Use a learning rate of 0.01 and a momentum of 0.9.

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

TRAINING AND EVALUATING CNN Now we will train the network. You can see some examples here. Please use 10 epochs and a batch size of 32. Please report the accuracy on test data after you have trained it as above.

EXPERIMENTATION

(a) Run the above training for 50 epochs. Using pyplot, graph the validation and training accuracy after every 10 epochs. Is there a steady improvement for both training and validation accuracy?

For accessing the required values while plotting, you can store the output of the fit method while training your network.

Make sure that your plot has a meaningful legend, title, and labels for X/Y axes.

(b) To avoid over-fitting in neural networks, we can 'drop out' a certain fraction of units randomly during the training phase. You can add the following layer (before the dense layer with 100 neurons) to your model.

```
nn.Dropout(0.5)
```

Now, train this CNN for 50 epochs.

Graph the validation and train accuracy after every 10 epochs.

(c) Add another convolution layer and maxpooling layer immediately following the existing max-pooling layer. For the additional convolution layer, use 64 output filters. Train this for 10 epochs and report the test accuracy.

(d) We used a learning rate of 0.01. Using learning rates of 0.001 and 0.1 respectively, train the model and report accuracy on test data-set. Use Dropout, 2 convolution layers and train for 10 epochs for this experiment.

ANALYSIS

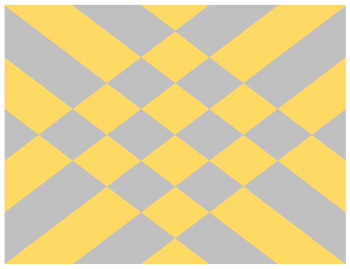
(a) Explain how the trends in validation and train accuracy change after using the dropout layer in the experiments.

(b) How does the performance of CNN with two convolution layers differ as compared to CNN with a single convolution layer in your experiments?

(c) How did changing learning rates change your experimental results in part (iv)?

Written Exercises

1. **Binary Classification** Following Lectures 18 and 19, we would like to train a *2-hidden layer* neural network with *threshold* active function to represent the following decision boundaries. If an input point lies in the yellow region, then the neural network should return 1, and if the input point lies within the grey region, then the neural network should return 0. What is the minimum number of neurons in each hidden layer? Explain your reasoning. How many neurons in total do you need in this neural network?



2. According to the best of our current understanding, why do modern deep neural networks generalize so well to unseen data? Choose all correct answers.

   (a) MLPs with *threshold activation function* are universal function approximators.

   (b) A *Relu* activation function carries more information about the input data than a *sigmoid* activation function.

   (c) MLPs are shift invariant.

   (d) Modern deep neural networks do not have enough capacity (number of nodes and number of layers) to overfit the training data.

   (e) New tricks like batch normalization and residual connections make it highly unlikely to overfit.

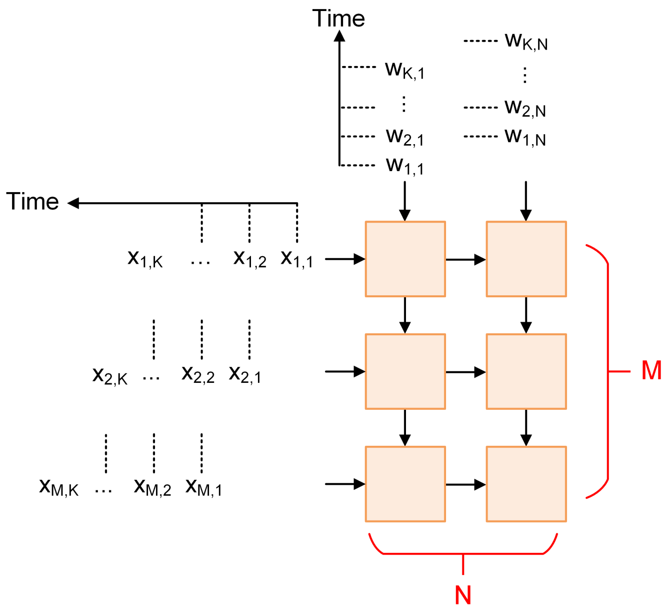
   (f) Large-scale datasets with 50,000+ examples are practically impossible for any model to overfit.

   (g) While some theoretical progress is being made on toy models, the research community still does not know why practical deep nonlinear networks generalize so well.

3. In the guest lecture about hardware for machine learning, you learned about systolic arrays for carrying out efficient matrix multiplication.

   Assume you have a 32×32 systolic array that can multiply a 32×32 input matrix by a 32×32 weight matrix. Assume the input matrix streams horizontally from left to right, and the weight matrix streams vertically from top to bottom (same as the example in class). How many cycles do you need to perform this matrix multiplication?

   Now, assume you have a $M \times N$ systolic array that can multiply a $M \times K$ input matrix by a $K \times N$ weight matrix, as shown below. How many cycles do you need to perform such a matrix multiplication? You should express the cycle count using M, N, K.


Time
$w_{K,1}$
$w_{2,1}$
$w_{1,1}$
$w_{K,N}$
$w_{2,N}$
$w_{1,N}$
Time
$x_{1,K}$ … $x_{1,2}$ $x_{1,1}$
$x_{2,K}$ … $x_{2,2}$ $x_{2,1}$
$x_{M,K}$ … $x_{M,2}$ $x_{M,1}$
M
N

4. (a) Can a multilayer perceptron with one hidden layer of ReLU units represent any function that a multilayer perceptron with one hidden layer of linear units (no activation function) can represent? If so, would you need more, less, or the same number of hidden nodes?

   (b) Can a MLP with one hidden layer of linear units (no activation function) represent the mapping that an MLP with ReLU units can represent? If so, would you need more, less, or the same number of hidden nodes?