CS 464

Introduction Machine Learning

Fall 2018

Homework 3

Due: Jan 1, 2019 23:59

Instructions

- This homework contains both written and programming questions about neural networks. You should implement your programming questions on this notebook. Your plots should also be produced in this notebook. You will also write a report (.pdf) which contains your written answers and plots you will produce here. Each programming question has its own cell for your answer. You can implement your code directly in these cells, or you can call required functions which are defined in a different location for the given question.
- For written questions, your answers have to be in the report as a ".pdf" file.
- For questions that you need to plot, your plot results have to be included in both cell output and your ".pdf" file.
- It is **NOT ALLOWED** to use different libraries than given libraries which are defined in the requirements.txt.
- It is **NOT ALLOWED** to use a different deep learning framework than PyTorch.
- While submitting the homework file, please package your report(".pdf") and notebook(".ipynb") files as a gzipped TAR file or a ZIP file with the name CS464_HW3_Firstname_Lastname. Please do not use any Turkish letters for any of your files including code files and report file.
- Any violation of these rules may lead to significant grade deduction.
- You can send an email for your questions to **can.uner@bilkent.edu.tr** address.

Environment Setup

Anaconda Installation

- Download anaconda from https://www.anaconda.com/download
- Follow the instructions provided in https://conda.io/docs/user-guide/install/index.html#regular-installation

Creation of Virtual Environment

- Create python3.7 virtual environment for your hw3 using follow command from the command line
  > conda create -n HW3 python=3.7 anaconda
- Activate your virtual environment
  > source activate HW3
- To install auxiliary libraries install attached "requirements.txt" and run following command in activated "hw3" environment
  > pip install -r requirements.txt
- *When you create your virtual environment with "anaconda" metapackage, jupyter notebook should be installed. Try:*
  > jupyter notebook

Pytorch Installation

You should install PyTorch to your virtual environment which is created for the hw3. Therefore, you should activate your homework virtual environment before to start PyTorch installation.

- *> source activate HW3*

After you have activated the virtual environment, then use one of the following commands to install pytorch for CPU for your system. See [https://pytorch.org/ (https://pytorch.org/)](https://pytorch.org/) for help.

- *For MacOS:*
  > conda install pytorch torchvision -c pytorch
- *For Linux:*
  > conda install pytorch-cpu torchvision-cpu -c pytorch
- *For Windows:*
  > conda install pytorch-cpu torchvision-cpu -c pytorch

Question 1 [20 pts]

Assume that you are provided a two hidden layer neural network designed to solve a binary classification problem. This architecture uses ReLU activation function between two hidden layers. You should choose the appropriate activation function of the output layer and loss function for the binary classification problem. Compute the gradients and update the weights only for a single iteration. You DO NOT need to provide any numeric inputs to feed the network to compute gradients. You only need to show derivations for the gradients in backpropagation. Provide a pdf file containing your answer (Hint: you do not need the neuron sizes if you use matrix notation).

Question 2 [27 pts]

In this question, you will perform multi-class clasification on a gene expression dataset. Specificically, you will implement a neural network with two hidden layers to predict discrete tumor types: BRCA, KIRC, COAD, LUAD and PRAD. Your features will be continuous gene activity (expression) levels. Download the dataset from the following link:
[https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq (https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq)](https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq). See details below.

Data Loader [4 pts]

An important part of such a task is to implement your own data loader. In this homework, we provide a partial loader for you. This loader is going to be based on a base class named "Dataset", provided in PyTorch library. You need to complete the code below to create your custom "GEDataset" class which will be able to load your dataset. Implement the functions whose proptotypes are given but the bodies are omitted. Follow the TODO notes below. You have to divide the dataset into three sections which are ***"train (50%)", "validation (20%)" and "test (30%)"****. These non-overlapping splits, which are subsets of GEDataset, should be retrieved using the "get_dataset" function.*
Hint: The dataset is not normalized and your results will heavily depend on your input.

In [ ]:

```
# https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq
class GEDataset(Dataset):

    # TODO:
    # Define constructor for GEDataset class
    # HINT: You can pass processed data samples and their ground truth values as
 parameters
    #def __init__(self, **kwargs):

    '''This function should return sample count in the dataset'''
    #def __len__(self):
    #    return self.data.shape[0]

    '''This function should return a single sample and its ground truth value fr
om the dataset corresponding to index parameter '''
    #def __getitem__(self, index):
        #return _x, _y


def get_dataset(root):
    # TODO:
    # Read dataset files from "data/Q1" folder
    # Split dataset samples into the 3 part for training, validation and test
    # Normalize datasets

    #return train_dataset, val_dataset, test_dataset#
```

Neural Network (5 pts)

Now implement your two hidden layer neural network. GENet class will represent your neural network. First hidden layer will contain 64 neurons and second hidden layer will contain 32 neurons. You will decide the number of input and output neurons. Use ReLU as your hidden activation functions. You need to pick a proper activation function for the output layer. Implement the functions with missing bodies.

In [ ]:

```
class GENet(nn.Module):
    '''Define your neural network'''
    def __init__(self, **kwargs): # you can add any additional parameters you wa
nt
    # TODO:
    # You should create your neural network here

    def forward(self, X): # you can add any additional parameters you want
    # TODO:
    # Forward propagation implementation should be here
```

Training (7 pts)

Complete below code snippet to train your network. You need to carefully select the appropriate loss function. You need to select appropriate values for hyper-parameters. You need to use SGD optimizer for this question. So far, you should have created three dataset splits for training, validation and testing above. Note that you will not do cross validation. You will need to load these splits at this phase. Make sure that you shuffle the samples in the training split. Plot training loss and training accuracy of each iteration (each batch). Also plot validation loss and accuracy at each epoch. Use matplotlib library for plotting. Your model is going to run upto the max epoch parameter. Pick the best model so far as your resulting model. You need to save this model in a ".pth" file.

In [ ]:

```
# HINT: note that your training time should not take many days.

# TODO:
# Pick your hyper parameters
# max_epoch =
# train_batch =
# test_batch =
# learning_rate =

#use_gpu = torch.cuda.is_available()


def main(): # you are free to change parameters

    # Create train dataset loader
    # Create validation dataset loader
    # Create test dataset loader
    # initialize your GENet neural network
    # define your loss function

    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=
0.9, weight_decay=5e-04) # you can play with momentum and weight_decay parameter
s as well

    # start training
    # for each epoch calculate validation performance
    # save best model according to validation performance

    #for epoch in range(max_epoch):
    #    train(epoch, model, criterion, optimizer, trainloader)
    #    acc = test(model, val_loader)
    #    if acc > best_acc:
    #       torch.save(model, best_path)

''' Train your network for a one epoch '''
def train(epoch, model, criterion, optimizer, loader): # you are free to change
 parameters
    model.train()

    for batch_idx, (data, labels) in enumerate(loader):
        # TODO:
        # Implement training code for a one iteration

        print('Epoch: [{0}][{1}/{2}]\t'
              'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
              'Data {data_time.val:.4f} ({data_time.avg:.4f})\t'
              'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
              'Accu {acc.val:.4f} ({acc.avg:.4f})\t'.format(
               epoch + 1, batch_idx + 1, len(trainloader),
               batch_time=batch_time,
               data_time=data_time,
               loss=losses,
               acc=accuracies))


''' Test&Validate your network '''
def test(model, loader): # you are free to change parameters

    model.eval()
```

```
    with torch.no_grad():
        for batch_idx, (data, labels) in enumerate(testloader):
            # TODO:
            # Implement test code

        print('Time {batch_time.avg:.3f}\t'
              'Accu {acc.avg:.4f}\t'.format(
               batch_time=batch_time,
               acc=accuracies))
```

Plotting Your Results [4 pts]

For us to assess if your model has learnt as expected, you have to provide training loss, training accuracy, validation loss and validation accuracy plots. You need to provide two distinct plots, one demonstrating training and validation loss scores and the other demonstrating training and validation accuracies.

In [ ]:

```
# write your code in this cell to plot your results
```

Testing [7 pts]

You will test your final model with test dataset in this section. You should provide confusion matrix as deliverable. Report confusion matrix in your pdf file.

In [ ]:

```
# write your code in this cell to test your best model with the test dataset
```

Question 3 [53 pts]

In this question, you will train a neural networks to recognize house numbers from images. You will use the dataset at the following link: http://ufldl.stanford.edu/housenumbers/ (http://ufldl.stanford.edu/housenumbers/). You are actually asked to predict the number in the middle (i.e., for 173, predict 7). Images are cropped accordingly and correct labels are given in the dataset. You do not need to worry about other numbers that might still be visible in the image. You have to use ***train_32x32.mat*** *and* ***test_32x32.mat*** *from* ***the CROPPED version (FORMAT2)****. All images in this dataset have the following dimensions: 32x32x3. You will solve this problem using two architectures: (1) Using a MLP and (2) Using a CNN.*

3.1. Multi Layer Perceptron (MLP) [23 pts]

DataLoader [3 pts]

In this part, you will train an MLP. Here, you are supposed to flatten the image into a vector (also to grayscale). Note that the pixel values also needs to be normalized to [0,1] range. First, implement the data loader (SVHNDataset) as you did in Question 2. For this question, do not perform cross validation also do not use validation split. Note that training and test splits are given to you in the provided link.

In [ ]:

```
# http://ufldl.stanford.edu/housenumbers/
class SVHNDataset(Dataset):
    # TODO:
    # Define constructor for SVHNDataset class
    # HINT: You can pass processed data samples and their ground truth values as
 parameters
    #def __init__(self, **kwargs): # you are free to change parameters

    '''This function should return sample count in the dataset'''
    #def __len__(self):
    #    return self.data.shape[0]

    '''This function should return a single sample and its ground truth value fr
om the dataset corresponding to index parameter '''
    #def __getitem__(self, index):
        #return _x, _y


def get_dataset(root): #you are free to change parameters
    # TODO:
    # Read dataset files from "data/Q2" folder
    # Normalize datasets

    #return train_dataset, test_dataset
```

Neural Network [4 pts]

Now implement your two hidden layered neural network. FCNet class will represent your neural network. First hidden layer will contain 256 neurons and second hidden layer will contain 256 neurons. You will decide the number of input and output neurons. Use ReLU as your hidden activation functions. You need to pick a proper activation function for the output layer.

In [ ]:

```
class FCNet(nn.Module):
    '''Define your neural network'''
    def __init__(self, **kwargs): # you can add any additional parameters you wa
nt
    # TODO:
    # You should create your neural network here

    def forward(self, X): # you can add any additional parameters you want
    # TODO:
    # Forward propagation implementation should be here
```

Training and Testing [10 pts]

Now, train your network. You need to select the appropriate loss function. You need to select appropriate hyper-parameters' values. Make sure that you shuffle the samples in the training split. Note that you will not do cross validation. Plot the training loss and accuracy for each iteration. Plot the test loss and accuracy for each epoch. Your model is going to run upto the max epoch parameter. Pick the best model as your resulting model. You need to save this model in a ".pth" file. (HINT: note that your training time should not take many days.)

In [ ]:

```
# TODO:
# Pick your hyper parameters
# max_epoch =
# train_batch =
# test_batch =
# learning_rate =

#use_gpu = torch.cuda.is_available()


def main(): # you are free to change parameters

    # Create train dataset loader
    # Create validation dataset loader
    # Create test dataset loader
    # initialize your GENet neural network
    # define your loss function

    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=
0.9, weight_decay=5e-04) # you can play with momentum and weight_decay parameter
s as well

    # start training
    # for each epoch calculate validation performance
    # save best model according to validation performance

    #for epoch in range(max_epoch):
    #    train(epoch, model, criterion, optimizer, trainloader)
    #    acc = test(model, val_loader)
    #    if acc > best_acc:
    #       torch.save(model, best_path)

''' Train your network for a one epoch '''
def train(epoch, model, criterion, optimizer, loader): # you are free to change
 parameters
    model.train()

    for batch_idx, (data, labels) in enumerate(loader):
        # TODO:
        # Implement training code for a one iteration

        print('Epoch: [{0}][{1}/{2}]\t'
              'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
              'Data {data_time.val:.4f} ({data_time.avg:.4f})\t'
              'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
              'Accu {acc.val:.4f} ({acc.avg:.4f})\t'.format(
               epoch + 1, batch_idx + 1, len(trainloader),
               batch_time=batch_time,
               data_time=data_time,
               loss=losses,
               acc=accuracies))


''' Test&Validate your network '''
def test(model, loader): # you are free to change parameters

    model.eval()

    with torch.no_grad():
```

```
        for batch_idx, (data, labels) in enumerate(testloader):
            # TODO:
            # Implement test code

        print('Time {batch_time.avg:.3f}\t'
              'Accu {acc.avg:.4f}\t'.format(
               batch_time=batch_time,
               acc=accuracies))
```

Weight Visualization [6 pts]

For the best model, extract the learned weights from first hidden layer. Visualize learned weights for each neuron of the first hidden layer as an image. Show these images in a grid and add this grid in the pdf file as well.

In [ ]:

```
# write your code in this cell to visualize first hidden layer weights. Produce
 your figure here.
```

3.2. Convolutional Neural Network (CNN) [30 pts]

DataLoader [3 pts]

In this part, you will train a CNN for the same problem. Again, the pixel values also needs to be normalized to [0,1] range. First, implement the data loader (SVHNDataset). Note that now you do not need to flatten the image. Again, for this question, do not perform cross validation. Also, just like 3.1., do not use a validation split. Use the same splits as in 3.1.

In [ ]:

```
# http://ufldl.stanford.edu/housenumbers/
class SVHNDataset(Dataset):
    # TODO:
    # Define constructor for SVHNDataset class
    # HINT: You can pass processed data samples and their ground truth values as
 parameters
    #def __init__(self, **kwargs): # you are free to change parameters

    '''This function should return sample count in the dataset'''
    #def __len__(self):
    #    return self.data.shape[0]

    '''This function should return a single sample and its ground truth value fr
om the dataset corresponding to index parameter '''
    #def __getitem__(self, index):
        #return _x, _y


def get_dataset(root): # you are free to change parameters
    # TODO:
    # Read dataset files from "data/Q2" folder
    # Normalize datasets

    #return train_dataset, test_dataset
```

Neural Network [9 pts]

Now implement your CNN. ConvNet class will represent your convolutional neural network. Implement 3 layers of convolution:

- *(1) 32 filters with size of 3 x 3 with stride 1 and no padding, (2) ReLU*
- *(3) 64 filters with size of 3 x 3 with stride 1 and no padding, (4) ReLU and (5) MaxPool 2 x 2*
- *(6) 128 filters with size of 3 x 3 with stride 1 and no padding, (7) ReLU and (8) MaxPool 2 x 2*

As a classifier layer, you need to add only one linear layer at the end of the network. You need to choose the appropriate input and output neuron sizes for the classification (linear) layer.

In [ ]:

```
class ConvNet(nn.Module):
    '''Define your neural network'''
    def __init__(self, **kwargs): # you can add any additional parameters you wa
nt
    # TODO:
    # You should create your neural network here

    def forward(self, X): # you can add any additional parameters you want
    # TODO:
    # Forward propagation implementation should be here
```

Training and Testing [10 pts]

Now, train your network. You need to select the appropriate loss function. You need to select your hyper-parameters. Make sure that you shuffle the samples in the training split. Note that you will not do cross validation. Plot the training loss and accuracy for each iteration. Plot the test loss and accuracy for each epoch. Your model is going to run upto the max epoch parameter. Pick the best model as your resulting model. You need to save this model in a ".pth" file. Report the validation performance change between MLP and CONV neural network and explain the reason for this change. You need to add this explanation and your plots into the pdf file.

In [ ]:

```
# HINT: note that your training time should not take many days.

# TODO:
# Pick your hyper parameters
# max_epoch =
# train_batch =
# test_batch =
# learning_rate =

#use_gpu = torch.cuda.is_available()


def main(): # you are free to change parameters

    # Create train dataset loader
    # Create validation dataset loader
    # Create test dataset loader
    # initialize your GENet neural network
    # define your loss function

    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=
0.9, weight_decay=5e-04) # you can play with momentum and weight_decay parameter
s as well

    # start training
    # for each epoch calculate validation performance
    # save best model according to validation performance

    #for epoch in range(max_epoch):
    #    train(epoch, model, criterion, optimizer, trainloader)
    #    acc = test(model, val_loader)
    #    if acc > best_acc:
    #       torch.save(model, best_path)

''' Train your network for a one epoch '''
def train(epoch, model, criterion, optimizer, loader): # you are free to change
 parameters
    model.train()

    for batch_idx, (data, labels) in enumerate(loader):
        # TODO:
        # Implement training code for a one iteration

        print('Epoch: [{0}][{1}/{2}]\t'
              'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
              'Data {data_time.val:.4f} ({data_time.avg:.4f})\t'
              'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
              'Accu {acc.val:.4f} ({acc.avg:.4f})\t'.format(
               epoch + 1, batch_idx + 1, len(trainloader),
               batch_time=batch_time,
               data_time=data_time,
               loss=losses,
               acc=accuracies))


''' Test&Validate your network '''
def test(model, loader): # you are free to change parameters

    model.eval()
```

```
    with torch.no_grad():
        for batch_idx, (data, labels) in enumerate(testloader):
            # TODO:
            # Implement test code

        print('Time {batch_time.avg:.3f}\t'
              'Accu {acc.avg:.4f}\t'.format(
               batch_time=batch_time,
               acc=accuracies))
```

Weight Visualization [8 pts]

For the best convolutional model you obtained, extract the learnt weights from the first convolutional layer. Visualize each learnt filter of the first convolutional layer as an image. Show these images in a grid and add this grid in to the pdf file as well.

In [ ]:

```
# write your code in this cell to visualize filters of the first convolutional l
ayer
```

BONUS: Question 4 [20 pts]

The learnt weights of a network that has been trained to solve a problem can be used as a starting point for the weights of neural networks that will be used to solve another similar problem. Instead of random initialization for the weights of the neural networks, using the weights of the neural network trained to solve a similar problem will make learning easier. This approach is called ***"Transfer Learning"****.*

Dataset

For this question, you will use CIFAR10 dataset. You can find detailed information about the dataset from https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html). Torchvision library provides a ready-to-use data loader for the CIFAR10 dataset. You ***DO NOT*** *need to implement your custom data loader for this question. You can easily split the dataset into training and test. Just set the "train" parameter of the constructor of torchvision.datasets.CIFAR10 class. You will the use test split as the validation set at each epoch.*

Model

You have to use AlexNet convolutional neural network which is one of the most basic CNN models. You ***DO NOT*** *need to implement your own AlexNet architecture. Torchvision has also a model zoo which contains commonly used CNN models including AlexNet. Therefore, you need to use the AlexNet model from the Torchvision library. For the question 4.2 Transfer Learning part, you will need to transfer pretrained network weights as a starting point. These weights will come from the result of the training with Imagenet dataset. These will be loaded automatically when you set the "pretrained" parameter to true (check the hints in the code). Otherwise, weights will be randomly initialized.*

4.1 Random Initialization

4.1.1 Train & Validation

You have to train randomly initialized AlexNet CNN model using CIFAR10 dataset. At the end of each epoch, you should evaluate your network with the test split. Print (not plot) training loss, training accuracy, validation loss and validation accuracy values for each epoch. When the training is completed, print the best validation score that you have obtained during training. Report this score in your pdf file as well. You will compare this score with the result of the next question. Note that this might take a long time because of randomly initilized weights. Stop training when validation score converges.

In [ ]:

```
# write your code in this cell to train your network that has randomly selected
 weights

# HINTS:
# alexnet = torchvision.models.alexnet(pretrained=False)
# torchvision.models.alexnet(pretrained=False) will return an alexnet model inst
ance with randomly initialized weights
# you need to make some changes in the classifier layer to get a proper network
 for your problem.
```

4.1.2 Kernel Output Visualization

You have trained AlexNet which has randomly initialized weights by using the CIFAR10 dataset. Now you have to plot the output of the each filter at the first convolution layer as an image by using randomly selected single validation image. Merge each image obtained from the corresponding kernel in a squared grid format. Add your plot in to the pdf file as well.

In [ ]:

```
# write your code in this cell to visualize output of the each filter at the fir
st conv layer
```

4.2 Transfer Learning

4.2.1 Train & Validation

In this case, you have to train ImageNet pretrained AlexNet model for the CIFAR10 dataset. At the end of each epoch, you should evaluate your network with test split. Print training loss, training accuracy, validation loss and validation accuracy values for each epoch as an output of below cell. Report the best validation accuracy score. Compare validation scores that are obtained from these two different training approaches. Add your explanations to the pdf file.

In [ ]:

```
# write your code in this cell to train your network using transfer learning app
roach

# HINTS:
# alexnet = torchvision.models.alexnet(pretrained=True)
# torchvision.models.alexnet(pretrained=True) will return an alexnet model insta
nce with ImageNet pretrained network weights.
# you need to make some changes in the classifier layer to get a proper network
 for your problem.
```

4.2.2 Kernel Output Visualization

You have trained AlexNet which is pretrained with ImageNet dataset for the CIFAR10 dataset. For this network, you have to plot the output of each filter at the first convolution layer as an image by using the same image that has picked 4.1.3 section. Merge each image obtained from the corresponding kernel in a squared grid format. Explain what these outputs mean. Compare your plot with the obtained from 4.1.2. You need to add your discussions and plot into the pdf file as well.

In [ ]:

```
# write your code in this cell to visualize output of the each filter at the fir
st conv layer
```