CSCE 5013-002

Deadline: Tuesday November 5, 2019

Where: In the class (Before the class begin) for Q1-2-3

Submit by email for Q4 to thile@uark.edu and bjlowe@email.uark.edu

with email title Q4 – Homework 2

Total point: 110 points

Question 1 (10 points)

You are trying to minimize the **L2 divergence** (as defined in the class) of the RELU function $y = \max(0,\ 0.5w)$ with respect to parameter w, when the target output is 1.0. Note that this corresponds to optimizing the RELU function $y = \max(0,\ wx)$ given only the training input $(x, d(x)) = (0.5, 1)$. You are using **momentum updates**. Your current estimate (in the k-th step) is $w^{(k)} = 1$. The last step you took was $\Delta w^{(k)} = 0.5$. Using the notation from class, you use $\beta = 0.9$ and $\eta = 0.1$. What is the value of $w^{(k+1)}$ when using momentum update? Truncate the answer to three decimals (do not round up).

Question 2 (10 points)

You are trying to minimize the **cross-entropy loss** of the logistic function $y = 1/(1 + \exp(0.5w))$ with respect to parameter w, when the target output is 1.0. Note that this corresponds to optimizing the logistic function $y = 1/(1 + \exp(-wx))$, given only the training input $(x, d(x)) = (-0.5, 1)$. You are using **momentum updates**. Your current estimate (in the k-th step) is $w^{(k)} = 0$. The last step you took was $\Delta w^{(k)} = 0.5$. Using the notation from class, you use $\beta = 0.9$ and $\eta = 0.1$. What is the value of $w^{(k+1)}$ when using momentum update? Truncate the answer to three decimals (do not round up).

Question 3 - Implement 1D convolutional layer (30 points)

In this section, you need to implement convolutional neural networks using the **NumPy library only**

Implement the Conv1D class in layers.py. The class Conv1D has four arguments: in channel, out channel, kernel size and stride. We do not consider other arguments such as padding.

The detail meaning of the arguments has been discussed in class and you can also find them in PyTorch document: https://pytorch.org/docs/stable/ nn.html#torch.nn.Conv1d.

Implement the Conv1D class so that it has similar usage and functionality to torch.nn.Conv1d. There are three parts to be implemented.

Forward [10 points]

Finish the returned value for the forward part.

The input x is the input of the convolutional layer and the shape of x is (batch size, in channel, in width). The return value is the output of the convolutional layer and the shape is (batch size, out channel, out width). Note that in width is an arbitrary possitive integer and out width is determined by in width, kernel size and stride.

Backward [20 points]

You will write the code of the backward method. Its argument is the backpropagated delta, i.e. the derivative of the loss w.r.t the output of the current layer (dW and db)

The input delta is the derivative of the loss with respect to the convolutional layer output. It has the same shape as the convolutional layer output. self.dW and self.db represent the unaveraged gradients of the loss w.r.t self.W and self.b. Their shapes are the same as the weight self.W and the bias self.b. We have already initialized them for you.

dx: Finish the returned value for the backward part. It is the derivative of the loss with respect to the input of the convolutional layer and has the same shape as the input.

Test: To test your implementation, we will compare the results of the three parts with Pytorch function torch.nn.Conv1d. Details can be found in function test cnn correctness once in the local grader.py file. Here are some brief codes to show the testing.

```
import torch
import torch.nn as nn
import numpy as np
from torch.autograd import Variable from layers import Conv1D

## initialize your layer and PyTorch layer
net1 = Conv1D(8, 12, 3, 2) ## Your own convolution
net2 = torch.nn.Conv1d(8, 12, 3, 2) ## generated by torch

## initialize the inputs
x1 = np.random.rand(3, 8, 20)
x2 = Variable(torch.tensor(x1),requires_grad=True)
## Copy the parameters from the Conv1D class to PyTorch layer
net2.weight = nn.Parameter(torch.tensor(net1.W))
net2.bias = nn.Parameter(torch.tensor(net1.b))
## Your forward and backward
y1 = net1(x1)
b, c, w = y1.shape
delta = np.random.randn(b,c,w)
dx = net1.backward(delta)
## PyTorch forward and backward
y2 = net2(x2)
delta = torch.tensor(delta)
y2.backward(delta)

## Compare

def compare(x,y):
    y = y.detach().numpy()
    print(abs(x).max())
    return
```

```
compare(y1, y2)
compare(dx, x2.grad)
compare(net1.dW, net2.weight.grad)
compare(net1.db, net2.bias.grad)
```

Question 4 Image Classification (60 points) – Competition

In this exercise, your task is to create a Convolutional Neural Network (CNN) based image classifier for the CIFAR-10 dataset of images.

Feel free to experiment with different architectures, hyper-parameters, and training procedures to make yourself familiar with training and tuning CNN models. Once you have the model implemented and working, start playing around with the number of layers, their sizes, the activation function, using dropout, stride, etc... whatever you would like!

The dataset you will be using is the CIFAR-10 Dataset which consists of a collection of 60,000 images of 32x32 pixels with three 8-bit RGB channels. 50,000 images for training nd 10,000 images for testing. The training data is devided into 5 batchs. Here is an example to read feature and label from each batch:

```
def load_cfar10_batch(cifar10_dataset_folder_path, batch_id):
    with open(cifar10_dataset_folder_path + '/data_batch_' + str(batch_id), mode='rb')
as file:
        # note the encoding type is 'latin1'
        batch = pickle.load(file, encoding='latin1')

    features = batch['data'].reshape((len(batch['data']), 3, 32, 32)).transpose(0, 2,
3, 1)
    labels = batch['labels']


return features, labels
```

The label data is just a **list of 10,000 numbers** ranging from **0 to 9**, which corresponds to each of the **10 classes in CIFAR-10**. The classes are: `['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']`
Before process, you may need to convert the label into one-hot-vector

index	label
0	airplane (0)
1	automobile (1)
2	bird (2)
3	cat (3)
4	deer (4)
5	dog (5)
6	frog (6)
7	horse (7)
8	ship (8)
9	truck (9)
...	...
...	...

original label data

label	index											
	0	1	2	3	4	5	6	7	8	9	...	...
airplane	**1**	0	0	0	0	0	0	0	0	0	...	...
automobile	0	**1**	0	0	0	0	0	0	0	0	...	...
bird	0	0	**1**	0	0	0	0	0	0	0	...	...
cat	0	0	0	**1**	0	0	0	0	0	0	...	...
deer	0	0	0	0	**1**	0	0	0	0	0	...	...
dog	0	0	0	0	0	**1**	0	0	0	0	...	...
frog	0	0	0	0	0	0	**1**	0	0	0	...	...
horse	0	0	0	0	0	0	0	**1**	0	0	...	...
ship	0	0	0	0	0	0	0	0	**1**	0	...	...
truck	0	0	0	0	0	0	0	0	0	**1**	...	...

one-hot-encoded label data

Detail about CIFAR 10: https://www.cs.toronto.edu/~kriz/cifar.html

An input to your system is an image and you have to predict the ID of the image. The true image ID will be present in the training data and so the network will be doing 10-way classification to get the prediction. You are provided with the train set and have the option to split it into a training and validation set. With the validation set, you can fine-tune the model based on the accuracy you get.

Loading Dataset

```
import torch
import torchvision
from torchvision.datasets import CIFAR10
TRAIN_TRANSFORMS = torchvision.transforms.ToTensor()
TEST_TRANSFORMS = TRAIN_TRANSFORMS
BATCH_SIZE = 16
### DO NOT MODIFY ANY CODE STARTING HERE ###
cifar_train = CIFAR10('./data', download=True, train=True,
                      transform=TRAIN_TRANSFORMS)
cifar_test = CIFAR10('./data', download=True, train=False,
                     transform=TEST_TRANSFORMS)
train_dataloader = torch.utils.data.DataLoader(cifar_train,
                                               batch_size=BATCH_SIZE,
                                               shuffle=True)
test_dataloader = torch.utils.data.DataLoader(cifar_test,
del cifar_train, cifar_test
```

It's very important to note that, for classification, the train, validation, and test contain the same set of images. That means your network cannot classify images unless it has seen it before, and the dataset is set up that way. In other words, the exact images are not the same between the train, validation, and test set but rather they are all sampled from the same distribution.

You will have one folder that will download after you run the provided script. Further, two data loaders will be generated:

```
train_dataloader
test_dataloader
```

DataLoader Usage: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

Official Page: https://pytorch.org/tutorials

Repository: https://github.com/pytorch/tutorials

Documentation: https://pytorch.org/docs/stable/index.html

Write up for your own reference describing what model architecture, loss function, hyperparameters, any other interesting detail led to your best result for the above. Such a write-up should limit the content to one page.

Output from your network is a text file (.txt) which the following format:

Image ID (name)	Predicted Label