BME69500DL ECE695
Spring 2020

Homework 2

Deadline : Thursday, February 13, 2020 , 11:59 am

1 Introduction

The primary goal of this homework is for you to become familiar with the data loading facilities provided by Torchvision and for you to customize data loading to your needs.

And a secondary goal is to hand-craft the backpropagation of loss for the calculation of the gradients of loss with respect to the learnable parameters for a neural network with an input layer, two hidden layers, and one output layer.

The data you will be working with is the CIFAR-10 dataset that you can download from

`https://www.cs.toronto.edu/~kriz/cifar.html`

As mentioned, your homework will involve training a simple neural network that consists of an input layer, two hidden layers, and one output layer. We will use the matrix w_1 to represent the link weights between the input and the first hidden layer, the matrix w_2 the link weights between the first hidden layer and the second hidden layer, and, finally, the matrix w_3 the link weights between the second hidden layer and the output.

For each hidden layer, we will use the notation hi as the output before the application of the activation function and hi_{relu} for the output after the activation. So if x is the vector representation of the input data, we have the following relationships in the forward direction:

$$\begin{aligned} h1 &= x.mm(w1) \\ h1_{relu} &= h1.clamp(min = 0) \\ h2 &= h1_{relu}.mm(w2) \\ h2_{relu} &= h2.clamp(min = 0) \\ y_{pred} &= h2_{relu}.mm(w3) \end{aligned}$$

where $.mm()$ does for tensors what $.dot()$ does for Numpy's ndarrays. Basically, mm stands for matrix multiplication. Remember that with tensors, a vector is a one-row tensor. That is, when an n-element vector stored in a tensor, its shape is $(1, n)$. So what you see in the first line, "$h1 = x.mm(w1)$" amounts to multiplying a matrix $w1$ with a vector x.

Before listing the tasks, you need to also understand how the loss can be backpropagated and the gradients of loss computed for simple neural networks. The following 3-step logic involved is as follows for the case of MSE loss for the last layer of the neural network. You repeat it backwards for the rest of the network.

- The loss at the output layer:

  $$L = (y - y_{pred})^t (y - y_{pred})$$

  where y is the groundtruth vector and y_pred the predicted vector.

- Propagating the loss backwards and calculating the gradient of the loss with respect to the parameters in the link weights involves the following three steps:

  1. Find the gradient of the loss with respect to the link matrix w_3 by:

     $$grad_{w3} = h2^t_{relu}.mm(2 * y_{error})$$

2. Propagate the error to the post-activation point in the hidden layer h_2 by

$$h2_{error} = 2 * y_{error} \cdot mm(w_3^t)$$

3. Propagate the error past the activation in the layer $h2$ by

$$h2_{error}[h2_{error} < 0] = 0$$

2 Tasks

Here are your tasks for the homework:

1. First become familiar with the CIFAR-10 image dataset by visiting the website. As you will discover this dataset consists of 50,000 training images, each of size 32×32 and 10,000 test images of the same size.

2. The dataset contains images for ten different classes of objects. If you use a dataloader provided by Torchvision, by default it will load images for all the classes. You construct instances of dataloaders in your own code by:

```
transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
##  Define where the training and the test datasets are located:
train_data_loc = torchvision.datasets.CIFAR10(root=self.dataroot, train=True,download=True, transform=transform)
test_data_loc = torchvision.datasets.CIFAR10(root=self.dataroot, train=False,download=True,transform=transform)
##  Now create the data loaders:
self.trainloader = torch.utils.data.DataLoader(train_data_loc, batch_size=5,shuffle=True, num_workers=2)
self.testloader = torch.utils.data.DataLoader(test_data_loc, batch_size=5, shuffle=False, num_workers=2)
```

   The statements shown above are snippets from my `DLStudio` class.. But there is nothing special about them. When default behavior is acceptable, all data loader statements for a well-known dataset look the same.

   Your task is to customize the dataloader so that it feeds into a training algorithm only two of the ten classes — cat and dog.

3. Now train the three layer neural network using the code shown below. The code is shown only to give you an idea of how you can structure your program. But it should get you started.

```
import torch

dtype = torch.float

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

N, D_in, H1, H2, D_out = 8, 3*32*32, 1000, 256, 2

# Randomly initialize weights
w1 = torch.randn(D_in, H1, device=device, dtype=dtype)
w2 = torch.randn(H1, H2, device=device, dtype=dtype)
w3 = torch.randn(H2, D_out, device=device, dtype=dtype)

learning_rate = 1e-6
for t in range(2500):

for i, data in enumerate(self.train_data_loader):

    inputs, labels = data

    inputs = inputs.to(device)
    labels = labels.to(device)
```

```
x = inputs.view(x.size(0), -1)

h1 = x.mm(w1)                              ##  In numpy, you would say    h1 = x.dot(w1)
h1_relu = h1.clamp(min=0)
h2 = h1_relu.mm(w2)
h2_relu = h2.clamp(min=0)
y_pred = h2_relu.mm(w3)

# Compute and print loss
loss = (y_pred - y).pow(2).sum().item()
if t % 100 == 99:
    print(t, loss)

y_error = y_pred - y

grad_w3 = h2_relu.t().mm(2 * y_error)   #<<<<<< Gradient of Loss w.r.t w3
h2_error  =  2.0 * y_error.mm(w3.t())   # backpropagated error to the h2 hidden layer
h2_error[h < 0] = 0                     # We set those elements of the backpropagated error

grad_w2 = h1_relu.t().mm(2 * h2_error)  #<<<<<< Gradient of Loss w.r.t w2
h1_error  =  2.0 * h2_error.mm(w2.t())  # backpropagated error to the h1 hidden layer
h1_error[h < 0] = 0                     # We set those elements of the backpropagated error

grad_w1 = x.t().mm(2 * h1_error)        #<<<<<< Gradient of Loss w.r.t w2

# Update weights using gradient descent
w1 -= learning_rate * grad_w1
w2 -= learning_rate * grad_w2
w3 -= learning_rate * grad_w3
```

4. Write a similar script to test the classification performance of your network on the CIFAR-10 test data.

5. Plot your loss as a function of the epochs during training and present the classification performance based on the test results.

3 Submission

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Name your main Python file as hw02.py
- All homework to be submitted on-line through the *turnin* command on the min.ecn.purdue.edu[1][2] server.
- Log into min.ecn.purdue.edu server using your Purdue career account username.
- From min.ecn.purdue.edu, go to the directory where your homework files are. e.g., if your files are at /home/hw1 directory then go to /home/hw1. Obviously, if you don't have your files on the server, move them there using, for example, the *scp* command.
- Type in the following command:

  `turnin -c ecedl -p hw<double digits hw#> <your files separated by a space>`

- As an example, for this homework you will enter the command:

  `turnin -c ecedl -p hw02 hw02.py`

[1] If you are an undergraduate student, use the shay.ecn.purdue.edu server instead of the **min** server
[2] If you registered this class as the BME695 course then use the weldon.ecn.purdue.edu server

- You should get a statement that says "Your files have been submitted to ecedl, hw02 for grading". You can verify your submission by typing:

  ```
  turnin -c ecedl -p hw<double digits homework number> -v .
  ```