[AI502/KSE527] Homework 1

SangMook Kim, sangmook.kim@kaist.ac.kr
Jaehoon Oh, jhoon.oh@kaist.ac.kr

March 26, 2020

1 Description

In this homework, we will deal with both simple regression and classification problems. This homework focuses on the overall flow of deep learning using Pytorch. Before you start, you should set 'jupyter' environment because we use .ipynb file for homework, which is an interactive python notebook file. There are three general ways to install or use 'jupyter notebook':

1. Install jupyter by commanding 'pip3 install jupyter'
2. Install 'Anaconda', which includes jupyter (Site)
3. Use 'google Colab' (Site)

2 Problem 1

Problem 1 is about polynomial approximation using Pytorch. Through this problem, we want to see how to approximate real function f using a given dataset through training. You can find the following code, which is [HW1-1]Polynomial Approximation.ipynb file.

- The code flow is as follows:
    1. First, we make the real polynomial function that we should approximate. (In our case, we fix the real function using the random seed for convenience.)

```
import torch
import torch.nn as nn
import torch.utils.data

POLY_DEGREE = 4
torch.manual_seed(2020)
W_target = torch.randn(POLY_DEGREE, 1) * 5
b_target = torch.randn(1) * 5

def poly_desc(W, b):
    """Creates a string description of a polynomial."""
    result = 'y = '
    for i, w in enumerate(W):
        result += '{:+.2f} x^{} '.format(w, len(W) - i)
    result += '{:+.2f}'.format(b[0])
    return result

print('==> The real function you should approximate:\t' + poly_desc(W_target.view(-1), b_target))
==> The real function you should approximate:   y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40
```

Figure 1: Step 1 in HW 1-1

2. Next, from this function, we can make as many datasets as we want. (In general, it is an impossible situation since we cannot know the real function.)

```python
def make_features(x):
    """Builds features i.e. a matrix with columns [x^4, x^3, x^2, x^1]."""
    x = x.unsqueeze(1)
    return torch.cat([x ** (POLY_DEGREE+1-i) for i in range(1, POLY_DEGREE+1)], 1)

def f(x):
    """Approximated function."""
    return x.mm(W_target) + b_target[0]

def get_dataset(dataset_size):
    """Builds a batch i.e. (x, f(x)) pair."""
    random = torch.randn(dataset_size)
    x = make_features(random)
    y = f(x)
    dataset = list(zip(x, y))
    return dataset

dataset = get_dataset(200) # you can make as many as dataset as you want
```

Figure 2: Step 2 in HW 1-1

3. We control various hyperparameters and a loss function such as L1 loss, MSE loss, and SmoothL1 loss in regression.

```python
num_epochs = 500
batch_size = 50
learning_rate = 0.1
criterion = nn.SmoothL1Loss()

dataset_loader = torch.utils.data.DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True)
```

Figure 3: Step 3 in HW 1-1

4. Next, we should define a network approximating the real function. In our case, since we already know the real function is the quartic function (polynomial of degree 4) and linear function, we define like this:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc = nn.Linear(W_target.size(0), 1)

        # For fixing the initial weights and bias
        self.fc.weight.data.fill_(0.)
        self.fc.bias.data.fill_(0.)

    def forward(self, x):
        output = self.fc(x)
        return output
```

Figure 4: Step 4 in HW 1-1

5. Also, we define the training process as a function of python with various hyperparameters.

```python
def fit(model,loader,criterion,learning_rate,num_epochs):
    model.train()
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

    for epoch in range(num_epochs):
        for i, data in enumerate(loader):
            if torch.cuda.is_available():
                x = data[0].type(torch.FloatTensor).cuda()
                y = data[1].type(torch.FloatTensor).cuda()
            else:
                x = data[0].type(torch.FloatTensor)
                y = data[1].type(torch.FloatTensor)

            y_hat = model(x)
            loss = criterion(y_hat, y)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

Figure 5: Step 5 in HW 1-1

6. Before training, you can check the initialized weights. And, as you train the model, you can see that the learned function is getting closer to the actual function.

```
net = Net().cuda() if torch.cuda.is_available() else Net()
print('==> Initial function:\t' + poly_desc(net.fc.weight.data.view(-1), net.fc.bias.data))
print('==> Actual function:\t' + poly_desc(W_target.view(-1), b_target))

==> Initial function:   y = +0.00 x^4 +0.00 x^3 +0.00 x^2 +0.00 x^1 +0.00
==> Actual function:    y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40

# train
fit(net,dataset_loader,criterion,learning_rate,num_epochs)

print('==> Learned function:\t' + poly_desc(net.fc.weight.data.view(-1), net.fc.bias.data))
print('==> Actual function:\t' + poly_desc(W_target.view(-1), b_target))

==> Learned function:   y = +6.02 x^4 -4.72 x^3 +7.70 x^2 -2.03 x^1 +4.47
==> Actual function:    y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40
```

Figure 6: Step 6 in HW 1-1

- The following contents should be submitted:

1. Make a python code for plotting some graphs that help your analysis. (2pts)

   (a) Loss graph

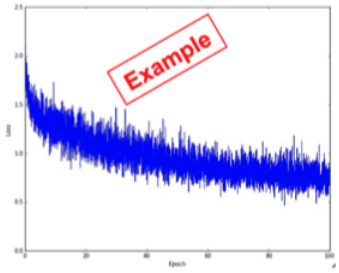

Figure 7: Loss Graph Example

   (b) The real function and the learned function

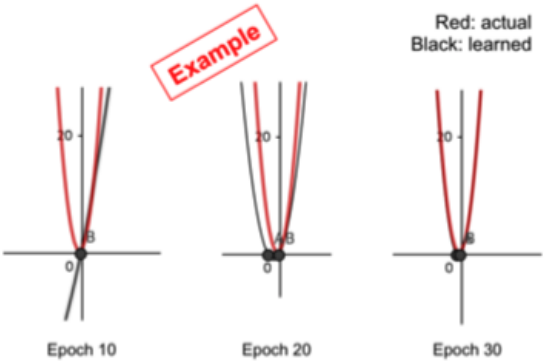

Figure 8: Function Approximate Example

2. Analyze about convergence according to learning rate. (Including loss graph per epoch, and the real function and the learned function per 100 epochs) (3pts)

   Fix other conditions as follow:

   – Loss function: SmoothL1 loss
   – Optimizer: SGD

- Number of epochs: 500
- Dataset size: 200
- Batch size: 50

Control learning rate:

- 0.1 / 0.01 / 0.001

In addition, we want you to understand the code flow exactly and to explore various situations through this problem. For example, analysis of loss functions, dataset size, and no-fixed initialization, and so on. (This is not related to your HW score.)

3 Problem 2

The goal of this problem is to see what functions are often used in PyTorch and to know the whole process of deep learning programming. For this problem, you need to analyze the code that performs the classification task using MNIST data. MNIST data is a set of learning data for the classification of handwritten digits and is composed of images and corresponding labels as follows.

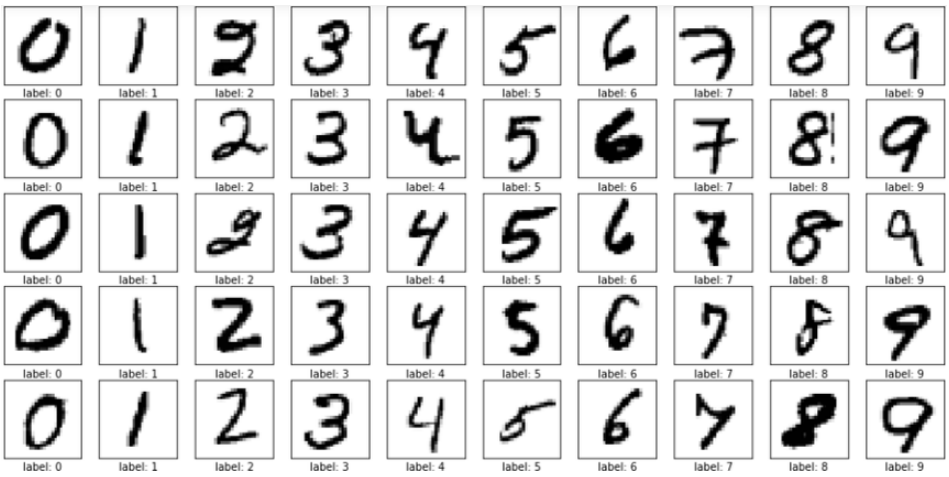

Figure 9: MNIST Image

You can find the following code, which is [HW1-2]MNIST_sample.ipynb file.

- The code flow is as follows:

1. Import modules to use PyTorch

```
import os
import torch
import torch.nn as nn
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torch.nn.functional as F
import torch.optim as optim

import matplotlib.pyplot as plt
```

2. Set the path to store MNIST dataset

```
root = './data'
if not os.path.exists(root):
    os.mkdir(root)
```

3. Download MNIST training and test dataset

```
train_set = dset.MNIST(root=root, train=True, transform=transforms.ToTensor(), download=True)
test_set = dset.MNIST(root=root, train=False, transform=transforms.ToTensor(), download=True)
```

4. Plot 10 random MNIST images

```
import random

tensor_to_PIL = transforms.ToPILImage()

rand_idx_lst = random.sample(range(50000), 10)
fig, axes = plt.subplots(1, 10, figsize=(20, 5))

for idx in range(len(axes)):
    axes[idx].imshow(tensor_to_PIL(train_set[rand_idx_lst[idx]][0]))

plt.show()
plt.close()
```

5. Set hyper parameters for training the model

```
batch_size = 100
total_epoch = 10
learning_rate = 0.01
use_cuda = torch.cuda.is_available() # [Question 1] What is 'torch.cuda.is_available' used for?
```

6. Create DataLoader

```
train_loader = torch.utils.data.DataLoader(  # [Question 2] What is 'torch.utils.data.DataLoader' used for?
                 dataset=train_set,
                 batch_size=batch_size,
                 shuffle=True)
test_loader = torch.utils.data.DataLoader(
                dataset=test_set,
                batch_size=batch_size,
                shuffle=False)
```

7. Neural Network for MNIST image classification

```
class MLPNet(nn.Module):
    def __init__(self):
        super(MLPNet, self).__init__()
        self.fc1 = nn.Linear(28*28, 500) # [Question 3] What is 'nn.Linear' used for?
        self.fc2 = nn.Linear(500, 256)
        self.fc3 = nn.Linear(256, 10)
    def forward(self, x):
        x = x.view(-1, 28*28) # [Question 4] (4-1)What is 'view' used for? (4-2)What does '-1' mean?
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def name(self):
        return "MLP"
```

8. Create neural network object

```
model = MLPNet()

if use_cuda:
    model = model.cuda()
```

9. Set up optimizer and loss function for model training

```
optimizer = optim.SGD(model.parameters(), lr=learning_rate) # [Question 5] What is 'optim.SGD' used for?
criterion = nn.CrossEntropyLoss() # [Question 6] What is 'nn.CrossEntropyLoss' used for?
```

10. Training the model

```
for epoch in range(total_epoch):
    # trainning
    total_loss = 0
    total_batch = 0
    for batch_idx, (x, target) in enumerate(train_loader):
        optimizer.zero_grad() # [Question 7] What is 'optimizer.zero_grad()' used for?
        if use_cuda:
            x, target = x.cuda(), target.cuda()

        out = model(x)
        loss = criterion(out, target) # [Question 8] What are 'out' and 'target' and what is output of criterion(out, target)?
        total_loss += loss.item()
        total_batch += 1
        loss.backward() # [Question 9] What is 'loss.backward()' used for?
        optimizer.step() # [Question 10] What is 'optimizer.step()' used for?
        if (batch_idx+1) % 100 == 0 or (batch_idx+1) == len(train_loader):
            print ('==>>> epoch: {}, batch index: {}, train loss: {:.6f}'
                   .format(epoch, batch_idx+1, total_loss / total_batch))

    # testing
    total_loss = 0
    total_batch = 0
    correct_cnt = 0
    total_cnt = 0

    for batch_idx, (x, target) in enumerate(test_loader):
        if use_cuda:
            x, target = x.cuda(), target.cuda()

        out = model(x)
        loss = criterion(out, target)
        _, pred_label = torch.max(out.data, 1)
        total_cnt += x.data.size()[0]
        correct_cnt += (pred_label == target.data).sum().item()

        total_loss += loss.item()
        total_batch += 1

        if(batch_idx+1) % 100 == 0 or (batch_idx+1) == len(test_loader):
            print ('==>>> epoch: {}, batch index: {}, test loss: {:.6f}, acc: {:.3f}'
                   .format(epoch, batch_idx+1, total_loss / total_batch, correct_cnt * 1.0 / total_cnt))
```

In the given code, you can find ten questions. You must fill out an answer to each question. **The answer should be written separately in the report, not in the code.** All problems are 0.5 points each.

4 About the Submission

- The deadline for submission is **23:59 on 10 April (Fri)**, and late submission is not permitted.
- You have to submit only **.pdf file**.(Please convert .doc file to **.pdf file**)
- File name should be **[hw1]student_ID.zip** (e.g., [hw1]20191234.zip)
  (If you do not keep this naming, there will be a disadvantage.)
- If you have a question about hw1, **please use Lecture Q&A on KLMS or Classum.**