IE801/KSE801 Machine Learning for Knowledge Service
[HW1] Polynomial Approximation using Pytorch

Main assistant for this HW: *Jaehoon Oh*, jhoon.oh@kaist.ac.kr

- HW Description

■ This HW is about polynomial approximation using Pytorch. You can download the following code, which is [HW1] Polynomial Approximation using Pytorch.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 random seed for convenience.)

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

```
POLY_DEGREE = 4
torch.manual_seed(0)
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 = +7.70 x^4 -1.47 x^3 -10.89 x^2 +2.84 x^1 -5.42
```

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

```
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
```

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

```
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)
```

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:

```
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
```

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

```
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(): # this condition is about gpu availability
                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() # set gradient as 0
            loss.backward() # backpropagation
            optimizer.step() # update parameters (weights & bias)
```

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))
```

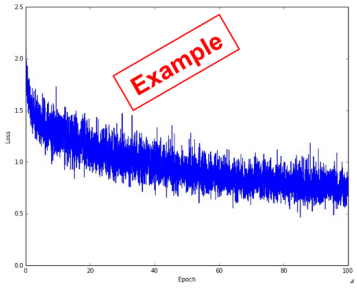
```
==> Initial function:   y = +0.00 x^4 +0.00 x^3 +0.00 x^2 +0.00 x^1 +0.00
==> Actual function:    y = +7.70 x^4 -1.47 x^3 -10.89 x^2 +2.84 x^1 -5.42
```

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

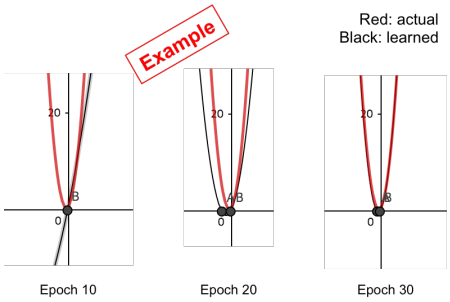
```
print('==> Learned function:\t' + poly_desc(net.fc.weight.data.view(-1), net.fc.bias.data))
print('==> Actual function:\t' + poly_desc(W_target.view(-1), b_target))
```

```
==> Learned function:   y = +7.67 x^4 -1.51 x^3 -11.04 x^2 +2.84 x^1 -5.43
==> Actual function:    y = +7.70 x^4 -1.47 x^3 -10.89 x^2 +2.84 x^1 -5.42
```

- **What you should do**
  - **Make a code for plotting some graphs that help your analysis. (5 pts)**
    1. Loss graph



    2. The real function and the learned function



  - **Analyze about convergence according to learning rate. (5pts)**
    (Including loss graph per epoch, and the real function and the learned function per 100 epochs)
    Fix other conditions as follow:
    - Loss function: SmoothL1
    - Optimizer: SGD
    - Number of epochs = 500
    - Dataset size = 200
    - Batch size = 50

    Control learning rate:
    - 0.1 / 0.01 / 0.001

  - **Analyze about convergence according to batch size. (5pts)**
    (Including loss graph per epoch, and the real function and the learned function per 100 epochs)
    Fix other conditions as follow:
    - Loss function: SmoothL1
    - Optimizer: SGD
    - Number of epochs = 500
    - Dataset size = 200
    - Learning rate = 0.01

    Control batch size:
    - 1 / 50 / 200

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

- **About the submission**
  - The due date is 27 March (Wed.). And, late submission is not permitted.
  - You have to submit .zip file including both .ipynb file and .pdf file.
    (PLEASE CONVERT .DOC FILE TO .PDF FILE)
  - File name should be **[hw1]your_student_number.zip** (e.g., [hw1]20191234.zip)
    (If you do not keep this naming, there will be a disadvantage.)

- **Pytorch official site will be a good material for you.**
  >> Link: https://pytorch.org/docs/stable/index.html → Especially, torch.nn & torch.optim
- **If you need a GPU machine, please use 'Google Colab.'**
  >> Link: https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d
- **If you have a question, please use Lecture Q&A on KLMS.**