assignment 08

SengHyun Lee

2019.12.12

Image Denoising

Develop a denoising algorithm based on an auto-encoder architecture using pytorch library in the supervised learning framework

- Denoising aims to reconstruct a clean image from a noisy observation
- We use a simple additive noise model using the Normal distribution:

  $f = u + \eta$

  where f denotes a noisy observation, u denotes a desired clean reconstruction, and η denotes a noise process following the normal distribution:

  $\eta \sim N(0, \sigma^2)$

  where $N(0, \sigma^2)$ denotes the normal distribution with mean 0 and standard deviation σ

Loss function

- My train loss function:

  $\ell(h, \hat{h}) = \frac{1}{m} \sum_{n=1}^{m} \| h_n - \hat{h}_n \|_2^2$

- h denotes a clean ground truth and $\hat{h}$ denotes an output of the network
- m denotes mini-batch size

Hyper Parameters

- learning rate : 1e-3
- batch size : 4
- optimizer : Adam Optimizer
- max number of epoch : 30

In [1]:

```
import torch
import random
import numpy as np
import torch.nn as nn
import torch.nn.init as init
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

import torchvision
import torch.optim as optim
import torchvision.transforms as transforms
```

```
from torch.autograd import Variable
import matplotlib.pyplot as plt
```

Implementations

In [2]:

```
# custom dataloader for .npy file
class numpyDataset(Dataset):
    def __init__(self, data, transform=None):
        self.data = torch.from_numpy(data).float()
        self.transform = transform

    def __getitem__(self, index):
        x = self.data[index]
        if self.transform:
            x = self.transform(x)

        return x

    def __len__(self):
        return len(self.data)
```

In [3]:

```
def output_plot(g1, std, title, color, scale, label, legend):
    plt.title(title)
    plt.plot(np.arange(1, len(g1) + 1), g1, color=color[0], alpha=0.5, label=label)
    if scale is not None:
        plt.yscale(scale)
    if std is not None:
        plt.fill_between(np.arange(1, len(g1)+1),
                         np.array(g1)-np.array(std),
                         np.array(g1)+np.array(std), color=color[0], alpha=0.3)
    plt.legend(loc=legend)
    plt.show()
```

Network Architecture

In [4]:

```
class DenoiseNetwork(nn.Module):
    def __init__(self, depth=17, n_channels=64, image_channels=1, kernel_size=3):
        super(DenoiseNetwork, self).__init__()
        kernel_size = 3
        padding = 1
        encoder_layers = []

        encoder_layers.append(nn.Conv2d(in_channels=image_channels,␣
↪out_channels=n_channels, kernel_size=kernel_size, padding=padding, bias=True))
        encoder_layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            encoder_layers.append(nn.Conv2d(in_channels=n_channels,␣
↪out_channels=n_channels, kernel_size=kernel_size, padding=padding, bias=False))
            encoder_layers.append(nn.BatchNorm2d(n_channels, momentum = 0.95))
            encoder_layers.append(nn.ReLU(inplace=True))
        encoder_layers.append(nn.Conv2d(in_channels=n_channels,␣
↪out_channels=image_channels, kernel_size=kernel_size, padding=padding, bias=False))

        self.auto_encode = nn.Sequential(*encoder_layers)

        self._initialize_weights()

    def forward(self, x):
        y = x
        out = self.auto_encode(x)
        return y-out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)
```

In [5]:

```
# import model
model = DenoiseNetwork()
model.cuda()



learning_rate = 1e-3
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[30, 60, 90], gamma=0.2)
objective = nn.MSELoss(reduction = 'sum')



loss_train = []
to_img = transforms.ToPILImage()
```

In [6]:

```
NUM_EPOCH = 30

transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Grayscale(num_output_channels=1),
    transforms.ToTensor(),
])

# for training
traindata = np.load('train.npy')
traindataset = numpyDataset(traindata, transform)
trainloader = DataLoader(traindataset, batch_size=4, shuffle=True, drop_last=True,␣
 ↪num_workers=2)

loss_train_mean, loss_train_std = [], []
prev_train_loss, next_train_loss = 0, 0
```

In [7]:

```
def train(batch_idx, epoch, clean_image, noisy_image):
    clean, noisy = Variable(clean_image).cuda(), Variable(noisy_image).cuda()

    optimizer.zero_grad()
    output = model(noisy)

    loss = objective(output, clean)

    loss.backward()
    optimizer.step()
    scheduler.step(epoch)

    loss_train_batch = loss.item() / len(data)

    return loss_train_batch
```

Train & Plots

In [ ]:

```
model.train()
for epoch in range(NUM_EPOCH):

    batch_train_loss = []

    for batch_idx, data in enumerate(trainloader):

        # Noisy Images #
        samples = [
            data + (0.01 * torch.randn(len(data), 1, 120, 80)),
            data + (0.02 * torch.randn(len(data), 1, 120, 80)),
            data + (0.03 * torch.randn(len(data), 1, 120, 80)),
            data + (0.04 * torch.randn(len(data), 1, 120, 80))
        ]
        ########

        loss_train = train(batch_idx, epoch, data, samples[random.randint(0, 3)])
        batch_train_loss.append(loss_train)

    loss_train_mean.append(np.mean(batch_train_loss))
    loss_train_std.append(np.std(batch_train_loss))

    print("[epoch %s] loss(training): %s" % (epoch, loss_train_mean[-1]))
```

```
[epoch 0] loss(training): 38.135274913907054
[epoch 1] loss(training): 4.239182426225056
[epoch 2] loss(training): 3.592938362956047
[epoch 3] loss(training): 3.439420774633234
[epoch 4] loss(training): 3.1291519671407615
[epoch 5] loss(training): 3.0552961523695426
[epoch 6] loss(training): 2.8136225374720314
[epoch 7] loss(training): 2.7243282652172174
[epoch 8] loss(training): 2.59743206016042
[epoch 9] loss(training): 2.2674315757914023
```

Graph

In [ ]:

```
output_plot(loss_train_mean, std=None, title="Loss", scale=None, color=('blue'),␣
 ↪label='train loss', legend='upper right')
```

In [ ]:

```
output_plot(loss_train_mean, std=loss_train_std, title="Loss with std (log scale)",␣
 ↪color=('blue'), scale='log', label='train loss with std', legend='upper right')
```

Training Visualization

In [ ]:

```
for batch_idx, data in enumerate(trainloader):
        # Noisy Images #
        sample = data + (0.04 * torch.randn(len(data), 1, 120, 80))

        fig, ax = plt.subplots(ncols=3, nrows=1, figsize=(9, 7))
        ax[0].set_title("Clean")
        ax[1].set_title("Noisy(input)")
        ax[2].set_title("Denoised")

        ax[0].imshow(to_img(data[0].cpu()), cmap='gray')
        ax[1].imshow(to_img(sample[0].cpu()), cmap='gray')
        ax[2].imshow(to_img(model(Variable(data).cuda())[0].cpu()), cmap='gray')

        break
```

Testing (Evaluation)

In [ ]:

```
# for testing
testdata = np.load('test.npy')
testdataset = numpyDataset(testdata, transform)
testloader = DataLoader(testdataset, batch_size=1, shuffle=False, drop_last=False,␣
 ↪num_workers=2)

result_for_submit = None  # this is for submit file

model.eval()
for batch_idx, data in enumerate(testloader):
    result_of_test = data

    if batch_idx == 0:
        result_for_submit = result_of_test
    else:
        try:
            result_for_submit = torch.cat([result_for_submit, result_of_test], dim=0)

        except RuntimeError:
            transposed = torch.transpose(result_of_test, 2, 3)
            result_for_submit = torch.cat([result_for_submit, transposed], dim=0)

# the submit_file.shape must be (400,1,120,80)
submit_file = result_for_submit.detach().numpy()
print(submit_file.shape)

np.save('20142921_08_SengHyun_Lee.npy', submit_file)
```

Testing Visualization

In [ ]:

```
for batch_idx, data in enumerate(testloader):

        fig, ax = plt.subplots(ncols=2, nrows=1, figsize=(9, 7))
        ax[0].set_title("Noisy")
        ax[1].set_title("Denoised")

        with torch.no_grad():
            fig, ax = plt.subplots(ncols=2, nrows=1, figsize=(7,9))
            result_of_test = model.forward(data.cuda())
            ax[0].imshow(to_img(data[0].cpu()), cmap='gray')
            ax[1].imshow(to_img(result_of_test[0].cpu()), cmap='gray')
            fig.show()

        break
```