Machine Learning Project 08

December 11, 2019

1 Denoising algorithm based on an auto-encoder architecture using pytorch library

Develop a denoising algorithm based on an auto-encoder architecture using pytorch library in tl

[23]:
```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

import torchvision.transforms as transforms
from torch.autograd import Variable
import matplotlib.pyplot as plt
plt.rcParams.update({'figure.max_open_warning': 0})
import torchvision
```

- Class numpyDataset
  - custom dataloader for .npy file

[24]:
```
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)
```

1.1 Addictive noise to image:

- Denoising aims to reconstruct a clean image from a noisy observation
- We use a simple additive noise model using the Normal distribution: $f=u+ $
- f denotes a noisy observation, u denotes a desired clean reconstruction, and $ $ denotes
- $N(0, 2)$ denotes the normal distribution with mean 0 and standard deviation $ $

[25]:
```
def add_noise(img):
    mu=0
    sigma_list=[0.01,0.02,0.03,0.04]
    sigma= np.random.choice(sigma_list, 1, replace=True, p=None)
    noise= np.random.normal(mu, sigma[0])
    noisy_img = img + noise
    return noisy_img
```

1.2 Auto-Encoder:

- Build an auto-encoder architecture based on the convolutional neural network using pytorch
- 2-Layer Convolutional Neural Network

[189]:
```
import torch.nn as nn
import torch.nn.functional as F

class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding=1)
        self.conv3 = nn.Conv2d(16, 8, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.t_conv1 = nn.ConvTranspose2d(8, 8, 2, stride=2)
        self.t_conv3 = nn.ConvTranspose2d(8, 16, 2, stride=2)
        self.conv_out = nn.Conv2d(16, 1, 3, padding=1)


    def forward(self, x):
        ## encode ##
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)

        ## decode ##
        x = F.relu(self.t_conv1(x))
        x = F.relu(self.t_conv3(x))
        x = F.sigmoid(self.conv_out(x))

        return x
```

1.3 Initialization and Optimization for Train.

- Batch Size = 120
- Optimizer -> Adam
- Learning Rate= 0.01
- Weight Decay = 0.0001
- Shuffle: allow for Train.npy, Not allow for Test.npy
- Print text about average of Loss at each EPOCH

[190]:
```
NUM_EPOCH       = 40
transform       = transforms.Compose([transforms.ToPILImage(),transforms.
 ↪Grayscale(num_output_channels=1),transforms.ToTensor(),])

batch_size = 120
learning_rate = 0.001


# for training
traindata       = np.load('train.npy')
traindataset    = numpyDataset(traindata, transform)
trainloader     = DataLoader(traindataset, batch_size=batch_size, shuffle=True,␣
 ↪num_workers=0)


device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = autoencoder().to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam( model.parameters(), lr=learning_rate,␣
 ↪weight_decay=0.0001)


loss_train_mean        = np.zeros(NUM_EPOCH)
loss_train_std         = np.zeros(NUM_EPOCH)



for epoch in range(NUM_EPOCH):
    loss_list = []

    for batch_idx, data in enumerate(trainloader):
        imagearr   = data
        noisy_img = add_noise(imagearr)


        optimizer.zero_grad()
        # ==================forward====================
        output = model(noisy_img)
        loss = criterion(output, imagearr)
```

```
        # ==================backward====================
        loss.backward()
        optimizer.step()

        loss_list.append(loss.item())


    loss_train_mean[epoch]     = np.mean(loss_list)
    print("""[EPOCH %5d ] LOSS     :(TRAIN) %3.
↪10f"""%(epoch,loss_train_mean[epoch]))
    loss_train_std[epoch]      = np.std(loss_list)
```

```
[EPOCH     0 ] LOSS     :(TRAIN) 0.0198780317
[EPOCH     1 ] LOSS     :(TRAIN) 0.0160668519
[EPOCH     2 ] LOSS     :(TRAIN) 0.0069088947
[EPOCH     3 ] LOSS     :(TRAIN) 0.0051031622
[EPOCH     4 ] LOSS     :(TRAIN) 0.0033200436
[EPOCH     5 ] LOSS     :(TRAIN) 0.0033183907
[EPOCH     6 ] LOSS     :(TRAIN) 0.0035144612
[EPOCH     7 ] LOSS     :(TRAIN) 0.0027366647
[EPOCH     8 ] LOSS     :(TRAIN) 0.0031634336
[EPOCH     9 ] LOSS     :(TRAIN) 0.0032307781
[EPOCH    10 ] LOSS     :(TRAIN) 0.0031741600
[EPOCH    11 ] LOSS     :(TRAIN) 0.0042520328
[EPOCH    12 ] LOSS     :(TRAIN) 0.0031634021
[EPOCH    13 ] LOSS     :(TRAIN) 0.0027373981
[EPOCH    14 ] LOSS     :(TRAIN) 0.0035006414
[EPOCH    15 ] LOSS     :(TRAIN) 0.0035905185
[EPOCH    16 ] LOSS     :(TRAIN) 0.0030672141
[EPOCH    17 ] LOSS     :(TRAIN) 0.0030138792
[EPOCH    18 ] LOSS     :(TRAIN) 0.0028545566
[EPOCH    19 ] LOSS     :(TRAIN) 0.0029190760
[EPOCH    20 ] LOSS     :(TRAIN) 0.0028220255
[EPOCH    21 ] LOSS     :(TRAIN) 0.0033480119
[EPOCH    22 ] LOSS     :(TRAIN) 0.0027349665
[EPOCH    23 ] LOSS     :(TRAIN) 0.0030449729
[EPOCH    24 ] LOSS     :(TRAIN) 0.0029228451
[EPOCH    25 ] LOSS     :(TRAIN) 0.0028262719
[EPOCH    26 ] LOSS     :(TRAIN) 0.0027430938
[EPOCH    27 ] LOSS     :(TRAIN) 0.0029727122
[EPOCH    28 ] LOSS     :(TRAIN) 0.0028346914
[EPOCH    29 ] LOSS     :(TRAIN) 0.0029729806
[EPOCH    30 ] LOSS     :(TRAIN) 0.0029953579
[EPOCH    31 ] LOSS     :(TRAIN) 0.0026182085
[EPOCH    32 ] LOSS     :(TRAIN) 0.0034177949
[EPOCH    33 ] LOSS     :(TRAIN) 0.0023692602
```

```
[EPOCH    34 ] LOSS     :(TRAIN) 0.0026975823
[EPOCH    35 ] LOSS     :(TRAIN) 0.0027421425
[EPOCH    36 ] LOSS     :(TRAIN) 0.0026022885
[EPOCH    37 ] LOSS     :(TRAIN) 0.0033238973
[EPOCH    38 ] LOSS     :(TRAIN) 0.0026635648
[EPOCH    39 ] LOSS     :(TRAIN) 0.0028610485
```

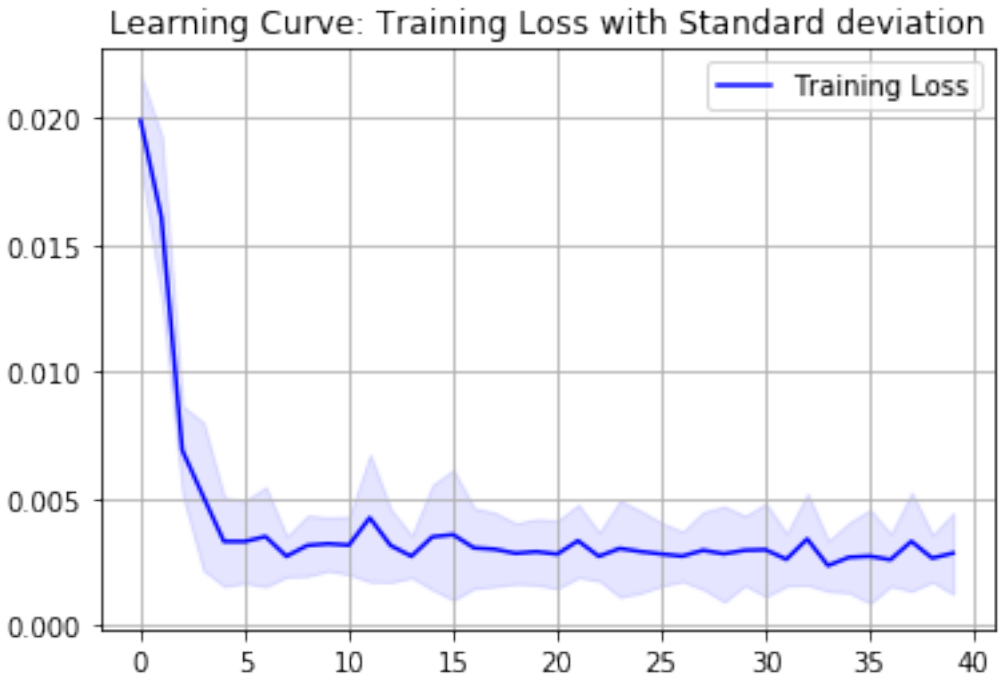
1.4 [Learning Curve] : Training Loss with Standard Deviation.

[191]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.svm import SVC
from sklearn.datasets import load_digits


train_sizes=np.array(range(NUM_EPOCH))


plt.title("Learning Curve: Training Loss with Standard deviation")
plt.grid()
plt.fill_between(train_sizes,loss_train_mean  - loss_train_std, loss_train_mean␣
 ↪+ loss_train_std, alpha=0.1,color="b")
plt.plot(train_sizes, loss_train_mean, color="b",label="Training Loss")
plt.legend(loc='upper right')

plt.show()
```



1.5 Ouput of .npy file for TEST

- Not Allow to Shuffle
- Result for submit shape [400,1,120,80]

[197]:
```
# for testing
testdata        = np.load('test.npy')
testdataset     = numpyDataset(testdata, transform)
testloader      = DataLoader(testdataset, batch_size=40, shuffle=False,␣
↪num_workers=0)

result_for_submit = None    # this is for submit file

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

submit_file = result_for_submit.detach().numpy()
np.save('YoungminKim.npy', submit_file)
```