hw11_code_output

June 10, 2024

[ ]:
```
import sys
import time

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import torch

%matplotlib inline
print(f"Python version: {sys.version}\nNumpy version: {np.
 ↪__version__}\nMatplotlib version: {matplotlib.__version__}\nPyTorch version:␣
 ↪{torch.__version__}")
```

```
Python version: 3.11.8 | packaged by conda-forge | (main, Feb 16 2024, 20:49:36)
[Clang 16.0.6 ]
Numpy version: 1.26.4
Matplotlib version: 3.8.0
PyTorch version: 2.3.0
```

1 Problem 3

[ ]:
```
MNIST_DATA_PATH = "/Users/lucah/Library/CloudStorage/OneDrive-DurhamUniversity/
 ↪Course Material & Work/SNU Year Abroad {SNU}/2-Spring Semester/Mathematical␣
 ↪Foundations of Deep Neural Networks {MFDNN}/Lectures Slides {MFDNN}/
 ↪Notebooks {MFDNN}/mnist_data"
NICE_MODEL_PATH = "/Users/lucah/Library/CloudStorage/OneDrive-DurhamUniversity/
 ↪Course Material & Work/SNU Year Abroad {SNU}/2-Spring Semester/Mathematical␣
 ↪Foundations of Deep Neural Networks {MFDNN}/Homeworks {MFDNN}/nice.pt"
```

1.1 Given steps

1.2 Steps 1+2

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets
```

```
from torchvision.transforms import transforms
```

```
/opt/anaconda3/envs/MFDNN/lib/python3.11/site-
packages/torchvision/io/image.py:13: UserWarning: Failed to load image Python
extension: 'dlopen(/opt/anaconda3/envs/MFDNN/lib/python3.11/site-
packages/torchvision/image.so, 0x0006): Symbol not found:
__ZN3c1017RegisterOperatorsD1Ev
  Referenced from: <CFED5F8E-EC3F-36FD-AAA3-2C6C7F8D3DD9>
/opt/anaconda3/envs/MFDNN/lib/python3.11/site-packages/torchvision/image.so
  Expected in:     <E6933B13-F4A0-3821-8466-03F22A3B3739>
/opt/anaconda3/envs/MFDNN/lib/python3.11/site-
packages/torch/lib/libtorch_cpu.dylib'If you don't plan on using image
functionality from `torchvision.io`, you can ignore this warning. Otherwise,
there might be something wrong with your environment. Did you have `libjpeg` or
`libpng` installed before building `torchvision` from source?
  warn(
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device = "mps"
batch_size = 128

'''
Step 1:
'''
test_val_dataset = datasets.MNIST(root=MNIST_DATA_PATH,
                                  train=False,
                                  transform=transforms.ToTensor())

test_dataset, validation_dataset = \
    torch.utils.data.random_split(test_val_dataset, [5000, 5000])

# KMNIST dataset, only need test dataset
anomaly_dataset = datasets.KMNIST(root='./kmnist_data/',
                                  train=False,
                                  transform=transforms.ToTensor(),
                                  download=True)

'''
Step 2:
'''
# Define prior distribution
class Logistic(torch.distributions.Distribution):
    def __init__(self):
        super(Logistic, self).__init__()

    def log_prob(self, x):
        return -(F.softplus(x) + F.softplus(-x))
```

```
    def sample(self, size):
        z = torch.distributions.Uniform(0., 1.).sample(size).to(device)
        return torch.log(z) - torch.log(1. - z)

# Implement coupling layer
class Coupling(nn.Module):
    def __init__(self, in_out_dim, mid_dim, hidden, mask_config):
        super(Coupling, self).__init__()
        self.mask_config = mask_config

        self.in_block = \
            nn.Sequential(nn.Linear(in_out_dim//2, mid_dim), nn.ReLU())
        self.mid_block = nn.ModuleList(
            [nn.Sequential(nn.Linear(mid_dim, mid_dim), nn.ReLU())
             for _ in range(hidden - 1)])
        self.out_block = nn.Linear(mid_dim, in_out_dim//2)

    def forward(self, x, reverse=False):
        [B, W] = list(x.size())
        x = x.reshape((B, W//2, 2))
        if self.mask_config:
            on, off = x[:, :, 0], x[:, :, 1]
        else:
            off, on = x[:, :, 0], x[:, :, 1]

        off_ = self.in_block(off)
        for i in range(len(self.mid_block)):
            off_ = self.mid_block[i](off_)
        shift = self.out_block(off_)

        if reverse:
            on = on - shift
        else:
            on = on + shift

        if self.mask_config:
            x = torch.stack((on, off), dim=2)
        else:
            x = torch.stack((off, on), dim=2)
        return x.reshape((B, W))

class Scaling(nn.Module):
    def __init__(self, dim):
        super(Scaling, self).__init__()
        self.scale = nn.Parameter(torch.zeros((1, dim)))
```

```
    def forward(self, x, reverse=False):
        log_det_J = torch.sum(self.scale)
        if reverse:
            x = x * torch.exp(-self.scale)
        else:
            x = x * torch.exp(self.scale)
        return x, log_det_J

class NICE(nn.Module):
    def __init__(self,in_out_dim, mid_dim, hidden,
                 mask_config=1.0, coupling=4):
        super(NICE, self).__init__()
        self.prior = Logistic()
        self.in_out_dim = in_out_dim

        self.coupling = nn.ModuleList([
            Coupling(in_out_dim=in_out_dim,
                     mid_dim=mid_dim,
                     hidden=hidden,
                     mask_config=(mask_config+i)%2) \
            for i in range(coupling)])

        self.scaling = Scaling(in_out_dim)

    def g(self, z):
        x, _ = self.scaling(z, reverse=True)
        for i in reversed(range(len(self.coupling))):
            x = self.coupling[i](x, reverse=True)
        return x

    def f(self, x):
        for i in range(len(self.coupling)):
            x = self.coupling[i](x)
        z, log_det_J = self.scaling(x)
        return z, log_det_J

    def log_prob(self, x):
        z, log_det_J = self.f(x)
        log_ll = torch.sum(self.prior.log_prob(z), dim=1)
        return log_ll + log_det_J

    def sample(self, size):
        z = self.prior.sample((size, self.in_out_dim)).to(device)
        return self.g(z)

    def forward(self, x):
        return self.log_prob(x)
```

1.2.1 Step 3: Load the pretrained model

[ ]:
```
nice = NICE(in_out_dim=784, mid_dim=1000, hidden=5).to(device)
nice.load_state_dict(torch.load(NICE_MODEL_PATH, map_location=device))
```

```
/opt/anaconda3/envs/MFDNN/lib/python3.11/site-
packages/torch/distributions/distribution.py:53: UserWarning: <class
'__main__.Logistic'> does not define `arg_constraints`. Please set
`arg_constraints = {}` or initialize the distribution with `validate_args=False`
to turn off validation.
  warnings.warn(
```

[ ]:
```
<All keys matched successfully>
```

1.3 Tasks

1.3.1 Step 4: Calculate standard deviation by using validation set

[ ]:
```
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset,␣
 ↪batch_size=batch_size)

log_p_thetas = torch.tensor([]).to(device)
for images, _ in validation_loader:
    log_lls = nice(images.view(-1, 784).to(device))
    log_p_thetas = torch.cat((log_p_thetas, log_lls))

std = torch.std(log_p_thetas)
mean = torch.mean(log_p_thetas)

threshold = mean - 3 * std
```

1.3.2 Step 5: Anomaly detection (mnist)

[ ]:
```
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,␣
 ↪batch_size=batch_size)
count = 0
for images, _ in test_loader:
    log_lls: torch.Tensor = nice(images.view(-1, 784).to(device))
    count += (log_lls < threshold).count_nonzero()  # images classed as␣
 ↪anomalies (i.e. below the threshold)
print(f'{count} type I errors among {len(test_dataset)} data')
```

```
66 type I errors among 5000 data
```

1.3.3 Step 6: Anomaly detection (kmnist)

[ ]:
```
anomaly_loader = torch.utils.data.DataLoader(dataset=anomaly_dataset,␣
 ↪batch_size=batch_size)
count = 0
for images, _ in anomaly_loader:
    log_lls: torch.Tensor = nice(images.view(-1, 784).to(device))
    count += (log_lls >= threshold).count_nonzero()  # images classed as␣
 ↪non-anomalies (i.e. above threshold)
print(f'{count} type II errors among {len(anomaly_dataset)} data')
```

```
20 type II errors among 10000 data
```