hw7

November 18, 2021

1 Mathematical Foundations on Deep Neural Network

1.1 Homework #7

1.1.1 2017-11362

1.2 Problem 1.

Consider $T_1 : \mathbb{R}^{2\times 2} \to \mathbb{R}$. Then, $\exists$ corresp. matrix A s.t $T_1(X) = AX$. $\Rightarrow T_1^T(Y) = A^T Y$.

In here, $A = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. $\Rightarrow A^T = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})^t$.

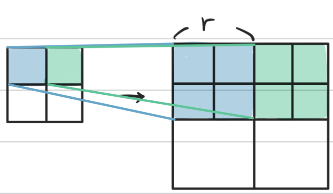
$\therefore T_1^T : \mathbb{R} \to \mathbb{R}^{2\times 2}$ is the upsampling operator s.t $\boxed{y} \to \begin{array}{|c|c|} \hline y/4 & y/4 \\ \hline y/4 & y/4 \\ \hline \end{array}$: $\frac{1}{4}\times$ nearest neighbor upsampling

Since T is the avg. pool with 2x2 kernel and stride 2, T_1 is a module of T.

In other words, $T = \begin{bmatrix} T_1 & \cdots & T_1 \\ \vdots & \ddots & \vdots \\ T_1 & \cdots & T_1 \end{bmatrix}$, $T^T = \begin{bmatrix} T_1^T & \cdots & T_1^T \\ \vdots & \ddots & \vdots \\ T_1^T & \cdots & T_1^T \end{bmatrix}$

$\therefore T^T$ is $\frac{1}{4} \times$ upsample(scale_factor =2, mode='nearest').

1.3 Problem 2.



$\Rightarrow$ stride=r , kernel_size =r

weight : only "1"

Assume input : (Batch, Channel, Height, Width).

layer = nn. ConvTranspose2d (Channel, Channel, r, r)

layer.weight.data = torch. ones_like (layer.weight. data) $\Leftrightarrow$ layer = nn. Upsample (scale_factor = r).

1.4 Problem 3.

$v_\beta(x) = \frac{1}{\beta} \log \sum_{i=1}^{n} \exp(\beta x_i)$

(a) $v_\beta(x) \longrightarrow \max\{x_1, \cdots, x_n\}$ as $\beta \to \infty$

Let $x_j = \max\{x_1, \cdots, x_n\}$.

$$\lim_{\beta\to\infty} v_\beta(x) = \lim_{\beta\to\infty} \frac{\log \sum \exp(\beta x_i)}{\beta} = \lim_{\beta\to\infty} \frac{\sum x_i \exp(\beta x_i)}{\sum \exp(\beta x_i)} \quad \text{(L'hospital rule)}$$

$$e^{\beta x_j} \geq e^{\beta x_1}, \cdots, e^{\beta x_{j-1}}, e^{\beta x_{j+1}}, \cdots, e^{\beta x_n} \Rightarrow \lim_{\beta\to\infty} \frac{\exp(\beta x_i)}{\exp(\beta x_j)} = \begin{cases} 0 & , x_j > x_i \\ 1 & , x_j = x_i \end{cases}$$

$$\lim_{\beta\to\infty} \frac{\sum x_i \exp(\beta x_i)}{\sum \exp(\beta x_i)} = \lim_{\beta\to\infty} \frac{\sum_{i=1}^{n} x_i \frac{\exp(\beta x_i)}{\exp(\beta x_j)}}{\sum_{i=1}^{n} \frac{\exp(\beta x_i)}{\exp(\beta x_j)}} = \frac{x_j |A|}{|A|} = x_j \quad (A = \{ i : x_i = x_j, 1 \leq i \leq n \})$$

$\therefore \lim_{\beta\to\infty} v_\beta(x) = \max\{x_1, \cdots, x_n\}$.

(b) $\nabla v_1 = \mu$, μ : softmax.

$$v_1 = \log \sum_{i=1}^{n} \exp(x_i) , \quad \frac{\partial}{\partial x_i} v_1 = \exp(x_i) \Big/ \sum_{i=1}^{n} \exp(x_i)$$

$$\therefore \nabla v_1 = \left(\frac{\partial}{\partial x_1} v_1, \cdots, \frac{\partial}{\partial x_n} v_n \right) = \exp(x) \Big/ \sum_{i=1}^{n} \exp(x_i) = \mu.$$

(c) $i_{max} = \operatorname*{argmax}_{1 \leq i \leq n} x_i$: uniquely defined $\Rightarrow$ $\nabla v_\beta(x) \to e_{i_{max}}$ as $\beta \to \infty$

$$\frac{\partial}{\partial x_j} v_\beta(x) = \frac{\beta \exp(\beta x_j)}{\beta \sum \exp(\beta x_i)} = \frac{\exp(\beta x_j)}{\sum \exp(\beta x_i)} \to \begin{cases} 0 & , j \neq i_{max} \\ 1 & , j = i_{max} \end{cases} \quad \text{(Same logic with (a))}$$

$$\therefore \nabla v_\beta(x) \to (0, \cdots, \underset{i_{max}\text{th}}{1}, \cdots, 0) = e_{i_{max}}.$$

1.5 Problem 4.

$F : \mathbb{R} \to [0,1]$ CDF (nondecreasing, right-continuous), $U \sim \text{Unif}[0,1]$.

$G(u) = \inf\{x \in \mathbb{R} : u \le F(x)\}$.

$G(U) \sim F \iff P(G(U) \le x) = F(x)$.

① $P(G(U) \le x) \le F(x)$.

$G(u) \le x \Rightarrow F(G(u)) \le F(x)$

F: right-continuous $\Rightarrow \{x : u \le F(x)\}$: closed $\Rightarrow u \le F(G(u))$

$\Rightarrow u \le F(x)$, which means $\{u : G(u) \le x\} \subseteq \{u : u \le F(x)\}$, $P(G(U) \le x) \le F(x)$.

② $P(G(U) \le x) \ge F(x)$.

$u_1 \le u_2 \Rightarrow \{x : u_1 \le F(x)\} \supseteq \{x : u_2 \le F(x)\} \Rightarrow G(u_1) \le G(u_2)$: G is nondecreasing

$u \le F(x) \Rightarrow G(u) \le G(F(x))$

$G(F(x)) = \inf\{y : F(x) \le F(y)\}$, $x \in \{y : F(x) \le F(y)\} \Rightarrow G(F(x)) \le x$.

$\Rightarrow G(u) \le x$, which means $\{u : u \le F(x)\} \subseteq \{u : G(u) \le x\}$, $F(x) \le P(G(U) \le x)$ $\square$.

1.6 Problem 5.

$X \sim P_X$, $Y \sim P_Y$. $D_f(X\|Y) = \int f\left(\frac{P_X(x)}{P_Y(x)}\right) P_Y(x)\, dx$. f: convex, $f(1) = 0$.

(a) $D_f(X\|Y) \ge 0$.

f is convex $\Rightarrow f(x) \ge f'(1)(x-1) + f(1) = f'(1)(x-1)$

$\int f\left(\frac{P_X(x)}{P_Y(x)}\right) P_Y(x)\, dx \ge \int f'(1)\left(\frac{P_X(x)}{P_Y(x)} - 1\right) P_Y(x)\, dx = f'(1)\int P_X(x)\, dx - f'(1)\int P_Y(x)\, dx = f'(1) - f'(1) = 0$.

(b) $f = -\log t$, $f = t\log t$.

$D_{-\log t}(X\|Y) = \int -\log\left(\frac{P_X(x)}{P_Y(x)}\right) P_Y(x)\, dx = \int \log\left(\frac{P_Y(x)}{P_X(x)}\right) P_Y(x)\, dx = D_{KL}(Y\|X)$.

$D_{t\log t}(X\|Y) = \int \frac{P_X(x)}{P_Y(x)} \log\left(\frac{P_X(x)}{P_Y(x)}\right) P_Y(x)\, dx = \int \log\left(\frac{P_X(x)}{P_Y(x)}\right) P_X(x)\, dx = D_{KL}(X\|Y)$.

1.7 Problem 6.

(a) $v_L = 1$, $v_\ell = \frac{\partial y_L}{\partial y_\ell} \operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell 1_{n_\ell}))$, $\ell = 1, \cdots, L-1$.

$\frac{\partial y_L}{\partial y_{L-1}} = \frac{\partial}{\partial y_{L-1}} (A_{w_L} y_{L-1} + b_L) = A_{w_L}$. HW 4-6 $\Rightarrow$ $\frac{\partial y_\ell}{\partial y_{\ell-1}} = \operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell 1_{n_\ell})) A_{w_\ell}$

$\frac{\partial y_L}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell} \cdot \frac{\partial y_\ell}{\partial b_\ell} = \frac{\partial y_L}{\partial y_\ell} \frac{\partial}{\partial b_\ell} \left(\sigma(A_{w_\ell} y_{\ell-1} + b_\ell 1_{n_\ell}) \right) = \frac{\partial y_L}{\partial y_\ell} \cdot \operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell 1_{n_\ell})) 1_{n_\ell} = v_\ell 1_{n_\ell}$. , $\ell = 1, \cdots, L-1$.

$\frac{\partial y_L}{\partial b_L} = 1 = v_L \cdot 1_{n_L}$.

Let's show $\frac{\partial y_L}{\partial w_\ell} = (C_{v_\ell^T} y_{\ell-1})$

① $\ell = L$. $\frac{\partial y_L}{\partial w_L}$.

$\frac{\partial y_L}{\partial w_L} = \frac{\partial}{\partial w_L}(A_{w_L} y_{L-1} + b_L) = \frac{\partial}{\partial w_L}(w_L^T \cdot y_{L-1} + b_L) = y_{L-1}^T = (C_{v_L^T} y_{L-1})^T$.

② $\ell = 1, \cdots, L-1$

$\frac{\partial y_L}{\partial w_\ell} = \frac{\partial y_L}{\partial y_\ell} \cdot \frac{\partial y_\ell}{\partial w_\ell}$

$$y_\ell = \sigma(A_{w_\ell} y_{\ell-1} + b_\ell \cdot 1_{n_\ell}) = \begin{bmatrix} \vdots \\ \sigma\left(\sum_j (A_{w_\ell})_{ij} (y_{\ell-1})_j + (b_\ell)_i\right) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \sigma\left(\sum_j (w_\ell)_j \cdot (y_{\ell-1})_{j+i-1} + (b_\ell)_i\right) \\ \vdots \end{bmatrix}$$

$$\frac{\partial y_\ell}{\partial (w_\ell)_j} = \begin{bmatrix} \vdots \\ \sigma'((A_{w_\ell} y_{\ell-1} + b_\ell)_i) \cdot (y_{\ell-1})_{i+j-1} \\ \vdots \end{bmatrix}$$: jth column of $\frac{\partial y_\ell}{\partial w_\ell}$.

$\frac{\partial y_\ell}{\partial w_\ell} = \operatorname{diag}(\sigma'(A_{w_\ell} y_{\ell-1} + b_\ell)) \cdot (C^*_{y_{\ell-1}})^T$

$\frac{\partial y_L}{\partial w_\ell} = \frac{\partial y_L}{\partial y_\ell} \cdot \frac{\partial y_\ell}{\partial w_\ell} = v_\ell \cdot (C^*_{y_{\ell-1}})^T = (C^*_{y_{\ell-1}} \cdot v_\ell^T)^T = (C_{v_\ell^T} \cdot y_{\ell-1})^T$. $\square$

$$C^*_{y_{\ell-1}} = \overbrace{\begin{bmatrix} (y_{\ell-1})_1 & (y_{\ell-1})_2 & \cdots & \\ (y_{\ell-1})_2 & (y_{\ell-1})_3 & \cdots & \\ & & \cdots & (y_{\ell-1})_{n_\ell} \end{bmatrix}}^{n_\ell} \Big\} f_\ell$$

(b) ① forward : we need matrix-vector product at $A_{w_\ell} \cdot y_{\ell-1}$

② backward
- $\frac{\partial y_L}{\partial w_\ell}$: mat-vec at $C_{v_\ell^T} \cdot y_{\ell-1}$, $\frac{\partial y_L}{\partial y_\ell}$: vec-mat at $\frac{\partial y_L}{\partial y_{\ell+1}} \cdot \frac{\partial y_{\ell+1}}{\partial y_\ell} = \left(A_{w_\ell}^T \operatorname{diag}(\sigma'(\sigma^{-1}(y_\ell))) \cdot \frac{\partial y_L}{\partial y_{\ell+1}} \right)^T$
- $\frac{\partial y_L}{\partial b_\ell}$: no such mat-vec / vec-mat.

1.8 Problem 7.

[1]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets
import torch.optim as optim
from torchvision.transforms import transforms
from torchvision.utils import save_image

import numpy as np
import matplotlib.pyplot as plt

lr = 0.001
batch_size = 100
epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
'''
Step 1:
'''

# MNIST dataset
dataset = datasets.MNIST(root='./mnist_data/',
                               train=True,
                               transform=transforms.ToTensor(),
                               download=True)

train_dataset, validation_dataset = torch.utils.data.random_split(dataset,␣
 ↪[50000, 10000])

test_dataset = datasets.MNIST(root='./mnist_data/',
                              train=False,
                              transform=transforms.ToTensor())

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

# print(len(train_dataset))  # 50000
# print(len(validation_dataset))  # 10000
# print(len(test_dataset))  # 10000
# print(len(anomaly_dataset))  # 10000
```

```
/home/zendo/anaconda3/lib/python3.8/site-
packages/torchvision/datasets/mnist.py:498: UserWarning: The given NumPy array
is not writeable, and PyTorch does not support non-writeable tensors. This means
you can write to the underlying (supposedly non-writeable) NumPy array using the
tensor. You may want to copy the array to protect its data or make it writeable
before converting it to a tensor. This type of warning will be suppressed for
the rest of this program. (Triggered internally at
../torch/csrc/utils/tensor_numpy.cpp:189.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

[2]:
```
'''
Step 2: AutoEncoder
'''
# Define Encoder
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(784, 256)
```

```python
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 32)
    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        z = F.relu(self.fc3(x))
        return z

# Define Decoder
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(32, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 784)
    def forward(self, z):
        z = F.relu(self.fc1(z))
        z = F.relu(self.fc2(z))
        x = torch.sigmoid(self.fc3(z))  # to make output's pixels are 0~1
        x = x.view(x.size(0), 1, 28, 28)
        return x
```

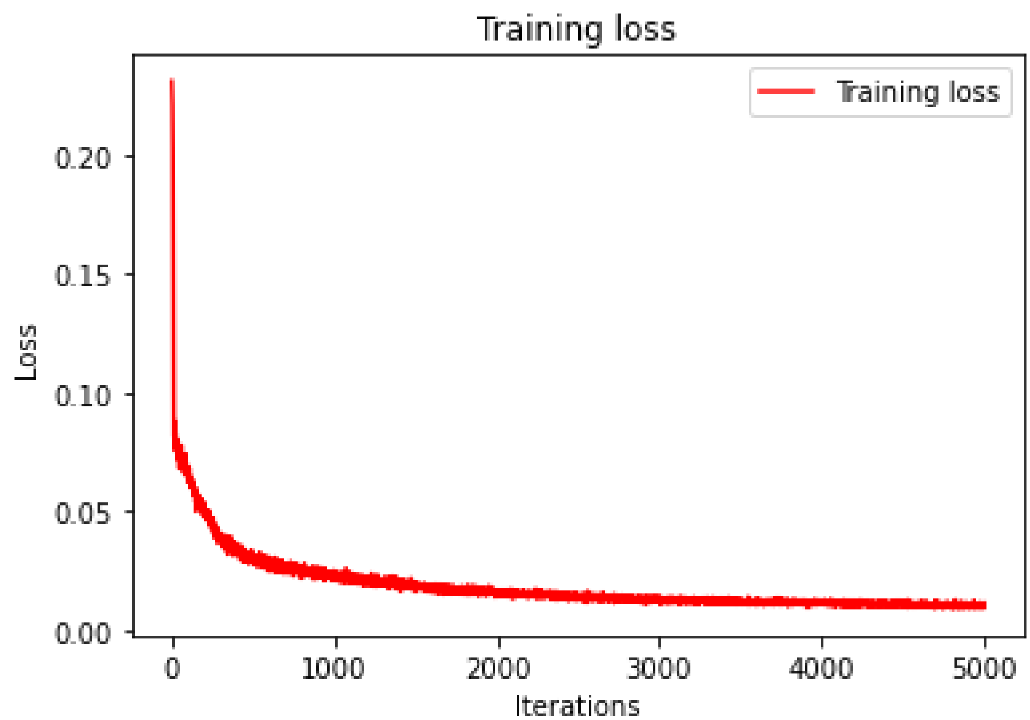
[3]:
```python
'''
Step 3: Instantiate model & define loss and optimizer
'''
enc = Encoder().to(device)
dec = Decoder().to(device)
loss_function = nn.MSELoss()
optimizer = optim.Adam(list(enc.parameters()) + list(dec.parameters()), lr=lr)
```

[4]:
```python
'''
Step 4: Training
'''
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,␣
 ↪batch_size=batch_size, shuffle=True)


train_loss_list = []

import time
start = time.time()
for epoch in range(epochs) :
    print("{}th epoch starting.".format(epoch))
    enc.train()
    dec.train()
    for batch, (images, _) in enumerate(train_loader) :
```

```
        images = images.to(device)
        z = enc(images)
        reconstructed_images = dec(z)

        optimizer.zero_grad()
        train_loss = loss_function(images, reconstructed_images)
        train_loss.backward()
        train_loss_list.append(train_loss.item())

        optimizer.step()

        print(f"[Epoch {epoch:3d}] Processing batch #{batch:3d} reconstruction␣
 ↪loss: {train_loss.item():.6f}", end='\r')
end = time.time()
print("Time ellapsed in training is: {}".format(end - start))

# plotting train loss
plt.plot(range(1,len(train_loss_list)+1), train_loss_list, 'r', label='Training␣
 ↪loss')
plt.title('Training loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.savefig('loss.png')

enc.eval()
dec.eval()
```

```
0th epoch starting.
1th epoch starting.ing batch #499 reconstruction loss: 0.029653
2th epoch starting.ing batch #499 reconstruction loss: 0.022749
3th epoch starting.ing batch #499 reconstruction loss: 0.019026
4th epoch starting.ing batch #499 reconstruction loss: 0.014858
5th epoch starting.ing batch #499 reconstruction loss: 0.014297
6th epoch starting.ing batch #499 reconstruction loss: 0.012640
7th epoch starting.ing batch #499 reconstruction loss: 0.012427
8th epoch starting.ing batch #499 reconstruction loss: 0.011287
9th epoch starting.ing batch #499 reconstruction loss: 0.010938
Time ellapsed in training is: 33.234450817108154 loss: 0.010582
```

[4]:
```
Decoder(
  (fc1): Linear(in_features=32, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=784, bias=True)
)
```



[5]:
```
'''
Step 5: Calculate standard deviation by using validation set
'''
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset,␣
 ↪batch_size=batch_size)

scores = []

for images, _ in validation_loader:
    images = images.to(device)
    z = enc(images)
    recon = dec(z)
    test = (images - recon)**2
    score = ((images - recon)**2).sum((1,2,3))
    scores.append(score.cpu().detach().numpy())
mean = np.mean(scores)
std = np.std(scores)

threshold = mean + 3 * std
print("threshold: ", threshold)
```

```
threshold:  22.62565851211548
```

[6]:
```python
'''
Step 6: Anomaly detection (mnist)
'''
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,␣
↪batch_size=batch_size)

anomaly, n = 0, 0

for images, _ in test_loader:
    images = images.to(device)
    n += images.shape[0]
    z = enc(images)
    recon = dec(z)
    score = ((images - recon)**2).sum((1,2,3))
    anomaly += torch.sum(score > threshold)
print(f'anomaly: {anomaly.item()}, type I error: {anomaly / n * 100:.2f}%')
```

```
anomaly: 108, type I error: 1.08%
```

[7]:
```python
'''
Step 7: Anomaly detection (kmnist)
'''
anomaly_loader = torch.utils.data.DataLoader(dataset=anomaly_dataset,␣
↪batch_size=batch_size)

anomaly, n = 0, 0

for images, _ in anomaly_loader:
    images = images.to(device)
    n += images.shape[0]
    z = enc(images)
    recon = dec(z)
    score = ((images - recon)**2).sum((1,2,3))
    anomaly += torch.sum(score > threshold)
print(f'anomaly: {anomaly.item()}, type II error: {100 - anomaly / n * 100:.
↪2f}%')
```

```
anomaly: 9756, type II error: 2.44%
```