#1

W_1을 W의 first layer, W_2를 second layer라고 할때, W_1과 W_2는 다음과 같이 주어진다.

$$W_{1,i,j} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{1\times3\times3}$$

$$X = \begin{bmatrix} X_{i-1,j-1} & X_{i-1,j} & X_{i-1,j+1} \\ X_{i,j-1} & X_{i,j} & X_{i,j+1} \\ X_{i+1,j-1} & X_{i+1,j} & X_{i+1,j+1} \end{bmatrix}$$

$$W_{2,i,j} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{1\times3\times3}$$

W_1과 X의 Convolution이 $-X_{i,j} + X_{i+1,j} = Y_{1,i,j}$

W_2와 X의 Convolution이 $-X_{i,j} + X_{i,j+1} = Y_{2,i,j}$

가 됨을 확인해볼 수 있다.

#2

filter $w \in \mathbb{R}^{c\times c\times k\times k}$. (Output의 layer가 c개이므로 c×k×k 짜리 filter가 c개 필요함을 알 수 있다.)

W의 i번째 layer, 즉 i번째 filter $W_i \in \mathbb{R}^{c\times k\times k}$를 생각하자.

$$W_{i,j}: \begin{cases} \begin{bmatrix} 1/k^2 & 1/k^2 & \\ 1/k^2 & 1/k^2 & \\ & & \ddots \end{bmatrix} & (\text{if } j=i) \\[2ex] \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \ddots \end{bmatrix} & (\text{else}) \end{cases}$$

W_i의 j번째 layer를 살피자. j가 i와 같은 $W_{i,j,:,:}$는 모든 원소가 $1/k^2$이어야 한다.

$j \neq i$인 경우, $W_{i,j,:,:}$의 모든 원소는 0이다.

이렇게 W를 구성하면 W는 AvgPool2d : X→Y 에 해당하는 filter이다.

단, 이 경우 Convolution의 stride는 k이다.

#3

$w \in \mathbb{R}^{3\times1\times1}$가 [0.299, 0.587, 0.114]면 그만이다.

$X_{:,i,j}$와 w의 Convolution을 생각해보면

Output $Y_{i,j}$는 $X_{1,i,j} \times W_{1,1,1} + X_{2,i,j} \times W_{2,1,1} + X_{3,i,j} \times W_{3,1,1}$

$= 0.299\, X_{1,i,j} + 0.587\, X_{2,i,j} + 0.114\, X_{3,i,j}$ 를 얻는다.

#4. $\sigma : \mathbb{R} \to \mathbb{R}$, non decreasing activation ftn.

$\rho : \mathbb{R}^{m \times n} \to \mathbb{R}^{k \times l}$: max pool operation.

Show for all $X \in \mathbb{R}^{m \times n}$, σ and ρ commute. X의 (i,j)entry 값을 X_{ij} 라 하자.

pf) $f_1 = \frac{m}{k}$, $f_2 = \frac{n}{l}$ 이라 하자. $\rho(X) \in \mathbb{R}^{k \times l}$ 의 (i,j) entry 를 $\rho_{ij}(X)$ 라 하자.

$$\rho_{i,j}(X) = \max Y_{i,j}, \quad Y_{i,j} := \{ X_{\alpha,\beta} \mid (i-1)f_1 < \alpha \leq if_1 ,\ (j-1)f_2 < \beta \leq jf_2 \}$$

$$\sigma(\rho_{i,j}(X)) = \sigma(\max(Y_{i,j})) = \sigma(\max(\{ X_{\alpha,\beta} \mid (i-1)f_1 < \alpha \leq if_1 ,\ (j-1)f_2 < \beta \leq jf_2 \}))$$

* Since σ is a non decreasing function,

$\max\{\sigma(x_1), \sigma(x_2) \cdots, \sigma(x_n)\} = \sigma(\max\{x_1, x_2 \cdots, x_n\})$ holds.

$$= \max(\{ \sigma(X_{\alpha,\beta}) \mid (i-1)f_1 < \alpha \leq if_1 ,\ (j-1)f_2 < \beta \leq jf_2 \})$$

$$= \rho_{i,j}(\sigma(X)) \ !$$

$\therefore \sigma(\rho_{i,j}(X)) = \rho_{i,j}(\sigma(X))$ for all (i,j)

and thus, $\sigma(\rho(X)) = \rho(\sigma(X))$. holds

Problem 5 : Non CE loss function

In [4]:

```
import torch
import torch.nn as nn
from torch.optim import Optimizer
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import transforms
import matplotlib.pyplot as plt
from random import shuffle
'''
Step 1: Data
'''
# Use data with only 4 and 9 as labels: which is hardest to classify
label_1, label_2 = 4, 9

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

# Use data with two labels
idx = (train_set.targets == label_1) + (train_set.targets == label_2)
train_set.data = train_set.data[idx]
train_set.targets = train_set.targets[idx]
train_set.targets[train_set.targets == label_1] = -1
train_set.targets[train_set.targets == label_2] = 1

# MNIST testing data
test_set = datasets.MNIST(root='./mnist_data/', train=False, transform=transforms.ToTensor())

# Use data with two labels
idx = (test_set.targets == label_1) + (test_set.targets == label_2)
test_set.data = test_set.data[idx]
test_set.targets = test_set.targets[idx]
test_set.targets[test_set.targets == label_1] = -1
test_set.targets[test_set.targets == label_2] = 1


'''
Step 2: (same step)
'''
class Linear(nn.Module) :

    def __init__(self, input_dim=28*28) :
        super().__init__()
        self.linear = nn.Linear(input_dim, 1, bias=False)

    def forward(self, x) :
        return self.linear(x.float().view(-1, 28*28))

'''
Step 3: Create the model, specify loss function and optimizer. (LOOK HERE)
'''
model_CE = Linear()
model_MSE = Linear()

def CE_loss(output, target):
    return torch.mean(-torch.nn.functional.logsigmoid(target.reshape(-1)*output.reshape(-1)))

def MSE_loss(output, target):
    output = output.reshape(-1)
    target = target.reshape(-1)
    B = len(output)
    total = 0
    for i in range(B):
        y = target[i]
        z = output[i]
        total += 0.5*(1-y)*((1-torch.sigmoid(-z))**2 + torch.sigmoid(z)**2)
        total += 0.5*(1+y)*(torch.sigmoid(-z)**2 + (1-torch.sigmoid(z))**2)
    return total/B


optimizer_CE = torch.optim.SGD(model_CE.parameters(), lr=255*1e-4)
optimizer_MSE = torch.optim.SGD(model_MSE.parameters(), lr=255*1e-4)


'''
Step 4: Train model with SGD (LOOK HERE)
'''
train_loader = DataLoader(dataset=train_set, batch_size=64, shuffle=True)

for epoch in range(3) :
    for images, labels in train_loader :
        optimizer_CE.zero_grad()
        optimizer_MSE.zero_grad()

        train_loss_CE = CE_loss(model_CE(images), labels.float())
        train_loss_CE.backward()

        train_loss_MSE = MSE_loss(model_MSE(images), labels.float())
        train_loss_MSE.backward()

        optimizer_CE.step()
        optimizer_MSE.step()

'''
Step 5: (same step)
'''
test_loss_CE, correct_CE = 0, 0
test_loss_MSE, correct_MSE = 0, 0

test_loader = DataLoader(dataset=test_set, batch_size=1, shuffle=False)

for ind, (image, label) in enumerate(test_loader) :

    output_CE = model_CE(image)
    output_MSE = model_MSE(image)
    test_loss_CE += CE_loss(output_CE, label.float()).item()
    test_loss_MSE += MSE_loss(output_MSE, label.float()).item()


    # Make a prediction
    if output_CE.item() * label.item() >= 0 :
        correct_CE += 1
    if output_MSE.item() * label.item() >= 0 :
        correct_MSE += 1

# Print out the results
print("Cross entropy loss")
print('[Test set] Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss_CE /len(test_loader), correct_CE, len(test_loader),
        100. * correct_CE / len(test_loader)))
print("Mean squared loss")
print('[Test set] Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss_MSE /len(test_loader), correct_MSE, len(test_loader),
        100. * correct_MSE / len(test_loader)))
```

```
Cross entropy loss
[Test set] Average loss: 0.1749, Accuracy: 1894/1991 (95.13%)

Mean squared loss
[Test set] Average loss: 0.1007, Accuracy: 1896/1991 (95.23%)
```

> 큰 차이가 없는 것으로 보여진다.

#6 (a) $y_L = A_L y_{L-1} + b_L$, $A_L \in \mathbb{R}^{1\times n_{L-1}}$, $b_L \in \mathbb{R}^1$, $y_L \in \mathbb{R}^1$

$$\Rightarrow \frac{\partial y_L}{\partial b_L} = 1 , \qquad \frac{\partial y_L}{\partial y_{L-1}} = \left(\frac{\partial y_L}{\partial (y_{L-1})_1}, \dots, \frac{\partial y_L}{\partial (y_{L-1})_{n_{L-1}}} \right)$$

(y_L, b_L is scalar)

$$\left(\frac{\partial y_L}{\partial y_{L-1}}\right)_i = \frac{\partial y_L}{\partial y_i} = \frac{\sum_{j=1}^{n_{L-1}} (A_L)_{1,j} \times (y_{L-1})_j}{\partial y_i} = (A_L)_{1,i}$$

$$\Rightarrow \frac{\partial y_L}{\partial y_{L-1}} = A_L.$$

$y_l = \sigma(A_l y_{l-1} + b_l)$ $(l=1,\dots,L-1)$ $A_l \in \mathbb{R}^{n_l \times n_{l-1}}$, $b_l \in \mathbb{R}^{n_l}$

$$\frac{\partial y_l}{\partial b_l} = \begin{pmatrix} \frac{\partial (y_l)_1}{\partial (b_l)_1} & \cdots & \frac{\partial (y_l)_1}{\partial (b_l)_{n_l}} \\ \vdots & & \\ \frac{\partial (y_l)_{n_l}}{\partial (b_l)_1} & \cdots & \frac{\partial (y_l)_{n_l}}{\partial (b_l)_{n_l}} \end{pmatrix} \qquad \left(\frac{\partial y_l}{\partial b_l}\right)_{i,j} = \frac{\partial (y_l)_i}{\partial (b_l)_j} = \frac{\partial \sigma\left((b_l)_i + \sum_{k=1}^{n_{l-1}} (A_l)_{i,k} (y_{l-1})_k\right)}{\partial (b_l)_j}$$

$$= \begin{cases} \sigma'\left((b_l)_i + \sum_{k=1}^{n_{l-1}} (A_l)_{i,k} (y_{l-1})_k\right) & (i=j) \\ 0 & (\text{else}) \end{cases}$$

$$\Rightarrow \frac{\partial y_l}{\partial b_l} = \begin{pmatrix} \sigma'((b_l)_1 + \sum_i (A_l)_{1,i} (y_{l-1})_i) & & 0 \\ & \ddots & \\ 0 & & \sigma'((b_l)_{n_l} + \sum_i (A_l)_{n_l,i} (y_l)_i) \end{pmatrix} = \operatorname{diag}\left(\sigma'(b_l + A_l y_{l-1})\right)$$

$$\left(\frac{\partial y_l}{\partial y_{l-1}}\right)_{i,j} = \frac{\partial (y_l)_i}{\partial (y_{l-1})_j} = \frac{\partial \left(\sigma\left((b_l)_i + \sum_{k=1}^{n_{l-1}} (A_l)_{i,k} (y_{l-1})_k\right)\right)}{\partial (y_{l-1})_j} = \sigma'\left((b_l)_i + \sum_{k=1}^{n_{l-1}} (A_l)_{i,k} (y_{l-1})_k\right) \times (A_l)_{i,j}$$

$$\Rightarrow \left(\frac{\partial y_l}{\partial y_{l-1}}\right) \in \mathbb{R}^{n_l \times n_{l-1}} = \begin{pmatrix} \sigma'((b_l)_1 + \sum_i (A_l)_{1,i} (y_{l-1})_i & 0 & \cdots & 0 \\ 0 & \ddots & & 0 \\ 0 & 0 & & \\ \vdots & & & \\ 0 & & & \sigma'((b_l)_{n_l} + \sum_i (A_l)_{n_l,i} (y_{l-1})_i) \end{pmatrix} (A_l)$$

$$= \operatorname{diag}\left(\sigma'(b_l + A_l y_{l-1})\right) A_l$$

(b) $\frac{\partial y_L}{\partial A_L} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ $\left(\frac{\partial y_L}{\partial A_\ell}\right)_{ij} = \frac{\partial y_L}{\partial (A_\ell)_{ij}}$

$$\left(\frac{\partial y_L}{\partial A_L}\right)_{1j} = \frac{\partial y_L}{\partial (A_L)_{1,j}} = \frac{\partial\left(b_L + \sum_{i=1}^{n_{L-1}} (A_L)_{1,i} (y_{L-1})_i\right)}{\partial (A_L)_{1,j}} = (y_{L-1})_j \Rightarrow \underline{\left(\frac{\partial y_L}{\partial A_L}\right) = y_{L-1}^T}$$

$$\left(\frac{\partial y_L}{\partial A_\ell}\right)_{i,j} = \frac{\partial y_L}{\partial (A_L)_{i,j}} = \frac{\partial y_L}{\partial y_\ell} \cdot \frac{\partial y_\ell}{\partial (A_\ell)_{ij}} = \left(\frac{\partial y_L}{\partial y_{L-1}} \cdot \frac{\partial y_{L-1}}{\partial y_{L-2}} \cdots \frac{\partial y_{\ell+1}}{\partial y_\ell}\right) \frac{\partial y_\ell}{\partial (A_\ell)_{ij}}$$

$$= \left(\frac{\partial y_L}{\partial y_\ell}\right) \cdot \begin{pmatrix} \frac{\partial\left(\sigma\left((b_\ell)_1 + \sum_{k=1}^{n_{\ell-1}} (A_\ell)_{1,k} (y_{\ell-1})_k\right)\right)}{\partial (A_\ell)_{i,j}} \\ \vdots \\ \frac{\partial\left(\sigma\left((b_\ell)_{n_\ell} + \sum_{1}^{n_{\ell-1}} (A_\ell)_{n_\ell,k} (y_{\ell-1})_k\right)\right)}{\partial (A_\ell)_{i,j}} \end{pmatrix}$$

$$= \left(\frac{\partial y_L}{\partial y_\ell}\right) \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \sigma'\left((b_\ell)_i + \sum_{1}^{n_{\ell-1}} (A_\ell)_{i,k} (y_{\ell-1})_k\right)(y_{\ell-1})_j \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \left(\frac{\partial y_L}{\partial y_\ell}\right)_i (y_{\ell-1})_j \times \sigma'\left((b_\ell)_i + \sum_{k=1}^{n_{\ell-1}} (A_\ell)_{i,k} (y_{\ell-1})_k\right)$$

$$\therefore \underline{\left(\frac{\partial y_L}{\partial A_\ell}\right)} = \operatorname{diag}\left(\sigma'(b_\ell + A_\ell y_{\ell-1})\right) \begin{pmatrix} \left(\frac{\partial y_L}{\partial y_\ell}\right)_1 (y_{\ell-1})_1, & \left(\frac{\partial y_L}{\partial y_\ell}\right)_1 (y_{\ell-1})_2 & \cdots & \left(\frac{\partial y_L}{\partial y_\ell}\right)_1 (y_{\ell-1})_{n_{\ell-1}} \\ \vdots & & & \\ \left(\frac{\partial y_L}{\partial y_\ell}\right)_{n_\ell} (y_{\ell-1})_1 & \cdots & \cdots & , \left(\frac{\partial y_L}{\partial y_\ell}\right)_{n_\ell} (y_{\ell-1})_{n_{\ell-1}} \end{pmatrix}$$

$$= \underline{\operatorname{diag}\left(\sigma'(b_\ell + A_\ell y_{\ell-1})\right) \left(\frac{\partial y_L}{\partial y_\ell}\right)^T y_{\ell-1}^T}$$

#Problem 7

먼저 trainable parameter의 수에 대해 계산을 하고 시작하자.

C3_layer_full, 즉 일반적인 convolutional net을 사용한다면,

$$6 \times (1 \times 5 \times 5 + 1) + 16 \times (6 \times 5 \times 5 + 1) + 120 \times (1 + 5 \times 5 \times 16) + 84 \times (1 + 120) + 10 \times (1 + 84) = 61706$$

(실제로 돌려보고 print로 확인해도 같은 결과가 나온다..)

Original LeNet의 C3 layer를 사용한다면,

$$6 \times (1 \times 5 \times 5 + 1) +$$
$$6 \times (3 \times 5 \times 5 + 1) + 9 \times (4 \times 5 \times 5 + 1) + (6 \times 5 \times 5 + 1) +$$
$$120 \times (1 + 5 \times 5 \times 16) + 84 \times (1 + 120) + 10 \times (1 + 84) = 60806$$

개의 parameter가 있을 것이다. 총 900개의 parameter reduction이 발생한다.

코드는 다음과 같이 짜주면 된다.

lenet_original.py

```python
import torch
import torch.nn as nn
from torch.optim import Optimizer
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import transforms

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

'''
Step 1:
'''

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

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


'''
Step 2: LeNet5
'''

# Modern LeNet uses this layer for C3
class C3_layer_full(nn.Module):
    def __init__(self):
        super(C3_layer_full, self).__init__()
        self.conv_layer = nn.Conv2d(6, 16, kernel_size=5)

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

```python
# Original LeNet uses this layer for C3
class C3_layer(nn.Module):
    def __init__(self):
        super(C3_layer, self).__init__()
        self.ch_in_3 = [[0, 1, 2],
                        [1, 2, 3],
                        [2, 3, 4],
                        [3, 4, 5],
                        [0, 4, 5],
                        [0, 1, 5]] # filter with 3 subset of input channels
        self.ch_in_4 = [[0, 1, 2, 3],
                        [1, 2, 3, 4],
                        [2, 3, 4, 5],
                        [0, 3, 4, 5],
                        [0, 1, 4, 5],
                        [0, 1, 2, 5],
                        [0, 1, 3, 4],
                        [1, 2, 4, 5],
                        [0, 2, 3, 5]] # filter with 4 subset of input channels
        # put implementation here
        self.conv3layers = nn.ModuleList([nn.Conv2d(3,1,kernel_size = 5) for i in range(6)])
        self.conv4layers = nn.ModuleList([nn.Conv2d(4,1,kernel_size = 5) for i in range(9)])
        self.conv6layer = nn.Conv2d(6,1,kernel_size = 5)



    def forward(self, x):
        L = []
        for i,layer in enumerate(self.conv3layers):
            L.append(layer(x[:,self.ch_in_3[i],:,:]) )
        for i,layer in enumerate(self.conv4layers):
            L.append(layer(x[:,self.ch_in_4[i],:,:]) )
        L.append(self.conv6layer(x))
        return torch.cat(L, dim=1) #batch

```

```python
class LeNet(nn.Module) :
    def __init__(self) :
        super(LeNet, self).__init__()
        #padding=2 makes 28x28 image into 32x32
        self.C1_layer = nn.Sequential(
                nn.Conv2d(1, 6, kernel_size=5, padding=2),
                nn.Tanh()
                )
        self.P2_layer = nn.Sequential(
                nn.AvgPool2d(kernel_size=2, stride=2),
                nn.Tanh()
                )
        self.C3_layer = nn.Sequential(
                #C3_layer_full(),
                C3_layer(),
                nn.Tanh()
                )
        self.P4_layer = nn.Sequential(
                nn.AvgPool2d(kernel_size=2, stride=2),
                nn.Tanh()
                )
        self.C5_layer = nn.Sequential(
                nn.Linear(5*5*16, 120),
                nn.Tanh()
                )
        self.F6_layer = nn.Sequential(
                nn.Linear(120, 84),
                nn.Tanh()
                )
        self.F7_layer = nn.Linear(84, 10)
        self.tanh = nn.Tanh()

    def forward(self, x) :
        output = self.C1_layer(x)
        output = self.P2_layer(output)
        output = self.C3_layer(output)
        output = self.P4_layer(output)
        output = output.view(-1,5*5*16)
        output = self.C5_layer(output)
        output = self.F6_layer(output)
        output = self.F7_layer(output)
        return output


'''
Step 3
'''
model = LeNet().to(device)
loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-1)

# print total number of trainable parameters
param_ct = sum([p.numel() for p in model.parameters()])
print(f"Total number of trainable parameters: {param_ct}")

'''
```

```python
'''
Step 4
'''
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100, shuffle=True)

import time
start = time.time()
for epoch in range(10) :
    print("{}th epoch starting.".format(epoch))
    for images, labels in train_loader :
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        train_loss = loss_function(model(images), labels)
        train_loss.backward()

        optimizer.step()
end = time.time()
print("Time ellapsed in training is: {}".format(end - start))


'''
Step 5
'''
test_loss, correct, total = 0, 0, 0

test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=100, shuffle=False)

for images, labels in test_loader :
    images, labels = images.to(device), labels.to(device)

    output = model(images)
    test_loss += loss_function(output, labels).item()

    pred = output.max(1, keepdim=True)[1]
    correct += pred.eq(labels.view_as(pred)).sum().item()

    total += labels.size(0)

print('[Test set] Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss /total, correct, total,
        100. * correct / total))
```

```
In [3]: runfile('C:/Users/sylee/OneDrive/바탕 화면/심신개/HW4/
lenet_original.py', wdir='C:/Users/sylee/OneDrive/바탕 화면/심신개/HW4')
Total number of trainable parameters: 60806
0th epoch starting.
1th epoch starting.
2th epoch starting.
3th epoch starting.
4th epoch starting.
5th epoch starting.
6th epoch starting.
7th epoch starting.
8th epoch starting.
9th epoch starting.
Time ellapsed in training is: 189.81498003005981
[Test set] Average loss: 0.0004, Accuracy: 9867/10000 (98.67%)
```

위와 같이 잘 train 된 것을 볼 수 있다. 계산한 파라미터 개수도 맞는다.