MLP Assignment

세션날짜: 2023년 1월 19일 과제제출 기한: 2023년 1월 25일 23:59:59

1. Make MNIST Generalized Model

[목적]

MNIST 데이터를 분류할 수 있는 Generalized 된 MLP 모델을 생성한다.

[조건]

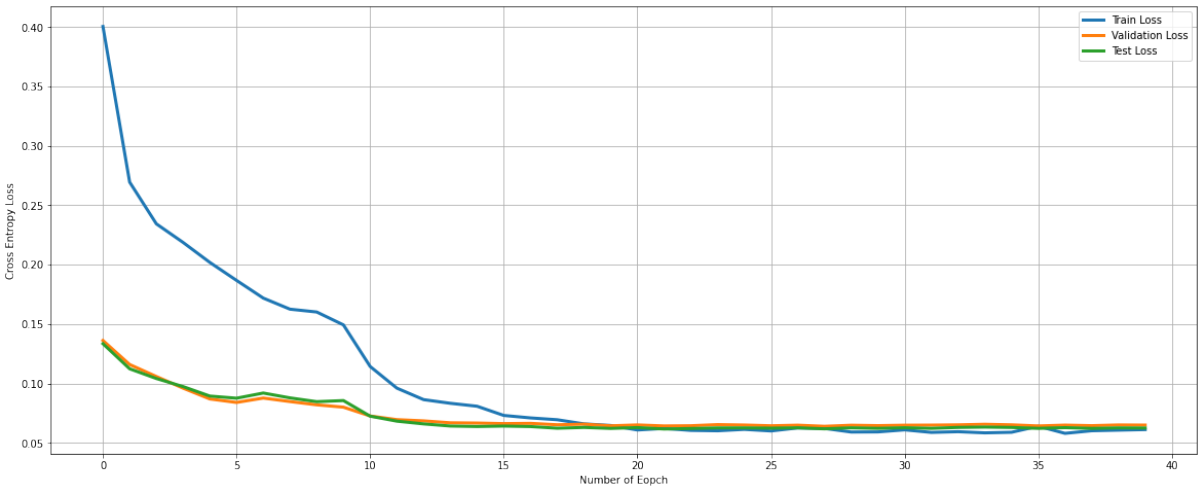
Hidden Layer

	Input Dimension	Output dimension
1st hidden layer	784	512
Activation function	ReLU	
2nd hidden layer	512	256
Activation function	ReLU	
Output layer	256	10

Loss Function : CrossEntropyLoss

Train Loss, Validation Loss, Test Loss가 모두 아래로 하향곡선을 그리도록 모델을 만들어 주세요. Test Loss의 크기가 0.1 이하로 나오도록 해주시길 바랍니다.

(결과예시)



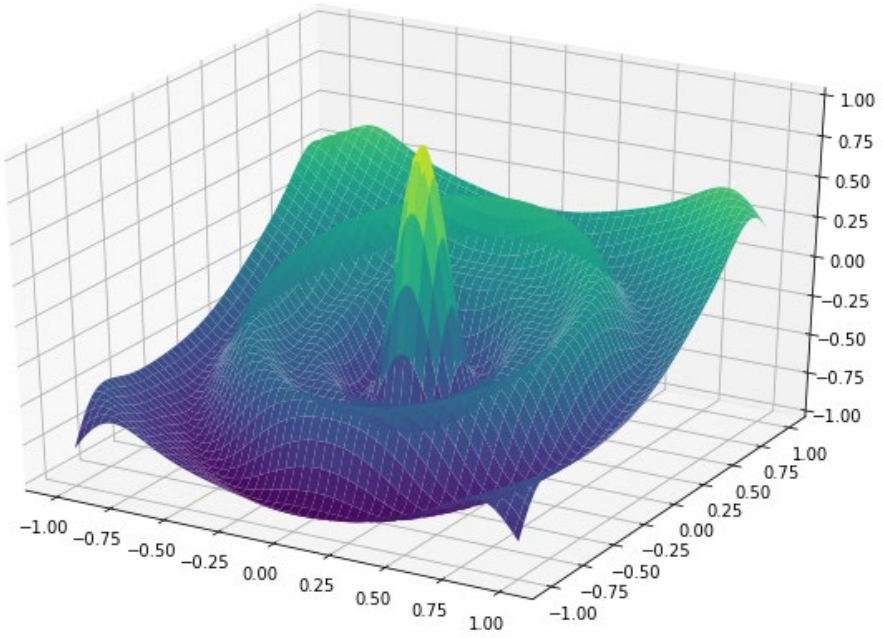

```
Epoch 40 / 40
        [Train loss : 0.06077] [Validation loss : 0.06515] [Test loss : 0.06266], [Test Accuracy : 98.31%],[curr LR = [1.25e-06]],[elapsed_time = 755sec]
```

2. Universal Approximation Theorem

[목적]

아래의 함수를 근사하는 MLP 모델을 생성한다.

$$f(x,y) = \frac{\sin(20\sqrt{x^2+y^2})}{20\sqrt{x^2+y^2}} + \frac{1}{5}\cos(10\sqrt{x^2+y^2}) + \frac{y}{2} - 0.3$$



[조건]

Hidden layer

- Hidden layer의 뉴런 수는 최소 56개 이상이다.
- 한 개의 hidden layer으로도 구현가능하며 hidden layer가 많아질수록 근사한다.

	Input Dimension	Output dimension
1^{st} hidden layer	2	?
Activation function	Tanh	
⋮		
n^{th}hidden layer	?	1

Loss Function : MSELoss

Batch_size : 데이터 크기보다 크게, Batch Gradient Descent

Num_epoch : 최소 6000 이상

Optimizer, Scheduler, Learning rate 등은 자유롭게 진행

Step 1) 함수를 생성한다.

Step 2) $-1 \leq x \leq 1$ 그리고 $-1 \leq y \leq 1$를 각각 100등분하여 data point를 100,000개 생성한다. data point를 함수에 넣어 sample data (x,y) 와 target data (f(x,y))를 생성한다.

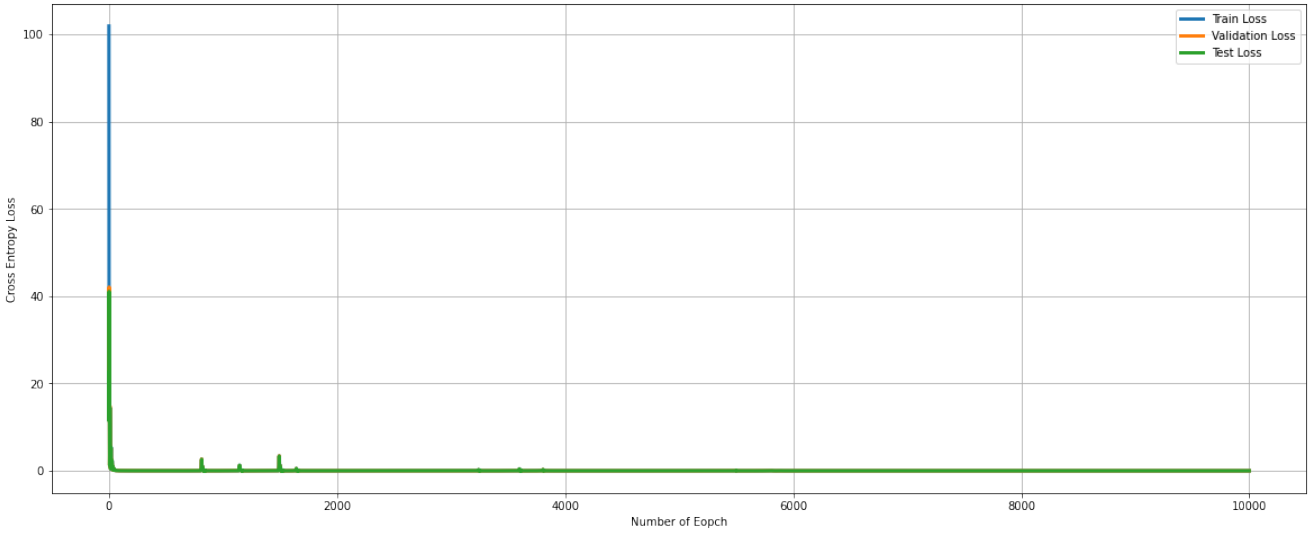
Step 3) Train data, Validation data, Test data로 0.75:0.15:0.15의 비율로 나눈다.

Step 4) DataLoader를 이용하여 데이터를 Shuffle하고 BGD로 활용할 수 있는 학습데이터를 생성한다.

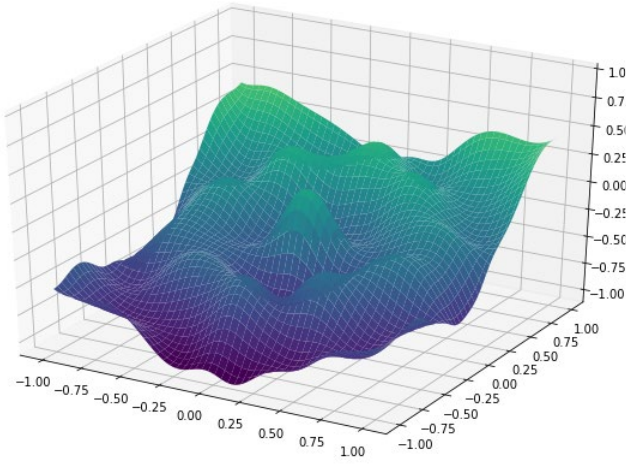
Step 5) 최소 1개의 Hidden layer를 가진 MLP를 구현한다. Input dimension은 2(x와 y), 최종 output dimension은 1 (예측된 f(x,y))이다. Hidden layer의 뉴런 수는 최소 56개 이상이다.

Step 6) Loss는 MSE이며 epoch는 6,000회 이상으로 한다.

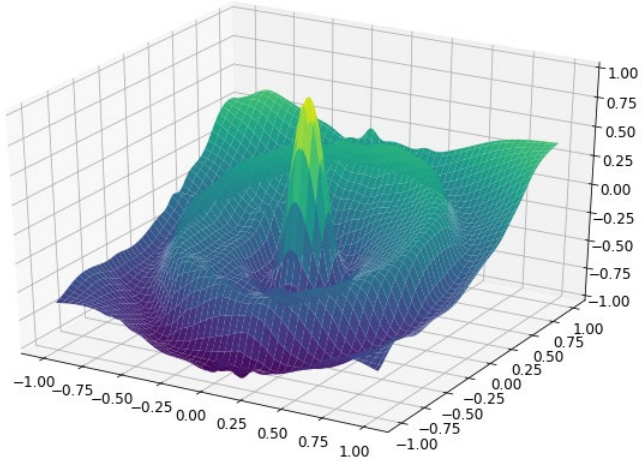
(결과예시)




```
Epoch 10000  /  10000
        [Train loss : 0.00106]  [Validation loss : 0.00134] [Test loss : 0.00137]  [curr LR = [0.00625]],[elapsed_time = 772sec]
```



Hidden Layer 1개 일 때



Hidden Layer 2개 일 때

APPENDIX

Import Packages

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from mpl_toolkits.mplot3d import Axes3D

import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
```

Setup a device

```
# Device configuration
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

print('device:', device)
```

Function

```
'''
 # function_2d
 # Input : x , y
 # output : z = f(x,y)
'''
def function_2d(x,y):
  term_sqrt = np.sqrt(x*x+y*y)
  term1 = np.sin(20*term_sqrt)/(20*term_sqrt)
  term2 = (1/5)*np.cos(10*term_sqrt)
  term3 = y/2 - 0.3

  label = term1 + term2 + term3

  return label


'''
 # generate_data
 # input : dim (int)
 # output : (10000, 3) dataset, type : np.array
    - first column : x
    - second column : y
    - third column : f(x,y) (True label)
'''
def generate_data(dim):
  x = np.linspace(-1, 1, dim)
```

```
  y = np.linspace(-1, 1, dim)
  xx, yy = np.meshgrid(x,y)
  zz = function_2d(xx,yy)
  zz = zz

  data_1 = xx.reshape(-1,1)
  data_2 = yy.reshape(-1,1)
  label = zz.reshape(-1,1)

  dataset = np.hstack((data_1, data_2))
  dataset = np.hstack((dataset, label))

  return dataset
```

Function Plot

```
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(-1, 1, 100)
y = np.linspace(-1, 1, 100)
xx, yy = np.meshgrid(x,y)
zz = function_2d(xx,yy)

surf = ax.plot_surface(xx, yy, zz, cmap='viridis')
surf.set_clim(-1.0, 1.0)
ax.view_init(30,-60)
ax.set_zticks([-1,1])
ax.zaxis.set_major_locator(ticker.MultipleLocator(0.25))
plt.tight_layout()
plt.show()
```

Load Data

```
class Mydataset(torch.utils.data.Dataset):

    def __init__(self, dataX_np , dataY_np):

        self.data_X = dataX_np.astype(np.float32)
        self.data_Y = dataY_np.astype(np.float32)

        print(f"My_dataset __init__ received : {self.data_X.shape}, {self.data_Y.sh
ape} ")

    def __getitem__(self, index):
        batch = self.data_X[index]
        target = self.data_Y[index]

        return batch, target

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

```
dataset = generate_data(100)
```

Train, Valid, Test data split

```
x = dataset[:, :2]
y = dataset[:,2:]

# train-test 분리
x_train, x_temp, y_train, y_temp = train_test_split(x,y,test_size=0.3, shuffle=True
)

# train-validation 분리
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5,shuff
le=True)

print(x_train.shape)
print(x_val.shape)
print(x_test.shape)
print(y_test.shape)
```

Model architecture

```
class SimpleMLP(nn.Module):

    def __init__(self, inp_ = 2,  output_ = 1):
        ?

    def forward(self, x):
        ?

        return ?
```

Setting DataLoader

```
batch_size = 1000000
train_loader = torch.utils.data.DataLoader(Mydataset(x_train, y_train), batch_size=
batch_size, shuffle=False, drop_last = False)
valid_loader = torch.utils.data.DataLoader(Mydataset(x_val, y_val), batch_size=batc
h_size, shuffle=False, drop_last = False)
test_loader = torch.utils.data.DataLoader(Mydataset(x_test, y_test), batch_size=bat
ch_size, shuffle=False, drop_last = False)
```

Training

```
from statistics import mean
import time
```

```
def train(model, criterion_, optimizer_, scheduler_, num_epochs=40, first_epoch=1):

    train_losses = []
    valid_losses = []
    test_losses = []

    print("--------------------------------------------------------------------------------
----")

    start_time = time.time()
    for epoch in range(first_epoch, first_epoch + num_epochs):
        # train phase
        model.train()

        # batch_loss
        batch_losses = []

        for samples, labels in train_loader:

            # Move the training data to the GPU
            samples = samples.to(device)
            labels = labels.to(device)

            # clear previous gradient computation
            optimizer_.zero_grad()

            # forward propagation
            outputs = model(samples)

            # calculate the loss
            loss = criterion_.forward(outputs, labels)
            batch_losses.append(loss.item())

            # backpropagate to compute gradients
            loss.backward()

            # update model weights
            optimizer_.step()

        # Train loss를 저장합니다.
        train_losses.append(mean(batch_losses))

        # validation phase
        # Dropout, BatchNormalization과 같은 layer가 동작하지 않도록 합니다.
        model.eval()

        # We don't need gradients for test, so wrap in
        # no_grad to save memory
        # 기울기 Gradient를 계산하지 않으며 backpropagation을 하지 않습니다.
        with torch.no_grad():

            correct_test = 0

            for samples, labels in valid_loader:
```

```
                # Move the training batch to the GPU
                samples = samples.to(device)
                labels = labels.to(device)

                # forward propagation
                outputs = model(samples)

                # calculate the loss
                loss = criterion_(outputs, labels)
                valid_losses.append(loss.item())

            for samples, labels in test_loader:
                # Move the training batch to the GPU
                samples = samples.to(device)
                labels = labels.to(device)

                # forward propagation
                outputs = model(samples)

                # calculate the loss
                loss = criterion_(outputs, labels)
                test_losses.append(loss.item())


        # 원하는 Epoch 마다 결과 출력
        if (epoch) % 1000== 0 :
            curr_time = round(time.time()-start_time)
            train_rec = round(train_losses[-1],5)
            valid_rec = round(valid_losses[-1],5)
            test_rec = round(test_losses[-1],5)
            print('Epoch', epoch, ' / ',num_epochs)
            print(f"\t [Train loss : {train_rec}]  [Validation loss : {valid_rec}]
[Test loss : {test_rec}]  [curr LR = {scheduler_.get_last_lr()}],[elapsed_time = {c
urr_time}sec] ")

        # Learning rate Scheduling
        # Scheduler 에 따라 learning rate 를 조절합니다.
        scheduler_.step()


    print(f"\nTrain Ended, total_elapsed_time = {round(time.time()-start_time)} ")
    print("----------------------------------------------------------------------")

    # Loss 값 반환
    return train_losses, valid_losses, test_losses
```

Model Train

```
model = SimpleMLP()
model.to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2000, gamma=0.5)
train_losses, valid_losses, test_losses = train(model, criterion_ = criterion, opti
mizer_ = optimizer, scheduler_ = scheduler, num_epochs=10000)
```

Plot Loss

```
def plot_result(train_loss, val_loss, test_loss):
  plt.figure(figsize=(20,8))
  plt.plot(train_loss, label='Train Loss', linewidth='3')
  plt.plot(val_loss, label='Validation Loss', linewidth='3')
  plt.plot(test_loss, label='Test Loss', linewidth='3')
  plt.grid()
  plt.rc('xtick', labelsize=12)
  plt.rc('ytick', labelsize=12)
  plt.legend()
  plt.rc('legend', fontsize=15)
  plt.xlabel('Number of Eopch')
  plt.ylabel('Cross Entropy Loss')
  plt.rc('axes', labelsize=17)
```

Plot

```
plot_result(train_losses, valid_losses, test_losses)
```

3D Visualize Graph

```
def visual_graph(model):
  x = np.linspace(-1, 1, 100)
  y = np.linspace(-1, 1, 100)
  xx,yy = np.meshgrid(x,y)
  xx = xx.reshape(-1,1)
  yy = yy.reshape(-1,1)
  inp_np = np.hstack((xx,yy))
  inp_tensor = torch.Tensor(inp_np)
  inp_tensor = inp_tensor.to(device)
  zz = model(inp_tensor)
  pred_output = zz.detach().cpu().numpy()
  fig = plt.figure(figsize=(9, 6))
  ax = fig.add_subplot(111, projection='3d')
  xx = xx.reshape(100,100)
  yy = yy.reshape(100,100)
  pred_output = pred_output.reshape(100,100)
  surf = ax.plot_surface(xx, yy, pred_output, cmap='viridis')
```

```
  surf.set_clim(-1.0, 1.0)
  ax.view_init(30,-60)
  ax.set_zticks([-1,1])
  ax.zaxis.set_major_locator(ticker.MultipleLocator(0.25))
  plt.tight_layout()
  plt.show()
```

Plot 3D Graph

```
visual_graph(model)
```