컴퓨터비전_메인프로젝트

문제#1. 교재 2장 이진판단에서 사용한 펄서 여부를 판정하는 문제를 colab 상에서 PyTorch 기반으로 해결하시오

[26]
```
import numpy as np
import csv
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

필요 모듈들 import

[27]
```
from torch._C import dtype
from numpy.lib.function_base import asarray_chkfinite
def load_pulsar_dataset():
  with open('/content/pulsar_stars.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader, None)
    rows = []
    for row in csvreader:
      rows.append(row)

  global data, input_cnt, output_cnt
  input_cnt, output_cnt = 8, 1
  data = np.asarray(rows, dtype='float32')
  #data = np.zeros([len(rows), input_cnt+output_cnt],dtype='float32')
```

[전처리 과정]

리스트 변수 rows에 저장된 내용을 배열 변수 data에 복사

입출력 벡터의 크기는 8과 1

np.asarray( ) 함수를 호출해 리스트 구조를 배열 구조로 변환

[27]
```
def arrange_data(mb_size):
  global data, shuffle_map, test_begin_idx
  shuffle_map = np.arange(data.shape[0])
  np.random.shuffle(shuffle_map)
  step_count = int(data.shape[0] * 0.8) // mb_size
  test_begin_idx = step_count * mb_size
  return step_count

def get_test_data():
  global data, shuffle_map, test_begin_idx, output_cnt
  test_data = data[shuffle_map[test_begin_idx:]]
  return test_data[:, :-output_cnt], test_data[:, -output_cnt:]

def get_train_data(mb_size, nth):
  global data, shuffle_map, test_begin_idx, output_cnt
  if nth == 0:
    np.random.shuffle(shuffle_map[:test_begin_idx])
  train_data = data[shuffle_map[mb_size*nth:mb_size*(nth+1)]]
  return train_data[:, :-output_cnt], train_data[:, -output_cnt:]

def eval_accuracy(output, y):
    estimate = np.greater(output, 0.5)
    answer = np.greater(y, 0.5)
    correct = np.equal(estimate, answer)

    return np.mean(correct)
```

데이터 수만큼의 일련번호를 발생시킨 후 무작위로 순서를 섞음. shuffle_map 리스크는 전역 변수로 전달되어 get_train_data( ) 함수와 get_test_data( ) 함수에서 이용됨. 학습용 데이터와 평가용 데이터의 경계를 test_begin_idx 에 저장한 후 한 에포크 학습에 필요한 미니배치 처리 스텝 수를 반환. 평가 데이터를 일괄 공급하는 get_test_data( ) 함수는 arrange_data( ) 함수가 test_begin_idx에 구해 놓은 위치를 경계 삼아 인덱스 배열 shuffle_map위 후반부가 가리키는 위치의 data 행들 을 평가용 데이터로서 반환. 학습 데이터를 공급하는 get_train_data( ) 함수는 get_test_data( ) 함수와 비슷한 처리. 미니배치 구간의 위치를 따져 그 구간에 해당하는 shuffle_map이 가리키는 데이터들만 반환

eval_accuracy()는 정확도를 계산. 이진 판단 문제에서 정확도는 신경망이 추정한 로짓값에 따른 판단과 정답으로 주어진 판단이 일치하는 비율임. 이때 output에 담긴 로짓값들의 부호를 조사하면 신경망의 판단을 알 수 있음. 따라서 여기서는 로짓값을 확률로 변환하는 시그모이드 함수를 부를 필요가 없음. 정답으로 주어진 판단은 참일 때 1, 거짓일 때 0 값을 갖지만 안전하게 0.5와 비교해 구함. 두 판단의 일치 여부는 불리언 값의 배열로 구해지지만 평균 연산을 수행하면 파이썬 내부에서 참은 1, 거짓은 0으로 간주해 계산하기 때문에 결국 올바른 판단의 비율이 구해지게 됨.

[28]
```
def relu(x):
    return np.maximum(x, 0)

def sigmoid(x):
    return np.exp(-relu(-x)) / (1.0 + np.exp(-np.abs(x)))

def sigmoid_derv(x, y):
    return y * (1 - y)

def sigmoid_cross_entropy_with_logits(z, x):
    return relu(x) - x * z + np.log(1 + np.exp(-np.abs(x)))

def sigmoid_cross_entropy_with_logits_derv(z, x):
    return -z + sigmoid(x)
```

relu( ) 는 x 배열의 원소 각각에 대해 해당 원소 값과 0 가운데 작지 않은 쪽을 골라주는데 이는 한 마디로 음수인 원소들을 모두 찾아내 0으로 대치하는 효과가 있음. 자칫 np.maximum(x, 0) 의 함수 대신 이름이 비슷한 np.max(x,0) 함수를 호출하면 배열 x의 원소들을 0과 비교하는데 대신 배열 x 전체와 숫자 0을 비교하는 바람에 비교 대상의 형태가 불일치하게 되니 주의해야함. 그런데 sigmoid_derv( ) 함수는 이 예제 어느 곳에서도 호출되지 않고 있음. sigmoid( ) 함수도 sigmoid_cross_entropy_with_logits_derv( ) 함수에서만 내부적으로 호출되고 있음. 이는 학습이 시그모이드 함수 자체보다는 손실 함수에 해당하는 교차 엔트로피값을 중심으로 이루어지기 때문.

[29]
```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(8, 1)
    self.sigm = nn.Sigmoid()

  def forward(self, x):
    x = self.fc1(x)
    x = self.sigm(x)
    return x
```

Net 정의
시그모이드 함수를 사용하여 이진출력

[30]
```
# hyperparameters
LEARNING_RATE = 0.01
epoch_count = 10
mb_size = 15

# dataset load
load_pulsar_dataset()
step_count = arrange_data(mb_size)
test_x, test_y = get_test_data()
```

하이퍼파라미터를 설정해줌 (learning rate, epoch 수, 미니배치 크기)

Dataset load

[31]
```
# gpu check
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
print("device:", device)

# model set (gpu)
model = Net().to(device)
print(model)

# SGD optimizer
optimizer = optim.SGD(model.parameters(), lr = LEARNING_RATE)
#optimizer = optim.Adam(model.parameters())

loss_fn = nn.BCELoss()

for epoch in range(epoch_count):
  losses, accs = [], []
```

Epoch 수에 맞게 학습 진행

```
[31]   # set train mode
       model.train()

       for n in range(step_count):
         # optimizer init
         optimizer.zero_grad()

         # get train data
         train_x, train_y = get_train_data(mb_size, n)

         # dataset to torch
         x = torch.from_numpy(train_x).float()
         y = torch.from_numpy(train_y).float()
         x, y = x.to(device), y.to(device)

         # forward
         y_pred = model(x)
         acc = eval_accuracy(y_pred.to('cpu').detach().numpy(), train_y)
         accs.append(acc)
```

train dataset load 및 forward 해 줌

```
         # loss
         loss = loss_fn(y_pred, y)

         # backprop
         loss.backward()
         losses.append(loss.item())

         # weight, bias 업데이트
         optimizer.step()

       # run test & eval
       model.eval()
       x = torch.from_numpy(test_x).float()
       x = x.to(device)
       y_pred = model(x)
       acc = eval_accuracy(y_pred.to('cpu').detach().numpy(), test_y)

       print('Epoch {}: loss={:5.3f}, accuracy={:5.3f}/{:5.3f}'. ₩
             format(epoch+1, np.mean(losses), np.mean(accs), acc))
```

loss 연산 후 backpropagation을 통한 가중치, 편향 업데이트

1 epoch 후 test와 eval 진행

```
device: cpu
Net(
  (fc1): Linear(in_features=8, out_features=1, bias=True)
  (sigm): Sigmoid()
)
Epoch 1: loss=1.059, accuracy=0.951/0.632
Epoch 2: loss=0.815, accuracy=0.962/0.974
Epoch 3: loss=1.138, accuracy=0.960/0.970
Epoch 4: loss=0.958, accuracy=0.962/0.965
Epoch 5: loss=0.935, accuracy=0.964/0.971
Epoch 6: loss=0.887, accuracy=0.965/0.967
Epoch 7: loss=1.002, accuracy=0.962/0.975
Epoch 8: loss=1.134, accuracy=0.964/0.970
Epoch 9: loss=0.798, accuracy=0.965/0.975
Epoch 10: loss=1.066, accuracy=0.964/0.975
```

63.2%의 정확도가 추가 학습을 진행해 97.5%로 향상

문제#2. titanic.csv 파일은 타이타닉 침몰 당시 어떤 사람이 살아남을지 예측

```
import numpy as np
import csv
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

필요 모듈들 import

```
def load_titanic_dataset():
  with open('/content/drive/MyDrive/titanic.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader, None)
    rows = []
    for row in csvreader:
      rows.append(row)

  global data, input_cnt, output_cnt
  input_cnt, output_cnt = 8, 1
  data = np.zeros([len(rows), input_cnt+output_cnt])

  for n,row in enumerate(rows):
    if row[2] == 'male':
      data[n, 2] = 1
    if row[2] == 'female':
      data[n, 3] = 1

  for n,row in enumerate(rows):
    if row[5] == 'S':
      data[n, 6] = 1
    if row[5] == 'C':
      data[n, 7] = 1
    if row[5] == 'Q':
      data[n, 8] = 1

  data.astype('float')
```

[전처리 과정]

리스트 변수 rows에 저장된 내용을 배열 변수 data에 복사

입출력 벡터의 크기는 8(티켓 클래스: 1, 2, 3 / 성별: 남성, 여성 / 티켓 요금 / 승선 위치: C, Q, S)과 1(생존여부, 0:사망/1:생존 출력)

성별과 승선 위치를 원-핫 벡터 표현으로 변환하는 처리를 함

```
def arrange_data(mb_size):
  global data, shuffle_map, test_begin_idx
  shuffle_map = np.arange(data.shape[0])
  np.random.shuffle(shuffle_map)
  step_count = int(data.shape[0] * 0.8) // mb_size
  test_begin_idx = step_count * mb_size
  return step_count

def get_test_data():
  global data, shuffle_map, test_begin_idx, output_cnt
  test_data = data[shuffle_map[test_begin_idx:]]
  return test_data[:, :-output_cnt], test_data[:, -output_cnt:]

def get_train_data(mb_size, nth):
  global data, shuffle_map, test_begin_idx, output_cnt
  if nth == 0:
    np.random.shuffle(shuffle_map[:test_begin_idx])
  train_data = data[shuffle_map[mb_size*nth:mb_size*(nth+1)]]
  return train_data[:, :-output_cnt], train_data[:, -output_cnt:]

def eval_accuracy(output, y):
    estimate = np.greater(output,0.5)
    answer = np.greater(y, 0.5)
    correct = np.equal(estimate, answer)

    return np.mean(correct)
```

(위와 동일)

```
def relu(x):
    return np.maximum(x, 0)

def sigmoid(x):
    return np.exp(-relu(-x)) / (1.0 + np.exp(-np.abs(x)))

def sigmoid_derv(x, y):
    return y * (1 - y)

def sigmoid_cross_entropy_with_logits(z, x):
    return relu(x) - x * z + np.log(1 + np.exp(-np.abs(x)))

def sigmoid_cross_entropy_with_logits_derv(z, x):
    return -z + sigmoid(x)
```

(위와 동일)

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(8, 4)
    self.fc2 = nn.Linear(4,2)
    self.fc3=nn.Linear(2,1)

  def forward(self, x):
    x = self.fc1(x)
    x=F.relu(x)
    x= self.fc2(x)
    x=F.relu(x)
    x=self.fc3(x)

    return x
```

Net 정의

ReLU함수를 사용하여

```
# hyperparameters
LEARNING_RATE = 0.01
epoch_count = 10
mb_size = 15

# dataset load
load_titanic_dataset()
step_count = arrange_data(mb_size)
test_x, test_y = get_test_data()
```

(위와 동일)

```
# gpu check
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
print("device:", device)

# model set (gpu)
model = Net().to(device)
print(model)

# SGD optimizer
optimizer = optim.SGD(model.parameters(), lr = LEARNING_RATE)
#optimizer = optim.Adam(model.parameters())

# MSE loss
loss_fn = nn.MSELoss()

for epoch in range(epoch_count):
  losses, accs = [], []

  # set train mode
  model.train()
```

Epoch 수에 맞게 학습 진행

```
for n in range(step_count):
  # optimizer init
  optimizer.zero_grad()

  # get train data
  train_x, train_y = get_train_data(mb_size, n)

  # dataset to torch
  x = torch.from_numpy(train_x).float()
  y = torch.from_numpy(train_y).float()
  x, y = x.to(device), y.to(device)

  # forward
  y_pred = model(x)
  acc = eval_accuracy(y_pred.to('cpu').detach().numpy(), train_y)
  accs.append(acc)

  # loss
  loss = loss_fn(y_pred, y)

  # backprop
  loss.backward()
  losses.append(loss.item())

  # weight, bias 업데이트
  optimizer.step()
```

train dataset load 및 forward 해 줌

```
    # run test & eval
    model.eval()
    x = torch.from_numpy(test_x).float()
    x = x.to(device)
    y_pred = model(x)
    acc = eval_accuracy(y_pred.to('cpu').detach().numpy(), test_y)

    print('Epoch {}: loss={:5.3f}, accuracy={:5.3f}/{:5.3f}'. ₩
          format(epoch+1, np.mean(losses), np.mean(accs), acc))
```

1 epoch 후 test와 eval 진행

```
device: cpu
Net(
  (fc1): Linear(in_features=8, out_features=4, bias=True)
  (fc2): Linear(in_features=4, out_features=2, bias=True)
  (fc3): Linear(in_features=2, out_features=1, bias=True)
)
Epoch 1: loss=0.084, accuracy=0.921/0.886
Epoch 2: loss=0.075, accuracy=0.921/0.886
Epoch 3: loss=0.074, accuracy=0.921/0.886
Epoch 4: loss=0.074, accuracy=0.921/0.886
Epoch 5: loss=0.074, accuracy=0.921/0.886
Epoch 6: loss=0.074, accuracy=0.921/0.886
Epoch 7: loss=0.074, accuracy=0.921/0.886
Epoch 8: loss=0.073, accuracy=0.921/0.886
Epoch 9: loss=0.073, accuracy=0.921/0.886
Epoch 10: loss=0.073, accuracy=0.921/0.886
```

문제 #3 고양이와 개 이미지를 판별

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import transforms
```

필요 모듈들 import

```
train_dataset_path = "/content/drive/MyDrive/training_set"
test_dataset_path = "/content/drive/MyDrive/test_set"
```

데이터셋이 저장된 주소를 불러옴

```
trans = transforms.Compose([transforms.Resize((96,96)), transforms.ToTensor()])
train_data = torchvision.datasets.ImageFolder(root = train_dataset_path, transform = trans)
test_data = torchvision.datasets.ImageFolder(root = test_dataset_path, transform = trans)
```

이미지를 96x96 사이즈로 변환

```
print('class:', train_data.classes)
print('train_data:', len(train_data))
print('test_data:', len(test_data))
```

```
class: ['cats', 'dogs']
train_data: 8015
test_data: 2023
```

클래스는 고양이와 개로 2개가 구성되어 있으며 훈련 데이터, 테스트 데이터의 개수는 각각 8015, 2023 이다.

```
train_data_loader = torch.utils.data.DataLoader(train_data,
                                                batch_size=32,
                                                shuffle = True)
test_data_loader = torch.utils.data.DataLoader(test_data,
                                                batch_size=32,
                                                shuffle = True)
```

데이터를 로드 해 줌, 배치 사이즈는 훈련, 테스트 모두 32

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 6,
                           kernel_size = 3, stride = 1, padding = 2)
    self.bn1 = nn.BatchNorm2d(6)
    self.max_pool1 = nn.MaxPool2d(kernel_size = 2)


    self.conv2 = nn.Conv2d(in_channels = 6, out_channels = 12,
                           kernel_size = 3, stride = 1, padding = 1)
    self.bn2 = nn.BatchNorm2d(12)
    self.max_pool2 = nn.MaxPool2d(kernel_size = 2)


    self.conv3 = nn.Conv2d(in_channels = 12, out_channels = 24,
                           kernel_size = 3, stride = 1, padding = 1)
    self.bn3 = nn.BatchNorm2d(24)
    self.max_pool3 = nn.MaxPool2d(kernel_size = 2)


    self.conv4 = nn.Conv2d(in_channels = 24, out_channels = 48,
                           kernel_size = 3, stride = 1, padding = 1)
    self.bn4 = nn.BatchNorm2d(48)
    self.max_pool4 = nn.MaxPool2d(kernel_size = 2)


    self.conv5 = nn.Conv2d(in_channels = 48, out_channels = 96,
                           kernel_size = 3, stride = 1, padding = 1)
    self.bn5 = nn.BatchNorm2d(96)
    self.max_pool5 = nn.MaxPool2d(kernel_size = 2)
    self.dropout1 = nn.Dropout2d(0.5)
    self.fc1 = nn.Linear(864, 512)
```

합성곱층은 5층까지 쌓아 줌.

```
def forward(self, x):
  x = self.conv1(x)
  x = self.bn1(x)
  x = F.relu(x)
  x = self.max_pool1(x)

  x = self.conv2(x)
  x = self.bn2(x)
  x = F.relu(x)
  x = self.max_pool2(x)

  x = self.conv3(x)
  x = self.bn3(x)
  x = F.relu(x)
  x = self.max_pool3(x)

  x = self.conv4(x)
  x = self.bn4(x)
  x = F.relu(x)
  x = self.max_pool4(x)

  x = self.conv5(x)
  x = self.bn5(x)
  x = self.max_pool5(x)
  x = self.dropout1(x)
  x = torch.flatten(x, 1)
  x = self.fc1(x)
  return x
```

합성곱, 정규화, ReLU, 맥스 풀링 순으로 반복하여 학습

```
# hyperparameters
LEARNING_RATE = 0.001
epoch_count = 10

# gpu check
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
print("device:", device)

# model set (gpu)
model = Net().to(device)
print(model)

# SGD optimizer
#optimizer = optim.SGD(model.parameters(), lr = LEARNING_RATE)
optimizer = optim.Adam(model.parameters())

# MSE loss
loss_fn = nn.CrossEntropyLoss()

for epoch in range(epoch_count):
```

하이퍼파라미터 값을 지정해준다

Device는 cuda로 설정

옵티마이저로는 아담 알고리즘을 사용

주석에는 MSE라 적혀 있지만 크로스 엔트로피를 사용한다

```
# set train mode
model.train()

for batch_idx, (x, y) in enumerate(train_data_loader):
  x, y = x.to(device), y.to(device)
  optimizer.zero_grad()
  y_pred = model(x)
  loss = loss_fn(y_pred, y)
  loss.backward()
  optimizer.step()

print('Train Epoch: {} [{} ({:.0f}%)]₩tLoss: {:.6f}'.format(
      epoch, len(train_data_loader.dataset),
      100. * batch_idx / len(train_data_loader), loss.item()))
```

훈련 모드에서의 정확도를 구한다

```
# set test mode
model.eval()
test_loss = 0
correct = 0
for x, y in test_data_loader:
  x, y = x.to(device), y.to(device)
  y_pred = model(x)
  test_loss += loss_fn(y_pred, y).item()  # sum up batch loss
  pred = y_pred.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
  correct += pred.eq(y.view_as(pred)).sum().item()

test_loss /= len(test_data_loader.dataset)

print('₩nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)₩n'.format(
    test_loss, correct, len(test_data_loader.dataset),
    100. * correct / len(test_data_loader.dataset)))
```

테스트 모드에서의 정확도를 구한다

```
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
  (bn1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (max_pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (max_pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (max_pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(24, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn4): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (max_pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv5): Conv2d(48, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn5): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (max_pool5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=864, out_features=512, bias=True)
)
Train Epoch: 0 [8005 (100%)]    Loss: 0.712862

Test set: Average loss: 0.0173, Accuracy: 1476/2023 (73%)

Train Epoch: 1 [8005 (100%)]    Loss: 0.118233

Test set: Average loss: 0.0235, Accuracy: 1317/2023 (65%)

Train Epoch: 2 [8005 (100%)]    Loss: 0.294983

Test set: Average loss: 0.0157, Accuracy: 1570/2023 (78%)

Train Epoch: 3 [8005 (100%)]    Loss: 0.761348

Test set: Average loss: 0.0154, Accuracy: 1563/2023 (77%)

Train Epoch: 4 [8005 (100%)]    Loss: 0.265825

Test set: Average loss: 0.0175, Accuracy: 1537/2023 (76%)

Train Epoch: 5 [8005 (100%)]    Loss: 0.126439

Test set: Average loss: 0.0168, Accuracy: 1554/2023 (77%)

Train Epoch: 6 [8005 (100%)]    Loss: 0.158501

Test set: Average loss: 0.0197, Accuracy: 1487/2023 (74%)

Train Epoch: 7 [8005 (100%)]    Loss: 0.760115

Test set: Average loss: 0.0164, Accuracy: 1597/2023 (79%)

Train Epoch: 8 [8005 (100%)]    Loss: 0.041094

Test set: Average loss: 0.0155, Accuracy: 1641/2023 (81%)

Train Epoch: 9 [8005 (100%)]    Loss: 0.810626

Test set: Average loss: 0.0149, Accuracy: 1618/2023 (80%)
```

합성곱을 5층 까지 쌓아주니 최대 81%의 정확도를 보인다.
사진에는 없지만 합성곱을 3층까지만 쌓아줬을 때 최대 정확도 약 70%인 것에 비해 증진된 결과를 얻었다.