딥알못 탈출하기 #9

고려대학교 지능시스템 연구실
유민형 제작

Pytorch - 딥러닝

- 파이토치를 이용해 딥러닝을 할 때 구성되는 구조와 필수적인 요소에 대해 알아보자.
- 신경망을 행렬곱으로 forward/backward를 직접 구현해주고 내부 그래디언트를 건드리는 경우도 있지만(새로운 방법을 제시하는 경우 직접하기도 한다), 대부분은 파이토치에서 제공하는 내부 함수를 통해 신경망의 구조만 만들어 간단하게 구현한다.

12ymh12@gmail.com

Pytorch - 딥러닝

- 파이토치를 이용한 학습은 다음과 같이 간단하게 진행된다.

한 Epoch 시작
미니배치 샘플링
Gradient 초기화
순전파(Forward)
Loss 계산
역전파(Backward)
Weight 업데이트

```
for epoch in range(total_epoch):
    for batch_idx, (x, y) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
```

- 위의 코드에 앞서 우리가 결정해 주어야 할 것에 대해 알아보자.
- total_epoch, train_loader, optimizer, model, device, criterion
- 그리고 사용된 함수, optimizer.zero_grad(), loss.backward(), optimizer.step()를 알아보자.

12ymh12@gmail.com

Pytorch – DataLoader

- 데이터를 불러오는 방법에 대해 알아보자. 앞의 코드에서는 train_loader에 해당하는 부분이다.

```
for batch_idx, (x, y) in enumerate(train_loader):
```

- 위 코드에서 train_loader의 특징

1. train_loader는 enumerate할 수 있고, 인덱스를 batch_idx변수에 할당하고, 값을 (x, y) 튜플에 할당한다.
2. 파이썬 for문의 in 다음에 들어가야 하므로 train_loader는 .__iter__()를 내부 함수로 가져야한다.
3. train_loader는 input값인 x와 ground truth인 y의 쌍의 미니배치를 return해주어야한다.

12ymh12@gmail.com

Pytorch - DataLoader

- 이 train_loader는 torch.utils.data.DataLoader의 instance다.
- 미니배치를 자동으로 생성해주는 DataLoader를 알아보자.

instance 생성

```
from torch.utils.data import DataLoader
train_loader = DataLoader()
```

```
Arguments

DataLoader(dataset, batch_size=1, shuffle=False,
           sampler=None, batch_sampler=None,
           num_workers=0,
           collate_fn=default_collate,
           pin_memory=False, drop_last=False,
           timeout=0, worker_init_fn=None)
```

사용 예시

```
train_loader = DataLoader(dataset, batch_size=128, shuffle=True, pin_memory=True, num_workers=4)
```

- DataLoader 함수는 dataset이 필수로 입력되어야 하고, mini-batch SGD를 위해 batch_size, shuffle(랜덤 여부)이 사용된다.
- (문제1) 그 외 pin_memory, num_workers 옵션이 자주 사용되는데, 어떤 옵션인지 조사하시오.

12ymh12@gmail.com

Pytorch - dataset

- 앞서 DataLoader에서 dataset을 입력 받아 미니배치를 구성하고 iteration에서 미니배치를 리턴한다. 데이터를 import하거나 전처리 혹은 augmentation하는 역할은 담당하지 않는다.
- 즉, dataset이 자체적으로 import, 전처리, augmentation을 하는 기능을 가지고 있어야 한다.
- import하는 관점으로 보면, 데이터가 한 파일 안에 전부 들어있는 경우도 있고, 파일 하나가 한 데이터인 경우가 많다.
- input data를 normalize하고, 좌우반전, crop, padding 등을 미니배치 생성전에 augment할 때도 있다.
- 이런 dataset을 구성하는 법을 알아보고, 파이토치에서 제공하는 데이터셋에는 무엇이 있고, 어떻게 이용하는 지도 알아보자.

12ymh12@gmail.com

Pytorch – dataset (1) import

- dataset은 torch.utils.data.Dataset을 상속해서 오버라이딩한다.
- __init__함수에 transform옵션이 있어야한다.
- __len__함수는 데이터의 개수 정보를 가지고 있고, 호출시 리턴해야한다.
- __getitem__함수는 index를 받아 해당 index의 데이터를 리턴하고, transform 옵션에 따라 변환하여 리턴한다.
- 위의 3가지 함수와 그 요건을 반드시 갖추어야 DataLoader와 호환이 된다.

```
class dogcat(Dataset):
    def __init__(self, transform=None):
        self.transform = transform

    def __len__(self):
        num = 10 # how many data
        return num

    def __getitem__(self, idx):
        image = image_of_idx

        if self.transform:
            image = self.transform(image)

        return image
```

12ymh12@gmail.com

Pytorch – dataset (1) import

- 강아지와 고양이 사진이 각각 5개씩 총 10개의 파일이 있다. 데이터셋을 구성한 코드를 살펴보자.

```python
from torch.utils.data import Dataset

class dogcat(Dataset):
    def __init__(self, path, transform=None):
        self.path = path
        self.transform = transform

    def __len__(self):
        num = 10
        return num

    def __getitem__(self, idx):
        file = ("%03d" % (idx+1)) + '.png'
        img_name = os.path.join(self.path,file)

        image = io.imread(img_name)
        height = image.shape[0]
        width = image.shape[1]
        image = Image.fromarray(image).convert("RGB")
        image = np.array(image.getdata())

        image = np.reshape(image, (height, width, 3))
        image = image.astype('uint8')
        image = Image.fromarray(image.reshape(height, width, 3), 'RGB')

        if self.transform:
            image = self.transform(image)

        return image
```

customize한 부분

- 파일 경로를 알아야 하므로 __init__의 옵션에 넣어준다.
- __len__함수에 개수 10를 지정해주고, 리턴하게 만든다.
- __getitem__에서 입력 받은 index에 따라 파일 이름을 변경해 읽어올 수 있도록 os.path.join 함수를 이용했다.
- 이미지를 읽어오는 함수인 skimage.io.imread를 이용했다.
- 이미지의 채널이 4개이기 때문에 RGB 3개 채널로 변경을 위해 PIL.Image.fromarray()함수와 .convert("RGB")함수를 썼다.
- numpy array로 변경해 3차원(height,width,channel)로 reshape해주고 'unit8' 타입으로 변경해준다.
- transform을 위해 PIL.Image.fromarray() 함수를 통해 이미지 형식으로 변경해준다.

필요한 라이브러리

```python
import os
from skimage import io
from PIL import Image
import numpy as np
```

Pytorch – dataset (1) import

- 위 class를 정의하고 아래처럼 instance를 만들어준다.
- 그 후 __getitem__() 함수에 인덱스를 넣으면 이미지가 출력된다.

```
In [42]: dataset=dogcat("C://유민형//사람 만들기//dogcat")

In [43]: dataset.__getitem__(6)
Out[43]:
```

```
In [44]: dataset.__getitem__(9)
Out[44]:
```

12ymh12@gmail.com

Pytorch - minibatch

- (문제2) 다음과 같이 데이터가 한 개의 파일에 전부 들어있는 경우에 데이터셋을 구성하려 한다. 225*225 한 장의 사진에서 75*75 9개 데이터를 만들어야 한다. 앞서 나온 예시처럼 torch.utils.data.Dataset 클래스를 상속하여 dataset 클래스를 구현하라.

파일 cats.png

12ymh12@gmail.com

Pytorch – dataset (2) augmentation

- 주로 dataset 클래스의 transform옵션을 통해 augmentation을 한다.
- Augmentation 연구를 할 때, 직접 transform 함수를 만드는 경우도 있다. dataset의 __getitem__에서 이미지를 numpy array로 만들면 다루기 쉽기 때문에 numpy array로 구현하는 방법이 가장 쉽다.
- torchvision.transforms 라이브러리가 기초적인 방법을 제공하기 때문에 이를 이용하기로 한다.
- torchvision.transforms.Compose([])를 통해 여러 방법을 조합하기 쉬워 대부분 이 방법을 이용한다.
- 여러 transform을 하고 pytorch tensor로 바꾸는 transform까지 하는 것이 가장 일반적이다.

12ymh12@gmail.com

Pytorch – dataset (2) augmentation

- dogcat 클래스에 transform 옵션을 준 다음 코드를 보자.

```
dataset=dogcat("C://유민형//사람 만들기//dogcat",
               transform=transforms.Compose([
                       transforms.CenterCrop((150,200)),
                       transforms.Resize((30,40)),
                       transforms.ToTensor(),
                       transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))]))

a = dataset.__getitem__(6)
```

```
In [46]: a
Out[46]:
tensor([[[-0.6941, -0.6235, -0.5765,  ..., -0.0431,  0.1216,  0.0588],
         [-0.6627, -0.6157, -0.5922,  ..., -0.2000,  0.0431,  0.0824],
         [-0.6157, -0.6157, -0.6235,  ..., -0.2863, -0.1373,  0.0039],
         ...,
         [-0.4588, -0.2314, -0.1608,  ...,  0.0667,  0.1216,  0.2235],
         [-0.6471, -0.2941, -0.0667,  ...,  0.3961,  0.4039,  0.3804],
         [-0.6863, -0.4980,  0.0039,  ...,  0.3882,  0.2863,  0.2471]],

        [[-0.7255, -0.6863, -0.6549,  ...,  0.0824,  0.1451, -0.0196],
         [-0.6941, -0.6706, -0.6627,  ..., -0.0667,  0.0980,  0.0275],
         [-0.6549, -0.6784, -0.6784,  ..., -0.1294, -0.0588, -0.0275],
         ...,
         [-0.3098, -0.0667,  0.0039,  ...,  0.2157,  0.2706,  0.3647],
         [-0.5059, -0.1294,  0.0902,  ...,  0.5216,  0.5216,  0.4980],
         [-0.5294, -0.3176,  0.1529,  ...,  0.5216,  0.4196,  0.3804]],

        [[-0.8196, -0.7804, -0.7333,  ..., -0.4118, -0.2235, -0.2471],
         [-0.7804, -0.7804, -0.7725,  ..., -0.5765, -0.3098, -0.2314],
         [-0.7176, -0.7961, -0.7961,  ..., -0.6706, -0.4980, -0.3176],
         ...,
         [-0.7412, -0.5922, -0.5529,  ..., -0.1765, -0.1059,  0.0039],
         [-0.8510, -0.6078, -0.4275,  ...,  0.1294,  0.1529,  0.1451],
         [-0.8353, -0.7098, -0.2784,  ...,  0.1686,  0.0745,  0.0353]]])
```

- transforms.Compose([])를 통해 4개 방법을 조합했다.
- CentorCrop을 통해 사진의 중심을 기준으로 height 150, width 200인 부분을 crop했다. (각기 다른 사진의 사이즈를 맞추기 위해)
- (150, 200)의 사이즈를 (30, 40)으로 downsample했다.
- ToTensor함수를 통해 이미지를 텐서로 변경했다.
- 3개의 채널에 대해 평균(0.5, 0.5, 0.5), 분산(0.5, 0.5, 0.5)로 정규화했다.

- 사진이 출력됐던 아까와는 달리, 텐서가 출력된다.

- (문제3) trasforms 라이브러리에서 또 어떤 방법들을 제공하는지 3가지 이상 조사하시오.

필요한 라이브러리

```
from torchvision import transforms
```

12ymh12@gmail.com

Pytorch - minibatch

- 이제 DataLoader와 dataset으로 minibatch를 만들어보자.

```
dataset=dogcat("C://유민형//사람 만들기//dogcat",
               transform=transforms.Compose([
                       transforms.CenterCrop((150,200)),
                       transforms.Resize((30,40)),
                       transforms.ToTensor(),
                       transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))]))

train_loader = DataLoader(dataset, batch_size=5)
```

```
In [3]: for x in train_loader:
   ...:     print(x.shape)
   ...:
torch.Size([5, 3, 30, 40])
torch.Size([5, 3, 30, 40])
```

전체 10개에서 5개씩 미니배치로 만들면
자동으로 2번만 출력되고 for문이 종료된다.

```
In [7]: train_loader.__iter__().next().shape
Out[7]: torch.Size([5, 3, 30, 40])
```

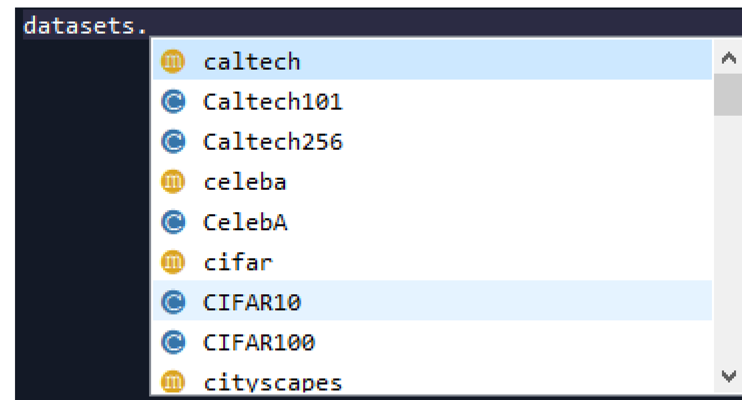
- DataLoader에 dataset을 넣고 인스턴스를 선언해준다.
- 인스턴스에 .__iter__().next() 하위 함수를 통해 샘플링을 해볼 수 있다.
- shape을 통해 확인해보니, 5개의 데이터가 3개의 차원에 대해 (30, 40) 사이즈인 텐서임을 알 수 있다.

12ymh12@gmail.com

Pytorch – Benchmark Dataset

- 어떤 새로운 방법이 제시되었을 때, 평가를 위한 기준이 되는 데이터셋을 benchmark dataset이라 한다. 대표적인 예로 MNIST, CIFAR10, ImageNet 등이 있다.
- torchvision.datasets 함수에서 여러 데이터셋을 제공한다.

```
train_loader = DataLoader(
            datasets.CIFAR10(
                    "../data/CIFAR10",
                    train=True,
                    download=True,
                    transform=transforms.Compose([
                            transforms.RandomHorizontalFlip(),
                            transforms.RandomCrop(32, padding=4),
                            transforms.ToTensor(),
                            transforms.Normalize(
                                    (0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))]
                            ),
                    ),
            batch_size=128, shuffle=True, pin_memory=True)
```



- (문제4) MNIST, CIFAR10, CIFAR100, ImageNet의 train/test셋은 각각 몇 개의 데이터가 있고, 차원은 어떻게 되는지 기본 정보를 조사하라.

12ymh12@gmail.com

Pytorch - optimizer

- 5장의 Optimization Strategy에서 다양한 최적화 방법을 배웠다. 이 최적화 방법들을 pytorch에서 제공해준다.
- torch.optim 라이브러리는 다음과 같이 활용된다.

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
```

```
optimizer_G = torch.optim.Adam(generator.parameters(), lr=opt.lr, betas=(opt.b1, opt.b2))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=opt.lr, betas=(opt.b1, opt.b2))
```

- torch.optim의 하위 함수로 SGD나 Adam같은 최적화 방법을 제공한다. 각 함수는 신경망의 파라미터를 입력 받는다. 즉, optimizer의 인스턴스를 만들기 전에, 신경망을 구성해야 한다.
- learning rate도 optimize 요소이기 때문에 필수 옵션으로 받는다

12ymh12@gmail.com

Pytorch - optimizer

- 가장 앞에 나온 pytorch의 딥러닝 절차를 보자.

```
for epoch in range(total_epoch):
    for batch_idx, (x, y) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
```

- optimizer가 두 번 등장한다.
- 마지막의 optimizer.step()은 optimizer가 담당하는 신경망의 파라미터를 loss로부터 얻은 gradient로 업데이트한다.
- 특이한 것은 optimizer.zero_grad()인데, gradient를 초기화하는 것이다. 이는 pytorch의 큰 특징 중 하나로, 이 함수를 이용해 초기화 하지 않으면 이전 iteration들의 gradient가 누적되어 남아있다.

12ymh12@gmail.com

Pytorch - Model

- 이제 신경망을 구성하는 방법을 알아보자.
- torch.nn.Module을 오버라이딩해서 신경망을 구성한다.
- __init__함수에서 super(신경망, self).__init__()을 통해 nn.Module의 __init__함수를 호출하는 부분이 있어야한다. 해당 기능이 없으면 신경망이 만들어지지 않는다.
- __forward__함수는 input을 받아 forward를 진행해준다. 즉, 이 부분에 우리가 원하는 신경망을 디자인해서 넣어준다. output으로 forward를 완료한 값이 리턴된다.

```
class NeuralNet(nn.Module):
  def __init__(self):
    super(NeuralNet, self).__init__()

  def forward(self, x):

    return output
```

12ymh12@gmail.com

Pytorch – Model (1) Design

- 개와 고양이를 분류하기 위한 3개의 linear layer를 가진 신경망을 구성해보자.

```
import torch.nn as nn
```

```
class NeuralNet(nn.Module):
  def __init__(self):
    super(NeuralNet, self).__init__()

    self.layer1 = nn.Linear(3*30*40, 1000)
    self.layer2 = nn.Linear(1000, 500)
    self.layer3 = nn.Linear(500, 2)

    self.relu = nn.ReLU()
    self.softmax = nn.Softmax()

  def forward(self, x):
      x_reshaped = x.view(-1,3*30*40)
      out1 = self.layer1(x_reshaped)
      out1_activated = self.relu(out1)
      out2 = self.layer2(out1_activated)
      out2_activated = self.relu(out2)
      out3 = self.layer3(out2_activated)
      output = self.softmax(out3)

      return output
```

- __init__함수에 사용할 layer의 클래스를 인스턴스화한다. torch.nn에 다양한 종류의 커널(혹은 필터)이 존재한다. nn.Linear를 사용해 Linear layer를 추가해보자.
- nn.linear()는 input feature의 차원과 output feature의 차원을 입력 받아야 한다. 가장 처음 layer에서는 RGB채널을 가진 30*40 이미지가 input이므로, 3*30*40의 차원을 입력하고 이를 1000 차원으로 줄이자(1000이란 숫자는 마음대로 정한 것이다). 그 다음 layer에서는 1000개의 차원을 500차원으로 줄이자. 마지막 layer에서는 500개의 차원을 2차원으로 줄이자.
- torch.nn에는 activation function도 제공한다. weight가 없는 activation function은 한 개의 인스턴스만 만들어도 여러 번 사용할 수 있다. nn.ReLU()와 nn.Softmax()를 만들자.
- RGB채널을 가진 30*40 이미지 5개가 합쳐진 미니배치가 forward에 입력된다. 그런데 linear layer에 들어가기 위해서는 벡터 형태로 바뀌어야 하기 때문에 x.view()를 통해 차원을 바꾼다. (5,3,30,40)의 차원이 (5, 3600)으로 바뀐다.

12ymh12@gmail.com

Pytorch – Model (1) Design

- 개와 고양이를 분류하기 위한 3개의 linear layer를 가진 신경망을 구성해보자.

```python
import torch.nn as nn
```

```python
class NeuralNet(nn.Module):
  def __init__(self):
    super(NeuralNet, self).__init__()

    self.layer1 = nn.Linear(3*30*40, 1000)
    self.layer2 = nn.Linear(1000, 500)
    self.layer3 = nn.Linear(500, 2)

    self.relu = nn.ReLU()
    self.softmax = nn.Softmax(dim=1)

  def forward(self, x):
      x_reshaped = x.view(-1,3*30*40)
      out1 = self.layer1(x_reshaped)
      out1_activated = self.relu(out1)
      out2 = self.layer2(out1_activated)
      out2_activated = self.relu(out2)
      out3 = self.layer3(out2_activated)
      output = self.softmax(out3)

      return output
```

- RGB채널을 가진 30*40 이미지 5개가 합쳐진 미니배치가 forward에 입력된다. 그런데 linear layer에 들어가기 위해서는 벡터 형태로 바뀌어야 하기 때문에 x.view()를 통해 차원을 바꾼다. (5,3,30,40)의 차원이 (5, 3600)으로 바뀐다.
- layer1을 통과한 벡터는 out1에 저장되고 out1은 ReLU를 통과해 out1_activated에 저장된다.
- 같은 방식으로 out3까지 layer를 통과한 후 Softmax를 통과해 output에 저장된다. 이 output이 출력된다. 이 때 softmax 과정이 배치 차원으로 되는 것이 아니라 output node끼리 되어야 하기 때문에 dim=1을 기준으로 잡아놨다.
- forward하는 과정에서 x, x_reshaped, out1, out1_activated, out2, out2_activated, out3, output은 메모리에 남아있게 된다. 메모리에 남는 값을 최소화하고 있는 값을 최대한 활용하는 것도 중요한 이슈이니 기억하자.
- nn.Linear(bias=True)로 bias 옵션이 True가 default이다. trainable한 bias가 output node의 개수만큼 생성된다. 예를 들면. nn.Linear(3*30*40, 1000)에서 1000개의 bias가 자동으로 생성된다.

12ymh12@gmail.com

Pytorch – Model (1) Design

- torch.nn.functional를 통해 activation function을 이용할 수도 있다.

```
import torch.nn.functional as F

class NeuralNet(nn.Module):
  def __init__(self):
    super(NeuralNet, self).__init__()

    self.layer1 = nn.Linear(3*30*40, 1000)
    self.layer2 = nn.Linear(1000, 500)
    self.layer3 = nn.Linear(500, 2)

  def forward(self, x):
      x_reshaped = x.view(-1,3*30*40)
      out1 = self.layer1(x_reshaped)
      out1_activated = F.relu(out1)
      out2 = self.layer2(out1_activated)
      out2_activated = F.relu(out2)
      out3 = self.layer3(out2_activated)
      output = F.softmax(out3)

      return output
```

- torch.nn과 torch.nn.functional 둘 다 수학적인 함수를 갖고 있다. 차이점은 torch.nn은 trainable parameter를 가진 layer를 만들 수 있다는 것이다. layer를 torch.nn의 인스턴스로 만들게 되면 원하는 layer에 쉽게 접근할 수 있기 때문에 torch.nn을 이용한다. torch.nn.functional의 함수 자체에는 접근할 필요가 많지 않아 인스턴스를 굳이 만들지 않아도 된다. 편의를 위해 만드는 경우가 대부분이다.

12ymh12@gmail.com

Pytorch – Model (1) Design

- torch.nn.Sequential() 활용하기

```
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()

        self.hidden_layer = self.make_layer()

    def make_layer(self):
        layers = [nn.Linear(3*30*40, 1000), nn.ReLU(), nn.Linear(1000, 500), nn.ReLU(), nn.Linear(500, 2), nn.Softmax(dim=1)]

        return nn.Sequential(*layers)

    def forward(self, x):
        x_reshaped = x.view(-1,3*30*40)
        output = self.hidden_layer(x_reshaped)

        return output
```

- torch.nn의 layer들을 list에 저장하고 torch.nn.Sequential()을 이용하면 순차적으로 layer를 연결한다.
- class 하위의 함수를 만들어 layer를 자동화해서 만드는 데 사용될 수 있다.

12ymh12@gmail.com

Pytorch – Model (2) Forward Pass

- 신경망을 인스턴스화 해보자.

```
model = NeuralNet()
device = torch.device("cuda:0")
model.to(device)
```

- 정말 간단히 인스턴스를 만들어주면 된다. GPU를 사용하기 위해 .to()함수로 device를 변경해준다.
- 이제 앞서 만든 train_loader의 샘플을 forward 해보자.

```
sample = train_loader.__iter__().next()
output = model(sample.to(device))
```

- sample역시 model의 device와 일치시킨 후 model()안에 입력하면 자동으로 .forward() 함수가 진행된다.

```
In [3]: output
Out[3]:
tensor([[0.4810, 0.5190],
        [0.4906, 0.5094],
        [0.5019, 0.4981],
        [0.4923, 0.5077],
        [0.4832, 0.5168]], device='cuda:0', grad_fn=<SoftmaxBackward>)
```

12ymh12@gmail.com

Pytorch – Model (3) Access to weight

- 앞의 output으로 출력된 값을 보고 의문이 들어야한다.
- 신경망 구조만 만들었는데 input을 넣었을 때 어떻게 계산이 되는 것인가? 즉, weight값들은 입력 안 했는데 어떻게 결과가 나왔는가? 답은 Pytorch에서 신경망을 만들면, 자동으로 random하게 아주 작은 값들이 초기 weight에 할당된다.
- 초기 weight를 내가 원하는 값으로 설정하는 경우나 내부 weight값을 알고 싶고, 접근해서 수정하는 경우 어떻게 하는지 알아보자.

12ymh12@gmail.com

Pytorch – Model (3) Access to weight

- model.modules()는 신경망의 layer 정보를 갖고 있다.

```
In [3]: model = NeuralNet()

In [4]: for m in model.modules():
   ...:     print(m)
   ...:
NeuralNet(
  (layer1): Linear(in_features=3600, out_features=1000, bias=True)
  (layer2): Linear(in_features=1000, out_features=500, bias=True)
  (layer3): Linear(in_features=500, out_features=2, bias=True)
  (relu): ReLU()
  (softmax): Softmax()
)
Linear(in_features=3600, out_features=1000, bias=True)
Linear(in_features=1000, out_features=500, bias=True)
Linear(in_features=500, out_features=2, bias=True)
ReLU()
Softmax()
```

- model.modules()를 통해 NeuralNet 클래스의 __init__함수에 정의된 인스턴스의 정보를 보여준다.
- 가장 처음에는 NeuralNet 클래스의 전체 인스턴스가 한번에 묶여있고, 그 다음부터 각각의 인스턴스 정보가 있다.
- nn.Sequential()을 통해 정의한 경우 nn.Sequential로 정의된 한 묶음도 하나의 정보로 들어가게 된다.

```
In [5]: model.modules()
Out[5]: <generator object Module.modules at 0x000002425127FB10>

In [6]: model.modules()[0]
Traceback (most recent call last):

  File "<ipython-input-6-7a3b13ea1f9b>", line 1, in <module>
    model.modules()[0]

TypeError: 'generator' object is not subscriptable
```

- 중요한 점은 model.modules() 자체에는 정보가 없고, 인덱싱을 할 수도 없다.
- 두 번째 layer인 nn.Linear(1000, 500)에 접근하기 위해서는 어떻게 해야할까?

12ymh12@gmail.com

Pytorch – Model (3) Access to weight

- for문 이용하기

```
In [3]: model = NeuralNet()

In [4]: for m in model.modules():
   ...:     print(m)
   ...:
NeuralNet(
  (layer1): Linear(in_features=3600, out_features=1000, bias=True)
  (layer2): Linear(in_features=1000, out_features=500, bias=True)
  (layer3): Linear(in_features=500, out_features=2, bias=True)
  (relu): ReLU()
  (softmax): Softmax()
)
Linear(in_features=3600, out_features=1000, bias=True)
Linear(in_features=1000, out_features=500, bias=True)
Linear(in_features=500, out_features=2, bias=True)
ReLU()
Softmax()
```

- .modules()는 iterative한 하위 함수를 갖고 있어 for문을 통해 사용할 수 있다.

- list 이용하기

```
In [7]: m_list = list(model.modules())

In [8]: m_list
Out[8]:
[NeuralNet(
   (layer1): Linear(in_features=3600, out_features=1000, bias=True)
   (layer2): Linear(in_features=1000, out_features=500, bias=True)
   (layer3): Linear(in_features=500, out_features=2, bias=True)
   (relu): ReLU()
   (softmax): Softmax()
 ),
 Linear(in_features=3600, out_features=1000, bias=True),
 Linear(in_features=1000, out_features=500, bias=True),
 Linear(in_features=500, out_features=2, bias=True),
 ReLU(),
 Softmax()]
```

- 리스트로 만들면 각 정보들이 리스트에 차례로 저장된다. deepcopy가 아니기 때문에 값을 바꾸면 model의 값을 바꿀 수 있다.

두 가지 방법을 이용해 initial weight를 수정해보자.

12ymh12@gmail.com

Pytorch – Model (3) Access to weight

- 앞의 두 가지 방법으로 모든 Linear layer의 weight를 0 bias를 1로 만들자.

```
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()

        self.layer1 = nn.Linear(3*30*40, 1000)
        self.layer2 = nn.Linear(1000, 500)
        self.layer3 = nn.Linear(500, 2)

        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

        for m in self.modules():
            if isinstance(m, nn.Linear):
                m.weight.data[:] = 0
                m.bias.data[:] = 1

    def forward(self, x):
        x_reshaped = x.view(-1,3*30*40)
        out1 = self.layer1(x_reshaped)
        out1_activated = self.relu(out1)
        out2 = self.layer2(out1_activated)
        out2_activated = self.relu(out2)
        out3 = self.layer3(out2_activated)
        output = self.softmax(out3)

        return output
```

- for문으로 weight 다루기
- __init__부분에 .modules()에 대한 for문을 만든다.
- .modules()에서 나온 값 m이 nn.Linear의 instance인지 확인되면
- m의 하위 값인 m.weight.data에 0을 할당하고, m.bias.data에 1을 할당한다. 각각이 nn.Linear에 해당하는 weight와 bias값이 저장된 곳이다.

12ymh12@gmail.com

Pytorch – Model (3) Access to weight

- 앞의 두 가지 방법으로 모든 Linear layer의 weight를 0 bias를 1로 만들자.

```
In [2]: model = NeuralNet()

In [3]: m_list = list(model.modules())

In [4]: m_list
Out[4]:
[NeuralNet(
   (layer1): Linear(in_features=3600, out_features=1000, bias=True)
   (layer2): Linear(in_features=1000, out_features=500, bias=True)
   (layer3): Linear(in_features=500, out_features=2, bias=True)
   (relu): ReLU()
   (softmax): Softmax()
 ),
 Linear(in_features=3600, out_features=1000, bias=True),
 Linear(in_features=1000, out_features=500, bias=True),
 Linear(in_features=500, out_features=2, bias=True),
 ReLU(),
 Softmax()]

In [5]: m_list[1].weight.data[:] = 0

In [6]: m_list[1].bias.data[:] = 1

In [7]: m_list[2].weight.data[:] = 0

In [8]: m_list[2].bias.data[:] = 1

In [9]: m_list[3].weight.data[:] = 0

In [10]: m_list[3].bias.data[:] = 1
```

- list로 weight 다루기
- 모델을 인스턴스로 하나 만들어준다.
- 인스턴스.modules()를 리스트로 만들어준다.
- list에서 layer의 인덱스를 찾고, 해당 인덱스에서 .weight.data에 0을 .bias.data에 1을 할당해준다.
- deepcopy를 한 것이 아니기 때문에 이렇게 할당해도 원본 텐서의 값이 변경된다.

12ymh12@gmail.com

Pytorch – Model (3) Access to weight

- 초기 linear layer의 weight를 0 bias를 1로 만들어주면 output은 다음과 같이 나온다. 22 페이지의 output과 비교해볼 때 균형 잡힌 output이 나왔다.

```
In [2]: sample = train_loader.__iter__().next()
   ...: output = model(sample.to(device))

In [3]: output
Out[3]:
tensor([[0.5000, 0.5000],
        [0.5000, 0.5000],
        [0.5000, 0.5000],
        [0.5000, 0.5000],
        [0.5000, 0.5000]], device='cuda:0', grad_fn=<SoftmaxBackward>)
```

- 초기 weight를 다루는 것 외에도 network pruning, regularizing norm 등을 만들 때 weight로 접근이 필요하니 여기서 다룬 내용을 기억해두자.

12ymh12@gmail.com

Pytorch - Criterion

- 이제 loss를 만드는 criterion에 대해 알아보자.

```
for epoch in range(total_epoch):
    for batch_idx, (x, y) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
```

- 어떤 Loss를 계산할 때, 한 미니배치 단위로 구해야 하기 때문에 수식이 복잡해지고, 코드가 더러워진다.
- 이를 방지하기 위해 pytorch에서는 다양한 Loss function 클래스를 제공하며, 인스턴스를 만들어 loss를 구한다. 이 loss에 대해 .backward()함수를 이용해 자동 미분해서 그래디언트를 구할 수 있다.

12ymh12@gmail.com

Pytorch - Criterion

- criterion은 표준, 기준이라는 뜻으로 손실 혹은 목적함수의 척도를 포함하여 나타내기에 적절한 단어이다. 딥러닝에서는 보통 gradient descent를 하므로 loss를 쓰지만 ascent를 하는 경우도 간혹 있기에 criterion이란 말을 썼다.
- torch.nn에서 다양한 loss function을 제공한다.

```
criterion = nn.CrossEntropyLoss()
```

- cross-entropy loss로 인스턴스를 생성해준다.

12ymh12@gmail.com

Pytorch - Criterion

- torch.nn의 loss function이 제공하는 아주 편리한 기능이 있는데, 바로 one-hot encoding을 자동으로 해준다는 것이다.

```
loss = criterion(output, y.to(device))
```

- 여기 코드에서 output은 softmax에 의해 출력된 텐서이며 (batch_size 수, class 수) 차원을 갖고 있고, ground truth인 y는 (batch_size 수) 차원을 가지고 있다. y는 0, 1, 2, 등의 클래스를 나타내더라도 자동으로 (1,0,0), (0,1,0), (0,0,1)로 바뀌어 계산된다는 것이다.

12ymh12@gmail.com

Pytorch - Loss

- 미니배치를 이용해 SGD를 할 때 1 Epoch의 loss를 구할 때 주의할 점을 알아보자.

```
for epoch in range(total_epoch):
    for batch_idx, (x, y) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
        losses[epoch] += loss.item()
    losses[epoch] /= len(train_loader)
```

- 순전파-역전파-업데이트는 미니배치 단위로 구해진다. 때문에 1 Epoch의 모든 미니배치에 대한 loss를 평균내서 그 Epoch의 Loss를 구한다.
- practical하게는 epoch의 모든 미니배치의 loss를 더해서, train_loader의 길이로 나눠주는 방식으로 구한다.

12ymh12@gmail.com

Pytorch – Test/Train/no gradient

- 종종 train할 때와 inference(test)할 때 신경망의 기능이 다른 경우가 있다. 예를 들면 dropout의 경우 train할 때는 몇 개 노드가 사용이 안되지만 test할 때는 전체가 다 사용된다.
- 이런 경우를 위해 사용되는 방법을 소개한다.

```
model.train()
model.test()
with torch.no_grad():
```

- 인스턴스로 선언된 model에 .train()과 .test()를 통해 각각의 경우에 해당하는 기능이 자동으로 켜지고 꺼지게 해놨다.
- with torch.no_grad():를 선언하고 하위에 코드를 입력하면 그래디언트가 흐르지 않게 해서 텐서를 이용할 수 있다.

12ymh12@gmail.com

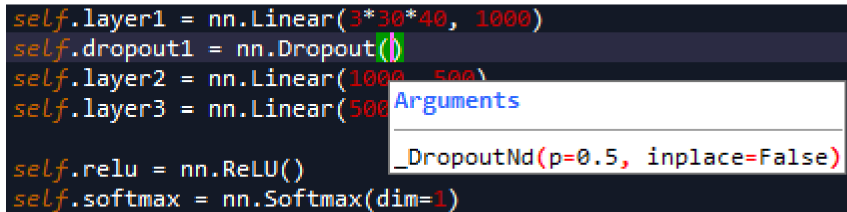
Pytorch - Test(Inference)

- Test할 때에는 loss도 중요하지만 accuracy가 어떻게 되는지도 중요하다. accuracy를 구하는 방법을 알아보자.

```
losses = torch.zeros((100))
for epoch in range(100):
    for batch_idx, (x, y) in enumerate(train_loader):
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
        losses[epoch] += loss.item()
    losses[epoch] /= len(train_loader)
    print("[Epoch:%d] [train_Loss:%f]" % ((1 + epoch), losses[epoch].item()), end=" ")

    accuracy = 0
    with torch.no_grad():
        model.eval()
        correct = 0
        test_loss = 0
        for x, y in test_loader:
            output = model(x.float().to(device))
            loss = criterion(output, y.to(device))
            pred = output.argmax(1, keepdim=True)
            correct += pred.eq(y.long().to(device).view_as(pred)).sum().item()
            test_loss += loss.item()

        accuracy = correct / len(test_loader.dataset)
    print("[Accuracy:%f] [Test_loss:%f]" % (accuracy, test_loss))
```

Python - Regularizer

- pytorch에서 Regularizer를 어떻게 사용할까
- 먼저 dropout의 경우 신경망을 구성할 때 인스턴스를 만들어 layer로 만들어야 한다. 다음 코드를 보자.



- dropout을 하고싶은 layer 뒤에 넣어 진행해주면 된다. 옵션으로 확률을 지정해줄 수 있다.

```
def forward(self, x):
    x_reshaped = x.view(-1,3*30*40)
    out1 = self.layer1(x_reshaped)
    out1_activated = self.relu(out1)
    dropout1 = self.dropout1(out1_activated)
```

12ymh12@gmail.com

Python - Regularizer

- L1, L2 norm을 loss에 추가하려 한다면 어떻게 해야 할까?

```
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()

        self.fc1 = nn.Linear(100, 10)

    def forward(self, x):

        output = self.fc1(x)

        return output

model = NeuralNet()
criterion = nn.CrossEntropyLoss()

loss = criterion(output, y) + torch.norm(model.fc1.weight.data)
loss.backward()
```

- 정답은 신경망의 weight로 직접 만들어 주어야 한다.
- 별다른 norm옵션은 제공하지 않는다. torch.norm()은 입력받는 텐서의 L1 norm을 계산해주는 함수일 뿐 신경망 전체의 norm을 계산해주지는 않는다. torch.abs()와 torch.sum() 등으로 적절히 만들어서 사용해야 한다.

12ymh12@gmail.com

Pytorch - tensorboard

- Tensorboard는 tensorflow에서 제공하는 툴로, log를 그래프로 시각화하여 보여주는 도구다. pytorch와도 호환이 가능하다.
- 우선 Tensorboard를 활용하지 않을 때 Epoch마다 loss와 accuracy를 보기 위해서 썼던 방법이다.

```
losses = torch.zeros((100))
for epoch in range(100):
    for batch_idx, (x, y) in enumerate(train_loader):
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
        losses[epoch] += loss.item()
    losses[epoch] /= len(train_loader)
    print("[Epoch:%d] [train_Loss:%f]" % ((1 + epoch), losses[epoch].item()), end=" ")

    accuracy = 0
    with torch.no_grad():
        model.eval()
        correct = 0
        test_loss = 0
        for x, y in test_loader:
            output = model(x.float().to(device))
            loss = criterion(output, y.to(device))
            pred = output.argmax(1, keepdim=True)
            correct += pred.eq(y.long().to(device).view_as(pred)).sum().item()
            test_loss += loss.item()

        accuracy = correct / len(test_loader.dataset)
    print("[Accuracy:%f] [Test_Loss:%f]" % (accuracy, test_loss))
```

- Loss를 저장하기 위해 losses라는 텐서를 미리 저장하고, 인덱싱으로 minibatch의 loss를 losses 텐서의 해당 epoch에 저장한다.
- print로 에포크와 로스, 정확도를 출력한다.
- plot을 해야하는 경우 matplotlib.pyplot을 활용해야하는데 이 경우에도 loss를 일일이 저장해야된다.
- 이럴 경우, 메모리를 과하게 차지하게 된다.

12ymh12@gmail.com

Pytorch - tensorboard

- 우선 텐서플로우 설치가 필요하다. 학습은 하지 않지만 라이브러리가 있어야한다.
- 아나콘다 터미널에 pip install tensorflow-gpu==1.14를 입력하자.

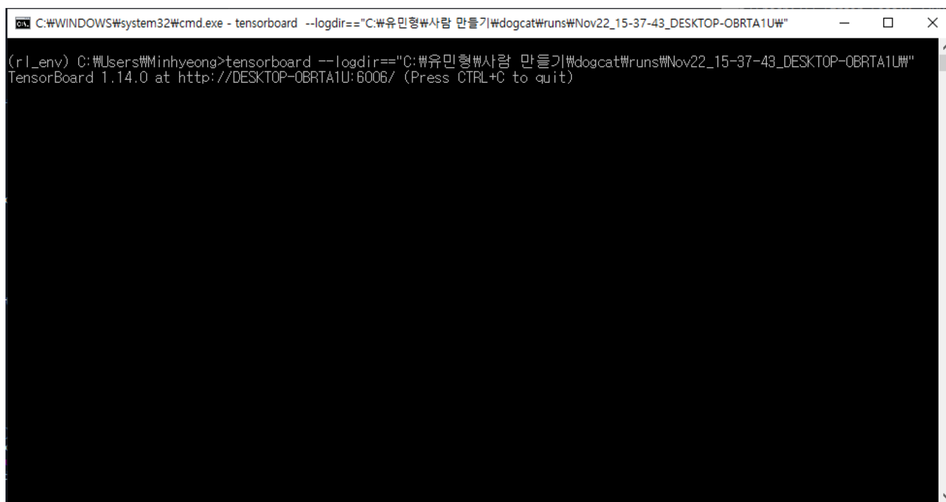
```python
from tensorboardX import SummaryWriter

losses = torch.zeros((100))
summary = SummaryWriter()
for epoch in range(100):
    for batch_idx, (x, y) in enumerate(train_loader):
        output = model(x.to(device))
        loss = criterion(output, y.to(device))
        loss.backward()
        optimizer.step()
        losses[epoch] += loss.item()
    losses[epoch] /= len(train_loader)
    summary.add_scalar('loss', losses[epoch].item(), epoch)
    print("[Epoch:%d] [train_loss:%f]" % ((1 + epoch), losses[epoch].item()), end=" ")

    accuracy = 0
    with torch.no_grad():
        model.eval()
        correct = 0
        test_loss = 0
        for x, y in test_loader:
            output = model(x.float().to(device))
            loss = criterion(output, y.to(device))
            pred = output.argmax(1, keepdim=True)
            correct += pred.eq(y.long().to(device).view_as(pred)).sum().item()
            test_loss += loss.item()

        accuracy = correct / len(test_loader.dataset)
    print("[Accuracy:%f] [Test_loss:%f]" % (accuracy, test_loss))
```

- 방법은 간단하다.
- SummaryWriter()의 인스턴스를 만들어주고
- 인스턴스에 .add_scalar()에 ‘이름’, loss, epoch를
- 입력해주면 로컬 저장 위치에 로그가 저장된다.

12ymh12@gmail.com

Pytorch - tensorboard

- 학습이 끝난 후에 아나콘다 터미널을 켜서
- tensorboard --logdir==“C:₩유민형₩사람만들기₩dogcat₩run₩”
- 위 명령어를 입력해준다. 파일명까지 디렉토리를 해줄 필요 없이 폴더만 해준다.
- 터미널에 나온 주소를 크롬에 복사해서 들어간다.



http://~의 주소를 복사해서 크롬에 입력하면 plot을 해준다.

12ymh12@gmail.com

Pytorch - Save/Load

- pytorch는 학습된 신경망을 저장할 때 숫자를 저장한다.
- 신경망 구조를 저장하지 않으니, 신경망 구조는 스크립트로 가지고 있어야한다.
- save/load 각 방법을 보자.

```
torch.save({'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': losses}, "저장할 경로")
```

- save할 때, epoch, model, optimizer, loss 모두 한 파일에 저장 가능하다.
- 파일 확장자는 pth, th 등을 사용한다.

```
model = NeuralNet()

if device == "cuda:0":
    checkpoint = torch.load("저장된 경로")
else:
    checkpoint = torch.load("저장된 경로", map_location=lambda storage, location: 'cpu')

model.load_state_dict(checkpoint['model_state_dict'])
```

- 모델을 불러오기 위해선 먼저 신경망을 인스턴스화 해야한다.
- torch.load는 저장된 모든 것을 가져오며 그 checkpoint에서 model이나 optimizer의 원하는 것을 인덱싱으로 .load_state_dict()한다.

12ymh12@gmail.com

Pytorch – MNIST tutorial

- (문제5)
- pytorch에서 제공하는 데이터셋 중 하나인 MNIST를 이용하여 손글씨를 분류하는 모델을 학습해보자.
- 자유롭게 모델을 구성하고 tensorboard를 이용해 plot해보자.
- 앞에서 나온 torch, torchvision을 이용해서 만들어보자.

12ymh12@gmail.com