Basic pytorch

Tensor

GPU와 같이 CPU가 아닌 다른 하드웨어 가속기에서 실행 가능한 sequence data.

numpy의 ndarray, array등과 유사한 data structure.

```
# tensor 생성
torch.tensor([1,2,3]) # 데이터 타입은 자동으로 유추

data = [1,2,3]
x_data = torch.tensor(data)
```

tensor operation

https://pytorch.org/docs/stable/torch.html

numpy to tensor

```
arr = np.array(data)
x_tensor = torch.from_numpy(arr)
```

tensor to numpy

```
x_tensor = torch.tensor([1,2,3])
x_np = x_tensor.numpy()
```

> CPU상의 텐서와 numpy는 메모리를 공유하므로 래퍼런스 됨.

random, const tensor

```
shape = (2,3)
rand_tensor = torch.rand(shape) # 2x3의 렌덤 텐서 생성
ones_tensor = torch.ones(shape) # 2x3의 모든 원소가 1
zeros_tensor = torch.zeros(shape) # 2x3의 모든 원소가 0
```

텐서의 속성

- 형태
- 자료형
- 장치 : tensor가 할당된 장치 (CPU, GPU 등...)

```
tensor = torch.rand(3,4)

print("tensor.shape")
```

```
print("tensor.dtype")
print("tensor.device")

'''
output
torch.Size([3, 4])
torch.float32
cpu
'''
```

차원 변환

```
tensor.rand(1,2)
print(tensor.shape)
tensor.reshape(2,1)
print(tensor.shape)

"""
output
(1,2)
(2,1)
"""
```

자료형 설정

원하는 자료형으로 선언

```
tensor = torch.rand((3,3), dtype=torch.float)
```

> https://pytorch.org/docs/stable/tensors.html#torch-tensor

장치 설정

```
# 현재 기기에 cuda가 사용가능한지 확인 후 이에 맞춰 변수에 할당
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
# 다음과 같이 텐서 선언과 함께 원하는 장치에 할당
tensor = torch.rand((1,1), device = DEVICE)

#다른 방법
cpu = torch.FloatTensor([1,2,3])
gpu = torch.cuda.FloatTensor([1,2,3])
```

in mac with apple silicon

```
#pytorch가 mps를 지원하는지 확인
torch.backends.mps.is_built()
#현재 기기에 mps가 있는지 확인
torch.backends.mps.is_available()
```

```
#mps에 할당
mps_device = torch.device("mps")
```

장치 변환

!서로 다른 장치간의 연산 불가능!

CPU상의 텐서와 넘파이는 연산 가능...

```
tensor = torch.FloatTensor([1,2,3])

tensor = tensor.cuda()
#or
tensor = tensor.to("cuda")
print(tensor)

"""
output
tensor([1., 2., 3.], device='cuda:0')
"""
```

참조

https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html

https://pytorch.org/docs/stable/tensors.html#torch-tensor

https://pytorch.org/docs/stable/torch.html

Dataset / Dataloader

why?

학습과정에서 데이터를 변형하고 매핑을 하면 모듈화, 재사용성, 가독성 등을 떨어트리게 됨으로 코드를 구좋거으로 설계할 수 있도록 사용

Dataset

> torch.utils.data.Dataset

학습에 필요한 데이터 샘플을 정제하고 레이블을 저장하는 기능을 제공한다.

Dataset 정의

Dataset class는 반드시 다음 3개의 함수를 구현해야 함

- `__init__`
- `__len__`
- `__getitem__`

`__init__`

객체가 생성될때 한번만 실행

- 학습에 사용될 데이터를 선언
- 학습에 필요한 형태로 변형

```
def __init__(self, file_path):
        df = pd.read_csv(file_path)
        self.x1 = df.iloc[:, 0].values
        self.x2 = df.iloc[:, 1].values
        self.y = df.iloc[:, 2].values
        self.length = len(df)
```

`__getitem__`

학습을 진행할 때 사용되는 하나의 행을 불러오는 과정

- 입력된 index에 해당하는 데이터 샘플을 불러오고 반환한다.
- 보통 데이터와 레이블을 맞춰서 반환

```
def __getitem__(self, index):
    x = torch.FloatTensor([self.x1[index], self.x2[index]])
    y = torch.FloatTensor([self.y[index]])
    return x, y
```

`__len__`

데이터셋의 샘플 갯수반환

```
def __len__(self):
    return self.length
```

```
class CustomDataset(Dataset):
    def __init__(self, file_path):
        df = pd.read_csv(file_path)
        self.x1 = df.iloc[:, 0].values
        self.x2 = df.iloc[:, 1].values
        self.y = df.iloc[:, 2].values
        self.length = len(df)

    def __getitem__(self, index):
        x = torch.FloatTensor([self.x1[index], self.x2[index]])
        y = torch.FloatTensor([self.y[index]])
        return x, y

    def __len__(self):
        return self.length
```

torchvision, torchtext, torchAudio

도메인 특화 라이브러리를 dataset과 같이 제공

ex)

```
from torchvison import datasets
```

```
train_data = datasets.CIFAR10(
    root = "./",
    train = True,
    download=True,
    transform = ToTensor()
)
```

- `root` 는 학습/테스트 데이터가 저장되는 경로입니다.
- `train` 은 학습용 또는 테스트용 데이터셋 여부를 지정합니다.
- `download=True` 는 `root` 에 데이터가 없는 경우 인터넷에서 다운로드합니다.
- `transform` 과 `target_transform` 은 특징(feature)과 정답(label) 변형(transform)을 지정합니다.

https://pytorch.org/vision/stable/datasets.html

https://pytorch.org/audio/stable/datasets.html

https://pytorch.org/text/stable/datasets.html

추가)

transform

- 데이터는 항상 머신러닝 알고리즘 학습에 필요한 최종 처리가 된 형태로 제공되지는 않음.
- transform을 통해 데이터를 조작하고 학습에 적합하게 만듦.
- pytorch에서 학습을 하기 위해선 정규화된 텐서 형태의 특징과 원-핫encode된 텐서 형태의 레이블이 필요
- 이러한 transform을 하기 위해 `ToTensor` 와 `Lambda` 를 사용합니다.

`ToTensor()`

PIL Image나 NumPy ndarray 를 FloatTensor 로 변환하고, 이미지의 픽셀의 크기 값을 [0., 1.] 범위로 비례하여 조정.

`lamda`

사용자 정의 람다 함수 사용

ex) 정수를 원-핫encoding된 텐서로 바꾸는 함수

```
target_transform = Lambda(lambda y: torch.zeros(
    10, dtype=torch.float).scatter_(dim=0, index=torch.tensor(y), value=1))
```

DataLoader

> torch.utils.data.DataLoader

Dataset에 저장된 데이터를 가져와 어떤 방식으로 활용할지 정의

- 배치 크기를 정해주거나, 데이터 셔플, 데이터로드 프로세스 수 등의 기능 제공

```
dataset = CustomDataset("./perceptron.csv")
train_dataset, test_dataset = random_split(dataset, [int(len(dataset)*0.8), int(len(data
train_dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True, drop_last=True
test_dataloader = DataLoader(test_dataset, batch_size=64, shuffle=True, drop_last=True)
```

> https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

---

참조

https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

https://pytorch.org/tutorials/beginner/basics/transforms_tutorial.html

https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset

https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

https://pytorch.org/vision/stable/datasets.html

https://pytorch.org/audio/stable/datasets.html

https://pytorch.org/text/stable/datasets.html

신경말 모델 설계

신경망은 데이터 연산을 수행하는 layer/module로 이루어짐, pytorch에서는 module이라고 부름

파이썬에서 제공하는 `torch.nn` 은 신경망을 구성하는 모든 구성 요소를 제공

pytorch의 모든 모듈은 nn.Module의 하의 클래스

> https://pytorch.org/docs/stable/nn.html
>
> https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module

클래스 정의

신경망 모델을 nn.Module의 하위클래스로 정의.

`__init__`에서 신경망 계층 초기화.

`nn.Module`을 상속받은 모든 클래스는

`forward` 메소드에 입력 데이터에 대한 연산을 구현.

```
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

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

        self.layer1 = nn.Sequential(
            nn.Linear(2, 2),
            nn.Sigmoid()
        )
        self.layer2 = nn.Sequential(
            nn.Linear(2, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        return x
```

> https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html#torch.nn.Sequential
>
> https://pytorch.org/docs/stable/nn.functional.html

NeuralNetwork의 인스턴스(instance)를 생성하고 이를 device로 이동

```
device = "cuda" if torch.cuda.is_available() else "cpu"
model = CustomModel().to(device)
```

loss function

```
criterion = nn.BCELoss().to(device)
```

> https://pytorch.org/docs/stable/nn.html#loss-functions

optimizer

```
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

> https://pytorch.org/docs/stable/optim.html

모델 학습, 평가

```
def train(model, train_dataloader, optimizer, criterion, device, epoch):
    model.train()
    cost = 0
    for x, y in train_dataloader:
        x = x.to(device)
        y = y.to(device)

        output = model(x) # 바로 forward가 실행
        loss = criterion(output, y)

        optimizer.zero_grad() # 이전 loss gradient 삭제
        loss.backward()
        optimizer.step() # 신경망 가중치 업데이트

        cost += loss

    cost = cost / len(train_dataloader)

    if (epoch + 1) % 1000 == 0:
        print(f"Epoch : {epoch+1:4d}, Cost : {cost:.3f}")
```

```
def evaluate(model, test_dataloader, criterion, device):
    model.eval()
    cost = 0

    with torch.no_grad():
        for x, y in test_dataloader:
            x = x.to(device)
            y = y.to(device)

            output = model(x)
            loss = criterion(output, y)
            cost += loss.item()

    cost /= len(test_dataloader)
    return cost
```

```
for epoch in range(10000):
    train(model, train_dataloader, optimizer, criterion, device, epoch)

    if (epoch + 1) % 1000 == 0:
        test_loss = evaluate(model, test_dataloader, criterion, device)
        print("\n[EPOCH: {}], Test cost: {:.4f}\n".format(epoch + 1, test_loss))
```

모델 저장

모델 저장

```
torch.save(model, 'model.pth')
```

모델 불러오기

```
model = torch.load('model.pth')
```

참조

> https://pytorch.org/docs/stable/nn.html
>
> https://pytorch.org/docs/stable/generated/torch.nn.Module.html#torch.nn.Module
>
> https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html#torch.nn.Sequential
>
> https://pytorch.org/docs/stable/nn.functional.html
>
> https://pytorch.org/docs/stable/nn.html#loss-functions

> https://pytorch.org/docs/stable/optim.html