PyTorch 기초 실습

주재걸 교수님 연구실
DAVIAN Lab.

강경필

PyTorch

- 딥러닝 라이브러리 (Python)
- GPU 연산을 통한 빠른 학습
- **코드가 간결하여 가독성이 좋고 디버깅이 쉬움**
- **Dynamic Network 구조이기 때문에**
  모델의 변경이 자유롭고 중간 결과를 확인하기 쉬움
- 파이썬, C++ 등의 언어와 연동이 편함
- **Documentation**이 잘 정리되어 있고,
  활발한 커뮤니티 교류 (Forum, PyTorch KR 등)

Tensor

In [5]:

```
import torch

x= torch.tensor([[1,2,3],[4,5,6]])
y= torch.tensor([[7,8,9], [10,11,12]])

f= 2*x + y
print(f)
```

```
tensor([[ 9,  12,  15],
        [ 18,  21,  24]])
```

- 기본적으로 Numpy Array와 비슷함
- Gradient 정보를 저장할 수 있음
  (requires_grad=True인 경우)
- torch.FloatTensor
  torch.LongTensor
  torch.from_numpy
- view, flatten, squeeze, unsqueeze, transpose 등의 함수 제공

DAVIAN
Data and Visual Analytics Lab

torch.cuda

```
[12] print(torch.cuda.is_available())
     print(torch.cuda.device_count())
```

```
True
1
```

```
x = torch.cuda.FloatTensor([1,2,3])
print(x)
```

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

```
[9]  device = torch.device('cuda')
     y = torch.FloatTensor([2,4,5])
     y = y.to(device)
     print(y)
```

```
tensor([2., 4., 5.], device='cuda:0')
```

- **CUDA 및 GPU 관련 함수 제공**

- torch.cuda.is_available()
  - GPU가 사용 가능한 상태인지 체크
- torch.cuda.device_count()
  - 사용가능한 GPU 개수 체크
- torch.cuda.FloatTensor 등
  - GPU 메모리에 올라가 있는 Tensor 생성
- torch.device('cuda') or torch.device('cpu')
  - 어떤 디바이스를 사용할지 선택
- y.to(device): 해당 텐서를 해당 디바이스의 메모리로 복사

DAVIAN
Data and Visual Analytics Lab

torch.utils.data

TORCH.UTILS.DATA

CLASS `torch.utils.data.Dataset`

An abstract class representing a Dataset.

All other datasets should subclass it. All subclasses should override
and `__getitem__`, supporting integer indexing in range from 0 to le

CLASS `torch.utils.data.TensorDataset(*tensors)`

Dataset wrapping tensors.

Each sample will be retrieved by indexing tensors along the first dir

Parameters

***tensors** (*Tensor*) – tensors that have the same size of the

CLASS `torch.utils.data.ConcatDataset(datasets)`

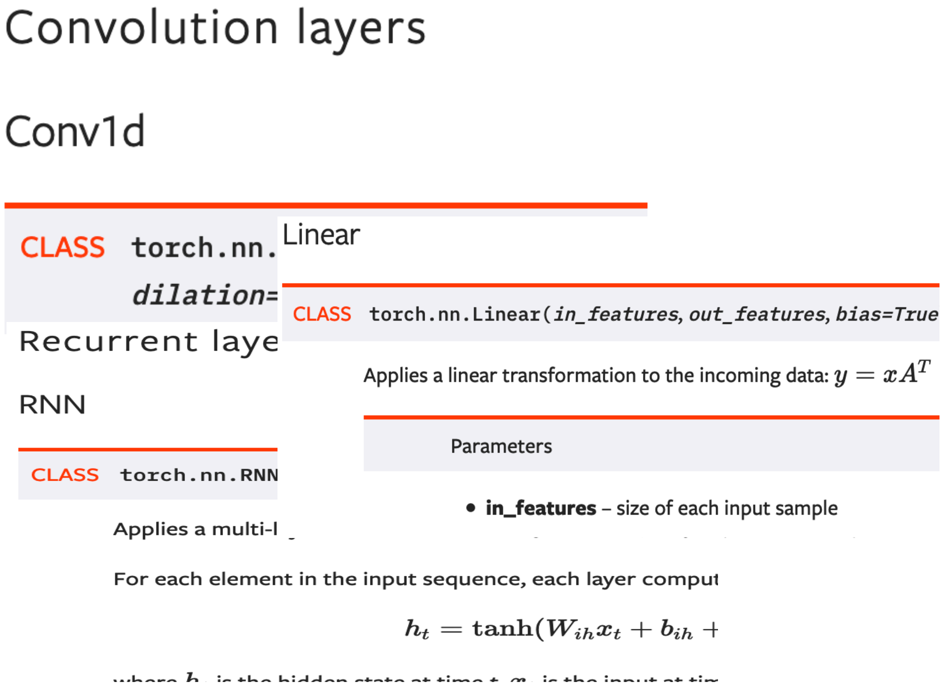
- **Dataset 관련 처리 모듈 제공**

- torch.utils.data.DataSet
  갖고 있는 데이터셋을 다루기 편하게 관리해주는 인터페이스

- torch.utils.data.DataLoader
  해당 Dataset을 멀티프로세싱, batch processing, iterating 등 다양한 기능을 추가 제공해주는 클래스

DAVIAN
Data and Visual Analytics Lab

torch.nn



- **Neural network layer 관련 클레스들 제공**
- Containers
  nn.Module, nn.Sequential, nn.ModuleList 등
- Linear: nn.Linear
- CNN: nn.Conv2d, nn.MaxPool2d 등
- RNN: nn.RNN, nn.GRU, nn.LSTM
- Regularization/Normalization:
  nn.Dropout, nn.BatchNorm2d 등

- **제공하는 기능들이 많으므로 꼭 Documentation 살펴볼것!**

DAVIAN
Data and Visual Analytics Lab

torch.nn.functional

TORCH.NN.FUNCTIONAL

Convolution functions

conv1d

```
torch.nn.functional.conv1d(input, weight, bias=None, stride=1,
groups=1, padding_mode='zeros') → Tensor
```

Applies a 1D convolution over an input signal composed of several in

See `Conv1d` for details and output shape.

- torch.nn과 유사하지만 **클래스가 아닌 함수로 제공**
- 각종 Non-linear Activation function들 제공 (relu, tanh, sigmoid, softmax 등)
- Loss 함수 제공
- **제공하는 기능들이 많으므로 꼭 Documentation 살펴볼것!**

Data and Visual Analytics Lab

torch.optim

CLASS `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.`
`amsgrad=False)`

Implements Adam algorithm.

It has been proposed in Adam: A Method for Stochastic Opt

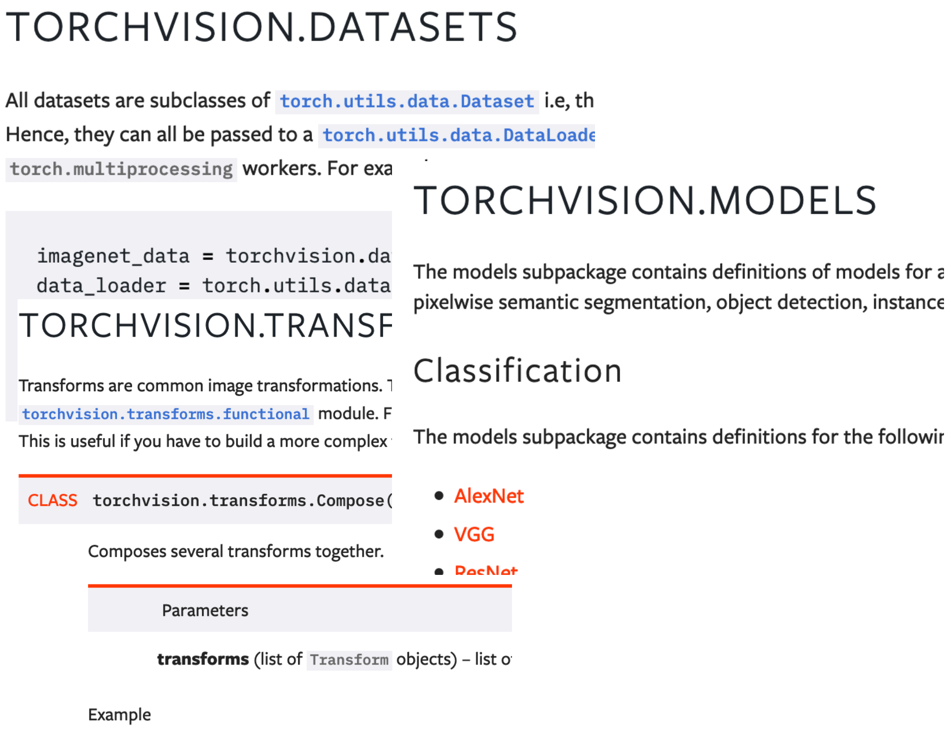
Parameters

- **params** (*iterable*) – iterable of parameters to
- **lr** (*float*, *optional*) – learning rate (default: 1e-
- **betas** (*Tuple*[*float*, *float*], *optional*) – coefficie
  and its square (default: (0.9, 0.999))

- **최적화 알고리즘 클래스 제공**
- SGD, Adam, AdaGrad 등 다양한 optimizer
- 많은 경우 Adam이 좋은 성능을 보임
- weight_decay는 l2 regularization 관련 Hyperparameter
- Learning rate는 중요한 Hyperparameter!

DAVIAN
Data and Visual Analytics Lab

torchvision



- **Vision 관련 기능들 제공**
- torchvision.dataset

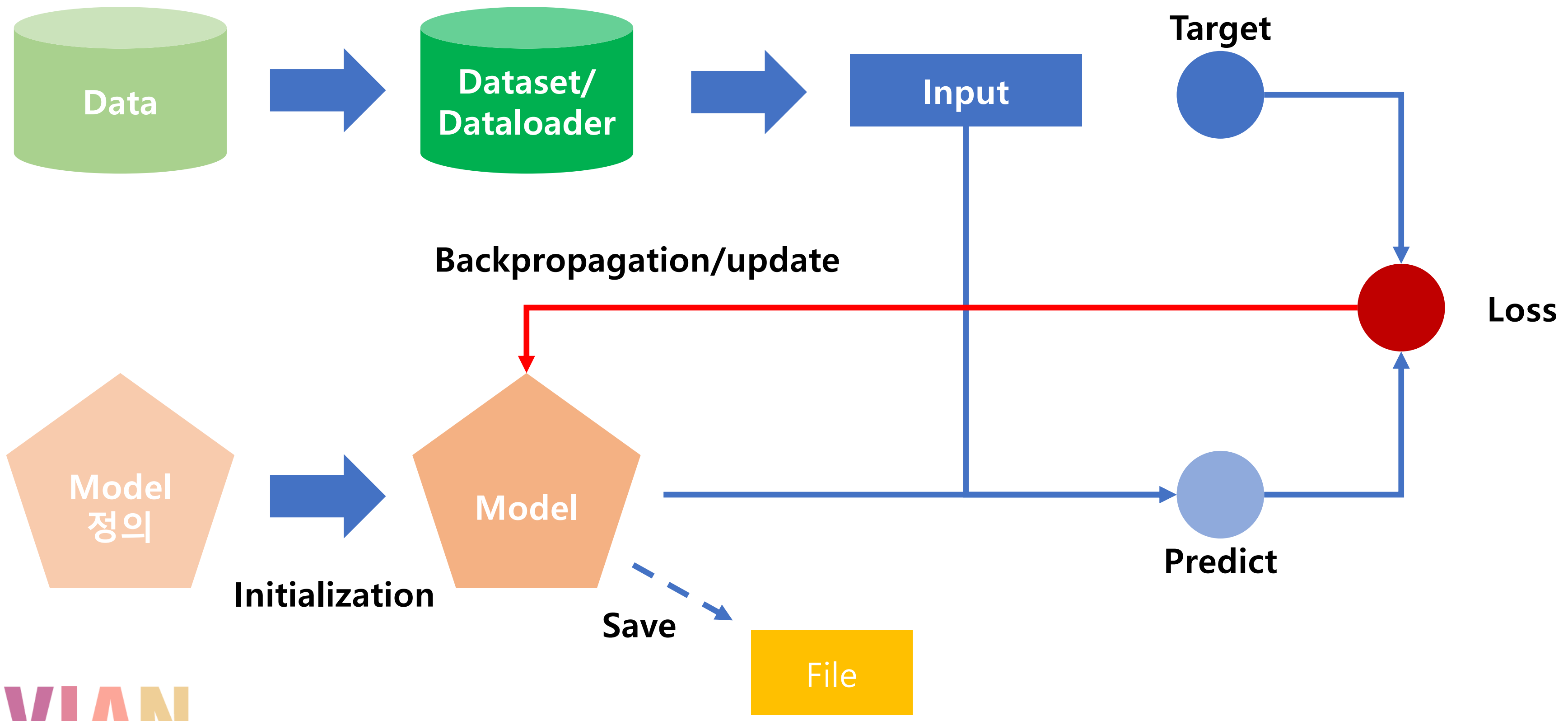
  MNIST, CIFAR10 등의 데이터셋 제공
- torchvision.transforms

  이미지 관련 전처리 기능 함수들 제공
- torchvision.models

  VGGNet, ResNet, Inception 등 미리 학습된 모델 제공

Data and Visual Analytics Lab

Training process



Model

모델명

최상위 부모 클래스

```
class Model(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(Model, self).__init__()

        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        self.layer1 = nn.Linear(input_dim, hidden_dim)
        self.layer2 = nn.Linear(hidden_dim, hidden_dim)
        self.layer3 = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        out = F.relu(self.layer1(x))
        out = F.relu(self.layer2(out))
        out = self.layer3(out)
        return out
```

파이썬 상속 문법

필수적으로
구현되어야 하는
함수

Data and Visual Analytics Lab

Model

```python
class Model(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(Model, self).__init__()

        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        self.layer1 = nn.Linear(input_dim, hidden_dim)
        self.layer2 = nn.Linear(hidden_dim, hidden_dim)
        self.layer3 = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        out = F.relu(self.layer1(x))
        out = F.relu(self.layer2(out))
        out = self.layer3(out)
        return out
```

네트워크 Layer 정의

모델의 입력값을 처리

모델의 예측 결과

Data and Visual Analytics Lab

Training Model

모델 생성 및 GPU 메모리에 복사

Opimizer 및 loss 함수 정의

학습 모드

Batch를 GPU 메모리에 복사

인풋에 대해 모델 예측

이전 gradient 값 제거

Loss 계산 후 모델 업데이트

```
device = torch.device("cuda")

model = Model(8, 128)
model = model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.MSELoss()

epochs = 10

model.train()
for e in range(epochs):
    for i, (batch_X, batch_y) in enumerate(train_loader):
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)

        predict = model(batch_X)

        optimizer.zero_grad()
        loss = criterion(predict, batch_y)
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            loss = loss.item()
            loss
            print(f"{e}epochs, {i} iters - {loss}")
```

DAVIAN
Data and Visual Analytics Lab

Testing Model

gradient 계산 안함! & 테스트 모드

L1 loss 계산

```
total_loss = []
test_num = 0

with torch.no_grad():
    model.eval()
    for batch_X, batch_y in test_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        predict = model(batch_X)
        predict = predict*(max_ - min_) + min_
        loss = F.l1_loss(predict, batch_y)

        batch_size = batch_y.size(0)
        test_num += batch_size
        total_loss.append(loss.item()*batch_size)

total_loss = np.sum(total_loss)/test_num
print(f"{e}epochs - {total_loss}")
```

Data and Visual Analytics Lab

Load/Save the Model

model의 각 정보가 dictionary 객체로 저장됨!

```
torch.save(the_model.state_dict(), PATH)
```

```
the_model = TheModelClass(*args, **kwargs)
the_model.load_state_dict(torch.load(PATH))
```

주의!
모델이 미리 생성이 된 뒤, load가 되어야 함!

* *https://discuss.pytorch.org/t/saving-and-loading-a-model-in-pytorch/2610*

DAVIAN Data and Visual Analytics Lab

Exercise - Predict California housing values

6.3.7. California Housing dataset

Data Set Characteristics:

Number of Instances:	
20640	
Number of Attributes:	
8 numeric, predictive attributes and the target	
Attribute Information:	
• MedInc median income in block • HouseAge median house age in block • AveRooms average number of rooms • AveBedrms average number of bedrooms • Population block population • AveOccup average house occupancy • Latitude house block latitude • Longitude house block longitude	
Missing Attribute Values:	
None	

This dataset was obtained from the StatLib repository. http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

* https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html

DAVIAN
Data and Visual Analytics Lab

감사합니다

Any Questions?

rudvlf0413@naver.com