컴퓨터비전

Computer Vision

딥러닝과 파이토치

본 강의자료는 한빛아카데미 〈파이썬으로 만드는 인공지능〉을 내용을 기반으로 제작 및 보완되었음을 알립니다.

PREVIEW

- 이번장 에서는 앞장에서 다룬 얕은 신경망을 깊게 만든 딥러닝 모델을 공부함
- 영상 인식기, 음성 인식기, 언어 번역기와 같은 최첨단 인공지능 제품은 대부분 딥러닝 기술로 만듦
- 딥러닝을 프로그래밍 하는데 사용하는 소프트웨어로는 구글의 텐서플로우(tensorflow), 페이스북의 파이토치(pytorch), 몬트리올 대학교에서 만든 씨아노(Theano), UC 버클리에서 만든 카페(caffe) 등이 있음
- 가장 널리 쓰이는 것은 텐서플로우와 파이토치이며, 파이썬 라이브러리 형태로 제공됨

그림 5-1 딥러닝 프로그래밍에 널리 쓰이는 텐서플로, 파이토치, 케라스의 로고

5.1 딥러닝의 등장

- 깊은 신경망에 대한 아이디어는 사실 다층 퍼셉트론(MLP)이 등장해 혁신을 이루던 1980년대에 이미 나왔었음. 다층 퍼셉트론에 단지 은닉층을 여러 개 추가하면 깊은 신경망이 되기 때문에 누구나 쉽게 생각할 수 있었음. 하지만 은닉층을 많이 추가해 신경망을 깊게 만들면 제대로 학습되지 않는 문제가 있었음

- 깊은 신경망 학습에 번번이 실패하는 이유
  - 1) 오류 역전파 알고리즘은 출력층에서 시작해 입력층 방향으로 진행하면서 그레디언트 계산하고 가중치를 갱신하는데, 여러 층을 거치면서 그레디언트 값이 점점 작아져 입력층에 가까워지면 변화가 거의 없는 그레디언트 소멸 문제gradient vanishing 가 발생함
  - 2) 게다가 훈련 집합의 크기는 작은 상태로 머물러 있는데 추정할 매개변수는 크게 늘어 과잉 적합over-fitting에 빠질 위험이 더욱 커짐
  - 3) 학습이 어려운 또 다른 이유는 과다한 계산 시간임. 예전에는 병렬 처리를 하려면 값비싼 슈퍼컴퓨터를 사용하는 수밖에 없었음

5.1.1 딥러닝의 기술 혁신

- 딥러닝의 기술 혁신 요인
  - 사실 딥러닝에 새로 창안된 이론이나 원리는 별로 없음. 신경망의 기본 구조와 동작, 학습 알고리즘의 기본 원리는 거의 그대로임

  - **저렴한 GPU 등장**
    - 성능이 뛰어나고 가격도 저렴한 GPU가 등장하면서 대학 실험실에서도 손쉽게 병렬 처리를 할 수 있게 되었고, 학습 시간이 10~100배 단축되어 성능 실험을 다채롭게 할 수 있게 되었음
    - 예들 들어, 딥러닝 모델 하나를 학습하는데 이틀이 걸린다면 200개의 서로 다른 하이퍼 매개변수 조합 중 최적을 선택하는 작업은 1년 이상 걸림. 하지만 GPU를 사용하면 일주일 이내에 마칠 수 있음

5.1.1 딥러닝의 기술 혁신

- 딥러닝의 기술 혁신 요인
  - **인터넷이 영향으로 학습 데이터가 크게 증가**
    - 예를 들어, ImageNet은 인터넷에서 수집한 1400만 장 정도의 자연 영상을 카테고리별로 분류해 제공함
    - 게다가, 데이터에 가우시안 잡음을 섞거나 영상을 조금 이동하거나 회전해 데이터를 인우적으로 수십~수백 배 증가시키는 기법이 개발되었음

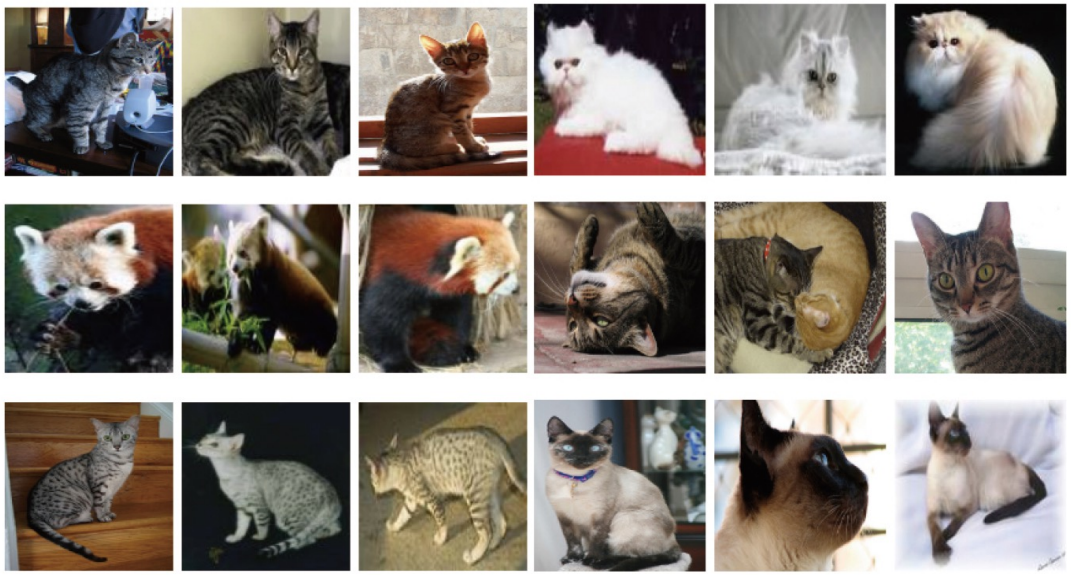


그림 5-2 자연 영상의 심한 변화 예(ImageNet의 고양이 부류 사진)

5.1.1 딥러닝의 기술 혁신

- 딥러닝의 기술 혁신 요인
  - **학습을 효과적으로 실행할 수 있는 다양한 알고리즘 개발**
    - 계산은 단순하고 성능은 더 좋은 활성 함수 ReLU를 발견함으로써 딥러닝의 성능이 향상되었고 그레디언트 소멸 문제가 크게 완화도었음
    - 학습에 효과적인 다양한 규제 기법regularization 이 개발되었음
      - 가중치를 작은 값으로 유지하는 가중치 축소weight decay 기법
      - 임의로 일정 비율의 노드를 선택해 불능으로 놓고 학습하는 드롭아웃dropout 기법
      - 조기 멈춤, 데이터 확대, 앙상블 등의 다양한 규제 기법이 개발되었음
      - 다양한 손실 함수와 옵티마이저 개발도 성능 향상에 크게 기여하였음

5.1.1 딥러닝의 기술 혁신

- 딥러닝의 기술 혁신 사례 (학술적인 측면)
  - **컨볼루션 신경망(CNN)이 딥러닝의 가능성을 열었음**
    - 컨볼루션 신경망은 특징 추출에 적합한 작은 크기의 컨볼루션 마스크를 사용하기 때문에 완전연결 구조인 다층 퍼셉트론 보다 매개변수가 훨씬 적지만 훨씬 우수한 특징을 추출함
    - 이런 이유로 컨볼루션 신경망에서 우수한 성능이 가장 먼저 입증되었음
    - 1990년대에 뉴욕 대학교의 리쿤 교수는 CNN으로 필기 숫자 인식에서 획기적인 성능 향상을 얻었고, 이 기술혁신으로 필기 숫자 인식이 수표 인식에서 실용화 수준에 도달하게 됨(1998)
    - 이후 CNN에서 개발된 여러 기법은 딥러닝 전반에 큰 영향을 미침

5.1.1 딥러닝의 기술 혁신

- 딥러닝의 기술 혁신 사례 (학술적인 측면)
  - **컨볼루션 신경망(CNN)이 딥러닝의 가능성을 열었음**
    - 르쿤이 필기 숫자에서 CNN의 가능성을 보였음에도 불구하고 자연 영상에 대한 시도는 소극적이 었음
    - 2012년 ILSVRC 에서 CNN을 사용한 AlexNet이 오류율 15.3%라는 경이로운 성능으로 우승함. 전년도에 고전적인 기계 학습을 사용한 팀이 오류율 25.8%로 우승한 사실을 감안하면 CNN의 가능성을 보여주기에 충분함
    - 이때부터 컴퓨터 비전 연구는 고전적인 기법에서 딥러닝으로 패러다임 전환이 이루어짐

  - **딥러닝은 음성인식 분야의 혁신을 가져옴**
    - 딥러닝 이전에는 음성 인식을 주로 통계적 기법과 HMM[Hidden Markov Model]을 이용해 구현하였음
    - 2009년경에 토론토 대학교의 힌튼 교수 연구팀은 음성 인식에 딥러닝을 적용하기 시작함
    - 힌튼 교수는 안드로이드 운영체제에 들어가는 음성 인식 소프트웨어의 단어 인식 오류율을 단숨에 25%나 줄였음

5.1.2 딥러닝 소프트웨어

- 여러 대학과 기업은 자체적으로 딥러닝 소프트웨어를 개발해 내부 연구 그룹끼리 공유하기 시작함
- 공유를 통해 딥러닝 소프트웨어는 점점 개선되어 큰 규모로 발전했고, 개발자들은 누구나 사용할 수 있도록 라이브러리 형태로 제작해 경쟁적으로 공개함
- 딥러닝 소프트웨어 자체는 대부분 C와 C++ 로 개발하지만, 소프트웨어를 불러 쓰는 인터페이스 언어로는 파이썬과 가은 스크립트 언어를 채택함

5.1.2 딥러닝 소프트웨어

표 5-1 딥러닝 소프트웨어

이름	개발 그룹	최초 공개일	작성 언어	인터페이스 언어	전이학습 지원	철저한 관리
씨아노(Theano)	몬트리올 대학교	2007년	파이썬	파이썬	O	X
카페(Caffe)	UC버클리	2013년	C++	파이썬, 매트랩, C++	O	X
텐서플로(TensorFlow)	구글 브레인	2015년	C++, 파이썬, CUDA	파이썬, C++, 자바, 자바스크립트, R, Julia, Swift, Go	O	O
케라스(Keras)	프랑소와 숄레(François Chollet)	2015년	파이썬	파이썬, R	O	O
파이토치(PyTorch)	페이스북	2016년	C++, 파이썬, CUDA	파이썬, C++	O	O

5.1.2 딥러닝 소프트웨어

- 구글 트랜드를 통해 텐서플로우와 파이토치의 최근 영향력을 비교한 그래프를 살펴보면, 2020년 12월 기준으로 둘의 영향력은 막상막하로 보임

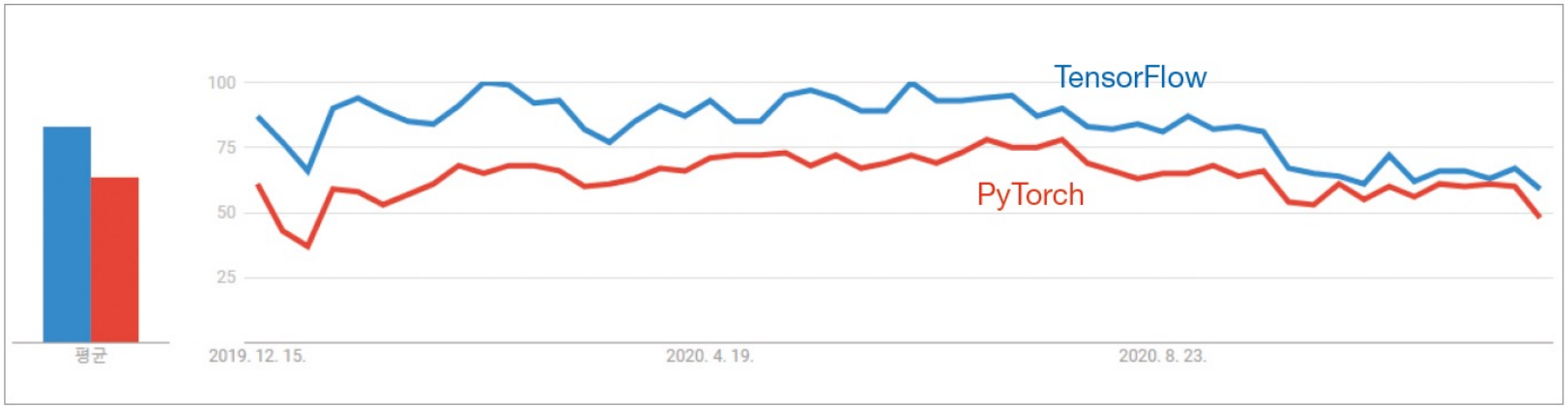


그림 5-3 구글 트렌드를 통해 비교한 텐서플로와 파이토치의 영향력

5.2 파이토치 개념 익히기

- 본 강의에서는 인공지능을 구현하기 위해 파이토치 라이브러리를 사용함
- 본 강의의 실험을 위해 파이토치를 자신의 컴퓨터에 직접 설치할 수 있으나, 본 강의에서는 환경 설정이 완료된 〈캐글 노트북〉을 사용할 예정임
- 즉, 개인 컴퓨터에 파이토치를 직접 설치하는 방법은 본 강의에서 다루지 않음

5.2 파이토치 개념 익히기

5.2.1 파이토치와 넘파이이 호환

[프로그램 5-1] 파이토치 버전과 동작 확인

```
import torch

print(torch.__version__) # pytorch 버전 확인
a=torch.rand(2,3)        # (2 by 3의 크기를 가지는 Random 텐서 생성)
print(a)
print(type(a))
```

03행 torch version 확인

04행 0~1사이의 값을 가지는 난수를 원하는 크기에 해당하는 Tensor 반환

```
2.0.0
tensor([[0.5809, 0.6648, 0.1949],
        [0.0583, 0.1108, 0.4529]])
<class 'torch.Tensor'>
```

5.2 파이토치 개념 익히기

5.2.1 파이토치와 넘파이이 호환

[프로그램 5-2] pytorch와 numpy 의 호환

```
import torch
import numpy as np

t=torch.rand(2,3)
print("pytorch로 생성한 텐서 : \n {}".format(t))

n=np.random.uniform(low=0.0, high=1.0, size=(2,3))
print("\nnumpy로 생성한 ndarray : \n {}".format(n))

res=t+n
print("\n덧셈 결과 : \n {}".format(res))  # pytorch도 numpy와 호환되기 때문에 덧셈이 가능
```

04행 0~1사이의 값을 가지는 난수를 원하는 크기에 해당하는 Tensor 반환

07행 low~high사이의 랜덤 난수를 생성하여 원하는 사이즈의 ndarray 반환

```
pytorch로 생성한 텐서 :
 tensor([[0.1852, 0.6459, 0.8372],
        [0.9553, 0.1209, 0.7753]])

numpy로 생성한 ndarray :
 [[0.64764965 0.82919971 0.27473929]
 [0.02202838 0.02749204 0.27035916]]

덧셈 결과 :
 tensor([[0.8328, 1.4751, 1.1120],
        [0.9773, 0.1484, 1.0457]], dtype=torch.float64)
```

5.2 파이토치 개념 익히기

5.2.2 텐서 이해하기

[프로그램 5-3] pytorch가 제공하는 데이터셋의 텐서 구조 확인하기

```
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor

# MNIST 읽고 텐서 구조 확인 (학습 데이터)
MNIST_training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
# MNIST 읽고 텐서 구조 확인 (평가 데이터)
MNIST_test_data = datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)

print("MNIST 학습 이미지 데이터 텐서 모양 : {}".format(MNIST_training_data[0][0].shape))
print("MNIST 학습 라벨 데이터 : {}".format(MNIST_training_data[0][1]))

print("MNIST 평가 이미지 데이터 텐서 모양 : {}".format(MNIST_test_data[0][0].shape))
print("MNIST 평가 라벨 데이터 : {}".format(MNIST_test_data[0][1]))
```

06행 root : data를 다운 받을 root 경로

07행 train : Train data를 받을지 말지 (True/False)

08행 download : 직접 다운로드 하여 root 경로에 저장할지 말지 (True/False)

09행 transform : 데이터를 불러올 때 어떤 변환을 적용할 지

5.2 파이토치 개념 익히기

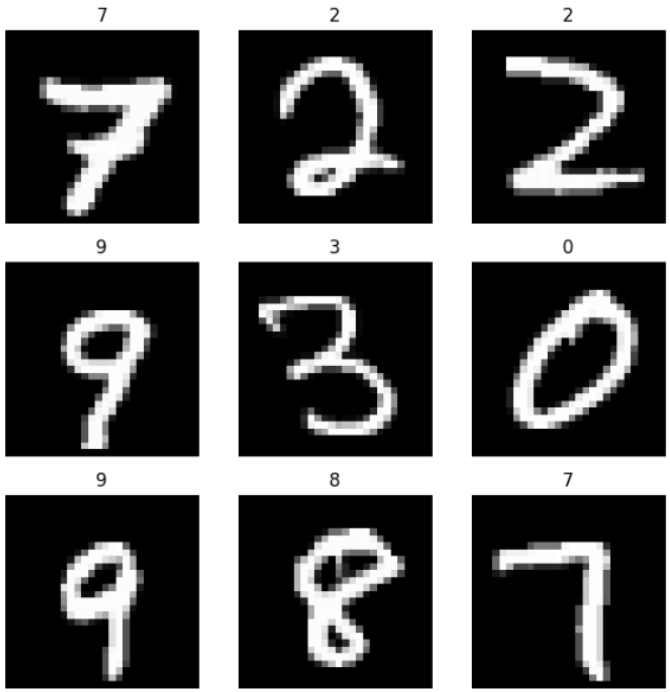
5.2.2 텐서 이해하기

[프로그램 5-3] pytorch가 제공하는 데이터셋의 텐서 구조 확인하기

```
import matplotlib.pyplot as plt

labels_map = {
    0: "0",
    1: "1",
    2: "2",
    3: "3",
    4: "4",
    5: "5",
    6: "6",
    7: "7",
    8: "8",
    9: "9",
}

figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(MNIST_training_data), size=(1,)).item()
    img, label = MNIST_training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```



5.3 파이토치 프로그래밍 기초

5.3.1 sklearn의 표현력 한계

- 4장에서는 sklearn 라이브러리를 활용해 기계 학습을 구현하였음
- MLP를 떠올려보면 sklearn 은 딥러닝을 구현하기에는 표현력의 한계가 있음
- 딥러닝 모델은 MLP 처럼 단순하지 않으며, 컨볼루션 연산을 사용하는 “컨볼루션 신경망”으로 확장되고, 은닉 노드끼리 에지로 연결되는 “순환 신경망”으로 확장되며, 신경망 구조를 벗어난 “강화학습”으로 확장됨
- 〈텐서플로우〉나 〈파이토치〉 같은 라이브러리는 이런 복잡도를 지원할 수 있도록 바닥부터 새롭게 설계한 딥러닝 전용 소프트웨어 이며, 이들은 명령어로 층을 쌓아가는 방식으로 프로그래밍을 진행함
- 딥러닝 프로그램에도 디자인 패턴이 있고, 디자인 패턴을 따른 좋은 예제 코드가 많으니 이들을 잘 관찰하고 기억하면 어렵지 않게 딥러닝 프로그래밍에 익숙해질 수 있음

sklearn library ➔ pytorch library

5.3 파이토치 프로그래밍 기초

5.3.2 파이토치를 이용한 퍼셉트론 프로그래밍

[프로그램 5-4] 파이토치 프로그래밍 : 학습된 퍼셉트론 동작

```
import torch

x = torch.tensor([[0.0,0.0],[0.0,1.0],[1.0,0.0],[1.0,1.0]])
y = torch.tensor([[-1],[1],[1],[1]])

w = torch.autograd.Variable(torch.tensor([[1.0],[1.0]]),requires_grad=True)
b = torch.autograd.Variable(torch.tensor([-0.5]),requires_grad=True)

pred = torch.sign(x.matmul(w)+b)

print("OR 프로그래밍 정답 값 : \n {}".format(y))
print("OR 프로그래밍 예측 값 : \n {}".format(pred))
```

```
OR 프로그래밍 정답 값 :
 tensor([[-1],
        [ 1],
        [ 1],
        [ 1]])
OR 프로그래밍 예측 값 :
 tensor([[-1.],
        [ 1.],
        [ 1.],
        [ 1.]], grad_fn=<SignBackward0>)
```

06행 gradient 추적을 허용하는 가중치 행렬 W 정의

07행 gradient 추적을 허용하는 편향 b 정의

09행 wx+b 형태로 계산한 다음 결과의 부호만을 반환

5.3 파이토치 프로그래밍 기초

5.3.2 파이토치를 이용한 퍼셉트론 프로그래밍

[프로그램 5-5] 파이토치 프로그래밍 : 퍼셉트론 학습

```
01 import torch
02
03 def forward(w,b,x):
04     x = x.matmul(w)+b
05     x = torch.tanh(x)
06     return w,b, x
07
08 def loss_mse(pred,y):
09     loss = torch.mean((y-pred)**2)
10     return loss
11
12 if __name__ == '__main__':
13     x = torch.tensor([[0.0,0.0],[0.0,1.0],[1.0,0.0],[1.0,1.0]])
14     y = torch.tensor([[-1],[1],[1],[1]])
15
16     w = torch.autograd.Variable(torch.randn(2,1),requires_grad=True)
17     b = torch.autograd.Variable(torch.zeros(1),requires_grad=True)
18
19     optimizer = torch.optim.SGD([w,b], lr=0.1)
20
21     for iter in range(500):
22         w,b,pred = forward(w,b,x)
23         cost = loss_mse(pred,y)
24
25         optimizer.zero_grad()
26         cost.backward()
27         optimizer.step()
28
29     w,b,pred = forward(w,b,x)
30     print(torch.sign(pred))
```

```
tensor([[-1.],
        [ 1.],
        [ 1.],
        [ 1.]], grad_fn=<SignBackward0>)
```

02행 wx+b 형태로 계산하고 tanh 함수를 적용하여 순전파 연산 진행

07행 예측값과 정답값의 제곱 오차를 계산하는 MSE 손실함수

15행 gradient 추적을 허용하는 가중치,편향 정의

18행 확률적 경사하강법(SGD)에 해당하는 옵티마이저 선언(W,b를 최적화)

24행 optimizer.zero_grad() : 매 iteration 마다 gradient 초기화

25행 cost.backward() : 손실 함수로부터 역전파(backpropagation) 수행

26행 optimizer.step() : 역전파로부터 계산된 gradient를 토대로 W,b 업데이트

5.3 파이토치 프로그래밍 기초

5.3.2 파이토치를 이용한 퍼셉트론 프로그래밍

[프로그램 5-6] 파이토치 프로그래밍 : 퍼셉트론 학습 (추상화)

```
01 class Perceptron(torch.nn.Module):
02     def __init__(self,input_dim,output_dim):
03         super(Perceptron, self).__init__()
04
05         self.perceptron = torch.nn.Linear(in_features=input_dim,
06                                           out_features=output_dim,bias=True)
07         self.activation = torch.nn.Tanh()
08
09         self.model = torch.nn.Sequential(self.perceptron,self.activation)
10
11     def forward(self, x):
12         return self.model(x)
13
14 def loss_mse(pred,y):
15     loss = torch.mean((y-pred)**2)
16     return loss
17
18 if __name__ == '__main__':
19     x = torch.tensor([[0.0,0.0],[0.0,1.0],[1.0,0.0],[1.0,1.0]])
20     y = torch.tensor([[-1],[1],[1],[1]])
21
22     Perceptron = Perceptron(input_dim=2, output_dim=1)
23     optimizer = torch.optim.SGD(Perceptron.parameters(), lr=0.1)
24
25     for iter in range(500):
26         pred = Perceptron(x)
27         cost = loss_mse(pred,y)
28         optimizer.zero_grad()
29         cost.backward()
30         optimizer.step()
31
32     pred=Perceptron(x)
33     print(torch.sign(pred))
```

```
tensor([[-1.],
        [ 1.],
        [ 1.],
        [ 1.]], grad_fn=<SignBackward0>)
```

01행 torch.nn.Module을 상속 받는 Class 정의

05행 torch.nn.Linear를 이용하여 선형 모델(퍼셉트론)을 정의

07행 활성화 함수는 torch.nn.Tanh()를 사용

09행 torch.nn.Sequential을 사용하여 모듈을 순차적으로 정의

11행 순전파를 다음과 같이 정의 (torch.nn.Module을 상속 받았기 때문) Perceptron(x) 처럼 사용가능

28행 optimizer.zero_grad() : 매 iteration 마다 gradient 초기화

29행 cost.backward() : 손실 함수로부터 역전파(backpropagation) 수행

30행 optimizer.step() : 역전파로부터 계산된 gradient를 토대로 W,b 업데이트

5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.1 MNIST 인식

[프로그램 5-7(a)] 파이토치 프로그래밍: 다층 퍼셉트론으로 MNIST 인식

```
import torch
import tqdm

if __name__ == '__main__':

    history = dict()
    history['train_acc'] = list()
    history['test_acc'] = list()
    history['train_loss'] = list()
    history['test_loss'] = list()

    # MNIST 읽고 텐서 구조 확인 (학습 데이터)
    training_data = datasets.MNIST(
      root="data",
      train=True,
      download=True,
      transform=ToTensor()
    )
    # MNIST 읽고 텐서 구조 확인 (평가 데이터)
    test_data = datasets.MNIST(
      root="data",
      train=False,
      download=True,
      transform=ToTensor()
    )
```

➔ 학습 중간에 정확도와 손실을 기록할 딕셔너리 정의

➔ MNIST 학습 데이터 정의

➔ MNIST 평가 데이터 정의

5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.1 MNIST 인식

[프로그램 5-7(a)] 파이토치 프로그래밍: 다층 퍼셉트론으로 MNIST 인식

```
train_dataloader = torch.utils.data.DataLoader(training_data,batch_size=128)
test_dataloader = torch.utils.data.DataLoader(test_data,batch_size=128,shuffle=False)

MLP = MultiLayerPerceptron(input_dim=784, hidden_dim=1024, output_dim=10)
MLP = MLP.cuda()
optimizer = torch.optim.Adam(MLP.parameters(), lr=0.001)

for iter in tqdm.tqdm(range(30)):
    MLP, history = train_epoch(train_dataloader, optimizer, MLP, loss_mse, history)
    with torch.no_grad():
        MLP, history = test_epoch(test_dataloader, optimizer, MLP, loss_mse, history)
```

➔ 배치 단위 학습을 위해 MNIST 학습 데이터 로더 정의

➔ 배치 단위 학습을 위해 MNIST 평가 데이터 로더 정의

➔ 신경망 최적화를 위해 Adam 사용

```
import torch

class MultiLayerPerceptron(torch.nn.Module):
    def __init__(self,input_dim, hidden_dim, output_dim):
        super(MultiLayerPerceptron, self).__init__()

        self.input_layer = torch.nn.Linear(in_features=input_dim,
                                           out_features=hidden_dim,bias=True)
        self.hidden_layer = torch.nn.Linear(in_features=hidden_dim,
                                            out_features=output_dim,bias=True)
        self.activation = torch.nn.Tanh()

        self.model =  torch.nn.Sequential(self.input_layer,self.activation,
                                          self.hidden_layer, self.activation)

    def forward(self, x):
        return self.model(x)
```

➔ torch.nn.Module을 상속 받는 Class 정의

➔ torch.nn.Linear를 이용하여 선형 모델(퍼셉트론)을 정의

➔ 활성화 함수는 torch.nn.Tanh()를 사용

➔ torch.nn.Sequential을 사용하여 모듈을 순차적으로 정의

➔ 순전파를 다음과 같이 정의 (torch.nn.Module을 상속 받았기 때문) Perceptron(x) 처럼 사용가능

5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.1 MNIST 인식

[프로그램 5-7(a)] 파이토치 프로그래밍: 다층 퍼셉트론으로 MNIST 인식

```
def train_epoch(dataloader, optimizer, model, loss_fn, log_dict):
    acc=0
    loss=0
    model.train()
    for X, y in dataloader:
        X = X.view(-1,784).cuda()
        y = y.cuda()

        pred = model(X)
        cost = loss_fn(pred, y)

        optimizer.zero_grad()
        cost.backward()
        optimizer.step()

        acc += torch.sum(torch.argmax(pred,dim=1)==y).item()
        loss += cost.item()

    log_dict['train_acc'].append(acc/len(dataloader.dataset))
    log_dict['train_loss'].append(loss)

    return model, log_dict
```

➔ model.train() : 모델을 학습 모드로 변경 (batchnorm이나 dropout 관련)

➔ 28x28 크기의 데이터(X)를 784x1 데이터로 reshape & GPU 메모리에 올려줌

➔ 정답 데이터(y)도 GPU 메모리에 올려줌

➔ optimizer.zero_grad() : 매 iteration 마다 gradient 초기화

➔ cost.backward() : 손실 함수로부터 역전파(backpropagation) 수행

➔ optimizer.step() : 역전파로부터 계산된 gradient를 토대로 W,b 업데이트

➔ 예측값과 정답값이 동일한 개수를 카운트하여 저장

5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.1 MNIST 인식

[프로그램 5-7(a)] 파이토치 프로그래밍: 다층 퍼셉트론으로 MNIST 인식

```
def test_epoch(dataloader, optimizer, model, loss_fn, log_dict):
    acc=0
    loss=0
    model.eval()
    for X, y in dataloader:
        X = X.view(-1,784).cuda()
        y = y.cuda()

        pred = model(X)
        cost = loss_fn(pred,y)

        acc += torch.sum(torch.argmax(pred,dim=1)==y).item()
        loss += cost.item()

    log_dict['test_acc'].append(acc/len(dataloader.dataset))
    log_dict['test_loss'].append(loss)

    return model, log_dict
```

➔ model.eval() : 모델을 평가 모드로 변경 (batchnorm이나 dropout 관련)

➔ 28x28 크기의 데이터를 784x1 데이터로 reshape & 동시에 GPU 메모리에 올림

➔ 정답 데이터도 GPU 메모리에 올림

14행 예측값과 정답값이 동일한 개수를 카운트하여 저장

5.4 파이토치 다층 퍼셉트론 프로그래밍

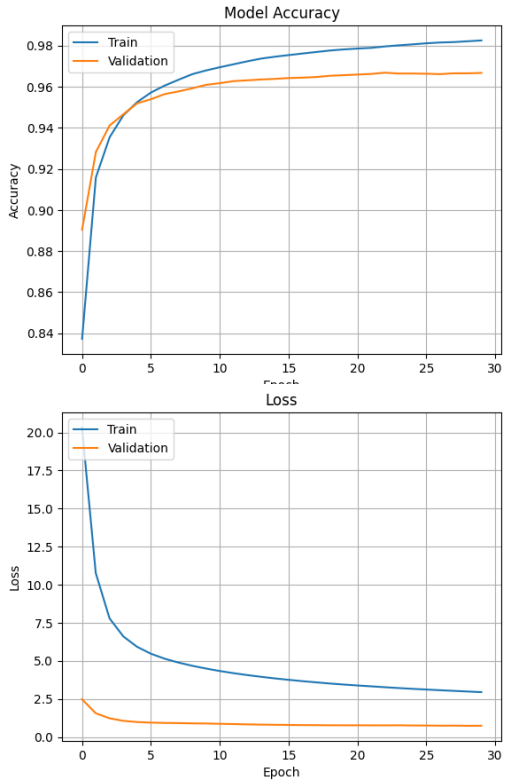
5.4.2 학습 곡선 시각화

[프로그램 5-7(b)] 파이토치 프로그래밍: 다층 퍼셉트론으로 MNIST 인식

```
import matplotlib.pyplot as plt

# 정확도 곡선
plt.plot(history['train_acc'])
plt.plot(history['test_acc'])
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend(['Train','Validation'],loc='upper left')
plt.grid()
plt.show()

# 손실 곡선
plt.plot(history['train_loss'])
plt.plot(history['test_loss'])
plt.title("Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(['Train','Validation'],loc='upper left')
plt.grid()
plt.show()
```

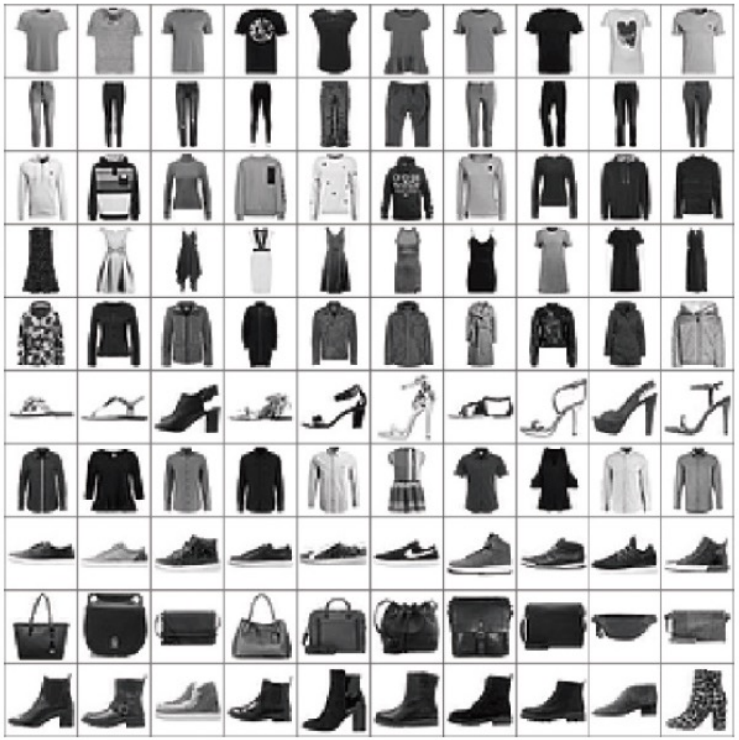


과적합 발생 여부 확인을 위한 학습 곡선 시각화

5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.3 fashion MNIST 인식

- 파이토치는 필기 숫자 데이터인 MNIST와 아주 비슷한 fashion MNIST 를 제공함
- 샘플은 28x28 맵으로 표현되며 훈련 집합 60,000개 샘플, 테스트 집합 10,000개 샘플로 구성됨
- 라벨이 {0,1,2,3,4,5,6,7,8,9}에서 {T-shirt, top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot}로 바뀌고 샘플 영상이 패션 아이템이라는 점만 다름



5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.3 fashion MNIST 인식

[프로그램 5-8]은 [프로그램 5-7] 에서 데이터셋 준비하는 코드만 변경하면 됨

[프로그램 5-8] 파이토치 프로그래밍: 다층 퍼셉트론으로 Fashion MNIST 인식

```
import torch
import tqdm

if __name__ == '__main__':

    history = dict()
    history['train_acc'] = list()
    history['test_acc'] = list()
    history['train_loss'] = list()
    history['test_loss'] = list()

    training_data = datasets.FashionMNIST(
      root="data",
      train=True,
      download=True,
      transform=ToTensor()
    )

    test_data = datasets.FashionMNIST(
      root="data",
      train=False,
      download=True,
      transform=ToTensor()
    )
```

12행 FashionMNIST 학습 데이터 정의

19행 FashionMNIST 평가 데이터 정의

5.4 파이토치 다층 퍼셉트론 프로그래밍

5.4.3 fashion MNIST 인식

[프로그램 5-8] 파이토치 프로그래밍: 다층 퍼셉트론으로 Fashion MNIST 인식

```
train_dataloader = torch.utils.data.DataLoader(training_data,batch_size=128)
test_dataloader = torch.utils.data.DataLoader(test_data,batch_size=128,shuffle=False)

MLP = MultiLayerPerceptron(input_dim=784, hidden_dim=1024, output_dim=10)
MLP = MLP.cuda()
optimizer = torch.optim.Adam(MLP.parameters(), lr=0.001)

for iter in tqdm.tqdm(range(30)):
    MLP, history = train_epoch(train_dataloader, optimizer, MLP, loss_mse, history)
    with torch.no_grad():
        MLP, history = test_epoch(test_dataloader, optimizer, MLP, loss_mse, history)
```

[프로그램 5-7]과 같음