# 딥러닝 파이토치 교과서 2, 4장

**20**기 분석 정재준

# Contents



## 실습 환경 설정과 파이토치 기초



## 딥러닝 시작

# 파이토치 개요

- 17년 초, 페이스북에서 루아(Lua)언어로 개발된 토치(Torch)를 파이썬 버전으로 개발한 프레임워크

- 파이썬 기반의 과학 연산 패키지로 **간결**하고 **구현이 빠르다**는 것이 특징

파이토치 공식 문서에서의 설명은 파이토치는 다음 두 집단을 대상으로 한다.

- 넘파이를 대체하면서 GPU를 이용한 연산이 필요한 경우
- 최대한 유연성과 속도를 제공하는 딥러닝 플랫폼이 필요한 경우

# 파이토치 vs 텐서플로

▪ 딥러닝 F/W 비교


✓Static Computational Graph (TF 1.x 기준)

- 그래프를 한번 만들어 여러 번 실행
- 최적화 및 직렬화 용이

✓Framework 성숙도 높음

- 레퍼런스 구현 및 모델 다수
- 프레임워크로써의 편의 기능 제공
- 분산병렬 학습 기능 제공

**실 서비스에 적합**

PyTorch

✓Dynamic Computational Graph

- 매 Forward Pass 마다 새로운 그래프 생성
- RNN 등 dynamic graph가 필요한 알고리즘 적합

✓개발 생산성 높음

- 개발, 디버깅 용이
- 직관적이고 깔끔한 API

**알고리즘 프로토타이핑, 연구에 적합**


TensorFlow

# 파이토치의 특징 및 장점

파이토치란? “GPU에서 텐서 조작 및 동적 신경망 구축이 가능한 프레임워크”

- 텐서 : 파이토치의 데이터 형태, 단일 데이터 형식으로 된 자료들의 다차원 행렬
- 동적 신경망 : 훈련을 반복할 때마다 실시간으로 네트워크 변경이 가능한 신경망

  - 디버깅이 직관적이고 ‘간결’함
  - CPU 사용률이 텐서플로와 비교하여 낮음 (성능 좋음)
  - 텐서플로처럼 잦은 API 변경이 없다. (배우기 쉬움)

# 다양한 API

## torch.autograd (자동 미분 패키지)

- 일반적인 프레임워크에서는 신경망에 약간의 변경이 있으면 신경망 구축을 다시 시작
- 파이토치는 '자동 미분'이라는 기술로 실시간으로 네트워크 수정이 반영된 계산을 실행

## torch.nn (신경망 구축 및 훈련 패키지)

- 신경망을 쉽게 구축, 사용하게 하는 패키지. (CNN, RNN, 정규화 등이 포함되어 있다.)

## torch.multiprocessing (파이썬 멀티프로세싱 패키지)

- 파이토치 프로세스 전반에 걸쳐 텐서의 메모리 공유가 가능
- 서로 다른 프로세스에서 동일한 데이터(텐서)에 대한 접근 및 사용이 가능

## torch.utils (DataLoader 및 기타 유틸리티 패키지)

- 모델에 데이터를 제공하기 위한 DataLoader 모듈이나, 병목 현상 디버깅, 모델의 일부를 검사는 모듈 등 다양하게 존재

# 파이토치 기초 문법

- 텐서 다루기
  - 텐서 생성 및 변환
  - 텐서의 인덱스 조작
  - 텐서 연산 및 차원 조작

```
[ ]  1 # 텐서 차원 조작
     2 print(temp.shape)
     3 print('-----------------------')
     4 print(temp.view(4,1))
     5 print('-----------------------')
     6 print(temp.view(-1))
     7 print('-----------------------')
     8 print(temp.view(1, -1))
     9 print('-----------------------')
    10 print(temp.view(-1, 1))
```

```
torch.Size([2, 2])
-----------------------
tensor([[1],
        [2],
        [3],
        [4]])
-----------------------
tensor([1, 2, 3, 4])
-----------------------
tensor([[1, 2, 3, 4]])
-----------------------
tensor([[1],
        [2],
        [3],
        [4]])
```

# 파이토치 기초 문법

- 데이터 준비
  - 단순하게 파일을 불러와서 사용
  - 커스텀 데이터셋을 만들어서 사용
  - 파이토치에서 제공하는 데이터셋 사용

```python
# 파이토치 제공 데이터셋 사용 !!!! 실행할꺼면 한번만!
import torchvision.transforms as transforms

mnist_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (1.0,))
])  # 평균이 0.5 표준편차가 1.00이 되도록 데이터 분포 조정

from torchvision.datasets import MNIST
import requests
download_root = '/content/drive/MyDrive/BOAZ/분석/080289-main/chap02/data/MNIST_DATASET'  #다운 받을 경로

train_dataset = MNIST(download_root, transform=mnist_transform, train=True, download=True)  # train 데이터셋
test_dataset = MNIST(download_root, transform=mnist_transform, train=False, download=True)  # test 데이터셋
valid_dataset = MNIST(download_root, transform=mnist_transform, train=False, download=True) # validation 데이터셋
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to /content/drive/MyDrive/BOAZ/분석/080289-main/chap02/data/MNIST
100% 9912422/9912422 [00:00<00:00, 144815674.46it/s]
Extracting /content/drive/MyDrive/BOAZ/분석/080289-main/chap02/data/MNIST_DATASET/MNIST/raw/train-images-idx3-ubyte.gz to /content/drive/

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to /content/drive/MyDrive/BOAZ/분석/080289-main/chap02/data/MNIST
100% 28881/28881 [00:00<00:00, 1556113.99it/s]
Extracting /content/drive/MyDrive/BOAZ/분석/080289-main/chap02/data/MNIST_DATASET/MNIST/raw/train-labels-idx1-ubyte.gz to /content/drive/
```

# 파이토치 기초 문법

- 모델 정의
  - 단순 신경망을 정의하는 방법
  - nn.Module()을 상속하여 정의하는 방법

  - Sequential 신경망을 정의하는 방법
    - 객체 안에 있는 모듈을 순차적으로 실행해준다.

  - 함수로 신경망을 정의 하는 방법
    - Sequential과 동일하지만, 함수로 재사용이 가능하다는 장점이 있다.
    - 모델이 복잡해지는 단점도 있다.

# 파이토치 기초 문법

- 모델의 파라미터 정의
  - 손실 함수(Loss function)
  - 옵티마이저(Optimizer)
  - 학습률 스케줄러(Learning Rate Scheduler)
  - 지표(metrics)

```
# 모델 파라미터 정의
from torch.optim import optimizer
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer=optimizer, ler_lambda=lambda epoch: 0.95 ** epoch)
```

# 파이토치 기초 문법

- 모델 훈련
  - 파이토치에서의 특별한 모델 훈련 방법이 있다.
  - optimizer.zero_grad() 메서드로 학습 할 때마다 기울기를 초기화해줘야 한다.
  - 왜냐하면 .backward() 메서드는 자동으로 이전 기울기 값에 누적하여 계산하기 때문.
  - optimizer.step() 으로 기울기 업데이트를 하면 훈련 코드 완료!

```
# 모델 훈련
for epoch in range(100):
  yhat = model(x_train)
  loss = criterion(yhat, y_train)
  optimizer.zero_grad()   # 기울기 초기화!
  loss.backward()   # 역전파 학습!
  optimizer.step()  # 기울기 업데이트
```

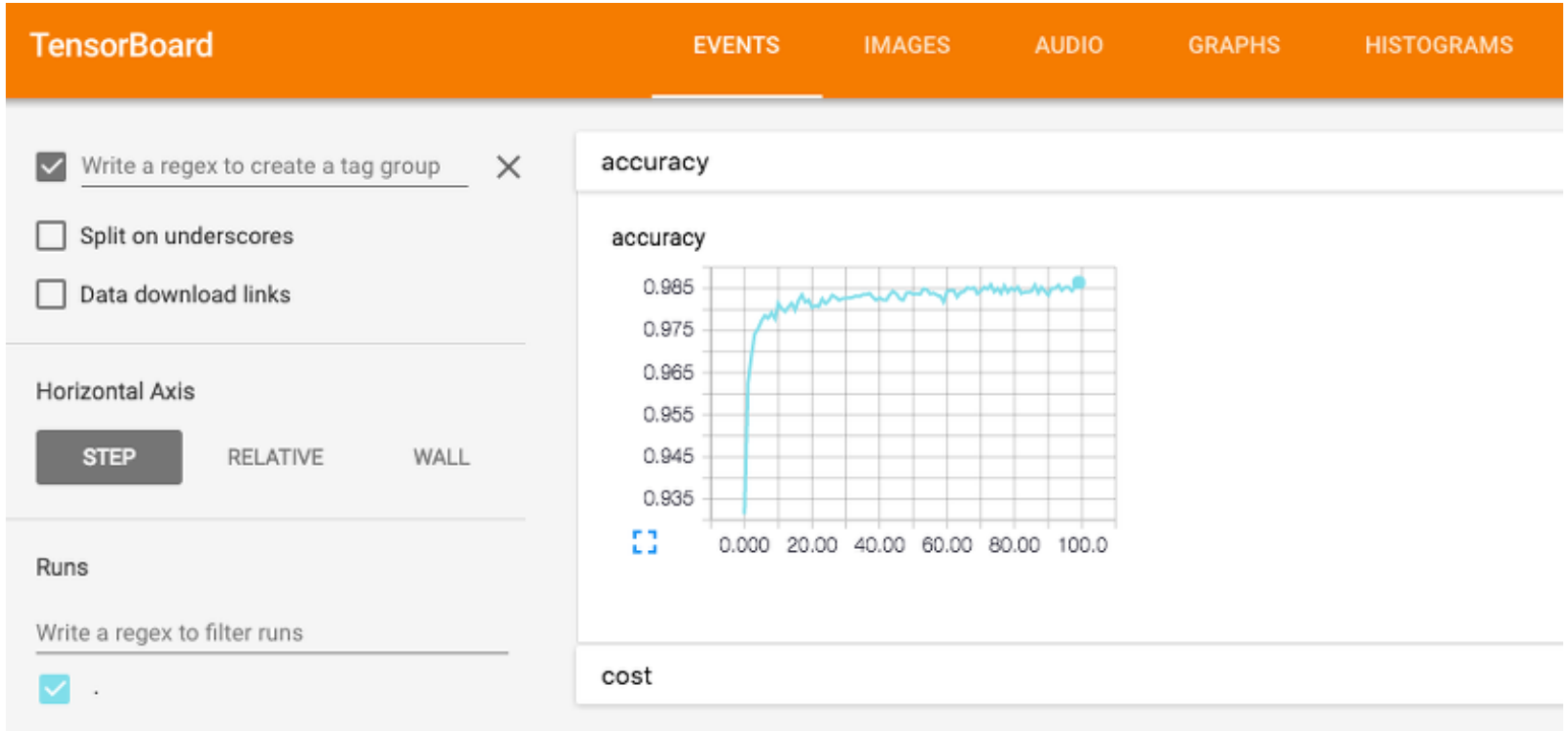
# 파이토치 기초 문법

- 모델 평가
  - 크게 함수를 이용하여 모델을 평가하는 방법과 모듈을 이용하여 모델을 평가하는 방법이 있다.
  - 위에 방법 외에도 사이킷런에서 제공하는 metrics 모듈을 이용할 수 있겠다.

```
1 # 모델 평가
2 import torch
3 import torchmetrics
4
5 preds = torch.randn(10, 5).softmax(dim=-1)
6 target = torch.randint(5, (10,))
7
8 acc = torchmetrics.functional.accuracy(preds, target)
```

# 파이토치 기초 문법

- 훈련 과정 모니터링
  - 사람이 직접 보고 성능을 추적하거나 평가하는 용도로 텐서보드를 통해 시각화하는 방법이 있다.



# 실습환경 설정

- 아나콘다
  - www.anaconda.com/products/individual　　환경 변수 주의!
- 파이토치
  - Anaconada Prompt를 통해 터미널로 설치　　가상 환경 설치 주의!

# 파이토치 코드 맛보기

- 간단한 분류 및 회귀 모델 개발 프로세스 보기
  - 라이브러리 호출
  - 데이터 호출
  - 데이터 전처리
  - 모델의 네트워크 생성
  - 모델의 파라미터 정의
  - 모델 학습
  - 테스트 데이터셋으로 모델 예측
  - 테스트 데이터 셋으로 정확도 확인

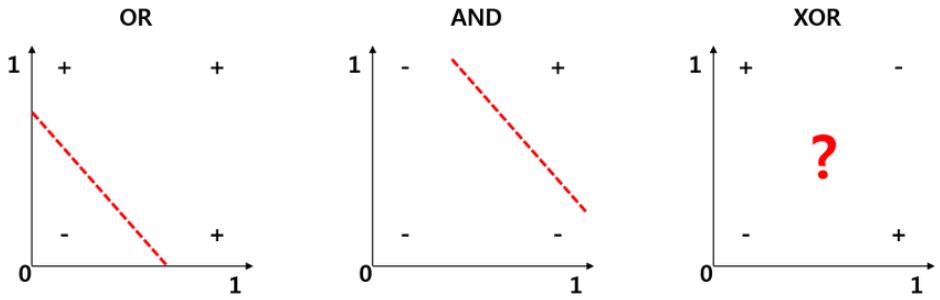
# 인공 신경망의 한계와 딥러닝 출현

•딥러닝의 근간이 되는 이론은 **프랭크 로젠블라트**(Frank Rosenblatt)이 제안한 **퍼셉트론**(perceptron)(1957)이라는 선형 분류기에서 온다.

•퍼셉트론은 **뇌신경**이 작동하는 방식에 영감을 받아 **조합 논리 게이트**로 구성된 단순한 알고리즘 모델이다.

•이 때 퍼셉트론은 **AND**와 **OR**등의 게이트로 분류가 되나, **XOR**게이트는 데이터가 비선형적으로 분리되기 때문에 제대로 된 분류가 어렵다는 한계가 있다.

•이를 극복하기위해 **입력층**과 **출력층**사이에 **은닉층**을 추가하여 비선형적으로 분류가 가능한 **다층 퍼셉트론**(multi-layer perceptron)을 제안했다.




| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# 딥러닝 구조

- 층 (layer)

- **층**은 하나 이상의 텐서를 입력으로 받아 하나 이상의 텐서를 출력하는 데이터 처리 모듈
- 여러개의 **텐서**들이 모여 층을 이룸

- 가중치 (weights)

- 입력 값이 연산 결과에 미치는 영향력을 조절하는 요소. 훈련 데이터를 신경망에 노출시켜서 학습된 정보가 담겨 있다.

- 가중합 (Weight sum, transfer function)

- 각 노드에서 들어오는 신호에 가중치를 곱해서 다음 노드로 전달되는데, 이 값들을 모두 더한 합계를 가중합이라고 한다.
- 노드의 가중합이 계산되면 이 가중합을 활성화 함수로 보내기 때문에 Transfer function이라고도 한다.

# 딥러닝 구조

- **편향** (bias)

- 가중합에 더해지는 상수이다. 활성화 함수의 기준을 조절하는 역할을 한다.

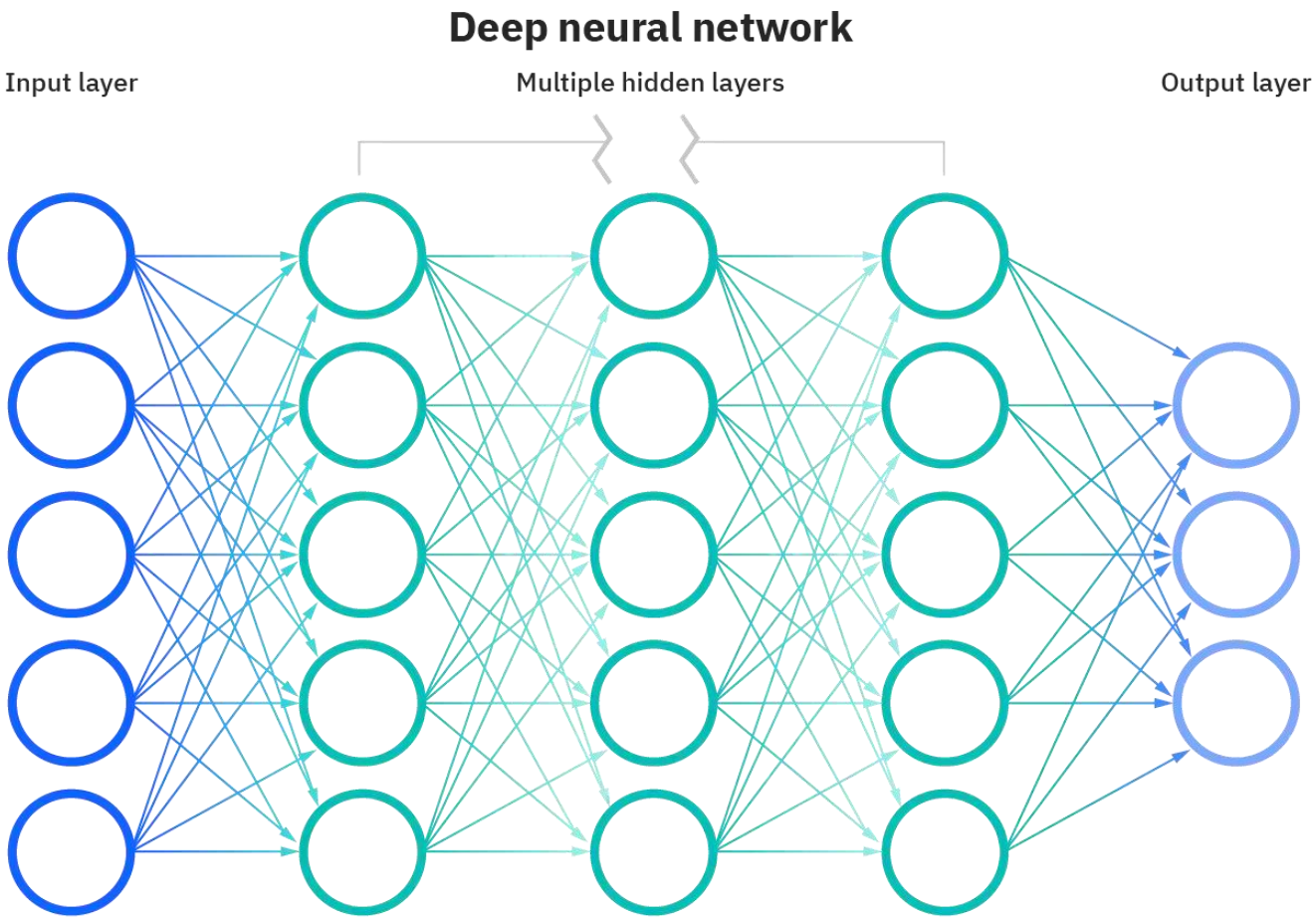
- **활성화 함수** (activation function)

- 전달 함수에서 전달받은 값을 출력할 때 일정 기준에 따라 출력 값을 변화시키는 비선형 함수
- ex) 항등 함수, 시그모이드 함수, tanh 함수, ReLU 함수, LeekyReLU 등...

- **손실 함수** (loss function)

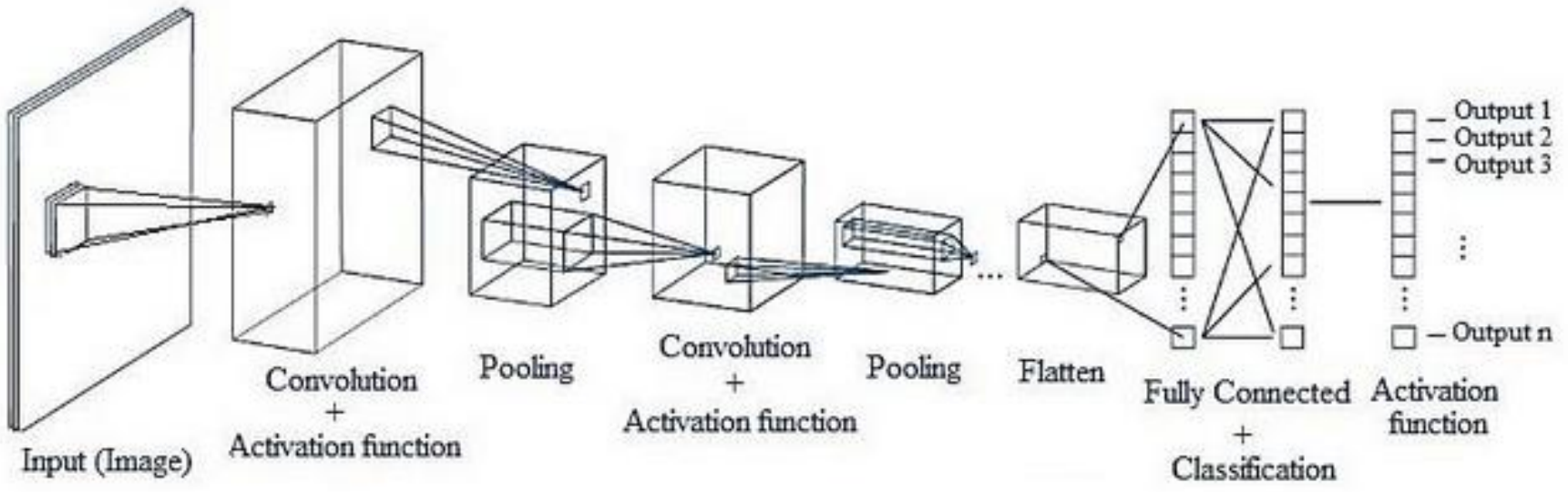
- 예측값과 실제값의 오차를 계산하는 함수, 훈련하는 동안 최소화될 값이다. 주어진 문제에 대한 성공 지표가 된다.
- **평균 제곱 오차**(mean squared error, MSE) : 실제 값과 예측 값의 차이를 제곱하여 평균을 낸 값. 회귀에서 주로 사용된다.
- **크로스 엔트로피 오차** (cross entrophy error, CEE) : 시그모이드  함수에 자연 상수 **e에 반대되는 자연 로그**를 모델의 출력 값에 취한 함수

# 딥러닝 구조


Deep neural network
Input layer
Multiple hidden layers
Output layer

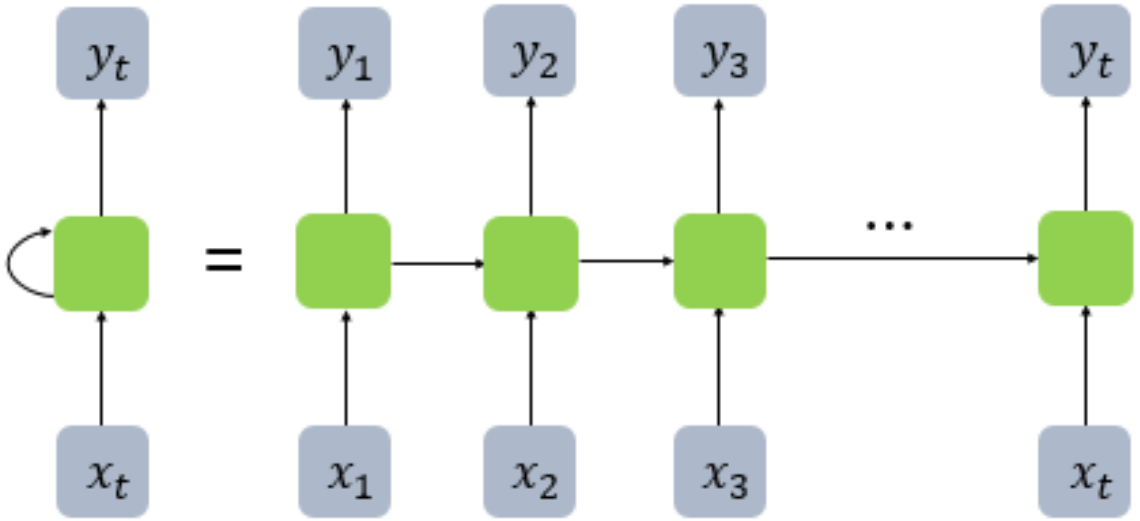
# 딥러닝 알고리즘

- CNN
  - 기존 신경망에 합성곱층(convolutional layer)과 풀링층(pooling layer)를 추가해 패턴 인식 문제(이미지, 영상)에 좋은 성능을 보이는 모델
  - 각 층의 입출력 형상을 유지해 데이터의 공간 정보를 유지하면서 인접 데이터와 차이가 있는 특징(feature)을 인식



# 딥러닝 알고리즘

- RNN
  - 시계열 데이터 같은 시간 흐름에 따라 변화는 데이터에 좋은 성능을 보이는 모델
  - 학습할 때 자기 자신을 참조, 이전 결과와 연관성 있는 결과를 도출



# 딥러닝 알고리즘

- DNN (심층 신경망)

심층 신경망은 결국 선형적으로 분류할 수 없는 인공 신경망의 한계를 극복하고자, 다수의 은닉층을 추가해 비선형적으로 분류할 수 있게 하는 모델.

- RBM (제한된 볼츠만 머신)

가시층과 은닉층으로만 구성된 모델. 특징으로는 가시층은 은닉층과만 연결되고 가시층 혹은 은닉층 내에서의 연결은 없는 것이다.

- DBN (심층 신뢰 신경망)

제한된 볼츠만 머신을 블록처럼 여러 층으로 쌓은 형태로 연결된 신경망.

사전 훈련된 제한된 볼츠만 머신을 층층이 쌓아 올린 구조이고 레이블 없는 데이터에 대한 비지도 학습 또한 가능하다.