인공지능 기술 개요

임세진

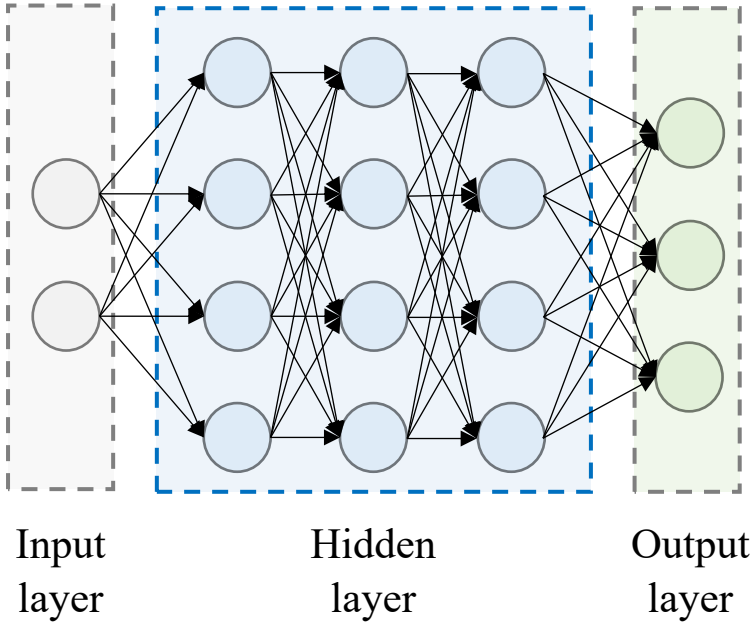
https://youtu.be/D4W_2eVx5RY

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

신경망

- **신경망**
- 가중치 매개변수의 적절한 값을 데이터로부터 자동으로 학습함
- 은닉층(Hidden Layer)을 여러겹(Deep) 쌓아서 복잡한 문제를 풀 수 있음 → 딥러닝



활성화 함수

활성화 함수 (Activation function)

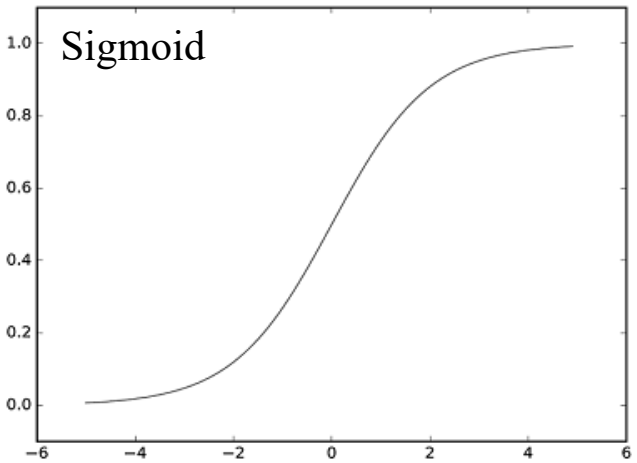
- 입력 신호의 총합을 출력 신호로 변환하는 함수
- 입력 신호의 총합이 활성화를 일으키는지 결정하는 역할 (이전 Layer에서 다음 Layer로 값을 전달할 때 사용)
- 선형 함수가 아닌 비선형 함수를 사용 (선형 함수를 사용하면 층을 깊게 쌓는 의미 X)
  - 선형 함수 : 출력이 입력의 상수배만큼 변하는 함수
  - 비선형 함수 : '선형'이 아닌 함수. 직선 1개로는 그릴 수 없는 함수
- 주로 ReLu 함수 사용

$f(x) = cx$ ← 선형 함수
$y(x) = f(f(f(x)))$ ← 레이어를 3개 쌓았다고 가정
$y(x) = c * c * c * x = c^3x$ ← 레이어를 하나 쌓은 효과

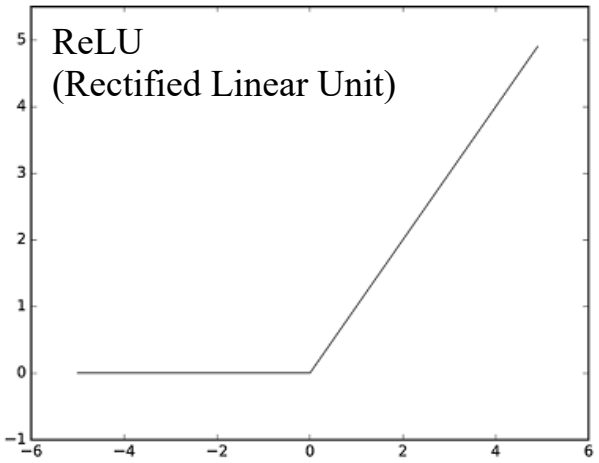
활성화 함수

활성화 함수 (Activation function)

$$s(z) = \frac{1}{1 + e^{-z}}$$

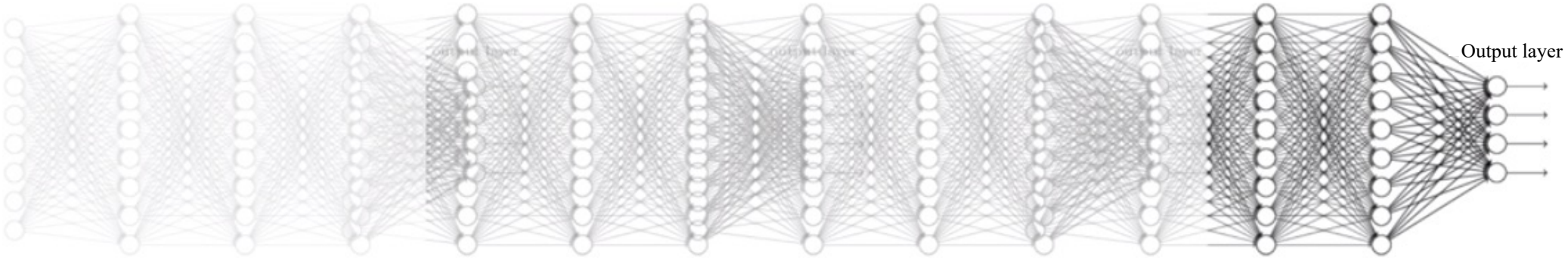


- 미분 결과가 간결하고 사용하기 쉬움
- Return 값이 확률이어서 해석할 때 유용
- 기울기 소실 문제가 발생할 수 있음



$$h(x) = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$$

- **가장 많이 사용되는 활성화 함수**
- 연산 비용이 적어 빠른 학습 가능
- 매우 간단히 구현 가능

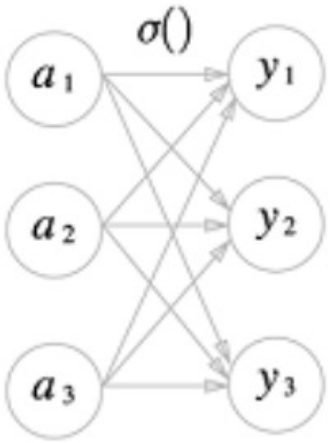


기울기 소실 문제

활성화 함수

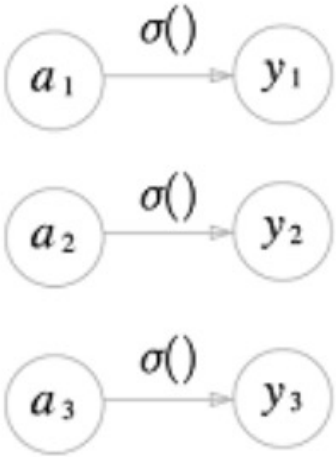
출력층 설계 → 문제의 종류 파악

- 분류 (Classification): 데이터가 어느 클래스에 속하는지 분류하는 문제
  - ✓ Ex) 개/고양이 분류 문제, MNIST 숫자 분류 문제
  - ✓ Sigmoid, Softmax 함수 사용 (각각 이진분류, 다중분류 시 사용)
    - 출력층의 각 뉴런이 모든 입력 신호에서 영향을 받음
    - 0 ~ 1 사이의 실수값 반환 ex) [0.2, 0.1, **0.7**]
    - 출력의 총 합은 1 → Softmax 함수의 출력을 확률로 해석 가능



Softmax 함수

- 회귀 (Regression) : 입력 데이터에서 (연속적인) 수치를 예측하는 문제
  - ✓ Ex) 사람 사진 → 사진 속 인물의 체중 예측
  - ✓ 항등 함수 (identity function) : 입력을 그대로 출력

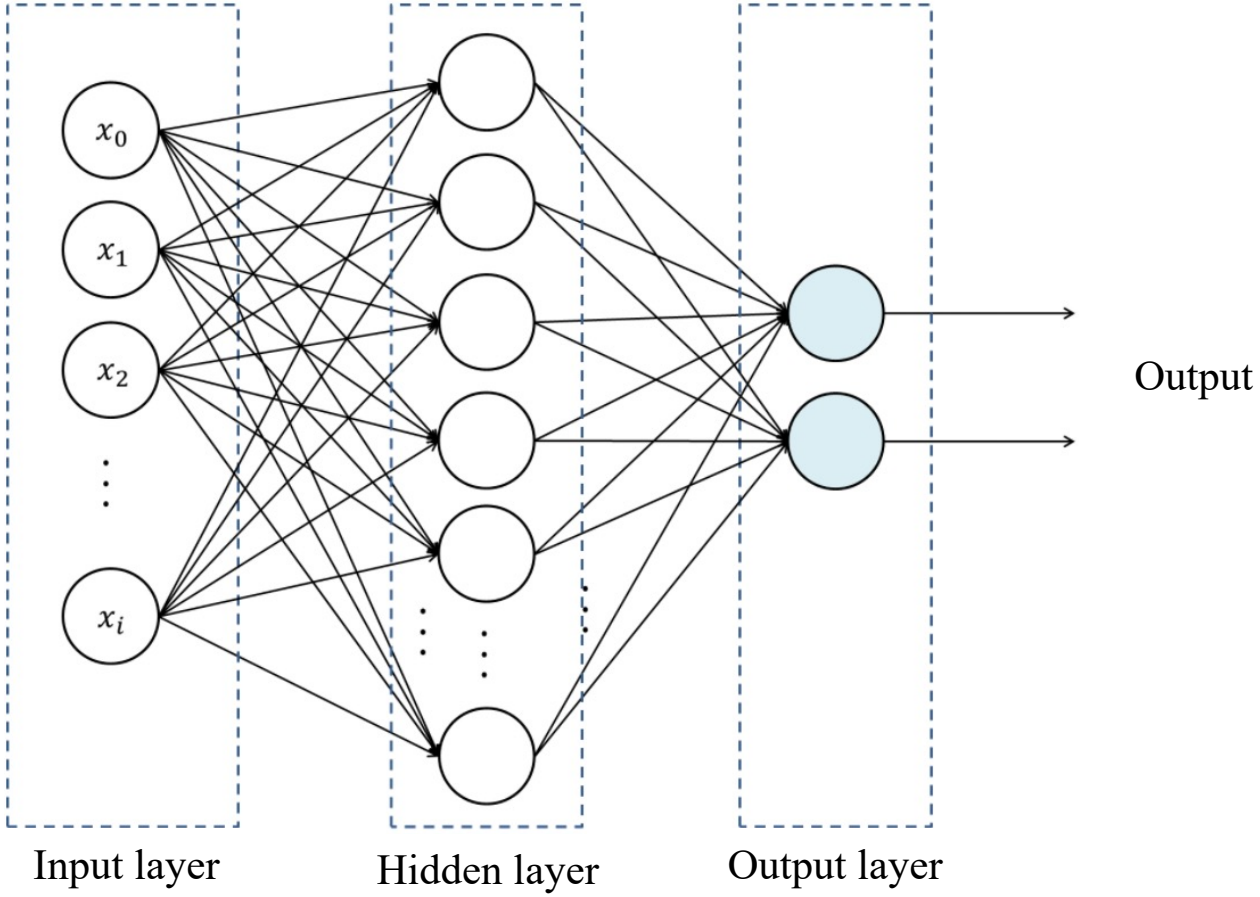


항등 함수

순전파 (Forward propagation)

순전파 (Forward Propagation)

- 입력층에서 은닉층으로 **정방향** 계산



손실 함수와 정확도

손실 함수 (Loss function)

- 순전파로 은닉층을 거쳐 얻은 결과값(Output)에 활성화 함수까지 거친 **예측값과 실제값의 차이를 계산하는 함수**
- 신경망 학습에서 최적의 매개변수 값을 탐색하기 위해 사용하는 지표
  → **높은 정확도**를 얻기 위한 매개변수 값 탐색을 왜 정확도가 아닌 **손실 함수**를 지표로 사용?
- 학습 시 최적의 매개변수 값을 탐색하기 위해 아래의 과정 반복

  매개변수의 미분(기울기)를 계산 → 이를 단서로 매개변수 값 갱신

- 가중치 매개변수의 손실 함수의 미분

  : 가중치 매개변수의 값을 미세하게 변화시켰을 때 손실 함수가 어떻게 변하는지

미분값
- 음수 : 가중치 매개변수를 양의 방향으로 변화시켜 손실 함수 값 줄이기
- 0 : 가중치 매개변수를 어떻게 변화시켜도 손실 함수 값은 줄어들지 않음 ➔ 매개변수 값 갱신 중단
- 양수 : 가중치 매개변수를 음의 방향으로 변화시켜 손실 함수 값 줄이기

손실 함수와 정확도

정확도를 지표로 삼을 경우?

- 미분 값이 대부분의 경우 **0** → **갱신 불가**
- 매개변수를 약간만 조정해서는 정확도가 개선되지 않음

  Ex) 100장의 훈련 데이터 중 35장만 올바르게 인식 → 정확도 35%

  정확도는 35%, 36% 처럼 불연속적으로 변화함

  즉, 매개변수의 미세한 변화에는 반응을 보이지 않으며, 반응이 있더라도 불연속적으로 변화함
- 손실 함수 값은 연속적으로 변화함 (0.93432 …)
- 매개변수의 작은 변화가 주는 파장을 반영해줄 수 있어야 하는데 정확도는 그렇지 못함

➔ **손실 함수**를 지표로 사용

역전파 (Backpropagation)

(오차)역전파 (Backpropagation)

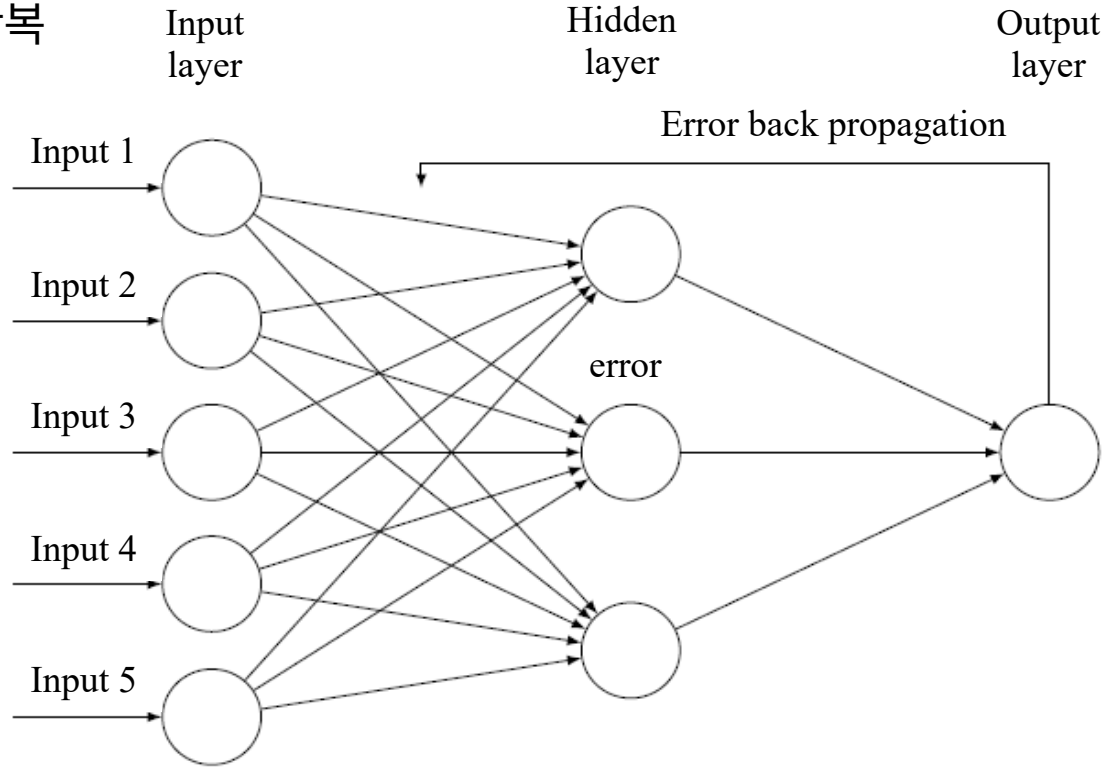
- **순전파의 반대 방향으로 이동하면서 가중치를 업데이트하는 과정**

  ➔ 순전파 알고리즘에서 발생한 오차를 줄이기 위해 가중치를 업데이트하고, 새로운 가중치로 다시 학습하는 과정

- 입력(Input)과 출력(Output) 값을 알고 있는 상태에서 신경망을 학습시키는 방법

  정답(Target)과 모델의 결과값(Output)의 차이 계산 → 오차 값을 뒤로(반대방향으로) 전파해서 각 노드가 가진 가중치 업데이트

- 오차(loss)가 0에 가까워질 때까지 역전파 학습을 반복



매개변수 갱신 (Optimizer)

최적화 (Optimization)

- 손실 함수의 값을 최대한 낮추는 매개변수를 찾는 것
- 손실 함수로 얻은 오차를 이용해 **역전파로 가중치를 업데이트하는 방법**

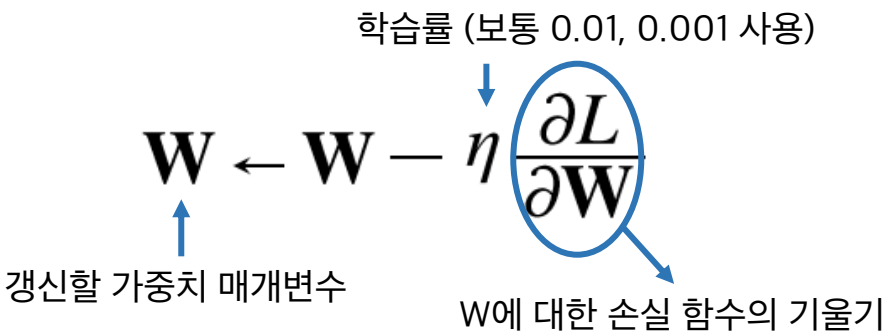
Optimizer 종류 (기준으로 삼는 단서에 따라 분류)

- 확률적 경사 하강법 (SGD)
- 모멘텀
- AdaGrad
- Adam

매개변수 갱신 (Optimizer)

확률적 경사 하강법 (SGD)

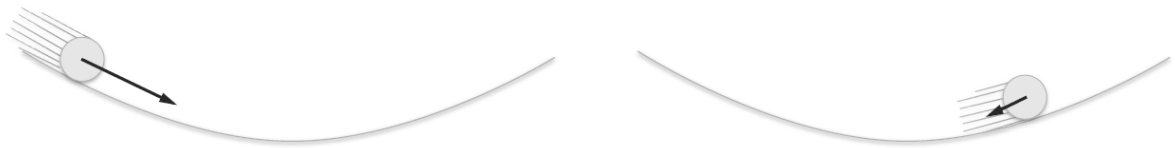
- 매개변수의 기울기(미분) → 기울어진 방향으로 매개변수 값 갱신
- 기울어진 방향으로 일정 거리만 가겠다는 단순한 방법
- 단순하여 구현이 쉽지만, 문제에 따라 비효율적일 때가 있음



매개변수 갱신 (Optimizer)

모멘텀 (Momentum)

- 기울기 값만 반영하는 SGD 개선 (기존의 SGD에 관성을 더해줌 ➔ 수렴속도 개선)
- 관성의 법칙 이용 (이전 물체의 속도는 현재 물체의 속도와 관련 있음)

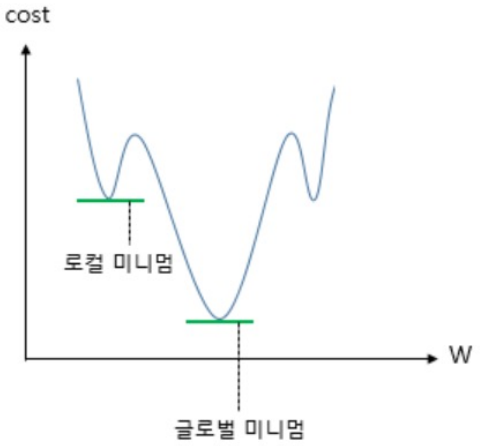


지면 마찰, 공기 저항의 역할

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

속도



기울기 방향으로 힘을 받아 물체가 가속되는 물리법칙 표현

(속도가 클수록 기울기가 크게 update 됨)

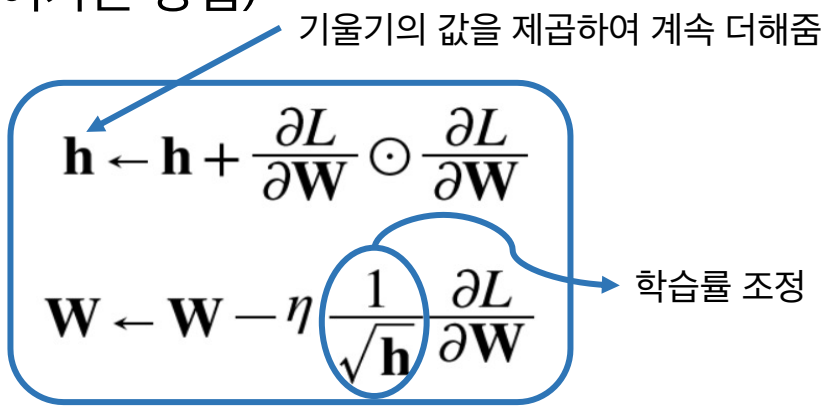
매개변수 갱신 (Optimizer)

신경망 학습에서 **학습률(learning rate)**은 중요

→lr이 너무 작으면 학습시간이 길어지고, 너무 크면 발산하여 학습이 제대로 X

→학습률 감소 기법 사용 (학습을 진행하면서 학습률을 점차 줄여가는 방법)

AdaGrad



- 각각의 매개변수에 맞춤형 값을 만들어줌

  개별 매개변수에 적응적으로 학습률을 조정하며 학습 진행

  많이 갱신된 매개변수는 학습률이 낮아지도록 수행 ← 학습률 감소가 매개변수마다 다르게 적용

- 과거의 기울기를 제곱하여 계속 더해감

  학습을 진행할수록 갱신 강도가 약해짐 → RMSProp으로 개선 (먼 기울기는 서서히 잊고 새로운 기울기 정보 크게 반영)

매개변수 갱신 (Optimizer)

Adam

- 모멘텀 + AdaGrad
- 학습률을 줄여나가고 속도를 계산하여 학습의 갱신 강도를 적응적으로 조정해 나가는 방법
- 하이퍼파라미터의 ‘편향 보정’이 진행됨
- 가장 많이 사용되는 Optimizer

신경망 학습

학습

- 훈련 데이터로부터 **가중치 매개변수의 최적값**을 **자동으로** 획득하는 것
- 신경망의 학습 지표 : 손실 함수
- 학습의 목표 : 손실 함수의 결과값을 가장 작게 만드는 가중치 매개변수 찾기
- **Train : Val : Test**의 비율은 일반적으로 **6 : 2 : 2**로 설정

Dataset

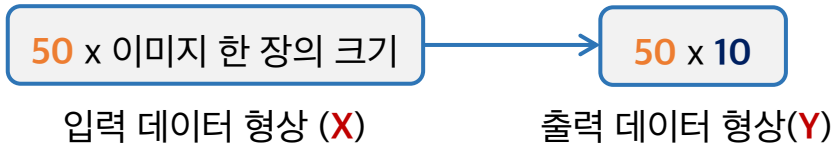
- 훈련 데이터 (Train set) : 이 데이터만 사용하여 학습 → 최적의 매개변수 찾기
- 검증 데이터 (Validation set) : 학습이 완료된 모델을 검증하는 데이터셋 ← 하이퍼파라미터 성능 평가 목적
- 시험 데이터 (Test set) : 학습과 검증이 완료된 모델의 범용 성능을 평가하는 데이터셋

신경망 학습

배치 처리

- Ex) 이미지 여러 장(**50장**)을 한꺼번에 입력 + **10개**의 class로 분류

  ➔50장 분량의 입력 데이터의 결과가 한번에 출력



이미지 한 장의 분류 결과 값

X \ Y	0	1	2	3	4	5	6	7	8	9
0	0.01	**0.8**	0.01	0.02	0.16	0	0	0	0	0
1										
2										
⋮										
49										

batch

- 배치(batch) : 하나로 묶은 입력 데이터
- 배치 처리의 이점 : 이미지 1장 당 처리 시간을 대폭 줄여줌 ➔ 효율&신속
  - ✓ 대부분의 수치 계산 라이브러리가 **큰 배열을 효율적으로 처리하도록 최적화**
  - ✓ 데이터를 읽는 횟수 감소 [느린 I/O의 수행이 감소] → CPU/GPU로 순수 계산 수행률↑
  - ✓ 큰 규모의 신경망의 경우, 데이터 전송 부분에서 병목이 일어나는 경우도 있음 → 배치 처리로 버스 과부하 감소

즉, 컴퓨터가 분할된 작은 배열을 여러 번 계산하는 속도 < 큰 배열을 한꺼번에 계산하는 속도

신경망 학습

신경망의 학습 절차

미니배치 : 훈련 데이터 중 일부를 가져옴
미니배치의 손실 함수 값을 줄이는 것이 학습의 목표

↓

기울기 산출 : 미니배치의 손실함수 값을 줄이기 위해
각 가중치 매개변수의 기울기를 구함

↓

매개변수 갱신 : 가중치 매개변수를 기울기 방향으로 아주 조금씩 갱신

↓

반복 : 위의 1~3단계 반복

하이퍼파라미터 (Hyperparameter)

하이퍼파라미터 (Hyperparameter)

- **각 layer의 뉴런 수, 배치 크기, 매개변수 갱신 시의 학습률과 가중치 감소 등**이 해당됨
- 하이퍼파라미터의 값을 적절히 설정하는 것은 모델의 성능 향상과 직결됨
- 하이퍼파라미터의 성능 평가용 데이터로 검증 데이터를 사용

하이퍼파라미터 최적화 단계

- 최적 값이 존재하는 범위를 줄여나가는 것이 핵심

1. 하이퍼파라미터 값의 대략적인 범위 설정
2. 설정된 범위 내에서 무작위로 값 추출 (샘플링)
3. 해당 값으로 학습 후, 검증 데이터로 정확도 평가 (시간 단축을 위해 에폭을 작게 설정)
4. 정확도에 따라 위 작업(2-3)을 반복하여 범위를 좁혀감
5. 압축된 범위에서 값을 하나 선택

하이퍼파라미터 (Hyperparameter)

하이퍼파라미터 튜닝 (Hyperparameter Tuning)

- **학습률, 손실함수, Layer 크기가 가장 큰 영향을 줌**
- 최적화 함수 및 배치 사이즈의 경우 영향이 적음
- **학습률, 손실함수 설정 → Layer 추가/제거 및 정규화 → … 순으로 진행**
- 학습률의 경우, 3-10배 단위로 변경하며 실험 진행
- 손실이 증가하거나, 지그재그 패턴으로 발산하는 경우 → lr이 너무 높음
- 손실이 충분히 감소하지 않는 경우 → lr이 너무 낮음
- 일반적으로 **lr은 0.0001-0.001**로 설정

Hyperparameter	Approximate sensitivity
Learning rate	High
Optimizer choice	Low
Other optimizer params (e.g., Adam beta1)	Low
Batch size	Low
Weight initialization	Medium
Loss function	High
Model depth	Medium
Layer size	High
Layer params (e.g., kernel size)	Medium
Weight of regularization	Medium
Nonlinearity	Low

하이퍼파라미터 튜닝 시 우선 순위

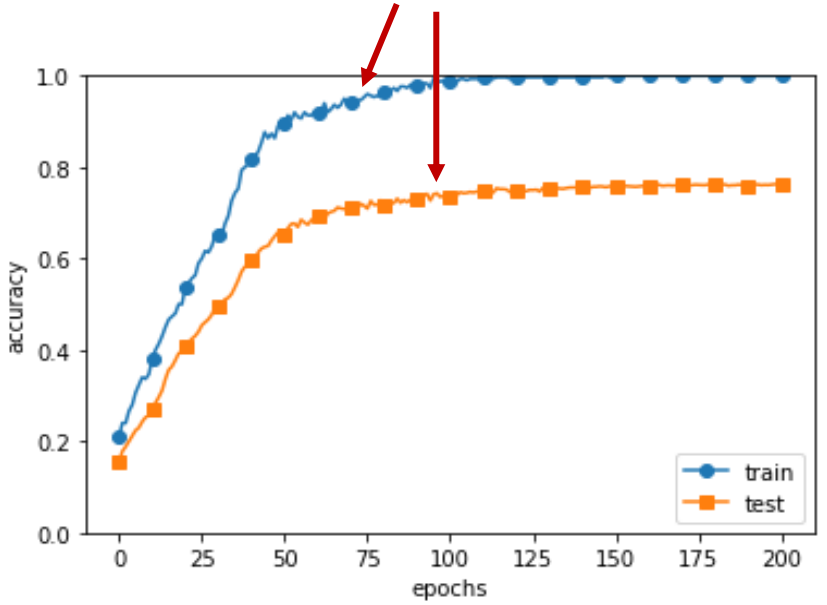
표 출처 : http://josh-tobin.com/assets/pdf/troubleshooting-deep-neural-networks-01-19.pdf

과적합 (Overfitting)

과적합 (Overfitting)

- **신경망이 훈련 데이터에 치우쳐 학습이 되어** 그 외의 데이터에는 제대로 대응하지 못하는 상태
- 학습 데이터를 과하게 암기하여 노이즈까지 학습한 상태로 볼 수 있음
- 하지만 딥러닝 모델은 **범용 성능이 요구**됨
  - ✓ **학습한 적 없는 데이터가 주어지더라도 바르게 식별**해야 바람직한 모델임
  - ✓ 과적합을 억제하는 기술 중요
- **과적합이 일어나는 경우**
  - ✓ 매개변수가 많고 네트워크가 복잡한 경우
  - ✓ 훈련 데이터가 적은 경우

Train 정확도와 Test 정확도 큰 차이 발생



과적합이 발생한 모습

과적합을 막는 방법

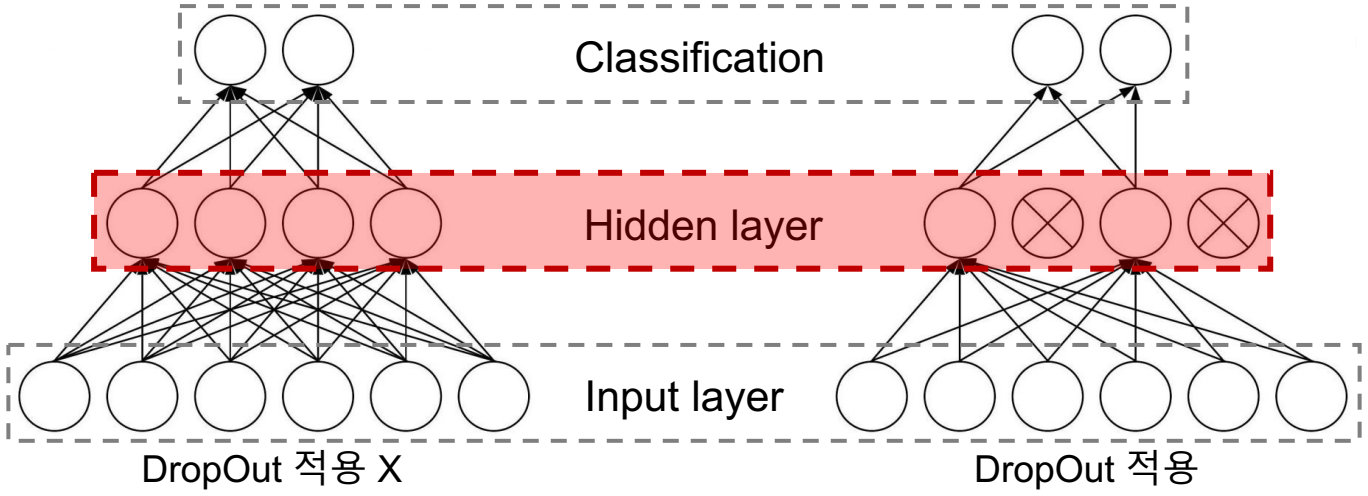
1. **데이터의 양을 늘리기**
   - 데이터의 양이 적으면 모델이 **데이터의 특정 패턴이나 노이즈까지 쉽게 암기하여** 과적합 발생할 확률↑
   - 데이터의 양을 늘릴수록 모델이 데이터의 **일반적인 패턴을 학습하여** 과적합 방지 가능
   - 데이터 증강(Data Augmentation) 기법을 통해 의도적으로 기존 데이터를 조금씩 변형하여 늘리기도 함

     Ex) 이미지의 경우 회전, 노이즈 추가, 일부분을 수정하는 등의 방법으로 데이터를 증식함
2. **모델의 복잡도 줄이기**
   - 은닉층의 수와 매개변수의 수 조정
3. **가중치 규제(Regularization) 적용하기**
   - 가중치 매개변수 값이 커서 오버피팅이 발생할 수 있음 → 손실 함수 연산 시 큰 가중치는 그에 상응하는 큰 페널티 부과
4. **드롭아웃(DropOut)**

과적합을 막는 방법

드롭아웃 (DropOut)

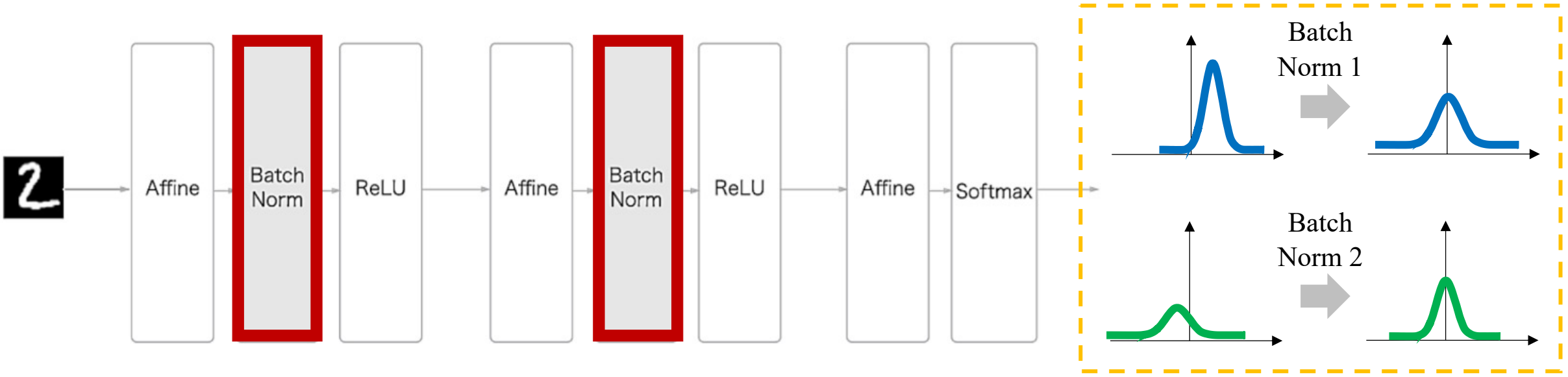
- 신경망 모델이 복잡해질 경우 가중치 규제만으로는 과적합을 억제하기 어려움 ➔ 드롭아웃 기법 이용
- 학습 시 모델이 **특정 뉴런에 너무 의존적이게 되는 것을 방지**해줌
- 뉴런을 임의로 삭제하면서 학습하는 방법
  - ✓ 학습 시 **은닉층의 뉴런을 무작위로 골라 삭제** → 삭제된 뉴런은 신호 전달 X (순전파, 역전파 모두 적용)
  - ✓ 테스트 시 **모든 뉴런에** 신호 전달 (테스트시에는 일반적으로 드롭아웃 사용 X)
- 무작위 삭제 ➔ 매번 다른 모델을 학습시키는 효과



배치 정규화 (Batch Normalization)

배치 정규화 (Batch Normalization)

- 학습 과정에서 평균과 분산을 조정 → **정규 분포로 만들어 학습 안정화**
- **기울기 소실 방지 → 높은 학습률 사용 가능 및 학습 가속화**
- **학습하는 과정 자체를 전체적으로 안정화**하여 학습 속도를 가속시킬 수 있는 근본적인 방법
- 각 layer마다 정규화하는 layer를 두어, 변형된 분포가 나오지 않도록(활성화 값이 적절히 분포되도록) 조절함



배치 정규화 (Batch Normalization)

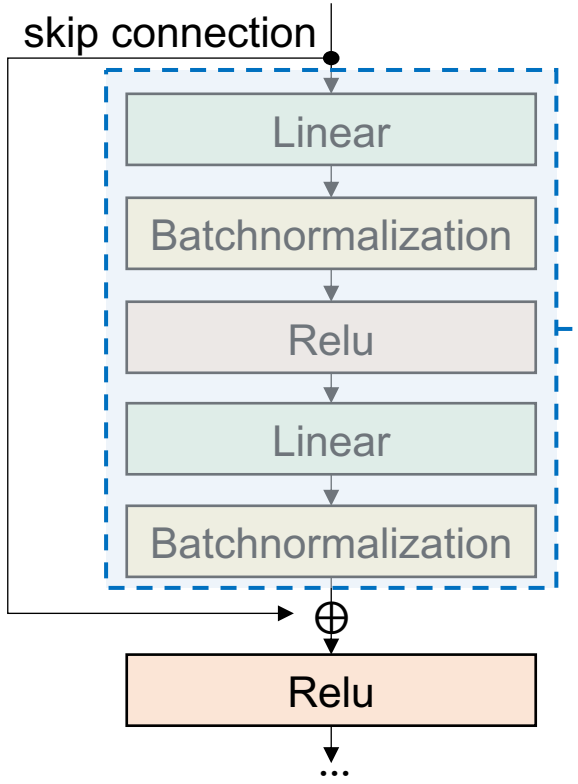
배치 정규화를 하는 이유

- **빠른 학습** 가능 (학습 속도 개선)
- 가중치 초기값에 크게 의존하지 않음 (원래는 가중치 초기값에 의존적)
- **오버피팅 억제** (드롭아웃 등의 필요성 감소)

➔ 딥러닝 학습에서의 부정적인 요소 제거

Residual Network

- 깊은 네트워크를 학습시키기 위한 방법
  - 일반적으로 네트워크 깊이가 깊어질수록 데이터로부터 풍부한 특징을 추출할 수 있음
  - 하지만 너무 깊을 경우(Layer가 너무 많을수록) 오히려 성능이 떨어짐 ex) 과적합, 기울기 소실, 기울기 폭주
- Output을 몇 개의 layer를 건너뛴 후, layer의 input에 추가하여 **추가 정보 학습 가능**
  → 기울기 소실 방지, 과적합 방지 + 더 쉽고 빠른 학습 가능



잔여블록(Residual Block)을 이용하여 네트워크의 최적화 난이도를 낮춤

제안 논문 : Deep Residual Learning for Image Recognition

Gated Linear Unit

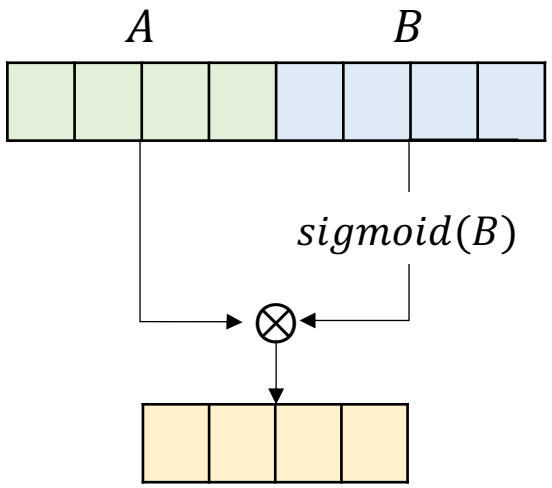
- **정보의 출입을 제어**

  중요 정보: sigmoid 곱에서 살아 남음

  아닌 경우: 사라짐

  → 중요 요소에 집중 가능

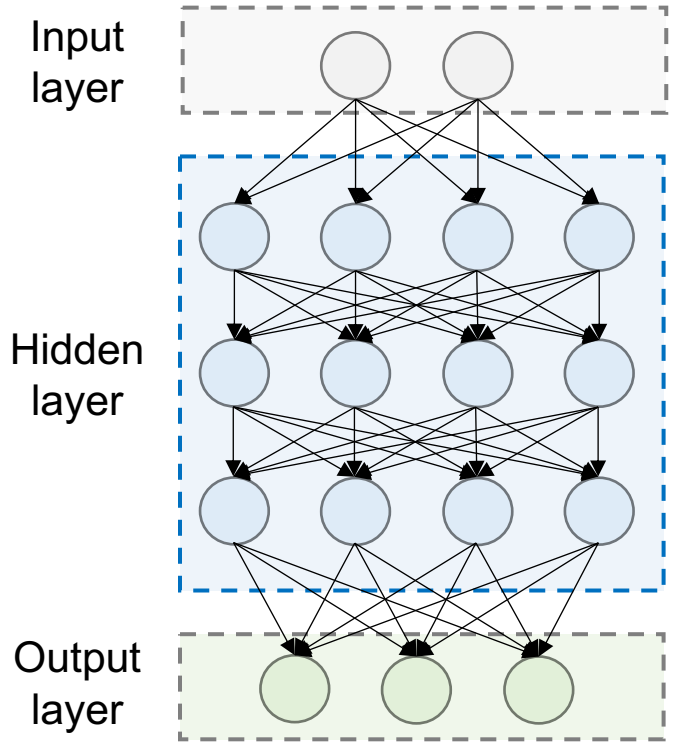
- **더 빠르고 안정적인 학습 가능**



인공 신경망 종류 - MLP

MLP (Linear Multi Layer Perceptron)

- 신경망의 **가장 기본적인 형태**
- 2개 이상의 은닉층(hidden layer) 필요
- 모든 unit이 연결 되어 있음
- **시계열, 이미지 신경망보다는 전체적인 데이터 특징 고려**



시계열 데이터란?

일정한 시간동안 수집된 일련의 순차적으로 정해진 데이터의 집합

시계열 데이터 분석을 통해 데이터가 가지고 있는 **규칙을 찾아 미래의 값을 예측하는 것이 목적**

Ex) 일일 주식 가격, 연도별 특정 사건의 수치

인공 신경망 종류 - MLP

프레임워크별 MLP 모델 구조 코드 [Pytorch / Tensorflow]

Pytorch

```pytorch
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()

        self.layer1 = nn.Linear(784, 256, bias = True)
        self.layer2 = nn.Linear(256, 256, bias = True)
        self.layer3 = nn.Linear(256, 10, bias = True)
        self.relu = nn.ReLU()
```

Tensorflow

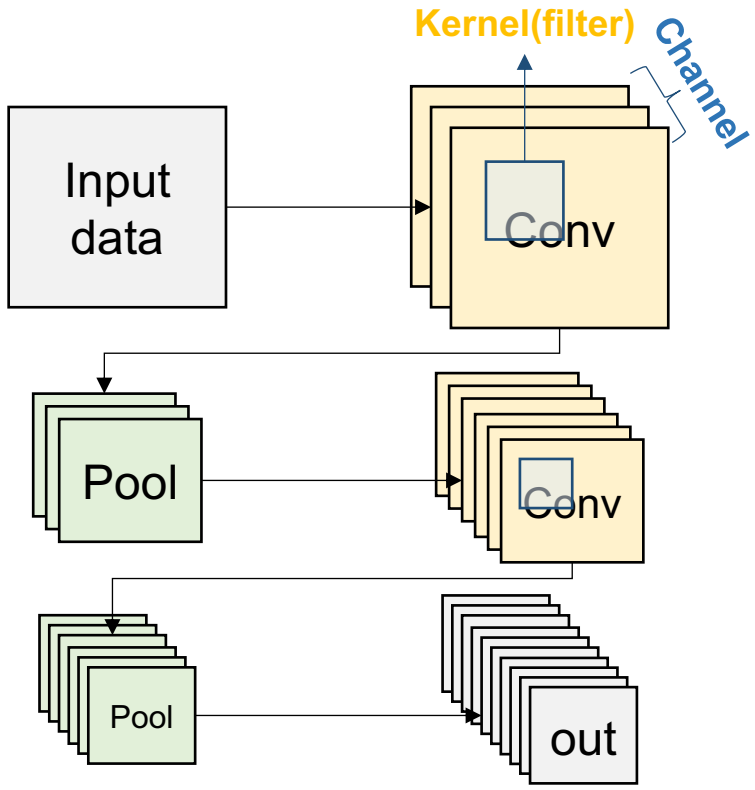
```tensorflow
# MLP 정의
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

인공 신경망 종류 - Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN)

- **시계열(1d), 이미지(2d)에 효과적**인 신경망
- Conv + Pool 구조 & Kernel(가중치 행렬)과 입력 데이터의 컨볼루션 연산→ **지역적 특징 학습**
- 출력(Feature Map)의 크기는 줄어들며 특징 추출 및 강화, Channel을 늘려 손실된 정보 보완

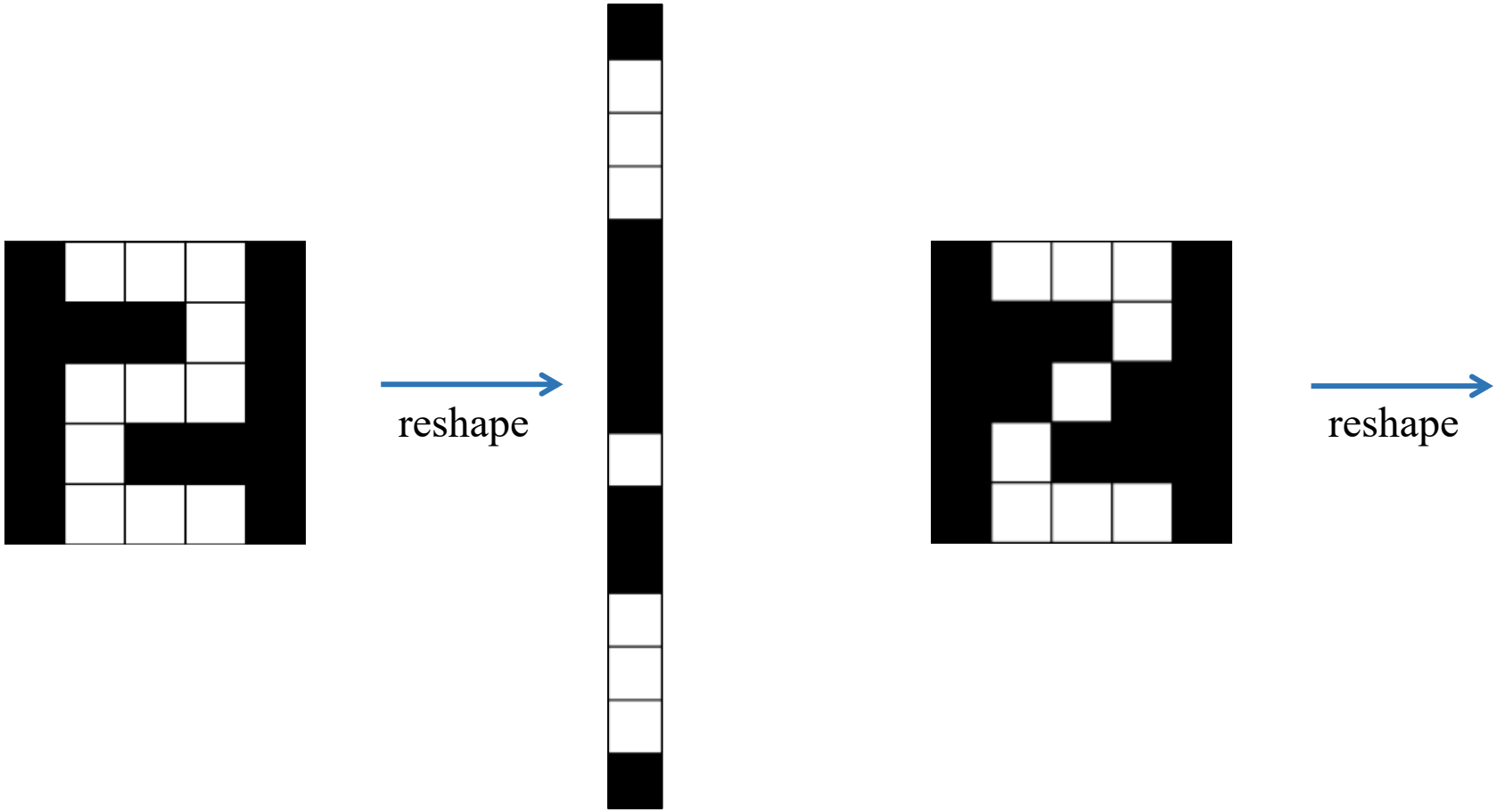
```
class CNN(nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv      -> (?, 28, 28, 32)
        #    Pool      -> (?, 14, 14, 32)
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
```



인공 신경망 종류 - Convolutional Neural Networks (CNN)

CNN의 필요성 (이미지 처리 분야에 CNN이 사용되는 이유)

- MLP방식으로 이미지 분류 시 오버피팅 발생 → MLP 방식은 **각각의 픽셀을** 이미지의 특징으로 인식
- 이미지에 존재하는 **공간적, 지역적인 정보를 고려**할 수 있음

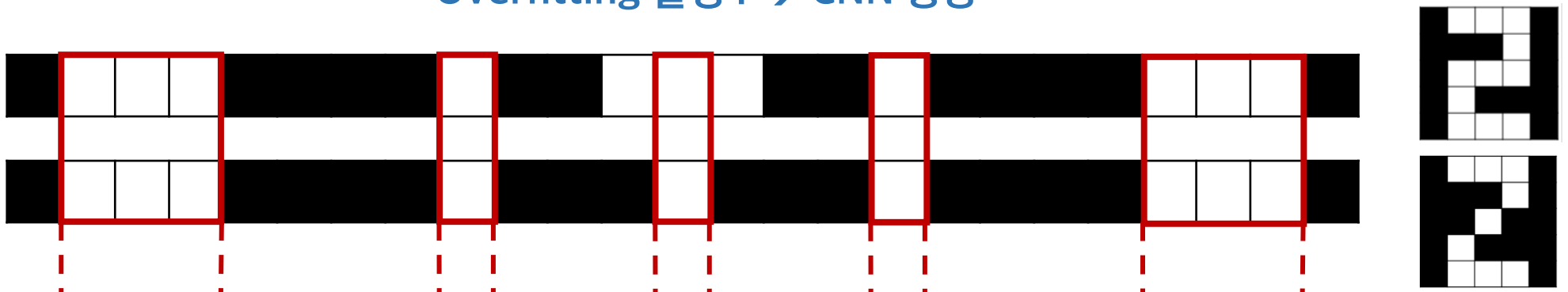


인공 신경망 종류 - Convolutional Neural Networks (CNN)

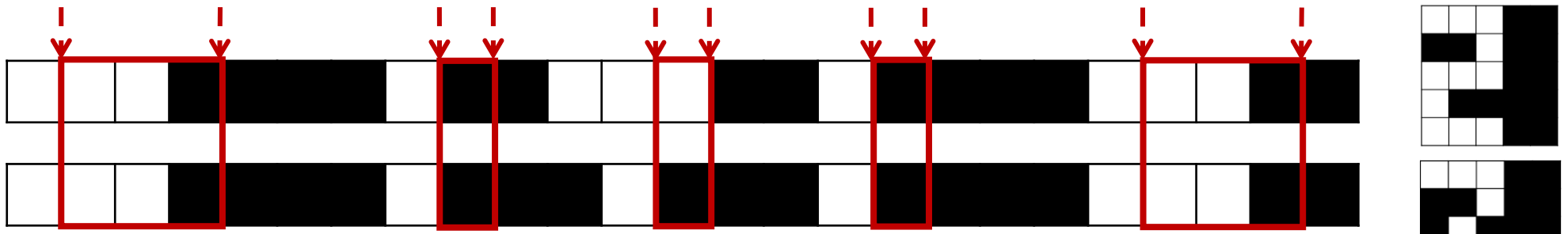
- **MLP 방식의 문제점** : 1차원으로 변형된 데이터로는 이미지를 구분하기 어려움 → 특징이 사라짐(오버피팅 발생)
- 2차원 데이터로는 이미지를 쉽게 구분하고 특징을 확인할 수 있음
- CNN은 2차원 데이터에서 특징을 찾아낼 수 있음

<Train data>

Overfitting 발생 ! → CNN 등장

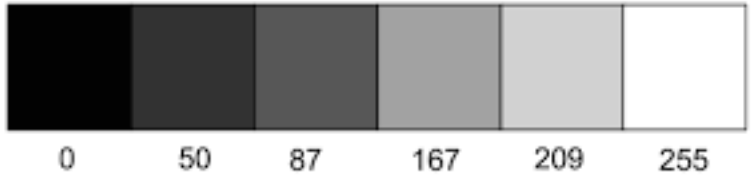


<Test data>



인공 신경망 종류 - Convolutional Neural Networks (CNN)

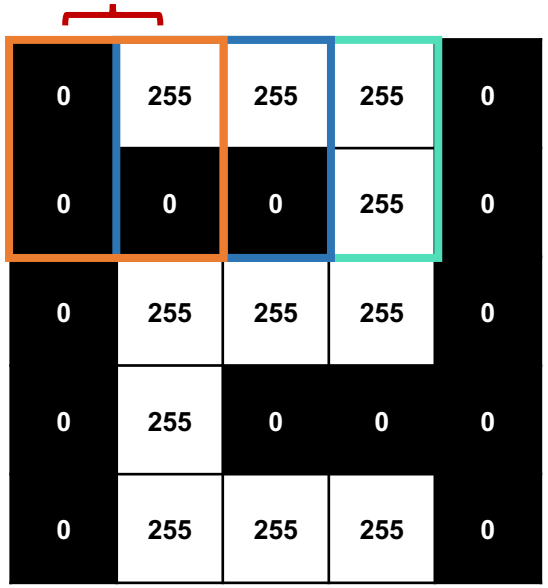
CNN이 이미지에서 특징을 찾아내는 방법



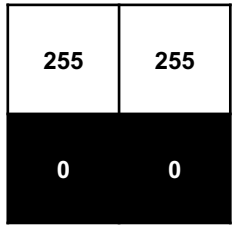
- 모양 + 색(숫자) 데이터를 함께 학습

  → 모양으로만 특징을 추출하면 회색 숫자 이미지의 경우, 다른 데이터라고 인식하게 되므로

stride(간격) = 1



input



Filter

(0*255)+(255*255)+(0*0)+(0*0)

65k	130k	130k	65k
0	0	65k	65k
65k	130k	130k	65k
65k	65k	0	0

Feature map

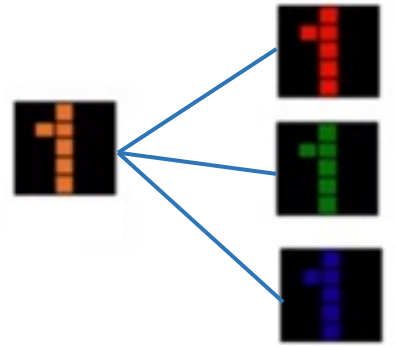
숫자 大 → 해당 영역에 찾고자 하는 특징이 있는 것

숫자 小 → 해당 영역에 찾고자 하는 특징이 없는 것

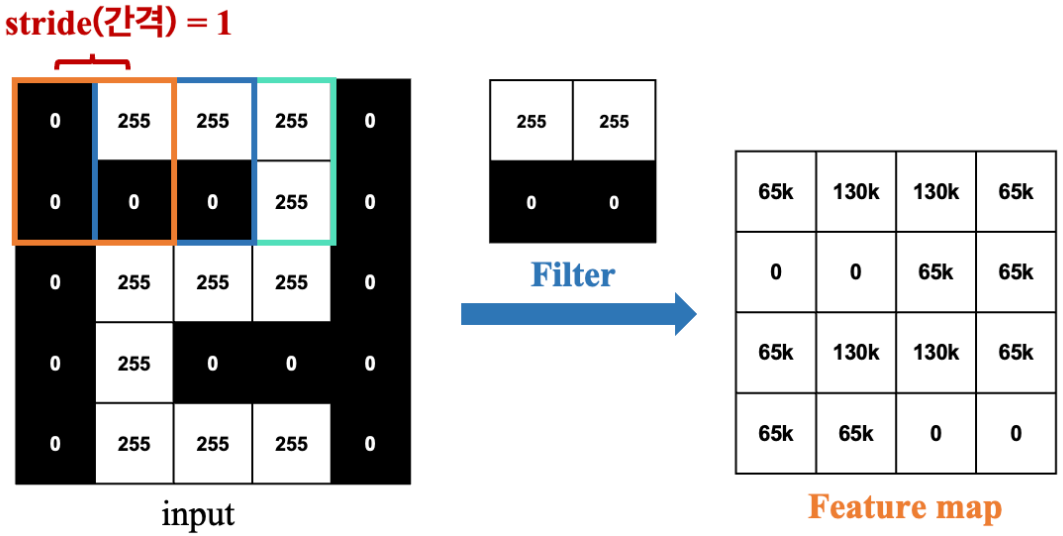
인공 신경망 종류 - Convolutional Neural Networks (CNN)

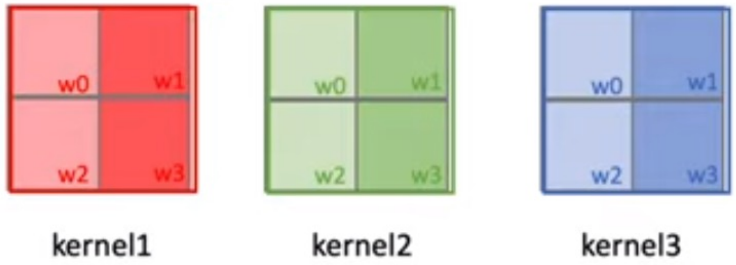
CNN의 주요 용어 정리

- Channel : 컬러 이미지는 3개의 채널로 구성 → Red, Green, Blue
- **Filter** : 이미지의 특징을 찾아내기 위한 공용 파라미터
- Kernel : 이미지 위에서 돌아다니는 2차원 행렬
- **Stride** : 필터가 입력데이터를 순회하는 간격
- Convolution : 2차원 입력 데이터와 필터의 합성곱 연산 → 연산 결과 **Feature map** 생성



컬러 이미지는 1장이 아닌 3장의 이미지임





수직선 특징을 찾기 위한 1개의 Filter와

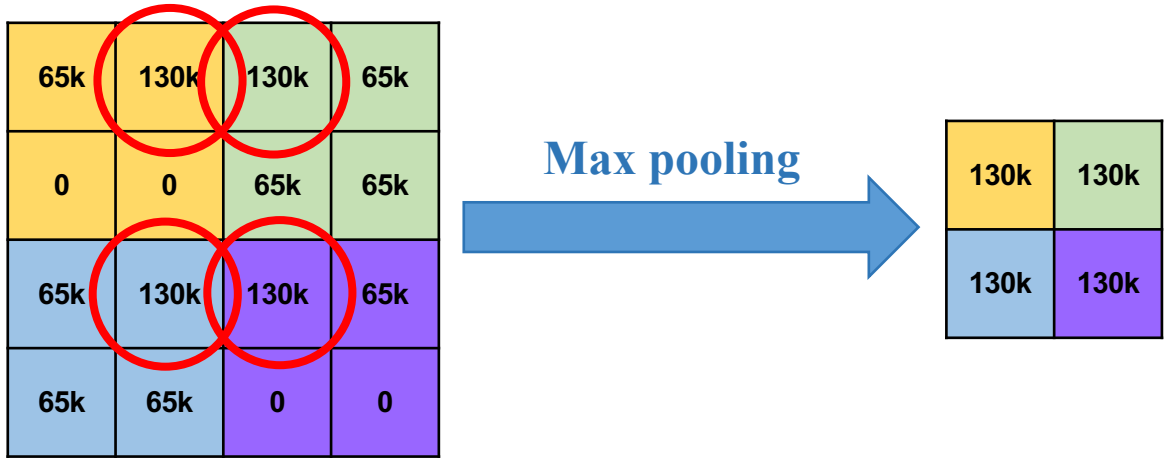
RGB를 돌아다닐 3개의 Kernel

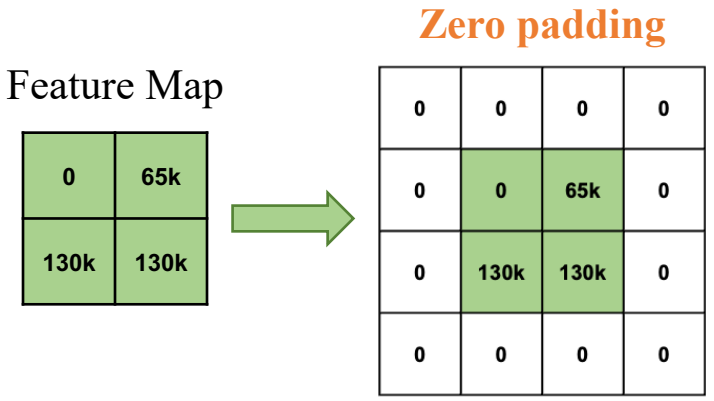
인공 신경망 종류 - Convolutional Neural Networks (CNN)

CNN의 주요 용어 정리

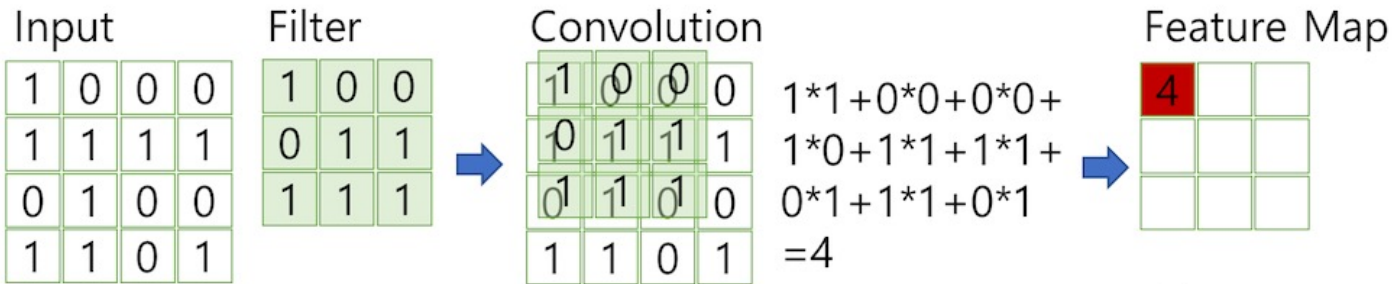
- **Max Pooling** : 파라미터의 개수를 줄임 → 메모리 사용량 감소, 속도 증가, Overfitting의 위험 감소
- **Zero padding** : Convolution Layer의 출력인 Feature Map의 크기가 줄어드는 것 방지

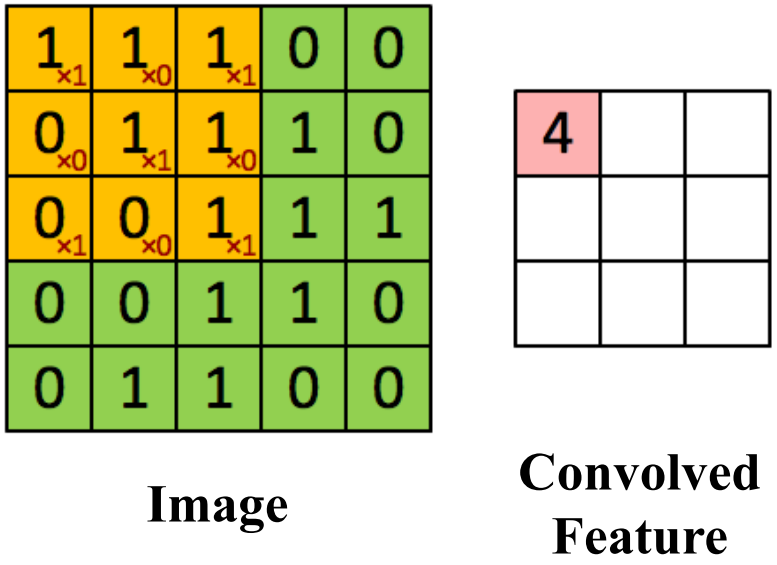
  Zero padding 결과 → 정보의 손실이 줄어듦, 이미지 외곽에 대한 정보를 모델에게 알려주는 효과





인공 신경망 종류 - Convolutional Neural Networks (CNN)





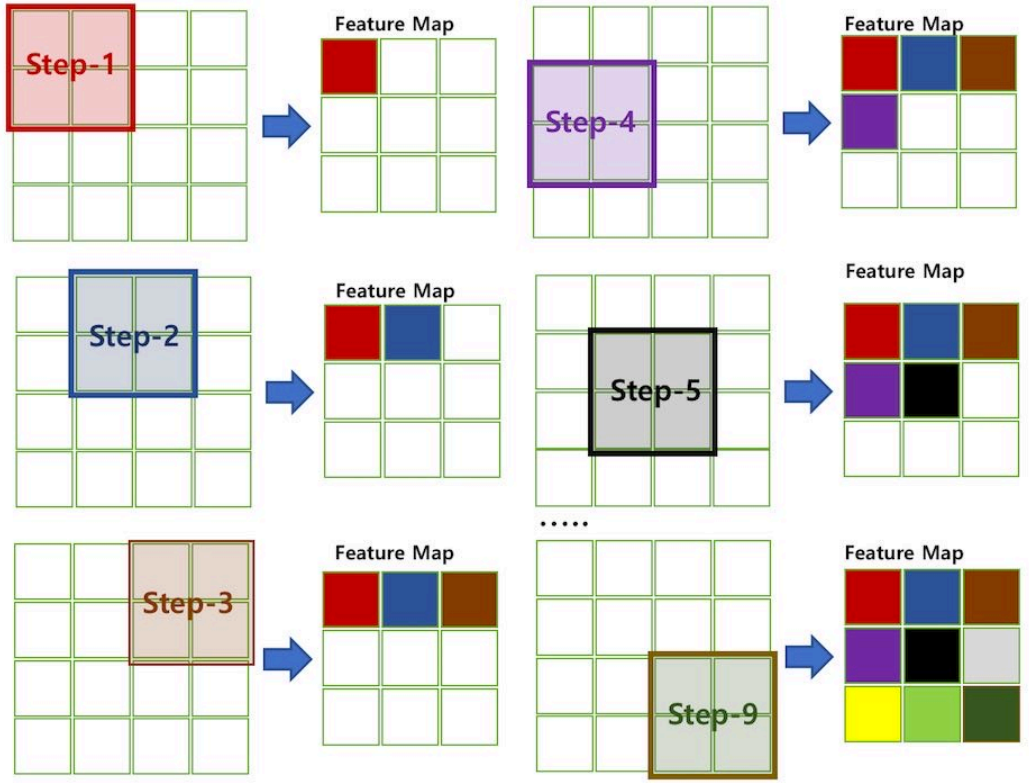


그림 출처
http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution
http://taewan.kim/post/cnn/

인공 신경망 종류 - Convolutional Neural Networks (CNN)

프레임워크별 CNN 모델 구조 코드 [Pytorch / Tensorflow]

Pytorch

```
class CNN(nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
```

Tensorflow

```
# CNN 모델 정의
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten()) # Dense layer에 입력하기 전에 3D -> 1D로 펼쳐야 함
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

감사합니다.