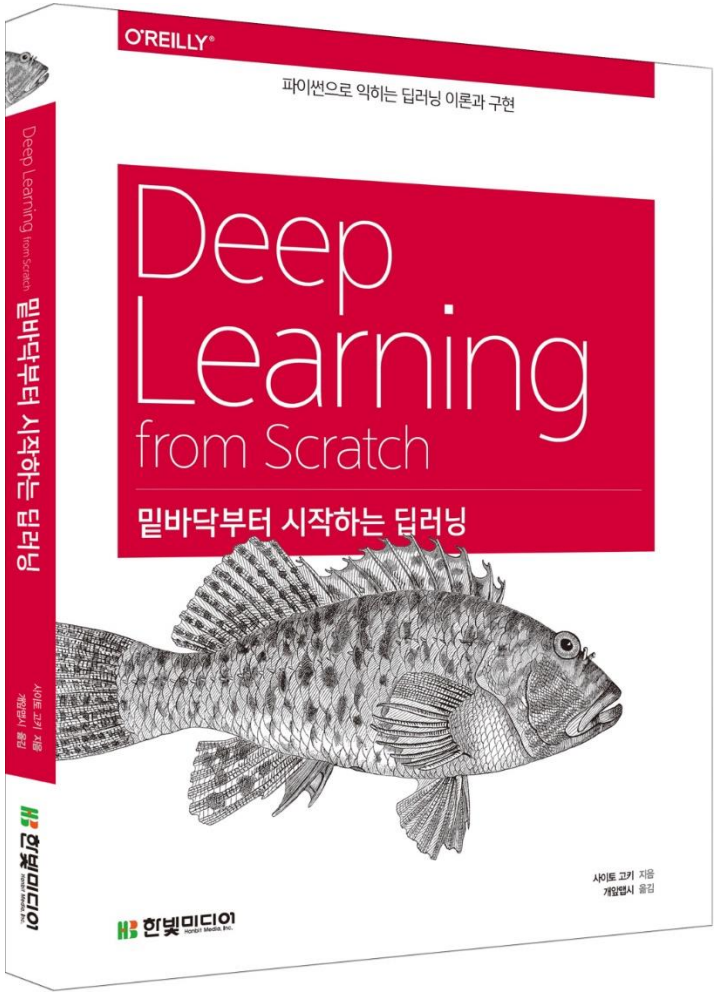
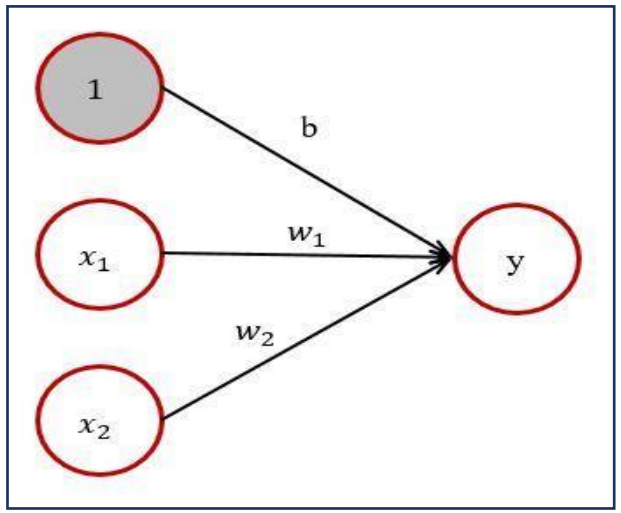


밑바닥부터 시작하는 딥러닝

통계학과 구병모

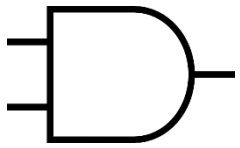


퍼셉트론

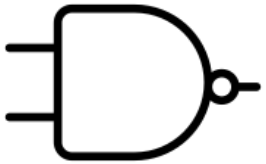
- 개념 : 가중치와 편향을 매개변수로 한 입출력을 갖춘 알고리즘
- 단층 퍼셉트론 : AND 게이트, NAND 게이트, OR 게이트와 같은 선형적 논리회로 구현
- 다층 퍼셉트론 : XOR 게이트와 같은 비선형적 논리회로 구현

논리 회로

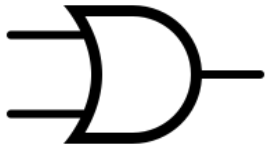
x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1



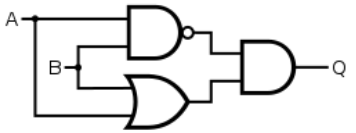
x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0



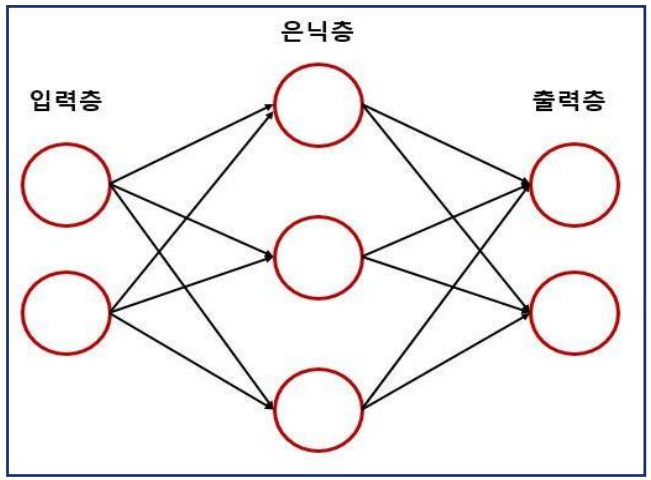
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1



x_1	x_2	x'_1	x'_2	Y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0



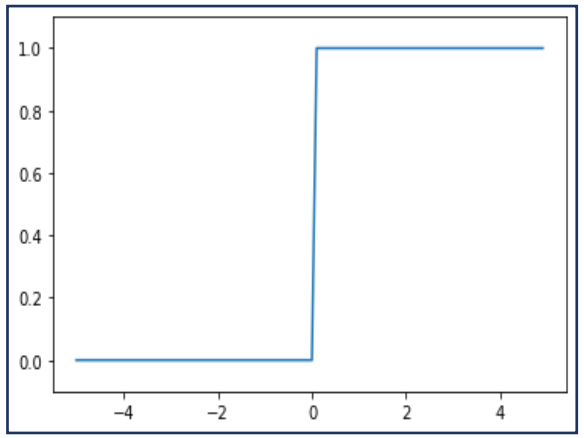
$Q=(A+B)\cdot(\bar{A}+\bar{B})$



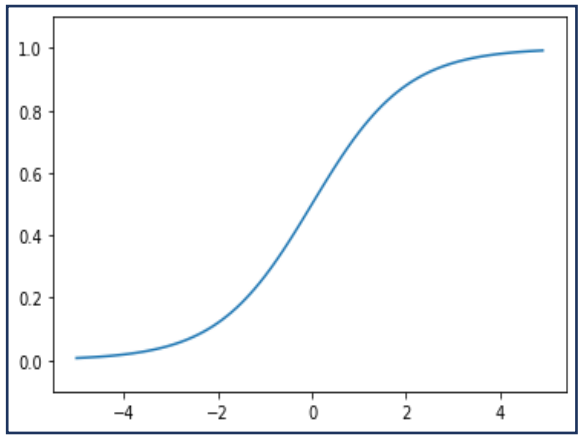
신경망

- 퍼셉트론과의 차이: 퍼셉트론은 계단 함수 / 신경망은 시그모이드 함수, Relu 함수, 소프트맥스 함수
- 분류(소프트맥스 함수), 회귀(항등 함수) 모두에 이용

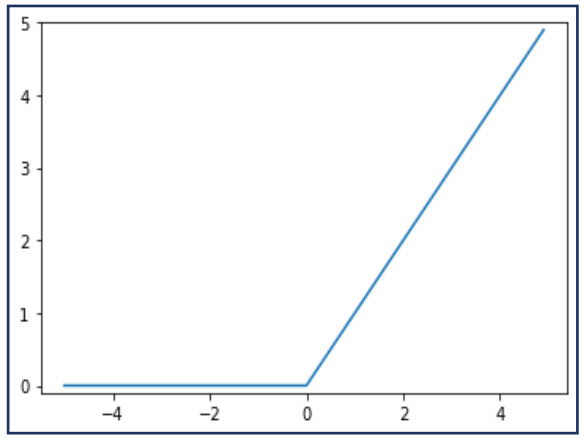
활성화 함수 : 입력 신호의 총합을 출력 신호로 변환하는 함수(입력 신호의 총합이 활성화를 일으키는지 결정, $y = h(b + w_1x_1 + w_2x_2)$)



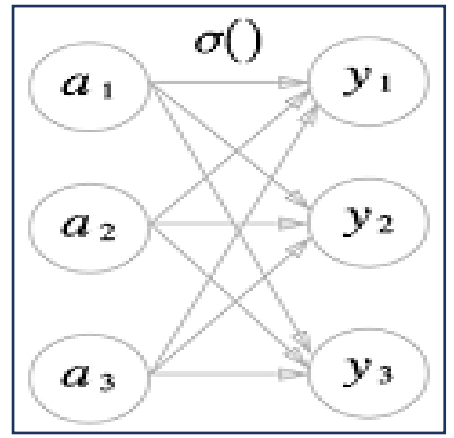
$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$



$$h(x) = \frac{1}{1 + exp(-x)}$$

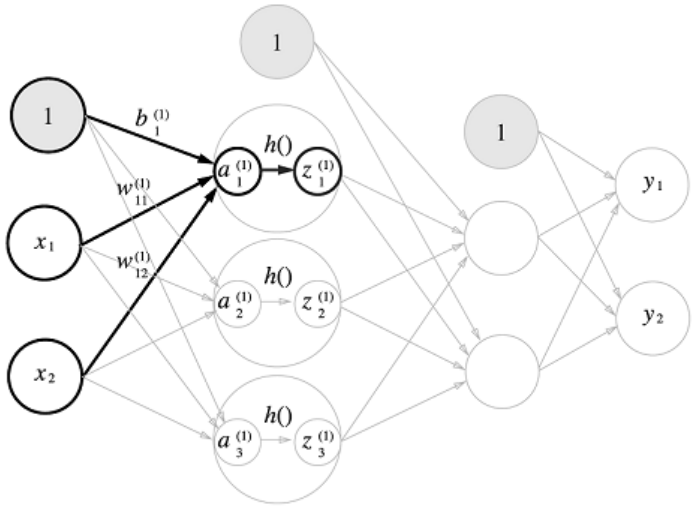


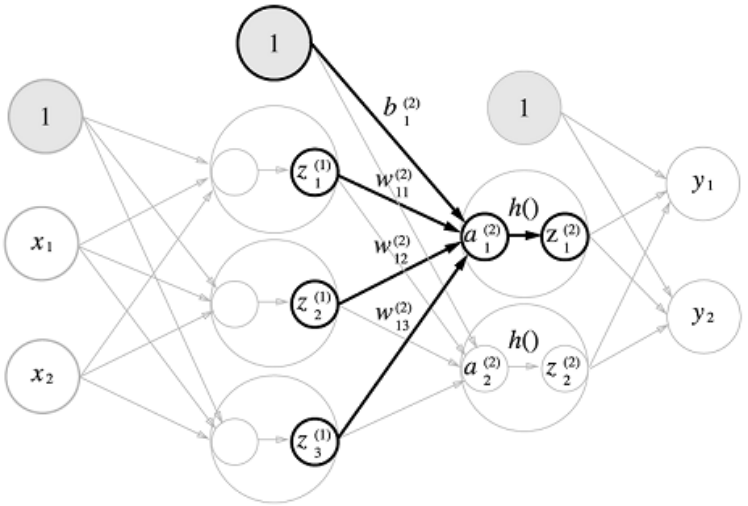
$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

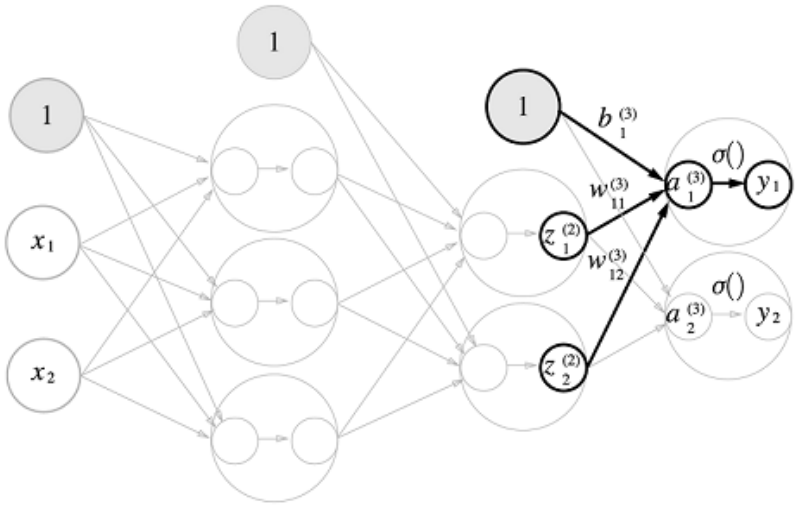


$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n}(a_i)}$$

- 3층 신경망 구현







$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

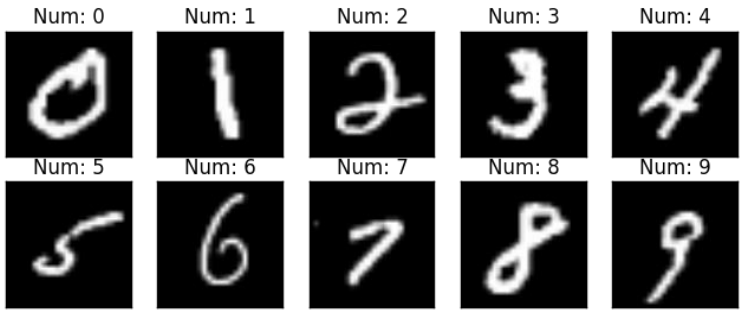
$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

$$A^{(1)} = \begin{pmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{pmatrix} \quad X = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$$

$$B^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{pmatrix} \quad W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

- 입력층 → 1층, 1층 → 2층 : 시그모이드 함수 / 2층 → 출력층 : 항등 함수

✓ 손글씨 숫자 인식(MNIST)



1. 훈련 데이터(학습 데이터)를 사용해 가중치 매개변수를 학습
2. 학습한 매개변수를 사용하여 입력 데이터 분류 → 순전파(이미 학습된 매개변수를 사용하는 추론 과정)

- 원-핫 인코딩 : 정답을 뜻하는 원소만 1 나머지는 모두 0 ([0,0,1,0,0,0,0,0,0,])
- 정규화 : 데이터를 특정 범위로 변환하는 처리(0~255 범위 픽셀 → 0.0~1.0 범위로 변환)
- 정확도(Accuracy)는 0.9352로 교재와 동일하게 출력

- 배치: 하나로 묶은 입력 데이터 → 배치 처리 수행으로 큰 배열로 이뤄진 계산을 하여 처리 시간 대폭 줄여 줌

❖ 어려웠던 점 & 궁금한 점

✓ 어려웠던 점

✓ 궁금한 점

- 출력층으로의 활성화 함수는 명확(Sigmoid, Softmax), 은닉층의 활성화 함수는 임의?
- 원-핫 인코딩을 해주는 이유

기계학습 문제는 데이터를 훈련 데이터 & 실험 데이터로 나누어 학습과 실험 수행하는 것이 일반적
→ For 범용 능력 BUT 오버피팅 주의

✓ 손실함수 : 최적의 매개변수 값을 탐색하기 위해 신경망 학습에서 사용하는 지표

- 오차제곱합 : $E = \frac{1}{2}\sum_k (y_k - t_k)^2$
- 교차 엔트로피 오차 : $E = -\sum_k t_k \log y_k$
- 최적의 매개변수(가중치, 편향)를 탐색할 때 손실 함수의 값 가능한 작게 → 미분
- 손실 함수의 미분 : 가중치 매개변수의 값 변화시킬 때 손실 함수의 변화

✓ 미니배치 학습 : 데이터 일부를 골라 학습을 수행(평균 손실함수 : $E = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_k$)

✓ 경사하강법 : 기울기를 이용해 함수의 최솟값을 찾으려는 방법
- η : 학습률(하이퍼파라미터) – 갱신하는 양

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

✓ 학습 알고리즘 구현

- 전제 : 신경망에 적응 가능한 가중치, 편향을 훈련 데이터에 적응하도록 조정하는 과정이 학습
- 1단계 – 미니배치 : 훈련 데이터 중 일부 무작위로 가져옴
- 2단계 – 기울기 산출 : 미니배치의 손실 함수 값을 줄이기 위한 각 가중치 매개변수의 기울기 구함
- 3단계 – 매개변수 갱신 : 가중치 매개변수를 기울기 방향으로 조금씩 갱신
- 4단계 – 반복 : 1~3단계 반복

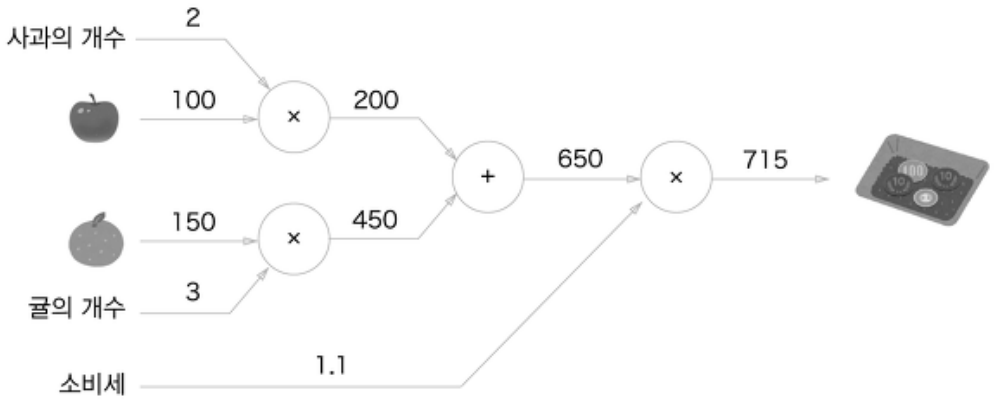
```
iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

for i in range(iters_num):
    # 1단계 - 미니배치
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
    # 2단계 - 기울기 산출
    grad = network.gradient(x_batch, t_batch)
    # 3단계 - 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)
```

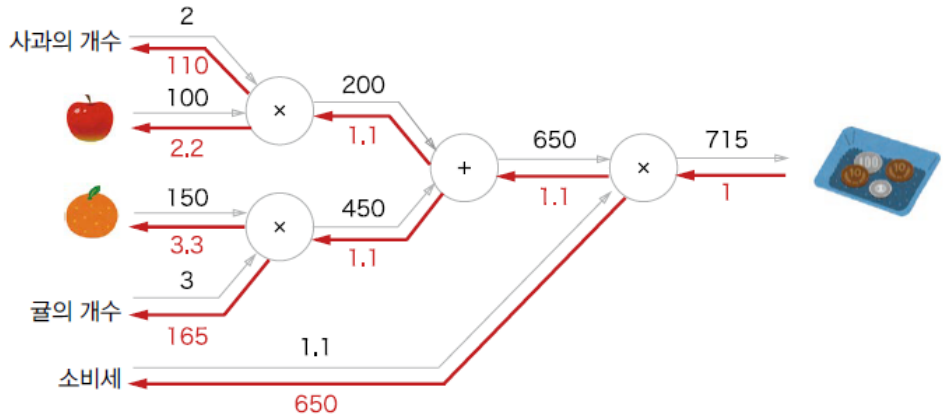
평가 결과

```
train acc, test acc |0.11236666666666667,0.1135
train acc, test acc |0.781,0.788
train acc, test acc |0.8771,0.8801
train acc, test acc |0.8991833333333333,0.903
train acc, test acc |0.9095,0.9129
train acc, test acc |0.9163666666666667,0.9181
train acc, test acc |0.9212833333333333,0.9231
train acc, test acc |0.9257333333333333,0.9259
train acc, test acc |0.9295333333333333,0.9293
train acc, test acc |0.93275,0.932
train acc, test acc |0.9357,0.9352
train acc, test acc |0.9378,0.9378
train acc, test acc |0.9402333333333334,0.9395
train acc, test acc |0.9418,0.9413
train acc, test acc |0.9432,0.9419
train acc, test acc |0.9462666666666667,0.9453
train acc, test acc |0.94765,0.9471
```

✓ 계산 그래프

- 노드(node)와 에지(edge)로 표현
- 국소적 계산 → 국소적 미분 for 역전파 → 연쇄법칙(합성함수 미분)





✓ 역전파

- 덧셈 노드 : 1 곱하기 / 곱셈 노드 : 서로 바꾼 값 곱하기
- Relu 계층 : x가 0보다 크면 그대로, 0보다 작으면 보내지 않기
- 시그모이드 계층 : $y^2 \exp(-x) = y(1-y)$
- Affine 계층 : 행렬 곱(dot)의 역전파 → 차원의 원소 수가 일치하도록 곱 조립
- Softmax-with-Loss 계층 : Softmax 계층의 출력과 정답 레이블의 차분

✓ 오차역전파 신경망 구현

```python
def numerical_gradient(self, x, t):
    # 수치 미분(순전파)
    loss_W = lambda W: self.loss(x, t)
    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    return grads
def gradient(self, x, t):
    self.loss(x, t)
    # 역전파
    dout = 1
    dout = self.lastLayer.backward(dout)
    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)
    grads = {}
    grads['W1'] = self.layers['Affine1'].dW
    grads['b1'] = self.layers['Affine1'].db
    grads['W2'] = self.layers['Affine2'].dW
    grads['b2'] = self.layers['Affine2'].db
    return grads
```

✓ 수치 미분 기울기와 오차역전파법 기울기의 오차
→ 교재의 값보다 오차가 살짝 크다. WHY?

```
W1:4.771259439491962e-10
b1:2.8724007174792817e-09
W2:5.069682924850204e-09
b2:1.398413217071348e-07
```

✓ 4장의 수치미분 방식의 학습보다 2~4% 정도 정확도 증가

```
0.13548333333333334 0.1366
0.9035833333333333 0.9073
0.9215 0.9237
0.9364833333333333 0.9347
0.9433166666666667 0.9424
0.9505333333333333 0.9491
0.9555833333333333 0.9514
0.96135 0.9561
0.9640833333333333 0.9589
0.9661166666666666 0.9612
0.9693166666666667 0.9644
0.9714833333333334 0.9664
0.97315 0.9669
0.9746166666666667 0.9664
0.9754833333333334 0.9697
0.97805 0.9692
0.9787833333333333 0.9691
```

❖ 어려웠던 점 & 궁금한 점

✓ 어려웠던 점

- Softmax-with-Loss 계층의 오차역전파법 도출 과정

✓ 궁금한 점

- 역전파는 미분을 효율적으로 계산가능한데 정확도 증대와 관련이 있는건지?

✓ 확률적 경사 하강법(SGD)

- $W \leftarrow W - \eta \frac{\partial L}{\partial W}$, W = 가중치 매개변수, η = 학습률, $\frac{\partial L}{\partial W}$ = 손실 함수의 기울기
- 단점 : 비등방성 함수에서 탐색 경로가 비효율적

✓ 모멘텀

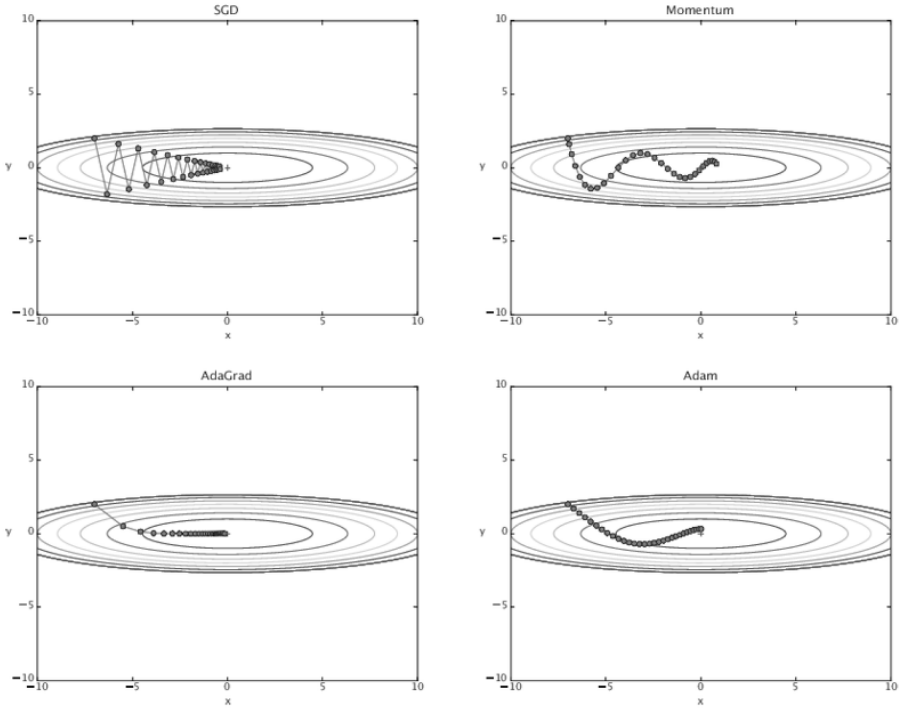
- $\begin{cases} \alpha \leftarrow \alpha v - \eta \frac{\partial L}{\partial W} \\ W \leftarrow W + v \end{cases}$, v = 속도 변수
- SGD에 비해 갱신 경로의 지그재그 움직임이 덜함 → 안정적으로 갱신

✓ AdaGrad

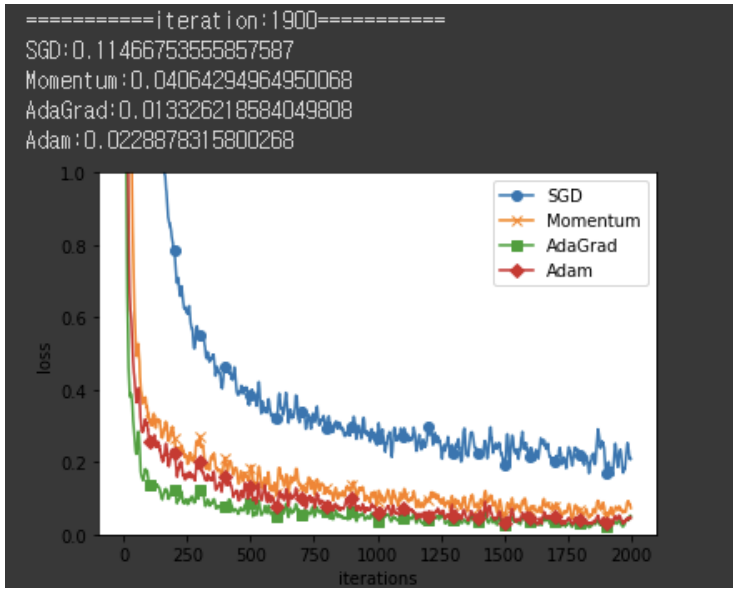
- 학습률 감소가 매개변수의 원소마다 다르게 적용
- $\begin{cases} h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W} \\ W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W} \end{cases}$, h = 기존 기울기 값을 제곱하여 더해준 값으로 학습률을 조정
- 최솟값을 향해 효율적으로 움직임

✓ Adam

- 모멘텀과 AdaGrad 기법을 융합
- 하이퍼파라미터의 '편향 보정'
- $\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ W \leftarrow W - \frac{\eta}{\sqrt{v_t}+\epsilon} m_t \end{cases}$, β_1, β_2 =일, 이차 모멘텀용 계수(기본 설정값 : 0.9, 0.999)



- 4개의 기법 모두 각자의 장단이 있어 적용 문제에 따라서 다름

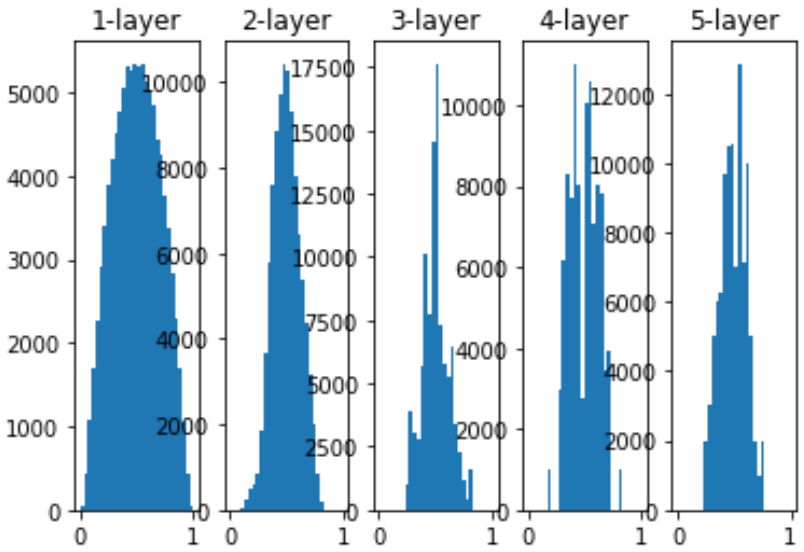


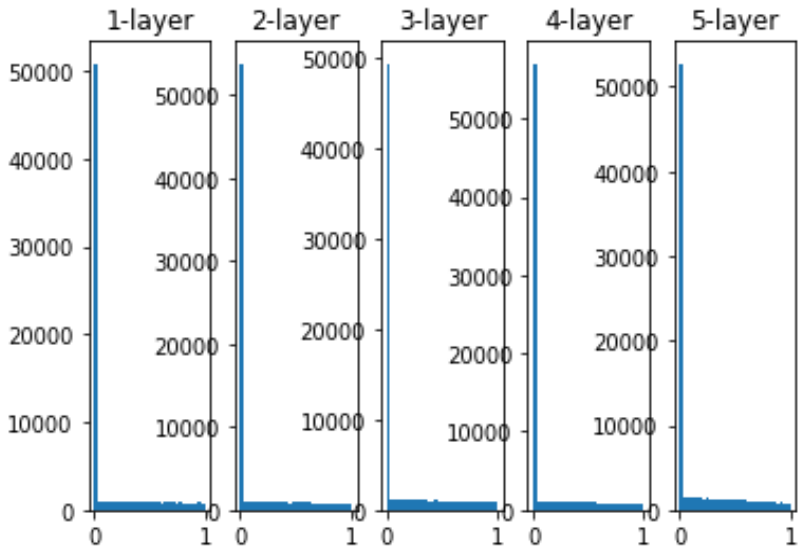
✓ Xavier 초깃값

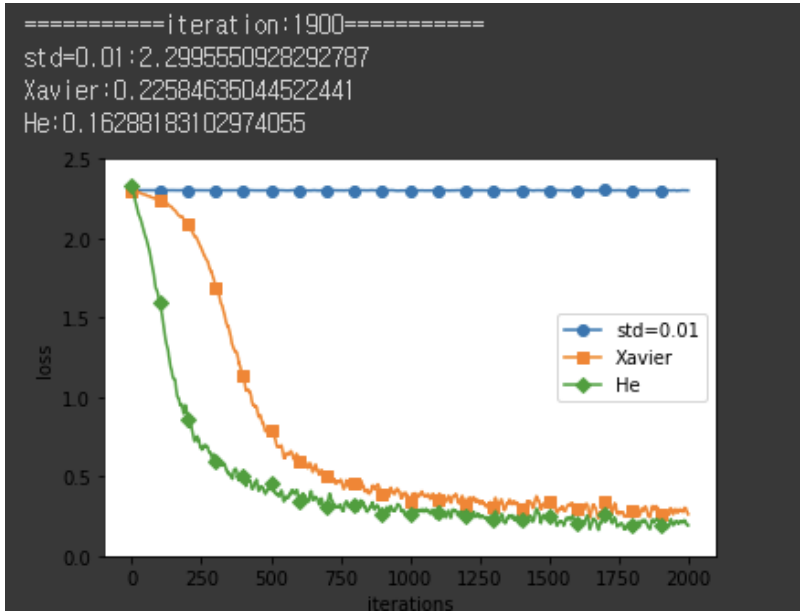
- 초깃값의 표준편차가 $\frac{1}{\sqrt{n}}$ 이 되도록 설정
- 일반적인 딥러닝 프레임워크들의 표준
- 앞 층에 노드가 많을수록 가중치가 좁게 퍼짐
- 시그모이드 or tanh 등의 S자 모양 곡선 함수에 적합

✓ He 초깃값

- 초깃값의 표준편차가 $\sqrt{\frac{2}{n}}$ 이 되도록 설정
- ReLU 함수에 특화된 초깃값
- ReLU의 음의 영역이 0이라서 더 넓은 분포위해 2배의 계수 필요

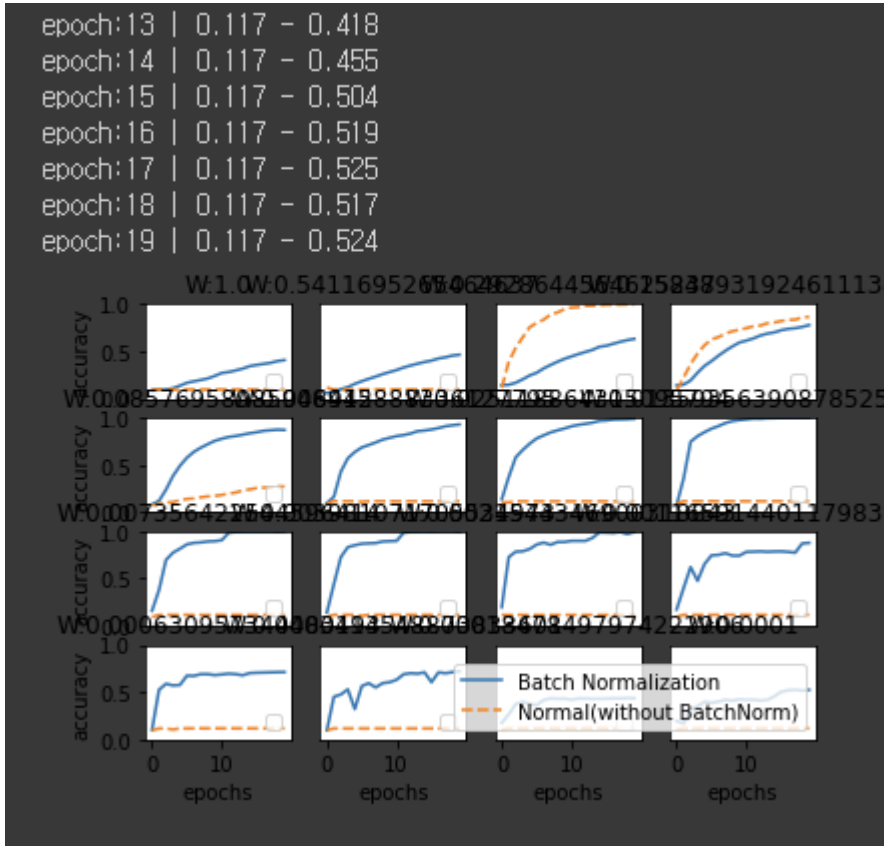






✓ 주목받는 이유
- 학습 속도 개선
- 초깃값 의존도 낮음
- 오버피팅 억제

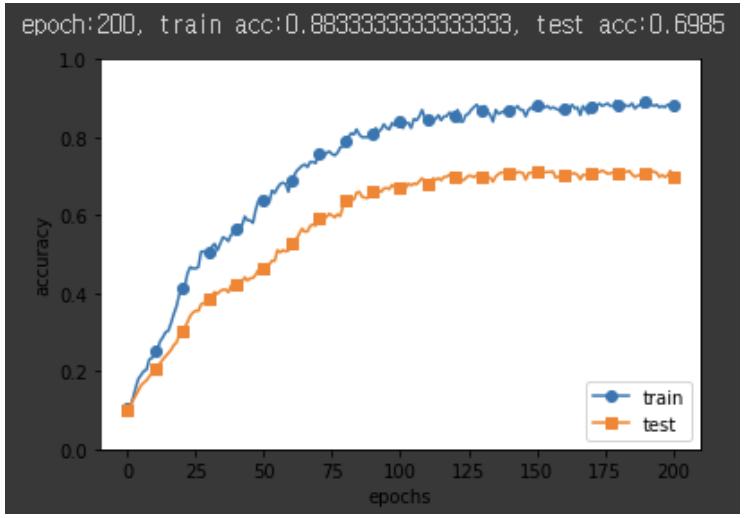
- 활성화 값이 적당히 분포되도록 조정 ← 미니배치를 단위로 정규화
- $$\begin{cases} \mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \\ \hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \end{cases}$$
- 계층마다 정규화된 데이터에 고유한 확대와 이동 변환 수행 : $y_i \leftarrow \gamma\hat{x}_i + \beta$

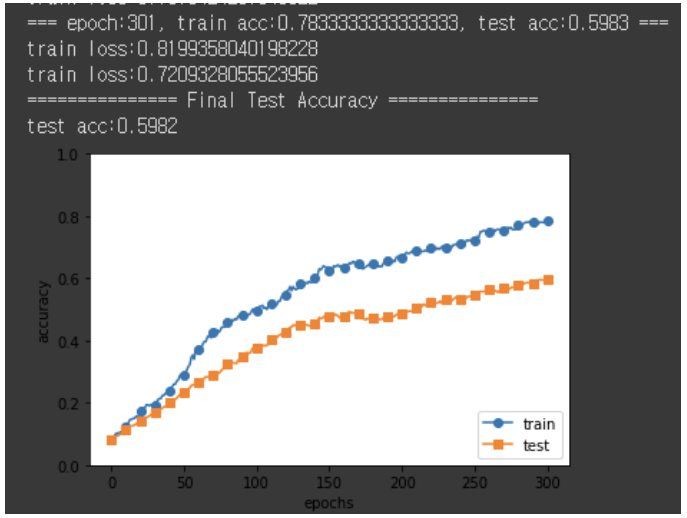


✓ 오버피팅
- 매개변수가 많고, 표현력이 높은 모델 / 훈련 데이터가 적음
- 훈련데이터에만 적응하여 범용성이 떨어지는 문제

✓ 가중치 감소
- 학습 과정에서 큰 가중치에 대해서는 그에 상응하는 큰 페널티 부여
- 가중치의 L2 노름을 손실 함수에 더하는 방법 for 오버피팅 억제

✓ 드롭아웃
- 뉴런을 임의로 삭제하면서 학습하는 방법
- 앙상블 학습과 밀접

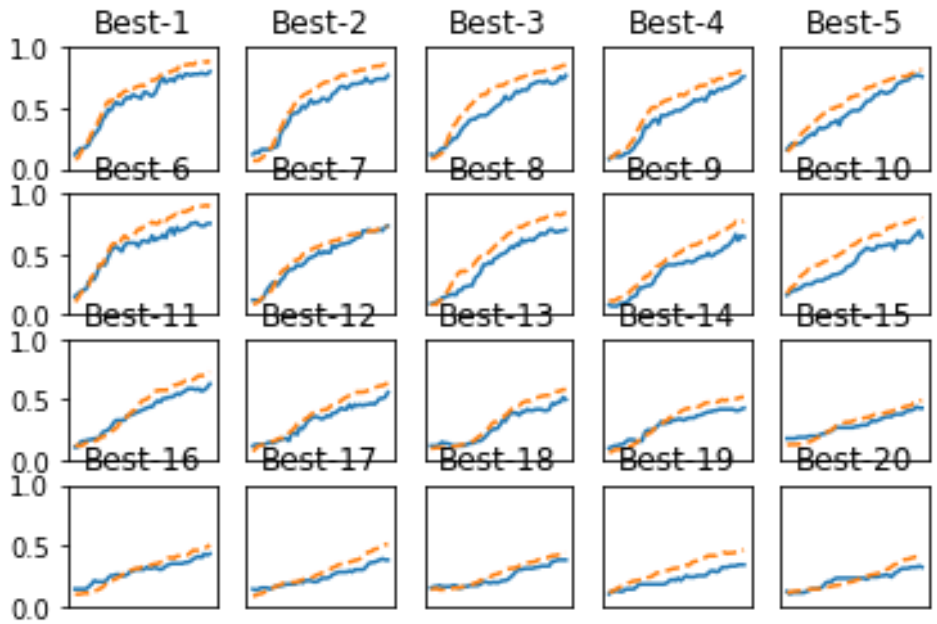




✓ 하이퍼파라미터
- 각 층의 뉴런 수, 배치 크기, 매개변수 갱신 시의 학습률과 가중치 감소
- 하이퍼파라미터를 조정할 때 확인 차 필요한 '검증 데이터' for 범용성
- 최적화 : 하이퍼파라미터의 '최적 값 ' 이 존재하는 범위를 조금씩 줄임

- 0단계 : 하이퍼파라미터 값 범위 설정
- 1단계 : 설정된 범위에서 하이퍼파라미터 값 무작위 추출
- 2단계 : 샘플링한 하이퍼파라미터 값으로 학습, 검증 데이터로 정확도 평가(에폭은 작게 설정)
- 3단계 : 1~2단계 특정 횟수(100회 정도) 반복, 범위 좁히기

```
Best-1(val acc:0.8) | lr:0.009462069422062262, weight decay:5.6575441800631986e-08
Best-2(val acc:0.77) | lr:0.008438814402843884, weight decay:2.467002103061292e-05
Best-3(val acc:0.77) | lr:0.006508724718184309, weight decay:9.607128497357675e-06
Best-4(val acc:0.76) | lr:0.008349863101158034, weight decay:2.000049584216556e-08
Best-5(val acc:0.76) | lr:0.006303767981626423, weight decay:5.534378176934482e-07
Best-6(val acc:0.75) | lr:0.008440790754019576, weight decay:7.099877126250029e-08
Best-7(val acc:0.73) | lr:0.00418222850832874, weight decay:3.7559742162100365e-08
Best-8(val acc:0.7) | lr:0.006224818351259083, weight decay:8.28344034231221e-08
Best-9(val acc:0.64) | lr:0.005069203113012069, weight decay:3.000323888538229e-05
Best-10(val acc:0.64) | lr:0.006120342511014444, weight decay:7.468079406180976e-06
Best-11(val acc:0.63) | lr:0.005620894688547213, weight decay:2.73160129341571e-06
Best-12(val acc:0.56) | lr:0.00329803164489741, weight decay:6.717224967110407e-07
Best-13(val acc:0.5) | lr:0.0035002095828783656, weight decay:5.511650265809857e-07
Best-14(val acc:0.43) | lr:0.002958234232085654, weight decay:1.531110943092176e-05
Best-15(val acc:0.43) | lr:0.002218552457091905, weight decay:1.0809635152062981e-07
Best-16(val acc:0.43) | lr:0.002232118967457657 3, weight decay:8.754810597044208e-05
Best-17(val acc:0.38) | lr:0.002694560284050588, weight decay:3.020378332165409e-05
Best-18(val acc:0.38) | lr:0.002042097436248571, weight decay:6.348999903881124e-08
Best-19(val acc:0.34) | lr:0.0024961388019703566, weight decay:8.742335403071123e-08
Best-20(val acc:0.32) | lr:0.001526077287334932, weight decay:5.117148902395264 5e-08
```



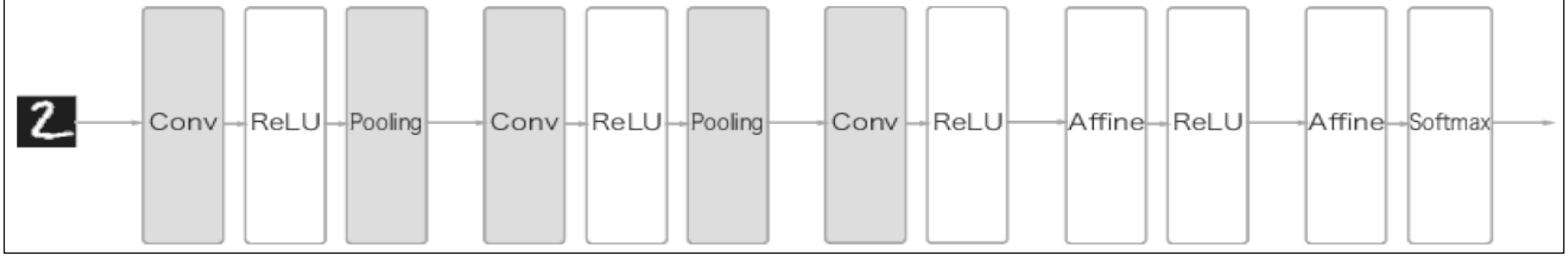
✓ 어려웠던 점

- 가중치 감소 기법
- 하이퍼파라미터 최적화

✓ 궁금한 점

- RMSProp
- Adam 기법에서 하이퍼파라미터의 '편향 보정'
- 배치 정규화 계층마다 정규화된 데이터에 고유한 확대와 이동 변환의 의미 및 목적

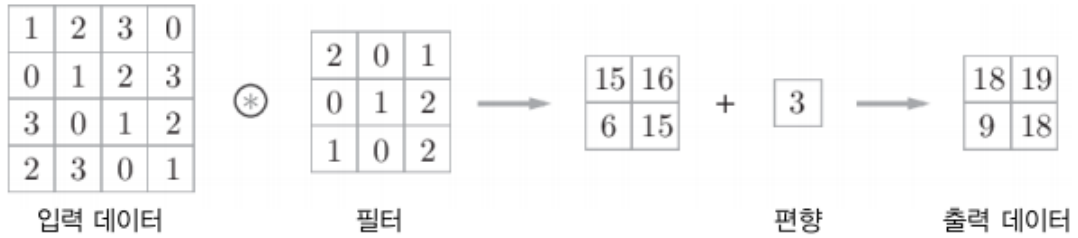
✓ 합성곱 신경망(CNN)
- 이미지 인식 & 음성 인식
- 네트워크 구조 : 합성곱 계층(Convolutional Layer), 풀링 계층(Pooling Layer)
- CNN 계층 : Conv – ReLU – (Pooling) 흐름



✓ 완전연결 계층(Affine)의 문제점
- 데이터의 형상 무시 → 이미지의 경우 : 세로, 가로, 채널(색상) 3차원 데이터
- 1차원 데이터로 평탄화 하여 사용
- 반면, CNN은 형상을 유지 → 3차원 데이터로 입력 받고 3차원 데이터로 전달
- CNN의 입출력 데이터 = 입출력 특징 맵

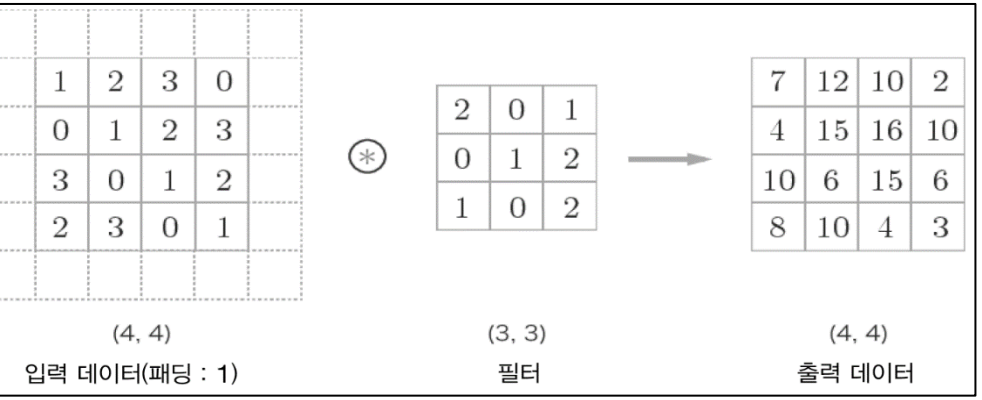
✓ 합성곱 연산

- 필터 연산 using 필터의 윈도우(window) → 단일 곱셈-누산
- 필터 적용 후 편향 합치기



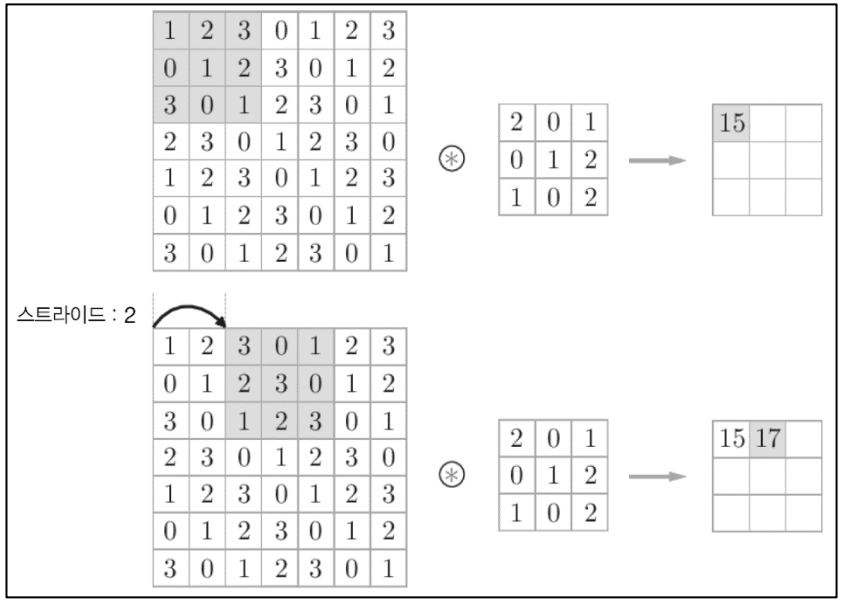
✓ 패딩(Padding)

- 입력 데이터 주변을 특정 값(ex. 0)으로 채우기 for 출력 크기 조정
- 합성곱 연산을 거칠 때마다 작아지는 단점을 보완



✓ 스트라이드(Stride)

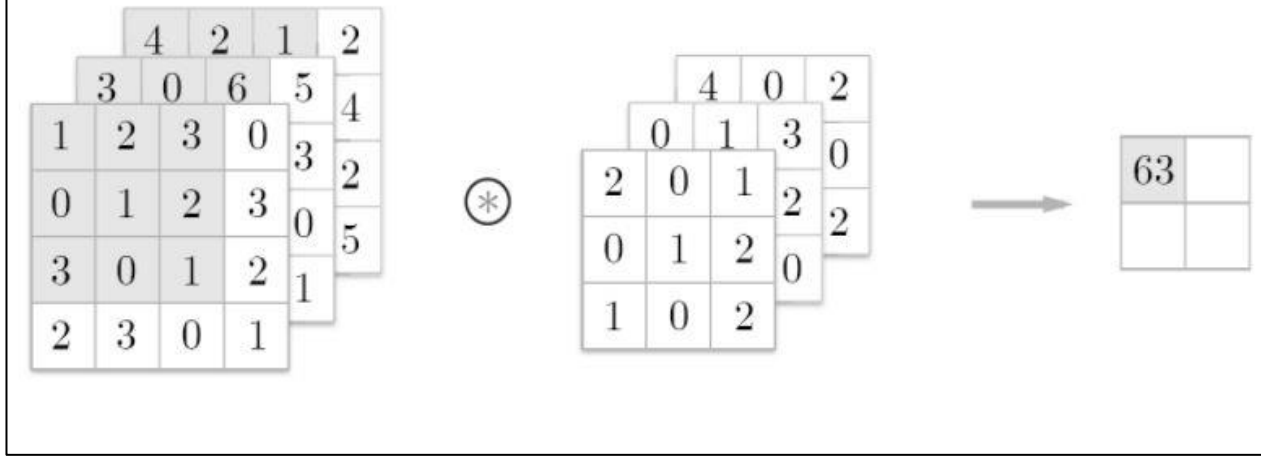
- 필터를 적용하는 간격을 지정

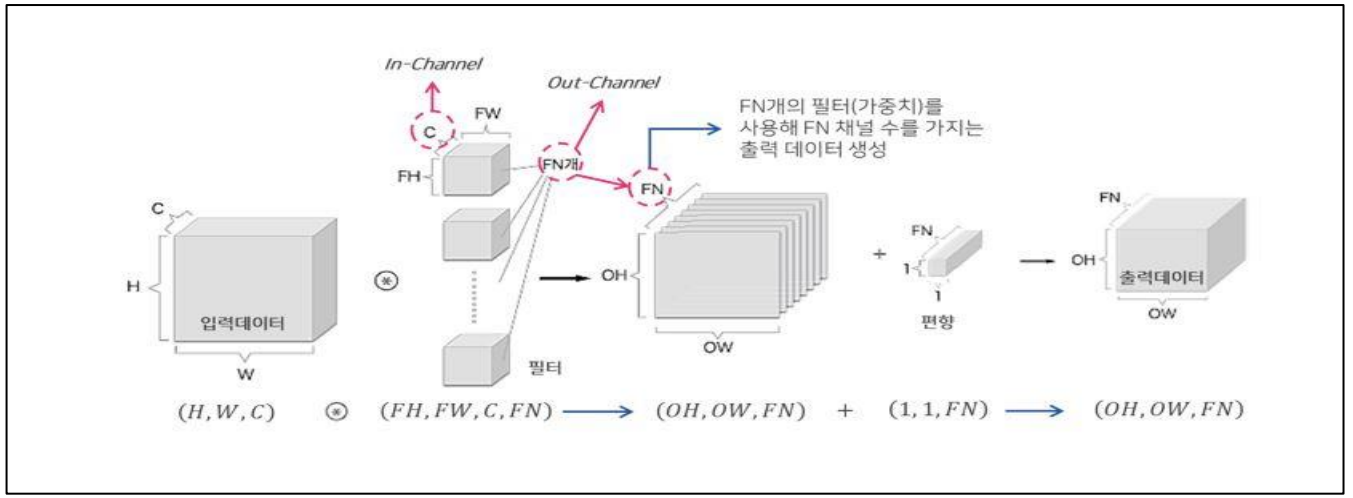


- 패딩을 크게 하면 출력 크기가 커지고, 스트라이드를 크게 하면 출력 크기가 작아진다
- 입력 크기 = (H, W), 필터 크기 = (FH, FW), 출력 크기 = (OH, OW), 패딩 = P, 스트라이드 = S
- 출력 크기 = $\begin{cases} OH = \frac{H+2P-FH}{S} + 1 \\ OW = \frac{W+2P-FW}{S} + 1 \end{cases}$

✓ 3차원 데이터 합성곱 연산

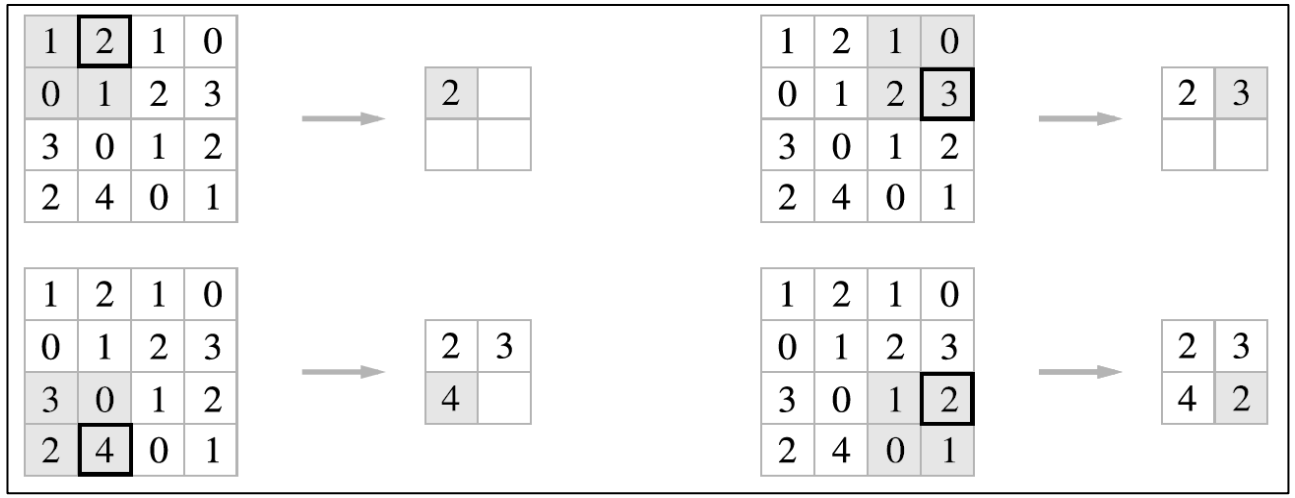
- 입력 데이터와 필터의 합성곱 연산 채널마다 수행한 후, 결과 더해서 하나의 출력
- 직육면체 블록으로 생각 / 필터의 수 → 출력 채널 수 / 미니배치 방식의 학습 가능 → 4차원 데이터





✓ 풀링 계층

- 풀링 : 세로, 가로 방향의 공간을 줄이는 연산
- 2x2 최대 풀링 → 스트라이드 2로 대상 영역의 크기에 속한 가장 큰 원소 출력
- 특징 : 학습해야 할 매개변수가 없음 / 채널 수가 변하지 않음 / 입력의 변화에 영향을 적게 받음(강건)

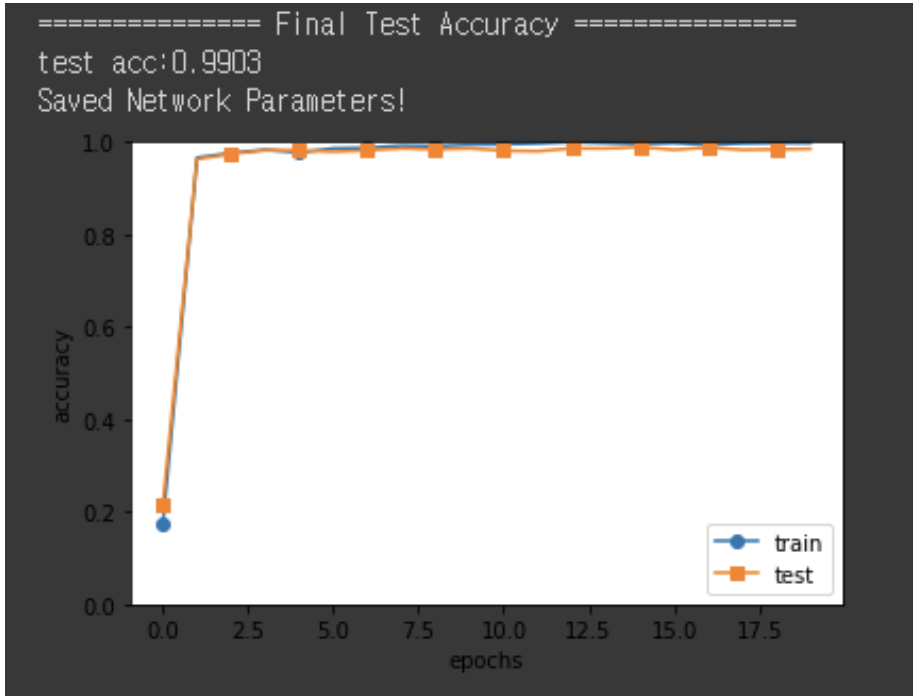


✓ im2col(image to column) 함수

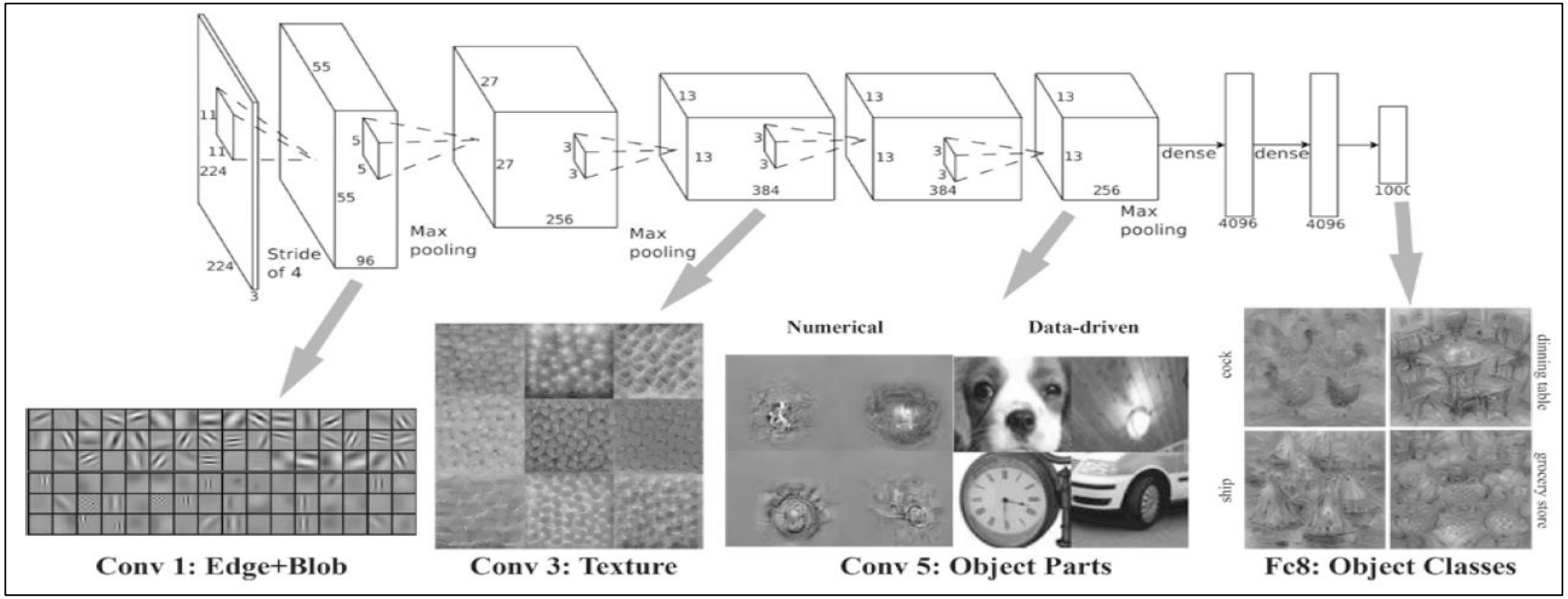
- 성능 떨어지는 for 반복문 대체
- 입력 데이터를 필터링(가중치 계산)하기 좋게 전개하는 함수
- 메모리를 더 많이 소비하는 단점 BUT 큰 행렬 계산이 효율적
- ↔ col2im 함수 : 합성곱 계층의 역전파, 2차원 배열을 이미지 묶음으로 변환

✓ CNN 구현

- 에폭을 20으로 잡고 학습 → 상당히 오랜 시간(약 30분)동안 학습
- 정확도 결과는 교재의 결과 98.96%보다 살짝 더 높은 99.03%
- 확실히 앞선 방식으로 MNIST를 분류한 것보다 높은 정확도



- CNN 계층이 깊어질수록 추출되는 정보는 추상화
- 단순한 에지 → 텍스처 → 복잡한 사물의 일부



- LeNet : CNN계층과 풀링 계층 반복, 마지막은 Affine 계층 / 시그모이드 함수 사용 / 서브샘플링(최대 풀링 대신) / 최초의 CNN
- AlexNet : LeNet과 구조 유사 / 차이점 : ReLU 함수 이용, LRN(국소적 정규화), 드롭아웃

✓ 어려웠던 점

- Im2col, col2im 함수 구현

✓ 궁금한 점

- LeNet의 서브샘플링은 랜덤으로?
- LRN 국소적 정규화

✓ 데이터 확장
- 입력 이미지를 인위적으로 확장 when 데이터가 몇 개 없는 경우
- 이미지 회전, 세로로 이동, crop(일부 잘라내기), flip(좌우를 뒤집기), 외형 변화(밝기), 스케일 변화(확대, 축소)

✓ 층 더욱 깊게 하기
- 신경망의 매개변수 수가 줄어듦 → 넓은 수용 영역 소화
- 학습의 효율성 증대
- 정보를 계층적으로 분해 및 전달

✓ 초기 역사
- VGG : 합성곱 계층, 풀링 계층으로 구성 & CNN계층과 Affine 계층을 16층(19층)으로 심화 / 3x3 필터 사용한 CNN 계층 연속
- GoogLeNet : 세로 방향 깊이 뿐 아니라 가로 방향도 깊음(인셉션 구조) / 크기가 다른 필터(& 풀링) 여러 개 적용 후 결합 / 1x1 필터 사용 for 채널 쪽으로 크기 줄이기 → 매개변수 제거, 고속 처리 기여
- ResNet : 스킵 연결(입력 데이터를 합성곱 계층을 건너뛰어 출력에 바로 더하는 구조) → 신호 감쇠 막아 층이 깊어져도 학습 효율적 합성곱 계층 2개 층마다 건너뛰면서 층 깊게 함

✓ 딥러닝 활용
- 사물 검출 : R-CNN(후보 영역 추출, CNN 특징 계산 → 클래스 분류)
- 분할(이미지를 픽셀 수준에서 분류 using 지도 데이터, FCN)
- 사진 캡션 생성(RNN – 자연어, 시계열 데이터에서 활용) / 강화학습(Deep Q-Network)

감사합니다