밑바닥부터 시작하는 딥러닝

20213093 정현우

CONTENTS

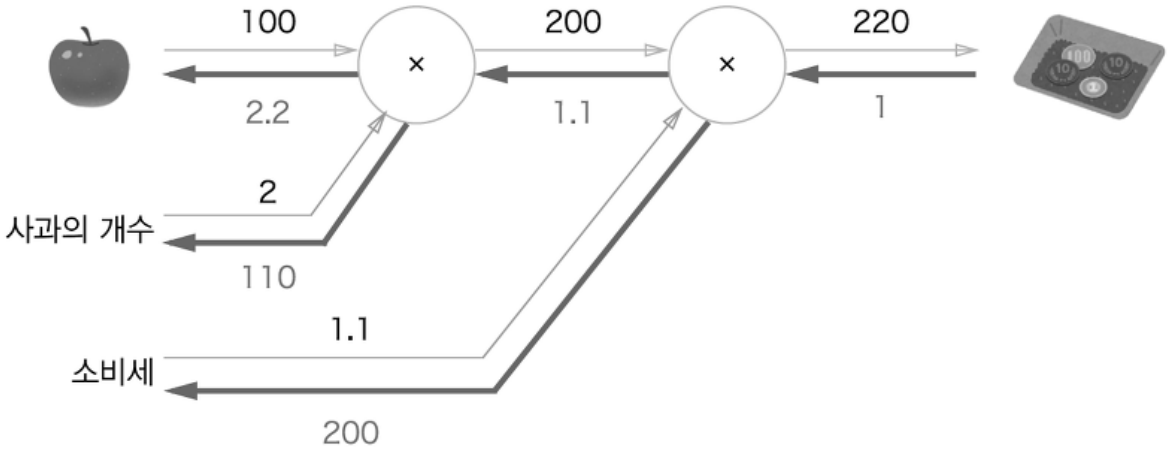
오차역전파법

학습 관련 기술들

합성곱 신경망

1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파법



계산 그래프

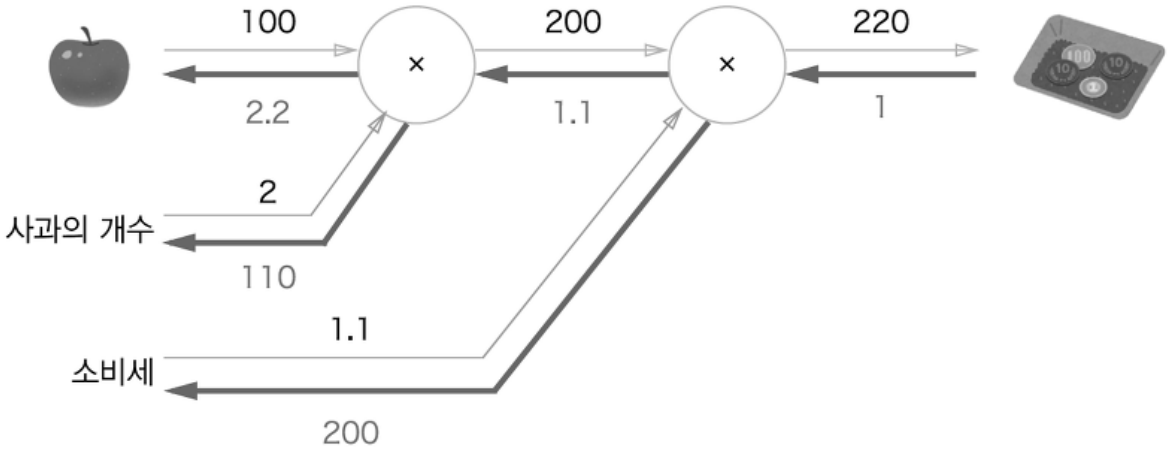
계산 그래프에서 순전파(foward propagation)와 역전파(backword propagation)

순전파: '계산을 왼쪽에서 오른쪽으로 진행'하는 단계를 의미합니다. 순전파는 계산 그래프의 출발점부터 종착점으로의 전파입니다.

역전파: '오른쪽에서 왼쪽으로 진행'하는 것을 의미합니다. 역전파는 이후에 **미분을 계산할 때 중요한 역할**을 합니다.

1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파법



계산 그래프

계산 그래프의 이점

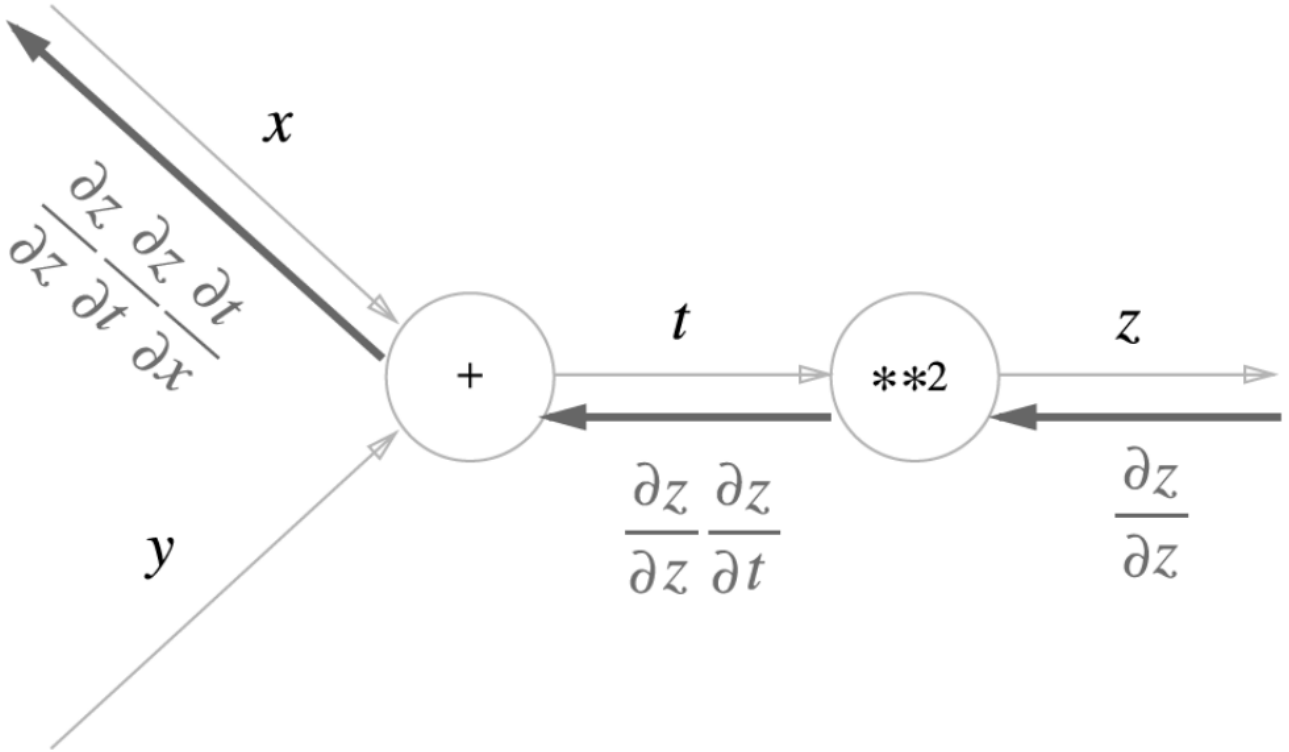
1. 국소적 계산
  전체가 아무리 복잡해도 각 노드에서는 단순한 계산에 집중하여 문제를 단순화 할 수 있습니다.

2. 중간 계산 결과를 모두 보관할 수 있습니다.
 예를 들어 사과 2개까지 계산했을 때의 200원, 소비세를 더하기 전의 금액은 650원인 식으로 중간 결과를 보관합니다.

3. 역전파를 통해 '미분'을 효율적으로 계산할 수 있습니다.

1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파법



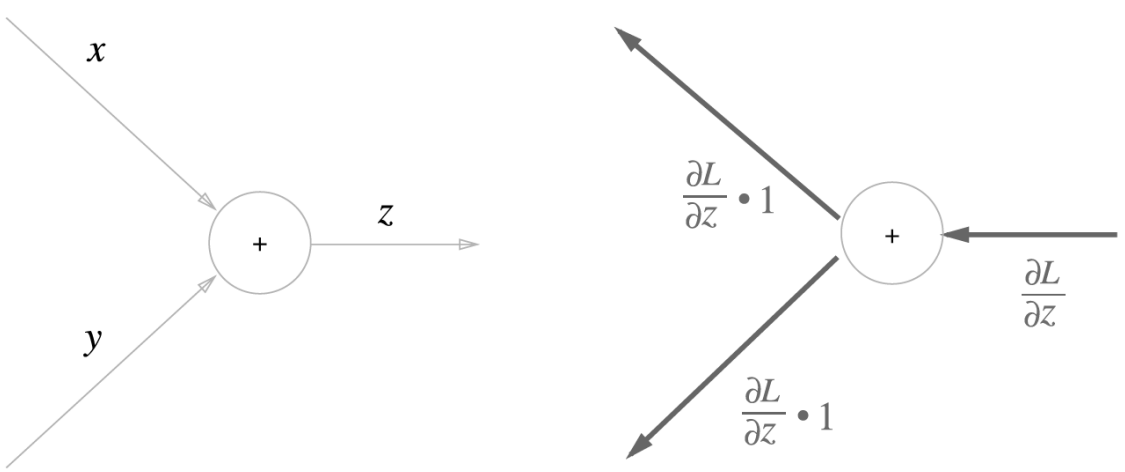
연쇄법칙이란?

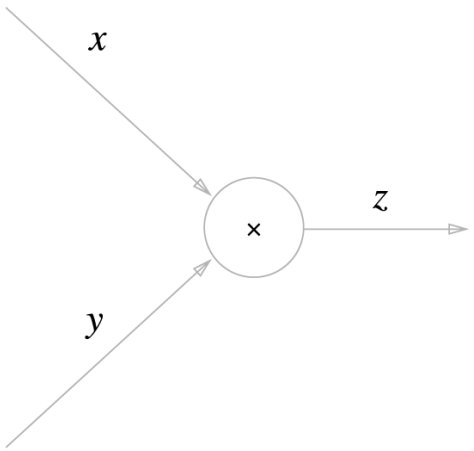
연쇄법칙을 설명하려면 우선 합성 함수 이야기부터 시작해야 합니다. **합성 함수**란 여러 함수로 구성된 함수입니다. 예를 들어 zz = (x+y)2(x+y)2라는 식은 다음의 식처럼 두 개의 식으로 구성됩니다.

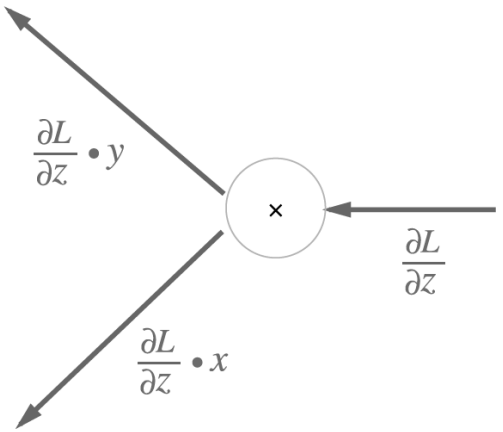
1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파

덧셈 노드와 곱셈 노드의 역전파

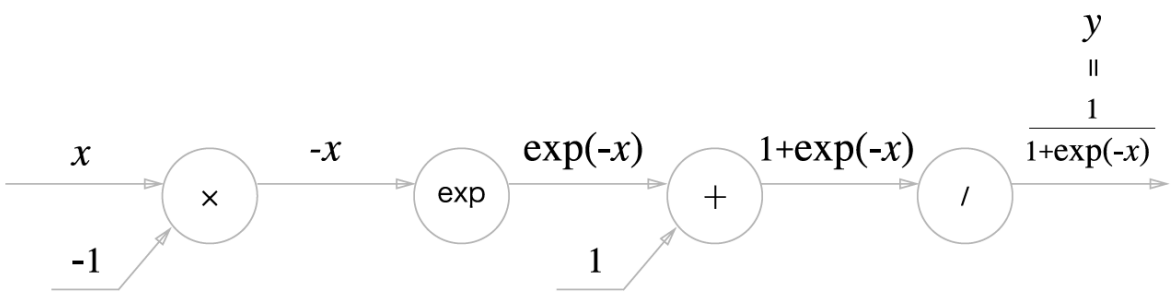




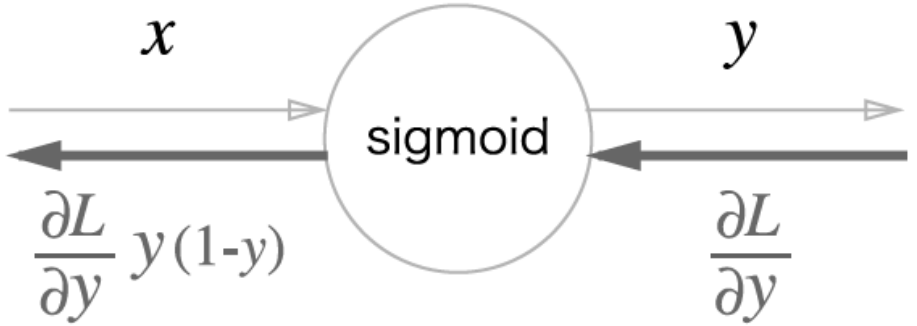


1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파법

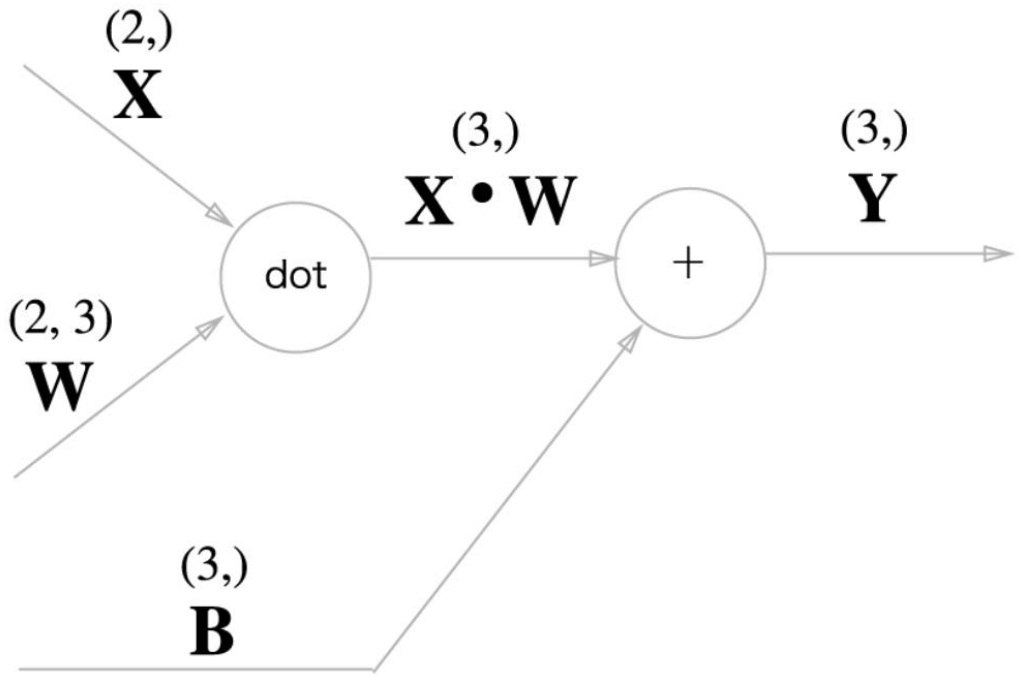


시그모이드 역전파

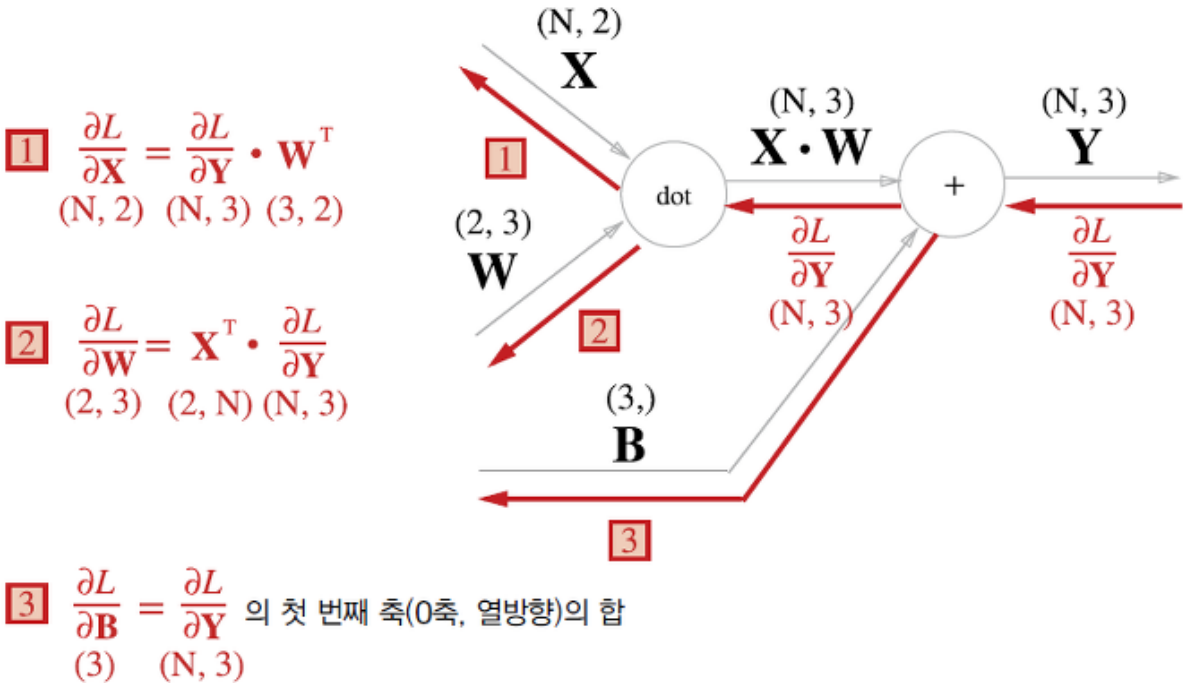


1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파법



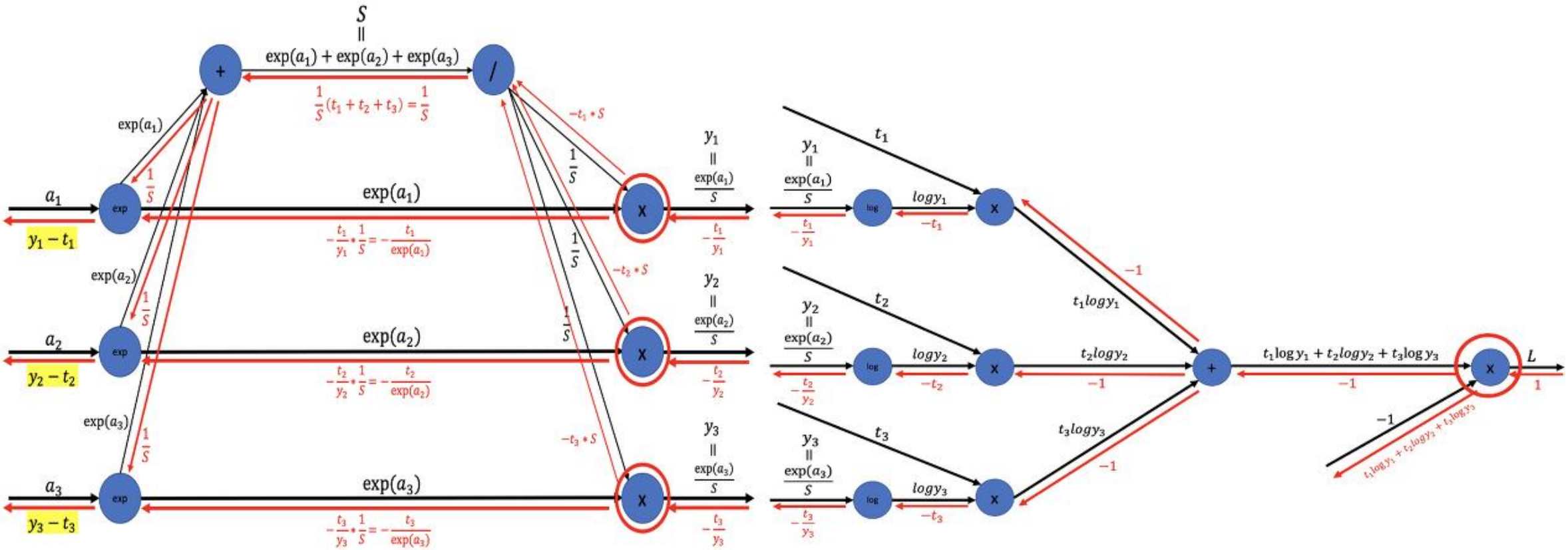
Affine 계층, Afiine계층 역전파



1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파

Softmax-with-Loss 계층과 역전파



1. 오차역전파법

계산 그래프, Affine 계층, 오차 역전파법

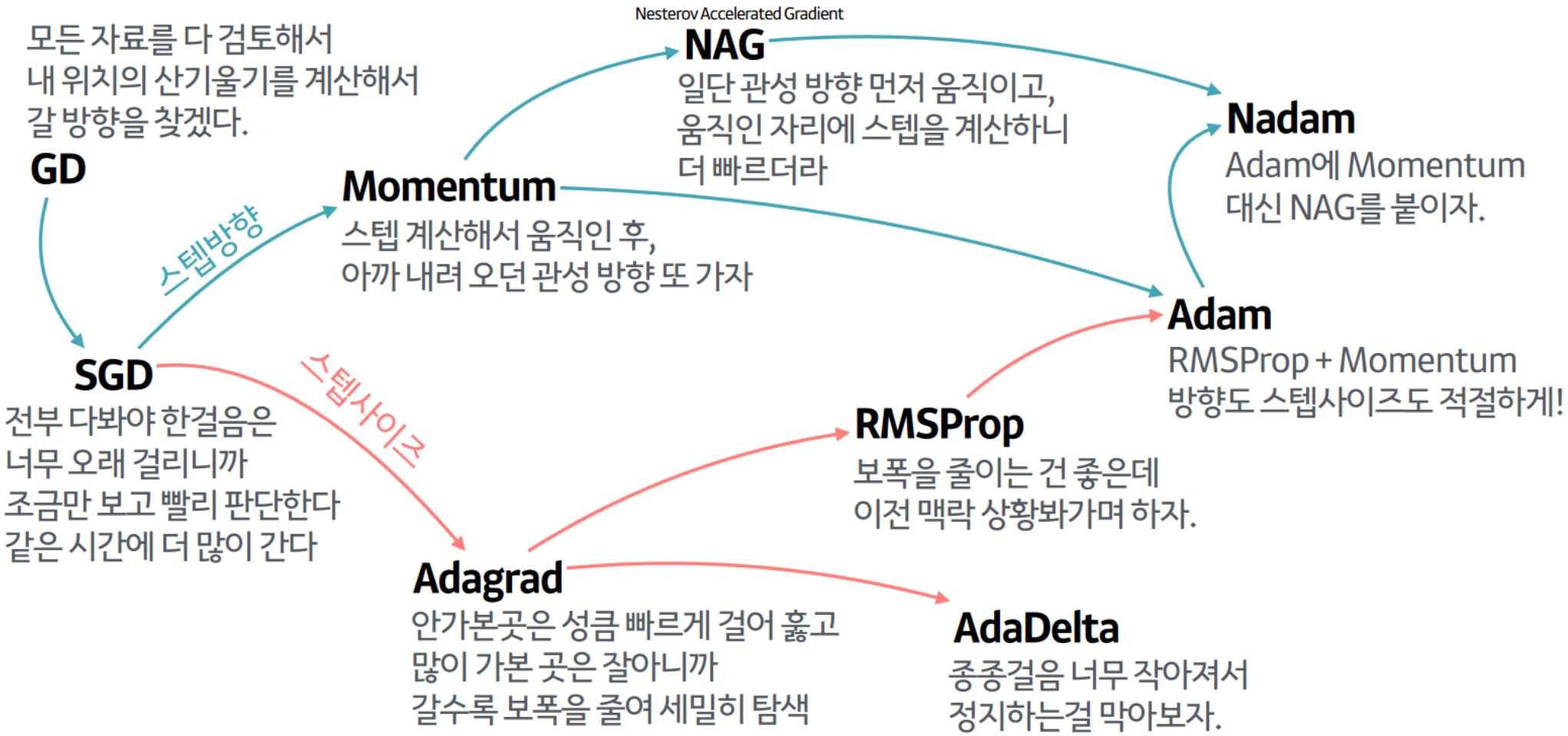
```python
# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    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
```

속도 향상된 기울기 계산

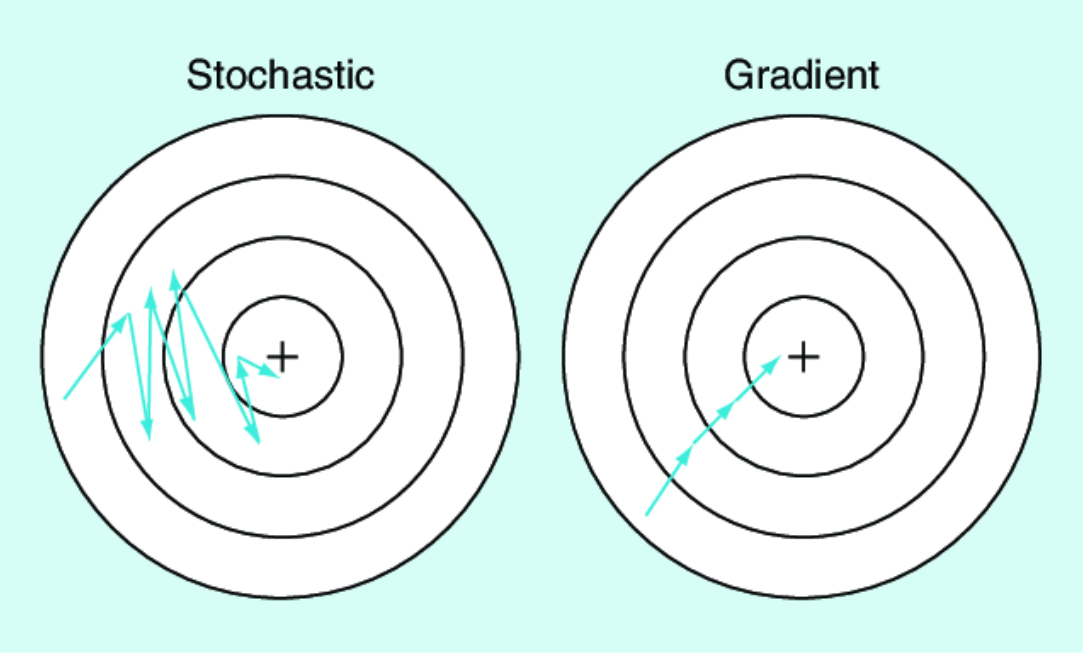
```python
def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads
```

2. 학습 관련 기술들

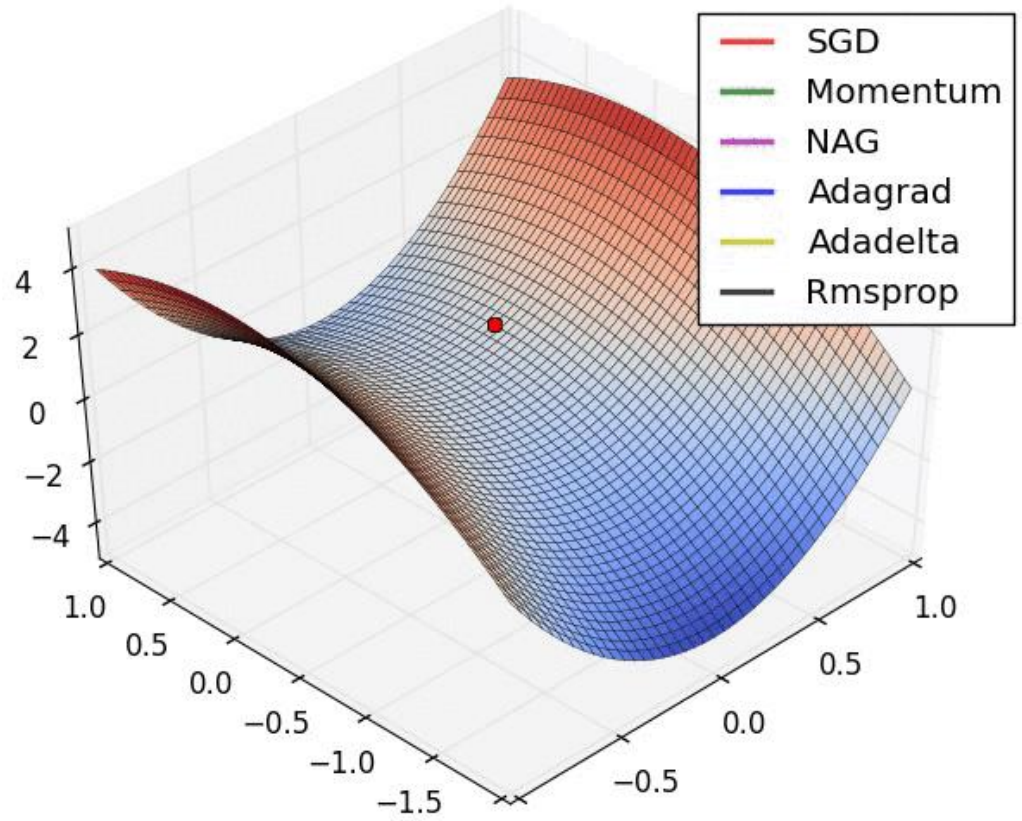
산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



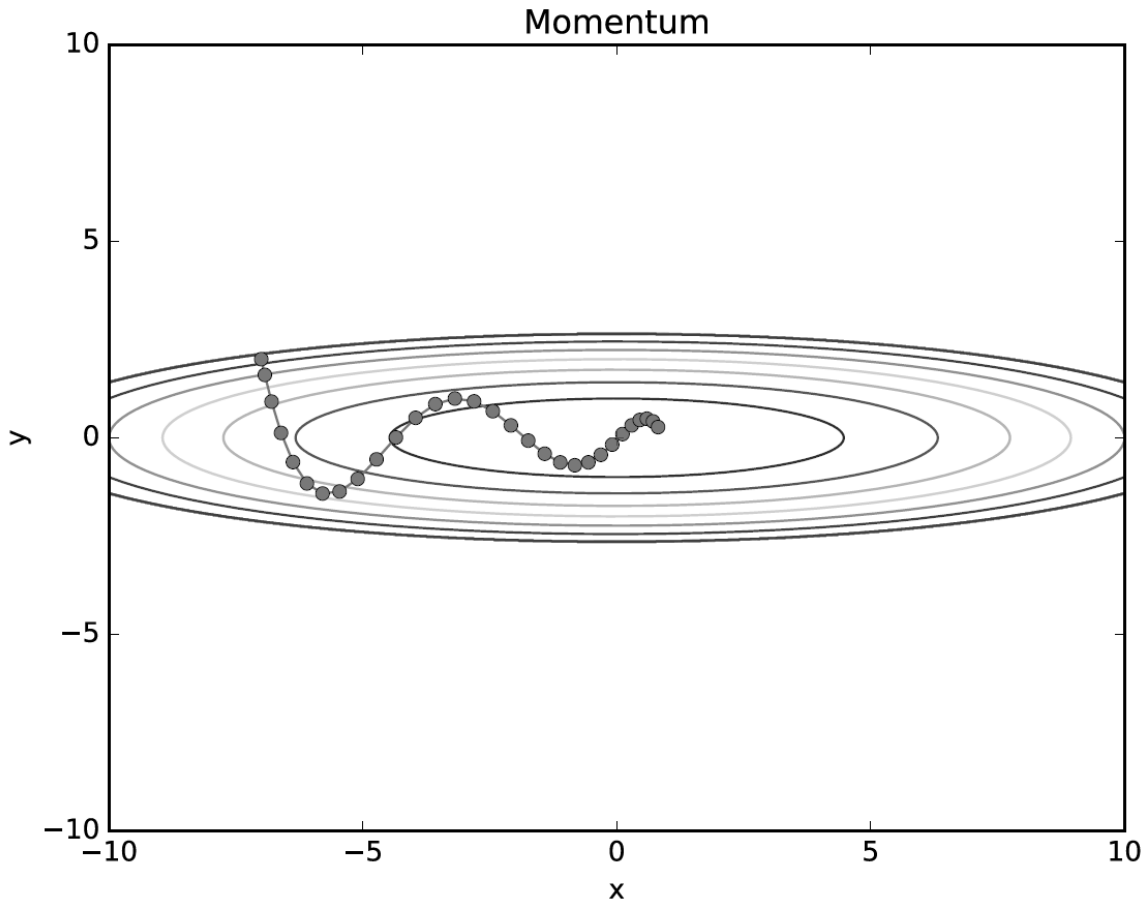
2. 학습 관련 기술들



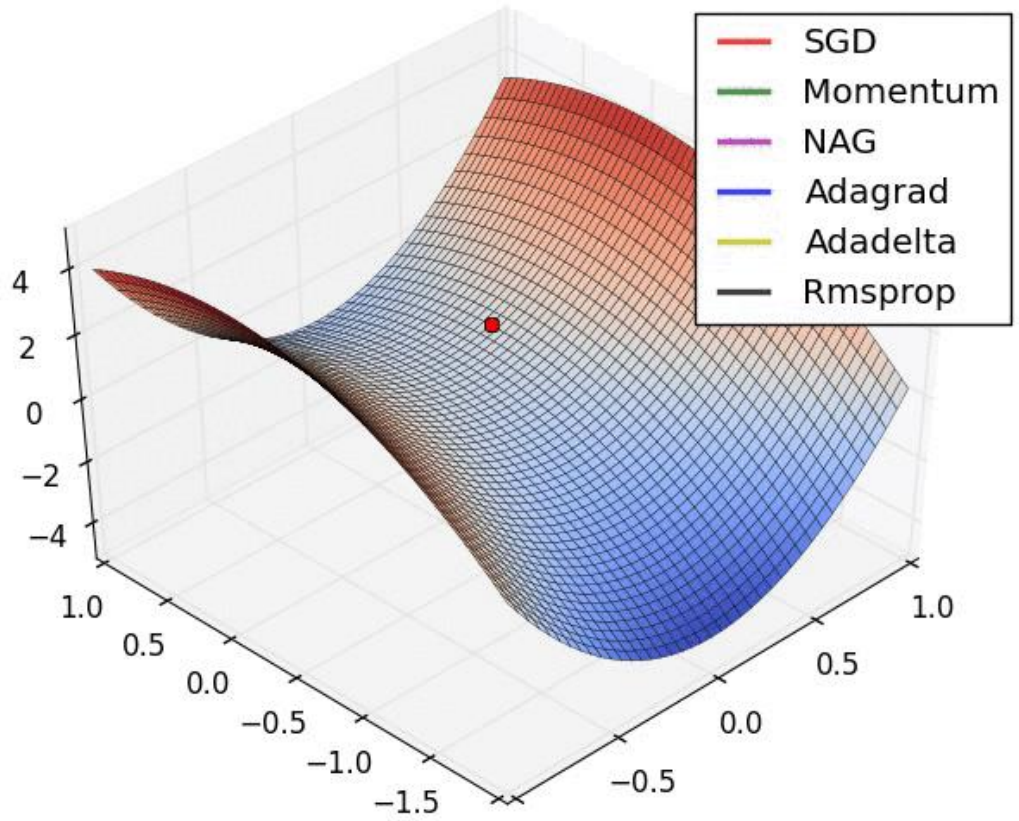
계산 그래프



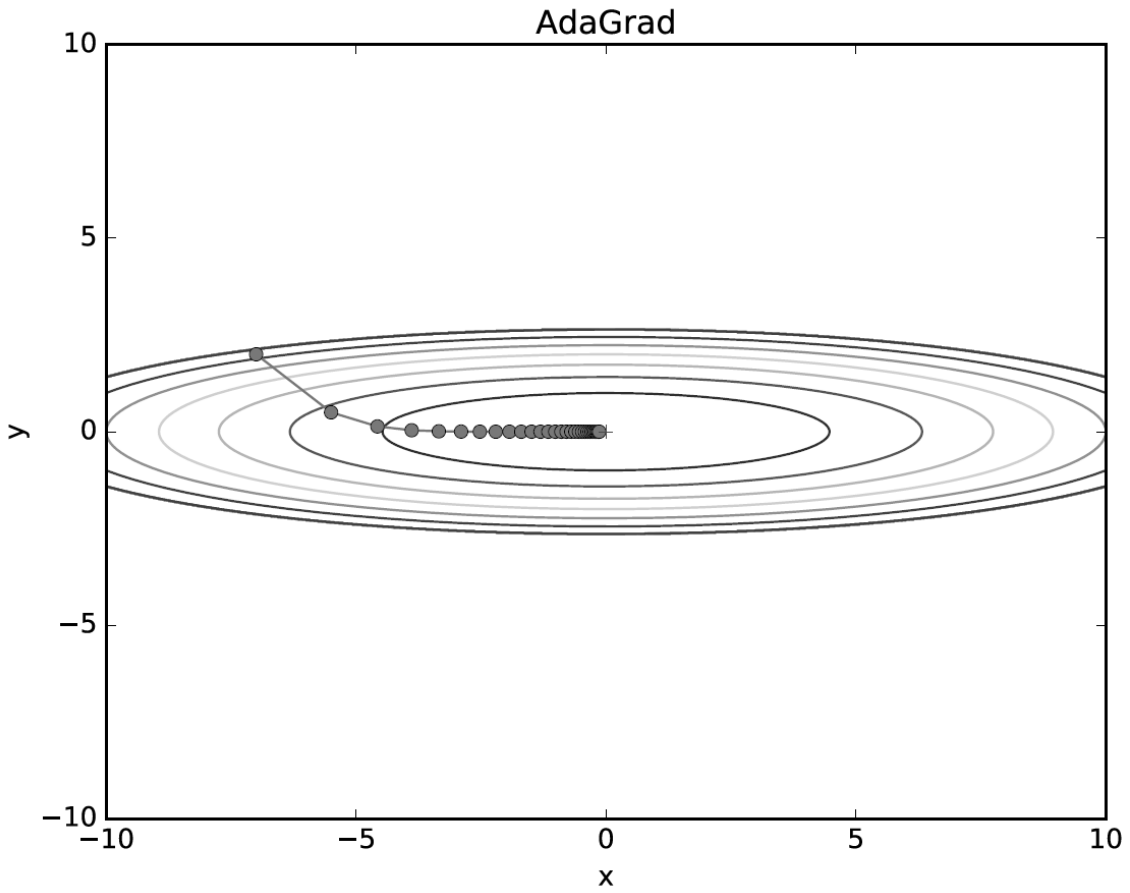
2. 학습 관련 기술들



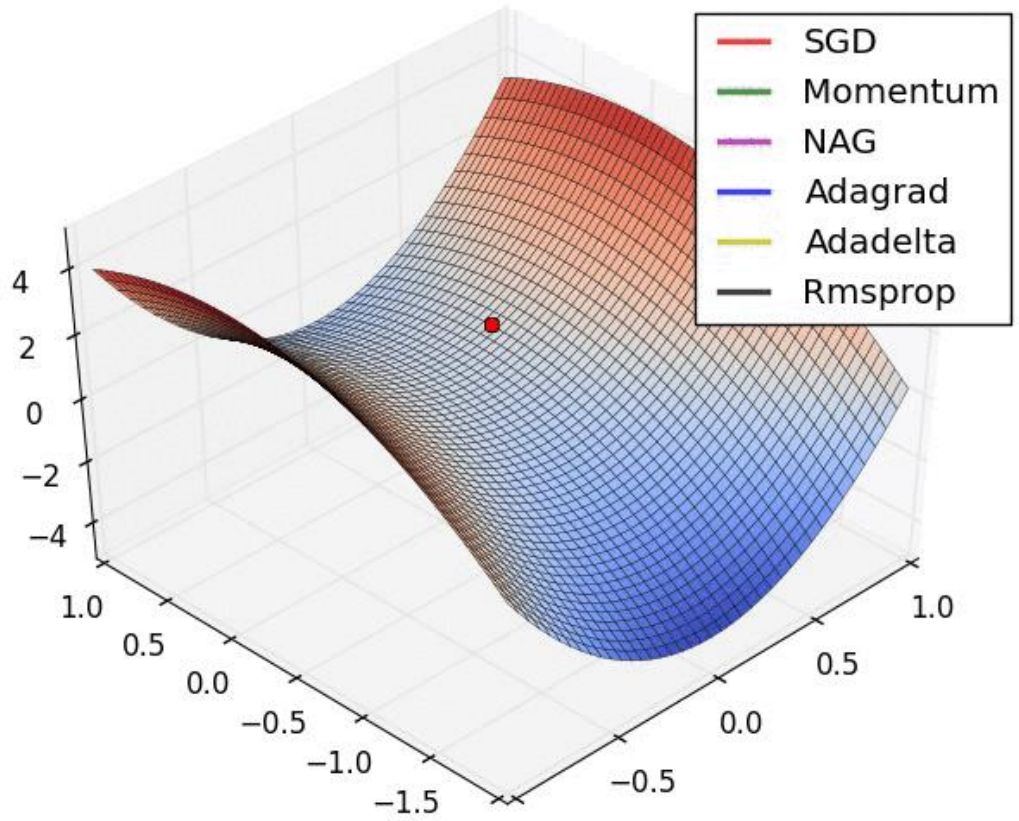
계산 그래프



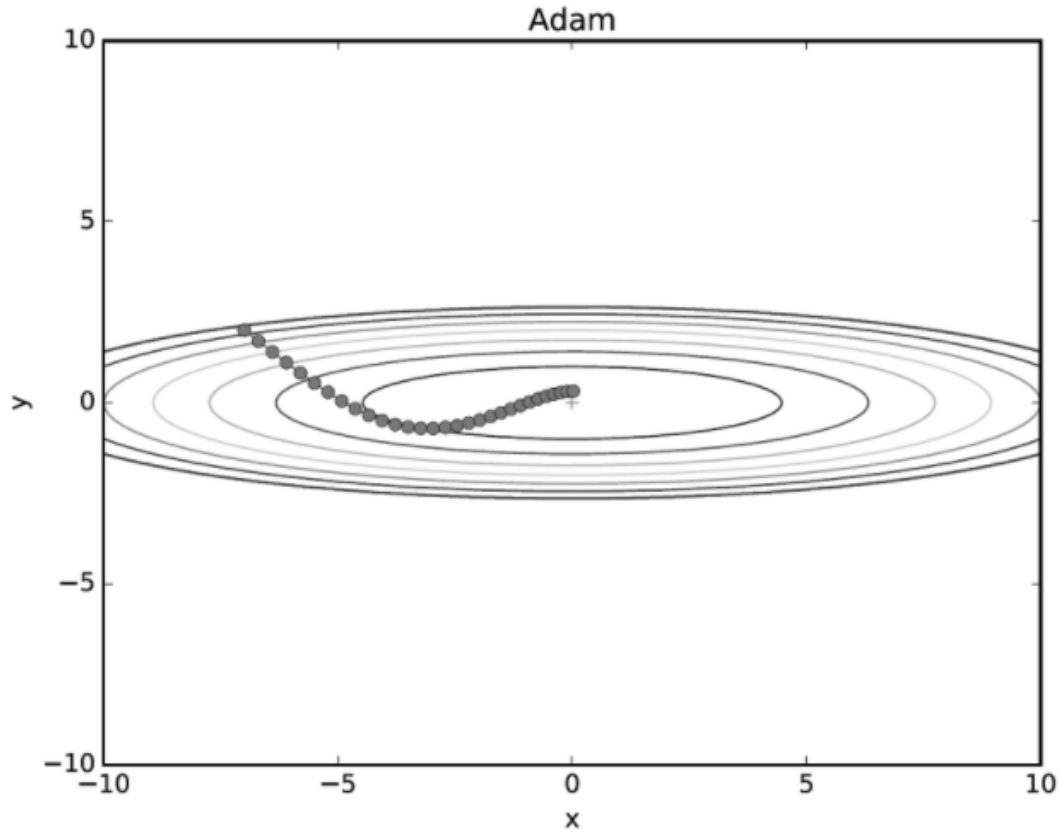
2. 학습 관련 기술들



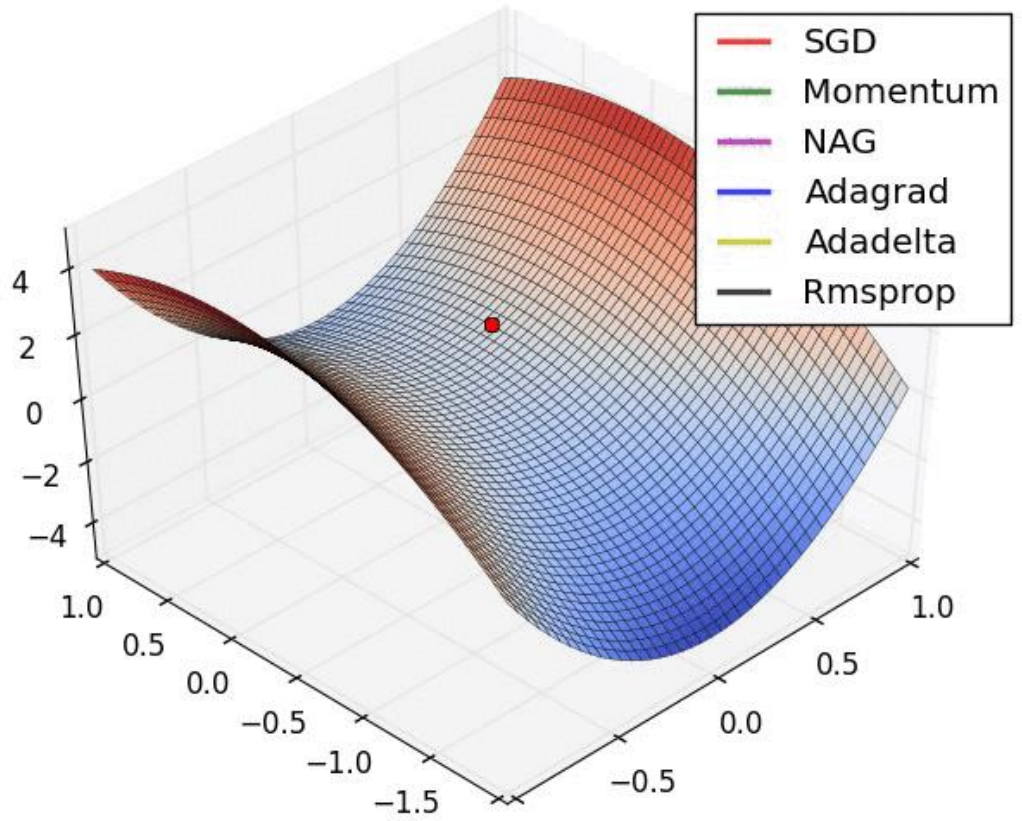
계산 그래프



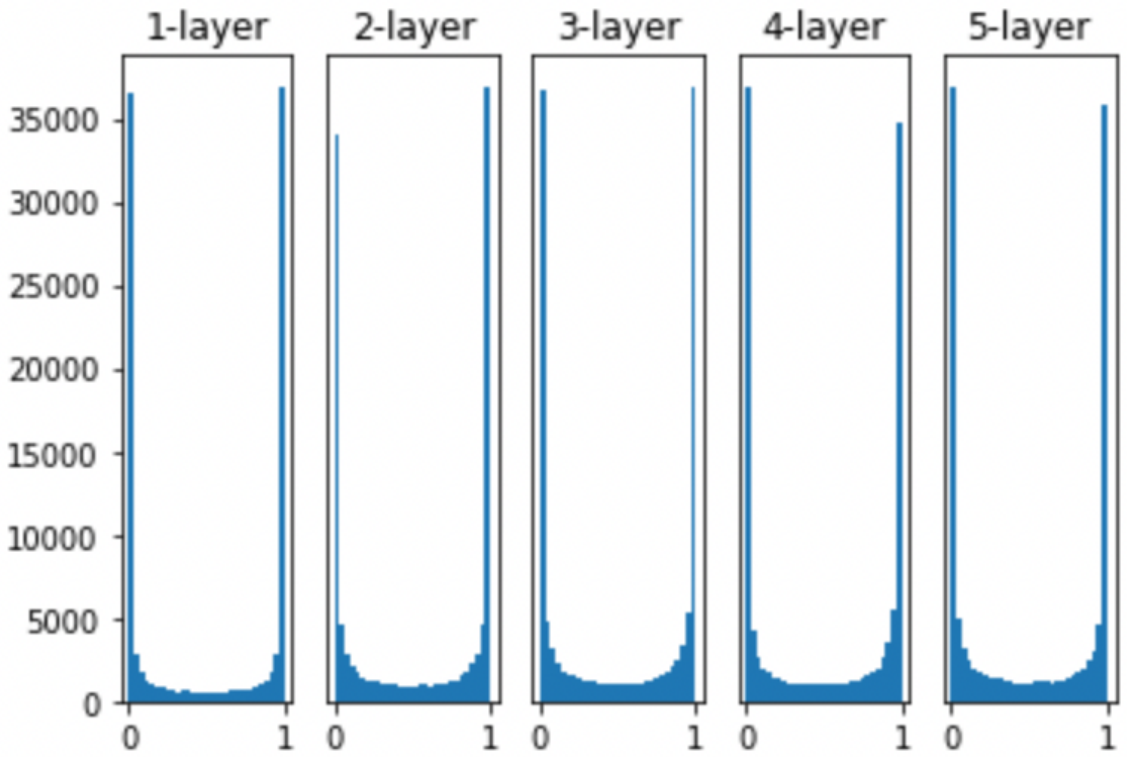
2. 학습 관련 기술들



계산 그래프



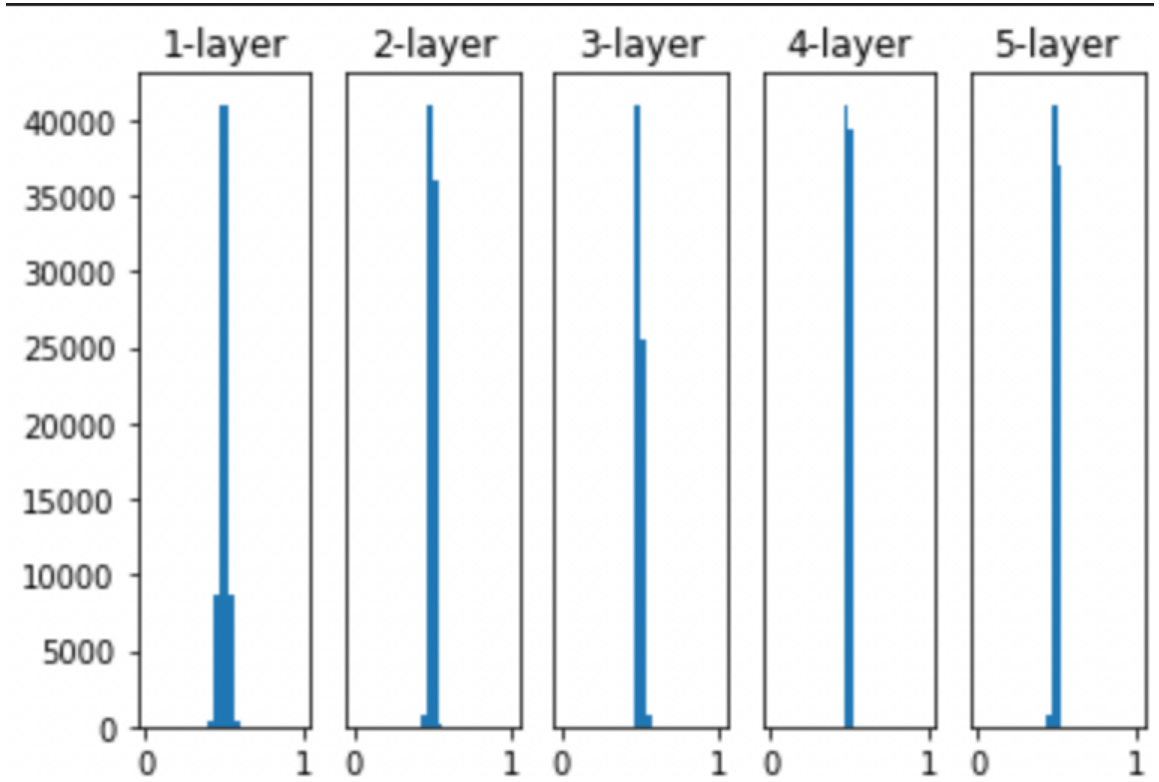
2. 학습 관련 기술들



초기값 그대로

```python
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    #w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    z = sigmoid(a)
    # z = relu(a)
    # z = tanh(a)

    activations[i] = z
```

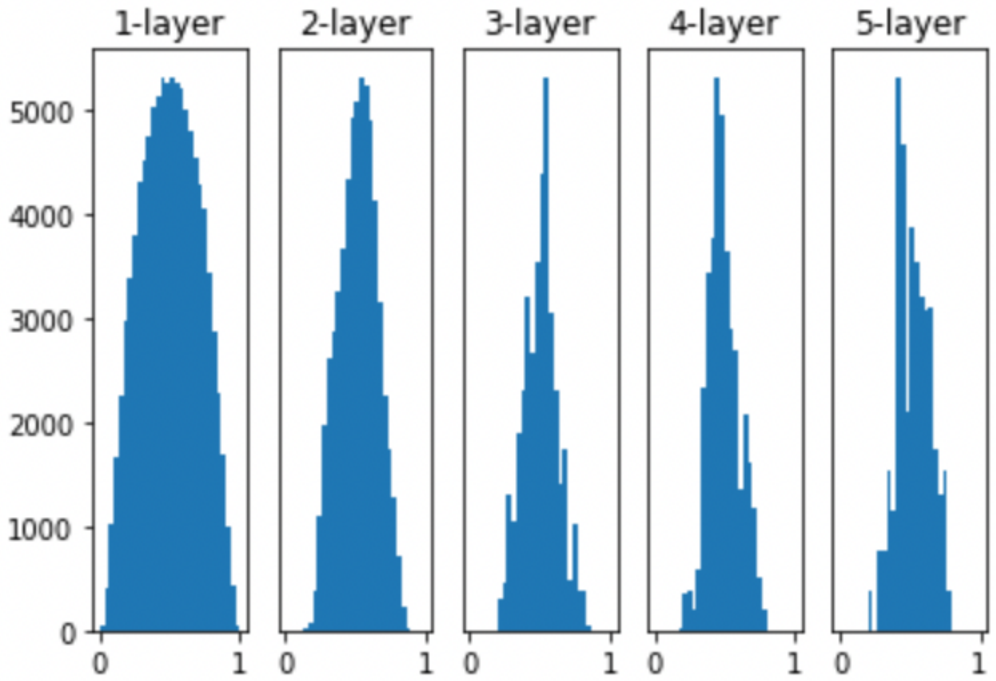
2. 학습 관련 기술들



초기값 * 0.01

```python
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    #w = np.random.randn(node_num, node_num) * 1
    w = np.random.randn(node_num, node_num) * 0.01

    #w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    z = sigmoid(a)
    # z = relu(a)
    # z = tanh(a)

    activations[i] = z
```

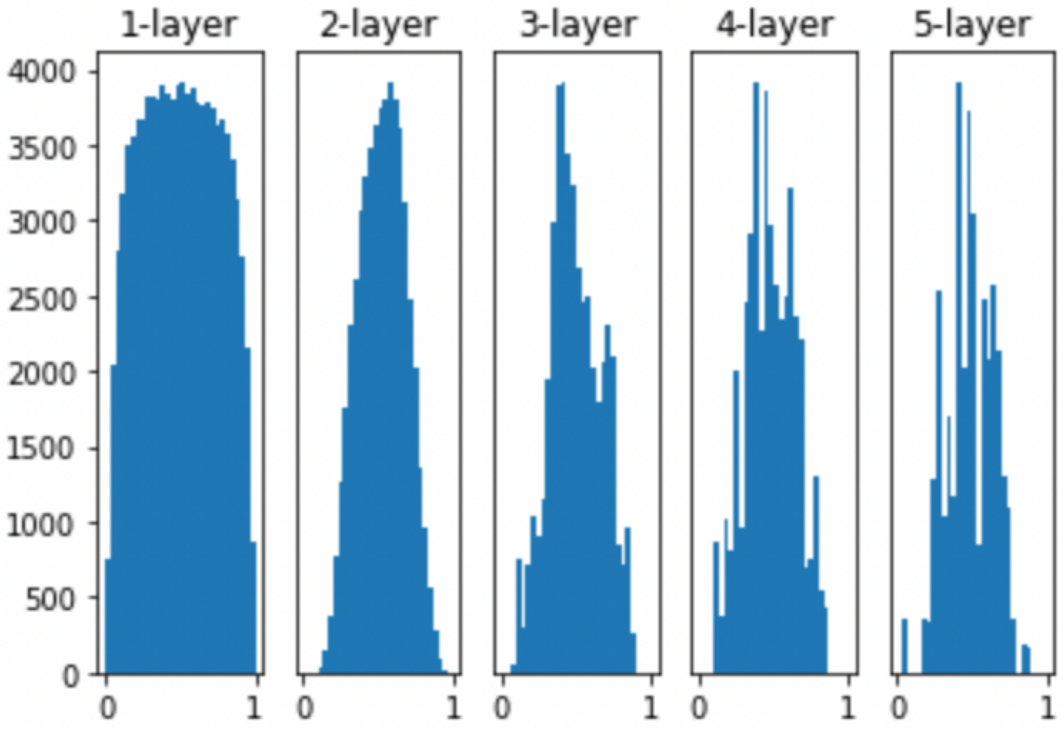
2. 학습 관련 기술들



Xavier 초기값

```python
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    # w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    z = sigmoid(a)
    # z = relu(a)
    # z = tanh(a)

    activations[i] = z
```

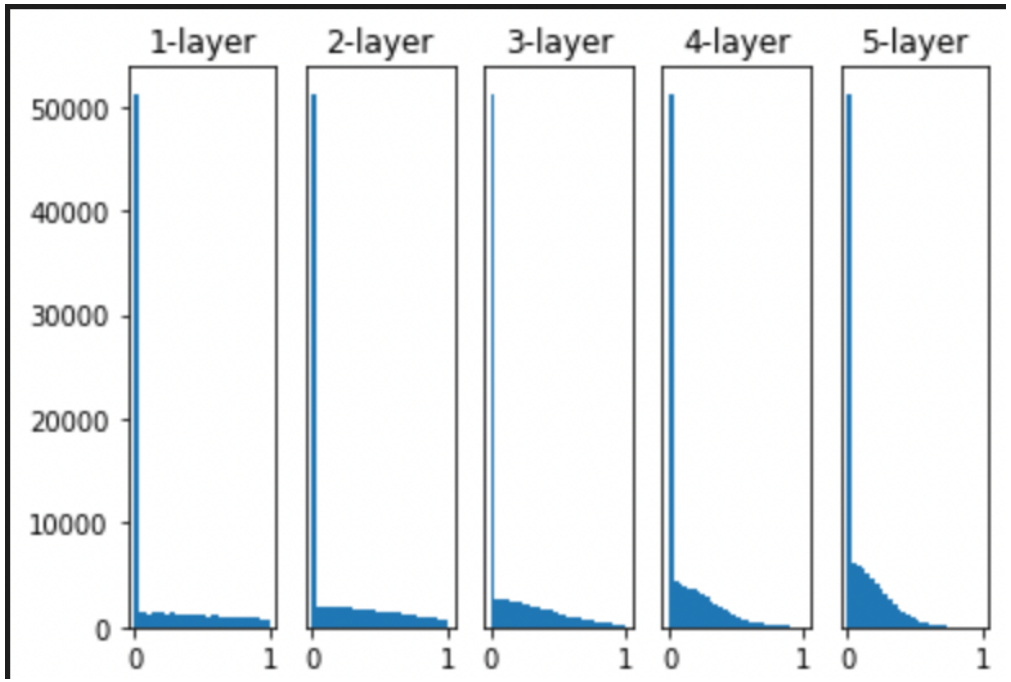
2. 학습 관련 기술들



He 초기값

```python
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    # w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    # w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    z = sigmoid(a)
    # z = relu(a)
    # z = tanh(a)

    activations[i] = z
```

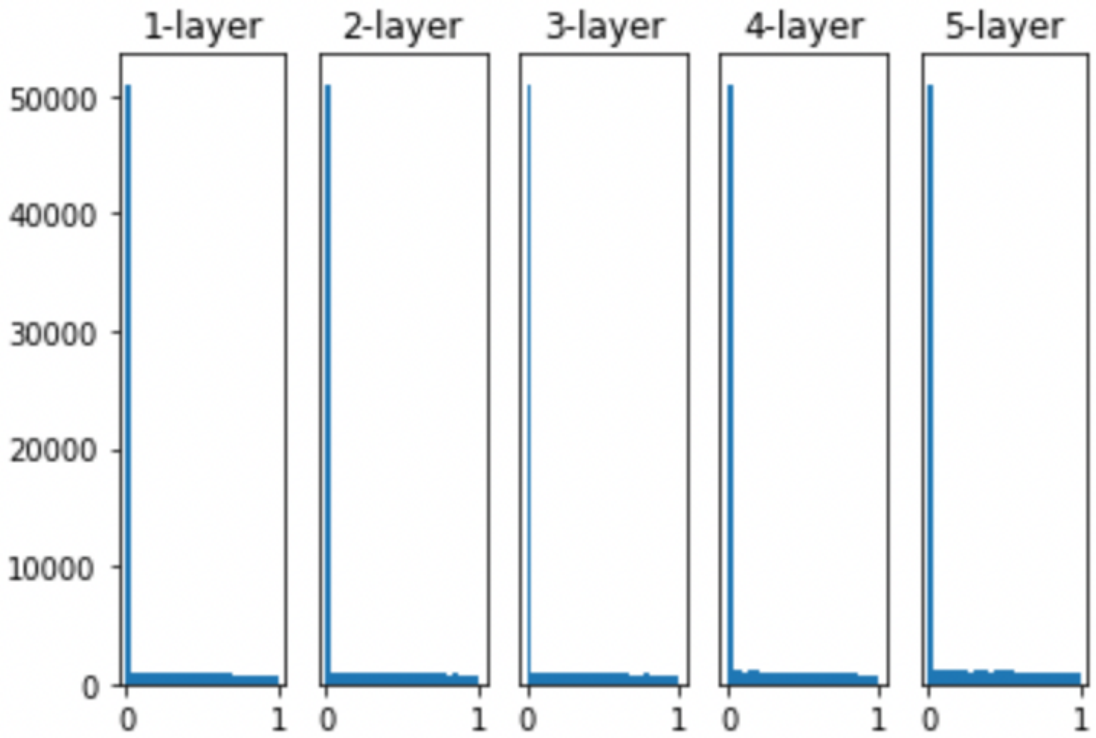
2. 학습 관련 기술들



Relu + Xavier 초기값

```python
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    # w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    # z = sigmoid(a)
    z = relu(a)
    # z = tanh(a)

    activations[i] = z
```

2. 학습 관련 기술들

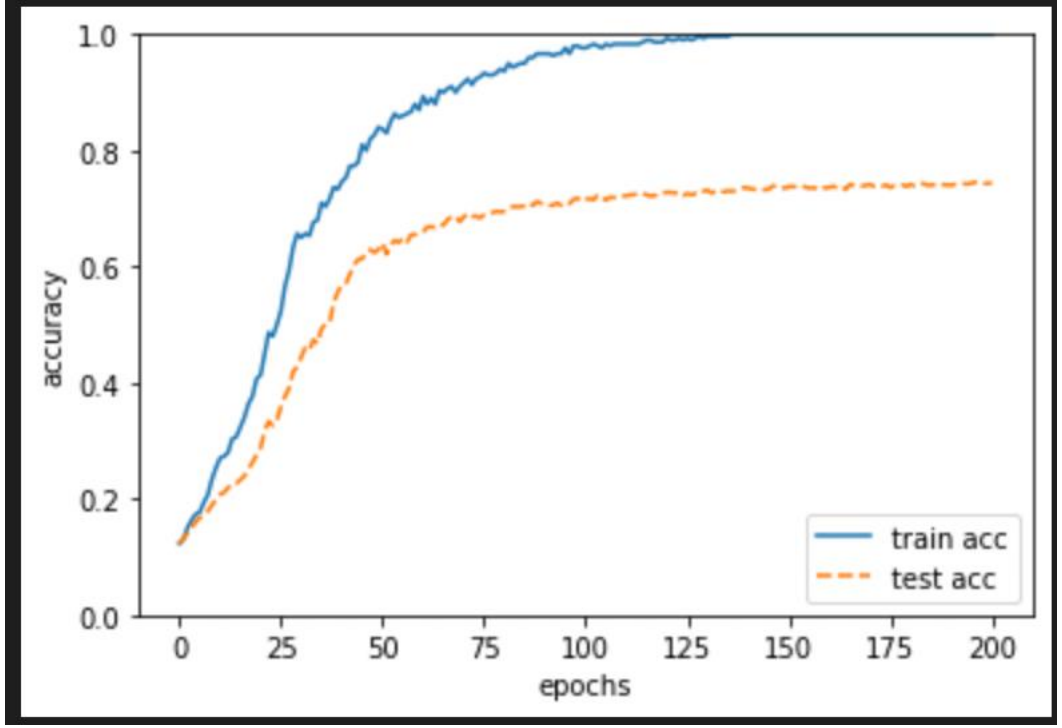


Relu + He 초기값

```python
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    # w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    # w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    # z = sigmoid(a)
    z = relu(a)
    # z = tanh(a)

    activations[i] = z
```

2. 학습 관련 기술들

```
train acc, test acc | 1.0, 0.7448
train acc, test acc | 1.0, 0.7431
train acc, test acc | 1.0, 0.7435
train acc, test acc | 1.0, 0.7441
```

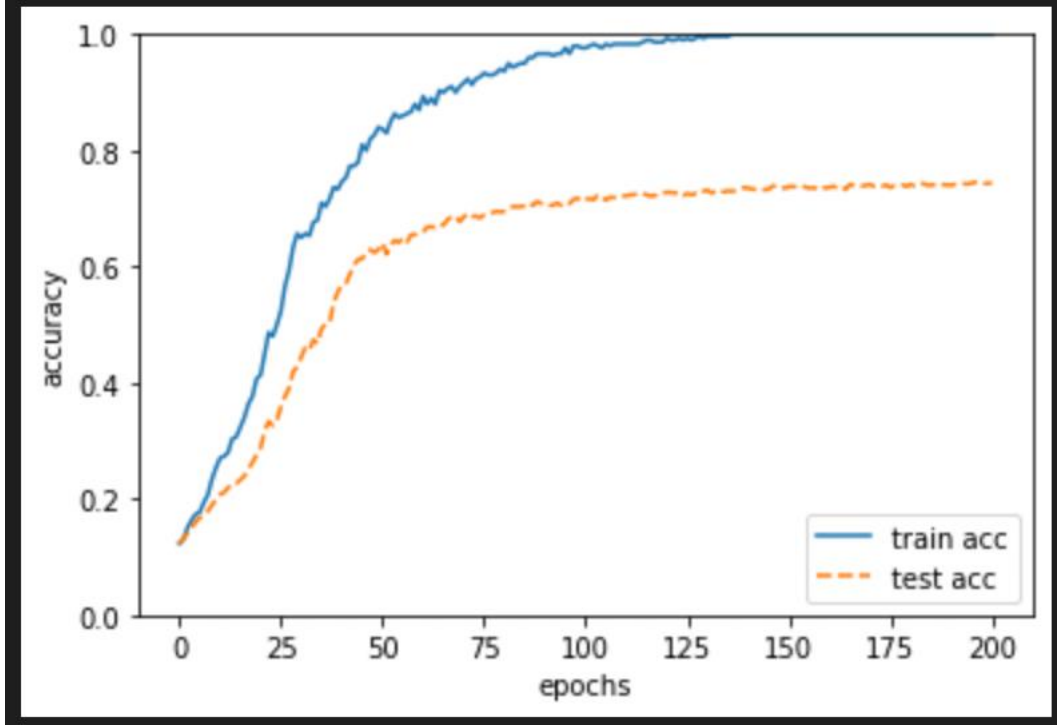


과적합

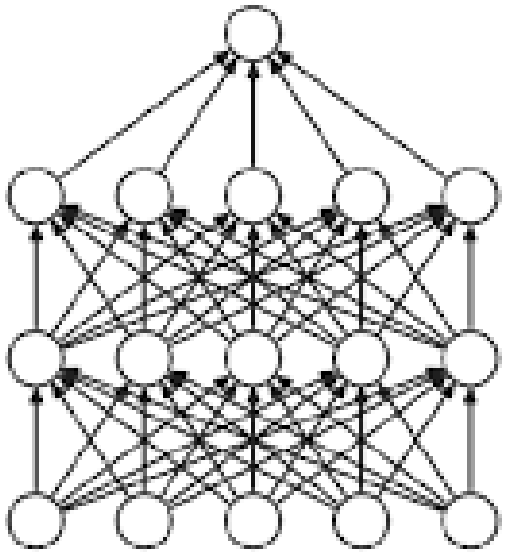
$$L(W) + \frac{1}{2}\lambda W^2$$

2. 학습 관련 기술들

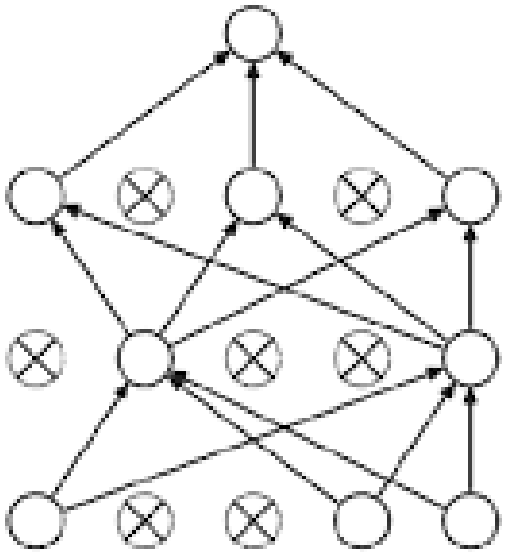
```
train acc, test acc | 1.0, 0.7448
train acc, test acc | 1.0, 0.7431
train acc, test acc | 1.0, 0.7435
train acc, test acc | 1.0, 0.7441
```



드롭아웃



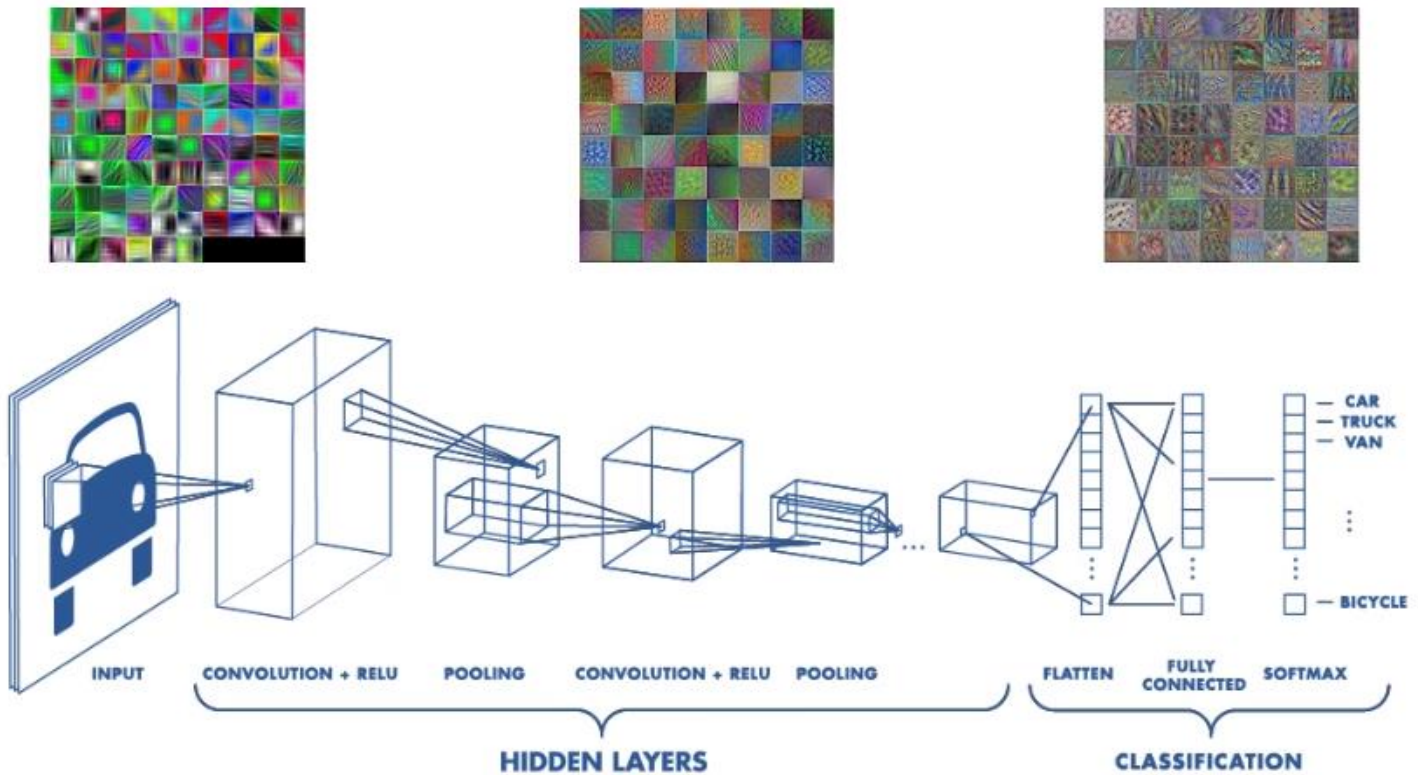
(a) 일반 신경망



(b) 드롭아웃을 적용한 신경망

3. CNN

Convolutional Neural Network



3. CNN

합성곱 계산

1*2+0*2+3*1+0*0+1*1+2*2+3*1+0*0+1*2 = 15

2*2+3*0+0*1+1*0+2*1+3*2+0*1+1*0+2*2 = 16

0*2+1*0+2*1+3*0+0*1+1*2+2*1+3*0+0*2 = 6

1*2+2*0+3*1+0*0+1*1+2*2+3*1+0*0+1*2 = 15

3. CNN

패딩

0	0	0	0	0	0
0	1	2	3	0	0
0	0	1	2	3	0
0	3	0	1	2	0
0	2	3	0	1	0
0	0	0	0	0	0

(4,4)
입력 데이터 패딩 : 1

*

2	0	1
0	1	2
1	0	2

(3, 3)
필터

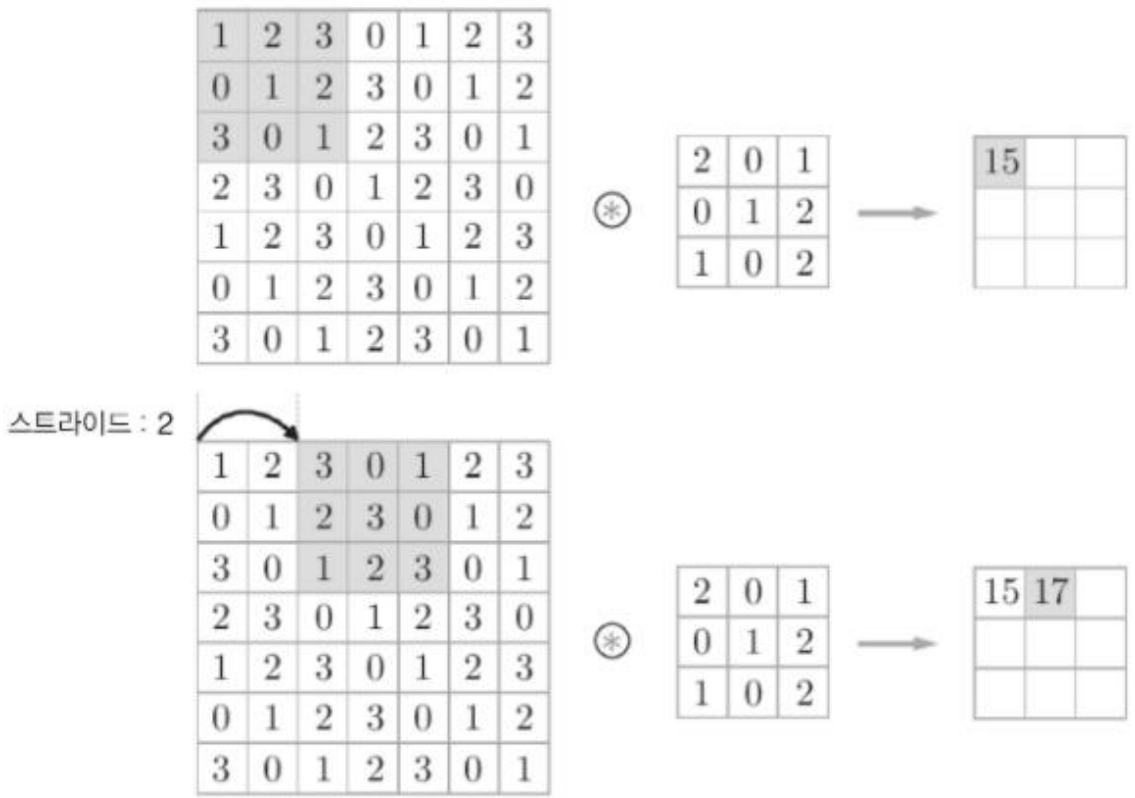
7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

(4, 4)
출력 데이터

출처 :
https://blog.naver.com/fbfbf1/222426802114

3. CNN

스트라이드



스트라이드, stride

명사 **체육**

육상 경기 등에서, 보폭(步幅)을 일컫는 말.

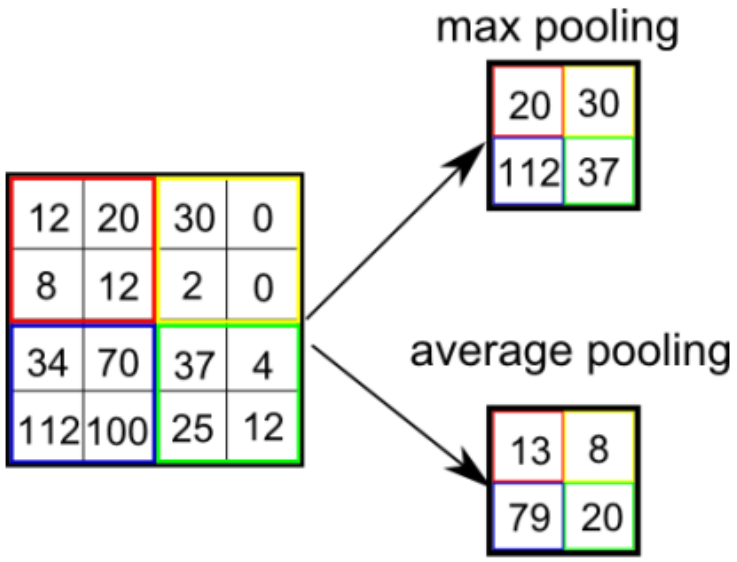
3. CNN

계산 그래프

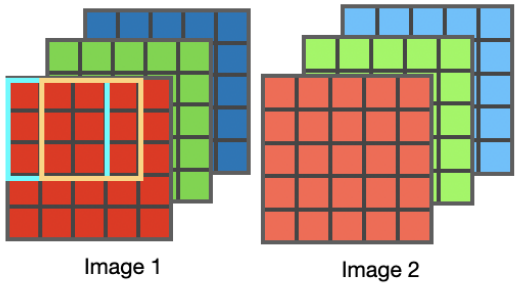
1. 특징 찾기
2. 데이터 크기 줄여주기
3. 오버피팅 억제

풀링 윈도우 – (2×2)

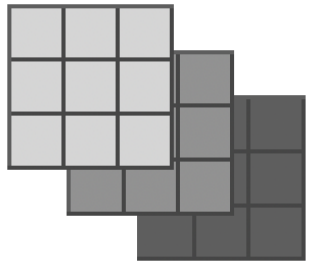
스트라이드 – 2



3. CNN

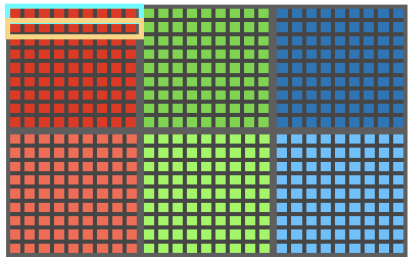


data: 5 x 5 x 3 x 2



filter: 3 x 3 x 3 x 1

1. im2col 연산



data: 18 x 27

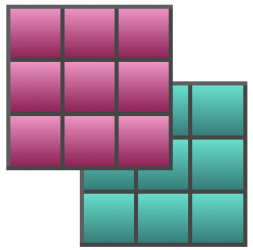
2. Reshape filter

filter: 27 x 1

3. 행렬곱

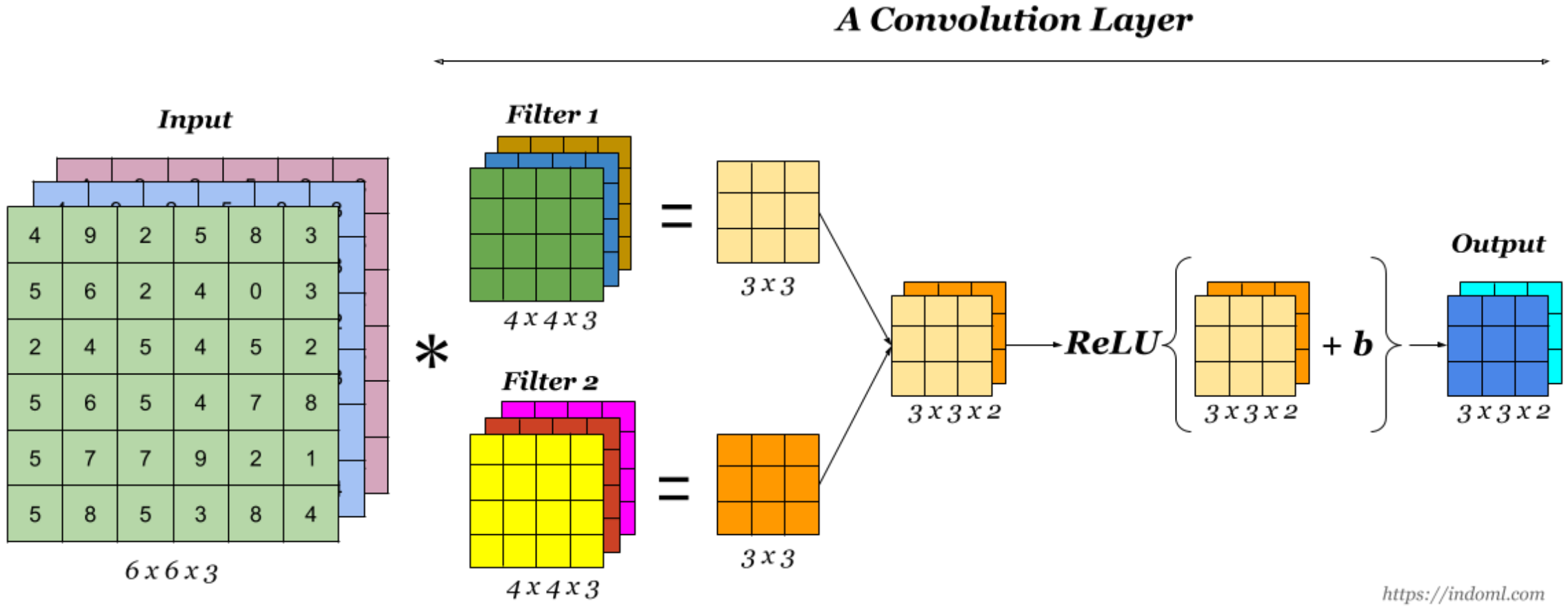
output: 18 x 1

4. Reshape output



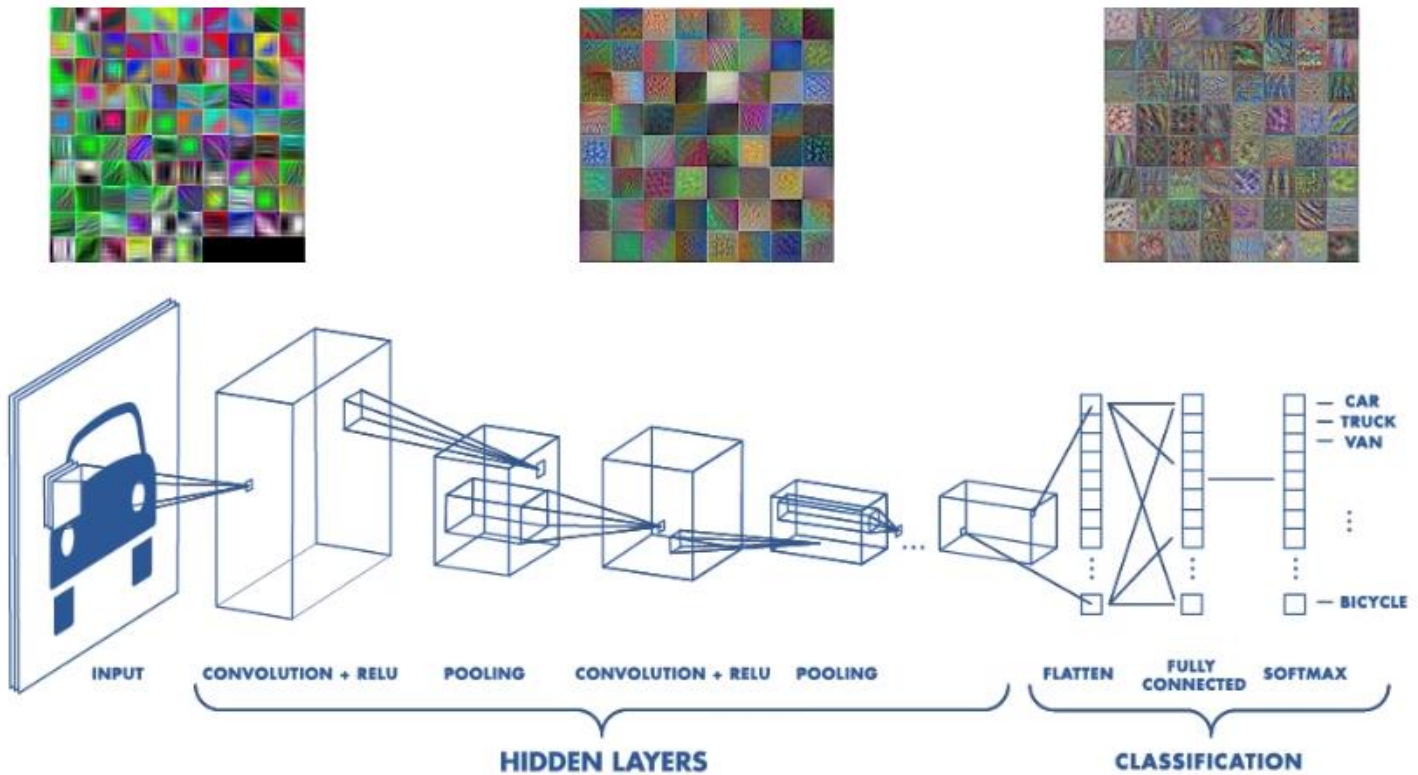
output: 3 x 3 x 1 x 2

3. CNN


A Convolution Layer
Input
4 9 2 5 8 3
5 6 2 4 0 3
2 4 5 4 5 2
5 6 5 4 7 8
5 7 7 9 2 1
5 8 5 3 8 4
6 x 6 x 3
*
Filter 1
4 x 4 x 3
=
3 x 3
Filter 2
4 x 4 x 3
=
3 x 3
3 x 3 x 2
ReLU
3 x 3 x 2
+ b
Output
3 x 3 x 2
https://indoml.com

3. CNN

Convolutional Neural Network



Thank you