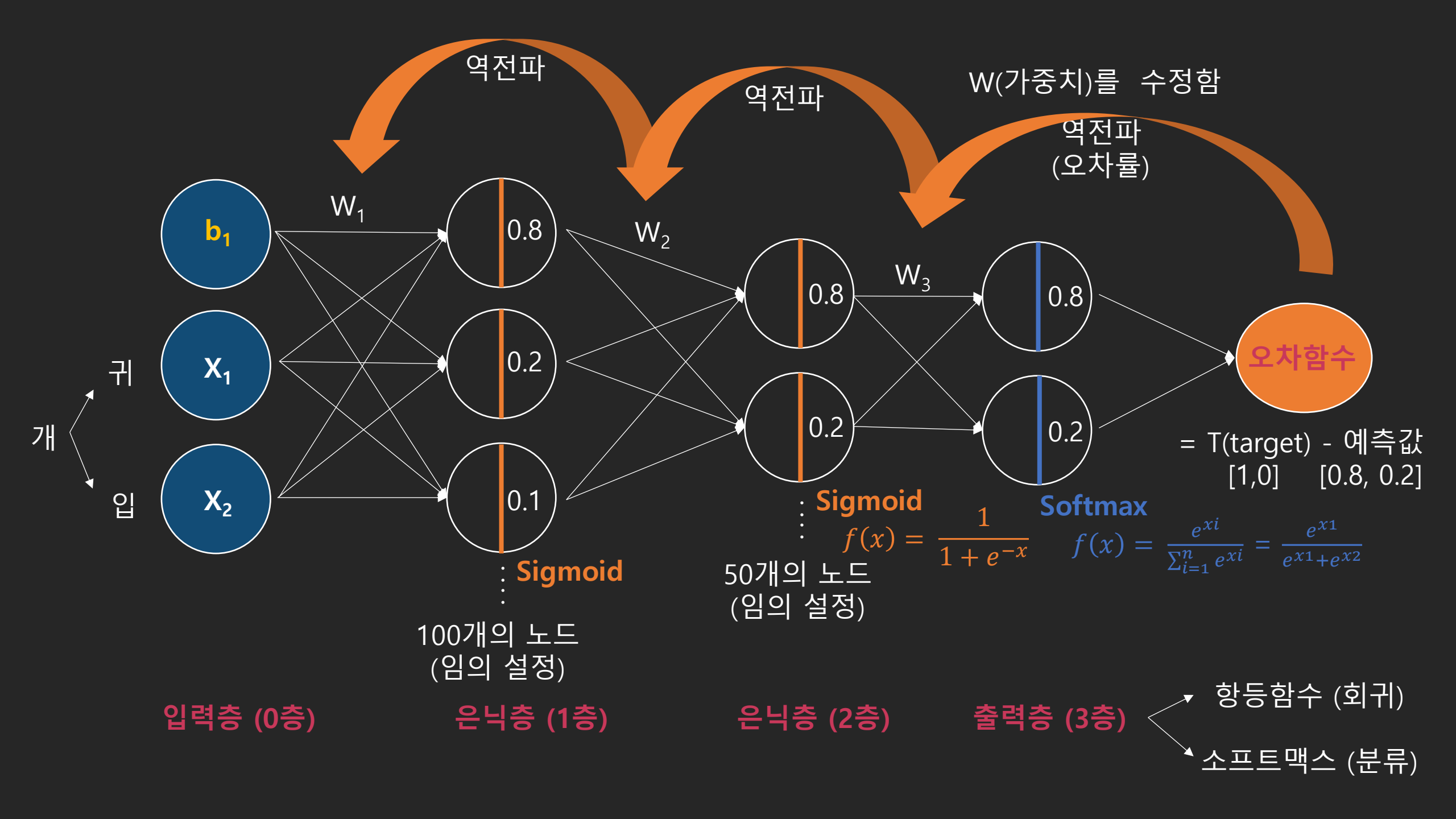
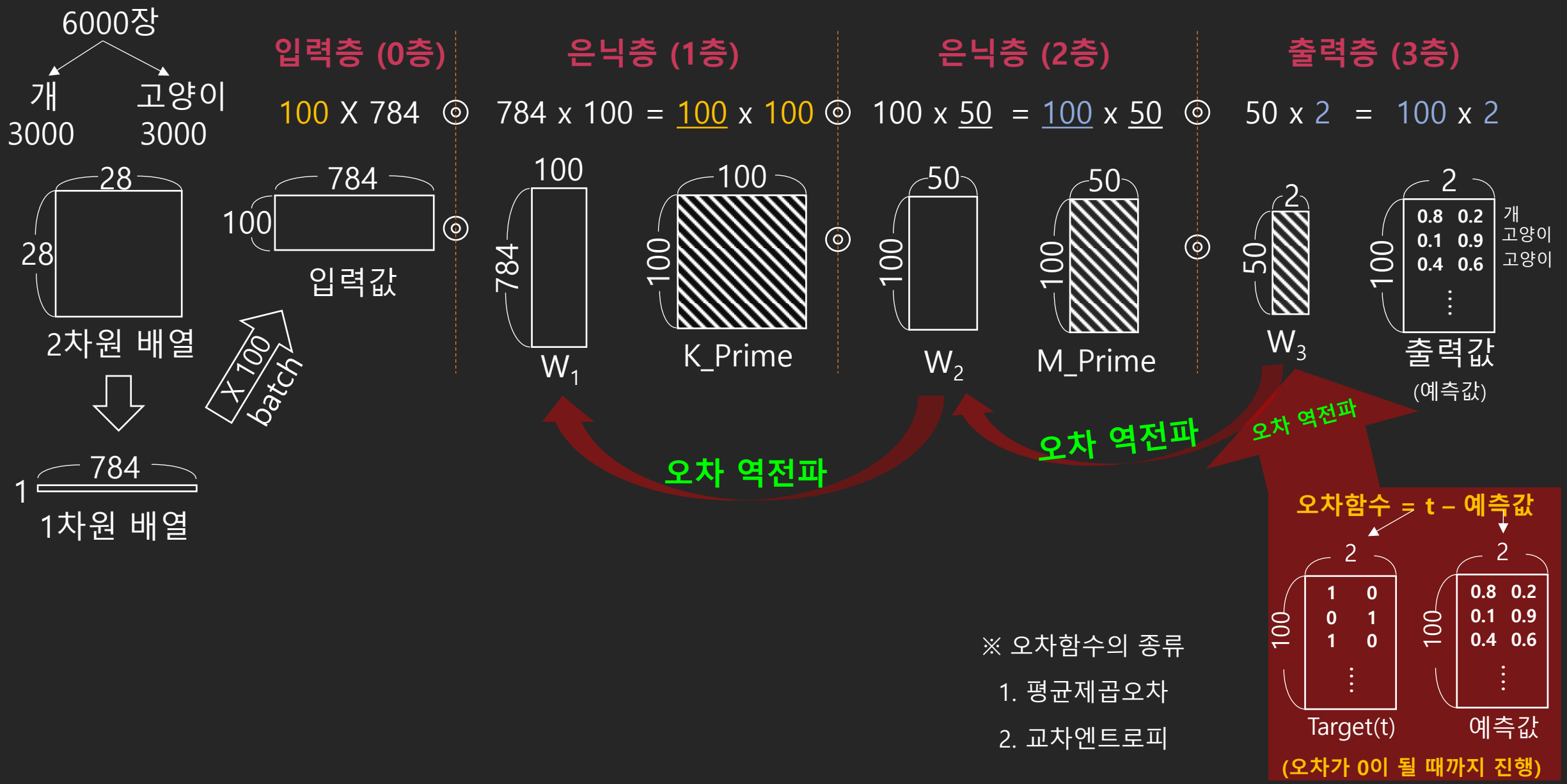

역전파
역전파
W(가중치)를 수정함
역전파
(오차률)
b_1
X_1
X_2
귀
개
입
W_1
W_2
W_3
0.8
0.2
0.1
0.8
0.2
0.8
0.2
오차함수
= T(target) - 예측값
[1,0] [0.8, 0.2]
Sigmoid
100개의 노드
(임의 설정)
Sigmoid
50개의 노드
(임의 설정)
$f(x) = \frac{1}{1+e^{-x}}$
Softmax
$f(x) = \frac{e^{xi}}{\sum_{i=1}^{n} e^{xi}} = \frac{e^{x1}}{e^{x1}+e^{x2}}$
입력층 (0층)
은닉층 (1층)
은닉층 (2층)
출력층 (3층)
항등함수 (회귀)
소프트맥스 (분류)

■ 배치(batch) 처리

- 큰 배열을 한꺼번에 계산하는 것이 분할된 작은 배열을 여러 번 계산하는 것보다 빠르기 때문입니다.



※ 오차함수의 종류

1. 평균제곱오차
2. 교차엔트로피

오즈함수 → 로짓함수 → 시그모이드 함수

$\frac{\text{성공}}{\text{실패}} = \frac{P}{1-p}$

$\log(\text{오즈함수}) = \log(\frac{P}{1-p})$

$\text{Ln}(\text{오즈함수}) = ln(\frac{P}{1-p})$

$ln(\frac{P}{1-p}) = w*\text{x} + \text{b}$

$ln(\frac{P}{1-p}) = w*\text{x} + \text{b}$

$e^{ln(\frac{P}{1-p})} = e^{w*\text{x} + \text{b}}$

$(\frac{P}{1-P})^{loge^e} = e^{w*\text{x} + \text{b}}$

$\frac{P}{1-P} = e^{w*\text{x} + \text{b}}$

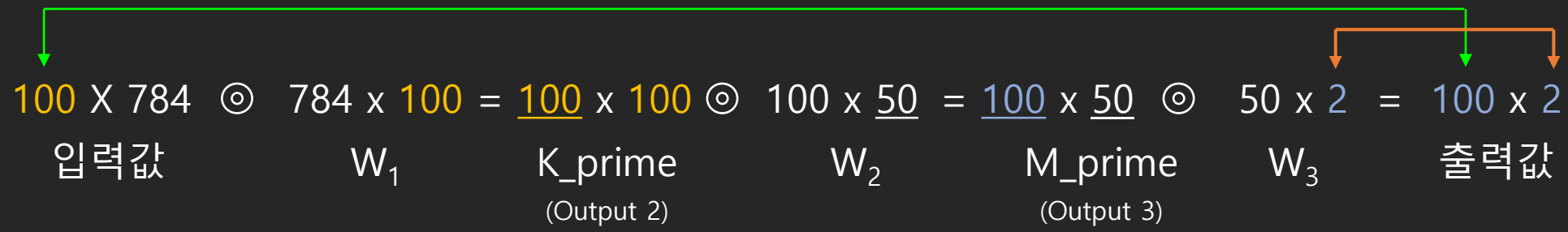
$\frac{1-P}{P} = e^{-(w*\text{x} + \text{b})}$

$\frac{1}{P} - 1 = e^{-(w*\text{x} + \text{b})}$

$\frac{1}{P} = 1 + e^{-(w*\text{x} + \text{b})}$

$\text{P} = \frac{1}{1+e^{-(w*\text{x} + \text{b})}}$ # $w*\text{x} + \text{b} = k$

$f(k) = \frac{1}{1+e^{-k}}$


100 X 784 ◎ 784 x 100 = 100 x 100 ◎ 100 x 50 = 100 x 50 ◎ 50 x 2 = 100 x 2
입력값
W1
K_prime
(Output 2)
W2
M_prime
(Output 3)
W3
출력값

경사하강법

1. GD는 정확하게 직선으로 내려가지만 굉장히 느림

2. SGD 지그재그로 불안하게 가지만 결국 0에 도달. 속도가 GD보다 빠름.
   안장점과 local minimum을 구분 x
   둘다 기울기가 0인 상황에서 구분이 안 되서 오류 가능성

3. Momentum → 가속도를 이용해서 Local minimum을 빠져나간다

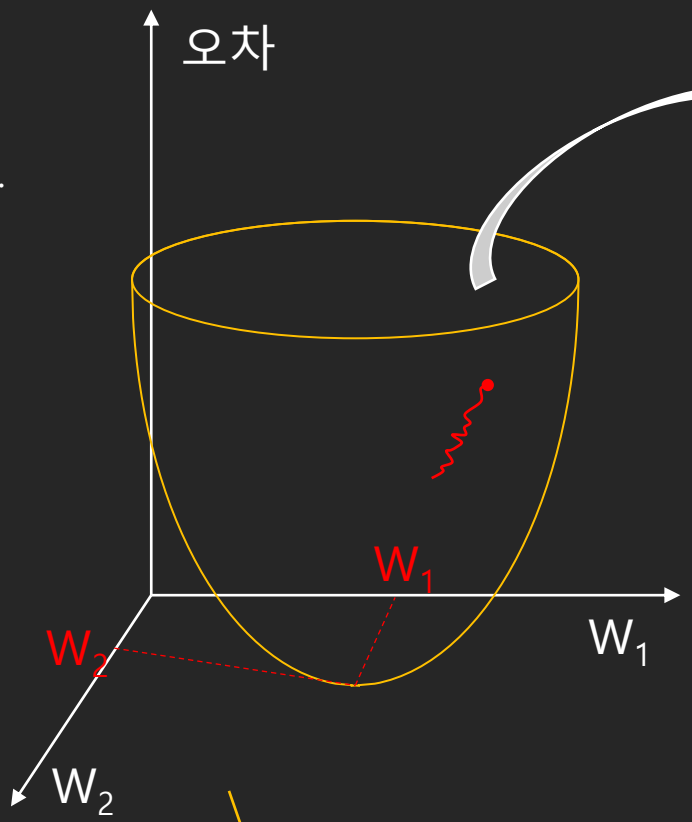
4. Adagrade → 러닝메이트가 자동조절

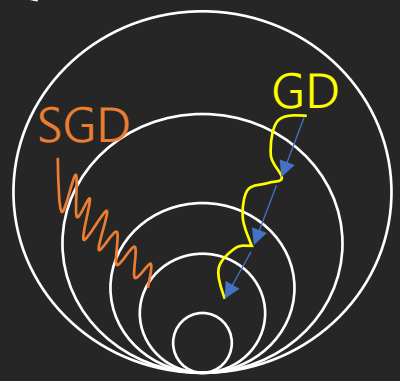
5. Adam → Momentum + Adagrade
   (장점) (장점)

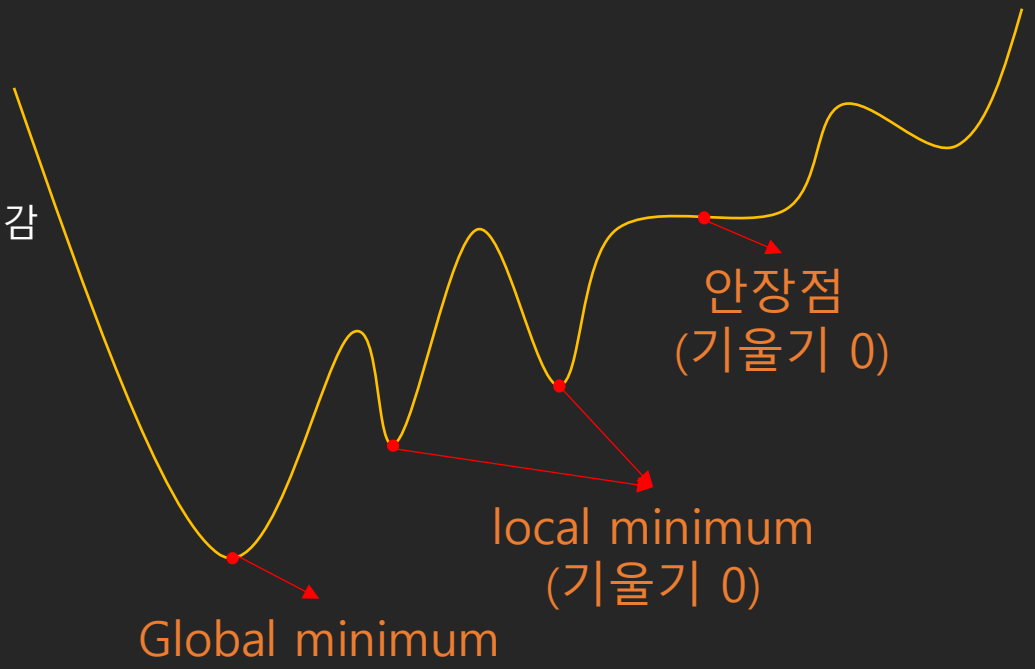
※ SVM(support vector machine)
→ Local minimum에 안 빠진다.

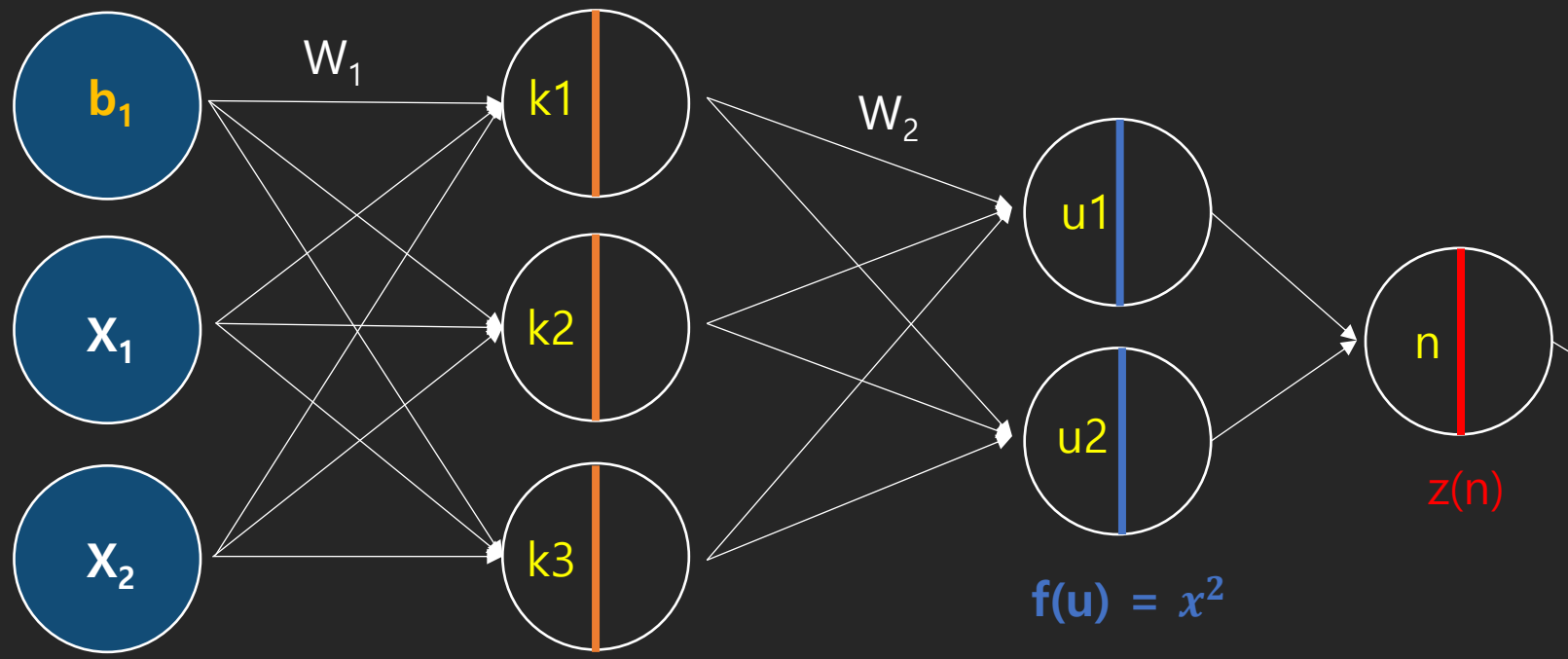
하이퍼 파라미터
→ 가장 최적화 러닝메이트

- 러닝메이트가 너무 크면 Global minimum을 넘어감
- 러닝메이트가 너무 작으면
  Global minimum까지 도달하지 못하고 소멸한다.









ex

softmax

$y = z(f(g(x)))$

오차함수 sigmoid

$y = z(f(g(x)))$
(오차)

$\frac{\partial z}{\partial x} = z(f(g(x)))' \times f(g(x))' \times g(x)'$
z(n) n=f(k) k=g(x)

$= z(n)' \times f(k)' \times g(x)'$

$= \frac{\partial z}{\partial n} \times \frac{\partial n}{\partial k} \times \frac{\partial k}{\partial x}$ (연쇄법칙)

$\frac{\partial z}{\partial x} = \frac{d\text{오차함수}}{\partial w1}$

$f(g(k)) = (3x+1)^2$
도함수 방법?

$f(g(k)) = (3x+1)^2$
$= 9x^2 + 6x + 1$
$f(g(k))' = 18x + 6$

$f(g(k))' = f(g(k))'g(k)'$
$= 2*(3x+1)*3$
$= 6(3x+1)$

$y = u^2$
y를 u에 대해서 미분하고
(겉미분)

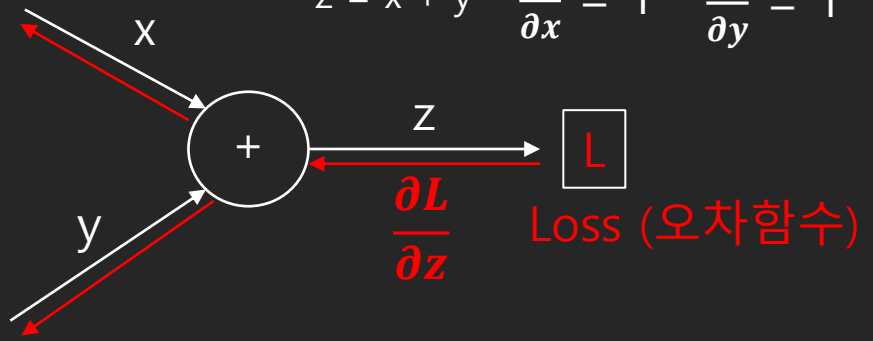
$u = 3x + 1$
u 를 x에 대해서 미분하고
(속미분)

$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} * \frac{\partial u}{\partial x}$ (연쇄법칙)

1. 덧셈 계산 그래프

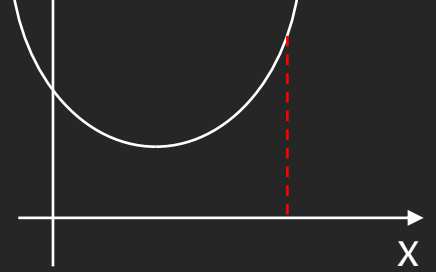
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} \times 1$$

$z = x + y \quad \frac{\partial z}{\partial x} = 1 \quad \frac{\partial z}{\partial y} = 1$



$$\frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial y} = \frac{\partial L}{\partial z} \times 1$$

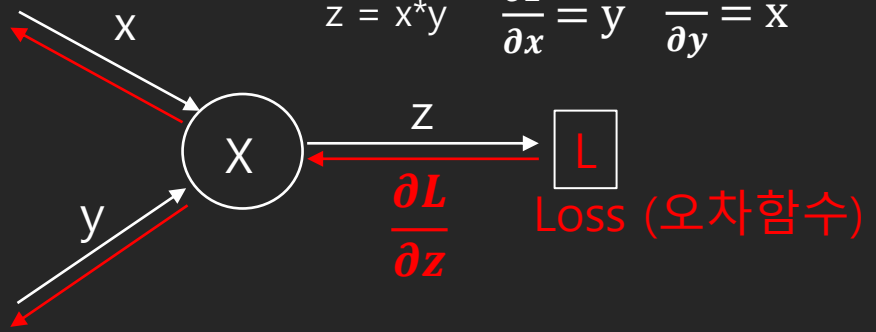


* 덧셈은 그대로 흘러간다. 덧셈의 편미분은 1이므로

2. 곱셈 계산 그래프 (p158)

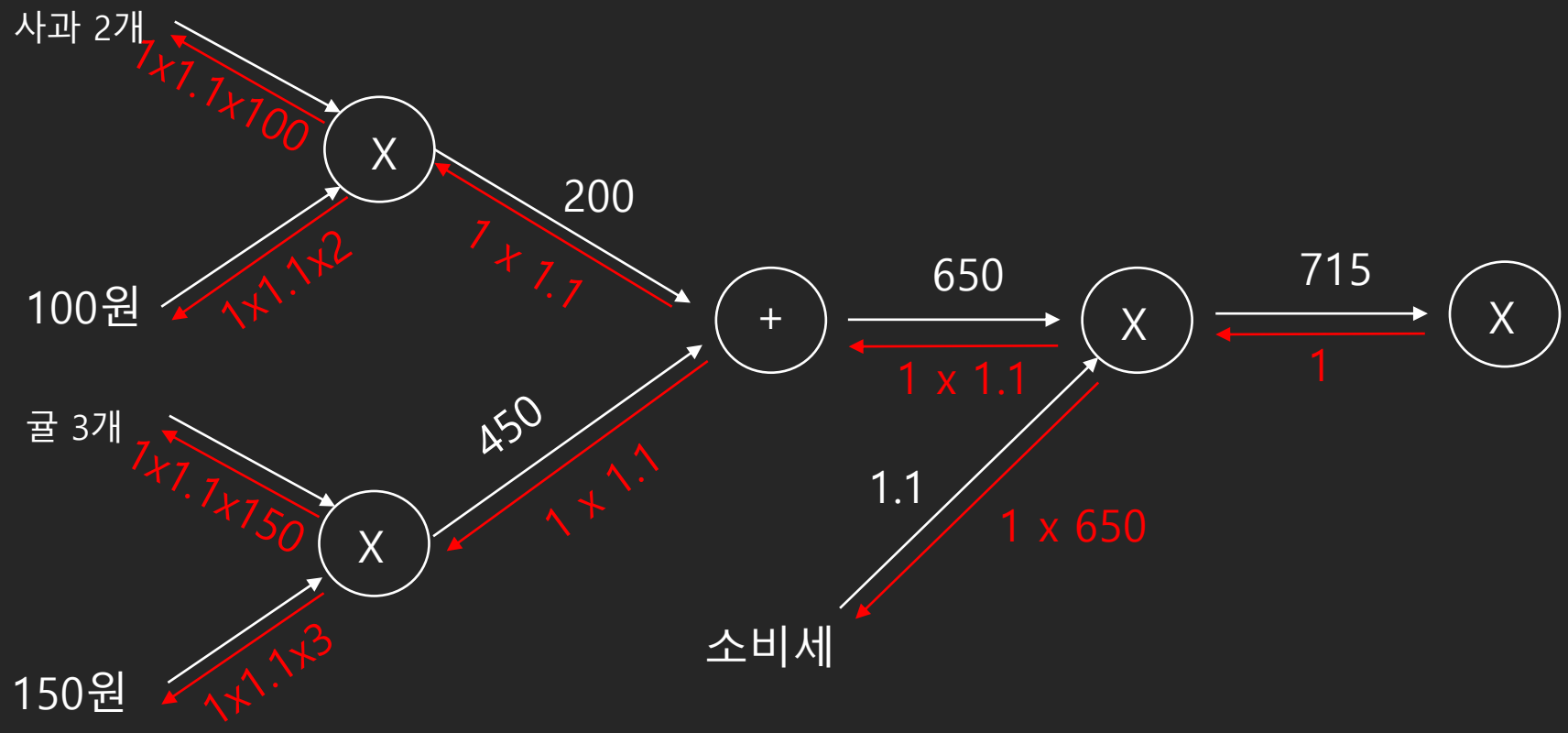
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} \times y$$

$z = x*y \quad \frac{\partial z}{\partial x} = y \quad \frac{\partial z}{\partial y} = x$

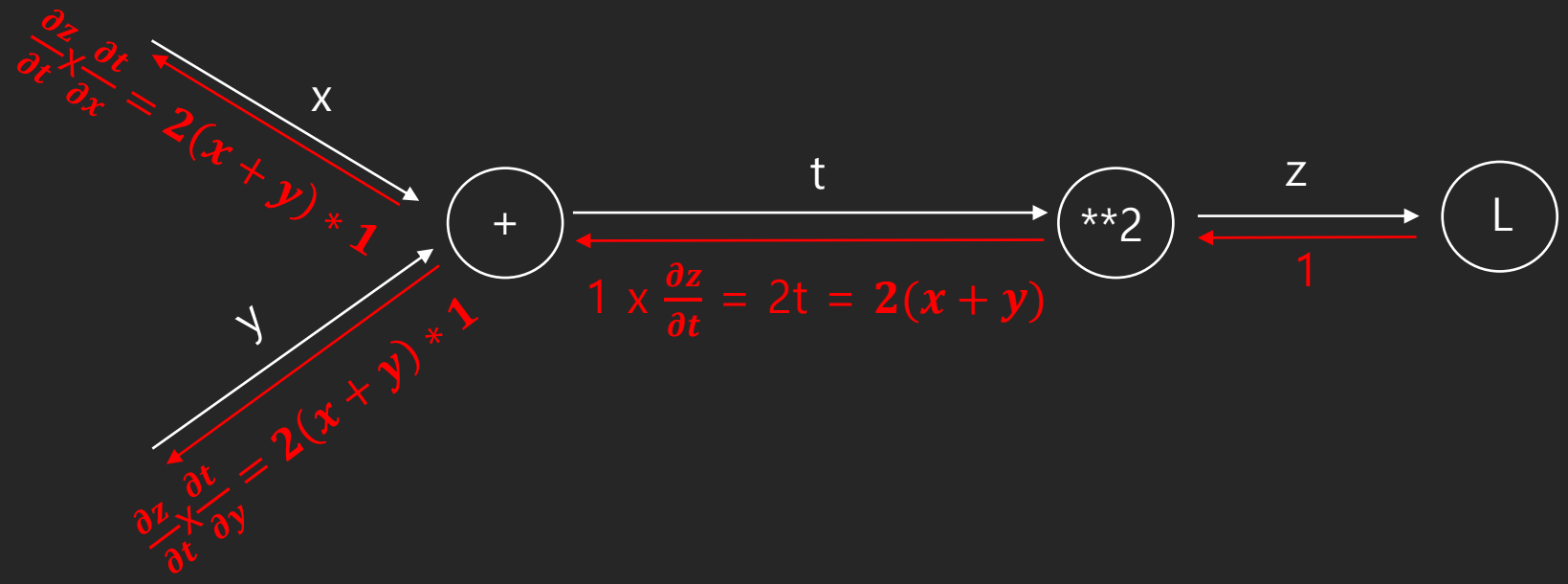


$$\frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial y} = \frac{\partial L}{\partial z} \times x$$


사과 2개
1x1.1x100
X
200
1 x 1.1
100원
1x1.1x2
+
650
X
715
X
1 x 1.1
1
귤 3개
1x1.1x150
450
1 x 1.1
X
1.1
1 x 650
150원
1x1.1x3
소비세

$z = (x+y)^2$, t = x + y, $z = t^2$

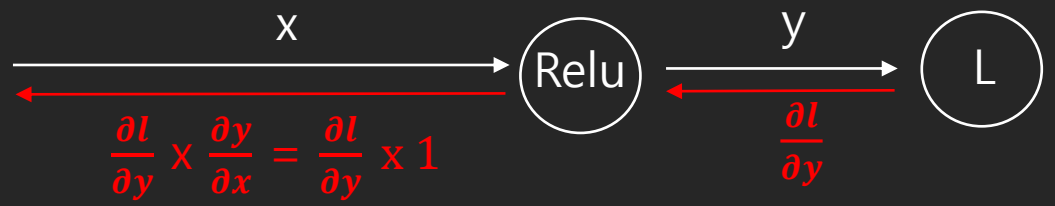


1. 렐루(Relu) 함수 계산그래프

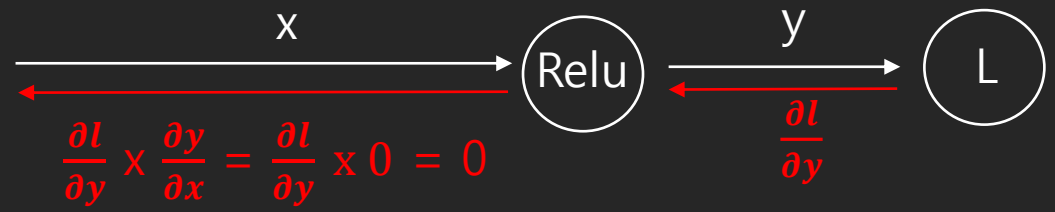
순전파: y = { x (x>0) / 0 (x<=0)

역전파: $\frac{\partial y}{\partial x}$ (기울기) = { 1 (x>0) / 0 (x<=0)

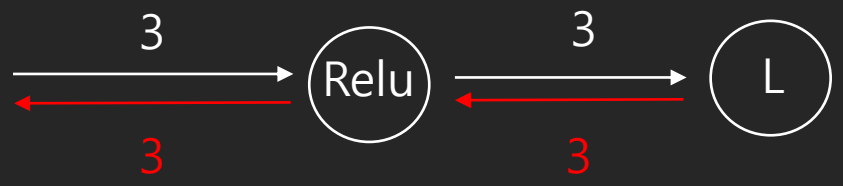
x > 0, y = x :



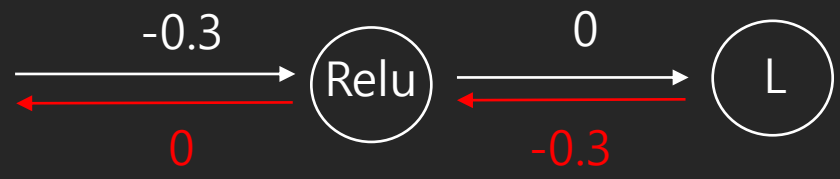
x <= 0, y = 0 :



x > 0, y = x :

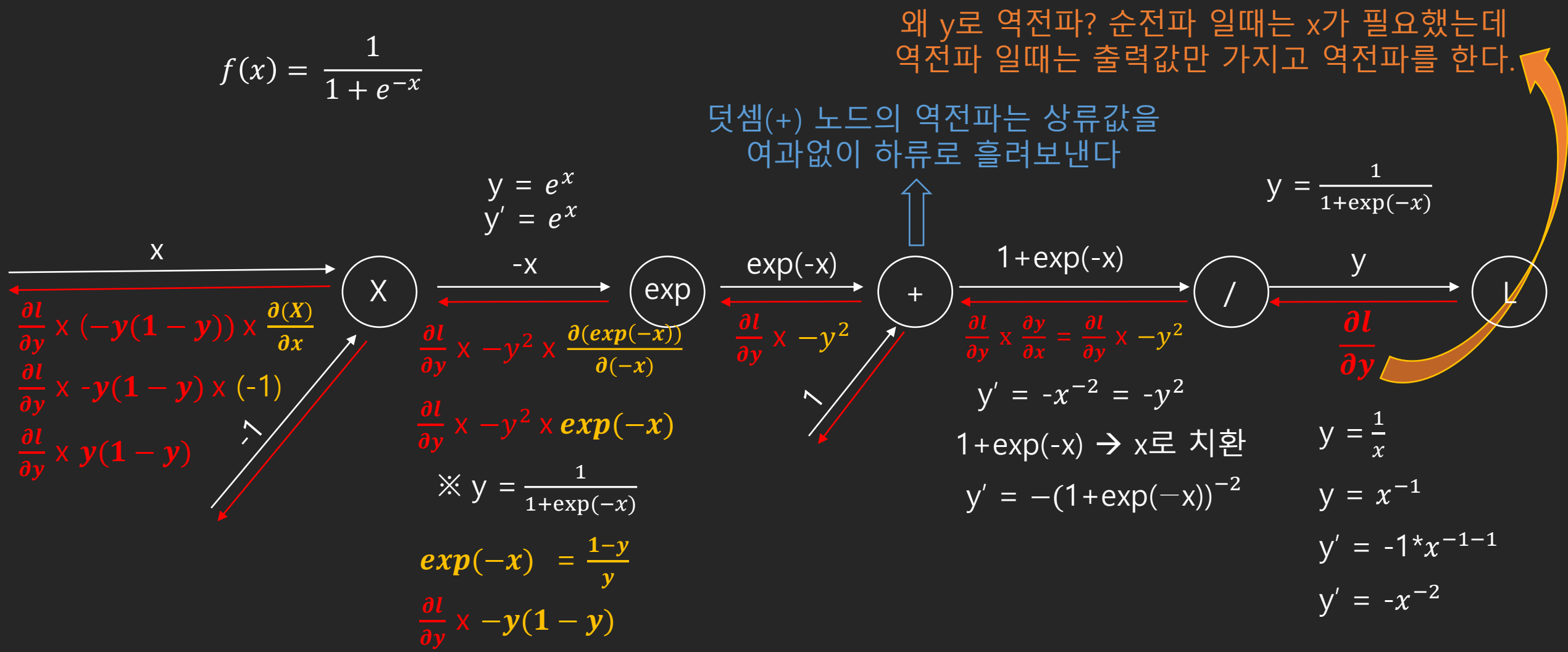


x <= 0, y = 0 :



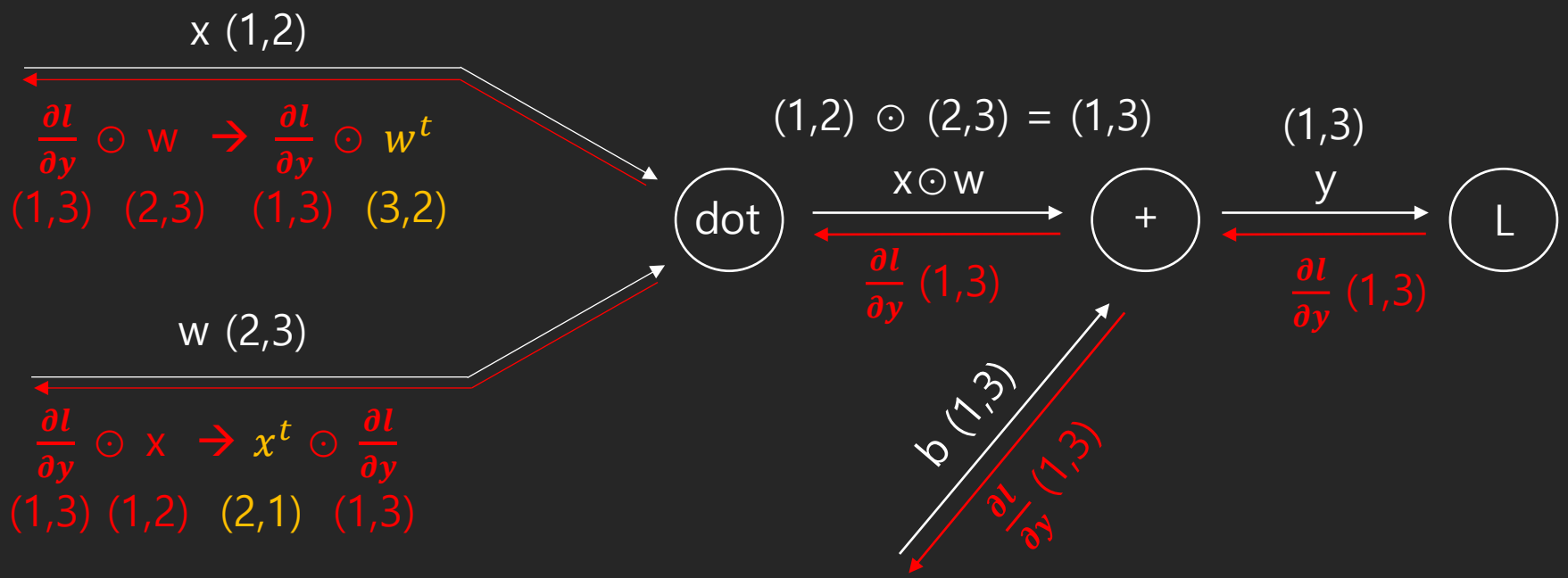
2. 시그모이드(sigmoid) 함수 계산그래프

$$f(x) = \frac{1}{1+e^{-x}}$$



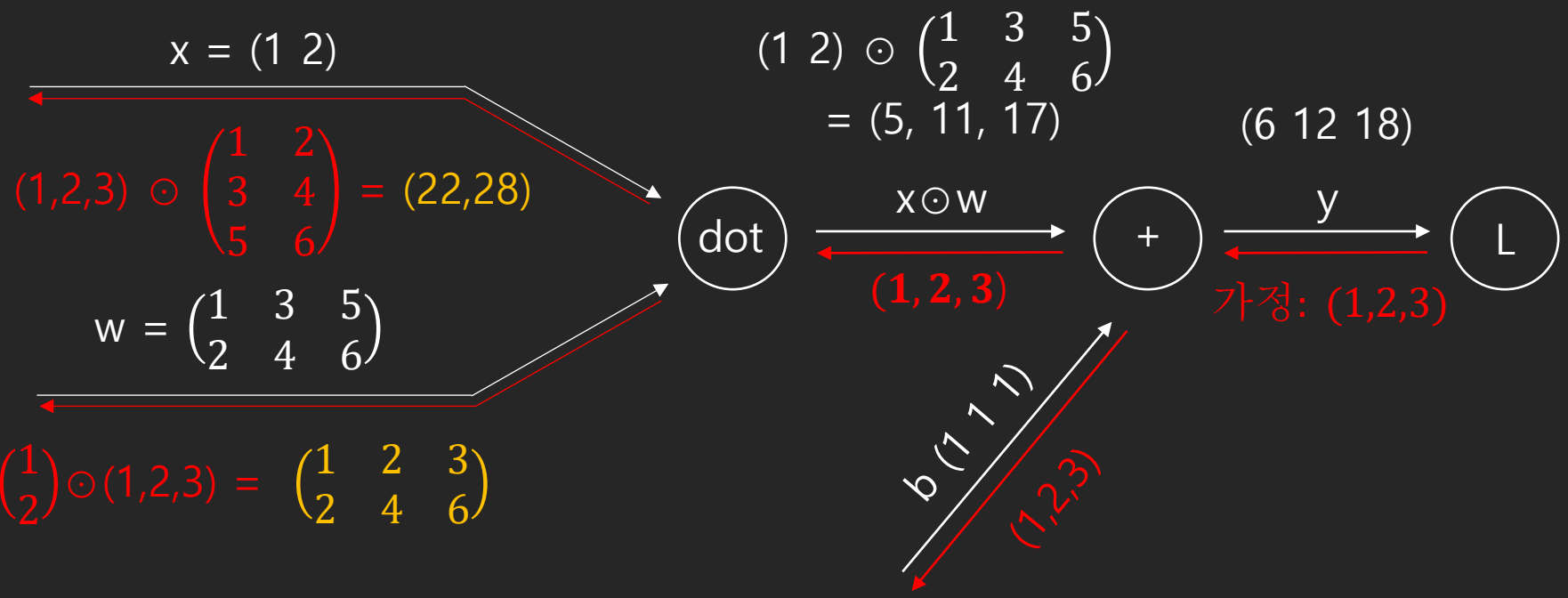
"역전파 일때의 행렬의 shape은 순전파 일때의 행렬의 shape와 동일 "

y = x ⊙ w + b



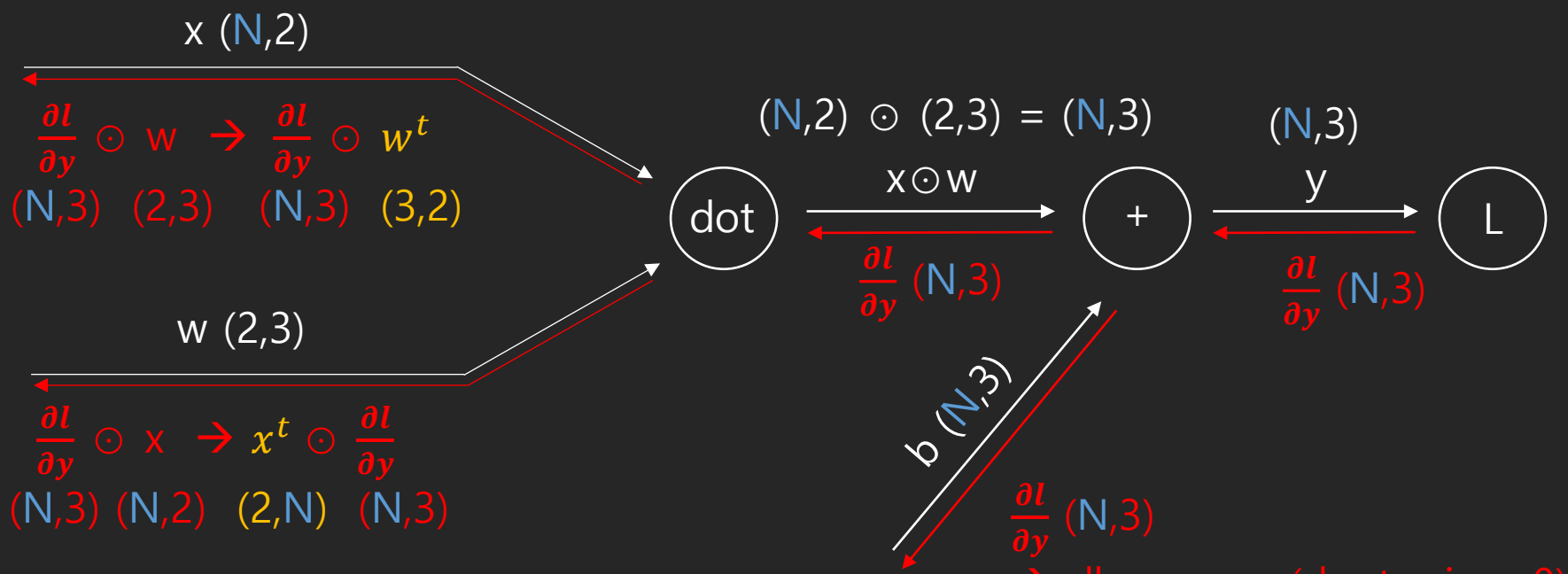
예제1. 실제 행렬 data로 구현

$$y = x \odot w + b$$



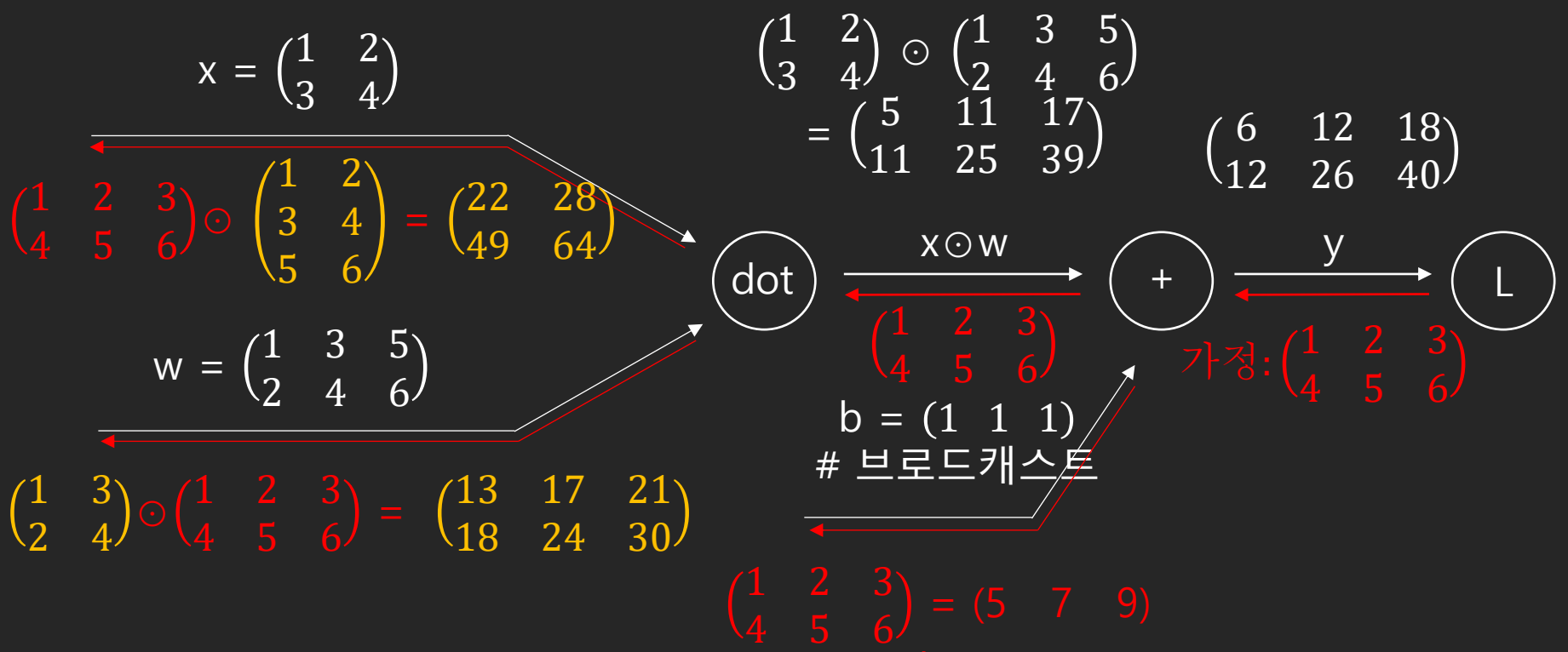
■ Batch용 Affine 계층

$$y = x \odot w + b$$



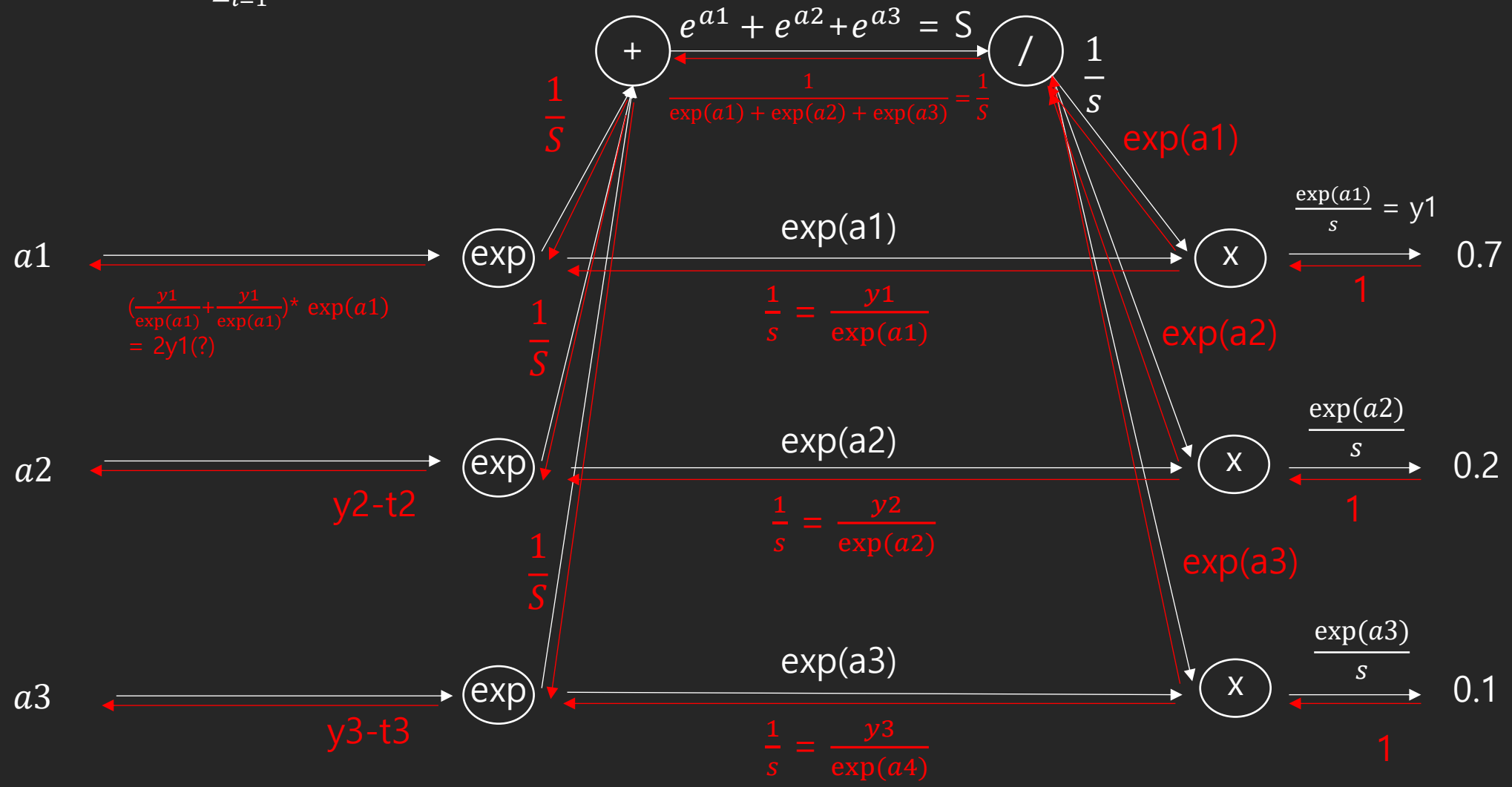
예제1. 다차원 행렬의 Affine 계층

$$y = x \odot w + b$$



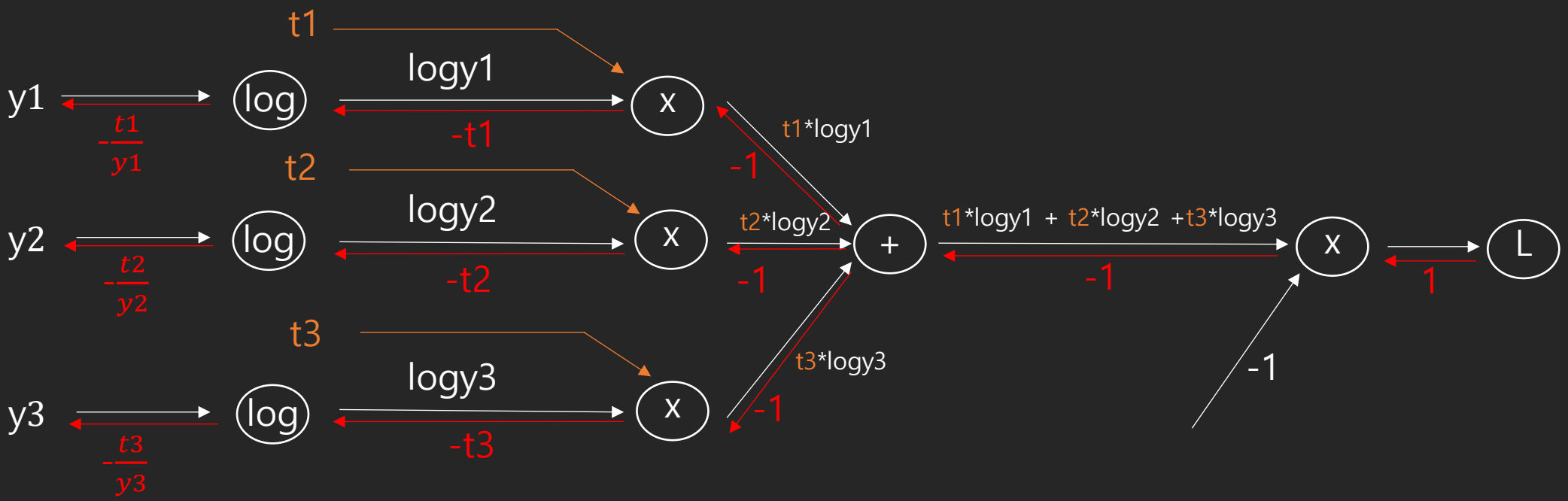
■ 소프트맥스 함수 계산그래프 p294

$$f(x) = \frac{e^{xi}}{\sum_{i=1}^{n} e^{xi}} = \frac{e^{x1}}{e^{x1}+e^{x2}}$$



■ 교차 엔트로피 계산그래프 p294

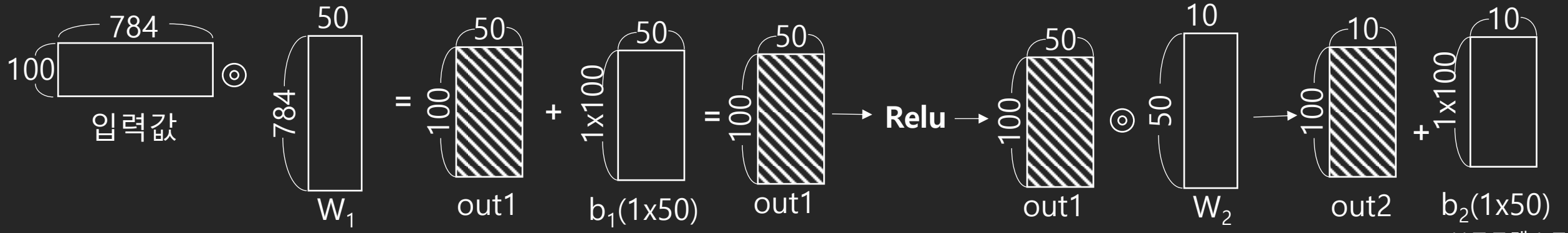
$L = -\sum t_k * logyk$



입력층 (0층) → 은닉 1층 (50개) → 출력층(10개)

Weight_init_std
x np.random.randn(784,50)
→ 0.01 x = 표준편차를 줄이기 위해

self.param['b1']
= np.zeros(hidden_size)



순전파 : Affine 1 → Relu → Affine2 →

역전파 :

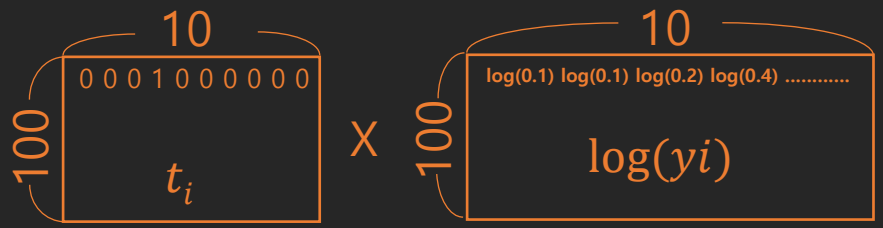
Affine 1: $dx = np.dot(dout, self.W.T)$, $dw = np.dot(self.x.T, dout)$, $db = np.sum(dout, axis = 0)$

Relu: 순전파 일 때 보내지 않는 것은 역전파 일 때는 0

Affine2: $dx = np.dot(dout, self.W.T)$, $dw = np.dot(self.x.T, dout)$, $db = np.sum(dout, axis = 0)$

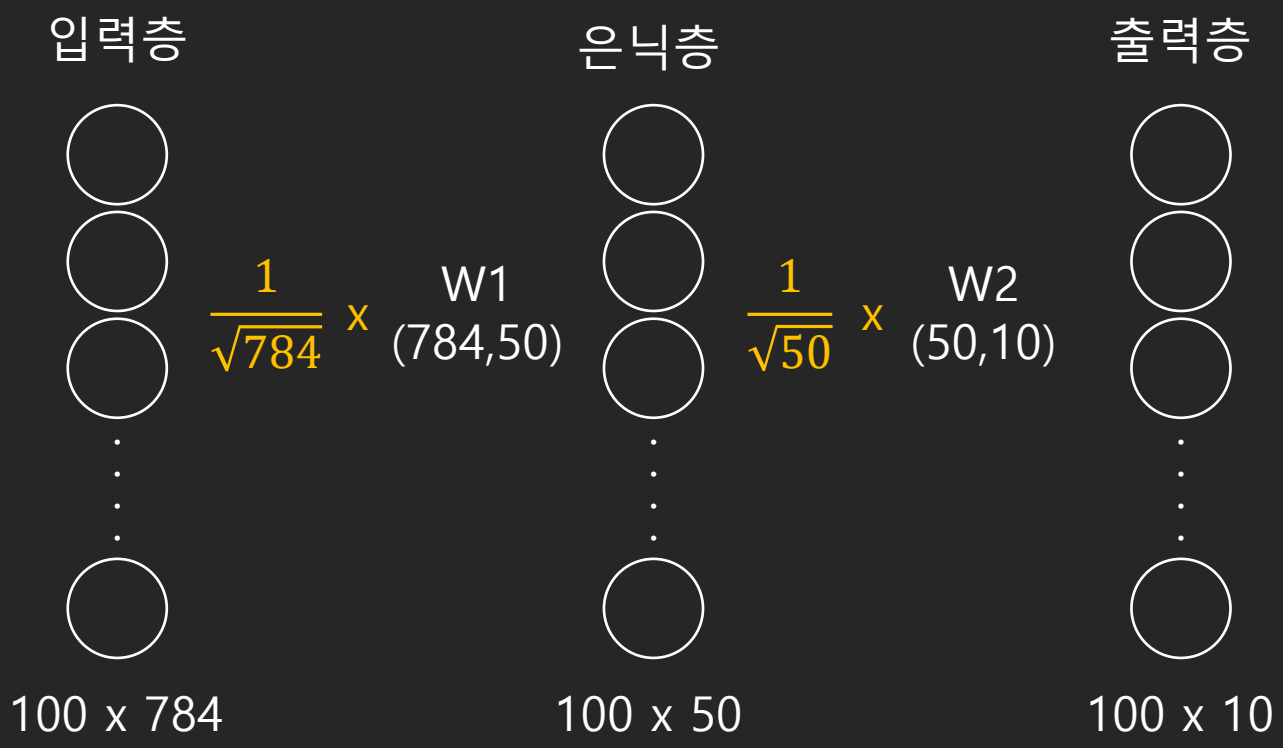
Softmax → 10 / 100 → y = 확률벡터

Cross entropy ($-\sum_{i=1} t_i * \log(yi)$)



Softmax → **Cross entropy**

$y - t(100, 10)$ ← $-\frac{t}{y}$



Xavier = $\frac{1}{\sqrt{\text{전층의 노드 수}}}$ x np.random.randn(784,50) → Sigmoid와 짝꿍

He = $\sqrt{\frac{2}{\text{전층의 노드 수}}}$ x np.random.randn(784,50) → Relu와 짝꿍

입력(숫자2) --> Affine1 ---------> **Batch norm** --> Relu --> Affine2 --> **Batch norm** --> Relu --> Affine3 --> Softmax

↓ ↓

(x◎w)+b --> out --> **Batch norm** --> 활성화함수

(예쁜 정규분포형태를 위해)

↓

1. out값에 대한 평균(μ) = $\frac{1}{m}\sum_{i=1}^{m} xi$

(m =미니배치)

2. 분산(σ^2) = $\frac{1}{m}\sum_{i=1}^{m}(xi-\mu)^2$

→ data의 퍼짐 정도

3. 정규화($\widehat{xi}$) = $\frac{xi-\mu}{\sqrt{\sigma^2+\varepsilon}}$ (ε = 아주 작은값으로 더해서 0으로 나누는 것을 방지)

4. Scale and Shift → yi = $\gamma*\widehat{xi}+\beta$

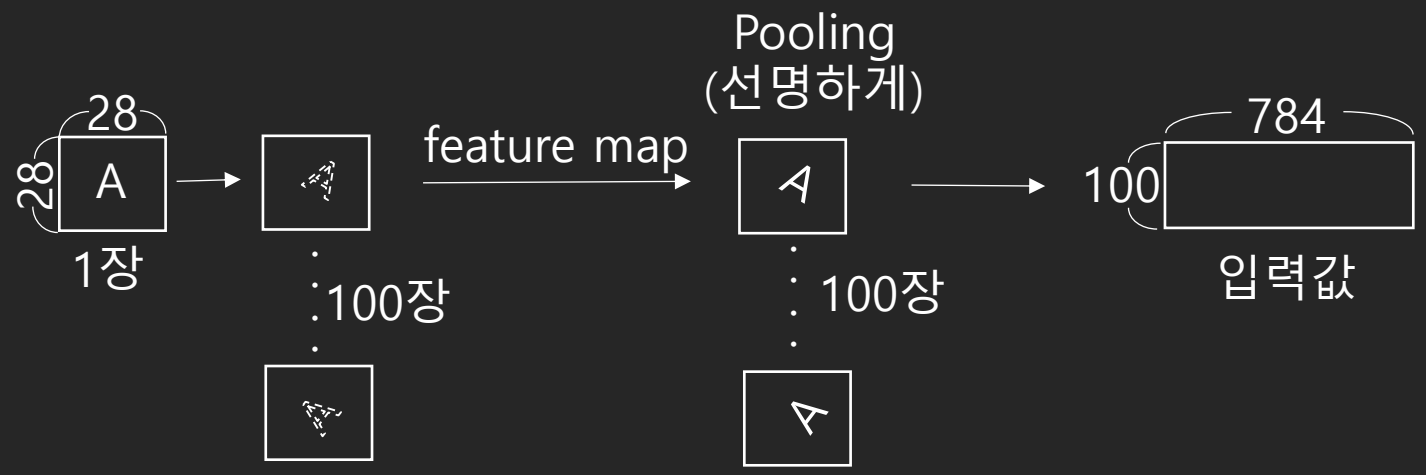
(확대) (이동)

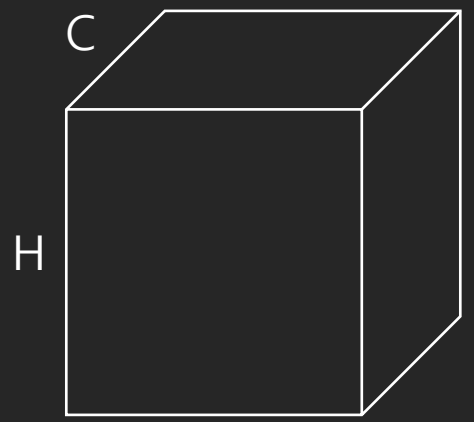
↓ ↓

확대 이동

1 0

(w/b처럼 학습해서 알아내야하는 값)

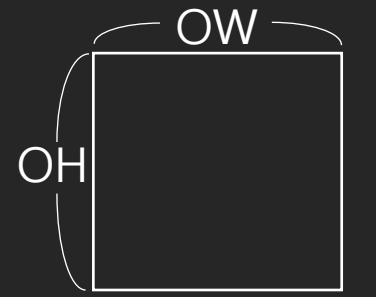

28
28
A
1장
100장
feature map
Pooling
(선명하게)
100장
784
100
입력값



(1,3,28,28)

⊛

(100,3,5,5)

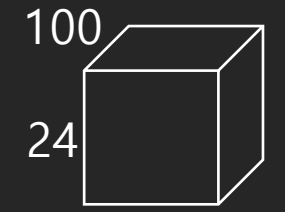


100 장

(100,24,24)

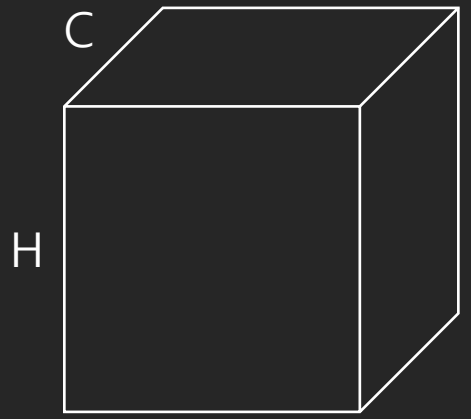
$$OH = \frac{H+2P-FH}{s} + 1 = \frac{28+0-5}{1} + 1 = 24$$

$$OW = \frac{W+2P-FW}{s} + 1 = \frac{28+0-5}{1} + 1 = 24$$



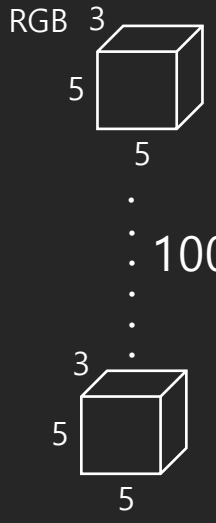
(1,100,24,24)

Feature map

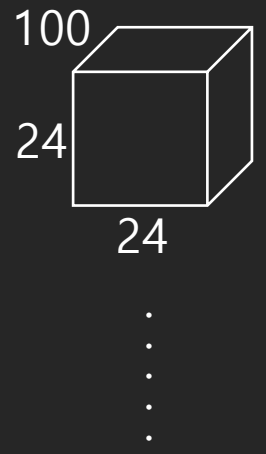


(10,3,28,28)

⊛



(100,3,5,5)



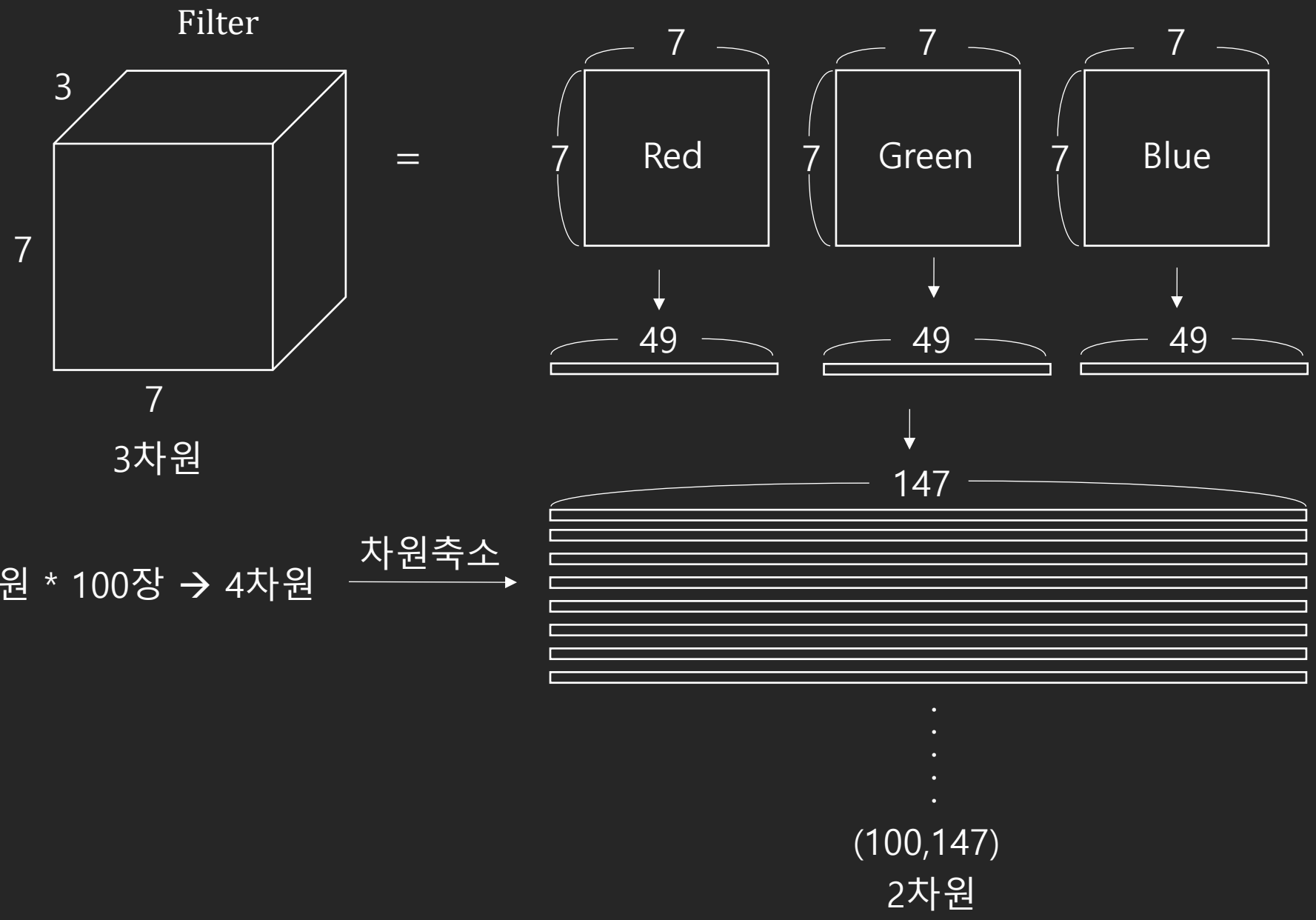
(10,100,24,24)

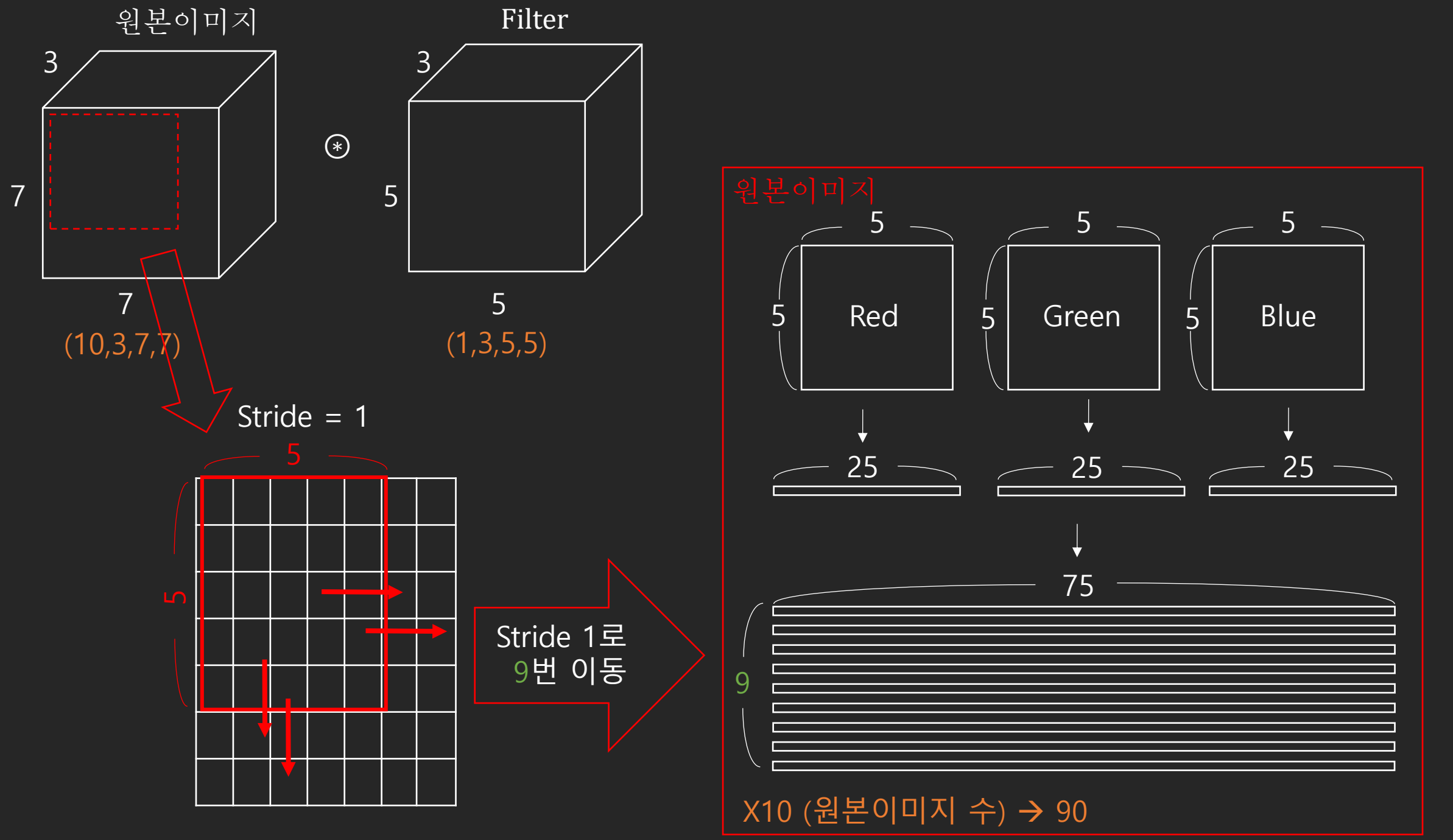
10장

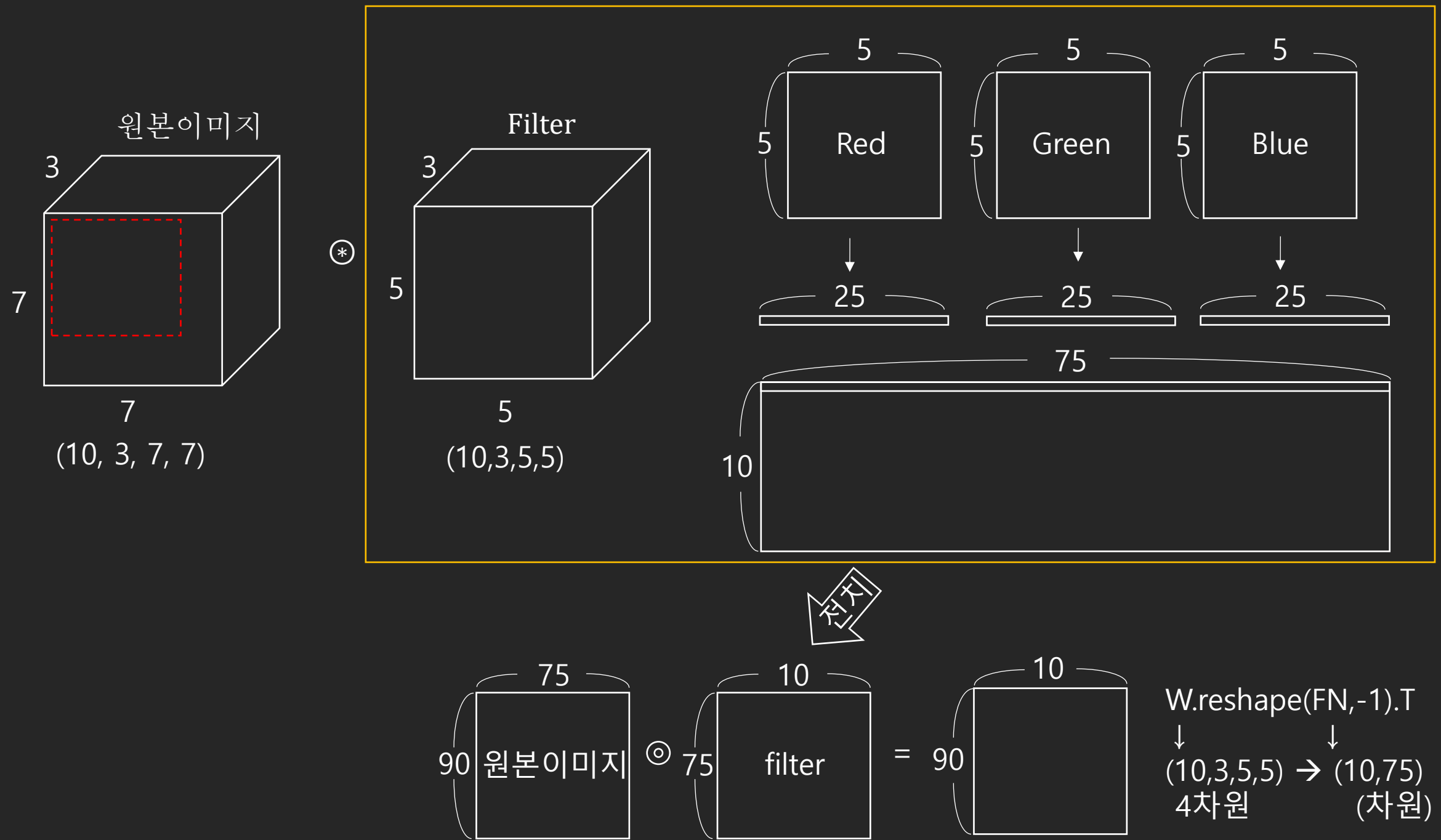
Feature map

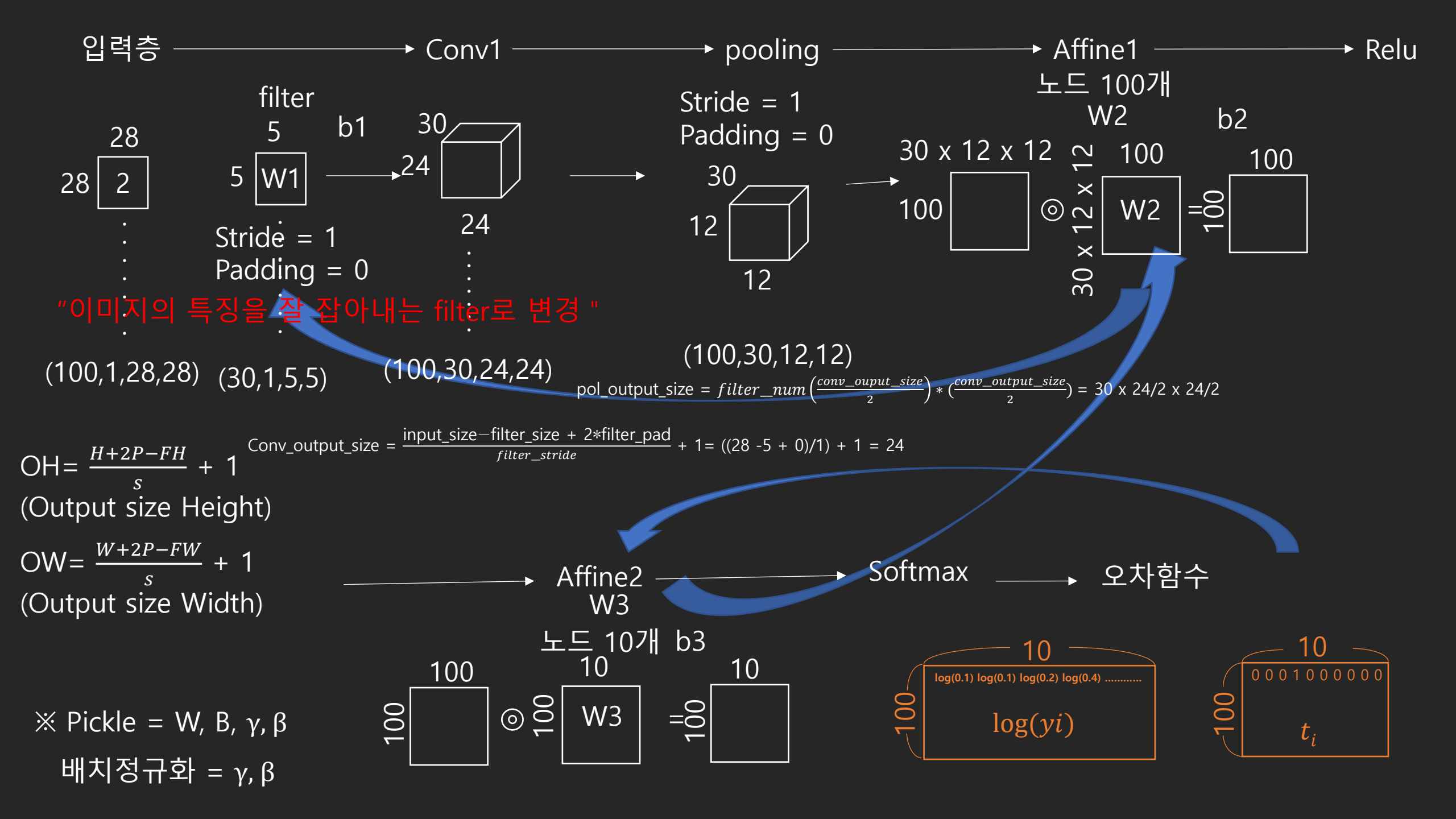
$$OH = \frac{H+2P-FH}{s} + 1 = \frac{28+0-5}{1} + 1 = 24$$

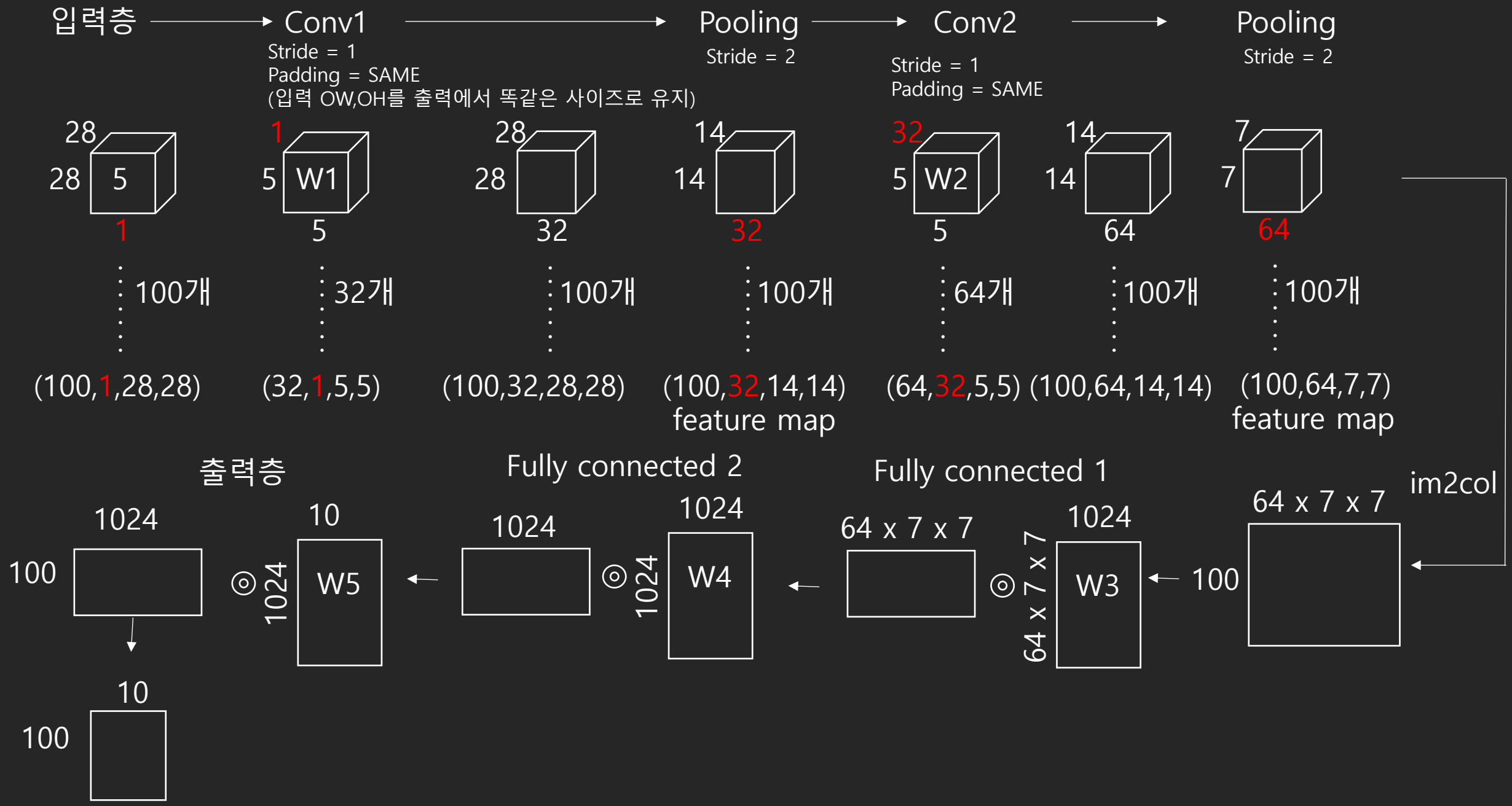
$$OW = \frac{W+2P-FW}{s} + 1 = \frac{28+0-5}{1} + 1 = 24$$

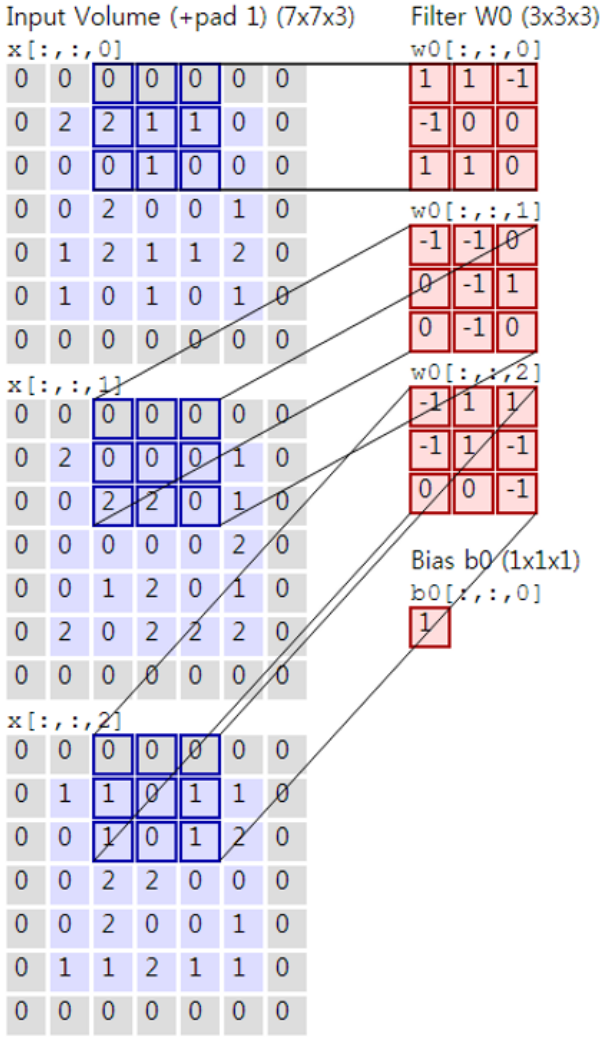

Filter
3
7
7
3차원
=
7
7
Red
7
7
Green
7
7
Blue
49
49
49
147
3차원 * 100장 → 4차원
차원축소
(100,147)
2차원


원본이미지
3
7
7
(10,3,7,7)
⊛
Filter
3
5
5
(1,3,5,5)
Stride = 1
5
5
Stride 1로
9번 이동
원본이미지
5
5
Red
25
5
5
Green
25
5
5
Blue
25
75
9
X10 (원본이미지 수) → 90


원본이미지
3
7
7
(10, 3, 7, 7)
⊛
Filter
3
5
5
(10,3,5,5)
5
5
5
5
Red
5
Green
5
Blue
25
25
25
75
10
전치
75
90
원본이미지
◎
10
75
filter
=
10
90
W.reshape(FN,-1).T
(10,3,5,5)
4차원
(10,75)
(차원)


입력층
Conv1
pooling
Affine1
Relu
filter
5
b1
28
28
2
5
W1
30
24
24
Stride = 1
Padding = 0
Stride = 1
Padding = 0
30
12
12
노드 100개
W2
b2
30 x 12 x 12
100
100
30 x 12 x 12
W2
100
100
"이미지의 특징을 잘 잡아내는 filter로 변경 "
(100,1,28,28)
(30,1,5,5)
(100,30,24,24)
(100,30,12,12)
pol_output_size = $filter\_num\left(\frac{conv\_ouput\_size}{2}\right) * \left(\frac{conv\_output\_size}{2}\right)$ = 30 x 24/2 x 24/2
Conv_output_size = $\frac{input\_size - filter\_size + 2*filter\_pad}{filter\_stride}$ + 1= ((28 -5 + 0)/1) + 1 = 24
OH= $\frac{H+2P-FH}{S}$ + 1
(Output size Height)
OW= $\frac{W+2P-FW}{S}$ + 1
(Output size Width)
Affine2
W3
Softmax
오차함수
노드 10개 b3
100
10
10
100
W3
100
log(0.1) log(0.1) log(0.2) log(0.4) .....
$\log(yi)$
0 0 0 1 0 0 0 0 0 0
t_i
※ Pickle = W, B, γ, β
배치정규화 = γ, β


입력층
Conv1
Stride = 1
Padding = SAME
(입력 OW,OH를 출력에서 똑같은 사이즈로 유지)
Pooling
Stride = 2
Conv2
Stride = 1
Padding = SAME
Pooling
Stride = 2
28
28
5
1
1
5
W1
5
28
28
32
14
14
32
32
5
W2
5
14
14
64
7
7
64
100개
32개
100개
100개
64개
100개
100개
(100,1,28,28)
(32,1,5,5)
(100,32,28,28)
(100,32,14,14)
feature map
(64,32,5,5)
(100,64,14,14)
(100,64,7,7)
feature map
im2col
64 x 7 x 7
100
1024
64 x 7 x 7
W3
Fully connected 1
1024
1024
W4
Fully connected 2
1024
1024
W5
10
출력층
100
1024
100
10



1) 첫번째 채널 : -1

0*1 + 0*1 + 0*(-1) + 2*(-1) + 1*0 + 1*0 + 0*1 + 1*1 + 0*0 = -1

2) 두번째 채널 : -2

0*(-1) + 0*(-1) + 0*0 + 0*0 + 0*(-1) + 0*1 + 2*0 + 2*(-1) + 0*0 = -2

3) 세번째 채널 : -3

0*(-1) + 0*1 + 0*1 + 1*(-1) + 0*1 + 1*(-1) + 1*0 + 0*0 + 1*(-1) = -3

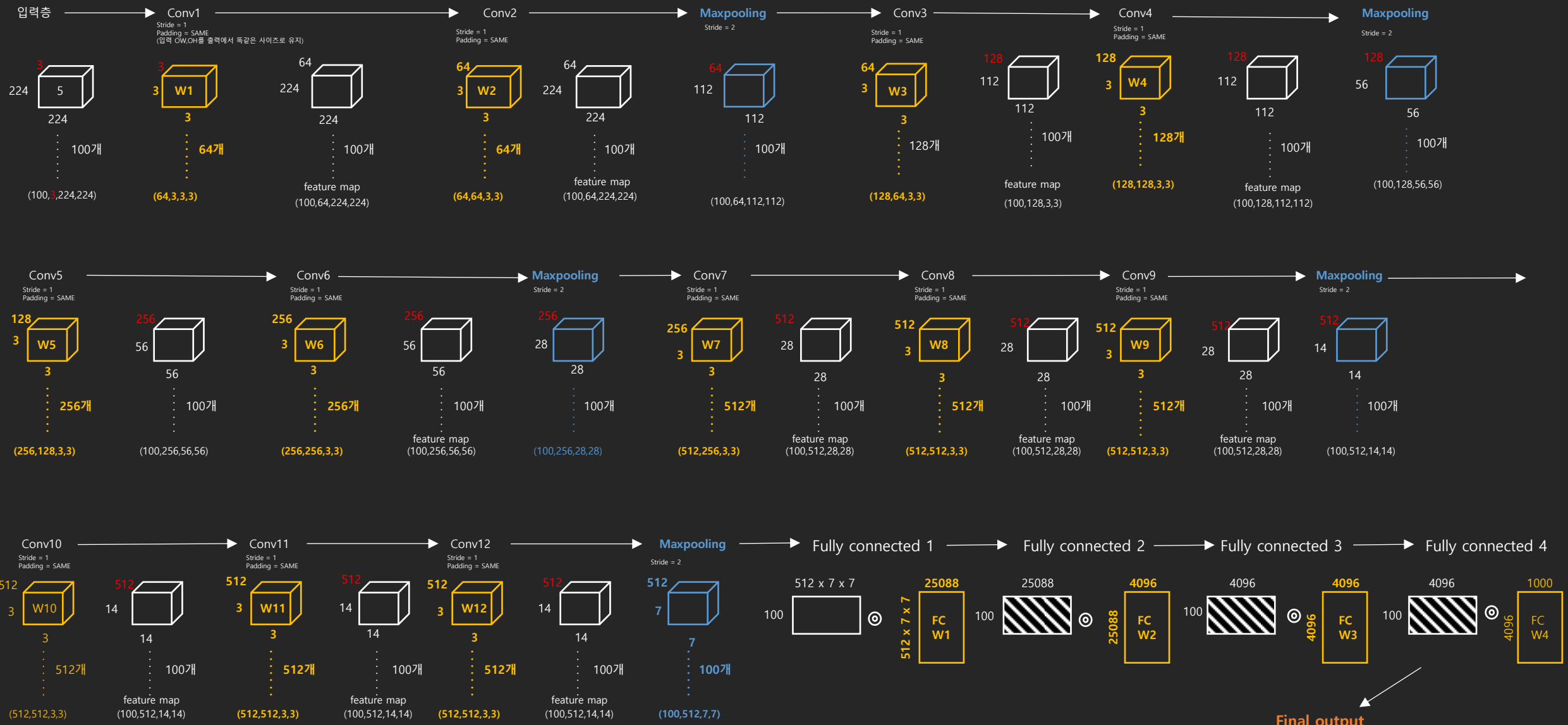
4) 1)+ 2) + 3) = -6

5) 4) + 1 = -5

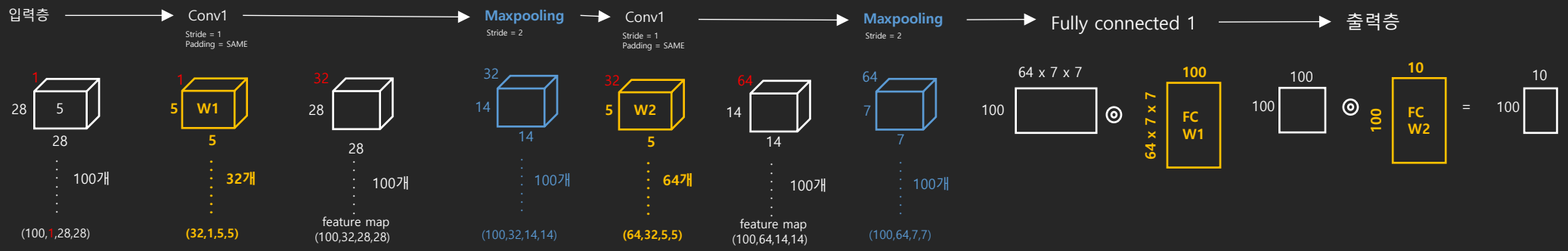
-2	-5	-2
-1	-3	1
0	-2	2

3차원 3차원 → 2차원(R+G+B+b)

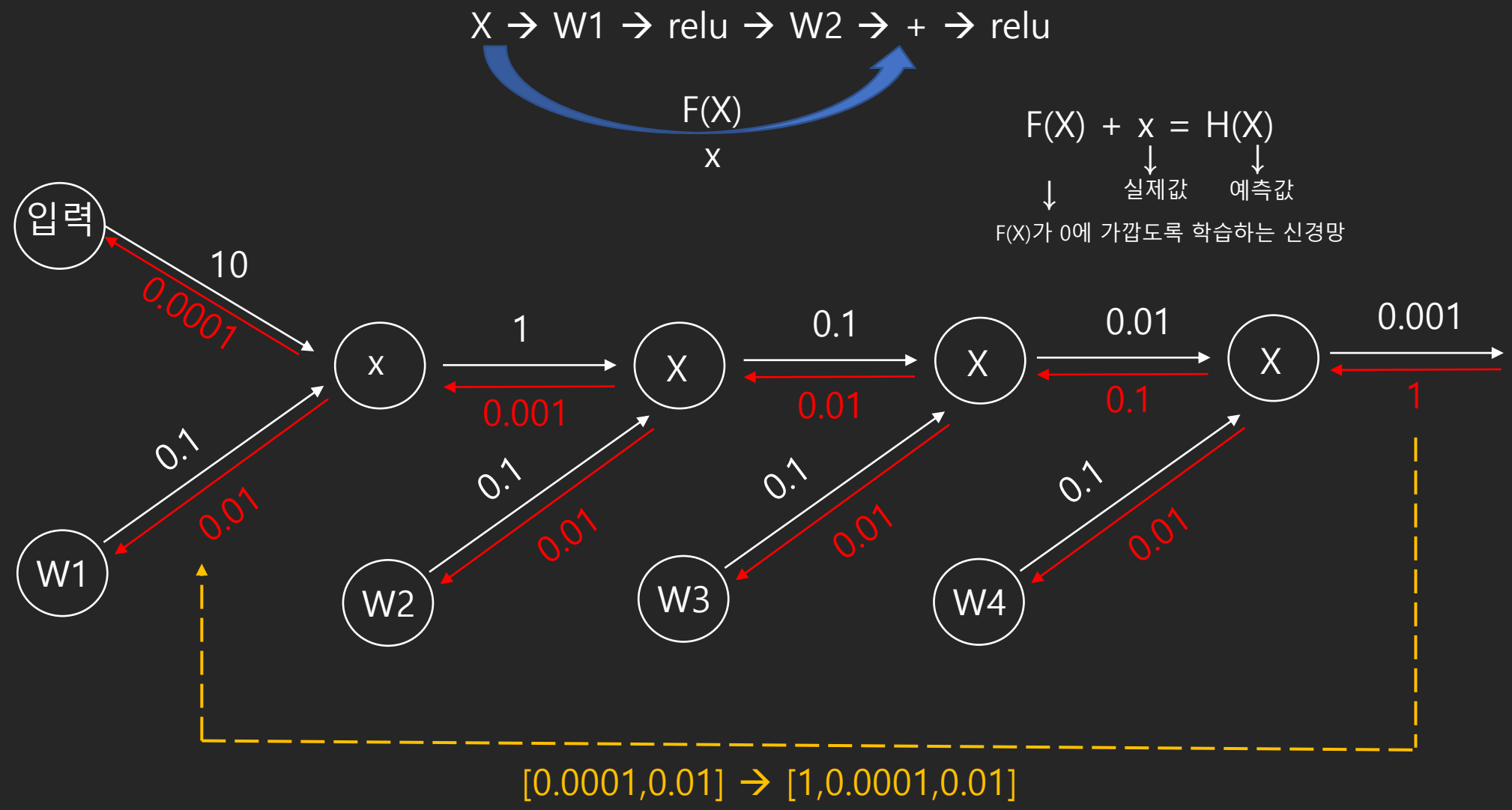
■ 2014년 옥스포드 VGG 신경망 (VGG16)



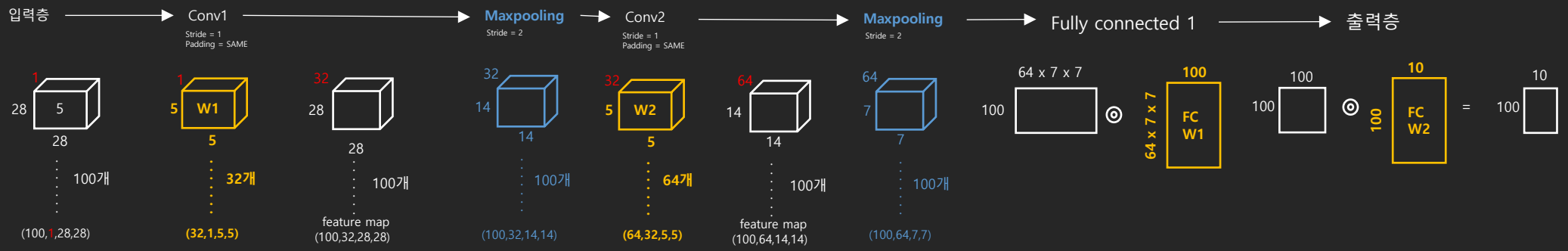
■ CNN층을 텐서플로우로 구현하는 방법



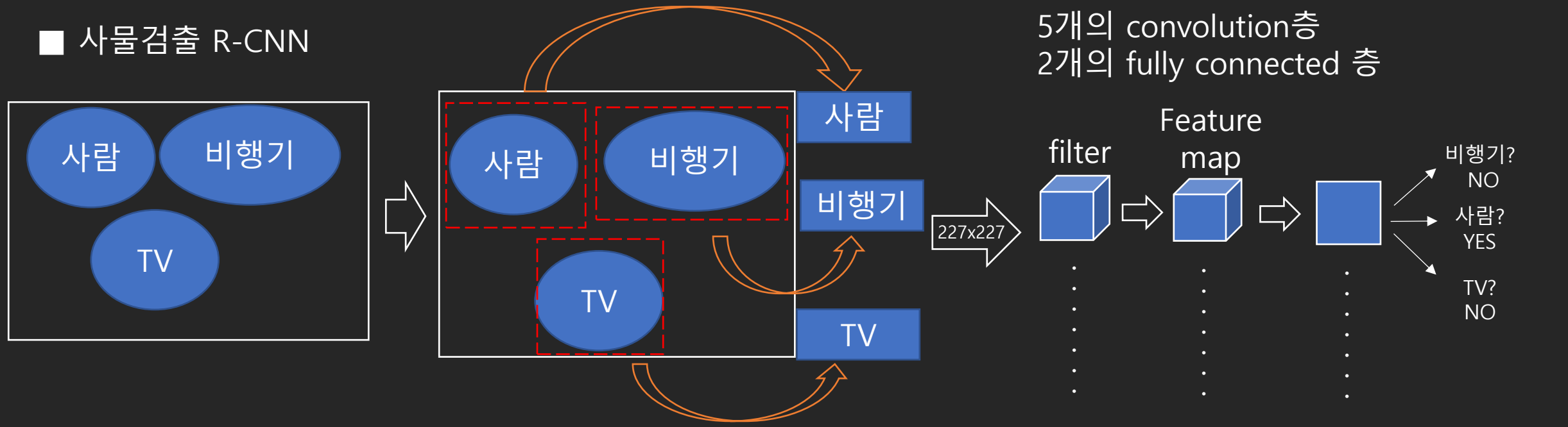
ResNet → 잔차를 학습하는 신경망



■ cifar10 VGG 신경망



■ 사물검출 R-CNN



1. 입력이미지

2. 후보영역검출

3. CNN 특징계산

4. 영역분류

추출된 feature를 SVM으로 분류한다.

첫번째 모듈: Region Proposals

물체가 존재할 확률이 있는 영역을 2000개 추출

↓

Selective search (ss) 알고리즘 사용

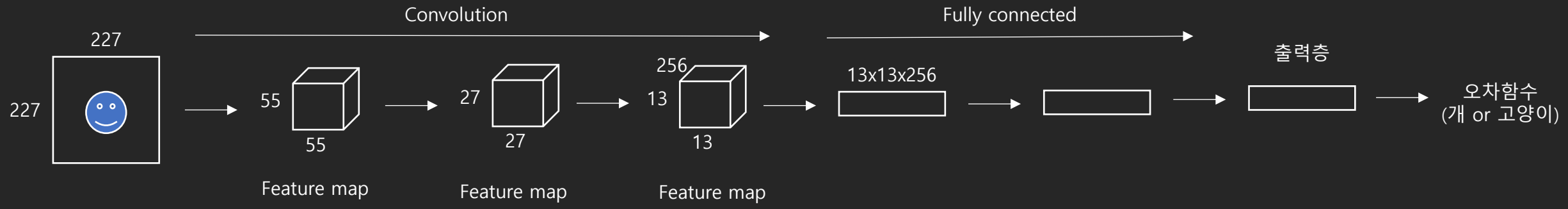
두번째 모듈: Feature extraction

추출된 Region Proposal 영역을 통해 feature를 추출하는 것인데 여기서 CNN이 사용된다.

■ R-CNN vs Faster R-CNN

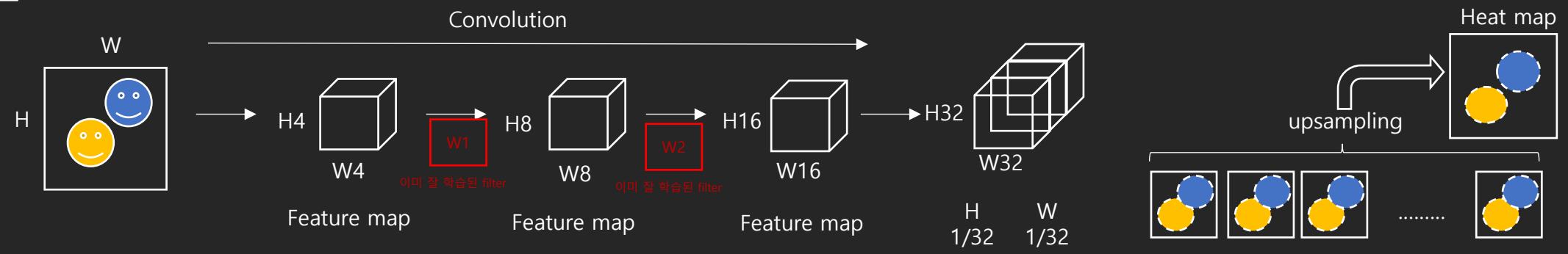
2000장 전체이미지 1장

■ 기존 CNN



■ FCNN

학습이 아닌 위치 정보만 찾기 때문에 Fully connected가 없다
(이미 학습을 따로 해서 정보를 가지고 있다.)



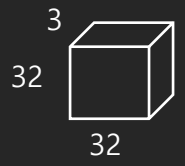
※ FCNN이 CNN과 다른점

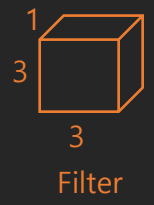
1. Fully connected 층 없다.
2. 입력되는 사이즈가 일정하지 않아도 된다.
   (CNN = 227 x 227)

(1) 기존 합성곱

RGB channel 한번에 합성곱 → 1장 나온다.

(2) Depthwise 합성곱





Chennel 당 각각 합성곱 → 3장 나온다

■ 미술작품 스타일로 일반사진 변환하기(p283)

콘텐츠
(베이스 이미지)
Variable

스타일
(미술작품)
Variable

최종결과물
(합성이미지)
Placeholder

Input tensor

저자가 만들어 온 VGG19

Image Net 데이터 셋에 있는 백만 개 이상의 이미지를 천개이상의 범주로 분류하도록 훈련된 신경망

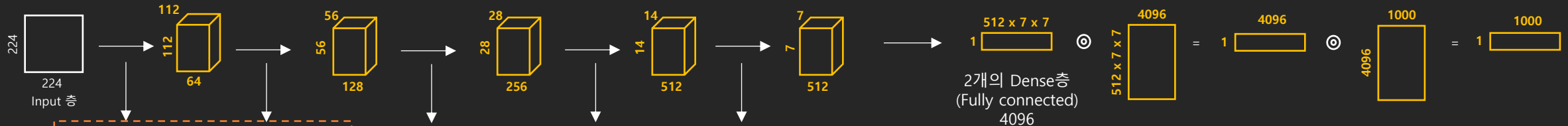
From keras.applications import vgg19

1. Contents loss

콘텐츠 스타일 최종결과물

건물, 강, 하늘과 같은 큰 윤곽이 잘 남아있는지 이미지를 점수화 → 컨텐츠의 손실을 계산해야한다 → 최종 결과물에서 컨텐츠의 윤곽이 남아있어야하므로 컨텐츠의 손실이 최소화되어야 한다.

■ VGG19



2개의 conv2D층
64개 3x3 필터
Conv1_1, s=1, same
Conv1_2, s=1, same
최대풀링 s=2

2개의 conv2D층
128개 3x3 필터
Conv2_1, s=1, same
Conv2_2, s=1, same
최대풀링 s=2

스타일의 손실을 계산하기 위해 사용한 층

4개의 conv2D층
256개 3x3 필터
Conv3_1, s=1, same
Conv3_2, s=1, same
Conv3_3, s=1, same
Conv3_4, s=1, same
최대풀링 s=2

4개의 conv2D층
512개 3x3 필터
Conv4_1, s=1, same
Conv4_2, s=1, same
Conv4_3, s=1, same
Conv4_4, s=1, same
최대풀링 s=2

4개의 conv2D층
512개 3x3 필터
Conv5_1, s=1, same
Conv5_2, s=1, same
Conv5_3, s=1, same
Conv5_4, s=1, same
최대풀링 s=2

콘텐츠의 손실을 계산하기 위해 사용한 층 (사진의 큰 윤곽만 남긴다)

학습하는 코드가 아니기 때문에
마지막 완전영역계층에 가중치를 사용하지 않는다.
Include_top = False

■ 일반사진을 미술작품으로 합성하기 위해 줄여야 할 loss 3가지 →

(1) contents loss

일반사진 ↔ 합성이미지

14층 Conv5_2 사용

(2) style loss

미술작품 ↔ 합성이미지

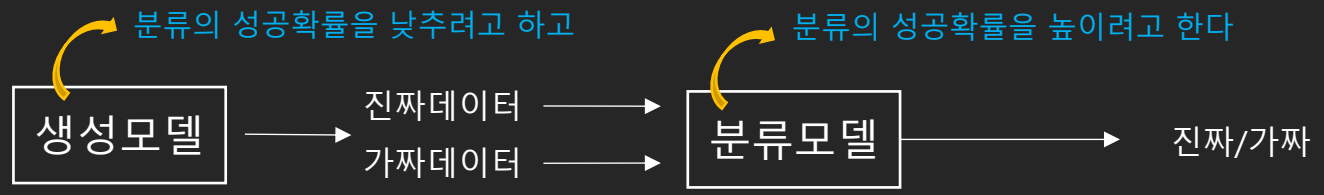
미술작품의 detail을 유지하도록
Detail을 계산해주어야 한다.

(3) 최종결과물 (합성이미지) 잡음 loss

■ GAN (Generative Adversarial Networks)
생성적 적대 신경망

"생성적 적대 신경망"
: 하나의 네크워크가 다른 네크워크와 겨루는 구조

↓ 경찰 (discriminative) 판별자 분류모델 vs ↓ 위조지폐범 (generative) 생성자 생성모델



1. 분류모델에 먼저 진짜 데이터를 입력해서 분류모델 신경망이 진짜를 알 수 있도록 학습을 한다.
2. 생성모델에서 만든 가짜 데이터를 분류 모델에 입력해서 해당 데이터를 가지고 가짜로 분류할 수 있도록 학습을 시킨다. → 이 과정을 통해서 분류모델은 진짜 데이터를 진짜로 가짜데이터는 가짜로 분류할 수 있게 된다
3. 이렇게 분류모델을 학습시킨 다음에는 학습된 분류모델을 속이는 방향으로 학습을 시켜줘야 한다. → 생성된 모델에서 만들어낸 가짜 데이터를 판별모델에 입력하고 가짜 데이터를 진짜라고 분류할만큼 진짜데이터와 유사한 데이터를 만들도록 학습을 시킨다.
4. 1~3의 학습과정을 반복하면 분류모델과 생성모델이 서로 적대적인 경쟁자로 인식하여 모두 발전하게 된다.

■ GAN (Generative Adversarial Networks)
생성적 적대 신경망

분류자이고 진짜일 확률을 의미하는 0~1사이의 값
→ 데이터가 진짜면 1
→ 데이터가 가짜면 0

G가 만들어낸 데이터인 G(z)가 진짜라고 판단되면 1,
가짜라고 판단되면 0을 내놓는다

$$\min_G \max_D V(D,G) = \underbrace{E_{x\sim P_{data}(x)}[\log D(x)]}_{\text{첫번째 항}} + \underbrace{E_{z\sim P_z(z)}[\log(1-D(G(z)))]}_{\text{두번째 항}}$$

실제 data에 대한 확률분포에서 샘플링한 data

가우시안 분포를 사용하는 임의 노이즈에서 샘플링한 data

1. D의 입장에서 첫번째 항과 두번째 항이 최대가 되어야하므로 D(x)가 1이 되어야하고 1-D(G(z))가 1이 되어야 해서 **D(G(z))**가 0이어야 합니다.

생성자가 만들어낸 가짜데이터를 가짜라고 분류하도록 학습하는 것을 의미합니다.

2. G의 입장에서 첫번째 항과 두번째 항이 최소가 되어야하므로 ~~D(x)가 0이 되어야하고~~ 1-D(G(z))가 0이 되어야 해서 **D(G(z))**가 1이어야 합니다.
(G가 첫번째 항에 없기 때문에 생략)

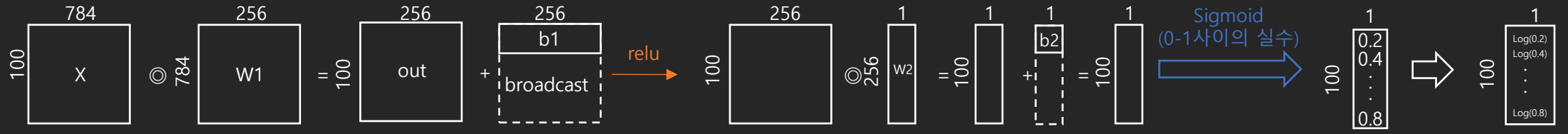
판별자가 진짜로 분류할만큼 가짜데이터를 생성하도록 생성자를 학습시키는 것이다. G가 없으므로 생략가능하다

Discriminator (판별자)

```
X = tf.placeholder(tf.float32, [None, 784])
result_of_real = discriminator(X , True)
d_loss = tf.reduce_mean( tf.log(result_of_real) + tf.log(1 - result_of_fake) )
```

↓

얼마나 discriminator 정확한가? d_loss는 높을 수록 좋다.

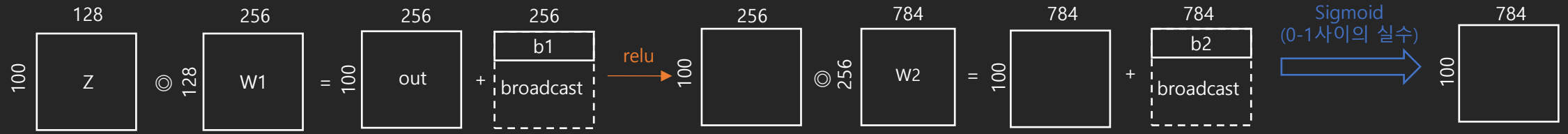


Generator (생성자)

```
Z = tf.placeholder(tf.float32, [None, 128])
fake_x = generator(Z)
result_of_fake = discriminator(fake_x)
g_loss = tf.reduce_mean( tf.log(result_of_fake) )
```

↓

얼마나 fake_x가 진짜 같은가? --? 최대화 해야한다.

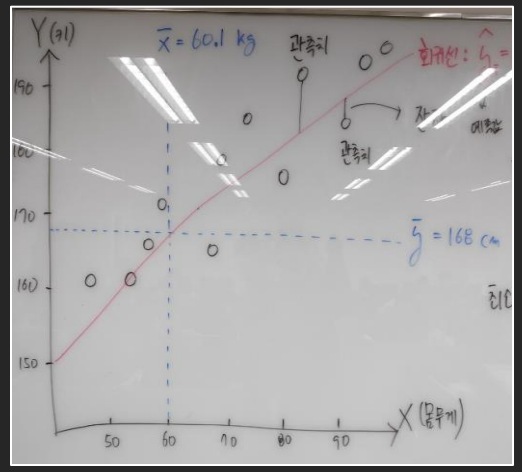


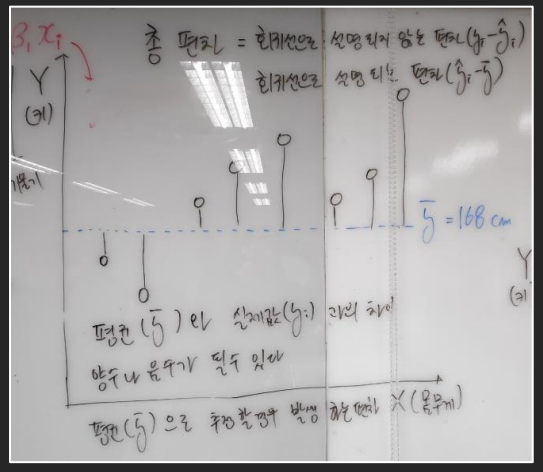

```
optimizer = tf.train.AdamOptimizer(learning_rate=0.0002)

g_train = optimizer.minimize(-g_loss, var_list= g_vars)
                                                 가중치들
d_train = optimizer.minimize(-d_loss, var_list = d_vars)
```

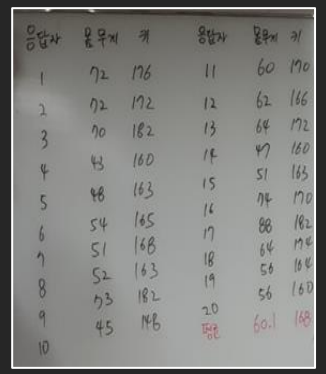
*optimizer는 최대화해야하는데 minimize 밖에 없어서 –를 붙여줘서 최대화시킴

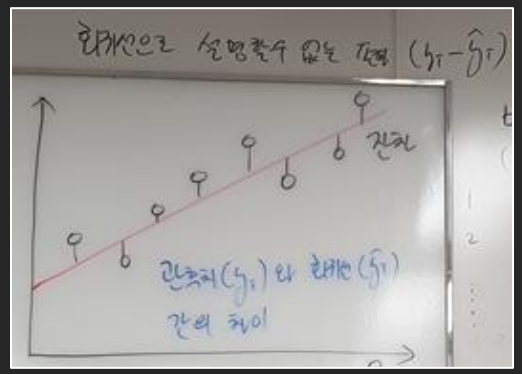
회귀선 : $\hat{y}_i = \beta_0 + \beta_1 x_i$

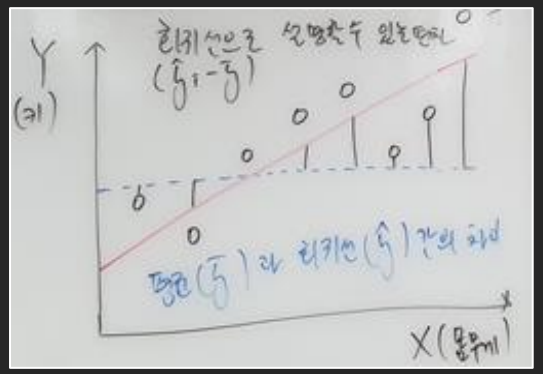
- 예측값
- 절편
- 변수
- 기울기
- 최소자승법











변수X (몸무게)	변수Y (키)	편차X $(x_i - \bar{x})$	편차Y $(y_i - \bar{y})$	편차곱 $(x_i - \bar{x})\ (y_i - \bar{y})$	편차제곱 $(x_i - \bar{x})^2$

$\bar{x} = 60.10$ $\bar{y} = 168$

$$\sum (x_i - \bar{x})\ (y_i - \bar{y}) = 1673$$

$$\sum (x_i - \bar{x})^2 = 2693.80$$

$$\text{기울기}(\beta_1) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})\ (y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} = \frac{1673}{2693.80} = 0.621$$

$$\text{절편}(\beta_0) = \bar{y} - \beta_1 \bar{x} = 168 - 0.621 \times 60.10 = 130.678$$

총 편차 = 회귀선으로 설명되지 않는 편차 $(y_i - \hat{y}_i)$ + 회귀선으로 설명되는 편차 $(\hat{y}_i - \bar{y})$

결정계수(R^2)란?

- 1에 가까울수록 회귀선은 설명력이 높은 바람직한 회귀선이 된다.
- 추정된 회귀선($\hat{y}_i$)이 실제값(y_i)과 평균($\bar{y}$) 사이의 편차를 얼마나 줄여주는가를 나타내는 지수

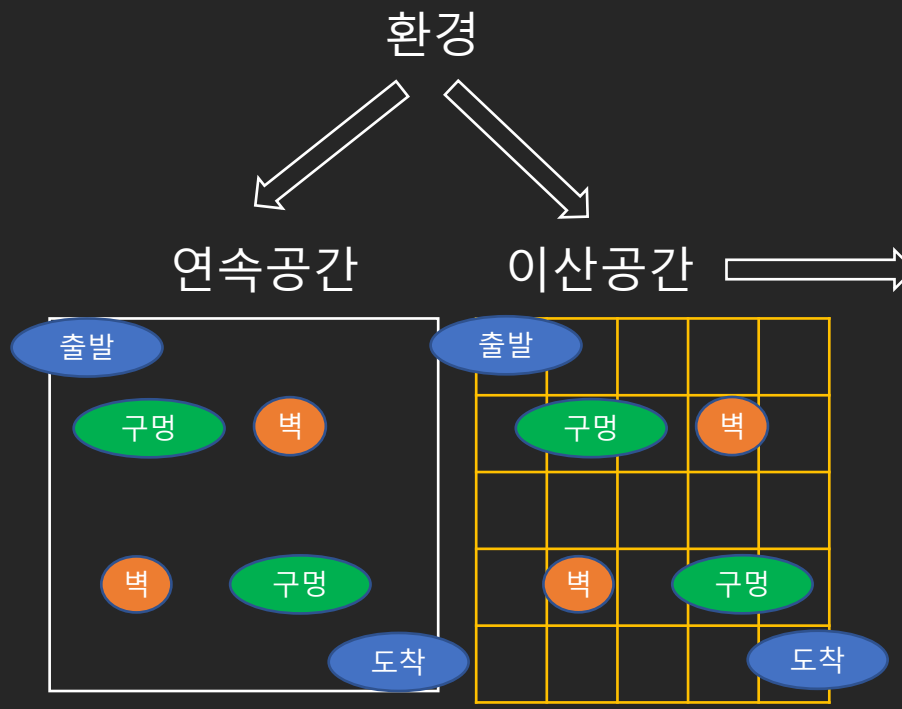
$$R^2 = \frac{\text{회귀선에 의해 설명되는 제곱합}}{\text{총 제곱합 (회귀선에 의해서 설명되지 않는 제곱합 + 회귀선으로 설명되는 제곱합)}}$$

$$= 1 - \frac{\text{회귀선에 의해 설명되지 않는 제곱합}}{\text{총 제곱합}}$$

Part2. 강화학습

■ 환경

" 강화학습을 이용해 풀고자하는 대상이나 문제"



연속공간을 일정한 구간으로 나눈 환경
연속공간의 문제를 풀기를 쉽지 않기 때문에
어떻게든 이산공간으로 나누는 과정이 필요하다

■ 상태

" 학습하는 주체가 위치하거나 감지하고 있는 상태 "

출발지점

S0	**S1**	**S2**
S3	**S4**	**S5**
S6	**S7**	**S8**

도착지점

St는 시간 t시점에서 에이전트가 위치한 상태

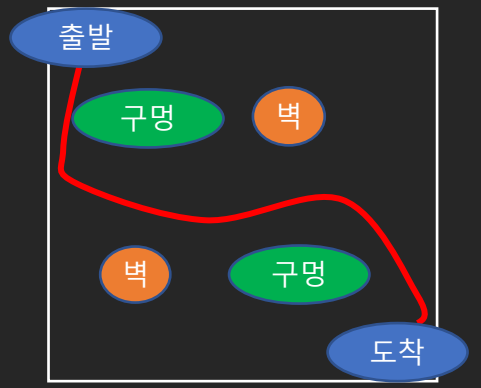
S = {S0, S1, S2, S3, S4, S5, S6, S7, S8}

■ 에이전트

"환경에 대해 특정 행동을 하고 학습하는 프로그램이나 로봇 "

예: 문제 에이전트

미로 탐색 문제 로봇
자동주식 트레이딩 주식 트레이딩 프로그램



■ 행동(A)

"에이전트가 상태 S에서 할 수 있는 행동 "

예: 문제 에이전트

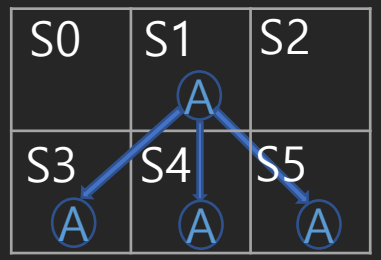
로봇의 미로 탐색 문제 상하좌우 이동, 멈춤, 회전
자동주식 트레이딩 사기, 팔기, 지켜보기
(Buy) (Sell)

■ 상태전이확률(P)

"시간 t일 때 에이전트가 상태 S에서 행동 a를 취했을 때 S'로 이동할 확률"
$P(S'|s,a) = Pr[St+1 = S'|St=s, A=a]$
S'로 이동했을 때 시간은 t의 다음 시간이기 때문에 t+1이 된다.

예: 결정론적 환경 확률적 환경

$P(S4|s1,a3) = 1$

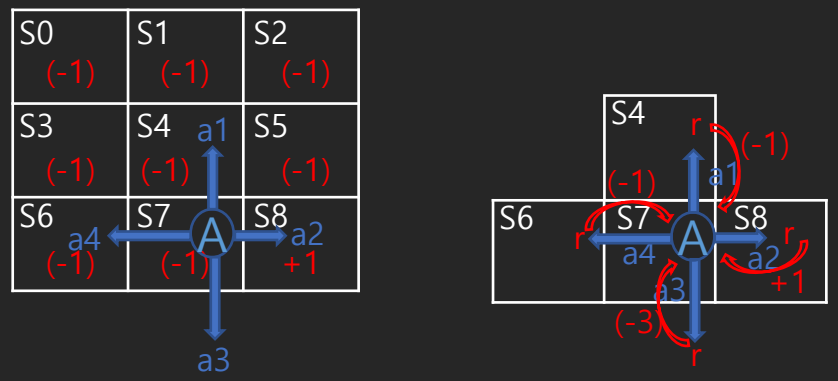


$P(S3|s1,a3) = 0.1$
$P(S4|s1,a3) = 0.8$
$P(S5|s1,a3) = 0.1$

■ 보상

"에이전트가 취한 행동에 대해 환경으로부터 좋고 나쁨의 평가를 수치적으로 받는 것"

예:



식: $r(s, a, s') = E[r_{t+1} | S_t = s, A_t = a, S_{t+1} = S']$

현재 행동 다음 기대값

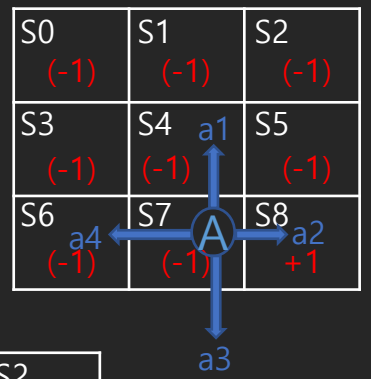
■ 수익(G) ----> 보상의 총합 ----→ 수익을 극대화하려면 감가율이 있어야한다. --> 에이전트가 가장 큰 수익을 주는 행동을 할 수 있도록 학습하게 해준다.

"시간 t로 부터 에이전트가 계속적으로 행동을 취한다고 가정했을 때 연결된 상태를
계속 이동하면서 계속해서 받는 보상(reward)의 총합"

예: $G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots\cdots\cdots\cdots = \sum_{k=0}^{\infty} r_{t+k+1}$

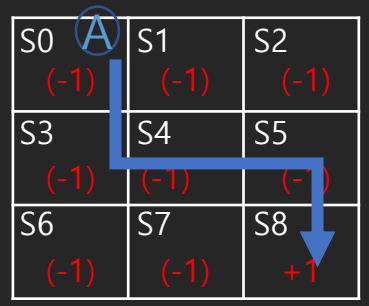
월급 100만원 ----→ 10년 ----→ 차이가 크다.
월급 1000만원
월급 1억 ------> ∞ ----→ 차이가 의미없다

그래서 도입된게 감가율



$$G_t = \gamma^0 r_{t+1} + \gamma^1 * r_{t+2} + \gamma^2 r_{t+3} + \cdots\cdots\cdots$$
$$= \sum_{k=0}^{\infty} \gamma^k * r_{t+k+1}$$

0~1사이의 감가율을 도입하면 수익G가
무한히 커지는 것을 방지할 수 있다.



- 감가율이 0.1일 때

  $G(수익) = 0.1^0 x -1 + 0.1^1 x -1 + 0.1^2 x -1 + 0.1^3 x 1 = 0.12$

- 감가율이 0.9일 때

  $G(수익) = 0.91^0 x -1 + 0.9^1 x -1 + 0.9^2 x -1 + 0.9^3 x 1 = -1.98$

수익의 차이가 난다

■ 정책(π)

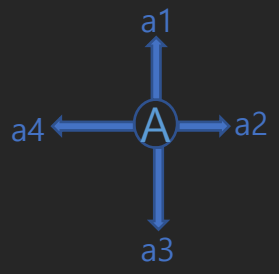
"에이전트가 어떤 상태에 있을 때 어떤 행동을 선택할지
결정하는 기준을 정책이라고 한다."

예: $\pi(a|s) = Pr[Ar = a \mid St = S]$

학구파	놀자파
π(치맥|내일시험) = 0.1	π(치맥|내일시험) = 0.9
π(공부|내일시험) = 0.9	π(공부|내일시험) = 0.1

"강화학습에서 가장 중요한 것은 최적 정책을 찾는 것이다.
최적 정책이란 어떤 상태에서 수익G가 최대가 되는 행동을 선택하는 정책을 의미한다."

$\Pi(a1|S4) = 0$
$\Pi(a2|S4) = 0.5$
$\Pi(a3|S4) = 0.5$
$\Pi(a4|S4) = 0$

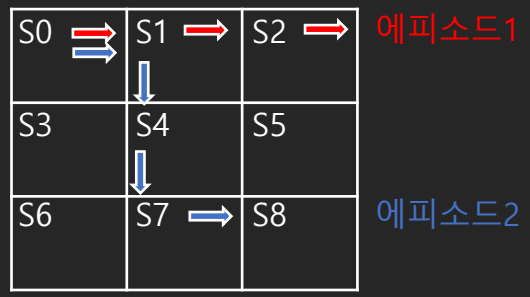


S0 (-1)	S1 (-1)	S2 (-1)
S3 (-1)	S4 (-1) A	S5 (-1)
S6 (-1)	S7 (-1)	S8 +1

$\Pi(a1|S5) = 0$
$\Pi(a2|S5) = 0$
$\Pi(a3|S5) = 1$
$\Pi(a4|S5) = 0$

■ 에피소드

"주어진 문제에서 초기 상태로부터 시작해서 목적완료에 의한
성공/상태/도착 이나 실패로 인한 상태 종료까지의 일련의 과정"



■ 마르코브 의사결정(MDP)

"강화학습 문제를 풀기 위해서는 문제를 MDP의 기본 요소로
정의하는 것부터 시작한다"

1. S : State
2. A : Action
3. P : State transition Probability(상태전이확률)
4. R : Reward
5. γ : discount faction(감가율)

문제1. (점심시간문제) 얼음판 게임을 강화학습시키기 위한 MDP기본 요소를 정의하시오!

1. S : State → {s0, s1, s2, s3, s4, s5, s6, s7, s8}
2. A : Action → {a1, a2, a3, a4}
3. P : State transition Probability(상태전이확률) → P(s'|s,a) = 1
4. R : Reward →
   - +1: s8도착
   - -1: 미로 안에서 이동
   - -3: 미로 밖으로 이동
5. γ : discount faction(감가율) → 0.9

S4에서 -2.12인 이유

$V_\pi(S_t) = r_{t+1} + \gamma \times V_\pi(S_{t+1})$

= -1.0 + 0.9 x (-1.5-1.0-1.0-1.5)/4

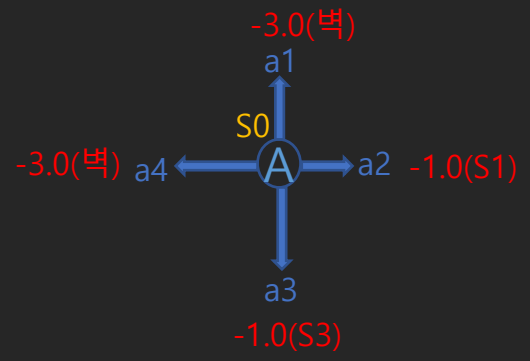
= -1.0 + 0.9 x (-1.25) = -1.0 + -1.125= -2.125

S0 **-2.0**	S1 -1.5	S2 -2.0
S3 -1.5	S4 -1.0	S5 -1.0
S6 -2.0	S7 -1.0	S8 +1.0

1번 반복

S0 -3.58	S1 -2.96	S2 -3.46
S3 -2.96	S4 -2.12	S5 -1.68
S6 -3.46	S7 -1.68	S8 +1.0

S0에서 -2.0이 나오는 이유



-3/4 + -3/4 + -1/4 + -1/4 = -8/4 = - 2.0

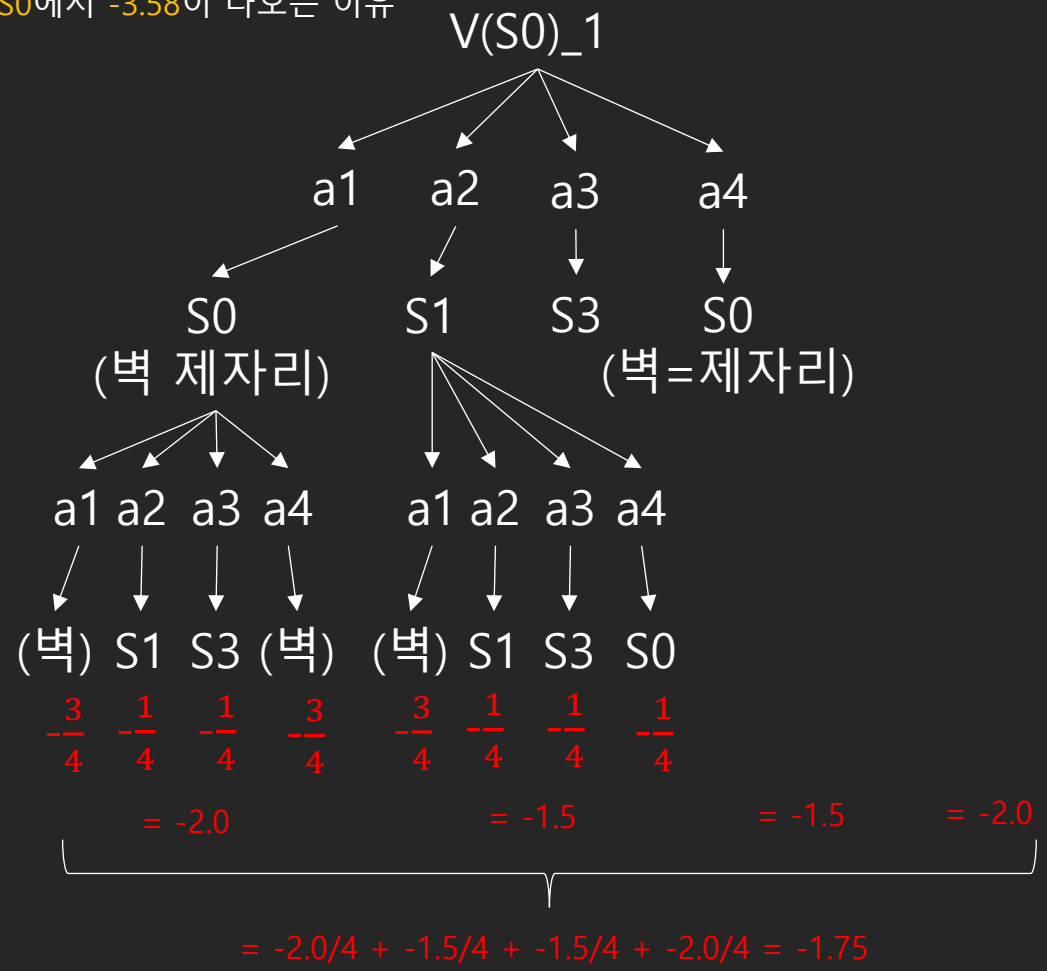
S1에서 -2.96이 나오는 이유

$V_\pi(S_t) = r_{t+1} + \gamma \times V_\pi(S_{t+1})$

= -1.5 + 0.9 x (-1.5-2.0-2.0-1.0)/4

= -1.5 + 0.9 x (-1.625) = -1.5 + -1.4625= -2.9635

S0에서 -3.58이 나오는 이유



$V_\pi(S_t) = r_{t+1} + \gamma \times V_\pi(S_{t+1})$

= -2.0 + 0.9 x (-1.75)

= -3.575

1.강화학습의 요소 10가지
2.가치함수
- 상태가치함수 V_π p33
- 행동가치함수 Q_π p46
  --> 수익(G)를 높게 받을 수 있는 장소(도착지점에 가까운) 가치가 높은 것을 알 수 있다.

3.최적 정책선택
- 정책 평가 p61
- 정책 개선 p63

■ 정책평가 → p59

$V(S) \leftarrow P(S'|s,a)[r(s,a,s') + \gamma x\ V(S')]$

Vo(S) K=0

S0 0	S1 0	S2 0
S3 0	S4 0	S5 0
S6 0	S7 0	S8 0

V1(S) K=0

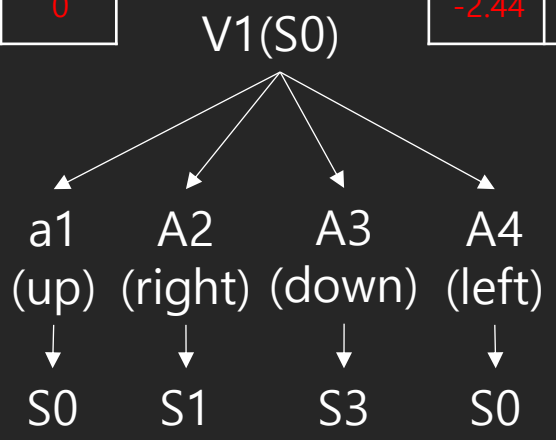
-2.0	-1.95	-2.44
-1.95	-1.88	-1.97
-2.44	-1.97	1.1

V2(S) K=2

-3.78	-3.76	
-3.76		

K=133

-11.72	-9.88	-8.87
-9.88	-6.76	-2.93
-8.87	-2.93	10.00



S0에 대한 k=1 일때의 가치

A1 = ¼ * (-3 + 0.9x-2.0) = -1.2
A2 = ¼ * (-1 + 0.9x-1.95) = -0.688
A3 = ¼ * (-1 + 0.9x-1.95) = -0.688
A4 = ¼ * (-3 + 0.9x-2.0) = -1.2

Sum = -3.78

S0에 대한 k=0 일때의 가치

A1 = ¼ * (-3 + 0.9x0) = -0.75
A2 = ¼ * (-1 + 0.9x0) = -0.25
A3 = ¼ * (-1 + 0.9x0) = -0.25
A4 = ¼ * (-3 + 0.9x0) = -0.75

Sum = -2.0

총 계산시간: 4초
(정책평가)
총 계산시간: 약 25분
(가치평가)

■ 행동 가치 함수 → p46~52

Qπ = 어떤 상태 S에서 최적의 행동 a가 무엇인지 찾기 위한 함수

$$V_\pi(S) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)\ [r(s,a,s') + \gamma * V(S')]$$

확률 보상 감가율 가치값

$$Q_\pi(s,a) = \sum_{s' \in S} P(s'|s,a)\ [r(s,a,s') + \gamma * V(S')]$$

Max_step = 0

-0.75	S1 -1.5	S2 -2.0
S3 -1.5	S4 -1.0	S5 -1.0
S6 -2.0	S7 -1.0	S8 +1.0

-0.75 -2.0 -0.25 -0.25

P52 = 행동 가치 함수 계산결과
P44 = 가치함수의 결과

Max_step = 0

-0.75

-0.75 -2.0 -0.25

-0.25

A1 = ¼ x (-3 +0.9 x -2) = -1.2
A2 = ¼ x (-1 +0.9 x -1.5) = -0.59
A3 = ¼ x (-1 +0.9 x -1.5) = -0.59
A4 = ¼ x (-3 +0.9 x -2) = -1,2

Max_step = 1

-1.2

-1.2 -2.0 -0.59

-0.59

■ 정책 → p53

- 어떤 상태에서 행동을 선택할 확률

■ 최적정책 → p53

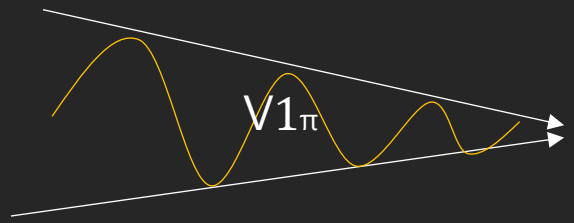
- 어떤 상태에서 최적의 행동을 알려주는 정책

■ 최적의 정책을 찾기 위해선?

- 여러가지 정책을 따르는 가치함수를 찾고 가치함수들로부터 **최적의 가치함수**를 찾으면 된다.

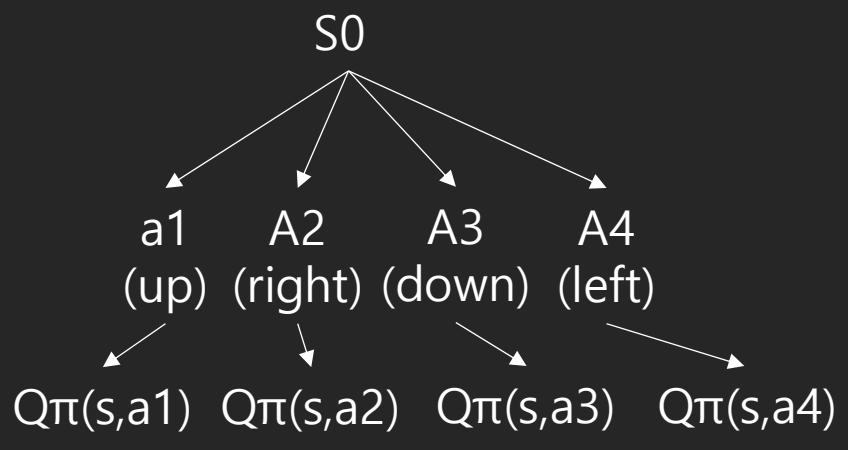
$V^*(S) = \max V_\pi(S)$
$Q^*(S) = \max Q_\pi(s,a)$

정책 π을 이용해 상태 가치함수를 **평가**하고
그 가치함수를 이용해 다시 정책을 **개선**하다보면
최적정책과 최적가치함수는 같은 곳으로 수렴한다.



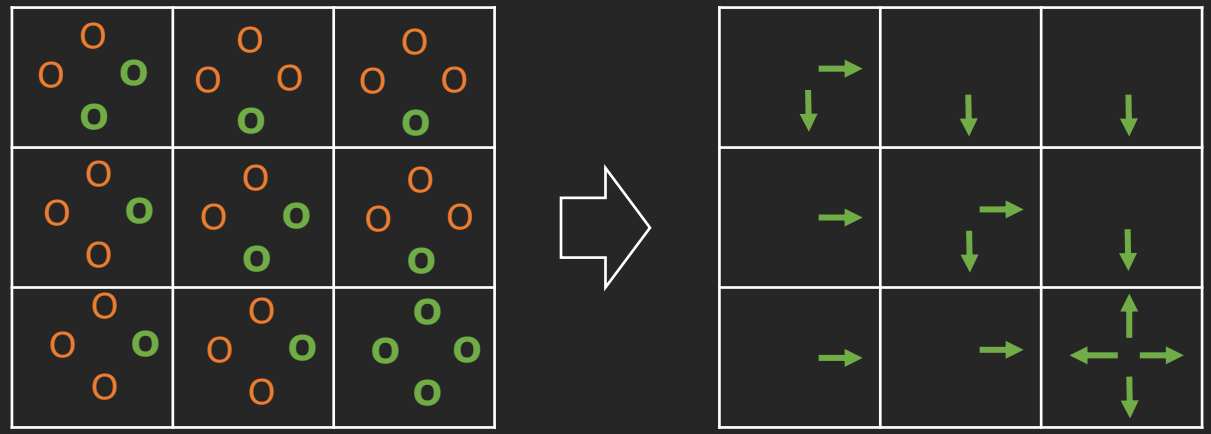
■ 정책 개선

"각 상태의 상태가치 $V_\pi(S)$를 이용해 새로운 정책 π을 개선하는 것"



$\pi^*(S) = \text{argmax}\ Q_\pi(s,a)$

새로운 정책



※ 강화학습 책 목차

1. 강화학습의 기본요서 10가지
"MDP"

2. MDP가 무엇인가?

" 순차적 행동결정 문제는 수학적으로 정의한 것"
상태, 행동, 보상, 상태변환확률, 감가율, 정책

"순차적 행동결정문제를 푸는 과정은 더 좋은 정책을 찾는 과정입니다."

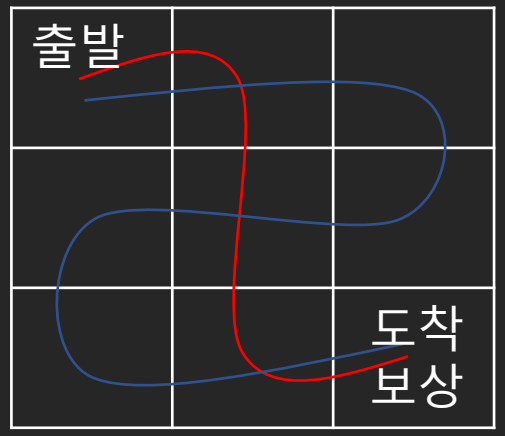
예: 환경, 상태, 행동, 보상(벽:-3, 미로내이동:-1, 도착:+1)
상태변환확률(p15) → 에이전트가 상태S에서 행동a를 취했을 때 S'으로 이동할 확률

P(S4|S1,a3) = 1
P(S3|s1,a3) = 0.1
P(S4|s1,a3) =
(결정론적 환경)

■ 감가율? 에이전트는 항상 현재에 판단을 내리기 때문에 현재에 가까운 보상일수록 더 큰 가치를 가지게하려고 감가율이 필요하다.

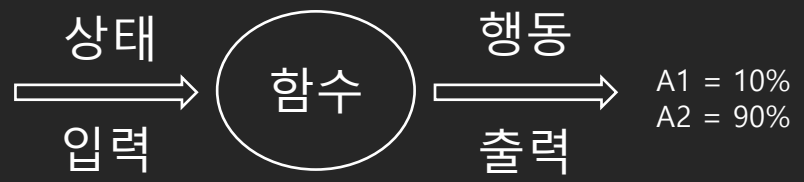
에이전트---보상　　수익--> 감가율을 곱한 보상들의 총합

----------보상

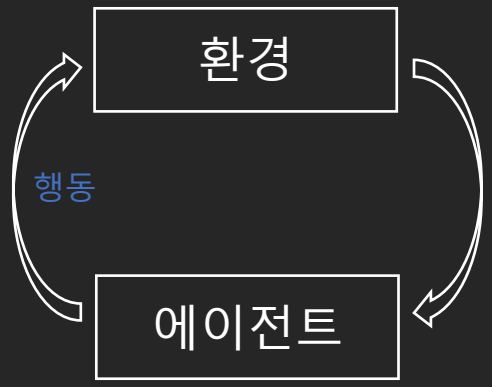


※ 감가율이 없다면 빨리 게임을 이기려고 하지않지만 "의지"를 안보인다.

■ 정책(π)? 모든 상태에서 에이전트가 할 행동



$\pi(a|S) = P(A_t=a|S_t=S)$



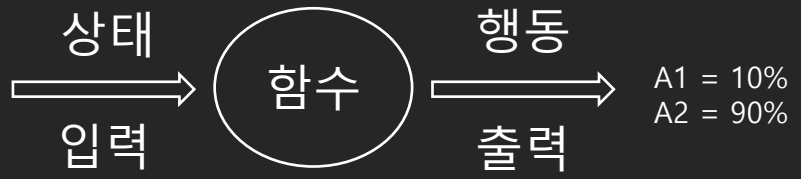
보상, 다음상태

이러한 과정을 반복하면서
에이전트는 앞으로 받을 것이라고
예상했던 보상(**가치함수**)에 대해
틀렸다는 것을 깨닫게 된다.

에이전트가 강화학습을 통해서 학습해야하는 것? "최적정책"

- 최적정책을 얻기 위해서는 현재의 정책보다 더 좋은 정책을 학습해 나가야한다. → 이게 "강화학습"이다.

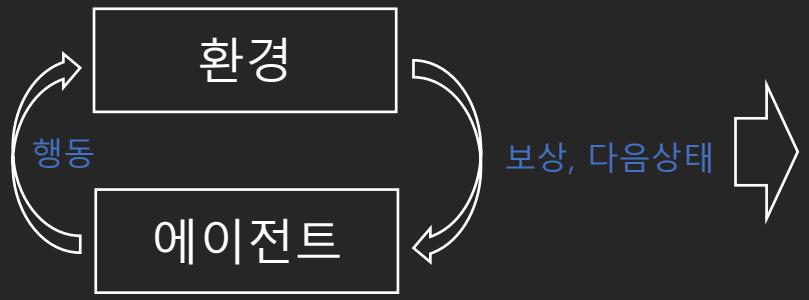
■ 정책(π)? 모든 상태에서 에이전트가 할 행동



$\pi(a|S) = P(A_t=a|S_t=S)$

에이전트가 강화학습을 통해서 학습해야하는 것? "최적정책"

- 최적정책을 얻기 위해서는 현재의 정책보다 더 좋은 정책을 학습해 나가야한다. → 이게 "강화학습"이다.



보상, 다음상태

이러한 과정을 반복하면서
에이전트는 앞으로 받을 것이라고
예상했던 보상(**가치함수**)에 대해
틀렸다는 것을 깨닫게 된다.

가치함수?

- 에이전트가 어떤 정책이 더 좋은 정책인지 판단하는 기준

↓

현재 상태로부터 정책을 따라갔을 때 받을 것이라 예상되는 보상의 합

$V_\pi(S) = E_\pi[R_{t+1} + \gamma \times V_\pi(S_{t+1})|S_t=S]$

기대 벨만방정식?

현재 상태의 가치함수와 다음 상태의 가치함수의 관계식
벨만기대방정식 → 특정 정책을 따라갔을 때 가치함수 사이에 관계식

$Q_\pi(S,a) = E_\pi[R_{t+1} + \gamma \times Q_\pi(S_{t+1}, A_{t+1})|S_t=S, A_t=a]$

행동가치함수
(큐함수)

$V_\pi(S) = \max E_\pi[R_{t+1} + \gamma \times V_\pi(S_{t+1})|S_t=S]$

= 벨만 최적 방정식

"최적의 가치함수를 받게 하는 정책"

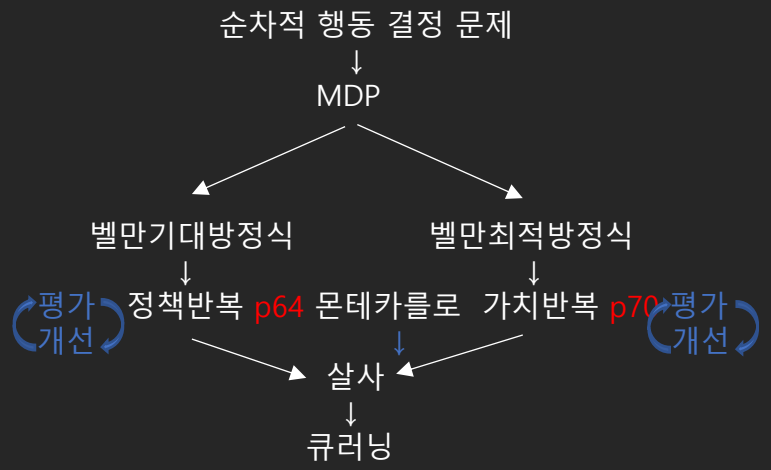
Max_step_number=0

-2.0	-1.5	-2.0
-1.5	-1.0	-1.0
-2.0	-1.0	-4.5 = -2.0-1.5-1-1+1 +1.0

Max_step_number=1

-3.58	-2.96	-3.46
-2.96	-2.12	-1.68
-3.46	-1.68	+1.0

■ 강화학습의 흐름



■ 몬테카를로 방법 → 상태전이확률을 " 경험 " 으로 대신한다.

- 모델기반알고리즘 → 환경에 대한 모든 정보를 알고 있는 상태에서 강화학습을 푸는 알고리즘
- 모델프리알고리즘 --> 상태전이확률을 몰라도 되는 알고리즘

$V\pi(S) = E\pi[R_{t+1} + \gamma \times V\pi(S_{t+1})|St=S]$

몬테카를로 방법은 탐색적인 방법을 이용해 "상태가치함수"와 " 행동가치함수 " 를 학습한다.

1. 결정론적 환경 = 1
2. 확률적 환경

상태전이확률을 지금까지 결정론적 환경으로 P(S'|s,a) = 1이라고 가정했었다.
P(S3|s1, a3) = 0.1, P(S4|s1,a3) = 0.8, P(s5|s1,a3) =0.1

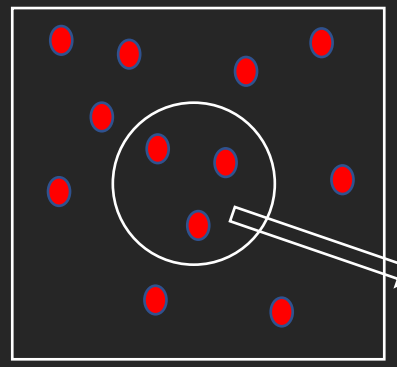
■ 몬테카를로 예측 → 원의 넓이를 구하는 방정식을 모른다면 원의 넓이는 어떻게 구할 수 있을까?

"몬테카를로 근사" 라는 방법을 이용하면 된다.

↓ 무작위로 무엇인가를 해본다.

↓ 원래의 값은 모르지만 샘플를 통해 원래의 값을 추정하는 것

"즉 무작위로 무엇인가를 해서 원의 넓이를 추정하는 것"



원의 넓이 / 정사각형의 넓이 = 원안에 들어간 점의 갯수 / 전체 점의 갯수

원안에 들어간 점을 → 샘플링 + 평균 --> 가치함수 추정

■ 몬테카를로 방법 → 상태전이확률을 " 경험 " 으로 대신한다.

- 모델기반알고리즘 → 환경에 대한 모든 정보를 알고 있는 상태에서 강화학습을 푸는 알고리즘
- 모델프리알고리즘 --> 상태전이확률을 몰라도 되는 알고리즘

$V\pi(S) = E\pi[R_{t+1} + \gamma \times V\pi(S_{t+1})|St=S]$

몬테카를로 방법은 탐색적인 방법을 이용해 "상태가치함수"와 " 행동가치함수 " 를 학습한다.

S0 -2.0	S1 -1.5	S2 -2.0
S3 -1.5	S4 -1.0 (0.3)	S5 -1.0
S6 -2.0	S7 -1.0	S8 +1.0

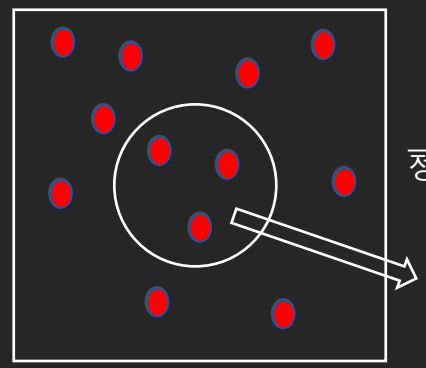
1. 결정론적 환경 = 1
2. 확률적 환경

상태전이확률을 지금까지 결정론적 환경으로 P(S'|s,a) = 1이라고 가정했었다.
P(S3|s1, a3) = 0.1, P(S4|s1,a3) = 0.8, P(s5|s1,a3) =0.1

■ 몬테카를로 예측 → 원의 넓이를 구하는 방정식을 모른다면 원의 넓이는 어떻게 구할 수 있을까?

"몬테카를로 근사" 라는 방법을 이용하면 된다.
↓ 무작위로 ↓ 원래의 값은 모르지만 샘플를 통해 원래의 값을 추정하는 것
무엇인가를 해본다.

"즉 무작위로 무엇인가를 해서 원의 넓이를 추정하는 것"



원의 넓이 / 정사각형의 넓이 = 원안에 들어간 점의 갯수 / 전체 점의 갯수
↓
샘플링 + 평균 --> 가치함수 추정
원안에 들어간 점을

V(S0) = average(G1, ..., Gn) :p78
↓
에피소드별로 얻은 수익 G

G1
S0 → A1 → 벽

Gn
S0 → A2 → S1 → A2 → S

예:
수익리스트 = [-10, -20, 25]

Average = $\frac{-10-20+15}{3}$ = -5

$Qn = \frac{G1 + G2 + G3 + Gn}{n} = \frac{\sum Gi}{n}$

= n번째 수 Gn까지의 평균

$Qn = \frac{G1 + G2 + G3 + Gn}{n} = \frac{1}{n}\sum_{i=1}^{n} Gi$

여기에 수익 Gn+1이 추가되면?

$Qn+1 = \frac{1}{n+1}(G_{n+1} + \sum_{i=1}^{n} Gi) = \frac{1}{n+1}(Gn_{+1} \sum_{i=1}^{n} Gi)$

$$Qn = \frac{G1 + G2 + G3 + Gn}{n} = \frac{1}{n}\sum_{i=1}^{n} Gi$$

여기에 수익 Gn+1이 추가되면?

$$Qn+1 = \frac{1}{n+1}\left(G_{n+1} + \sum_{i=1}^{n} Gi\right) = \frac{1}{n+1}\left(Gn_{+1} + n * \frac{1}{n}\sum_{i=1}^{n} Gi\right) = \frac{1}{n+1}\left(Gn_{+1} + n * Qn + Qn - Qn\right)$$

$$= \frac{1}{n+1}\left(G_{n+1} + (n+1) * Qn - Qn\right) = \frac{1}{n+1} Gn_{+1} + \frac{n+1}{n+1} * Qn - \frac{1}{n+1} * Qn$$

$$= Qn + \frac{1}{n+1} G_{n+1} - \frac{1}{n+1} * Qn$$

$$= Qn + \frac{1}{n+1}\left(G_{n+1} - Qn\right)$$

$$Qn+1 = Qn + \frac{1}{n+1}\left(G_{n+1} - Qn\right)$$

새로운 평균 이전평균 러닝레이트 새로운 데이터 이전평균

원래식 : V(S) ←-------average(G1,.....Gn)

변경된 식: $V(St) \leftarrow V(St) + \alpha[Gt - V(St)]$

현재까지 학습한 상태가치 / 러닝레이트 (스텝사이즈) / 보상의 총합 (수익) / 형재까지의 학습한 상태가치

현재까지 학습한 상태가치 V(St)가 목표수익 Gt와 같다면 더는 학습이 필요없다는 의미가 된다.
따라서 Gt-V(St) = 0이 되도록 V(St)를 학습한다.

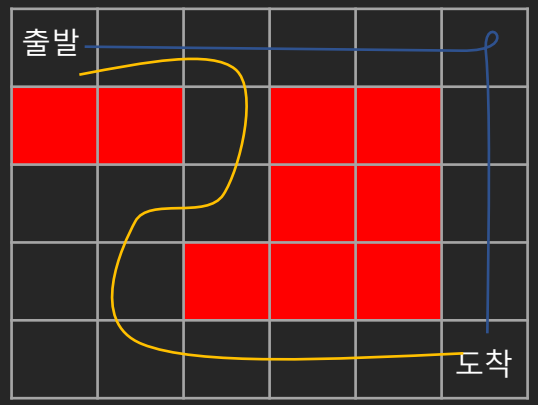
얼음판 몬테카를로 Ai VS 틱텍토 몬테카를로 Ai

1. V(St) ←-----V(St) + α[Gt – V(St)]

현재까지 학습한 상태가치

S0 -2.0	S1 -1.5	S2 -2.0
S3 -1.5	S4 -1.0	S5 -1.0
S6 -2.0	S7 -1.0	S8 +1.0

2. ε-greedy p95



(1) 경험에 의한 수 (greedy policy)→ 10번 중 9번
(2) 랜덤 수 ----------→ 10번 중 1번

ε-greedy
↓
입실론

(랜덤이 없으면)
국소화에 빠질 수 있다.
(local minima)

■ 몬테카를로 Tic Tac Toe AI

(1) 플레이 아웃을 구현

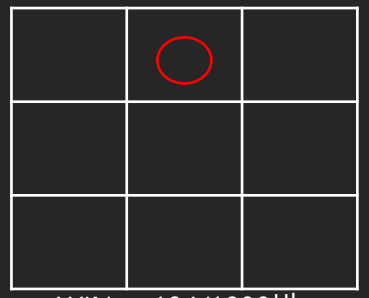
↓

현재 상태에서 게임을 끝까지 진행하는 것

제일 처음

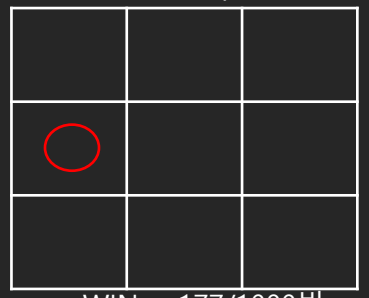
1000번

WIN = 357/1000번

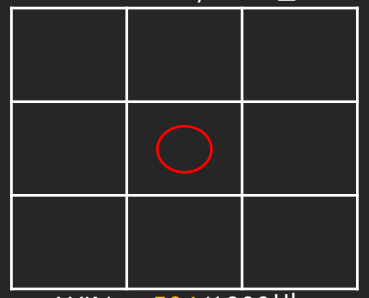


WIN = 194/1000번

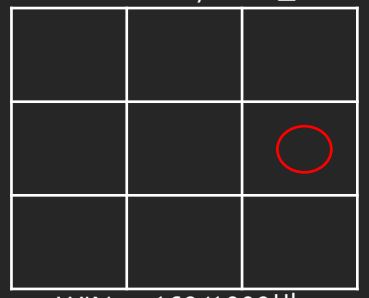
WIN = 312/1000번



WIN = 177/1000번

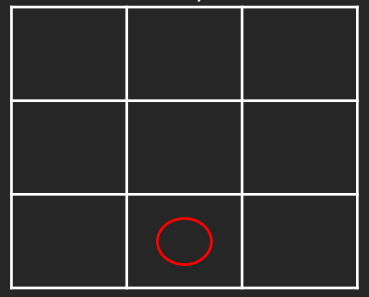


WIN = 504/1000번

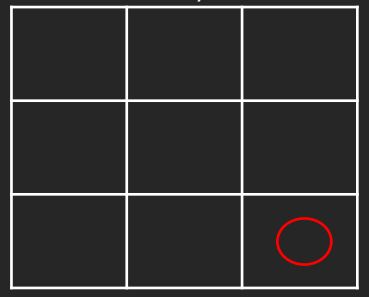


WIN = 168/1000번

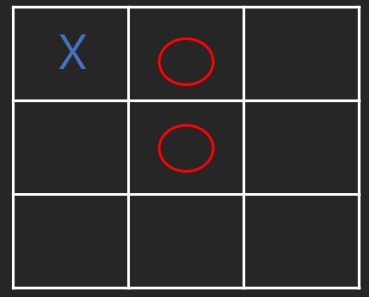
WIN = 377/1000번

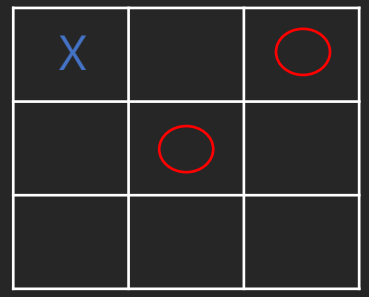


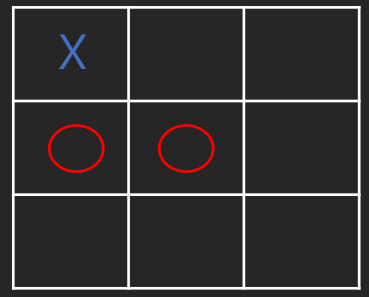
WIN = 188/1000번



WIN = 381/1000번







다시 → 수를 기록한다.

■ 틱텍토 게임의 구성 요소

1. 환경클래스
2. 인간 플레이어 클래스
3. AI 플레이어 클래스
    - 몬테카를로 p265
    - 살사
    - 큐러닝
    - DQN

■ 몬테카를로 클래스

1. Init 함수: num_playout = 1000 #플레이 아웃할 횟수 → 현재상태에서 끝까지 게임
2. select_action 함수: 현재 상태에서 어느 곳에 수를 둘지 수를 출력하는 함수
    1. 남은 수 중에 둘 수 있는 곳이 어딘지
    2. 플레이 아웃을 1000번 수행, 남은 수 자리에 승리한 횟수를 넣는다. (가치함수)
    3. 남은 자리 중에 가치가 가장 높은 곳에 수를 둔다.

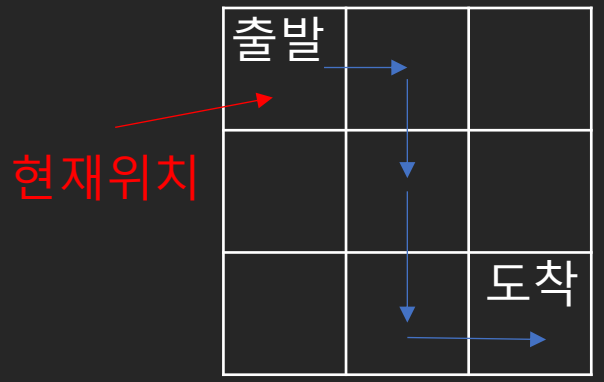
3. Playout 함수: 현재 상태에서 끝까지 게임을 해서 승리, 패, 비김이 될 때까지 게임을 진행하는 함수

문제2. 플레이 아웃횟수를 줄이면 몬테카를로 틱텍토 AI가 먼저 두는 선수여도 이길 수 있는지 확인하시오.

1000--> 10이하로 줄이면 이길 수 있다

```
p1 = Monte_Carlo_player()
p1.num_playout = 8
```

■ 순차적 행동결정문제



"결정을 순차적으로 내려야하는 문제"

↓

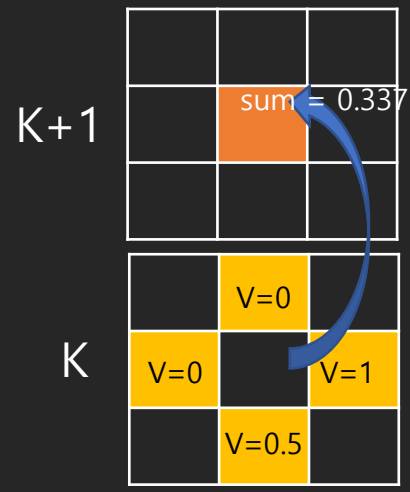
현재 위치에서 행동을 한번 선택하는게 아니라
계속적으로 선택해야 하는 문제

■ 벨만 기대 방정식

$$Vk+1 = \sum \pi(a|s)(R + \gamma Vk(s'))$$

행동 상태

k번째 가치함수를 통해 k+1번째 가치함수를 계산한다.



상: 0.25 x (0+0.9x0) = 0
하: 0.25 x (0+0.9x0.5) = 0.1125
좌: 0.25 x (0+0.9x0) = 0
우: 0.25 x (0+0.9x01) = 0.225
sum = 0.3375

■ MDP

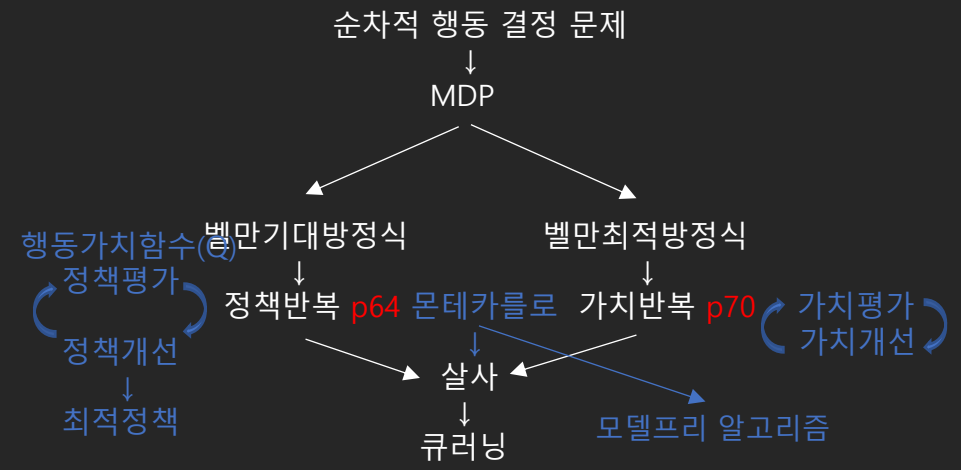
순차적 행동 결정문제를 수학적으로 풀기위해서 정의해야하는 구성 요소들

1. 상태 2. 행동 3. 보상 4. 정책 5. 감가율 6. 상태 전이 확률

벨만 기대 방정식:

$$Vk+1 = \sum \pi(a|s)(R + \gamma Vk(s'))$$

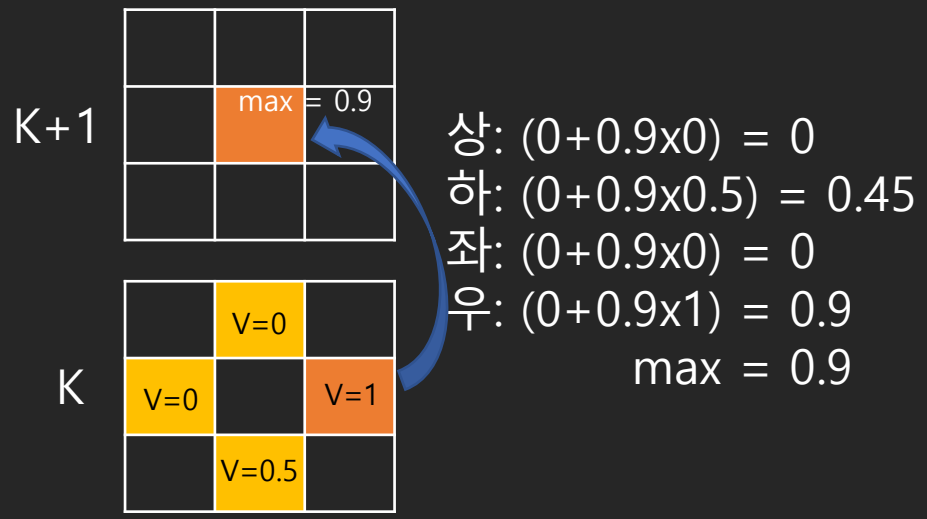
■ 강화학습의 흐름



■ 벨만 최대 방정식

$$Vk+1(S) = \max\ (R + \gamma Vk(s'))$$

벨만 기대방정식과 다르게 정책값 π(a|s)를 이용해 기대값을 계산하는 부분이 없고 MAX가 있다.



상: (0+0.9x0) = 0
하: (0+0.9x0.5) = 0.45
좌: (0+0.9x0) = 0
우: (0+0.9x1) = 0.9
max = 0.9

벨만기대방정식 → 정책 반복 → 살사
벨만최적방정식 → 가치 반복 → 큐러닝

■ 시간차 학습(p105)

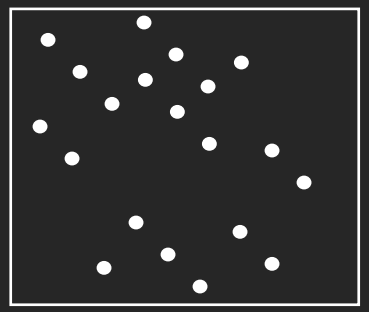
몬테카를로의 단점을 개선한 것이 시간차 학습입니다.
몬테카를로 알고리즘은 바둑의 경우처럼 에피소드의 길이가 길거나
주식시장처럼 에피소드의 끝이 없는 경우는 몬테카를로 예측이 적합하지 않습니다.

틱텍토 이긴 횟수를 가치함수로

S0 357	S1 194	S2 312
S3 179	S4 504	S5 168
S6 377	S7 188	S8 381

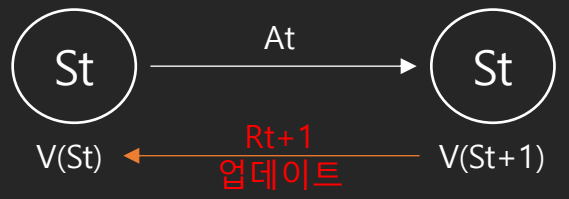
에피소드 길이가 짧다

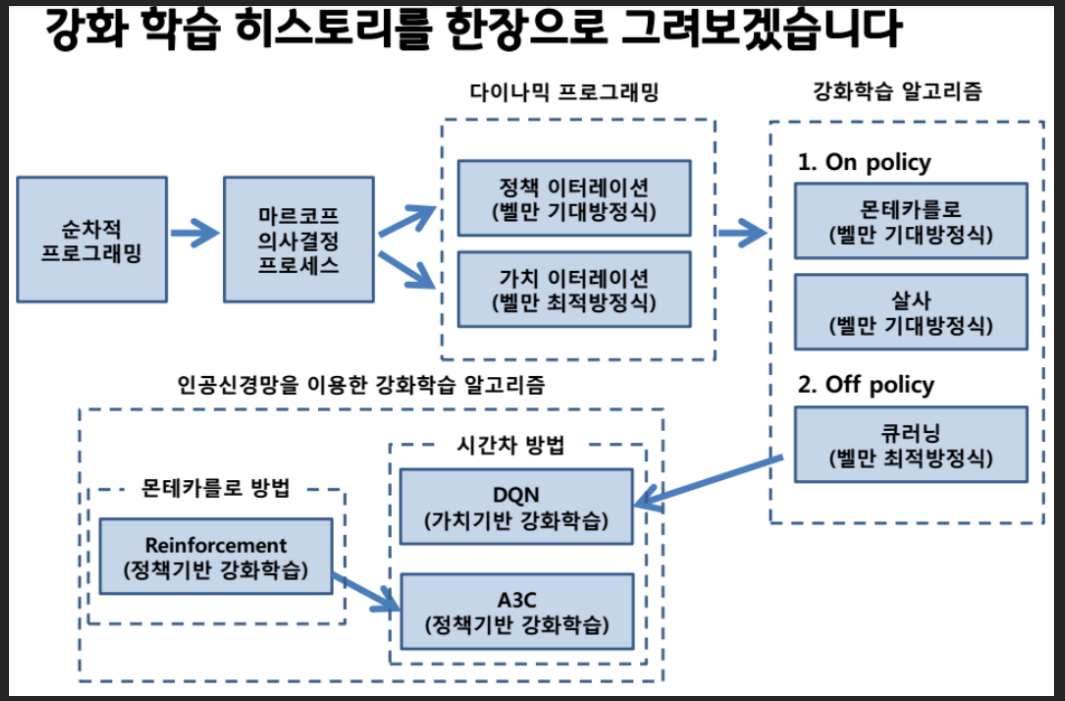
바둑



에피소드 길이가 길다

그래서 나온 것이 "시간차 학습"이다.





Off policy는 말 그대로 현재 행동하는 정책과는 독립적으로 학습한다는 것 입니다. 즉 행동하는 정책과 학습하는 정책을 따로 분리하는 것을 말합니다. Off policy를 적용한게 큐러닝인데 1989년 chris watkin라는 분에 의해서 소개가 되었습니다.

■ 큐러닝 (p115)

벨만 기대 방정식 → 정책 반복 → 살사
벨만 최대 방정식 → 가치 반복 → 큐러닝 (off policy)

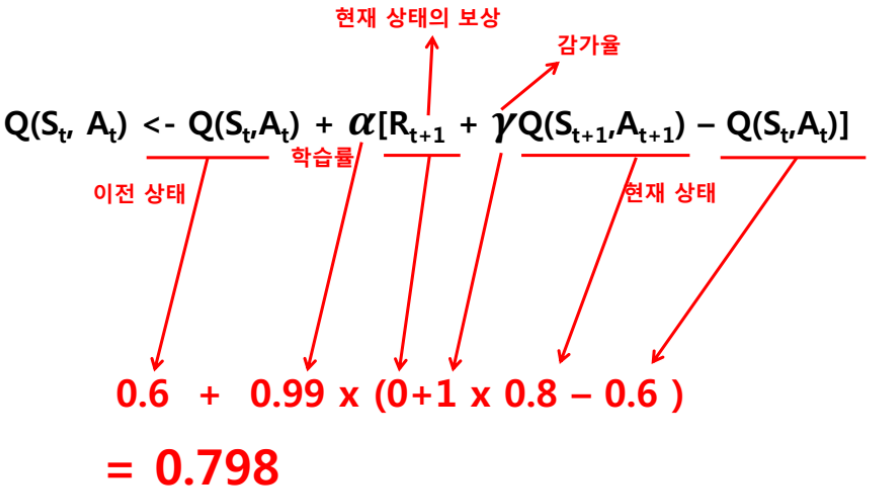
Off policy ? 현재 행동하는 정책과는 독립적으로 학습한다는 것
행동하는 정책과 학습하는 정책을 따로 분리

큐러닝.ppt

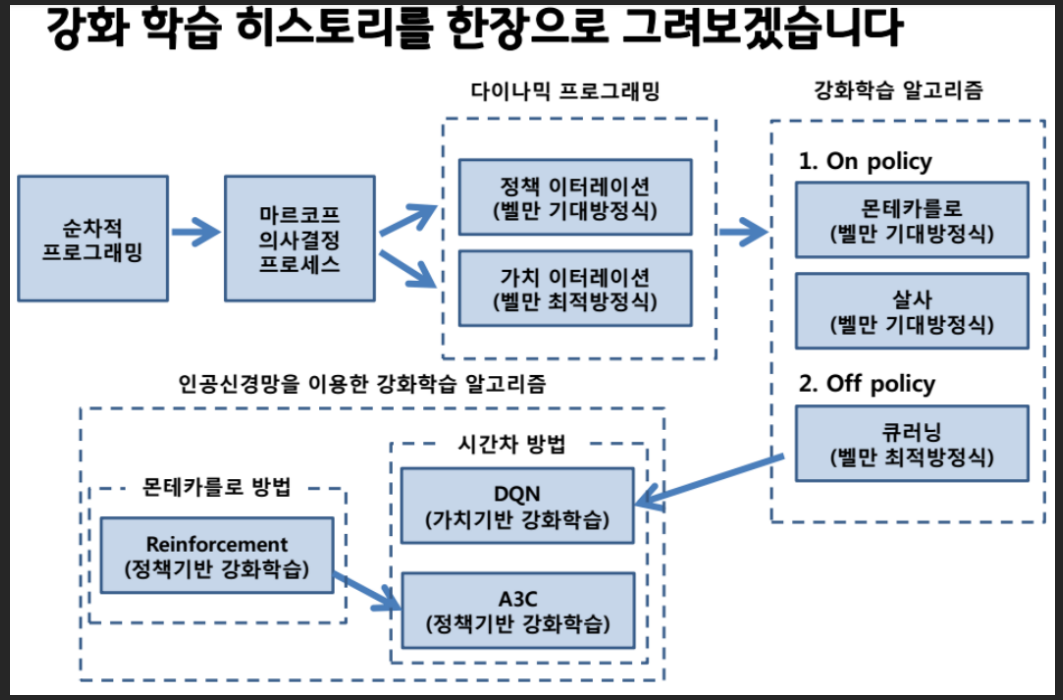
살사의 학습원리를 구현하는 수학식은 아래와 같습니다

$$Q(S_t, A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$$

잊어버리셨을까봐 다시 정리해 봅니다



강화 학습 히스토리를 한장으로 그려보겠습니다



살사는 행동한데로만 학습하지만

$$Q(S_t, A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$$

큐러닝은 행동한것보다 더 좋은수가 있었다면 그 수로 학습합니다

$$Q(S_t, A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \mathrm{MAX} Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)]$$

■ 오차함수 $= \frac{1}{2}(target\ value - V(S|W))^2$

$$\frac{\partial 오차함수}{\partial W} = (target\ value - V(S|W))x\frac{\partial V(S|W)}{\partial W}$$

$$Wnew \leftarrow Wold \leftarrow \alpha\, x \frac{\partial 오차함수}{\partial w}$$

$$Wt+1 \leftarrow Wt - \alpha(\text{target values} - V(S|W)) * \frac{\partial V(S|W)}{\partial w}$$

$$Wt+1 \leftarrow Wt - \alpha(\underbrace{r + \gamma * V(s')}_{\text{목표값}} - V(S|W) * \frac{\partial V(S|W)}{\partial w}$$

V(S|W) = W0 + W1*X1 + W2*X2

$\frac{\partial V(S|W)}{\partial W0} = 1$

$\frac{\partial V(S|W)}{\partial W1} = x1$

$\frac{\partial V(S|W)}{\partial W2} = x2$

W0 ← W0 - α(target values – V(S|W)) * 1

W1 ← W1 – α(target values – V(S|W)) * X1

W2 ← W2 – α(target values – V(S|W)) * X2

(P142)

Q1(s, a0|w0) = W00 + W01*X1 + W02*X2
Q2(s, a1|w1) = W10 + W11*X1 + W12*X2
Q3(s, a2|w2) = W20 + W21*X1 + W22*X2
Q4(s, a3|w3) = W30 + W31*X1 + W32*X2

(P145)

Q1(s, a0|w0) = -9.44 + 1.37*x1 + 0.37*X2
Q2(s, a1|w1) = 4.76 + 1.94*X1 + 0.78X2
Q3(s, a2|w2) = 4.68 + 0.74*X1 + 2.04*X2
Q4(s, a3|w3) = -7.18 + 4.9*X1 + 4.02*X2

■

*ping pong 게임을 하려면 필요할게 무엇인가?

1. 켄버스(canvas)
2. 공(ball) 클래스
   - init 함수 (공의 색깔과 크기, 캔버스에서의 위치)
   - 공을 움직이게 하는 함수
   - 패들에 부딪히면 공이 튀기게 하는 함수
3. 패들(paddle) 클래스
   - init 함수 (패들의 색깔과 크기, 캔버스에서의 위치)
   - 패들이 움직이게 하는 함수
     1. 패들이 캔버스 밖으로 안나가게 하는 코드
     2.
   - 왼쪽으로 움직이게 하는 함수
   - 오른쪽으로 움직이게 하는 함수

■ ping pong 게임을 머신러닝화 하는 방법

1. Ping pong 게임만 되게하는 클래스

   - canvas 클래스: 캔버스의 크기, 캔버스이 색깔
   - ball 클래스: - init 함수: 공 색깔, 공 크기, 공 위치
     - draw 함수: 공이 화면 밖으로 안 나가게하는 함수
     - hit_paddle 함수: 공이 패들에 닿으면 튀어오르게
   - paddle 클래스
     - init 함수: 패들의 크기, 위치, 키보드 바인딩
     - draw 함수: 패들이 화면 밖으로 안나가게 하는 함수
     - right_shift 함수: 패들을 오른쪽으로 움직이게 하는 함수
     - left_shift 함수: 패들을 왼쪽으로 움직이게 하는 함수

2. Ping pong 게임을 학습시키는 함수 (MDP의 구성요소들을 가지고 학습데이터를 만드는 함수들)
   1. keystat 함수: 학습 데이터의 모양을 만드는 함수
   2. add 함수: 학습 데이터를 values 딕셔너리에 추가하는 함수
   3. lookup 함수: 학습 데이터를 values 딕셔너리에서 찾아보고 못찾았으면 새로 add하는 함수
   4. greedy 함수: 현재 상황에서 가장 좋은 방향을 찾는 함수
   5. backup 함수: 학습 데이터의 가치함수의 수학식이 구현되어있는 함수
   6. action 함수: 랜덤 move와 greedy move 중에 하나를 결정하는 함수
   7. winnerval 함수: 가중치를 보상하는 함수 (1, -1)
   8. cyclestate 함수: 패들의 위치부터 학습 데이터를 모으겠금 해주는 함수
3. 부가적인 함수

3. 부가적인 함수
   1. gameover 함수: 볼이 한 사이클을 돌아서 hit 했는지 miss했는지를 알려주는 함수
   2. randomChoice 함수: 0 또는 1을 랜덤으로 선택하게 하는 함수
   3. wirtecsv 함수: 학습 데이터를 csv로 저장시키는 함수
   4. loadcsv 함수: 학습 데이터를 values 딕셔너리에 읽어들이는 함수

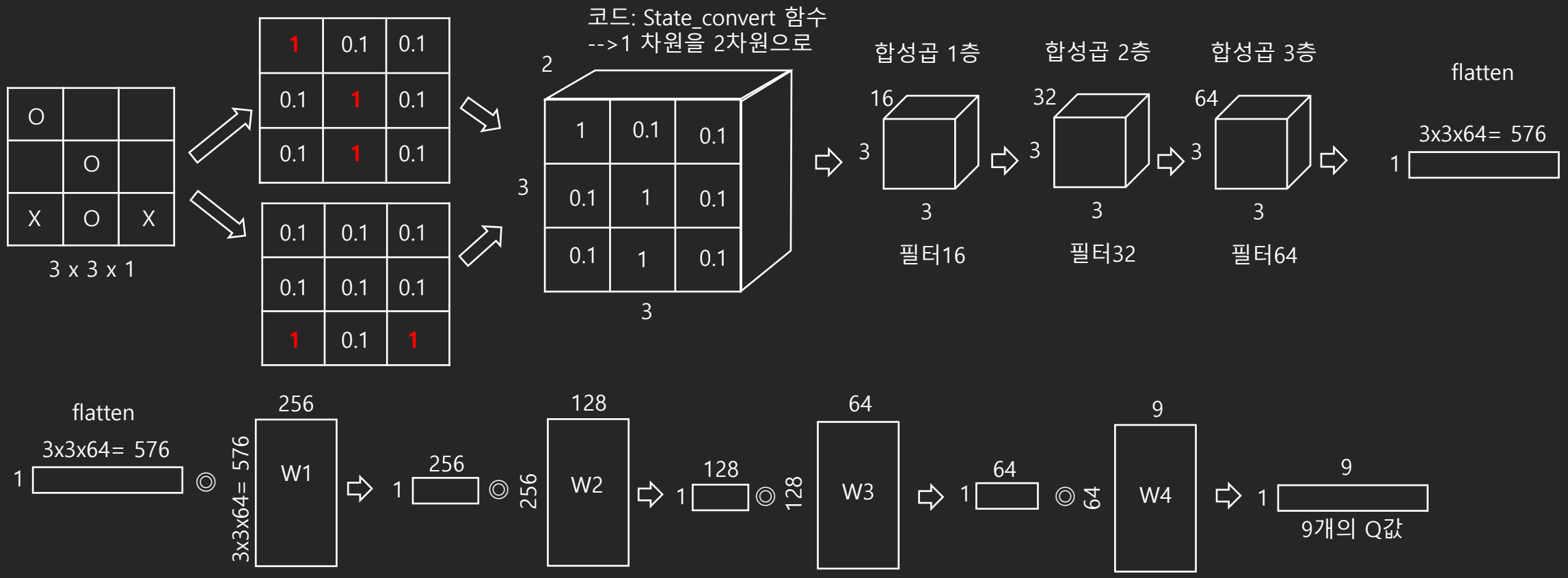
문제1. 캔버스를 그리시오.

```
from tkinter import *
import random
import time

tk = Tk()       # 1. tk 를 인스턴스화 한다.
tk.title("Ping Pong Game")  # 2. tk 객체의 title 메소드(함수)로 게임창에 제목을 부여한다.
tk.resizable(0, 0) # 3. 게임창의 크기는 가로나 세로로 변경될수 없다라고 말하는것이다.
tk.wm_attributes("-topmost", 1) #4. 다른 모든 창들 앞에 캔버스를 가진 창이 위치할것을 tkinter 에게 알려준
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
# bg=0,highlightthickness=0 은 캔버스 외곽에 둘러싼
# 외곽선이 없도록 하는것이다. (게임화면이 좀더 좋게)



canvas.configure(background='black')  # 캔버스 색깔 변경하는 방법
canvas.pack()       # 앞의 코드에서 전달된 폭과 높이는 매개변수에 따라 크기를 맞추라고 캔버스에에 말해준
tk.update()  # tkinter 에게 게임에서의 애니메이션을 위해 자신을 초기화하라고 알려주는것이다.
```

282P 메인신경망: (학습신경망) 입력값 → Q값 9개 출력

↓ 주기적으로 가중치를 복사 ⟹ Copy_network함수

타깃신경망: (목표값을 메인신경망이 제공) 입력값 → MaxQ값



Def select_action → policy

$\triangle = \alpha[R+\gamma \max Q(s',a') - Q(s,a)]$

역전파를 통한 가중치는 이 차이를 줄인다.

Learn_dQN → 1. 게임이 끝났을 때 학습 → 보상
(p283) 2. 게임 진행 중에 학습 → next Qvalue 값