

6 주 차 교 육

ToBig's 10기 박규리

# Advanced Neural Network

신경망 마스터 하기

# Contents

---

Unit 01 | Deep Learning Neural Network

---

Unit 02 | DNN problem & solution

---

Unit 03 | DNN Training Tip

---

Unit 04 | transfer learning & model save

---

## 1. Deep Learning Neural Network

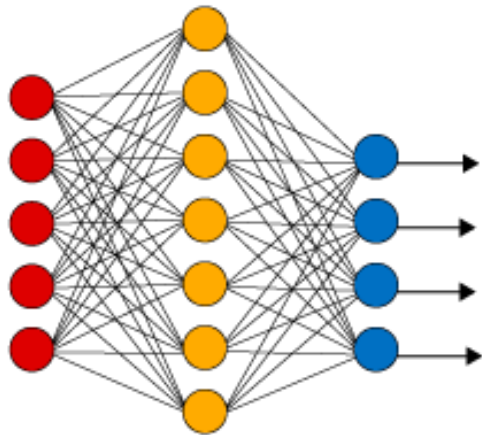
# 1. Deep Learning Neural Network

# 1. Deep Learning Neural Network

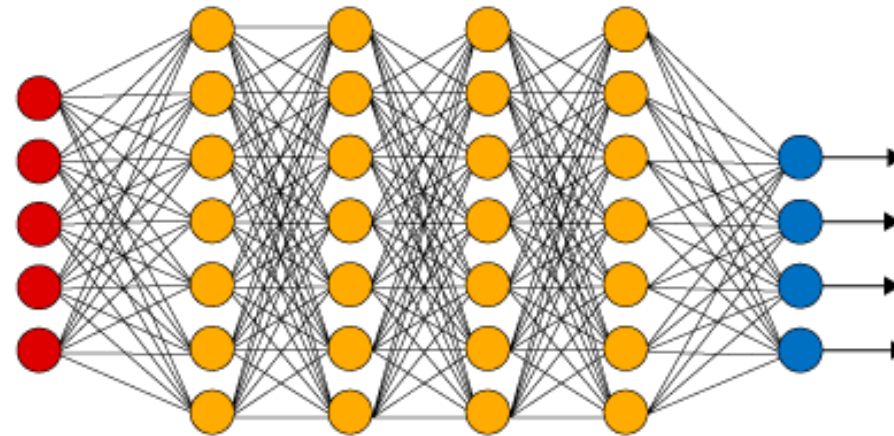
저번주에 Neural Network를 배웠다

대부분의 문제는 한 레이어를 쌓는 SNN(얇은 신경망)이 아닌 여러 레이어를 쌓는 DNN(깊은 심층 신경망)을 써야한다

Simple Neural Network



Deep Learning Neural Network



● Input Layer

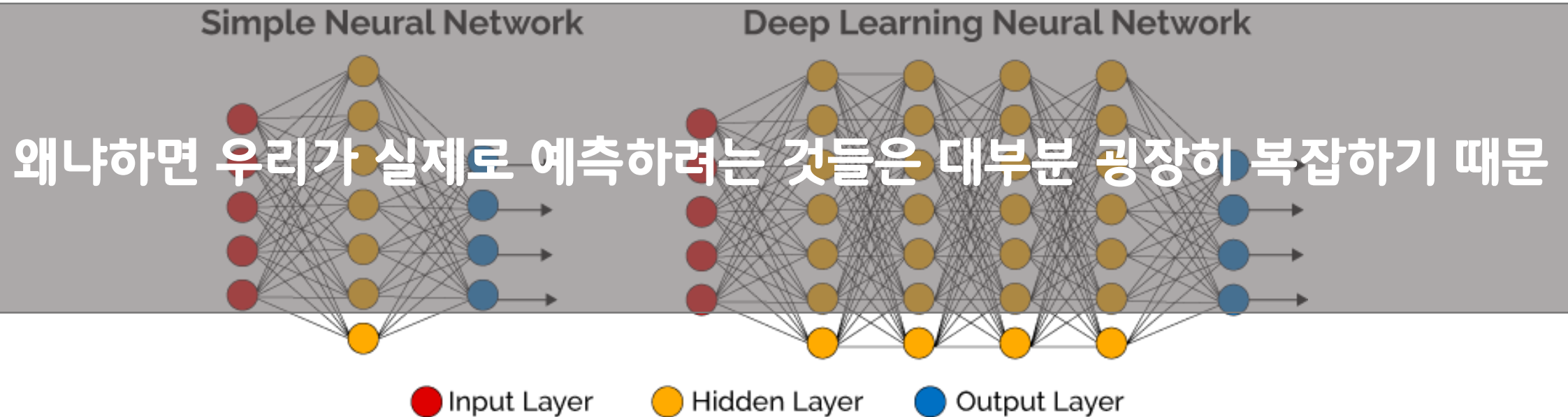
● Hidden Layer

● Output Layer

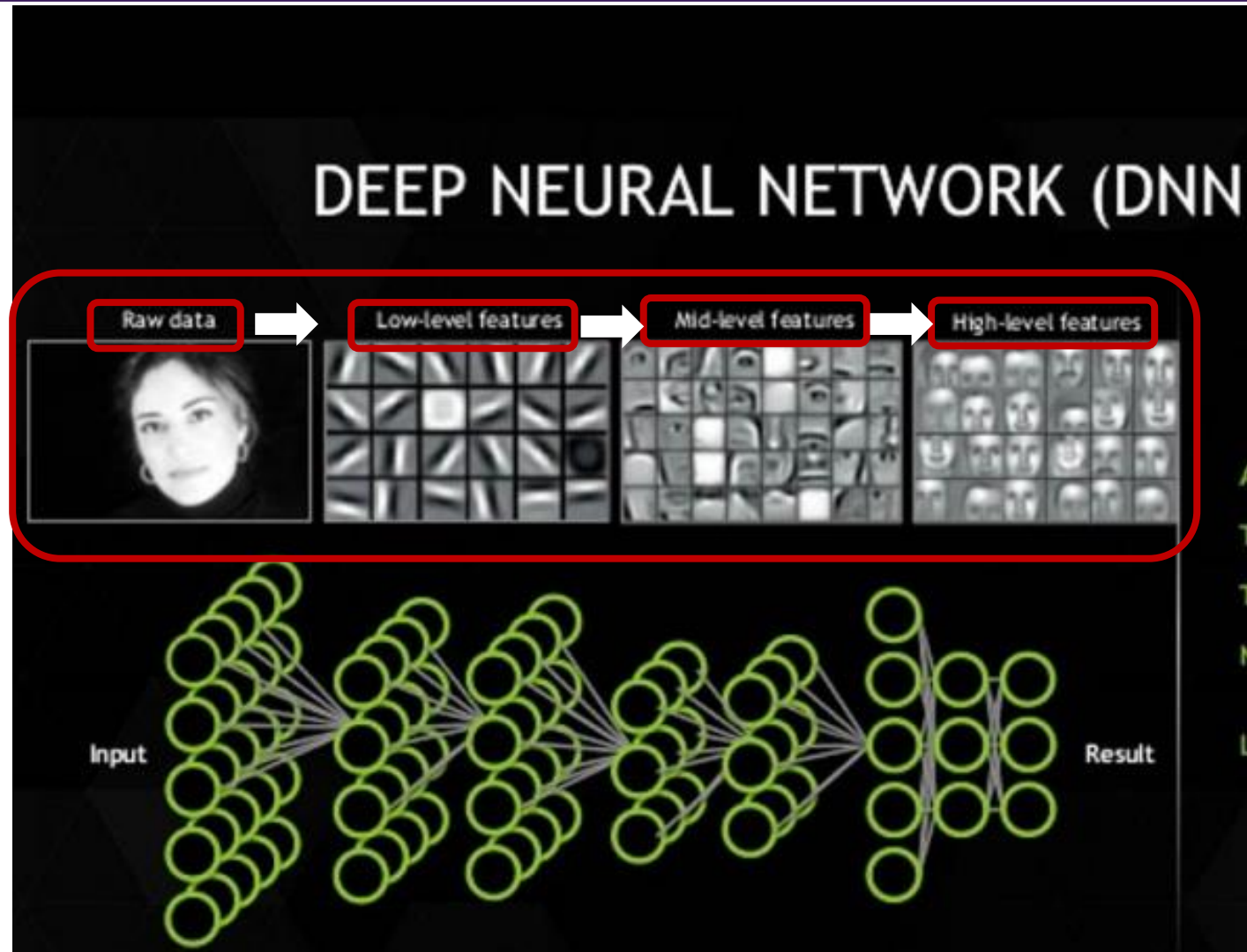
# 1. Deep Learning Neural Network

저번주에 Neural Network를 배웠다

대부분의 문제는 한 레이어를 쌓는 SNN이 아닌 여러 레이어를 쌓는 DNN(깊은 심층 신경망)을 써야한다



# 1. Deep Learning Neural Network



# 1. Deep Learning Neural Network

## DEEP NEURAL NETWORK (DNN)

Raw data

Low-level features

Mid-level features

High-level features

처음 레이어는 이미지의 edge(모서리)정도만 학습하다가 점점 더 복잡한 patten을 익히게 된다  
그러므로 복잡한 패턴인식이 필요한 경우 신경망의 레이어를 많이 쌓아야 학습을 잘 할 수 있다.

Input

Result

# 1. Deep Learning Neural Network

## DEEP NEURAL NETWORK (DNN)

Raw data

Low-level features

Mid-level features

High-level features

그렇다면 심층신경망(DNN)이 발생시킬 수 있는 문제는 무엇인가?

Input

Result



## 2. DNN problem & solution

## 2. DNN problem & solution

## 2. DNN problem & solution

심층신경망(DNN)이 발생시킬 수 있는 문제

1. 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제에 직면한다
2. 대규모 신경망에서는 훈련이 극단적으로 느려질 것이다
3. 수백만 개의 파라미터를 가진 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

## 2. DNN problem & solution

심층신경망(DNN)이 발생시킬 수 있는 문제

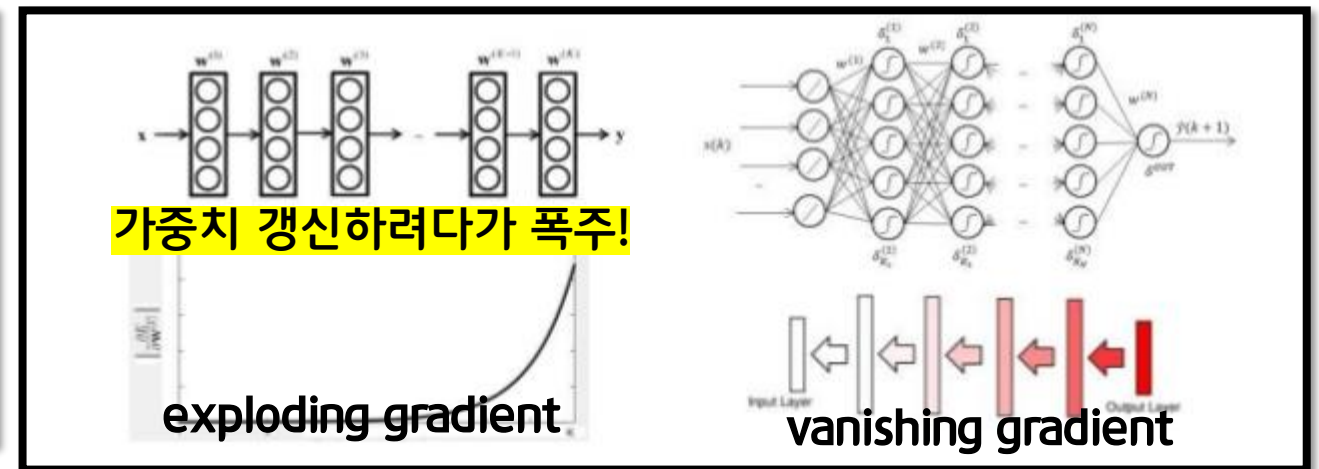
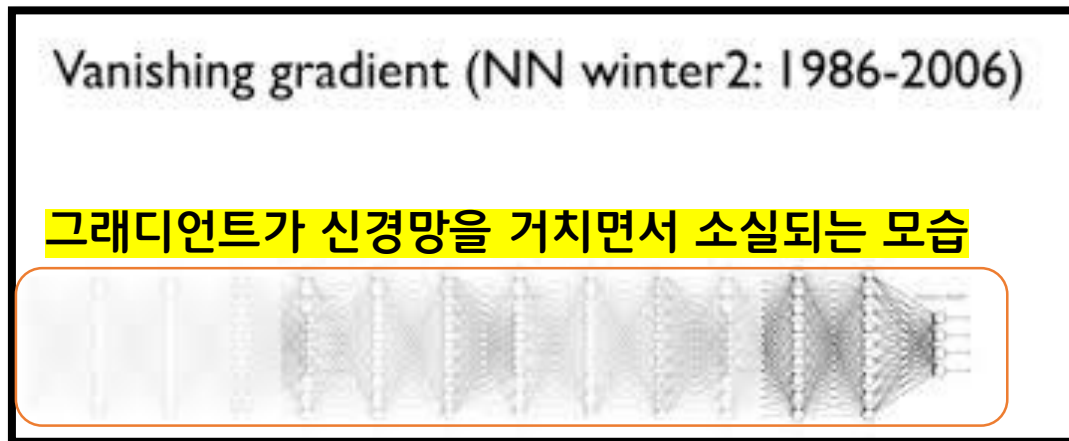
1. 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제에 직면한다  
해결책 → 세이버 초기화와 He 초기화, 수렴하지 않는 활성화 함수 사용, 배치 정규화
2. 대규모 신경망에서는 훈련이 극단적으로 느려질 것이다  
해결책 → 옵티마이저 사용(모멘텀 최적화, , 네스테로프 가속 경사, AdaGrad, RMSProp, Adam 최적화)
3. 수백만 개의 파라미터를 가진 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다  
해결책 → 조기 종료, L1&L2 규제, 드롭아웃, 데이터 증식

## 2. DNN problem & solution

### 1. 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭발(exploding gradient) 문제



역전파 알고리즘이 출력층에서 입력층으로 오차 그래디언트를 계산하면서 각 파라미터를 수정한다

이 과정에서 알고리즘이 **하위층으로 진행됨에 따라 그래디언트가 점점 작아져** 좋은 솔루션을 내지 못한다

어떤 경우에는 **그래디언트가 점점 커져 여러 개의 층이**

**비정상적으로 큰 가중치로 갱신**되면 그래디언트가 폭발(엄청 커진다)한다

이를 vanishing gradient 혹은 exploding gradient라고 하며, 이로써 심층 신경망은 불안정한 그래디언트에 영향을 받아 좋은 학습을 하지 못한다

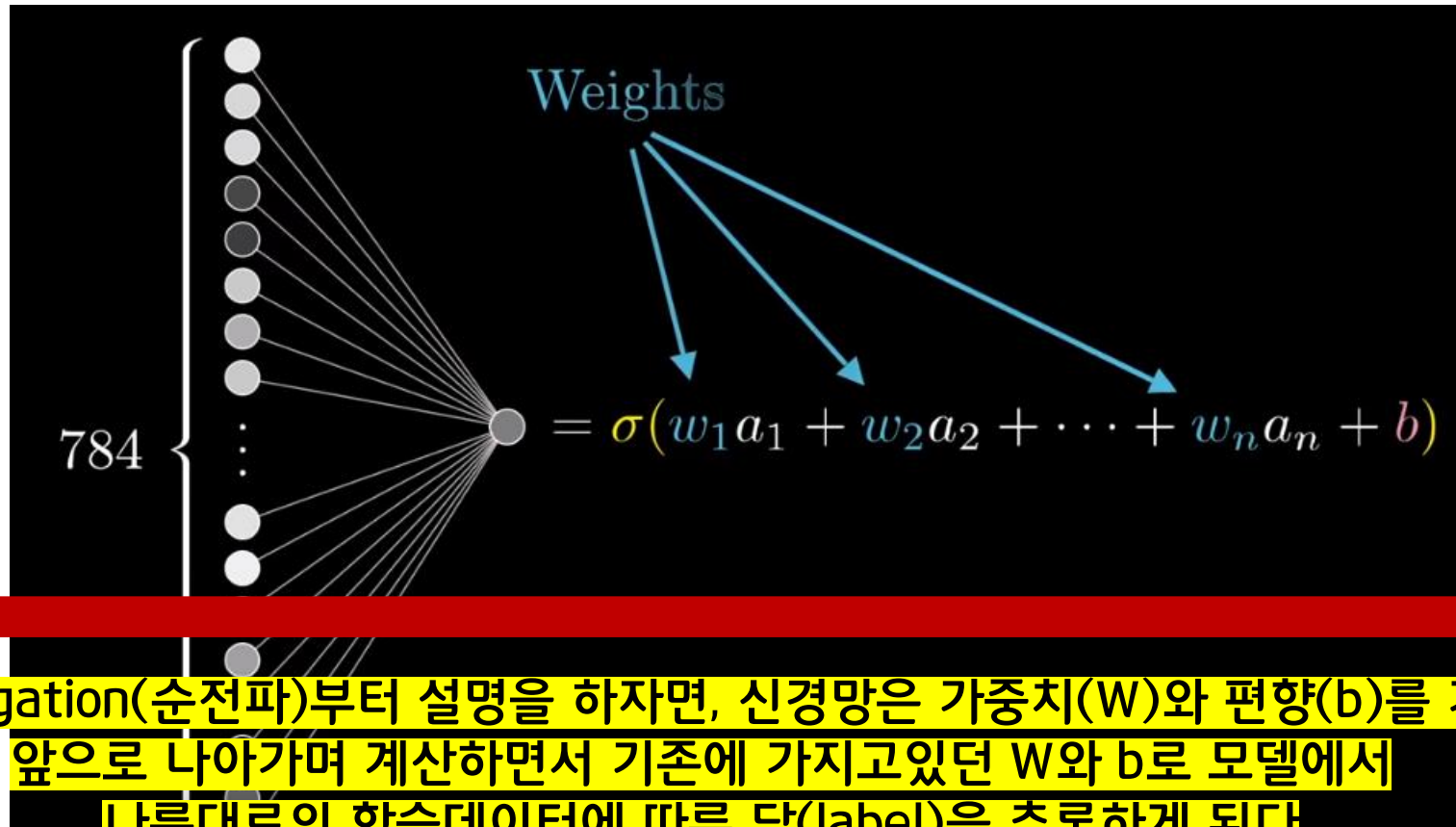
## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

backpropagation(역전파)란 무엇인가?

## 2. DNN problem & solution

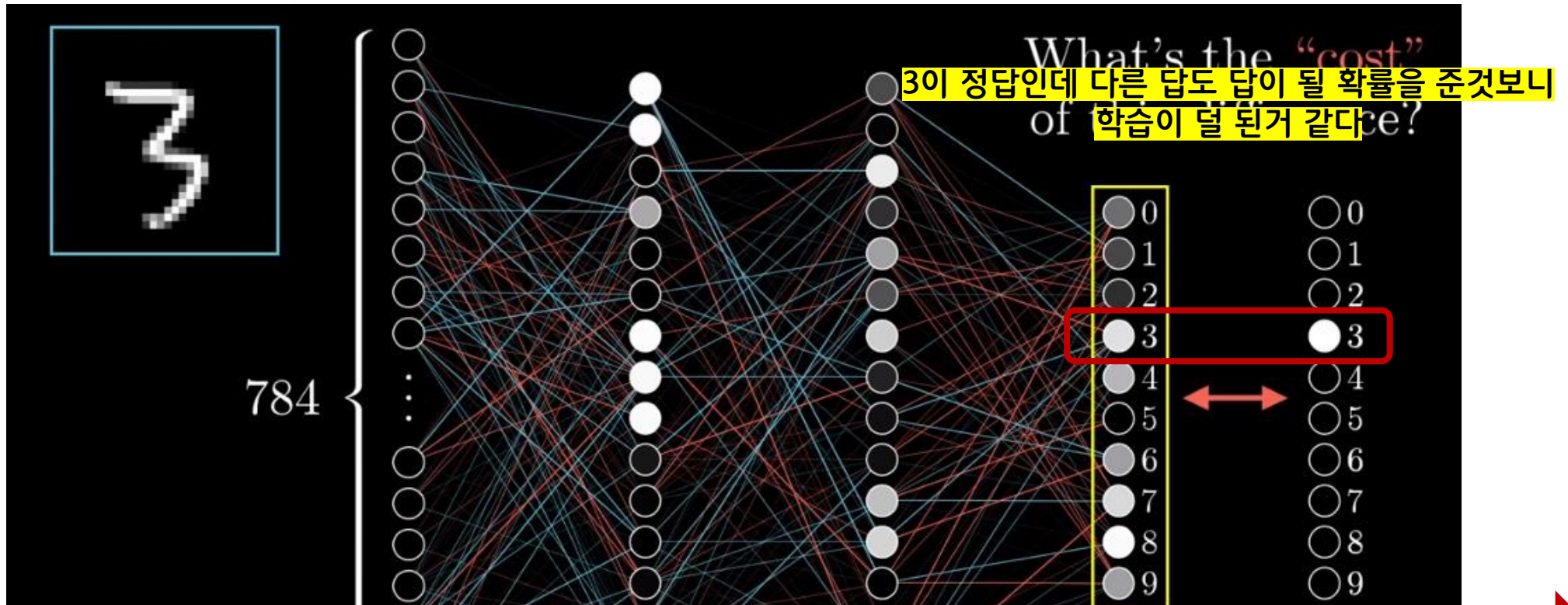
그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



forward propagation(순전파)부터 설명을 하자면, 신경망은 가중치(W)와 편향(b)를 가지고 있는데 앞으로 나아가며 계산하면서 기존에 가지고있던 W와 b로 모델에서 나름대로의 학습데이터에 따른 답(label)을 추론하게 된다

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

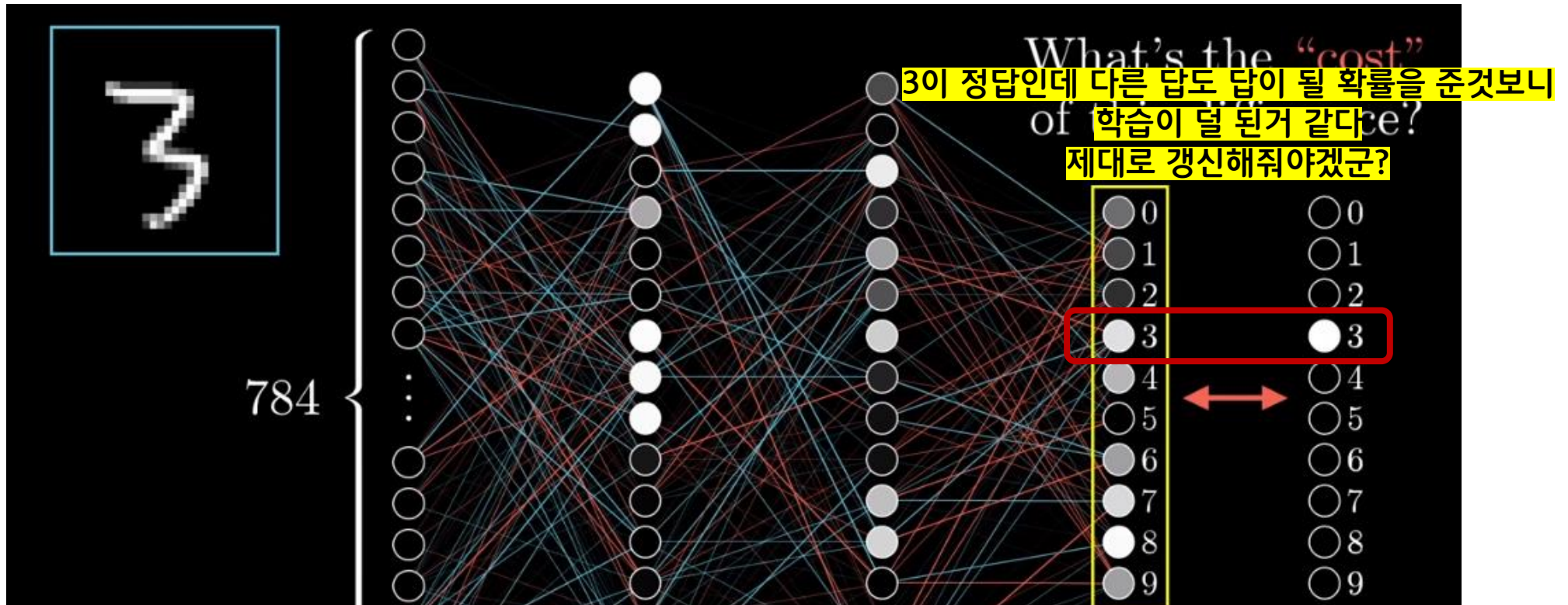


forward propagation(순전파)부터 설명을 하자면, 신경망은 가중치( $W$ )와 편향( $b$ )를 가지고 있는데 앞으로 나아가며 계산하면서 기존에 가지고있던  $W$ 와  $b$ 로 모델에서 나름대로의 학습데이터에 따른 답(label)을 추론하게 된다(label을 inference하는 단계)



## 2. DNN problem & solution

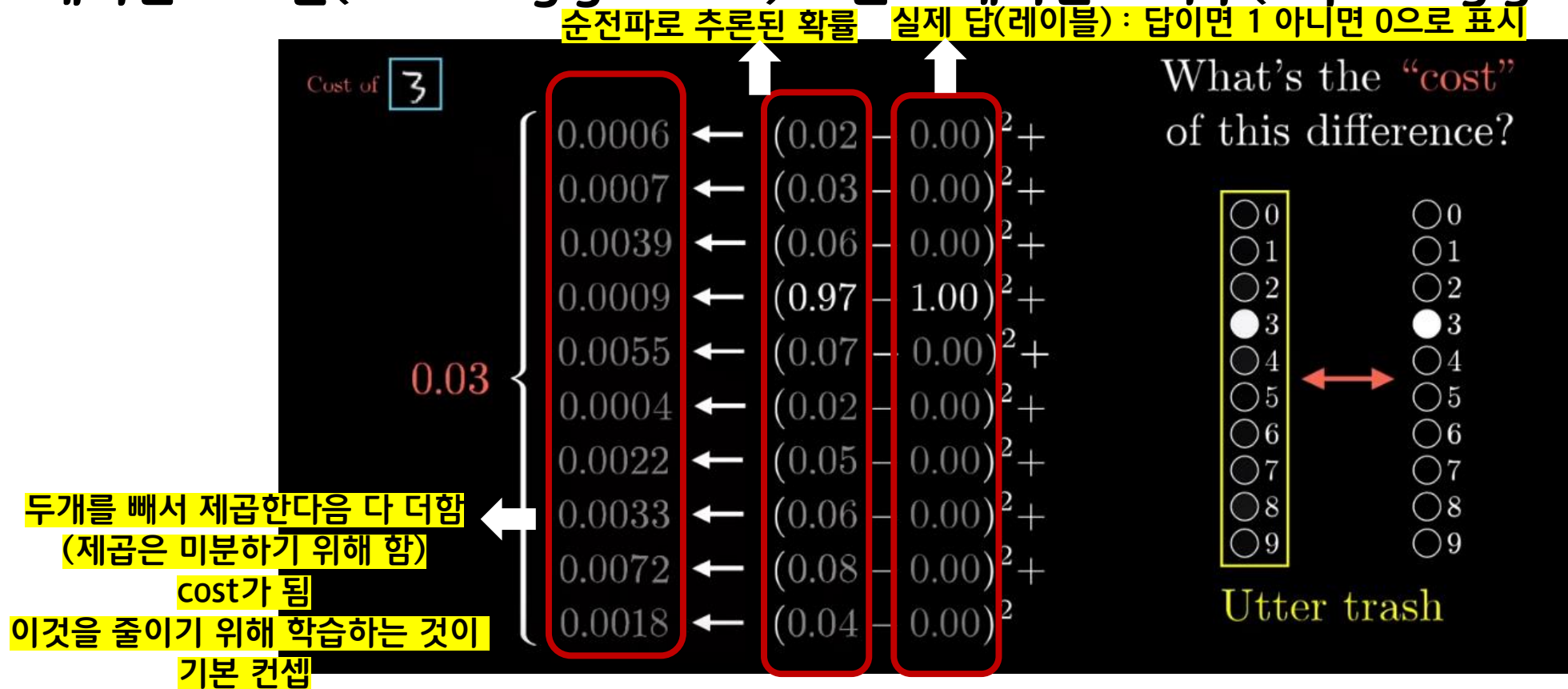
### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



backpropagation(역전파)는 순전파에서 나온 출력을 토대로 실제 답과 비교하여, 거기서 나온 오류(loss)를 토대로 가중치를 개선해 나가는 것이다 이것이 실제 모델의 training 단계이다

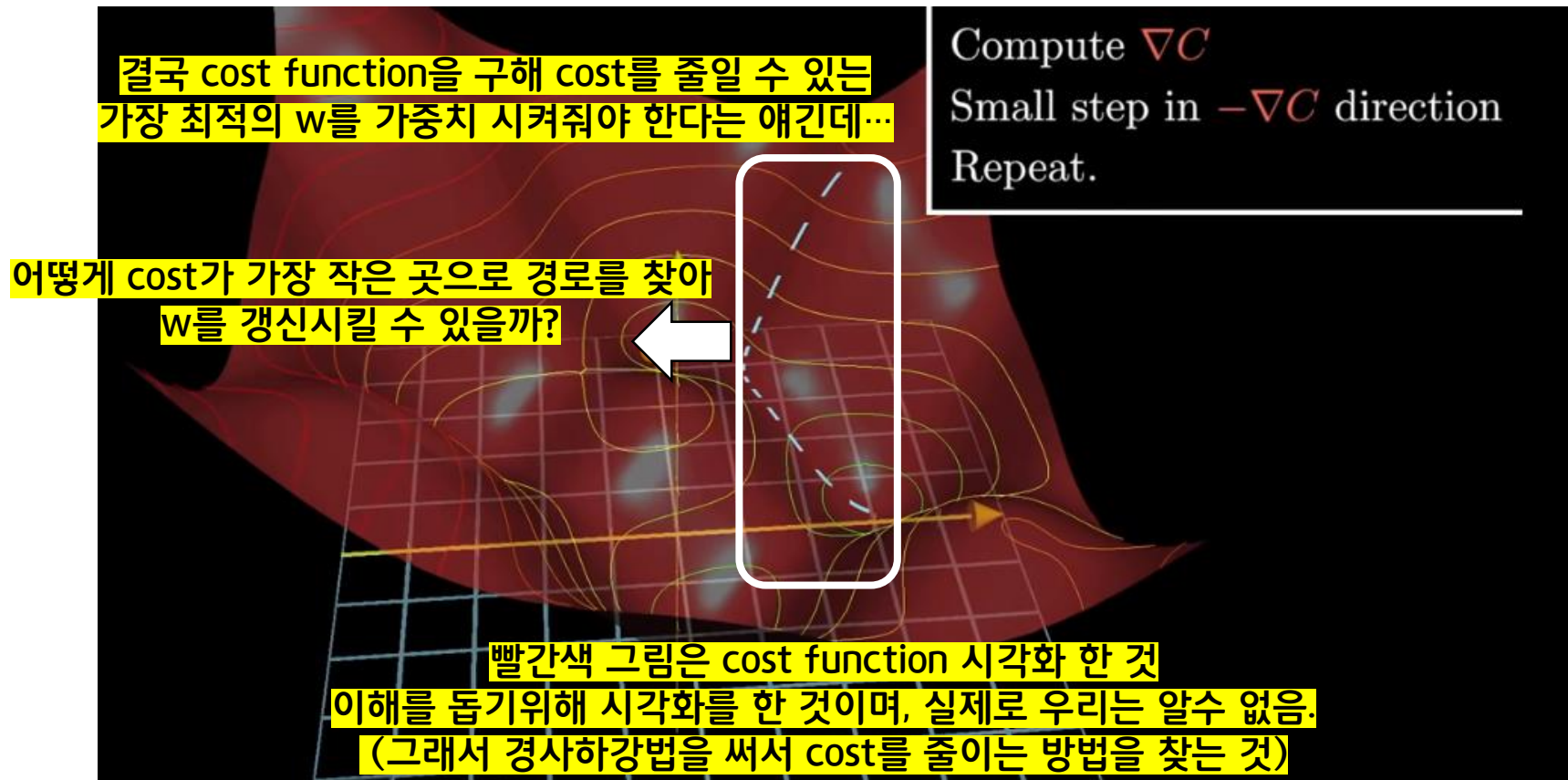
## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



## 2. DNN problem & solution

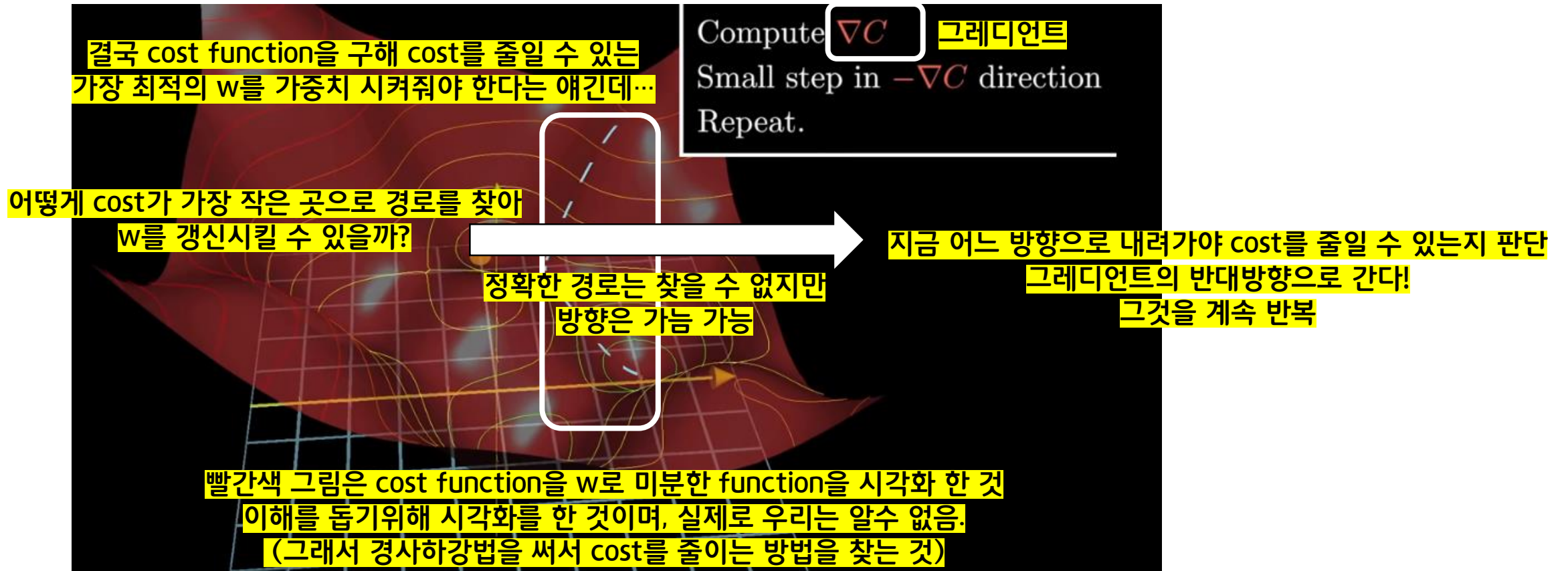
### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

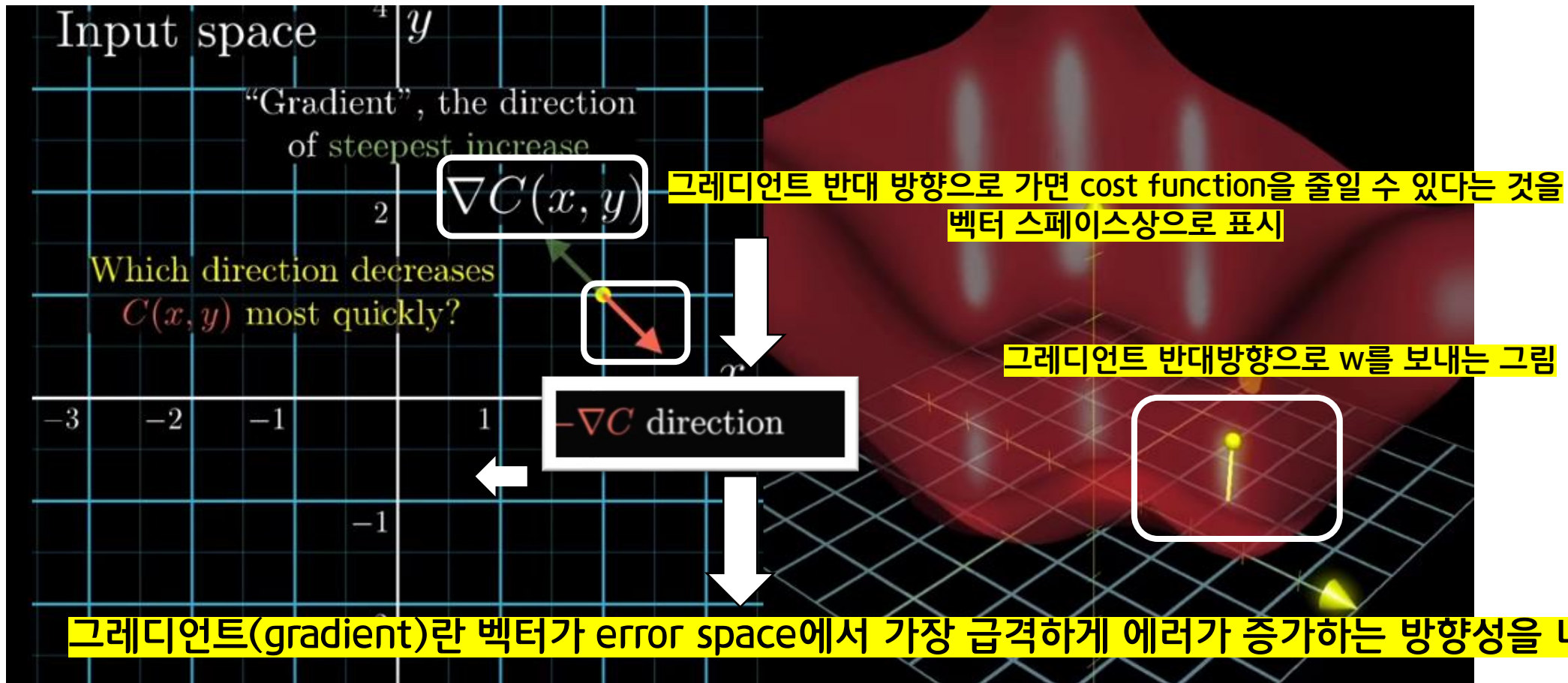
\*그래디언트 표기법(기호)은 사람마다 조금씩 다 다름... 그래서 기본 컨셉을 잘 이해하자





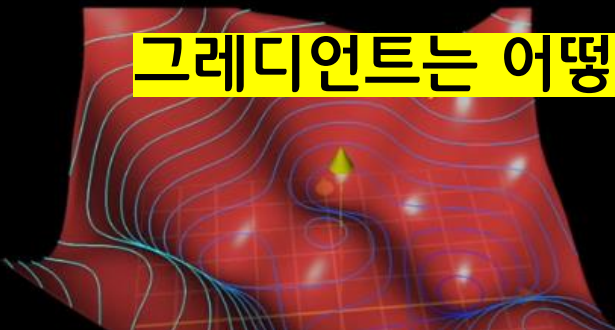
## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



$\nabla C$

그래디언트는 어떻게 구하나요?

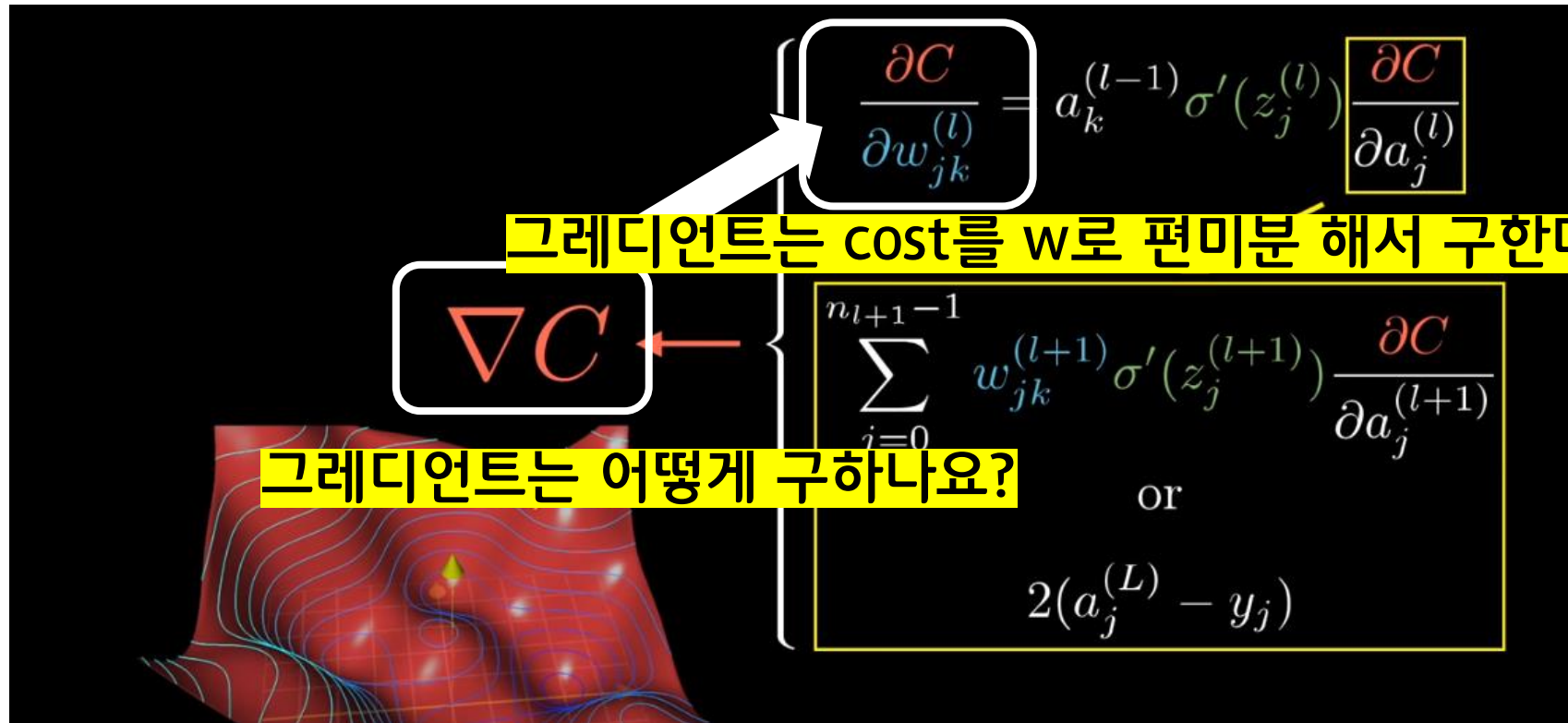
$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \boxed{\frac{\partial C}{\partial a_j^{(l)}}}$$
$$\sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{(l+1)}}$$

or

$$2(a_j^{(L)} - y_j)$$

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제



그래디언트는 cost를 w로 편미분 해서 구한다

그래디언트는 어떻게 구하나요?

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}$$

$$\nabla C \leftarrow \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{(l+1)}}$$

or

$$2(a_j^{(L)} - y_j)$$

## 2. DNN problem &amp; solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}$$

그래디언트는 cost를 w로 편미분 해서 구한다

$$\nabla C \leftarrow \sum_{j=0}^{n_{l+1}-1} w_{jk}^{(l+1)} \sigma'(z_j^{(l+1)}) \frac{\partial C}{\partial a_j^{(l+1)}}$$

그래디언트는 어떻게 구하나요?

or

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}$$

왜냐면 C를 W로 미분했다는 뜻은 W를 아주 조금 움직였을 때 C가 어떻게 변하느냐를 뜻하기 때문!

그러면 C를 W로 미분한 값 부호의 반대로 가면 C를 줄일 수 있다는 것!



## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

$$w^T = [w^{(1)}, w^{(2)}, \dots, w^{(m)}] \quad X = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix} \quad b = [b^{(1)}, b^{(2)}, \dots, b^{(m)}]$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(a, y)$$

$x_1, w_1, x_2, w_2, b$

$$z = [z^{(1)}, z^{(2)}, \dots, z^{(m)}] = w^T X + b$$

$$z = w_1 x_1 + w_2 x_2 + b$$

$$a = [a^{(1)}, a^{(2)}, \dots, a^{(m)}]$$

$$a = \sigma(z)$$

$$L = [L^{(1)}, L^{(2)}, \dots, L^{(m)}]$$

$$L = \mathcal{L}(a, y)$$

$$\frac{dL}{dz} = \frac{dL}{da} \frac{da}{dz} = (a - y)$$

$$\frac{da}{dz} = a(1 - a)$$

$$\frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

그림 말고 이제 수식을 봅시다..  
기호가 C가 L로 바꿨다...  
cost를 loss로 바꾼건데 비슷한 개념이니  
신경쓰지 말자..  
(loss function의 평균이 cost function)

$$\frac{dL}{dw} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dz}{dw} = \frac{dL}{dz} X$$

$$\frac{dL}{db} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dw}{db} = \frac{dL}{dz} 1$$

$$\frac{dJ(w, b)}{dw} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{dw}$$

$$\frac{dJ(w, b)}{db} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{db}$$

복잡해보이지만 천천히 뜯어봅시다

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
 생각같아서는 비용 L을 바로 w로 미분해버리고 싶다 그게 쉽자나..

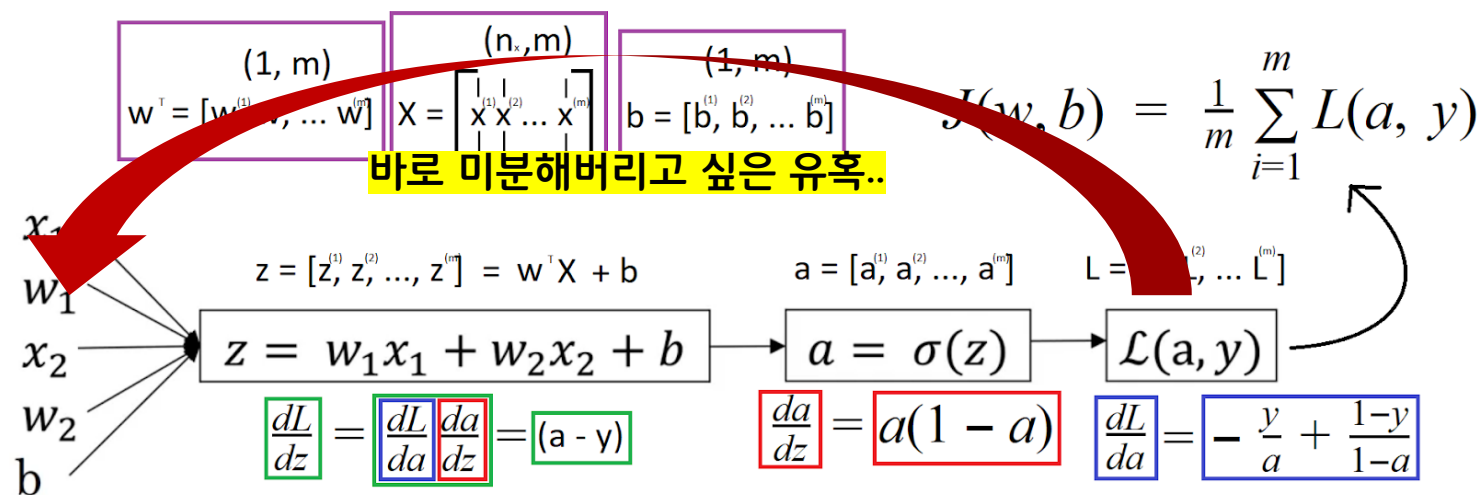


그림 말고 이제 수식을 봅시다..  
 기호가 C가 L로 바꿨다..  
 cost를 loss로 바꾼건데 비슷한 개념이니  
 신경쓰지 말자..  
 (loss function의 평균이 cost function)

$$\frac{d\mathcal{L}}{dw} = \frac{d\mathcal{L}}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{d\mathcal{L}}{dz} \frac{dz}{dw} = \frac{d\mathcal{L}}{dz} x$$

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{da} \frac{da}{dz} \frac{dz}{db} = \frac{d\mathcal{L}}{dz} \frac{dz}{db} = \frac{d\mathcal{L}}{dz} \cdot 1$$

$$\frac{dJ(w, b)}{dw} = \frac{1}{m} \sum_{i=1}^m \frac{d\mathcal{L}}{dw}$$

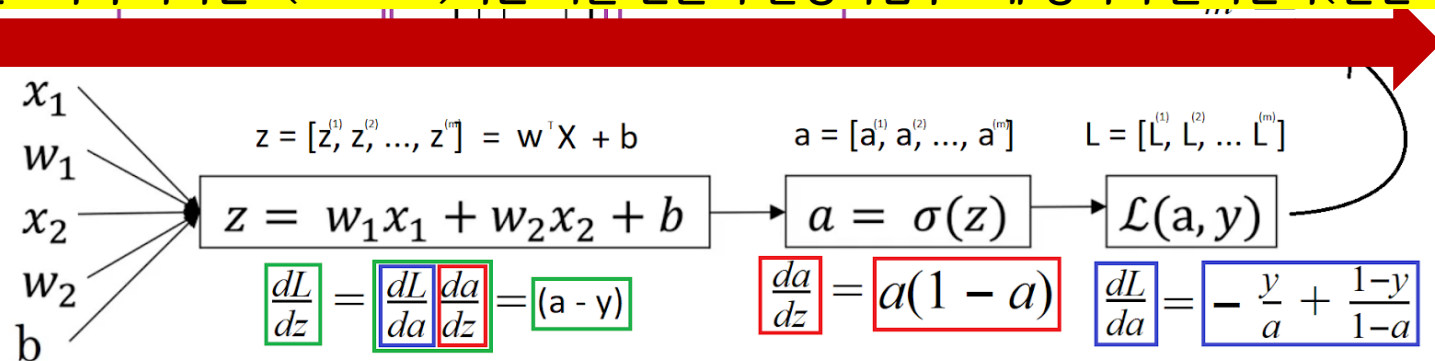
$$\frac{dJ(w, b)}{db} = \frac{1}{m} \sum_{i=1}^m \frac{d\mathcal{L}}{db}$$

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

근데 그렇게 하면 많이 탈난다고 하니 단계적으로 해보자

신경망은  $x$ 가 주어지면  $z(wx+b\dots)$ 라는 식을 만들어 활성화함수  $a$ 에 넣어서 출력한다(일단 레이어 하나만 생각하자)



$$\frac{dL}{dw} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dz}{dw} = \frac{dL}{dz} x$$

$$\frac{dL}{db} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dw}{db} = \frac{dL}{dz} 1$$

$$\frac{dJ(w, b)}{dw} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{dw}$$

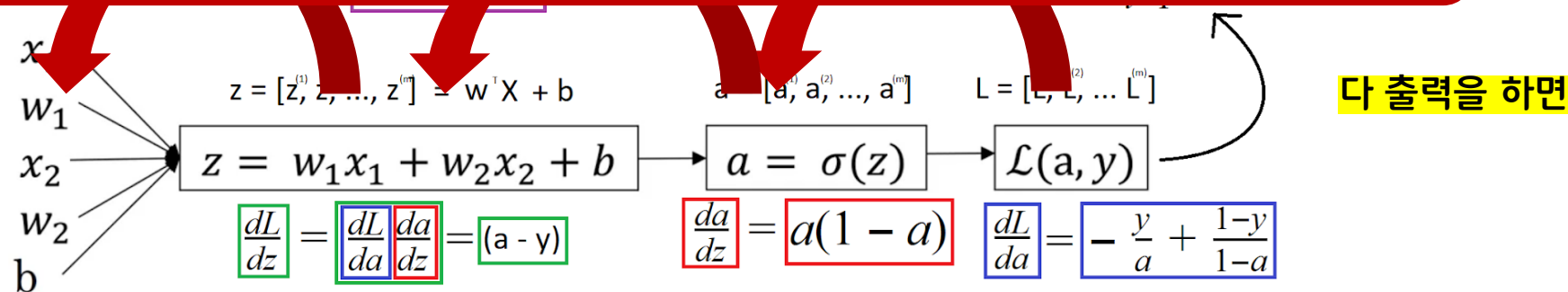
$$\frac{dJ(w, b)}{db} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{db}$$

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

근데 그렇게 하면 많이 탈난다고 하니 단계적으로 해보자

z를 w로 미분해서, 활성화 함수 a를 z로 미분하고, 출력과 레이블의 비교로 계산한 L을 활성화 함수 a로 미분하고



$$\frac{dL}{dw} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dz}{dw} = \frac{dL}{dz} [x]$$

$$\frac{dL}{db} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dw}{db} = \frac{dL}{dz} [1]$$

$$\frac{dJ(w, b)}{dw} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{dw}$$

$$\frac{dJ(w, b)}{db} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{db}$$

\*\*\*직관적으로 왜 그런지 생각하면  
w는 z에 영향을 주고 z는 활성화 함수 a에 영향을 주고 활성화 함수 a는 loss에 영향을 주니까 3개 각각 미분한 걸 곱하면 어떻게 L이 w에 미분되는지 알 수 있다.

**\*\*활성화 함수 a를 미분하는 과정이 있다는 것을 기억하자!**

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

근데 그렇게 하면 많이 탈난다고 하니 단계적으로 해보자

z를 w로 미분해서, 활성화 함수 a를 z로 미분하고

출력과 레이블의 비교로 계산한 L을 활성화 함수 a로 미분하고

그렇게 미분한 것 다 곱함 그러면 L을 w로 미분한 값이랑 동일

다 출력을 하면

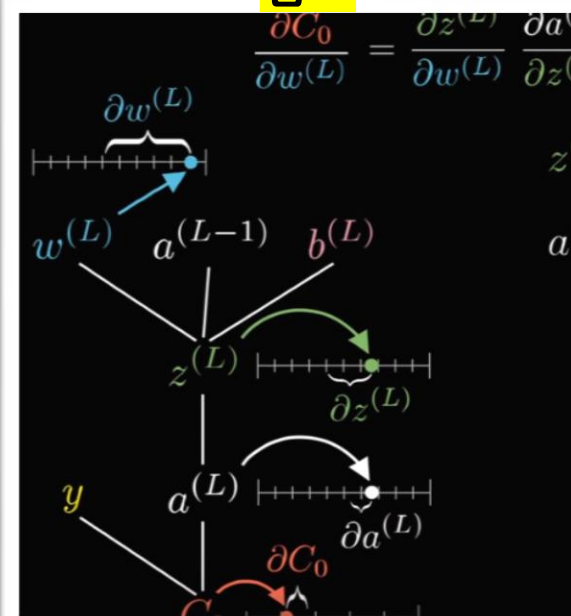
$$w = [w_1, w_2, \dots, w_m] \quad x = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix} \quad b = [b^{(1)} & b^{(2)} & \dots & b^{(m)}]$$

$$z = [z^{(1)} & z^{(2)} & \dots & z^{(m)}] = w^T x + b$$

$$a = [a^{(1)} & a^{(2)} & \dots & a^{(m)}] \quad L = [L^{(1)} & L^{(2)} & \dots & L^{(m)}]$$

$$a = \sigma(z) \quad \mathcal{L}(a, y)$$

$$\frac{dL}{da} = \frac{dL}{da} \frac{da}{dz} = (a - y) \quad \frac{da}{dz} = a(1 - a) \quad \frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$



$$\frac{dL}{dw} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dz}{dw} = \frac{dL}{dz} x$$

$$\frac{dL}{db} = \frac{dL}{da} \frac{da}{dz} \frac{dz}{dw} = \frac{dL}{dz} \frac{dw}{db} = \frac{dL}{dz} 1$$

$$\frac{dJ(w, b)}{dw} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{dw}$$

$$\frac{dJ(w, b)}{db} = \frac{1}{m} \sum_{i=1}^m \frac{dL}{db}$$

\*\*\*직관적으로 왜 그런지 생각하면

w는 z에 영향을 주고 z는 활성화 함수 a에 영향을 주고 활성화 함수 a는 loss에 영향을 주니까 3개 각각 미분한 걸 곱하면 어떻게 L이 w에 미분되는지 알 수 있다.

**\*\*활성화 함수 a를 미분하는 과정이 있다는 것을 기억하자!**

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

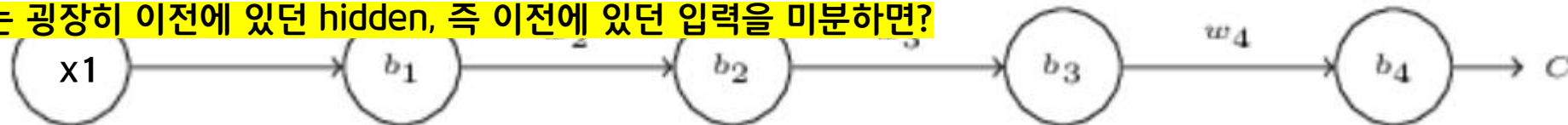
레이어가 많아지면 어떻게 될까?

$$\sigma'(z_1)$$

레이어가 많아지면 이전 w(weight)에 활성화함수의 미분을 계속 곱해줘야 함

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

C에 b1이라는 굉장히 이전에 있던 hidden, 즉 이전에 있던 입력을 미분하면?



b는 다 h(hidden)라고 생각해주세요

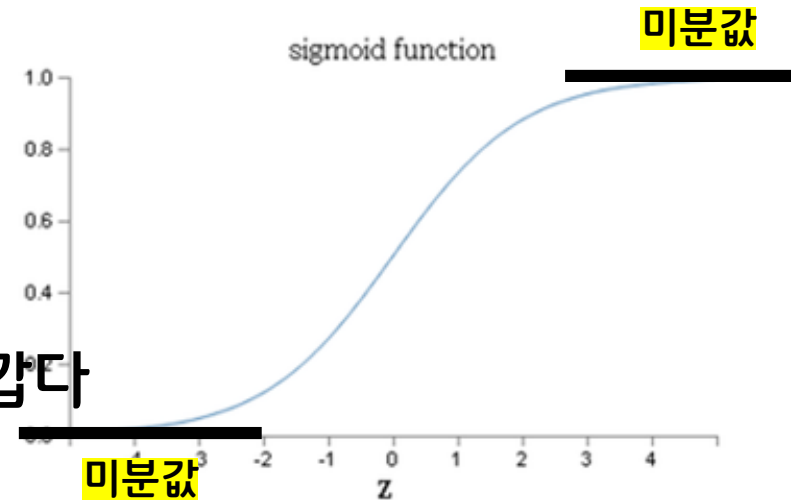
\*hidden layer가 b로 표시되었다...(기호는 왜이렇게 사람마다 다른지...)

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

활성화함수가 시그모이드라면?

시그모이드 함수 생긴 것을 보면  
양 극단값의 기울기가 거의 0이 되므로  
도함수(미분값)를 구한다면 당연히 0에 가깝다



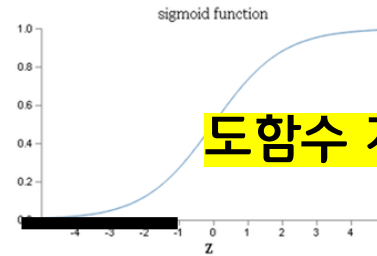
$$a = \sigma(z)$$

$$z = wx + b$$

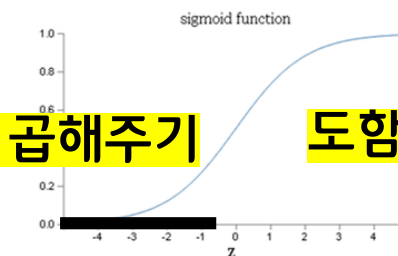
## 2. DNN problem &amp; solution

## 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

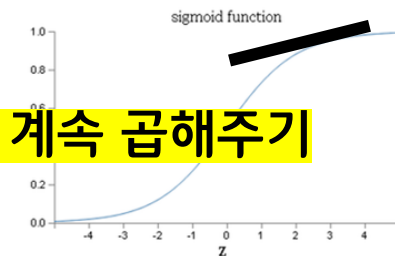
활성화함수가 시그모이드라면?



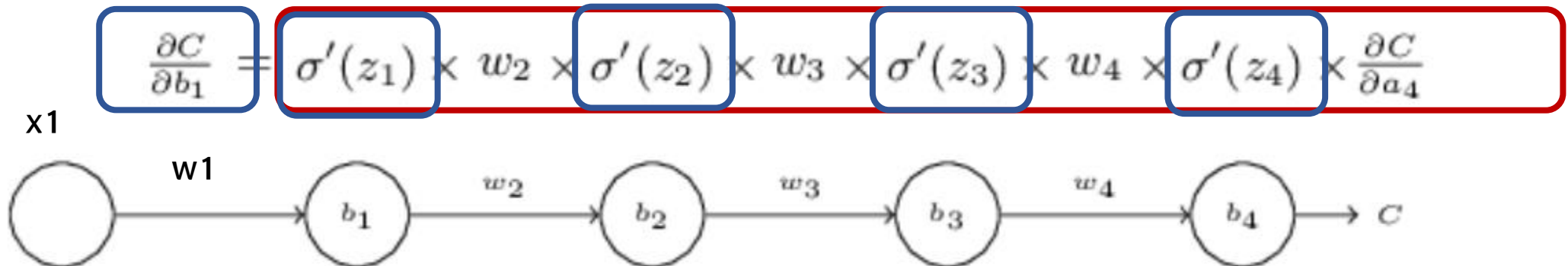
도함수 계속 곱해주기



도함수 계속 곱해주기



계산값이 양 극단값에 가깝다면 0에 가까운 값을 계속 곱해주는 꼴이므로 그래디언트가 소실되는 것!



\*hidden layer가 b로 표시되었다...(기호는 왜이렇게 사람마다 다른지...)



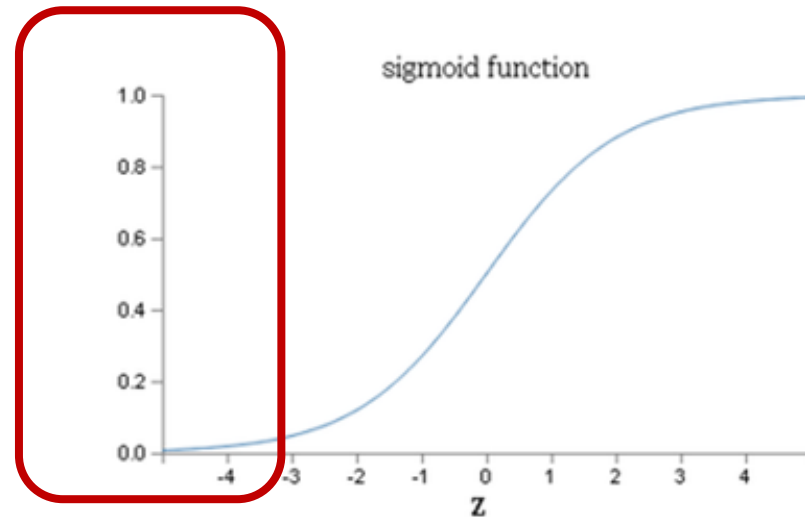
## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

활성화함수가 시그모이드라면?

또 하나의 문제점  
범위가 0부터 1사이다  
즉 모조리 다 양수(+)다

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



요기가 범위니까

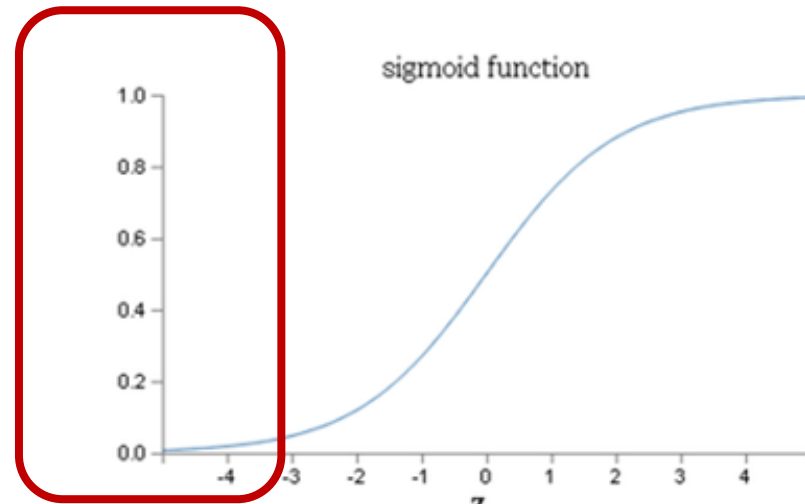
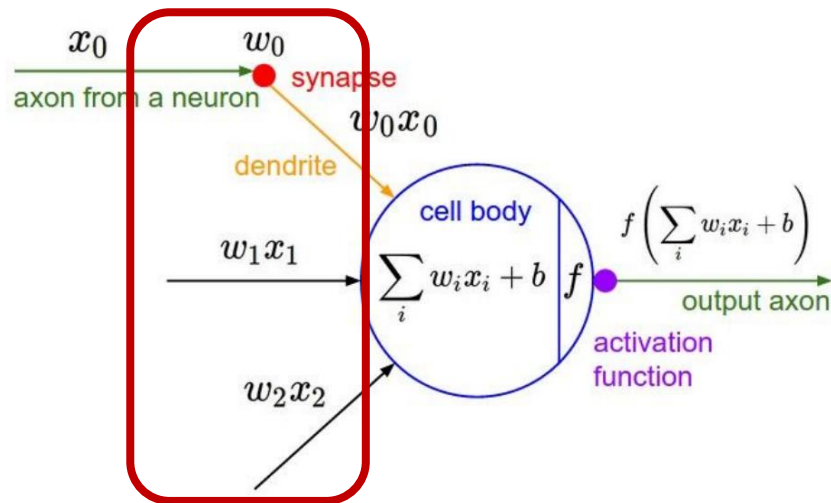
$$a = \sigma(z)$$

$$z = wx + b$$

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

활성화함수가 시그모이드라면?



W의 방향이 똑같아져서 학습이 제대로 안된다고 함

이 값이 들어가는건데 이거의 결과(z)에  
활성화함수 취한 값이 모조리 다 양수  
(부호가 같다)

$$a = \sigma(z)$$

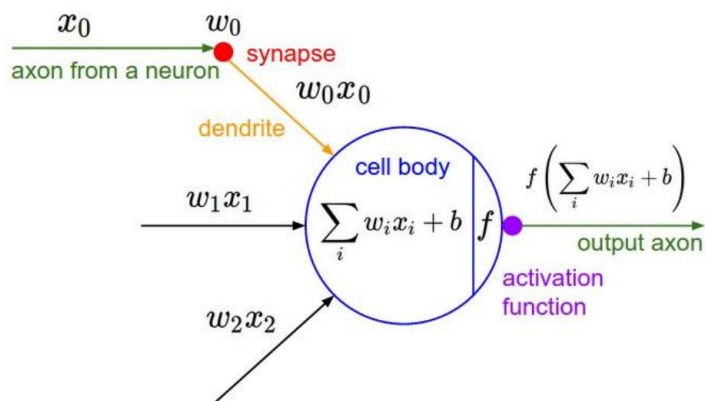
$$z = wx + \text{그래디언트 방향이 같아짐 그러면}$$

어떤 히든유닛이든 입력에 활성화함수 취한 것의 부호가 같으면  
나중에 그 값의 loss를 w로 미분해서 그래디언트 구해도 모두 부호가 같지 않을까?  
w1이든 w2든 w3이든 양수(+)이니까 (직관적으로 생각!!)

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

#### 활성화함수가 시그모이드라면?



정식 설명은 이렇다...

$x_0, x_1, x_2$ 는 모두 0 이상의 값을 갖습니다. 이들은 직전 층에서 시그모이드 함수에 의해 그 값이 양수로 활성화됐기 때문입니다. 여기에서 역전파시 최종 Loss에서 출발해 시그모이드 적용 직전의  $w_i x_i + b$  각각에 들어오는 그래디언트를  $\delta$ 라고 두겠습니다. 그렇다면  $w_i$ 의 그래디언트는 아래와 같습니다.

$$\frac{\partial L}{\partial w_i} = x_i \times \delta$$

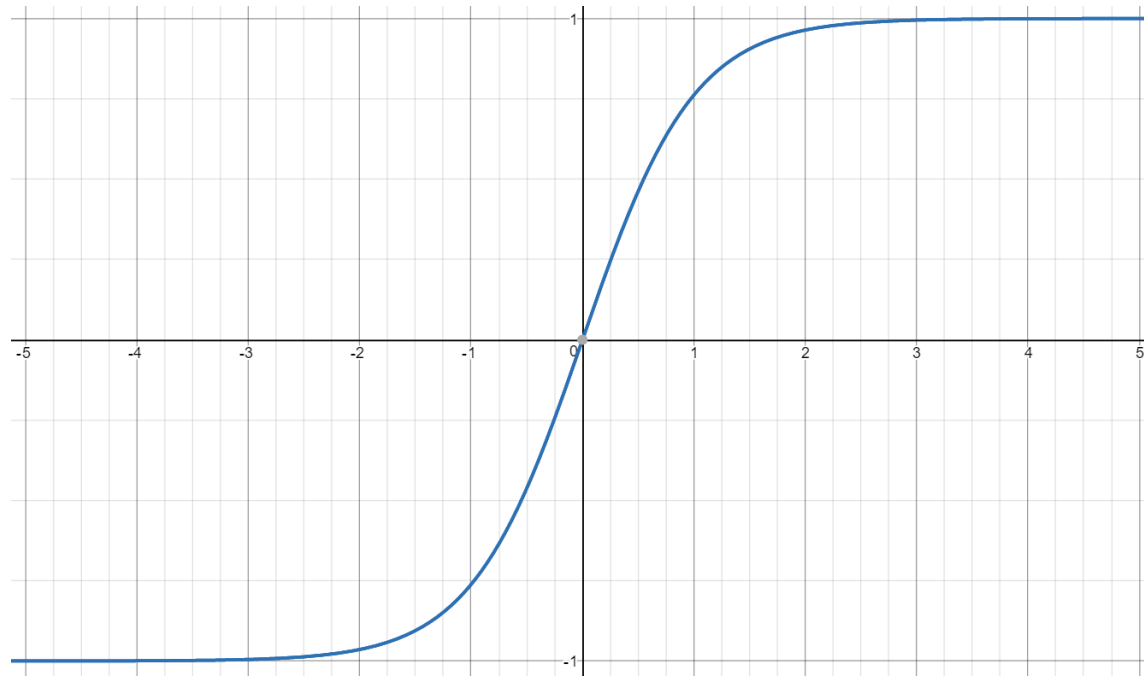
앞서 말씀드렸듯  $x_0, x_1, x_2$ 는 모두 0 이상이기 때문에  $\delta$ 가 양수라면 Loss에 대한  $w_0, w_1, w_2$  각각의 그래디언트가 모두 양수, 반대라면 모두 음수 값이 될 것입니다. 따라서 데이터  $x$ 와 파라미터  $w$ 를 2차원 벡터로 가정해 본다면  $w$ 의 그래디언트는 2사분면과 4사분면 쪽 방향이 될 수는 없습니다.

결과적으로  $w$  학습시 아래 그림처럼 허용되는 방향에 제약이 가해져(요소값이 모두 양수인 '1사분면'과 모두 음수인 '3사분면' 쪽 방향만 선택 가능) 학습속도가 늦거나 수렴이 어렵게 됩니다. 이 문제는 함수값이 0에 대해 대칭(zero-centered)인 하이퍼볼릭탄젠트 같은 함수를 쓰면 극복할 수 있다고 합니다.

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

그래서 나온 함수  $\tanh(x)$

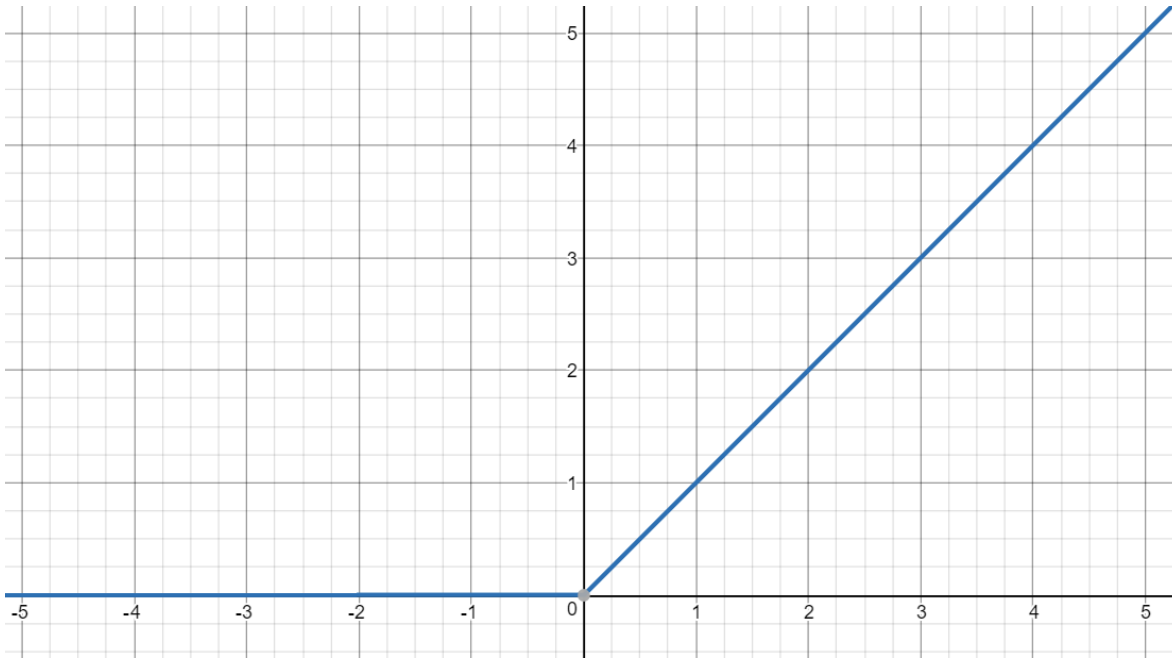


$$\begin{aligned}\tanh(x) &= 2\sigma(2x) - 1 \\ &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ \tanh'(x) &= 1 - \tanh^2(x)\end{aligned}$$

0에서 1사이의 범위가 아닌 -1부터 1사이. 그건 해결했으나 역시 극단값에서 도함수가 0에 가까워지는 것은 해결 못함

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)



$$f(x) = \max(0, x)$$

신경망이 침체기 때 찬밥신세에서  
제대로 붐 떠오르게 해준 장본인이  
바로 ReLU 함수이다

우린 지금 너무나도 당연하게 쓰고있지만... 이 함수를 발견하기 전까지 많은 학자들이 엄청난 삽질과 좌절을 했다고 한다...

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)



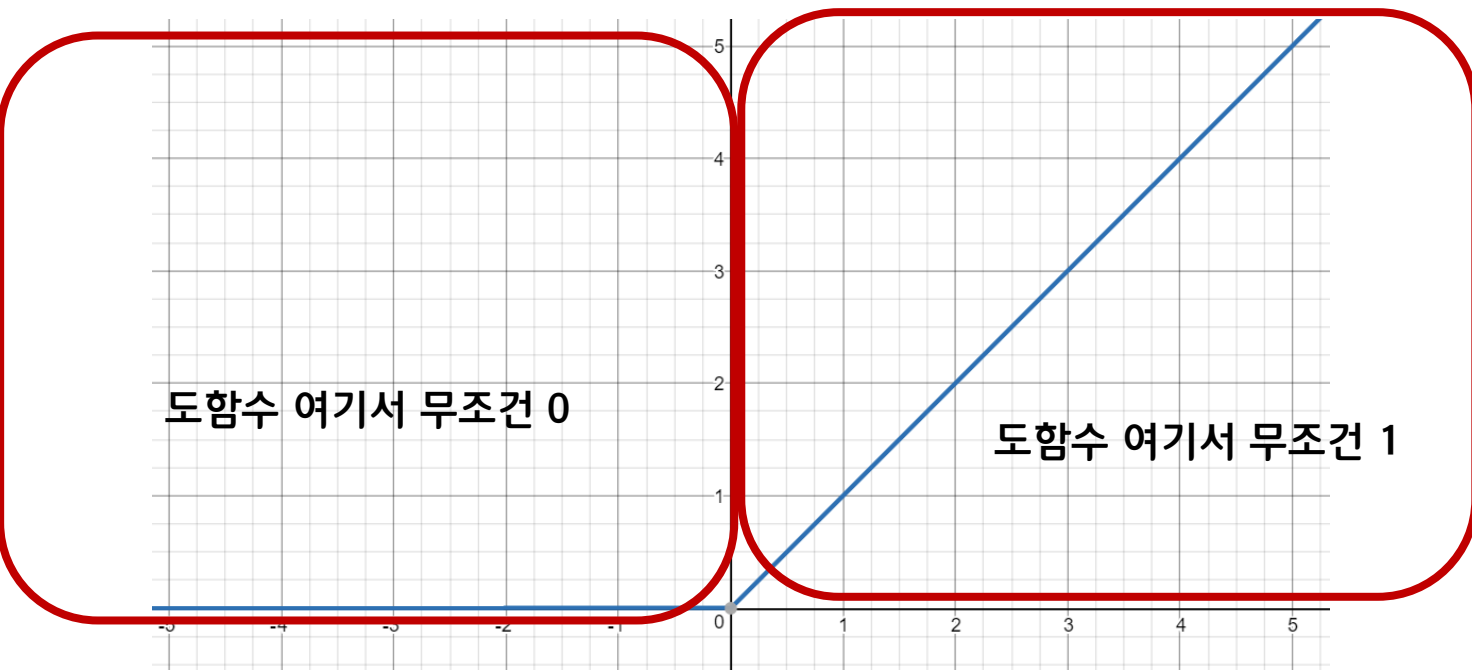
$$f(x) = \max(0, x)$$

일단 입력값이 0보다 작으면 0으로 죽여버리고??  
(일정값 이상 안나오면 학습을 안하겠다는 소리)  
0보다 크면 일차함수로 올라가는 형태이다

학습속도가 빠르다고 알려져있다  
(공식에 e의 지수 이런거 없어서...이게 은근 계산 잡아먹는다고 함)

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)

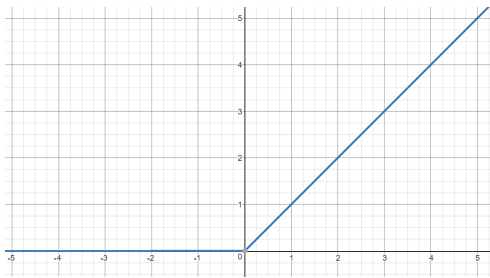


$$f(x) = \max(0, x)$$

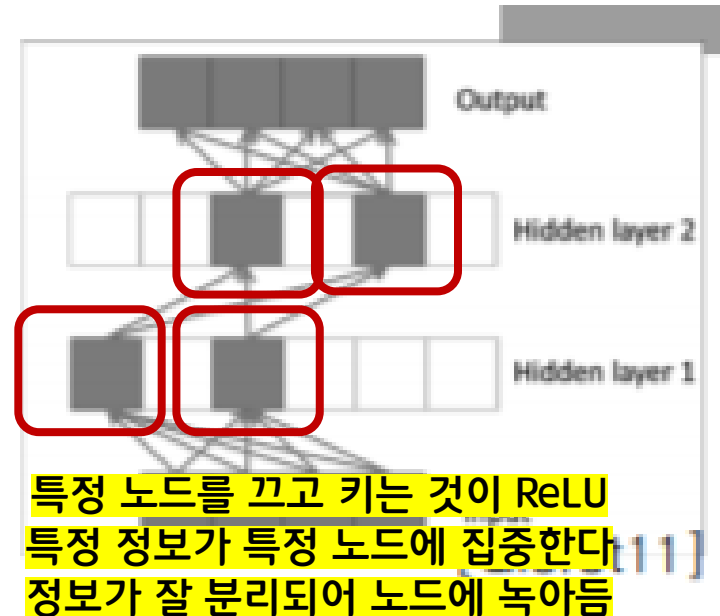
극단값 도함수를 다 0으로 만드는 고질적인 문제 해결,  
또한 어떤 정보를 쓸 건지 매우매우 확실하게 결정

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)



특정 정보를 죽이고살리고가  
왜 중요한가??



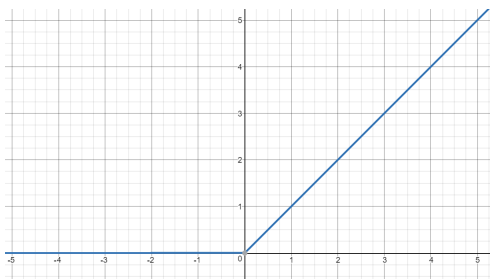


## 2. DNN problem & solution

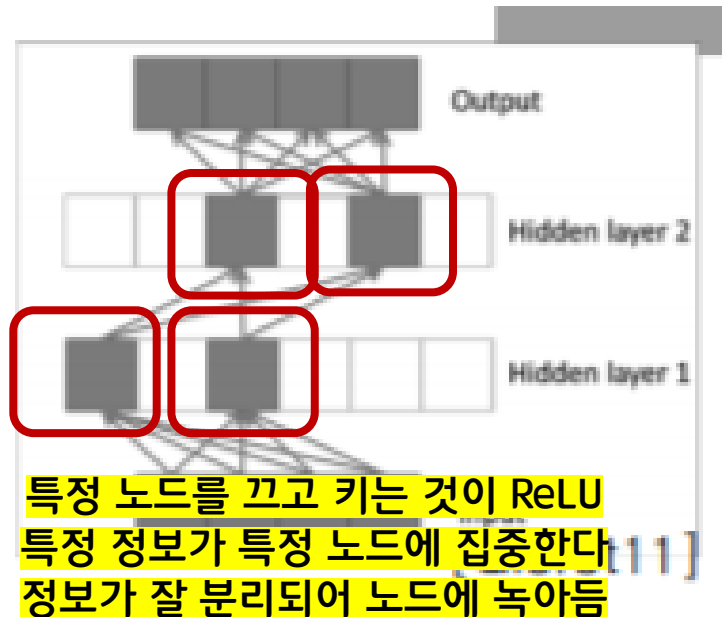
그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)

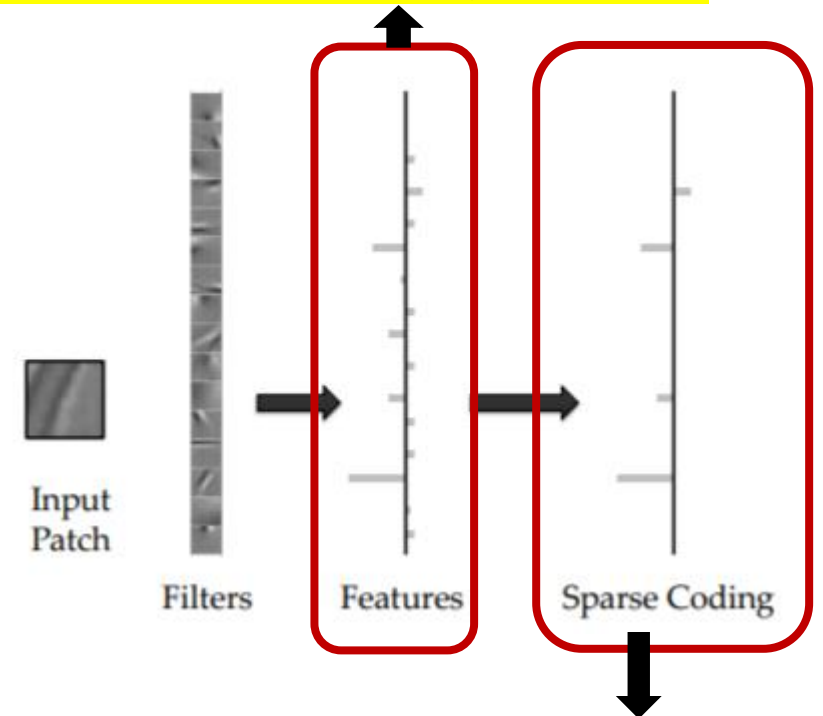
features 정보가 많은 곳에 고르고 퍼져있음  
근데 조금조금 살아있는 애들 대부분 노이즈 임



특정 정보를 죽이고살리고가  
왜 중요한가??



특정 노드를 끄고 키는 것이 ReLU  
특정 정보가 특정 노드에 집중한다  
정보가 잘 분리되어 노드에 녹아들



features 정보가 있을 곳엔 있고 없을 곳에는 없게  
행렬로 치면 0이 많은 sparse한 형태가 노드에 정보가 잘 녹아들 것!  
이것이 ReLU

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)

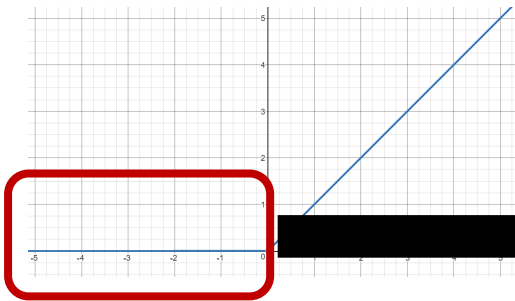


결론은 ReLU가 짱이다

그러나 애도 몇가지 단점이 있다

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)



결론은 ReLU가 짱이다

그러나 애도 몇가지 단점이 있다

입력값이 0안 넘으면 학습 자체를 아예 안시킴  
어느정도 정보 손실시키는 것 가능

## 2. DNN problem & solution

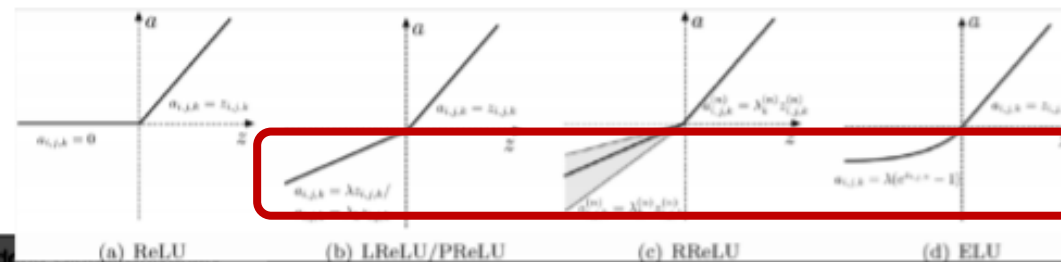
그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

ReLU를 보완하기 위한 렐루 함수의 변형들

- Leaky ReLU (LReLU)
  - $\lambda$  is fixed
- Parametric ReLU (PReLU)
  - $\lambda$  is learned
- Randomized ReLU (RReLU)
  - $\lambda$  is randomly sampled
- Exponential LU (ELU)

$$a_{i,j,k} = \max(z_{i,j,k}, 0) + \lambda_k \min(z_{i,j,k}, 0)$$

$$a_{i,j,k} = \max(z_{i,j,k}, 0) + \min(\lambda(e^{z_{i,j,k}} - 1), 0)$$

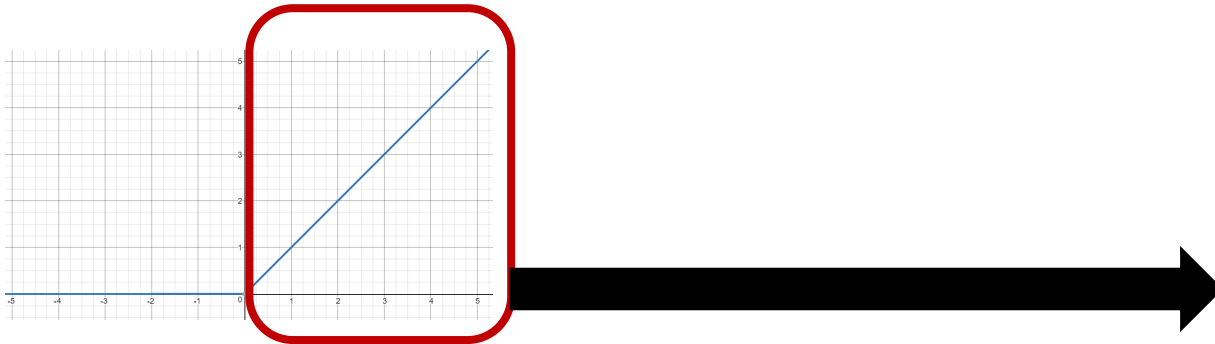


0안넘어도 죽이지 말아주세요 ㅠㅠ

그래서 0이 안 넘어도 좀 덜 죽이려고 많은 노력들을 하여 만든 다양한 ReLU 변형들이 있음

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제  
그래서 나온 렐루 함수 Rectified Linear Unit (ReLU)



결론은 ReLU가 짱이다

그러나 애도 몇가지 단점이 있다

범위가 무한대  
weight 계속 곱하고 더해주다가 굉장히 큰 값  
튀어나올 수 있음 → overfitting

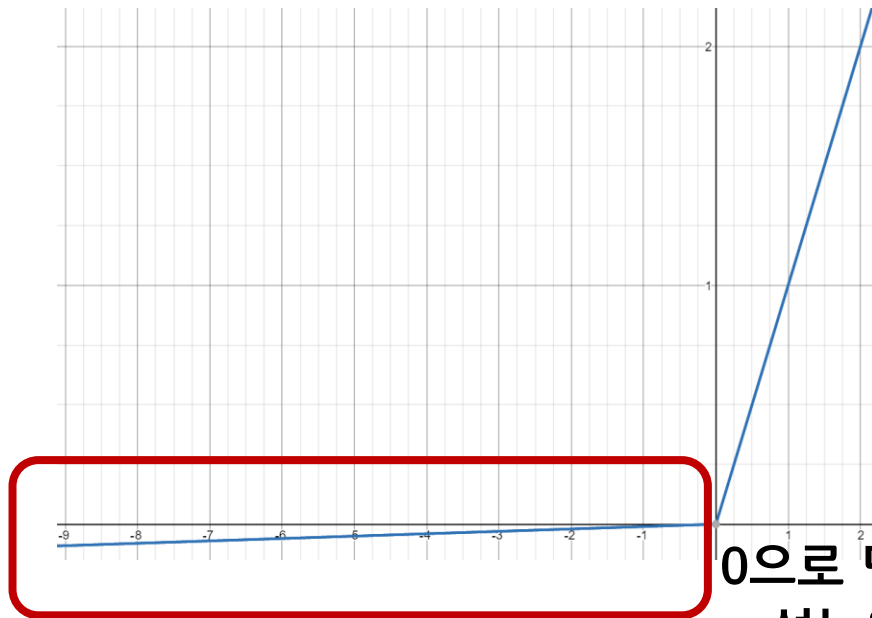
그래서 나중에 정규화라는 걸 통해  
w에 패널티 주는 여러가지 기법이 나옴

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

ReLU를 보완하기 위한 렐루 함수의 변형들

$$f(x) = \max(0.01x, x)$$



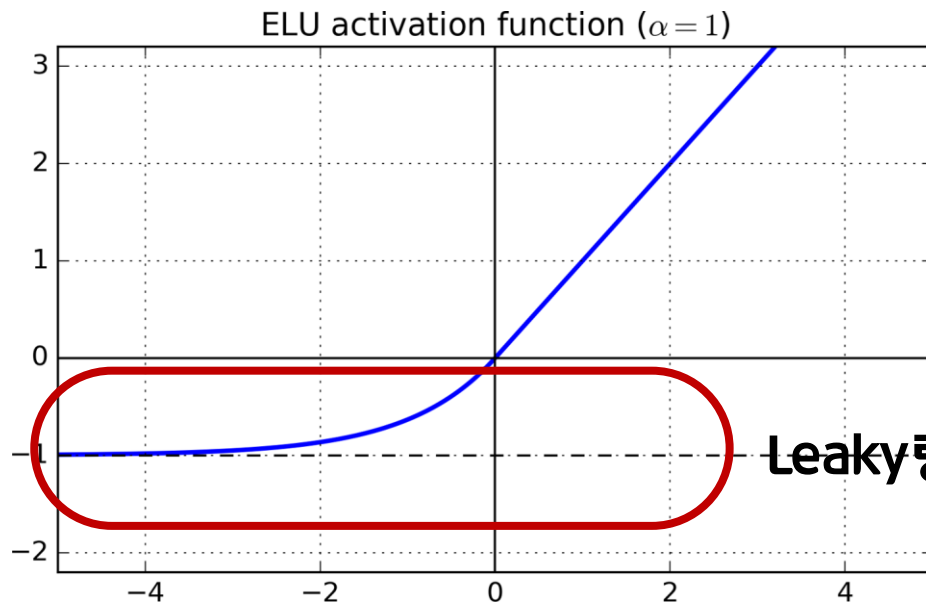
Leaky ReLU

0으로 만드는건 좀 그러니까 애도 학습시킬 여지가 주게 기울기 좀 달리함  
성능은 좋아질 수 있음, 0이 많은 행렬인 sparse한 형태를 벗어나니까  
학습 시간 다소 더 걸릴수도

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

ReLU를 보완하기 위한 렐루 함수의 변형들



ELU

$$f(x) = x \quad \text{if } x > 0$$
$$f(x) = \alpha(e^x - 1) \quad \text{if } x \leq 0$$

알파는  $z$ 가 음수값일 때 ELU 함수 통과한 값이  
어디로 수렴할 지 정의해준다고 함

\*지수함수라서 속도 더 느림

Leaky랑 다른 점은 너무 작으면 어디로 수렴한다는 것?

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

activation 함수가 너무 많다... 도대체 뭘 써야할까?

default는 당연히 LeLU이다. 무난무난하게 학습시키고 싶으면 LeLU쓰자.

성능은

ELU > LeakyReLU > ReLU > Tanh > sigmoid

속도가 좀 더 중요하다면 LeakyReLU가 ELU보다 나을수도 있다...

\*핸즈온 머신러닝에서 추천한 순서... 아직 연구가 이루어지는 중 이라 정답은 없는 것 같다



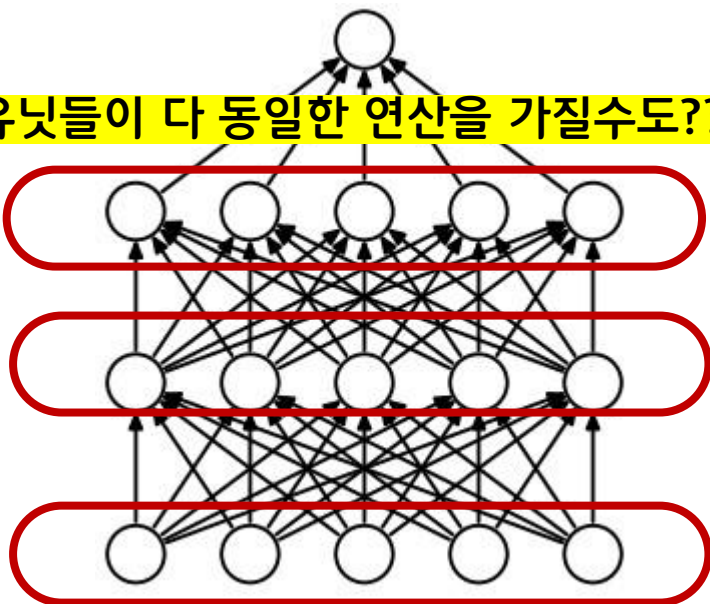
## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

#### 또 하나의 해결책 : 가중치 초기화

데이터를 신경망에 넣기 전에  $w$ 들의 초기값을 정해줘야한다

유닛들이 다 동일한 연산을 가질수도??



가중치( $w$ )를 모두 0으로 초기화하면 안된다!

가중치가 0으로 초기화된 신경망 내의 뉴런들은 모두 동일한 연산 결과를 낼 것

이고 따라서 backpropagation 과정에서 동일한 그래디언트(gradient) 값을 얻게 될 것이고 결과적으로 모든 파라미터(parameter)는 동일한 값으로 업데이트 될 것이기 때문이다.

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

또 하나의 해결책 : 가중치 초기화  
데이터를 신경망에 넣기 전에 w들의 초기값을 정해줘야한다

#### Xavier initialization(세이버 초기화)

$w = \text{np.random.randn}(n) / \text{sqrt}(n)$   
이렇게 초기화 하면 좋다고 함

#### 분산 보정

랜덤값으로 초기화된 뉴런으로 학습되어 나온 결과의 분포가 입력 데이터 수에 비례하여 커지는 분산을 갖는다. 가중치 벡터를 팬인(*fan-in*) (입력 데이터 수)의 제곱근 값으로 나누는 연산을 통하여 뉴런 출력의 분산이 1로 정규화 할 수 있다. 이 방법은 근사적으로 동일한 출력 분포를 갖게 할 뿐만 아니라 신경망의 수렴률 또한 향상시키는 것으로 알려져 있다.

#### He initialization(헤 초기화)

$w = \text{np.random.randn}(n) * \text{sqrt}(2.0/n)$   
나중에 LeLU 쓸 땐 이렇게 초기화하는 것이 좋다는 것이 밝혀져서 이렇게 한다

\*앤드류 교수님께서 초기화 하이퍼 파라미터는 우선순위에 있는 조정값이 아니라고 했다

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

그러다가 가중치 초기화에 딱히 영향을 받지 않는 방법이 나왔다

## Batch Normalization

신경망 학습단계에서 activation 값이 표준정규분포를 갖도록 강제하는 기법으로 신경망을 적절한 값으로 초기화하여 그동안 많은 연구자들을 괴롭혀왔던 초기화 문제의 상당부분을 해소해 주었다.

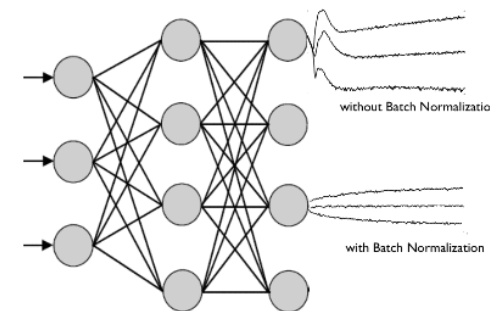
이 방법 쓰면 웬만하면 성능은 올라간다고 함

## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

#### Batch Normalization

Batch Normalization은 기본적으로 Gradient Vanishing / Gradient Exploding 이 일어나지 않도록 하는 근본적인 해결책으로 나온 아이디어 중의 하나이다.



어떤 이들은 이러한 불안정화가 일어나는 이유가 'Internal Covariance Shift(내부 공변량 변화)' 라고 주장하고 있다.

Internal Covariance Shift라는 현상은 Network의 각 층이나

Activation 마다 input의 distribution이 달라지는 현상을 의미한다.

학습 시 현재 layer의 입력은 모든 이전 layer의 파라미터의 변화에 영향을 받게 되며,

망이 깊어짐에 따라 이전 layer에서의 작은 파라미터 변화가 증폭되어 뒷 단에 큰 영향을 끼치게 될 수도 있다.

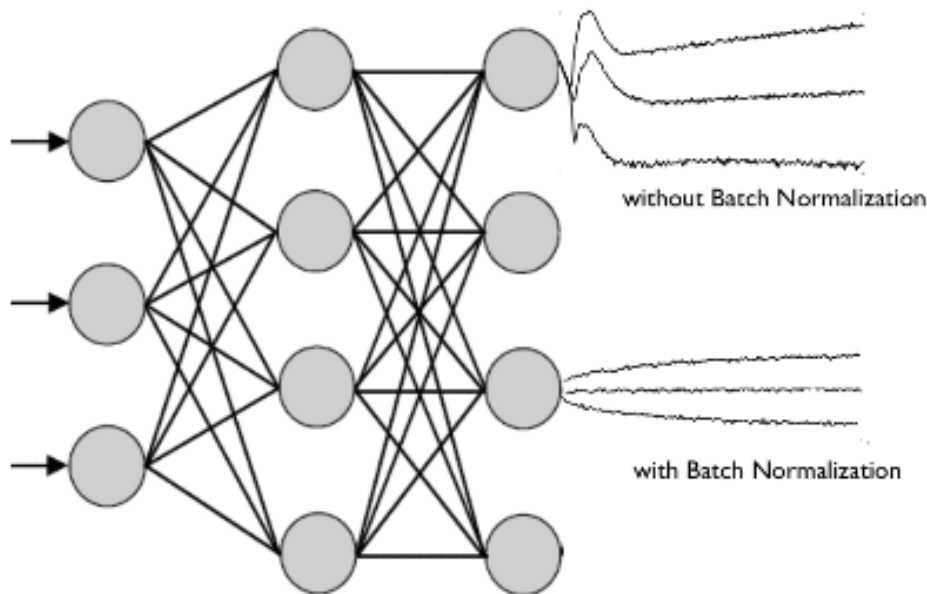
이처럼, 학습하는 도중에 이전 layer의 파라미터 변화로 인해

현재 layer의 입력의 분포가 바뀌는 현상을 "Covariate Shift"라고 한다

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

### Batch Normalization



각 레이어를 걸치면서 입력값의 분포가 달라진다

그래서 각 레이어를 거칠 때마다 그 분포를 normalize해준다

전체의 평균과 분산보다는 mini-batch의 평균과 분산을 구해서 그것으로 정규화

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

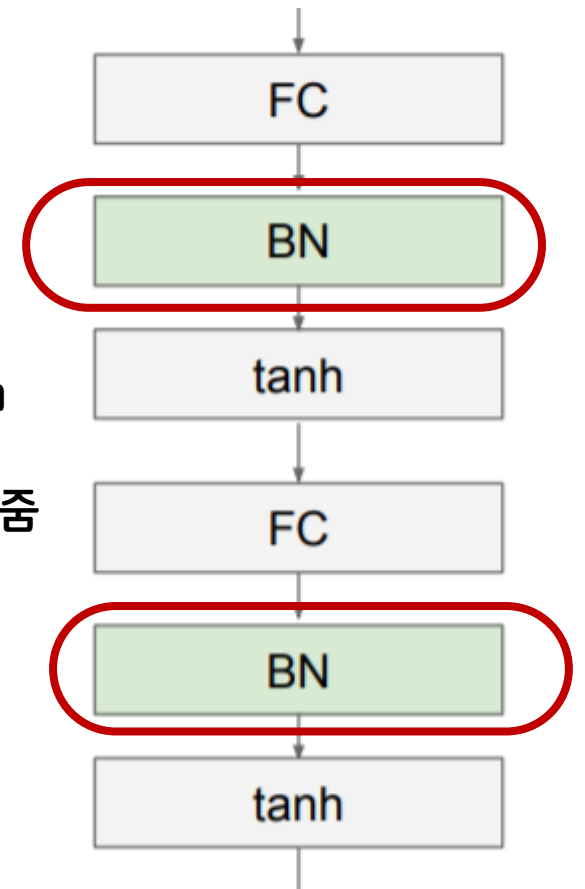
### Batch Normalization 하는 법

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

미니배치의 분산

미니배치의 평균

보통 activation function  
거치기 전에  
Batch Normalization을 해줌



## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

### Batch Normalization 하는 법

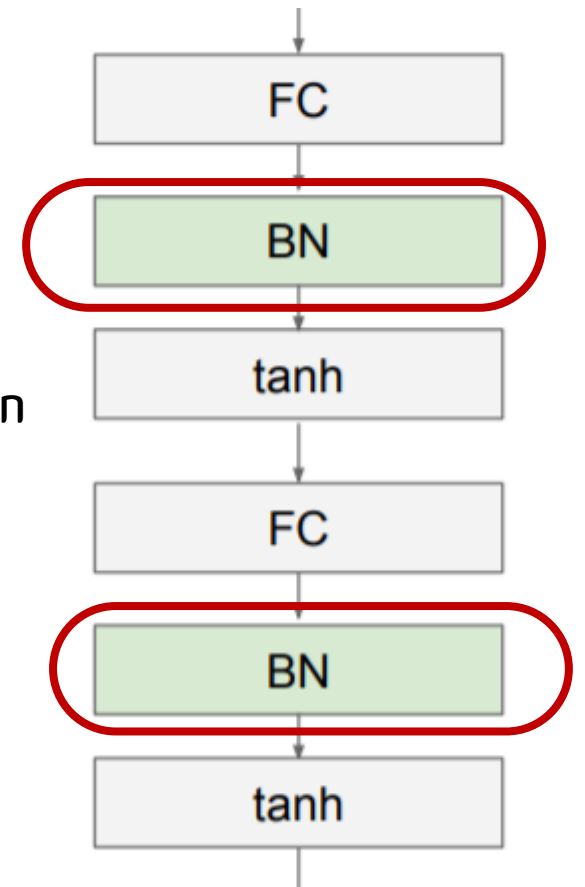
$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

미니배치의 평균

미니배치의 분산

가우시안 분포???

보통 activation function  
거치기 전에 Batch  
Normalization을 해줌



**의문 : 모두 가우시안 분포로 맞춰주는게 맞음?**

## 2. DNN problem &amp; solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

## Batch Normalization 하는 법

의문 : 모두 가우시안 분포로 맞춰주는게 맞음?

그렇다면 그것조차 학습해버리겠다!

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1...m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

정규화 된 것을 다시 한번 조정하는 파라미터 넣음 이것도 훈련

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

scaling

shift

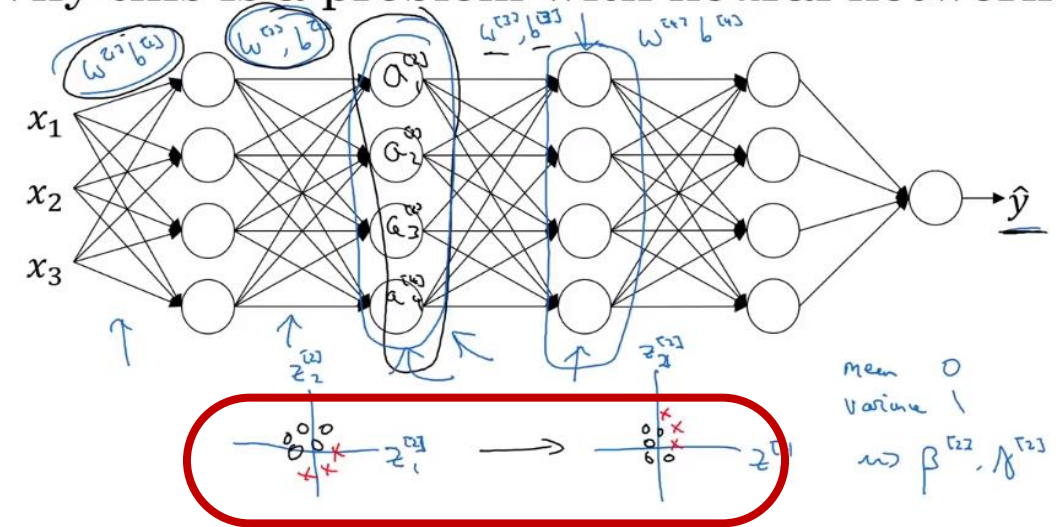


## 2. DNN problem &amp; solution

# 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제 Batch Normalization 좋은 점

일단 우리가 같은 classification 문제라도 training셋이 조금만 달라지면 처음부터 다시 학습을 해야 하는데 그 가장 큰 이유는  $W$  값 즉 parameter 값이 학습할 때마다 달라지기 때문이다.

Why this is a problem with neural networks?



학습셋 분포의 상대적 위치가 같다

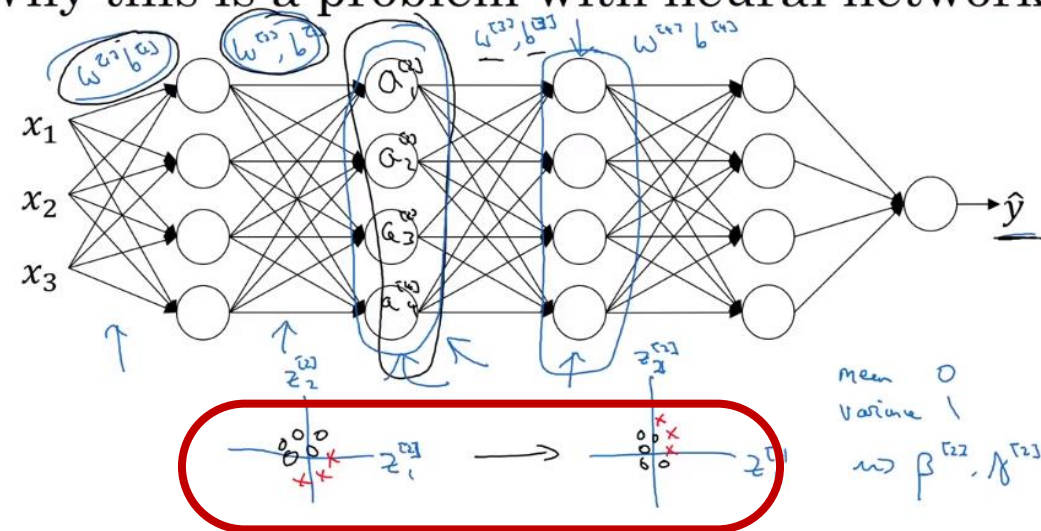
## 2. DNN problem & solution

### 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제 Batch Normalization 좋은 점

일단 우리가 같은 classification 문제라도 training셋이 조금만 달라지면

면 처음부터 다시 학습을 해야 하는데 그 가장 큰 이유는 W값 즉 parameter값이 학습할 때마다 달라지기 때문이다. 조금 더 자세하게 NN내부를 들여다보면 중간노드 A[2]의 경우 앞의 노드 A[1] 및 X값에 의해 영향을 받는다. 하지만, **Batch norm을 적용**하게 되면 X, A[1], A[2]의 값 모두가 전체 대비 **해당 값으로 normalization되기 때문에** 일정한 비율로서 계산이 이루어진다 따라서 Vector space상의 그래프로 본다면 아래 그림의 두 가지 그래프처럼, 같은 training셋으로 다른 순서로 학습을 하든 조금 학습셋을 틀던 간에 **vector space상에 값은 다를지 몰라도 상대적인 위치는 유지하게 된다.**

Why this is a problem with neural networks?



**학습셋 분포의 상대적 위치가 같다**

## 2. DNN problem & solution

그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제

### Batch Normalization 좋은 점

1. 기존 Deep Network에서는 learning rate를 너무 높게 잡을 경우 gradient가 explode/vanish 하거나, 나쁜 local minima에 빠지는 문제가 있었다. 이는 parameter들의 scale 때문인데, Batch Normalization을 사용할 경우 propagation 할 때 parameter의 scale에 영향을 받지 않게 된다. 따라서, **learning rate를 크게 잡을 수 있게** 되고 이는 빠른 학습을 가능케 한다.
2. Batch Normalization의 경우 자체적인 **regularization** 효과가 있다. 이는 기존에 사용하던 weight regularization term 등을 제외할 수 있게 하며, 나아가 Dropout을 제외할 수 있게 한다 (Dropout의 효과와 Batch Normalization의 효과가 같기 때문.) . Dropout의 경우 효과는 좋지만 학습 속도가 다소 느려진다는 단점이 있는데, 이를 제거함으로써 **학습 속도도 향상**된다.
3. 실제 적용 사례를 보면 배치 정규화(Batch Normalization)를 사용하여 학습한 신경망은 특히 나쁜 초기화의 영향에 강하다는 것이 밝혀졌다.

## 2. DNN problem & solution

# 그래디언트 소실(vanishing gradient) 또는 그래디언트 폭주(exploding gradient) 문제 Batch Normalization 좋은 점

앤드류 응 교수님의 첨언을 빌리자면 배치 정규화의 regularization는 부수적인 것이며,  
자신은 그 목적으로 사용하지 않는다고 했다  
(드롭아웃이랑 같이 사용하라고 했음)

또한 배치 정규화의 가장 큰 목적은 하이퍼파라미터의 탐색을 쉽게 해주며,  
신경망에 하이퍼파라미터의 상관관계를 줄여준다고 하였다.  
더 많은 하이퍼파라미터에 잘 작동하기 때문에

아마도 learning rate나 가중치 초기화에 영향을 덜 받기 때문이 아닌가 싶다.

## 2. DNN problem & solution

### 2. 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

```
In [*]: learn.fit(epochs=3, lr=1e-4, callbacks=[Debugger()])
```

66.67% [2/3 00:09<00:04]

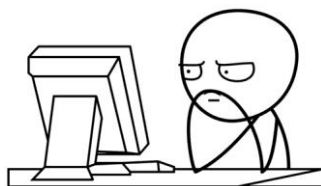
epoch train loss valid loss

0 2.332582 2.394771

1 2.161388 2.445782

0.00% [0/5 00:00<00:00]

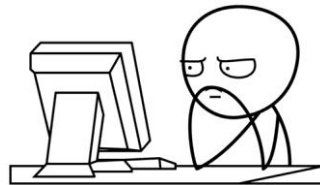
끝나라... 끝나라...



신경망은 깊어지면 연산이 많아져서 매우 매우 느려짐

## 2. DNN problem & solution

**대규모 신경망에서는 훈련이 극단적으로 느려지는 문제**

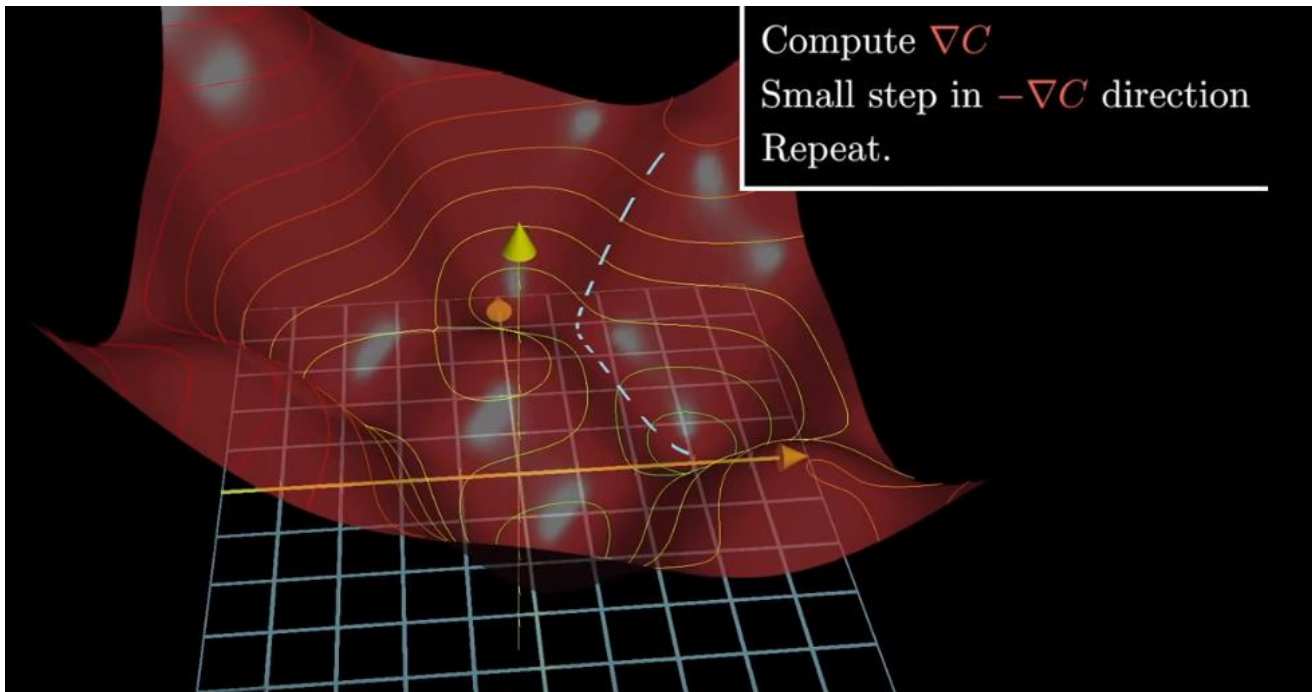


**신경망 학습을 빠르게 하는 옵티마이저(Optimization)를 소개한다**

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

### 신경망 Optimization(최적화)



결국 신경망 연산을 빠르게 한다는 것은  
이 gradient decent(경사하강법)을  
빠르게 진행한다는 뜻이다



## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

### 신경망 Optimization(최적화)

결국 데이터도 행렬로 들어감

이걸 한번에 모델에 넣어서 학습 시켜야 하나?

그러면 gradient decent(경사하강법)도 한번에 됨

$$X = \begin{pmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m-1)} & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m-1)} & x_1^{(m)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{12286}^{(1)} & x_{12286}^{(2)} & \dots & x_{12286}^{(m-1)} & x_{12286}^{(m)} \\ x_{12287}^{(1)} & x_{12287}^{(2)} & \dots & x_{12287}^{(m-1)} & x_{12287}^{(m)} \end{pmatrix}$$

한번에 학습하셈

$$Y = \begin{pmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m-1)} & y^{(m)} \end{pmatrix}$$

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

### 신경망 Optimization(최적화)

그러나 이렇게 계산을 할 경우 한번 step을 내딛을 때 전체 데이터에 대해 Loss Function을 계산해야 하므로 너무 많은 계산량이 필요하다.

$$X = \begin{pmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m-1)} & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m-1)} & x_1^{(m)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{12286}^{(1)} & x_{12286}^{(2)} & \dots & x_{12286}^{(m-1)} & x_{12286}^{(m)} \\ x_{12287}^{(1)} & x_{12287}^{(2)} & \dots & x_{12287}^{(m-1)} & x_{12287}^{(m)} \end{pmatrix}$$

한번에 학습하셈

$$Y = \begin{pmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m-1)} & y^{(m)} \end{pmatrix}$$

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

### 신경망 Optimization(최적화)

그래서 한번에 학습을 안 시키고 **일정 개수로 나눠서**

**행렬(numpy)로 넣은 다음 학습을 시키는 것을**

mini batch gradient descent(미니 배치 경사하강법) 이라고 한다

가중치는  $4 * 3$  매트릭스

150개 데이터에 대해 5000번씩  
학습시키고,

1000번째마다 가중치를 찍어본 것

배치 당 weight 표시

```
[ -8.74615472e-05  3.03732950e-05  4.65428693e-05 ]  
[ -3.23668516e-05  4.42560055e-06 -2.56037962e-05 ]  
[ -2.43068233e-05  9.68563102e-05 -6.81574199e-05 ]  
[ -1.63824136e-04  2.12286420e-04  7.43973291e-05 ]  
[ -0.00321581  0.00061467  0.0025906 ]  
[  0.02404027 -0.01065741 -0.01343641 ]  
[ -0.05779136  0.01472877  0.04306699 ]  
[ -0.0255728  0.00370264  0.02199303 ]  
[  0.00861248 -0.00040284 -0.00822019 ]  
[  0.05412096 -0.02178615 -0.03238836 ]  
[ -0.10186908  0.02754678  0.07432669 ]  
[ -0.04607576  0.00652109  0.03967753 ]  
[  0.02155924 -0.00163555 -0.01993423 ]  
[  0.08324693 -0.03262187 -0.0506786 ]  
[ -0.1418079  0.03937622  0.10243607 ]  
[  0.06481271  0.00882456  0.05601101 ]  
[  0.03370442 -0.00258894 -0.03112603 ]  
[  0.11051824 -0.0428869 -0.06768488 ]  
[ -0.17908648  0.05050046  0.12859041 ]  
[ -0.08228968  0.01100061  0.07141193 ]
```

일주일만에 강의자료 재활용..

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### 신경망 Optimization(최적화)

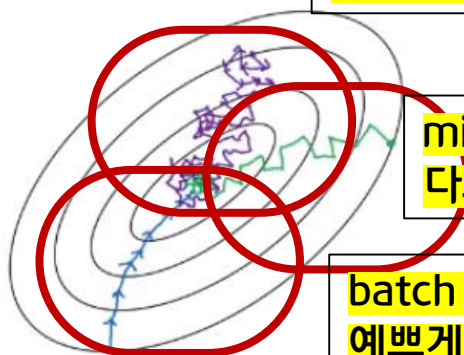
기존 batch (전체데이터 기준) gradient decent의 경우 loss값이 전체 트레이닝셋(평균)에 대한 W값의 업데이트 이므로 항상 낮아지는 반면 batch의 경우 일부분의 데이터에 초점을 맞춰 전체 W값을 업데이트 하니 중구난방 식으로 업데이트가 된다 그럼에도 불구하고 시간이 갈수록 Loss는 줄어든다. 결국 계속 돌면 전체가 되니. 여기서 위에서 정의한 mini-batch 사이즈  $t = 1$ 로 하면 각 training예제 마다 W를 업데이트 한다는건데, 이런걸 stochastic gradient descent라고 한다

stochastic gradient descent. 계속 실시간으로 하나씩 w를 업데이트하니까 중구난방

mini batch gradient decent. 일부를 받아서 loss를 줄이는 방향을 향하니까 다소 중구난방. 배치마다 loss 방향과 크기가 다르므로. 그래도 전체적으로 줄어듬

batch gradient decent. 전체를 받아서 loss를 줄이는 방향을 향하니까 예쁘게 줄어든다 대신 속도 개느림

loss가 줄어드는 그림



## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

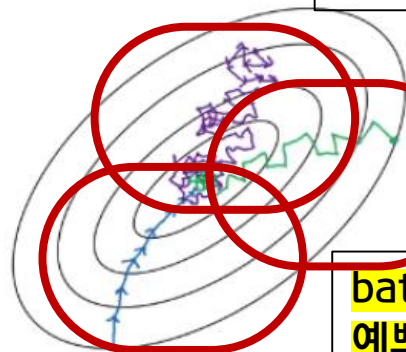
### 신경망 Optimization(최적화)

batch size의 적정선을 찾는 것은 매우 중요

컴퓨터 연산 방식 특성상 2의 배수로 하는 것이 적절하다고 함. 32, 64, 128 ...

64개에서 512개 사이가 일반적으로 정하는 배치 사이즈라고 함

loss가 줄어드는 그림



stochastic gradient descent. 계속 실시간으로 하나씩  $w$ 를 업데이트하니까 중구난방

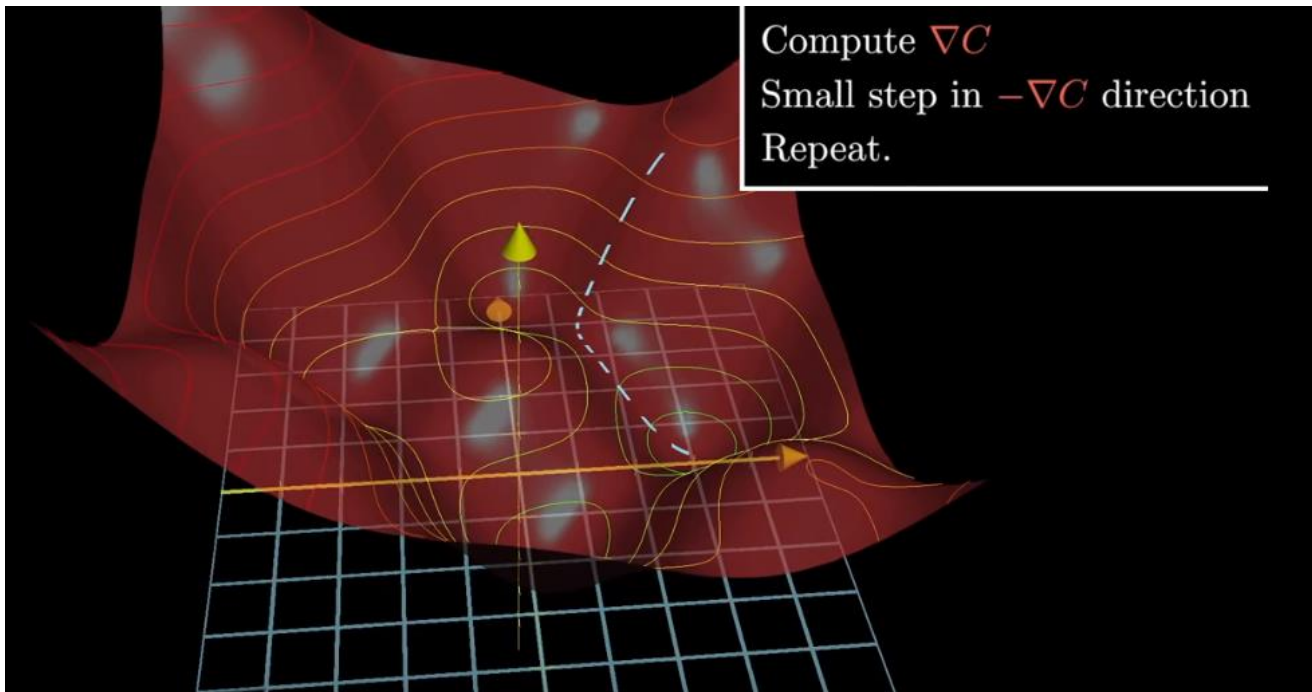
mini batch gradient decent. 일부를 받아서 loss를 줄이는 방향을 향하니까 다소 중구난방. 배치마다 loss 방향과 크기가 다르므로. 그래도 전체적으로 줄어듬

batch gradient decent. 전체를 받아서 loss를 줄이는 방향을 향하니까 예쁘게 줄어든다 대신 속도 개느림

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

### 신경망 Optimization(최적화)



혹시 gradient decent(경사하강법) 자체를  
좀 더 효율적으로 하는 방법이 있을까?

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

신경망 Optimization(최적화)

gradient decent(경사하강법)는  $w$ 를 loss를 고려해 업데이트 시켜주는 것이며,

그 과정에서 **속도**와 **방향**이 중요하다

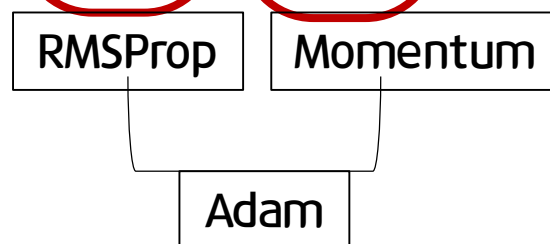
## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

신경망 Optimization(최적화)

gradient decent(경사하강법)는  $w$ 를 loss를 고려해 업데이트 시켜주는 것이며,

그 과정에서 **속도**와 **방향**이 중요하다



모멘텀은 경사하강법의 방향 조절, 즉 이전에 하강이 많이 된  $w$ 에 더 가중치를 줘서 하강시키는 것이며, RMSProp는  $w$ 의 하강 속도가 빠르지도 느리지도 않게 해주는 것, 너무 빠르면 최적점에 도착하지 못하고 주위를 맴돌 수도 있음



## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### Momentum

현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서  
그 방향으로 일정 정도를 추가적으로 이동하는 방식이다

모멘텀 없으면 방향 제대로 못 잡고 더 헤멘다



Image 2: SGD without momentum

모멘텀이 있으면 가던길로 더 밀어버려서 방향을 잘 잡아줌



Image 3: SGD with momentum

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### Momentum

현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서  
그 방향으로 일정 정도를 추가적으로 이동하는 방식이다

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$\gamma$ 는 얼마나 momentum을 줄 것인지에 대한

학습률과 그레디언트 곱한 것

momentum term 보통 0.9를 쓰며,

$$\theta = \theta - v_t$$

얼마나 이전 그레디언트에 step(비중)을 주면서 밀건가를 나타냄

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### Momentum

현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서  
그 방향으로 일정 정도를 추가적으로 이동하는 방식이다

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$\gamma$ 는 얼마나 momentum을 줄 것인지에 대한

학습률과 그레디언트 곱한 것

momentum term 보통 0.9를 쓰며,

$$\theta = \theta - v_t$$

얼마나 이전 그레디언트에 step(비중)을 주면서 밀건가를 나타냄

계속 누적시켜온 모멘텀을

반영한 값으로 loss 업데이트

누적된 v만큼 더 내려가라는 의미

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Nesterov Accelerated Gradient(NAG) 네스테로프 모멘텀 최적화 혹은 네스테로프 가속 경사

$$v_t = \eta \nabla_{\theta} J(\theta)_t + \gamma \eta \nabla_{\theta} J(\theta)_{t-1} + \gamma^2 \eta \nabla_{\theta} J(\theta)_{t-2} + \dots$$

이동항  $v_t$  는 다음과 같은 방식으로 정리할 수 있어, Gradient들의 지수평균을 이용하여 이동한다고도 해석할 수 있다.

누적 누적 누적...

```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

python 코드

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Nesterov Accelerated Gradient(NAG) 네스테로프 모멘텀 최적화 혹은 네스테로프 가속 경사

$$v_t = \eta \nabla_{\theta} J(\theta)_t + \gamma \eta \nabla_{\theta} J(\theta)_{t-1} + \gamma^2 \eta \nabla_{\theta} J(\theta)_{t-2} + \dots$$

이동항  $v_t$  는 다음과 같은 방식으로 정리할 수 있어, Gradient들의 지수평균을 이용하여 이동한다고도 해석할 수 있다.

누적 누적 누적...

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$
$$\theta = \theta - v_t$$

지수가중평균을 이용하는 건데  $\gamma$  을 크게 하면  $v$ 가 커지고 그러면 미는 가중치를 더 주게 된다  
보통은 0.9를 쓴다는 것 (조금만 올려도 가중치는 많이 올라감.. 누적누적이기 때문)

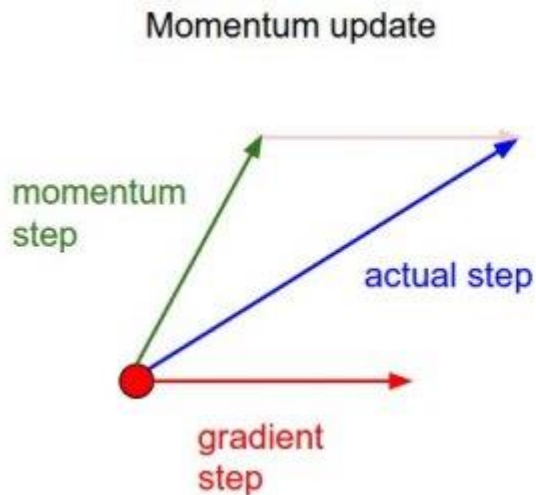
\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### Momentum

NAG에서는 momentum step을 먼저 고려하여, momentum step을 먼저 이동했다고 생각한 후 그 자리에서의 gradient를 구해서 gradient step을 이동한다.



$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

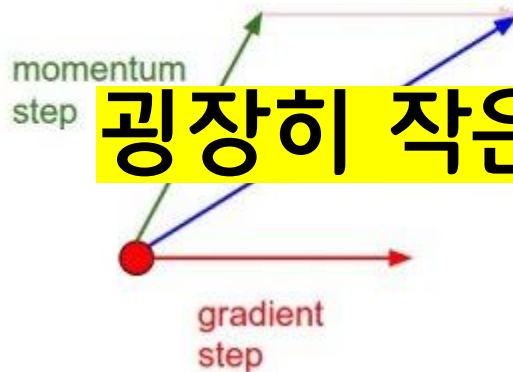
### Momentum

NAG0

```
v_prev = v # back this up
v = mu * v - learning_rate * dx # velocity update stays the same
x += -mu * v_prev + (1 + mu) * v # position update changes form
```

한 후

Momentum update



Nesterov momentum update

이게 더 정확하겠죠?

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

굉장히 작은 개선이지만 축적되면 나름 힘을 발휘해

더 좋은 성능을 낸다고 합니다

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### Adagrad

learning rate을 경사의 상태에 따라 조정하는 방법이다

변화를 많이 한 변수(파라미터)들의 경우 optimum에 가까이 있을 확률이 높기 때문에

작은 크기로 이동하면서 세밀한 값을 조정하고,

적게 변화한 변수들은 optimum 값에 도달하기 위해서는 많이 이동해야할 확률이 높기 때문에

먼저 빠르게 loss 값을 줄이는 방향으로 이동하려는 방식

경사를 하강할 때 가파르게 하강한 차원에서는 그래디언트 벡터의 크기를 감소시키자

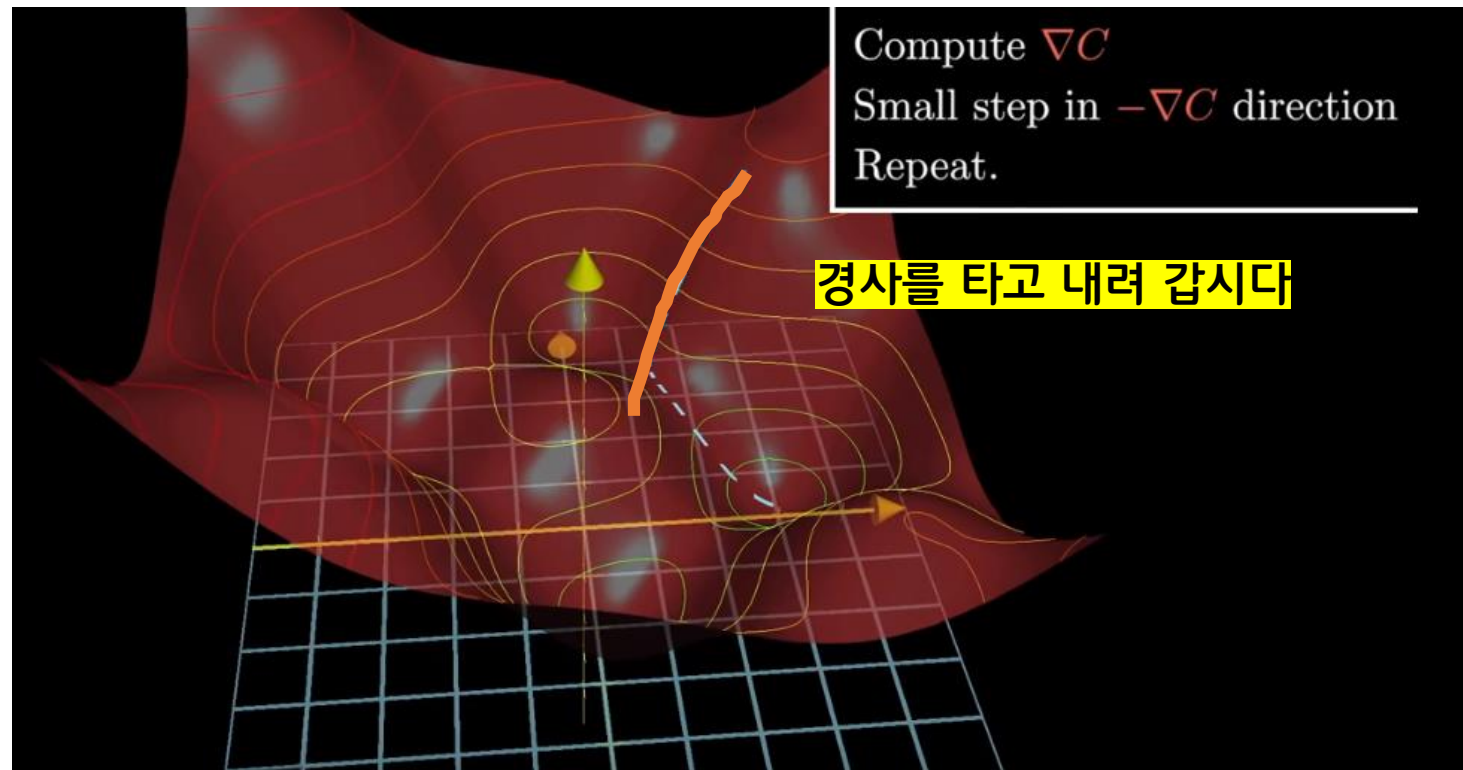


## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

그레디언트 크기에 따라 학습률(경사 한번 타는 폭)을 조정해야 되는 이유

Adagrad

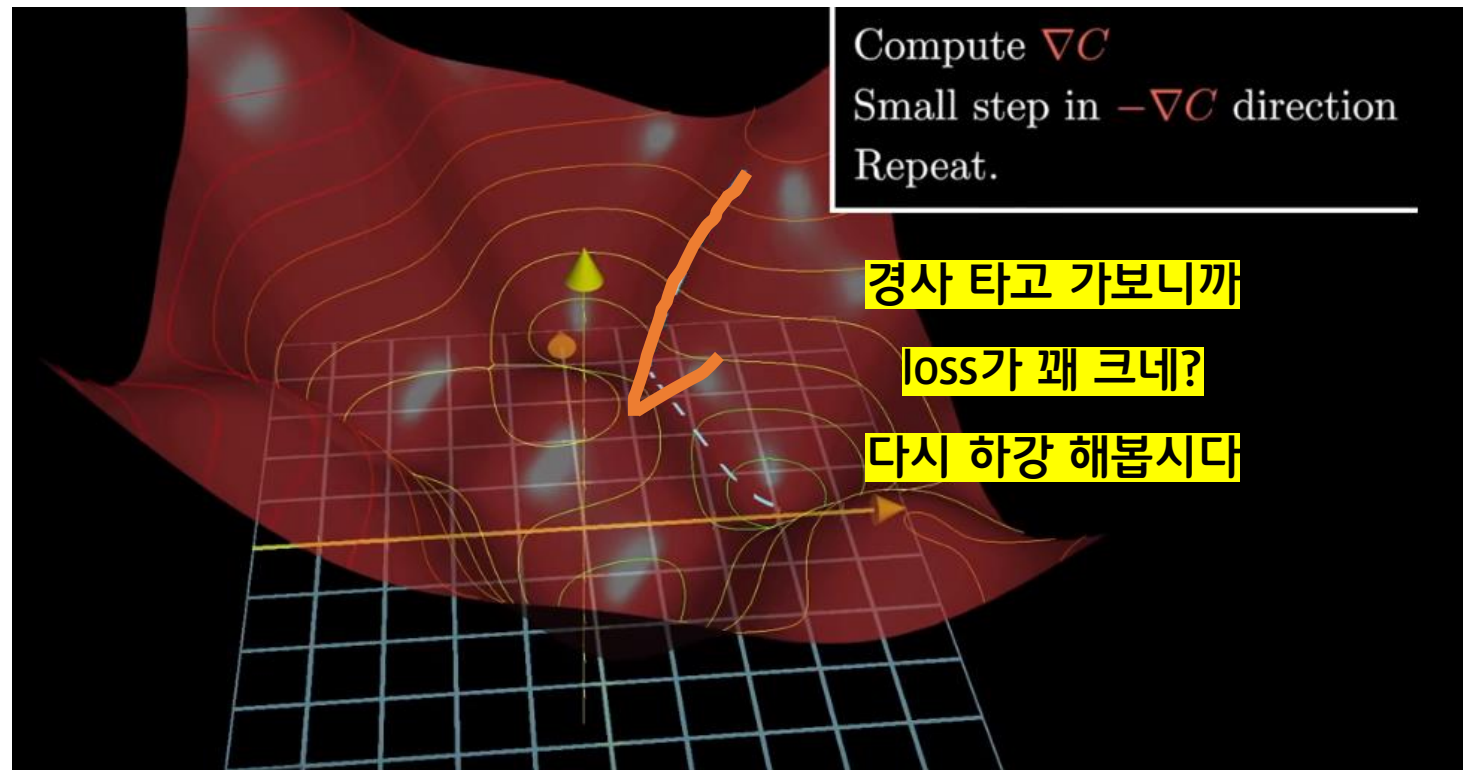


## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

그레디언트 크기에 따라 학습률(경사 한번 타는 폭)을 조정해야 되는 이유

Adagrad

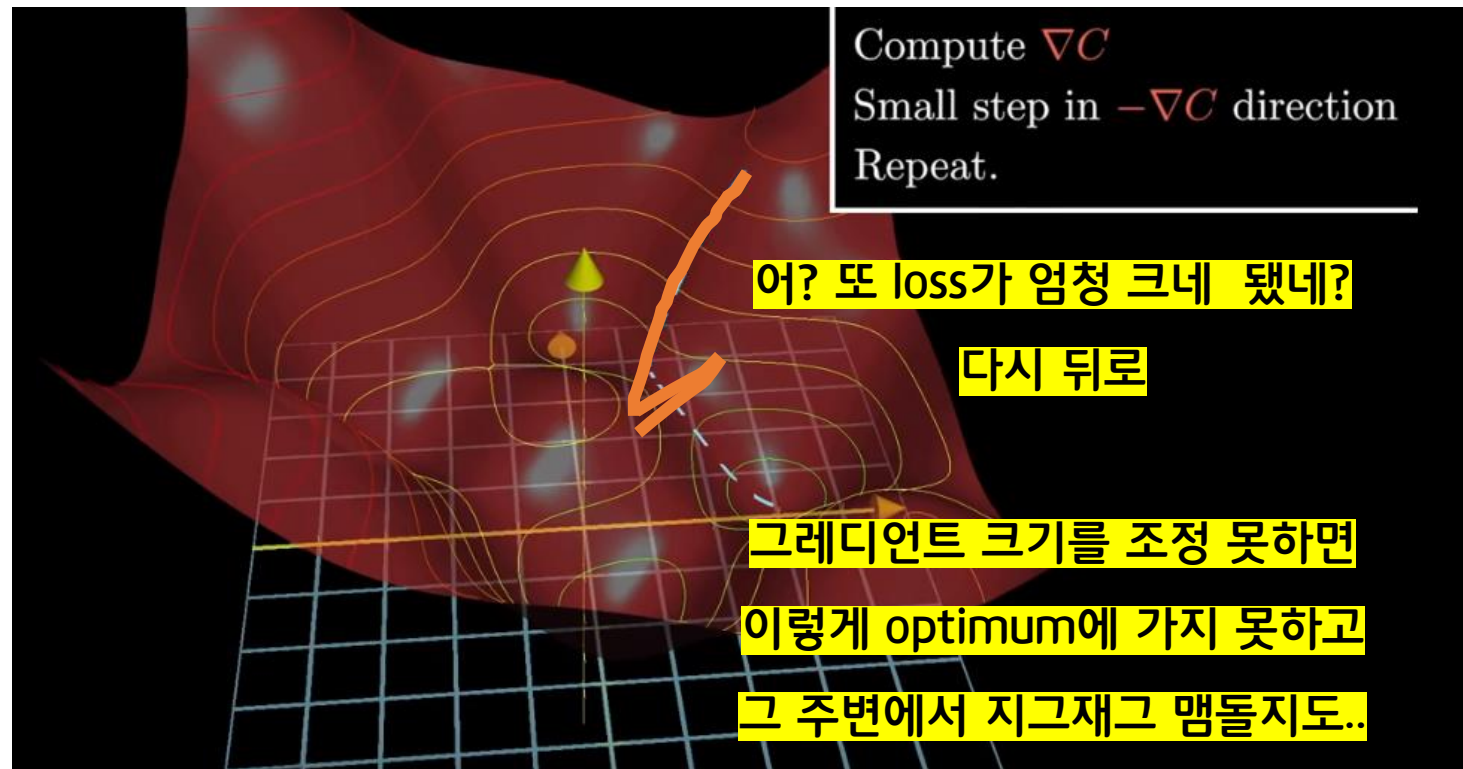


## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

그레디언트 크기에 따라 학습률(경사 한번 타는 폭)을 조정해야 되는 이유

Adagrad



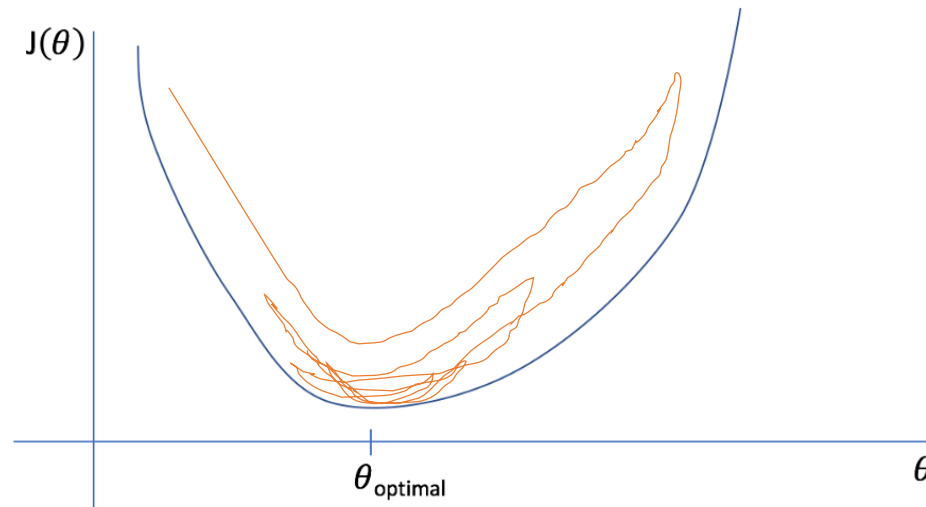
## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

그레디언트 크기에 따라 학습률(경사 한번 타는 폭)을 조정해야 되는 이유

Adagrad

정확한 설명은 아니겠지만 2차원으로 설명 가능



## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Adagrad

그레디언트를 누적시켜 제공

$$G_t = G_{t-1} + (\nabla_{\theta} J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

누적시킨 G를 루트 씌워 기존 learning rate를 나눠서

learning rate를 지금까지 경사가 얼마나 컸나에 따라 비율적으로 조정

뒤에 붙은 이상한 기호는 0으로 나누면 안되니까 미세한 값 더해주는 것

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Adagrad

그레디언트를 누적시켜 제공

$$G_t = G_{t-1} + (\nabla_{\theta} J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

그레디언트를 누적시켜서 제공하면

그레디언트의 방향은 없어지고 속도만 남는다

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Adagrad

그레디언트를 누적시켜 제공

$$G_t = G_{t-1} + (\nabla_{\theta} J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

그러나 너무 빨리 Learning rate(학습률)이 낮아져 (0에 가까워져)

어느순간 학습이 멈춰버리는 현상 발생

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

#### RMSProp

learning rate을 나눌 누적하는 값을 지수평균으로 바꾸어서 대체한 방법 너무 빨리 안줄어듬

$\gamma$  또한 하이퍼파라미터

$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$

$\theta = \theta - \frac{\eta}{\sqrt{G}}$  요 녀석은 옛날거를 어느정도 떨치게 해주는 녀석이다  
현재 값으로 수렴하는데 천천히 하게 해 줌

정확한 원리가 궁금하면

지수가중평균 Exponentially weighted averages을  
공부해보길 바람



## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

RMSProp

learning rate을 나눌 누적하는 값을 지수평균으로 바꾸어서 대체한 방법 너무 빨리 안줄어듬

$\gamma$  또한 하이퍼파라미터

$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$
$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

python 코드

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$
$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

**RMSProp**

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

**모멘텀(네스테로프)**

왜 속도와 방향 얘기를 한 줄 아시겠나요..ㅎ

**모멘텀은 gradient조정으로 방향조정하고**

**RMSProp은 gradient의 크기 조정으로 속도를 조정한다!**

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Adam



$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

RMSProp

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

모멘텀(네스테로프)

우와 둘 다 좋네 섞자

Adam 공식

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

Adam



$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

RMSProp

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

모멘텀(네스테로프)

Adam을 제일 많이 쓴다고 함

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

```

m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)

```

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_{\theta} J(\theta))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

\*책마다 자료마다 기호가 달라요...

## 2. DNN problem & solution

### 대규모 신경망에서는 훈련이 극단적으로 느려지는 문제

이 알고리즘은 많은 하이퍼파라미터가 있다

보통  $\beta_1$ (모멘텀 관련) 로는 0.9,  $\beta_2$ (RMSprop)관련로는 0.999,

$\epsilon$ (엡실론, 0으로 안 나누게 해주는 작은 값) 으로는  $10^{-8}$  정도의 값을 사용한다고 한다.

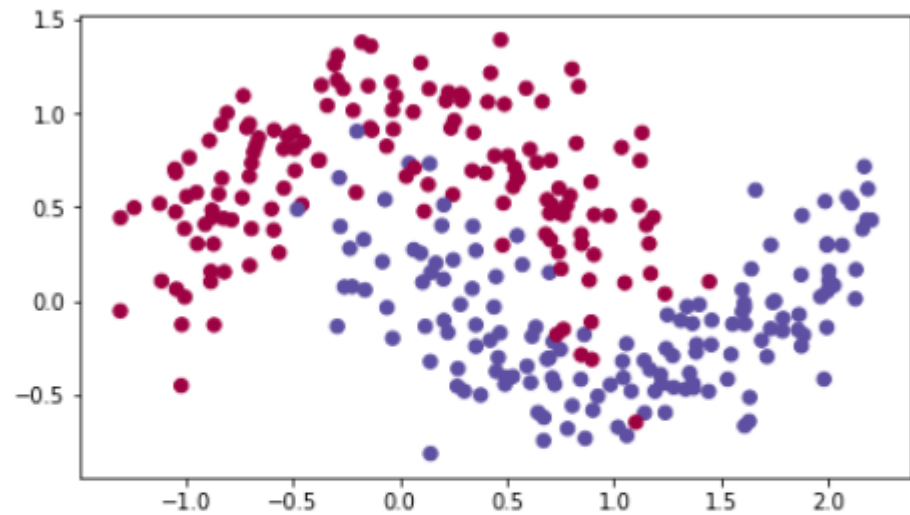
보통 다른 건 조정 잘 안하고 학습률만 조정한다고 합니다

Adam 공식

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta) \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} J(\theta))^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\ \theta &= \theta - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{aligned}$$

## 행렬로 표현해보았다

```
In [100]: train_X, train_Y = load_dataset()
```



```
In [117]: train_X.shape
```

```
Out[117]: (2, 300)
```

```
In [118]: train_Y.shape
```

```
Out[118]: (1, 300)
```

이해하기 위해 실제 데이터로 값을 내봄

2개의 클래스로 나뉘져 있는 데이터

# 행렬로 표현해보았다

두개의 피쳐, 300개 데이터

In [120]: train\_X

```
8.34584820e-01, 6.07553628e-01, 4.39755419e-01,
2.95626404e-01, 7.95245262e-01, -2.32597515e-01,
1.13026039e+00, -8.07312499e-02, 1.04985508e+00,
2.08741891e+00, -8.32856358e-01, -8.05161553e-01,
-5.22864195e-01, -1.34878701e-01, -9.81764880e-01,
1.36562815e+00, -1.99293557e-01, 7.79079038e-01,
-5.41884806e-01, -1.00265146e+00, -8.68221827e-01,
-9.44233966e-01, 1.96018850e-01, 6.97623239e-01,
-1.11938949e+00, 1.86277284e+00, 1.63238229e+00,
0.88518373e-01, 1.88873315e+00, 5.47725530e-01],
[ 0.01544937e+00, -5.57973395e-01, -4.10060241e-01,
 5.20814914e-01, 4.59349431e-01, 1.89195603e-01,
-1.38994928e-01, -1.22931655e-01, 3.56746183e-01,
-1.35143765e-01, 5.40099343e-01, 7.48716064e-01,
-2.73135578e-01, 1.00753602e+00, 8.97523249e-01,
6.00922628e-01, -2.84601480e-01, -3.91844343e-01,
6.56767552e-01, 6.94651275e-01, 1.06866038e+00,
-4.41381580e-01, -1.88074515e-01, 4.40047373e-01,
2.73709682e-01, -6.21054526e-01, 1.08911514e+00,
1.1800024e-01, -4.46300917e-01, 1.46216926e-01]
```

300개 레이블

train\_Y

```
array([[0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
        1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
        0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
        1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
        0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
        0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,
        0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
        1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1,
        0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]])
```

## 행렬로 표현해보았다

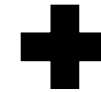
데이터 300개 제대로 안보여서 30개만 넣어봄(forward 과정)

[[ 1.78862847, 0.43650985],  
[ 0.09649747, -1.8634927 ],  
[-0.2773882, -0.35475898],  
[-0.08274148, -0.62700068],  
[-0.04381817, -0.47721803]]

$w_1(5,2)$

[[ -0.54036985 0.68047869 0.6347694 -0.01740858 2.10855562 -0.3799942  
0.34433628 1.22151229 0.11924657 -0.69975448 0.68789897 0.61767062  
0.71127451 0.90746636 0.28957238 1.17044851 0.33781812 -0.21687013  
0.98253281 0.06562705 -0.28196809 1.6114809 0.80505015 1.38239247  
0.06322389 -0.82185448 1.21620856 -0.4897675 1.54524651 0.67251321]  
[ 0.60092263 0.35674618 -0.13899493 1.08911514 0.52081491 0.74871606  
0.89752325 -0.44138158 -0.13514376 0.1891956 0.54009934 -0.18807451  
-0.41006024 0.45934943 1.06866038 0.14621693 0.69465128 1.01544937  
-0.44630992 0.27370968 0.65676755 -0.12293166 -0.5579734 -0.27313558  
1.00753602 0.44004737 -0.39184434 0.81180092 -0.28460148 -0.62105453]]

$X_1(2,30)$



[[0.,  
[0.,  
[0.,  
[0.,  
[0.]]

$b_1(5,1)$

1.78x 0.54 + 0.43 x 0.60

0.09 x 0.68 + 1.86 x 0.35

..

...

...

...

...

...

...

...

...

$w_1 X_1(5,30)$



# 행렬로 표현해보았다

데이터 300개 제대로 안보여서 30개만 넣어봄(forward 과정)

$$w1X1 + b1 = Z1$$

```
[[ -0.70421226, 1.37284679, 1.07469397, 0.444272, 3.99876346,
  -0.3528465, 1.00766741, 1.99216425, 0.15429622, -1.16901505,
  1.46615438, 1.02268688, 1.09321051, 1.82363071, 0.98441818,
  2.15732266, 0.90745324, 0.05535356, 1.56256748, 0.23685938,
  -0.21765064, 2.82867974, 1.19637474, 2.35336016, 0.55288344,
  -1.27790731, 2.00430134, -0.521653, 2.63964056, 0.93177986],
 [-1.17195925, -0.59912944, 0.32026967, -2.031238, -0.76706451,
  -1.4318954, -1.63930045, 0.9403842, 0.26334641, -0.42008916,
  -0.94009068, 0.41007914, 0.83278046, -0.76842611, -1.96349783,
  -0.15952886, -1.26187899, -1.91320992, 0.9265072, -0.50372315,
  -1.25109075, 0.38458607, 1.11746465, 0.64238353, -1.87143507,
  -0.89933195, 0.84756012, -1.56004642, 0.67946516, 1.2222264 ],
 [-0.06329048, -0.31531567, -0.12676785, -0.38154444, -0.76965222,
  -0.16020784, -0.41391925, -0.18224902, 0.01486587, 0.1269848,
  -0.38242015, -0.10461342, -0.05182661, -0.4146788, -0.45944083,
  -0.37654038, -0.34014054, -0.30008257, -0.11421056, -0.11530514,
  -0.15477957, -0.40339468, -0.02536534, -0.28656206, -0.37497001,
  0.07186198, -0.19835161, -0.15213794, -0.32766822, 0.03377744],
 [-0.33206789, -0.27998391, 0.03462815, -0.68143552, -0.50101632,
  -0.4380042, -0.59123858, 0.17567681, 0.07486859, -0.06072705,
  -0.39556043, 0.06681587, 0.19825614, -0.36309751, -0.69401043,
  -0.18852276, -0.46349839, -0.61874329, 0.1985404, -0.17704624,
  -0.38846324, -0.05625809, 0.28323865, 0.05687499, -0.636957,
  -0.20790854, 0.14505577, -0.46847564, 0.05058934, 0.33375687],
 [-0.2630931, -0.20006304, 0.03851645, -0.51898257, -0.34093531,
  -0.34065016, -0.44340246, 0.15711082, 0.05926787, -0.05962559,
  -0.28788762, 0.06268735, 0.16452139, -0.25897334, -0.52267254,
  -0.12106426, -0.34630269, -0.4750879, 0.16993435, -0.13349485,
  -0.30106599, -0.01194694, 0.23099914, 0.06977132, -0.48358471,
  -0.17398638, 0.13370315, -0.36594532, 0.06810709, 0.26691012]]
```

Z1 (5,300)



ReLU

```
[[0. , 1.37284679, 1.07469397, 0.444272, 3.99876346,
  0. , 1.00766741, 1.99216425, 0.15429622, 0. ,
  1.46615438, 1.02268688, 1.09321051, 1.82363071, 0.98441818,
  2.15732266, 0.90745324, 0.05535356, 1.56256748, 0.23685938,
  0. , 2.82867974, 1.19637474, 2.35336016, 0.55288344,
  0. , 2.00430134, 0. , 2.63964056, 0.93177986],
 [0. , 0. , 0.32026967, 0. , 0. ,
  0. , 0. , 0.9403842, 0.26334641, 0. ,
  0. , 0.41007914, 0.83278046, 0. , 0. ,
  0. , 0. , 0. , 0.9265072, 0. ,
  0. , 0.38458607, 1.11746465, 0.64238353, 0. ,
  0. , 0.84756012, 0. , 0.67946516, 1.2222264 ],
 [0. , 0. , 0. , 0. , 0. ,
  0. , 0. , 0. , 0.01486587, 0.1269848,
  0. , 0. , 0. , 0. , 0. ,
  0. , 0. , 0. , 0. , 0. ,
  0. , 0. , 0. , 0. , 0. ,
  0.07186198, 0. , 0. , 0. , 0.03377744],
 [0. , 0. , 0.03462815, 0. , 0. ,
  0. , 0. , 0.17567681, 0.07486859, 0. ,
  0. , 0.06681587, 0.19825614, 0. , 0. ,
  0. , 0. , 0. , 0.1985404, 0. ,
  0. , 0. , 0.28323865, 0.05687499, 0. ,
  0. , 0.14505577, 0. , 0.05058934, 0.33375687],
 [0. , 0. , 0.03851645, 0. , 0. ,
  0. , 0. , 0.15711082, 0.05926787, 0. ,
  0. , 0.06268735, 0.16452139, 0. , 0. ,
  0. , 0. , 0. , 0.16993435, 0. ,
  0. , 0. , 0.23099914, 0.06977132, 0. ,
  0. , 0.13370315, 0. , 0.06810709, 0.26691012]]
```

a1(상당수 0이 나온다. ReLU라서 그런 것!)

## 행렬로 표현해보았다

backward로 나온 gradient들 요걸로 업데이트 한당

```
[[[-0.015049 , 0.01433494],  
 [ 0.01013246, -0.00965168],  
 [ 0.02630837, -0.01634069],  
 [ 0.01958144, -0.01865232],  
 [ 0.00057308, -0.00054589]]]
```

dw1(5,2)

w1은 이값으로 업뎃

```
[[[-0.03675993],  
 [ 0.02475038],  
 [ 0.00336266],  
 [ 0.04783124],  
 [ 0.00139986]]]
```

db1(5,1)

b1은 이값으로 업뎃

요 숫자들이 일반적인 gradient decent를 한 것..!

```
[[ 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , -0.0174826 , 0.01572422,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 0.01602268, 0. , 0. , 0. , -0.01927733],  
 [ 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0.011771 , -0.01058708,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 -0.01078803, 0. , 0. , 0. , 0.01297938],  
 [ 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0.01172703, -0.01054754,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 -0.01074774, 0. , 0. , 0. , 0.0129309 ],  
 [ 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0.02274799, -0.02046002,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 -0.02084837, 0. , 0. , 0. , 0.02508325],  
 [ 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0.00066576, -0.0005988 ,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 0. , 0. , 0. , 0. , 0. ,  
 -0.00061016, 0. , 0. , 0. , 0.00073411]]])
```

da1 (5,300)

## 행렬로 표현해보았다

### 일반적인 gradient decent

➡  
18번째

```
[[ -0.0150322 , 0.01431915],
 [ 0.01010962, -0.00963007],
 [ 0.02629363, -0.01632692],
 [ 0.0195501 , -0.01862274],
 [ 0.00057054, -0.00054347]])
dw1(5,2)
```

➡  
30번째

```
[[ -0.01502195, 0.01430952],
 [ 0.0100957 , -0.0096169 ],
 [ 0.02628465, -0.01631852],
 [ 0.01953098, -0.0186047 ],
 [ 0.00056898, -0.000542  ]])
dw1(5,2)
```

### Adam적용

➡  
18번째

```
[[ -0.015049 , 0.01433494],
 [ 0.01013246, -0.00965168],
 [ 0.02630837, -0.01634069],
 [ 0.01958144, -0.01865232],
 [ 0.00057308, -0.00054589]])
dw1(5,2)
```

➡  
30번째

```
[[ -0.015049 , 0.01433494],
 [ 0.01013246, -0.00965168],
 [ 0.02630837, -0.01634069],
 [ 0.01958144, -0.01865232],
 [ 0.00057308, -0.00054589]])
dw1(5,2)
```

## 행렬로 표현해보았다

### 일반 gradient 배치사이즈 32의 첫번째 배치 gradinet

1에포크	15에포크	39에포크
$\begin{bmatrix} [-0.02258159, 0.0232553], \\ [0.01520414, -0.01565775], \\ [0.04971003, -0.0297037], \\ [0.04778765, -0.03138438], \\ [0.00139859, -0.00091852]] \end{bmatrix}$	$\begin{bmatrix} [-0.0217046, 0.02061243], \\ [0.0144665, -0.01373855], \\ [0.05611215, -0.02727066], \\ [0.04540655, -0.02708383], \\ [0.00129323, -0.00077138]] \end{bmatrix}$	$\begin{bmatrix} [-0.01099443, 0.01261449], \\ [0.01483643, -0.0073296], \\ [0.05430372, -0.02378154], \\ [0.02914784, -0.01439982], \\ [0.00079355, -0.00039204]] \end{bmatrix}$

### Adam적용 배치사이즈 32의 첫번째 배치 gradinet

1에포크	15에포크	39에포크
$\begin{bmatrix} [-0.02258159, 0.0232553], \\ [0.01520414, -0.01565775], \\ [0.04971003, -0.0297037], \\ [0.04778765, -0.03138438], \\ [0.00139859, -0.00091852]] \end{bmatrix}$	$\begin{bmatrix} [-0.02181807, 0.0207153], \\ [0.01469006, -0.01394757], \\ [0.04784046, -0.020674], \\ [0.04577618, -0.02732182], \\ [0.00133972, -0.00079962]] \end{bmatrix}$	$\begin{bmatrix} [-0.01113395, 0.01274288], \\ [0.01535387, -0.00760631], \\ [0.05497697, -0.02408019], \\ [0.02967206, -0.01469954], \\ [0.0008684, -0.00043021]] \end{bmatrix}$

## 행렬로 표현해보았다

### 일반 gradient 배치사이즈 32의 첫번째 배치 gradinet

1에포크	15에포크	39에포크
[[ -0.02258159, 0.0232553 ], [ 0.01520414, -0.01565775 ], [ 0.04971003, -0.0297037 ], [ 0.04778765, -0.03138438 ], [ 0.00139859, -0.00091852 ]]	[[ -0.0217046, 0.02061243 ], [ 0.0144665, -0.01373855 ], [ 0.05611215, -0.02727066 ], [ 0.04540655, -0.02708383 ], [ 0.00129323, -0.00077138 ]]	[[ -0.01099443, 0.01261449 ], [ 0.01483643, -0.0073296 ], [ 0.05430372, -0.02378154 ], [ 0.02914784, -0.01439982 ], [ 0.00079355, -0.00039204 ]]

w마다 gradinet의 크기와 방향을 잘 설정하여,  
Adam 적용 배치사이즈 32의 첫번째 배치 gradinet  
경사하강으로 optimum에 더 빠르고 정확하게 도달하게 해주는 것이 Adam

1에포크	15에포크	39에포크
[[ -0.02258159, 0.0232553 ], [ 0.01520414, -0.01565775 ], [ 0.04971003, -0.0297037 ], [ 0.04778765, -0.03138438 ], [ 0.00139859, -0.00091852 ]]	[[ -0.02181807, 0.0207153 ], [ 0.01469006, -0.01394757 ], [ 0.04784046, -0.020674 ], [ 0.04577618, -0.02732182 ], [ 0.00133972, -0.00079962 ]]	[[ -0.01113395, 0.01274288 ], [ 0.01535387, -0.00760631 ], [ 0.05497697, -0.02408019 ], [ 0.02967206, -0.01469954 ], [ 0.0008684, -0.00043021 ]]

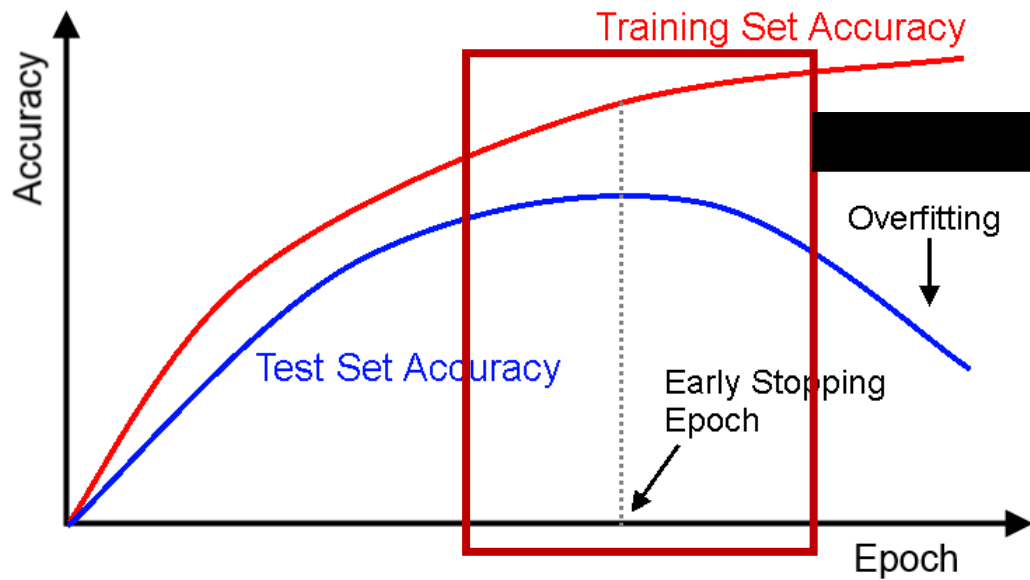
## 2. DNN problem & solution

3. 깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

하나 좋은 해결책은 조기종료(early stopping)

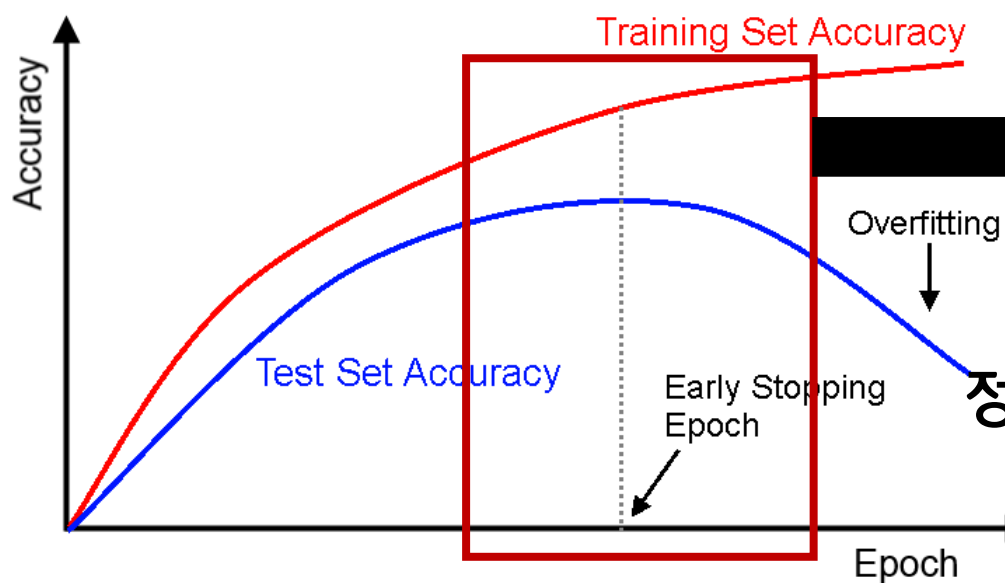


기준을 정해 놓고 모델을 일찍 멈춰 버림

## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

하나 좋은 해결책은 조기종료(early stopping)



기준을 정해 놓고 모델을 일찍 멈춰 버림

정말 빠른 학습으로 빠른 판단을 내릴 수 있다는 장점이  
있으나 optimization과 regularization을 섞어버려  
이 둘을 독립적으로 다룰 수 없다는 문제가 있음  
이 방법은 급할 때만 써야함



## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다  
overfitting을 막기 위한 regularization을 알아봅시다

정규화는 학습시 **특정 데이터가 최적의 parameter를 선택하는 문제에서 너무 큰 영향력을 미칠수 없도록 하는 일종의 장치**이다. 예를들면 LOSS함수에서 전체 데이터의 LOSS를 합한 후 training set의 수로 나누는 게 일반적인데 이때 특정 학습 데이터 예를들어 1000개중 5번째 데이터의 LOSS가 10이고 나머지를 합한 게 5라고 하면 해당 데이터 하나가 LOSS의 66%를 차지하게 된다는거다.  
특정 데이터가 영향을 너무 많이 미칠 수도 있음

## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

### L1, L2 규제

L2 regularization은 **큰 값이 많이 존재하는 가중치에 제약**을 주고, 가중치 값을 가능한 널리 퍼지도록 하는 효과를 주는 것으로 볼 수 있다. 선형 분류(Linear Classification) 장에서도 이야기 했던 가중치와 입력 데이터가 곱해지는 연산이므로 **특정 몇개의 입력 데이터에 강하게 적용되기 보다는 모든 입력데이터에 약하게 적용되도록** 하는 것이 일반적이다.

$$L2: \frac{\lambda}{2} \|\mathbf{w}\|^2 = \frac{\lambda}{2} \sum_{j=1}^m w_j^2$$

$$J(\mathbf{w}) = \sum_{i=1}^n \left[ -y^{(i)} \log(\phi(z^{(i)})) - (1 - y^{(i)}) \log(1 - \phi(z^{(i)})) \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

옆은 L2 정규화 비용함수 J 공식

## 2. DNN problem &amp; solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

## L1, L2 규제

L2 regularization은 **큰 값이 많이 존재하는 가중치에 제약**을 주고, 가중치 값을 가능한 널리 퍼지도록 하는 효과를 주는 것으로 볼 수 있다. 선형 분류(Linear Classification) 장에서도 이야기 했던 가중치와 입력 데이터가 곱해지는 연산이므로 **특정 몇개의 입력 데이터에 강하게 적용되기 보다는 모든 입력데이터에 약하게 적용되도록** 하는 것이 일반적이다.

cf) L1정규화는 w를 좀 더 0에 가까운 값을 많이 만들어 모델을 압축하는 건데 보통 L2를 더 많이 씀

$$L2: \frac{\lambda}{2} \|\mathbf{w}\|^2 = \frac{\lambda}{2} \sum_{j=1}^m w_j^2$$

비용함수 J에 w의 제곱을 더해주면 w가 너무 커지면 커진만큼 비용함수가 커져서 w업데이트를 줄여주는 효과가 있어용!!

$$J(\mathbf{w}) = \sum_{i=1}^n \left[ -y^{(i)} \log(\phi(z^{(i)})) - (1 - y^{(i)}) \log(1 - \phi(z^{(i)})) \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

옆은 L2 정규화 비용함수 J 공식

w1와 w2가 있다고 하면

w1와 w2를 적절히 줄임

## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

### Dropout

dropout은 간단하지만 아주 효과적인 regularization 방법으로  
위에서 소개한 다른 regularization 기법들과 (L1, L2, maxnorm)  
상호 보완적인 방법으로 알려져 있다.

각 뉴런들을  $p$  의 확률로 활성화 시켜 학습에 적용 하는 방식으로 구현할 수 있다.

## 2. DNN problem & solution

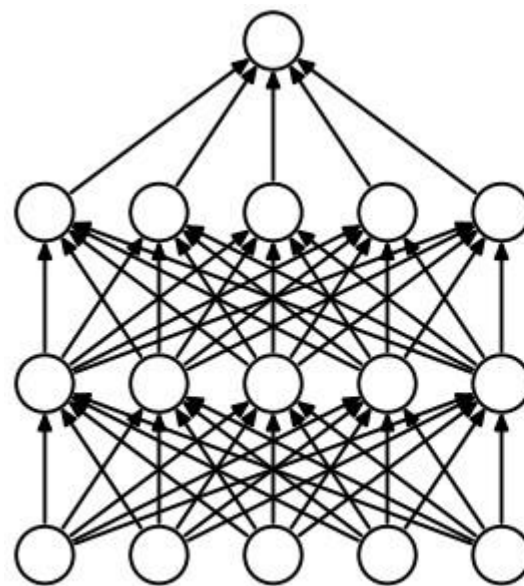
깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

### Dropout

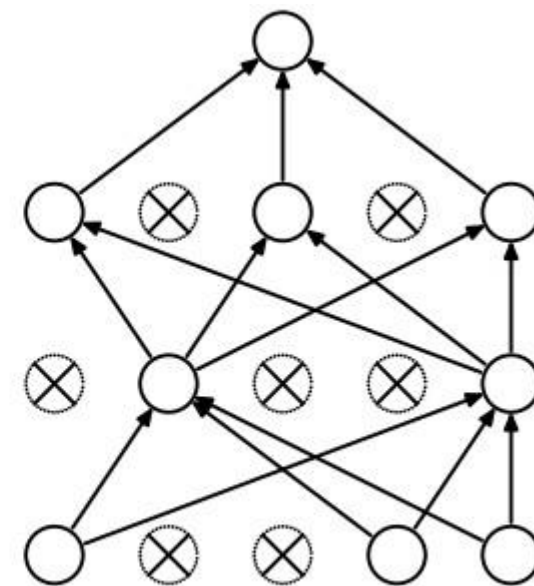
**강제로 학습할 때마다 랜덤으로 노드  
몇 개를 꺼버리면 어떨까?**

굉장히 재미있는 아이디어면서 간단한  
방법으로 overfitting을 막는데  
효과적으로 알려져있다  
(노드가 꺼질 확률을 부여해 dropout을 한다  
보통 0.5의 확률)

\*주의 test시에는 dropout적용 안함  
(랜덤으로 test가 되면 안되기 때문)



(a) Standard Neural Net



(b) After applying dropout.

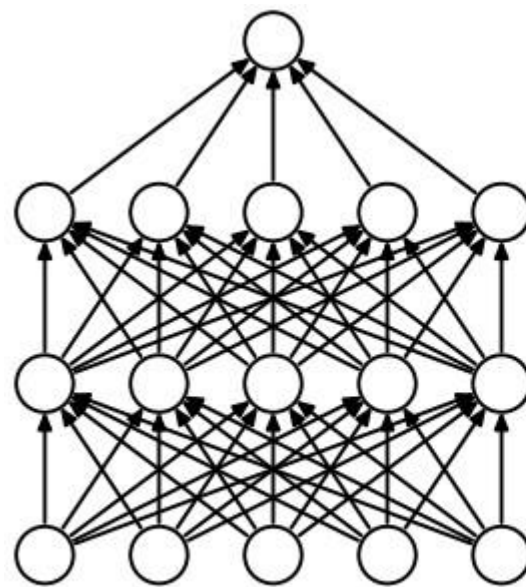
## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

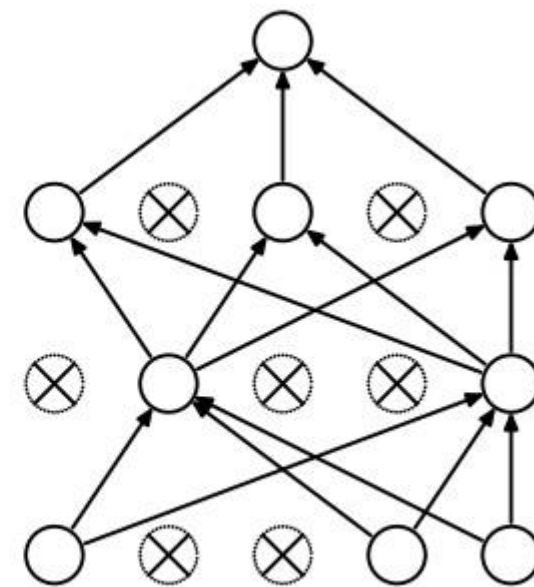
### Dropout

이게 왜 잘 돼지? 하는 데에는 많은 연구가 있는데 기본적으로 **특정  $w$ 에 의존하지 않아 overfitting 가능성을 낮춘다는 것이다**

또한 노드를 몇 개 꺼버리면 다른 모델이 되어 **앙상블 효과**도 있으며,  
노드를 꺼버리면 한 노드가 하나의 특징만 훈련하지 않고 **여러 특징을 중첩해서 특징을 훈련해서 더 잘된다는 말도 있다**



(a) Standard Neural Net



(b) After applying dropout.

## 2. DNN problem & solution

깊은 모델은 훈련 세트에 과대적합(overfitting)이 일어날 것이다

### Dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
```

### 3. DNN Training Tip

## 3. DNN Training Tip



### 3. DNN Training Tip

#### 1. 문제에 빠졌는지 판단하기

train시키고 있는 당신의 network가 문제에 빠졌는지 어떻게 판단하는가?

- 1) 첫번째로 loss를 확인하라. 이것은 네트워크가 어디에 근거하여 최적화 할 것인가를 정하는 과정으로서, 우리는 모델이 학습하는 동안 손실 값을 보다가 손실 값이 내려가지 않으면 문제에 빠졌다는 것을 알아차려야 한다
- 2) 두번째로 accuracy를 살펴보아라. 손실 값이 감소하면 정확도는 올라가야 하지만, 정확도가 올라가지 않는다면, 이는 우리의 네트워크가 무언가를 학습하고 있긴 한데, 우리가 원하는 것을 학습하고 있지 않다는 것이다.
- 3) 세번째로 바로 overfitting이다. 가장 흔한 문제 중 하나인데, train accuracy와 test accuracy가 함께 증가하다가, test정확도가 정체가다 떨어지면 네트워크가 일반적인 것을 학습하기 보다는 입력데이터와 출력데이터를 직접 매핑하고 있을 수 있다.

### 3. DNN Training Tip

## 2. 런타임 오류 해결하기

**train하려고 하는 network와 호환되지 않는 data를 만날 어떻게 해야하나**

오류메시지 예시

`InvalidArgumentError : Incompatible shapes : X vs. Y`

1) 어떤 데이터라도 X이거나 Y의 형태인지 찾아보라 그거이 주요 오류. 또한 레이어들의 이름도 주의깊게 보아야 한다. 레이어의 이름이 가끔은 원래와 달리 지저분한 형태로 오류 메시지에 등장하기도 한다. Keras는 이름 없는 레이어에 자동적으로 이름을 부여하기 때문에, 이런 면에서 모델에 대한 요약 정보(summary)를 들여다보는 것이 도움이 된다. 필요하다면 직접 레이어에 이름을 붙이는 것도 가능하다.

### 3. DNN Training Tip

## 2. 런타임 오류 해결하기

**train하려고 하는 network와 호환되지 않는 data를 만날 어떻게 해야하나**

오류메시지 예시

`InvalidArgumentError : Incompatible shapes : X vs. Y`

2) 다양한 하이퍼파라미터들의 숫자들을 살펴보아라.

데이터가 배치의 크기로 나누어질 수 없는 크기일수도?

### 3. DNN Training Tip

## 2. 중간 결괏값을 확인하기

**network의 정확도가 처음에는 빨리 증가하다가 어느 수준을 지나면 증가를 멈춘다.**

- 1) 레이블이 불균형 할 경우 생길 수 있다. 데이터의 분포를 바꿔본다. 데이터의 95%가 개이고 5%개 고양이라면 65%/35%로 바꿔본다.
- 2) loss가 NaN이 보인다면 그래디언트가 급격하게 증가한다는 것을 의미하고, 만약 네트워크의 결과 값이 어딘가에서 잘려서 옳은 값을 얻지 못하는 것 같아 보이는 경우, 마지막 레이어의 활성화 함수를 잘못 골랐을 지도 모른다.

### 3. DNN Training Tip

#### 3. 옳은 (마지막 레이어) 활성화 함수 고르기

상황이 좋지 않은 경우, 마지막 레이어의 활성화 함수를 어떻게 고를 것인가?

- 1) softmax 활성화 함수는 출력 벡터의 합이 정확히 1이 되도록 한다. 이 함수는 네트워크가 정확히 하나의 레이블을 결정해야 하는 경우에 적합하다. 출력 벡터는 확률 분포를 보여준다. 출력 벡터에 '고양이'가 65%라면 그것은 네트워크가 이미지가 고양이라는 것을 65%라는 것을 확신하는 덕이다. 여러 개 답이 가능한 경우는 sigmoid를 사용해보라.

### 3. DNN Training Tip

#### 3. 옳은 (마지막 레이어) 활성화 함수 고르기

상황이 좋지 않은 경우, 마지막 레이어의 활성화 함수를 어떻게 고를 것인가?

- 1) 선형 활성화 함수는 입력 데이터에 숫자를 예측해야 하는 회귀 문제에 적합하다. 가끔 정규화가 필요할 수도 있다. 영화 점수가 0에서 5점 사이라면 훈련 데이터를 만들 때 2.5를 빼고 (평균) 2.5로 나누어 (표준편차) 정규화된 출력값을 준비할 수 있다.
- 2) 출력값이 이미지라면, 당신이 고른 활성 함수가 당신이 픽셀을 정규화 한 방법과 일맥상통하도록 해야 한다.

### 3. DNN Training Tip

#### 4. 정규화와 드롭아웃

**network가 과적합되었다는 것을 알고 나면 무엇을 해야 하는가**

- 1) 과적합을 막는 가장 쉽고 당연한 방법은 레이어의 수를 줄이거나 각 레이어를 작게 만들어서 전체 파라미터 수를 줄이는 것이다. 하지만 이 방법은 당연히 네트워크의 표현 능력을 줄인다.
- 2) 그래서 정규화와 드롭아웃을 통해, 학습 능력에는 큰 피해를 주지 않으면서 네트워크의 표현 능력을 통제하여 과적합을 적당히 조절할 수 있다.

### 3. DNN Training Tip

#### 5. 네트워크 구조, 배치 크기, 학습 속도

어떻게 하면 주어진 문제에 대한 최고의 네트워크 구조, 배치 크기, 학습 속도를 찾을 수 있는가

- 1) 작은 것에서부터 시작한다. 적합한 네트워크 종류를 찾은 후에도 네트워크 구조, 학습 속도, 배치 크기를 정하는 것은 중요하다. 네트워크 구조를 본다면, 몇 개의 레이어를 가져야 하나를 고민할 때 가장 좋은 전략은 작은 크기부터 시작하는 것이다. deep한 성격에 집중하다 보면 더욱 많은 레이어로부터 시작하고 싶은 유혹이 든다. 하지만 실제로 하나나 두 개의 레이어를 가진 네트워크가 돌아가지 않는다면 더 많은 레이어를 추가하는 것이 큰 도움이 되지 않을 가능성이 크다.



### 3. DNN Training Tip

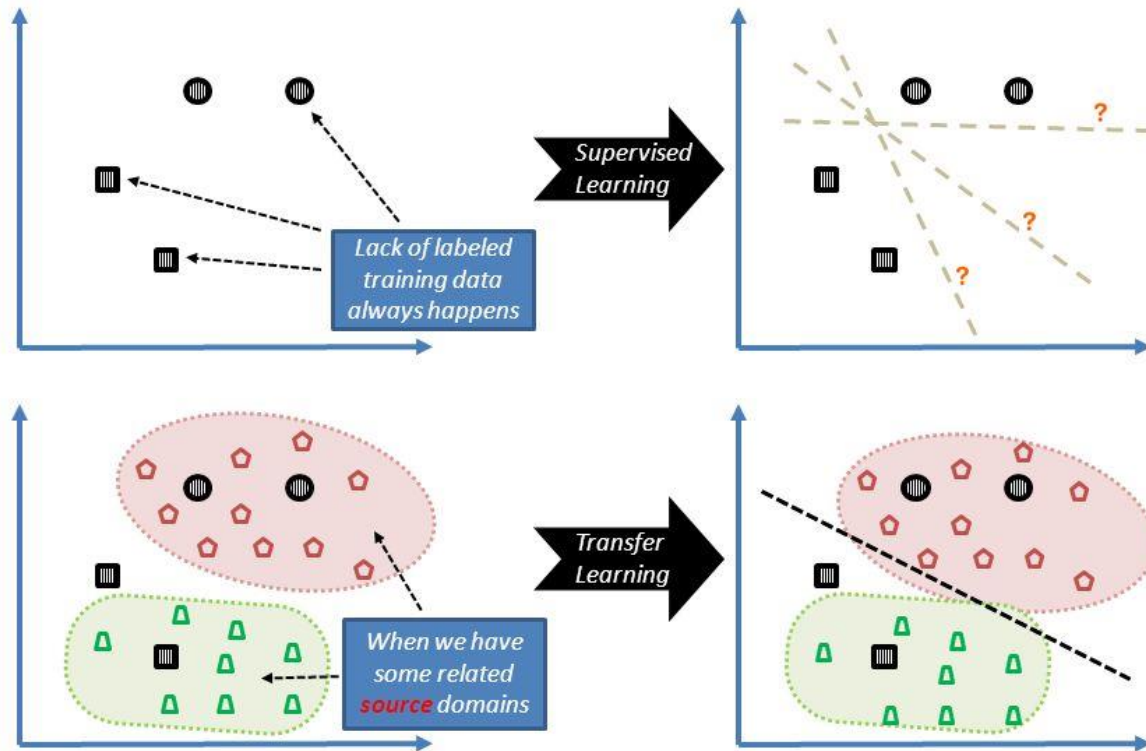
#### 5. 네트워크 구조, 배치 크기, 학습 속도

**어떻게 하면 주어진 문제에 대한 최고의 네트워크 구조, 배치 크기, 학습 속도를 찾을 수 있는가**

2) 배치는 클수록 더 많은 시간이 걸리기는 하지만 그래디언트가 더 정확해진다. 결과를 보다 빨리 얻기 위해서 작은 크기의 배치 (32가 적당하다)부터 시작하는 것이 좋다. 학습 속도는 추론된 그래디언트 방향에 맞게 우리가 얼마나 많은 네트워크의 가중치를 조정하는가를 의미한다. 학습 속도가 클수록 우리는 값들 사이를 보다 빨리 움직이고 때로는 좋은 값을 지나칠 수 있다. 작은 크기의 배치가 덜 정확한 그래디언트를 일으킨다는 점을 고려하면, 우리는 작은 크기의 배치와 작은 속도를 함께 섞어야 한다. 따라서 여기서 추천하는 것은 작은 배치에서부터 시작하여 큰 배치와 높은 학습 속도를 실험한다

## 4. Transfer Learning & model save

### Transfer Learning



HKUST - IJCAI 2011

2

기존의 만들어진 모델을 사용하여 새로운 모델을 만들  
시 학습을 빠르게 하며, 예측을 더 높이는 방법입니다.

이미 잘 훈련된 모델이 있고, 특히 **해당 모델과 유사  
한 문제를 해결시** transfer learning을 사용

## 4. Transfer Learning & model save

### 실질적 조언

**새로 훈련할 데이터가 적지만 original 데이터와 유사할 경우**

데이터의 양이 적어 fine-tune (전체 모델에 대해서 backpropagation을 진행하는 것) 은 over-fitting의 위험이 있기에 하지 않습니다.

새로 학습할 데이터는 original 데이터와 유사하기 때문에 이 경우 최종 linear classifier 레이어만 학습을 합니다.

**새로 훈련할 데이터가 매우 많으며 original 데이터와 유사할 경우**

새로 학습할 데이터의 양이 많다는 것은 over-fitting의 위험이 낮다는 뜻이므로, 전체 레이어에 대해서 fine-tune을 합니다.

## 4. Transfer Learning & model save

### 실질적 조언

**새로 훈련할 데이터가 적으며 original 데이터와 다른 경우**

데이터의 양이 적기 때문에 최종 단계의 linear classifier 레이어를 학습하는 것이 좋을 것입니다. 반면서 데이터가 서로 다르기 때문에 거의 마지막부분 (the top of the network)만 학습하는 것은 좋지 않습니다. 서로 상충이 되는데.. 이 경우에는 네트워크 초기 부분 어딘가 activation 이후에 특정 레이어를 학습시키는게 좋습니다.

**새로 훈련할 데이터가 많지만 original 데이터와 다른 경우**

데이터가 많기 때문에 아예 새로운 ConvNet을 만들수도 있지만, 실적으로 transfer learning이 더 효율이 좋습니다. 전체 네트워크에 대해서 fine-tune을 해도 됩니다.

코드

코드

## 코드

```
class CNN(nn.Module):
    def __init__(self, num_classes=10):
        super(CNN, self).__init__()
        self.block1 = nn.Sequential(
            # kernel_size = (5X5)
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=16, momentum=0.1),
            nn.LeakyReLU(),
            nn.Conv2d(in_channels=16, out_channels=16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=16, momentum=0.1),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.block2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=32, momentum=0.1),
            nn.LeakyReLU(),
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(num_features=32, momentum=0.1),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(8*8*32, num_classes)

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = x.view(x.size(0), -1)
        output = self.fc(x)
    return output
```

가장 좋은 예시가 여기에

[http://www.datamarket.kr/xe/index.php?mid=board\\_BoGi29&document\\_srl=49335&sort\\_index=title&order\\_type=desc](http://www.datamarket.kr/xe/index.php?mid=board_BoGi29&document_srl=49335&sort_index=title&order_type=desc)

## 코드

```
class ResNet(nn.Module):  
    def __init__(self, block, num_blocks, num_classes=10):  
        super(ResNet, self).__init__()   
        self.in_channel = 64
```

```
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1, bias=False)  
        self.bn1 = nn.BatchNorm2d(num_features=64)  
        self.layer1 = self.MakeLayer(block=block, out_channel=64, num_blocks=num_blocks[0], stride=1)  
        self.layer2 = self.MakeLayer(block=block, out_channel=128, num_blocks=num_blocks[1], stride=2)  
        self.layer3 = self.MakeLayer(block=block, out_channel=256, num_blocks=num_blocks[2], stride=2)  
        self.layer4 = self.MakeLayer(block=block, out_channel=512, num_blocks=num_blocks[3], stride=2)  
        self.linear = nn.Linear(512, num_classes)
```

```
    def MakeLayer(self, block, out_channel, num_blocks, stride):  
        # Convblock으로 형성한 Layer들을 nn.Sequential을 이용하여 구성한다.  
        # stride가 1일 경우 아래 strides = [1, 1]  
        strides = [stride] + [1]*(num_blocks-1)  
        layers = []  
  
        # Block 형성이 끝난 이후에는 채널 확장을 위해 self.in_channel = out_channel로 확장된 채널 수로 업데이트를 해준다.  
        for stride in strides:  
            layers.append(block(self.in_channel, out_channel, stride))  
        self.in_channel = out_channel  
        return nn.Sequential(*layers)
```

파이토치의 torch.nn.Module을 상속받는다

super의 경우 무조건 써줘야함(그 모델의 적용되어있는 init값을 모두 받겠다는 의미)

#부모 class로부터 상속받은 class는 처음 initialize 해줄 때 부모의 init을 해 주어야 한다.

batchnorm구현이 끼어있음

모델을 정의한다

레이어를 만드는 부분인데 CNN이해 후 다시 보자

## 코드

```
def forward(self, x):
```

```
    out = F.relu(self.bn1(self.conv1(x)))
```

relu(activation function)이전에 batchnorm을 적용하는 것을 볼 수 있따

```
    out = self.layer1(out)
```

```
    out = self.layer2(out)
```

```
    out = self.layer3(out)
```

```
    out = self.layer4(out)
```

```
    out = F.avg_pool2d(out, 4)
```

```
    out = out.view(out.size(0), -1)
```

```
    out = self.linear(out)
```

```
    out = out.cuda()
```

```
return out
```

forward(순전파) 과정을 수행하는 함수

```
learning_rate = 0.01
```

learning late 적용

```
# 0.001로 낮춤
```

```
model = ResNet(block=ConvBlock, num_blocks=[2,2,2,2], num_classes=10)
```

변수에 모델을 넣음

```
criterion = nn.CrossEntropyLoss()
```

loss function 정의

```
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

optimization 적용(Adam)



## 코드

```
#4 Train the network
def train(epochs=1):
    for epoch in range(epochs):
```

원하는 epoch만큼 반복하는 for문

```
        running_loss = 0.0
```

```
        for i, data in enumerate(trainloader, 0):
```

```
        # get the inputs
```

```
            inputs, labels = data
```

```
            inputs, labels = inputs.to(device), labels.to(device)
```

train 시키는 function

```
        # zero the parameter gradients
```

```
            optimizer.zero_grad()
```

Adam 누적된 gradinet를 초기화한다

```
        # forward + backward + optimize
```

```
            outputs = model(inputs)
```

model에 데이터를 넣고 예측값을 뱉는다

```
            loss = criterion(outputs, labels)
```

예측값과 실제값을 비교한다

```
            loss.backward()
```

backpropagation한다

```
            optimizer.step()
```

step()이란 함수를 실행시키면 우리가 미리 선언할 때 지정해 준 model의 파라미터들이 업데이트 된다

## 코드

```
# print statistics
# item(): retrieve a single value
running_loss += loss.item()

    print_op = 500

if (i + 1) % print_op == 0:    # print every `print_op` mini-batches
    print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss / print_op))
    running_loss = 0.0

print('Finished Training')
```

loss를 효율적으로 뽑는 것,  
loss는 개수로 나눠줘야함

```
Train(epochs=15)
# learning_rate = 0.01로 Epoch 15까지
# learning_rate = 0.001로 Epoch 5 추가
```

train

## 코드

```
#5 Test
correct = 0
total = 0

class_correct = list(0.0 for i in range(10))
class_total = list(0.0 for i in range(10))

with torch.no_grad():
    for data in test_loader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        c = (predicted == labels).squeeze()

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    for i in range(4):
        label = labels[i]
        class_correct[label] += c[i].item()
        class_total[label] += 1

    for i in range(10):
        print('Accuracy of %5s: %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i]))

    print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / total))
```

정확율을 넣을 list를 미리 정의해둠

gradient 추적 방지

model에 test값을 넣어 예측해봄

확률 제일 높은 것 튀어나와랏

예측값이랑 맞는 것 계산

정확율 계산

test 결과 프린트

## 파이토치 참조자료

<https://pytorch.org/docs/stable/index.html>  
(파이토치 공식문서)

[https://tutorials.pytorch.kr/beginner/deep\\_learning\\_60min\\_blitz.html](https://tutorials.pytorch.kr/beginner/deep_learning_60min_blitz.html)  
(파이토치 튜토리얼 모음집)

<https://wingnim.tistory.com/entry/Pytorch-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-%ED%8A%9C%ED%86%A0%EB%A6%AC%EC%96%BC-%EA%B0%95%EC%9D%98-0-Overview>  
(김성훈교수님 파이토치 영어강의 정리 블로그)

[https://www.youtube.com/channel/UCK24Wy\\_G-6V-quKvVRjflgA/videos](https://www.youtube.com/channel/UCK24Wy_G-6V-quKvVRjflgA/videos)  
(김군이 파이토치 튜토리얼 강의 – 유튜브)

<https://greeksharifa.github.io/pytorch/2018/11/10/pytorch-usage-03-How-to-Use-PyTorch/>  
(how to pytorch)

## Reference

### book

1. 핸즈온 머신러닝 (한빛미디어)
2. Deep Learning cookbook (느린생각)

### Lecture

1. Andrew Ng's Improving deep neural networks  
(<https://www.youtube.com/playlist?list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc>)
  2. 3Blue1Brown's Neural networks  
([https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1\\_67000Dx\\_ZCJB-3pi](https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi))
  3. 딥러닝기반영상분석 (cs231n)  
(<https://www.youtube.com/playlist?list=PL1Kb3QTCLIVty0uMgyVgT-OeW0PYXI3j5>)
  4. 2019겨울 한동머신러닝캠프 강의  
(<https://www.youtube.com/playlist?list=PLhicYQInZBfXID1EsUb53pKD5p9jMWLeZ>)
- 강의노트 : [https://github.com/callee2006/2019-Winter-HGU-Machine-Learning-Camp/blob/master/2019\\_HGU\\_ML\\_Camp\\_LectureNote.pdf](https://github.com/callee2006/2019-Winter-HGU-Machine-Learning-Camp/blob/master/2019_HGU_ML_Camp_LectureNote.pdf)

## Reference

### Link

1. CS231n Convolutional Neural Networks for Visual Recognition 강의노트  
(<http://aikorea.org/cs231n/>)
2. jujbob  
(<http://blog.naver.com/jujbob/221120204646>)
3. 라온피플  
(<https://laonple.blog.me/221366130381>)
4. batch normalization 관련 글  
(<https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%ec%84%a4%eb%aa%85-%eb%b0%8f-%ea%b5%ac%ed%98%84/>)
5. Gradient Descent Optimization Algorithms 정리  
(<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>)
6. ratsgo'blog  
(<https://ratsgo.github.io/blog/categories/>)
7. 훈철오빠 강의자료 캡처 사진
8. Transfer Learning in Pytorch  
(<http://incredible.ai/artificial-intelligence/2017/05/13/Transfer-Learning/>)

Q & A

들어주셔서 감사합니다.