Copyright Notice

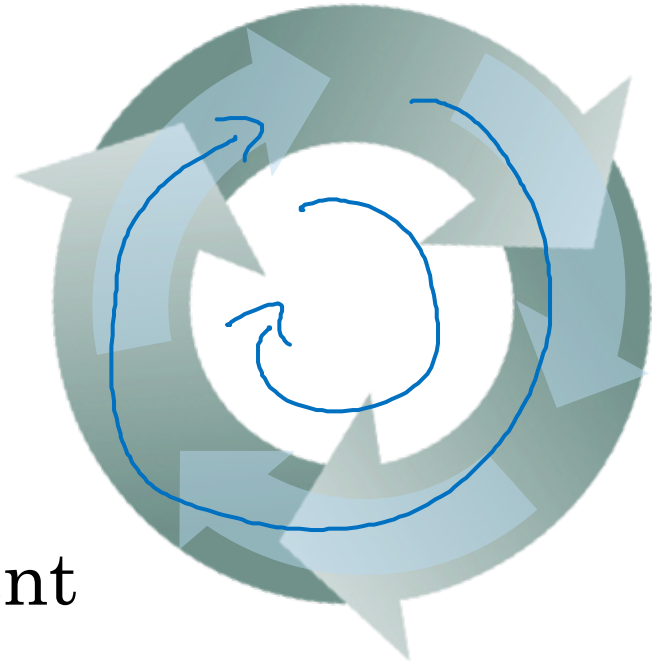
deeplearning.ai

Setting up your ML application

Train/dev/test sets

Applied ML is a highly iterative process

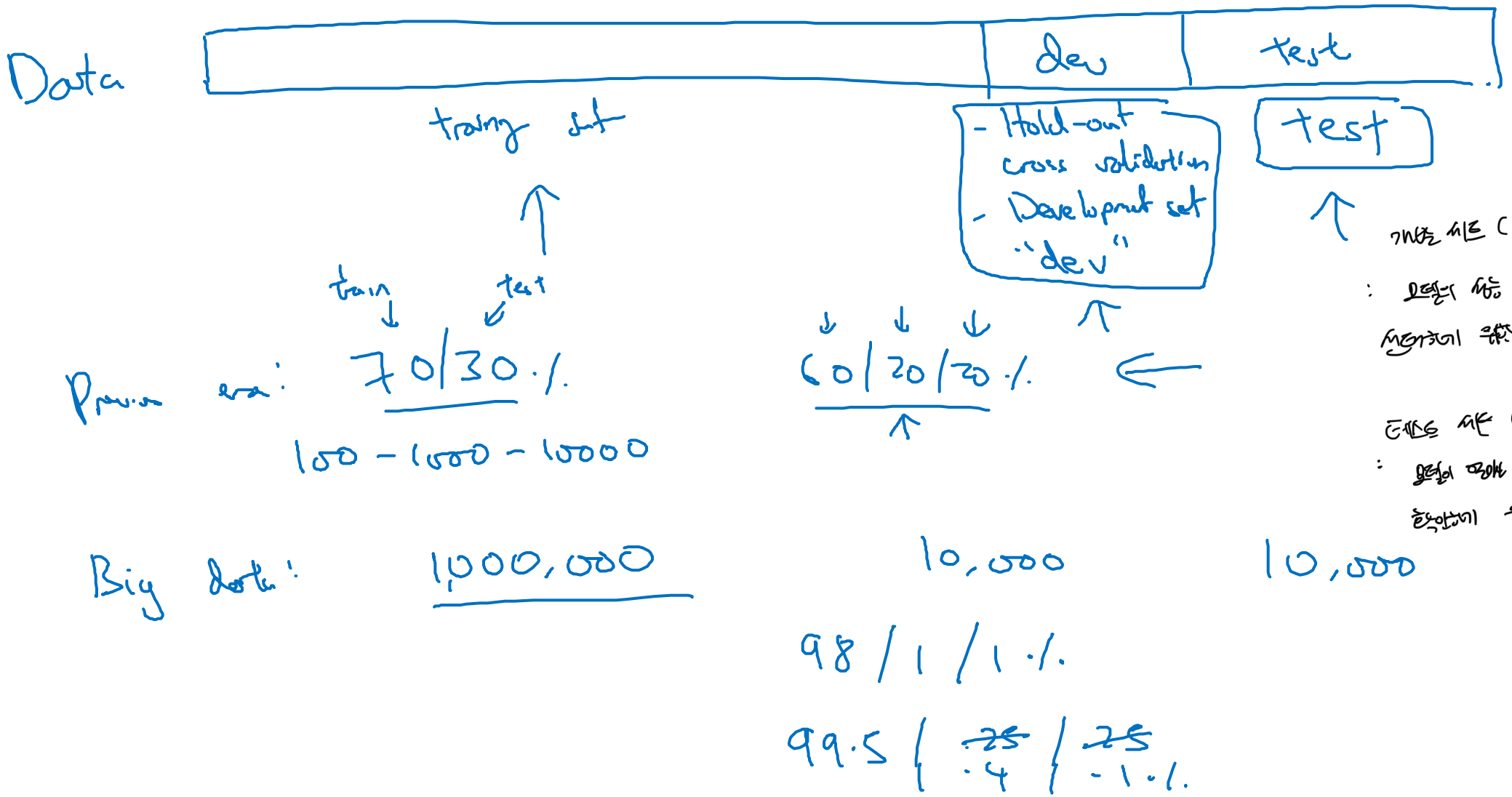
- # layers
- # hidden units
- learning rates
- activation functions
- ...



NLP, Vision, Speech, Structured data

Ads Search Security Logistic ....

Train/dev/test sets



Andrew Ng

Mismatched train/test distribution

Cats

Training set:
Cat pictures from webpages

Dev/test sets:
Cat pictures from users using your app

→ Make sure dev and test come from same distribution.

"test"

train / dev

train / test

→ train / dev

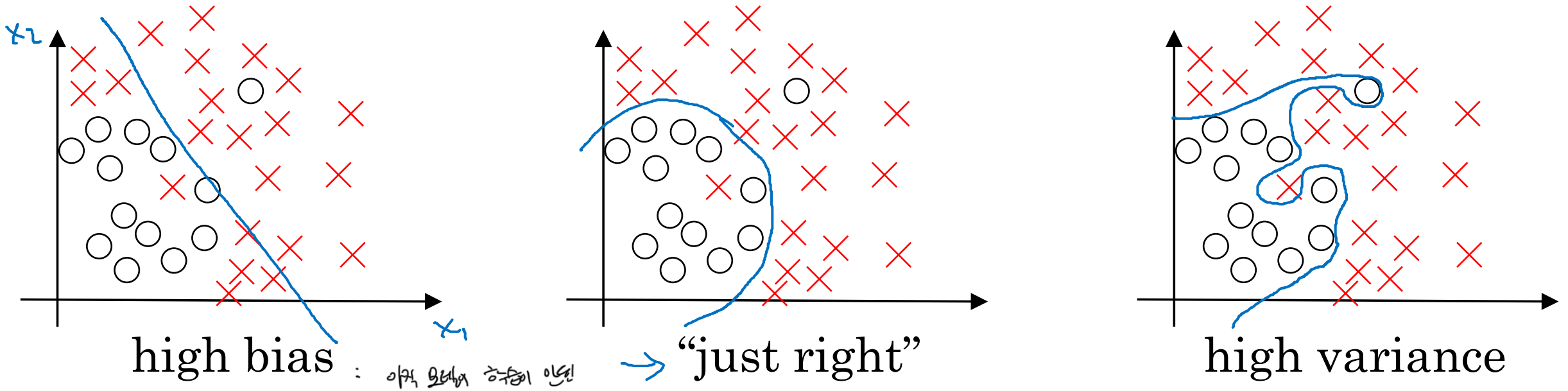
Not having a test set might be okay. (Only dev set.)

deeplearning.ai

Setting up your ML application

Bias/Variance

Bias and Variance



high bias : 이전 모델에 학습이 안된 과소 적합된 상태

→ "just right"

high variance : 모델이 너무 학습된 과적합 상태

Underfitting

Overfitting

Bias: 예측값과 정답의 차이

Variance: 예측값끼리의 차이

Bias and Variance

Cat classification

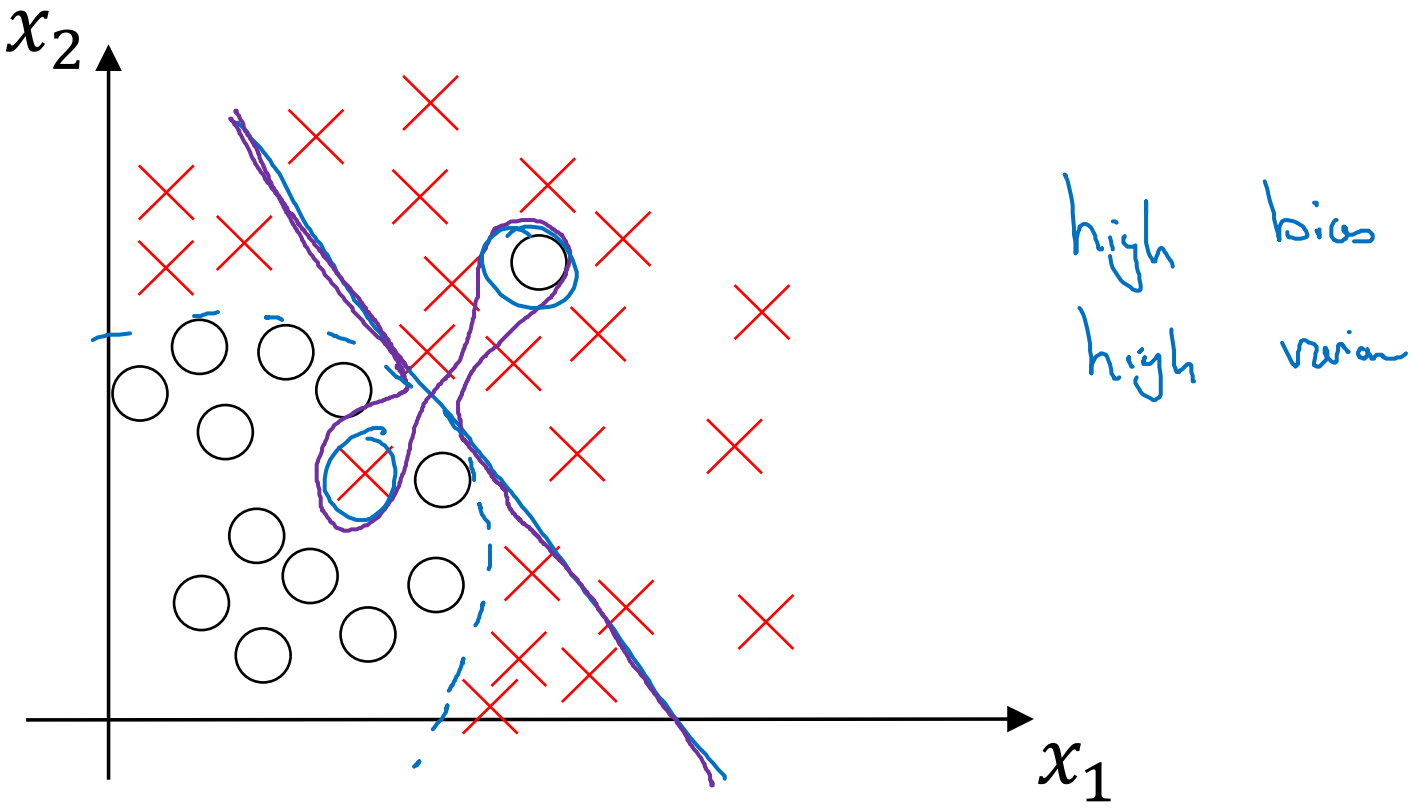
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human: ≈0%

Optimal (Bayes) error: ~~≈0%~~ 15%

Blurry images

High bias and high variance

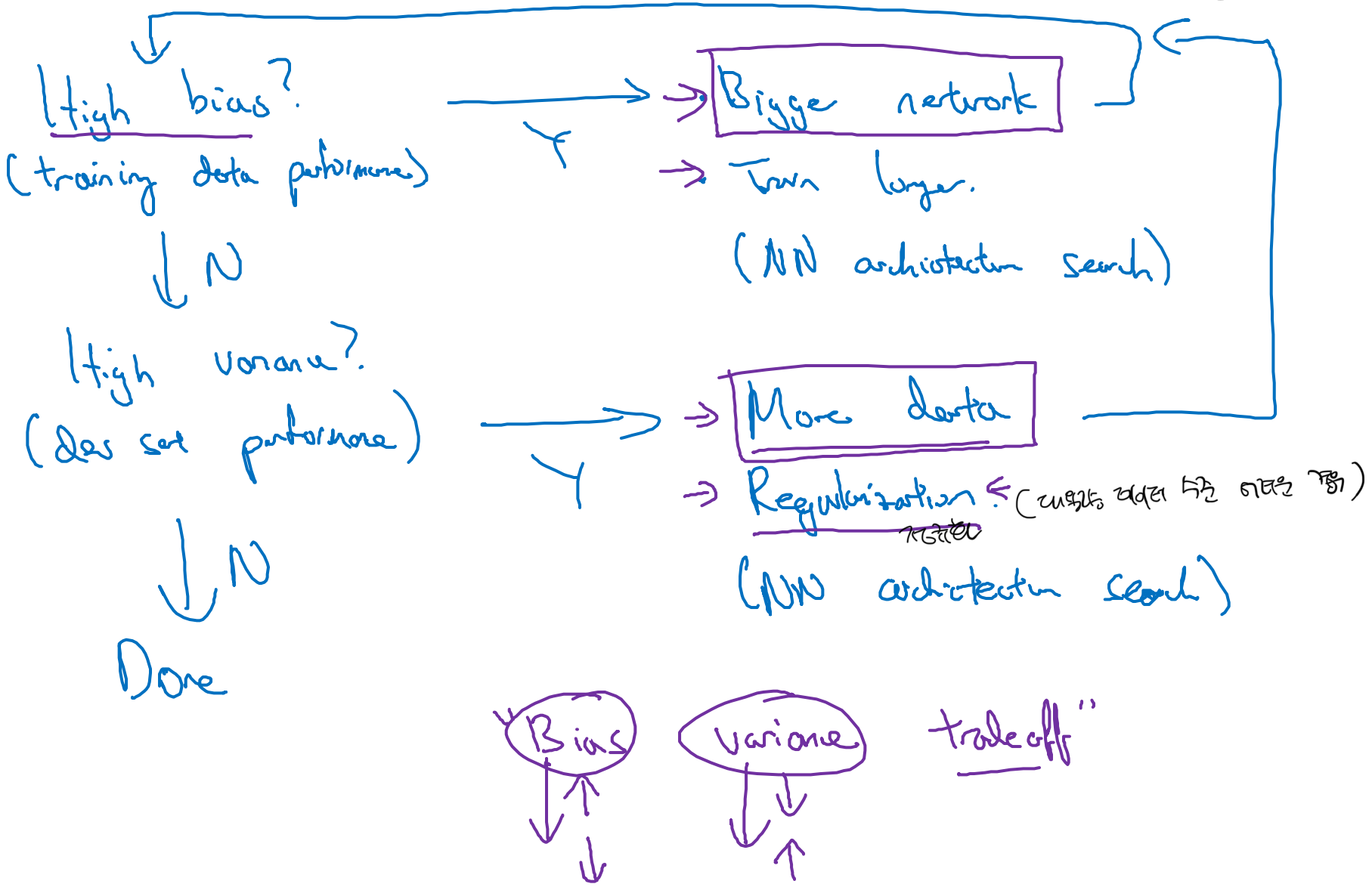


deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning



Andrew Ng

deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

$w \in \mathbb{R}^{n_x}, \quad b \in \mathbb{R}$

λ = regularization parameter

lambda lambd

$$\min_{w,b} J(w,b)$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \frac{\lambda}{2m} b^2$$

omit

L2 regularization $\quad \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

: w(가중치)를 활용하고, 절편(b)를 사용하지 않는 이유는 w는 높은 분산을 가지고, b는 상대적으로 큰 영향을 주지 않기 때문

L1 regularization $\quad \frac{\lambda}{2m}\sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m}\|w\|_1$

w will be sparse

: w벡터 안에 0 값이 많이 존재하게 되어 sparse 상태가 된다. 파라미터 set이 0이면, 모델을 저장하는게 더 적은 메모리를 사용해서, 모델을 압축하는 데 유용하다. 하지만 그렇게 많이 사용되지는 않고, 대신에 L2 regulation이 많이 사용된다.

Andrew Ng

Neural network

$$\rightarrow J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

$w^{[l]} : (n^{[l]}, n^{[l-1]})$

$w^{[l]} : (n^{[l]}, n^{[l-1]})$ 차원의 matrix

"Frobenius norm" $\|\cdot\|_2^2$ $\|\cdot\|_F^2$

$$dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} w^{[l]}$$

$$\frac{\partial J}{\partial w^{[l]}} = dw^{[l]}$$

$$\rightarrow w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

$$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}\right]$$

$$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha\,(\text{from backprop})$$

$$= \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{<1} w^{[l]} - \alpha\,(\text{from backprop})$$

L_2 : "Weight decay" 라고 불리는 이유

∴ weight matrix 가 (원래 가지는 값) $\left(1 - \frac{\alpha\lambda}{m}\right)$ 배

줄어들게 되므로

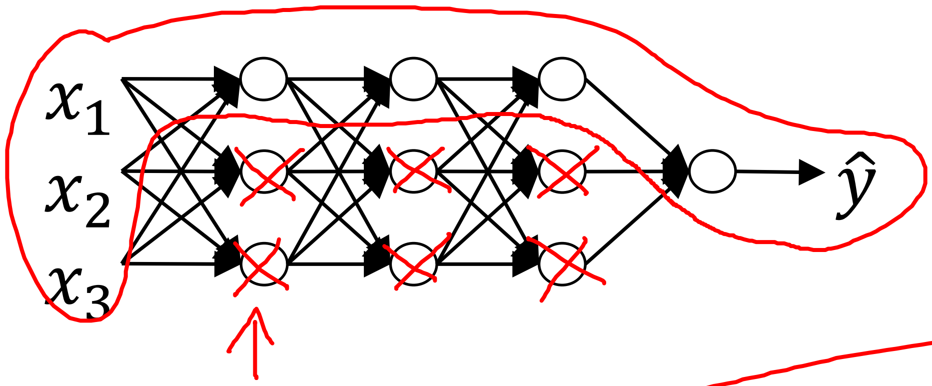
Andrew Ng

deeplearning.ai

Regularizing your neural network

Why regularization reduces overfitting

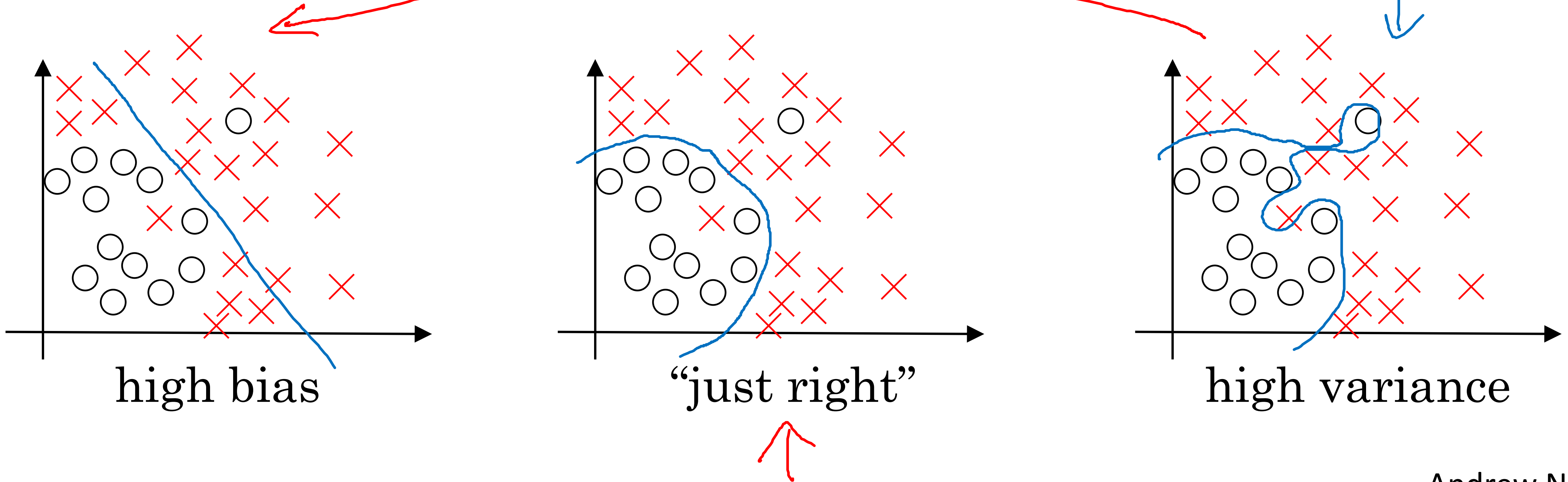
How does regularization prevent overfitting?



$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

lambda 값을 크게 만들면, 가중치 행렬 w를 0에 가깝게 설정할 수 있다. (이것에 의해 왼쪽 그림처럼 hidden unit의 영향을 거의 0으로 만듦)
이에 따라 신경망이 간단하고 작은 망으로 이루어져
과대적합이 덜 일어난다.

$$w^{[l]} \approx 0$$

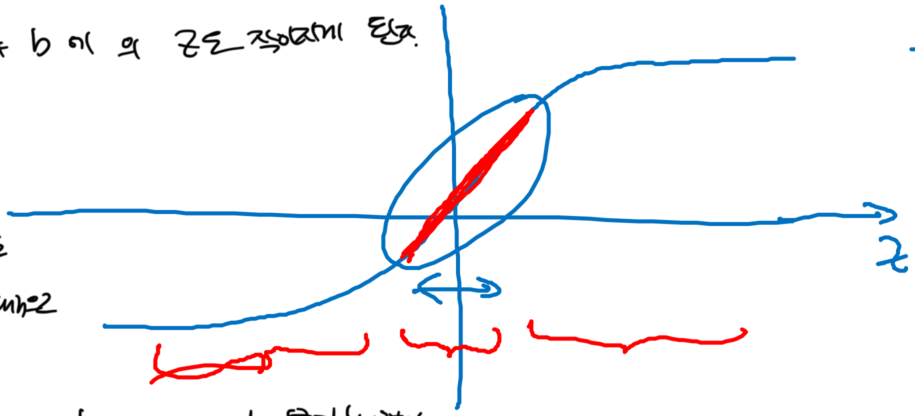


How does regularization prevent overfitting?

tanh 함수를 사용하면 lambda 값이 커지면 비용함수에 의해
가중치 W는 작아지게 되며, $z = w * x + b$ 에 의 z도 작아지게 된다.

결과적으로 일반화 파라미터 λ가 커질
w 매개변수는 작아지고 이에 따라 결과값 z도
작은 범위를 가지게 되며, tanh 함수는 선형
선형관계를 보이게 된다.

=> Activation이 linear 하게 바뀌고 decision 이 단순해져 overfitting 을 할 수 없게 된다.

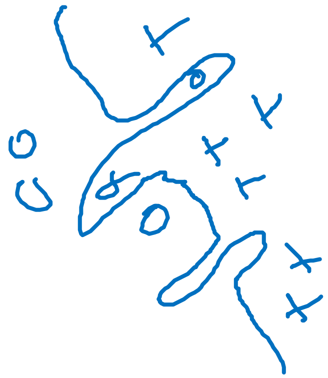


tanh

$g(z) = \tanh(z)$

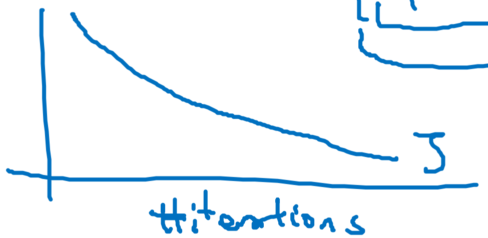
$\lambda \uparrow \quad W^{[l]} \downarrow$

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$



Every layer ≈ linear.

$$J(\ldots) = \sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_l \|W^{[l]}\|_F^2$$



Andrew Ng

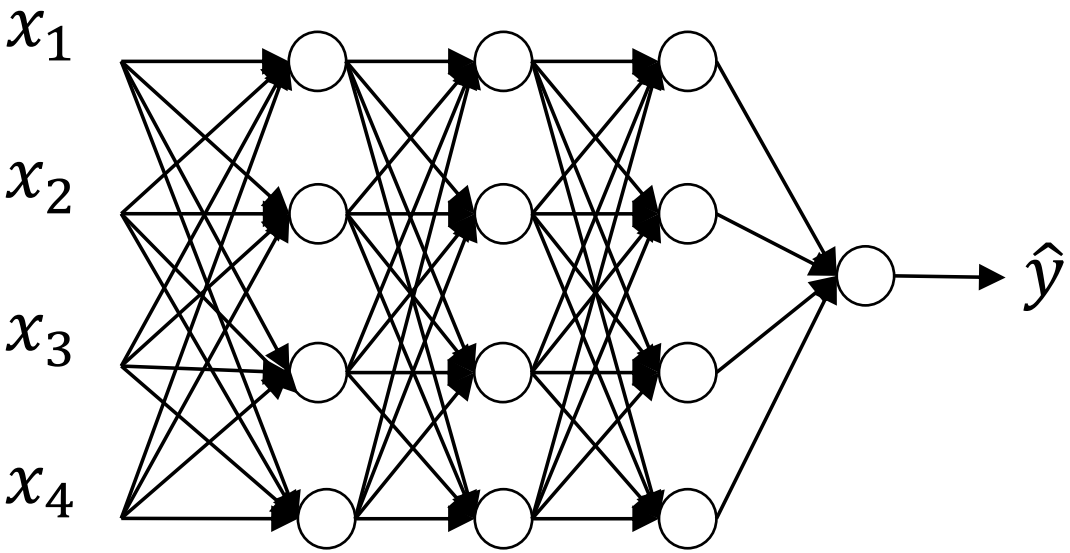
deeplearning.ai

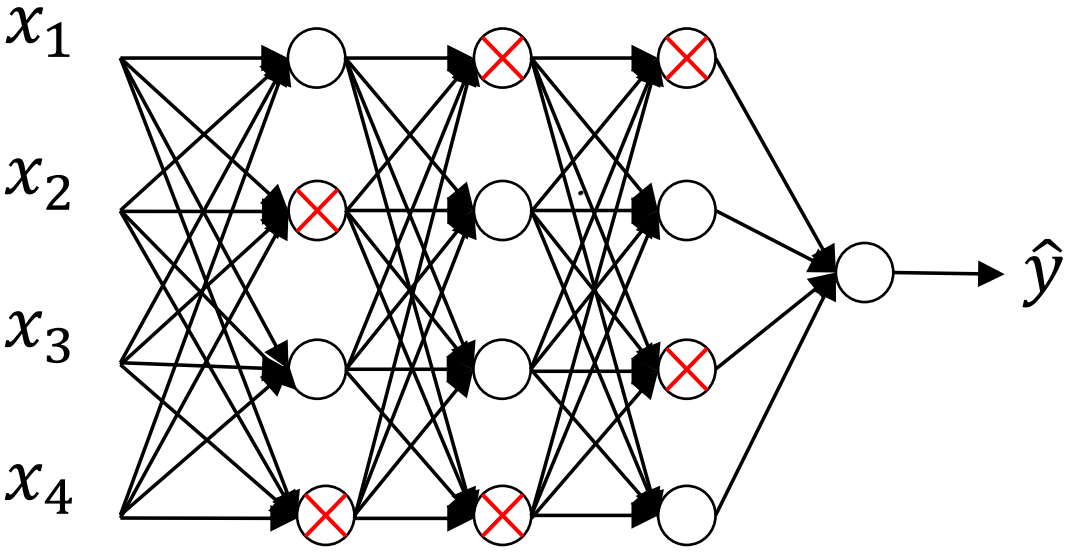
Regularizing your neural network

Dropout regularization

Dropout regularization

: 신경망 각각의 층에 대해 노드를 삭제하는 확률 (1- keep_prob) 을 지정하는 것
이다. 삭제할 노드를 확률에 따라 랜덤으로 선택하여 삭제한 노드의 들어가는
모든 나가는 링크를 모두 삭제한다.
이에 따라 기존 모델에 비해 작고 간단해진 네트워크가 만들어져, 작아진
네트워크로 학습을 진행하게 된다.





각 layer 에 대해 노드에서 확률을 설정

0.5 0.5 0.5

Andrew Ng

Implementing dropout ("Inverted dropout")

Illustrate with layer $l = 3$. keep-prob = 0.8 (0.2)

→ `d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_prob`

`a3 = np.multiply(a3, d3)` # a3 *= d3.

→ `a3 /= keep_prob` ←

50 units. ⇝ 10 units shut off

$z^{[4]} = w^{[4]} \cdot a^{[3]} + b^{[4]}$

↑ reduced by 20%.

$/= 0.8$

Test

역 드롭아웃 (Inverted dropout) 은 드롭아웃 방식에 대해 노드 삭제의 영향을 받는 확률을 줄여 삭제하지 않는 확률 (keep-prob) 을 나누는 것에서 이는 기존의 삭제하여 영향을 준 확률을 기대값으로 맞추기 위함이다.

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

No drop out.

$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[3]} = \ldots$

$\downarrow$

$\hat{y}$

/= keep-prob

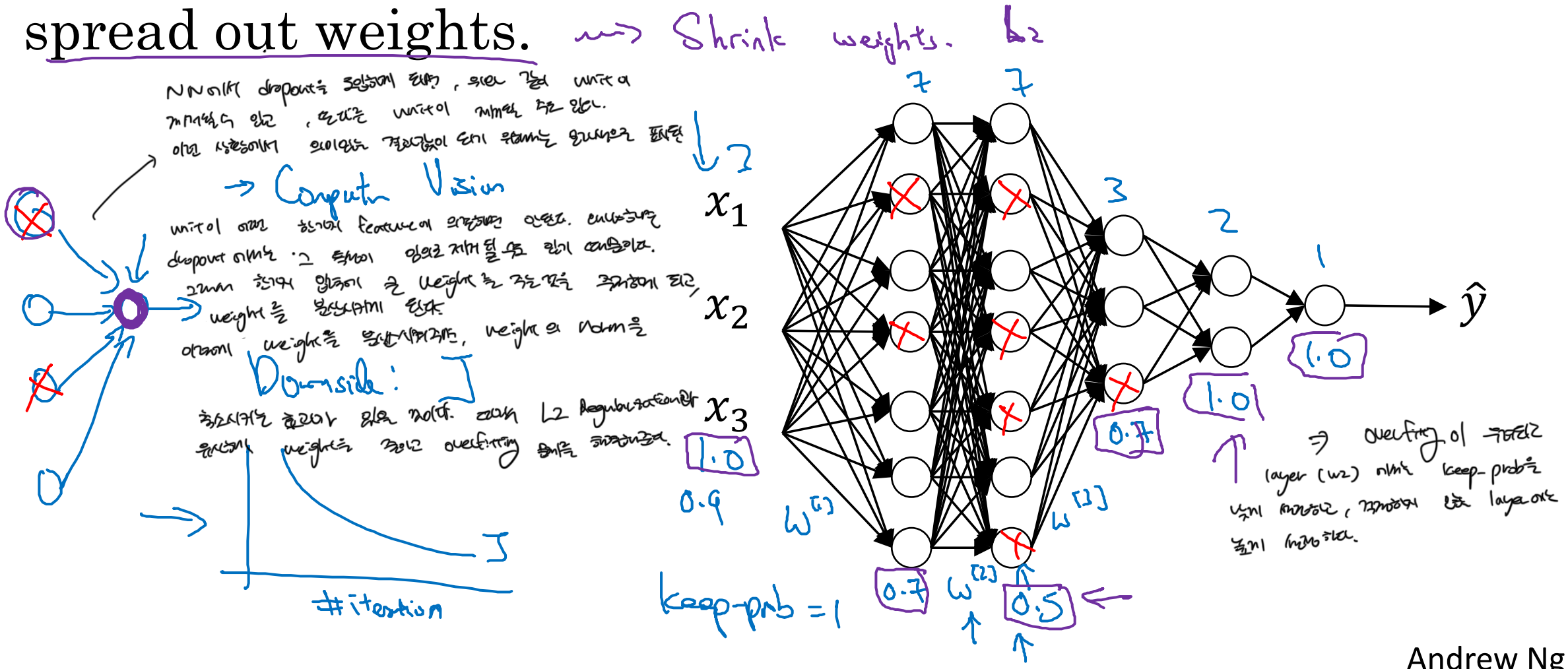
deeplearning.ai

Regularizing your neural network

Understanding dropout

Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights. ⇒ Shrink weights.



Andrew Ng

< Dropout의 단점 >

1. Cross Validation에서 시간낭비해야 함 하이퍼파라미터가 증가했다.
2. Cost Function J가 명확히 계산되지 않았다.
3. 각 반복마다 노드를 확률적으로 제거하기 때문에, Gradient Descent 성능을 최적화하기 어렵다.

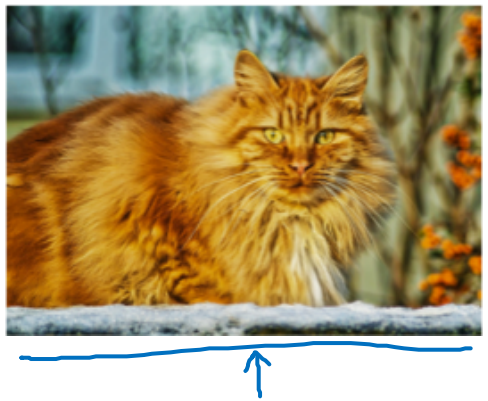
Dropout은 Computer Vision에서 도입되었다.
Computer Vision에서 입력값이 너무 커서, 모든 특성들을 알고리즘이 학습 하이퍼파라 부족했다.

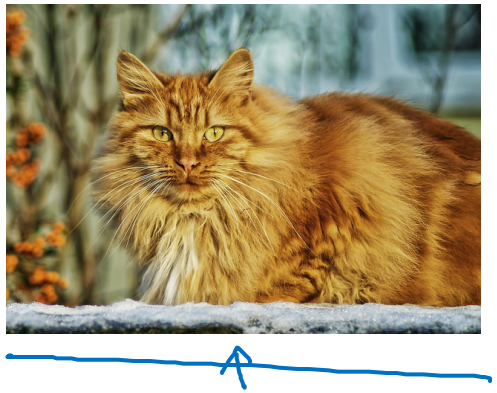
deeplearning.ai

Regularizing your neural network

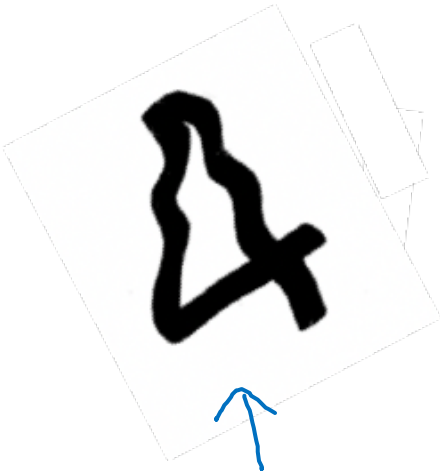
Other regularization methods

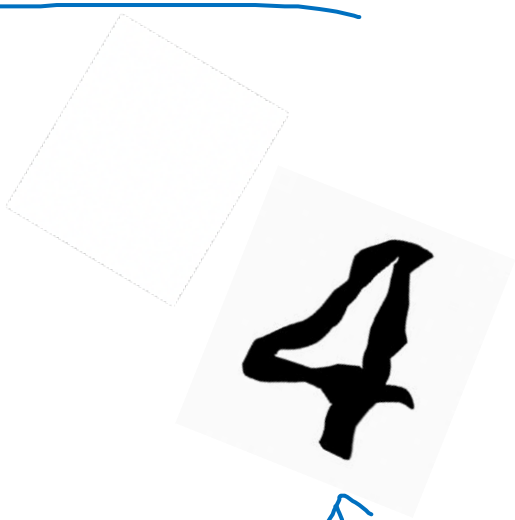
Data augmentation





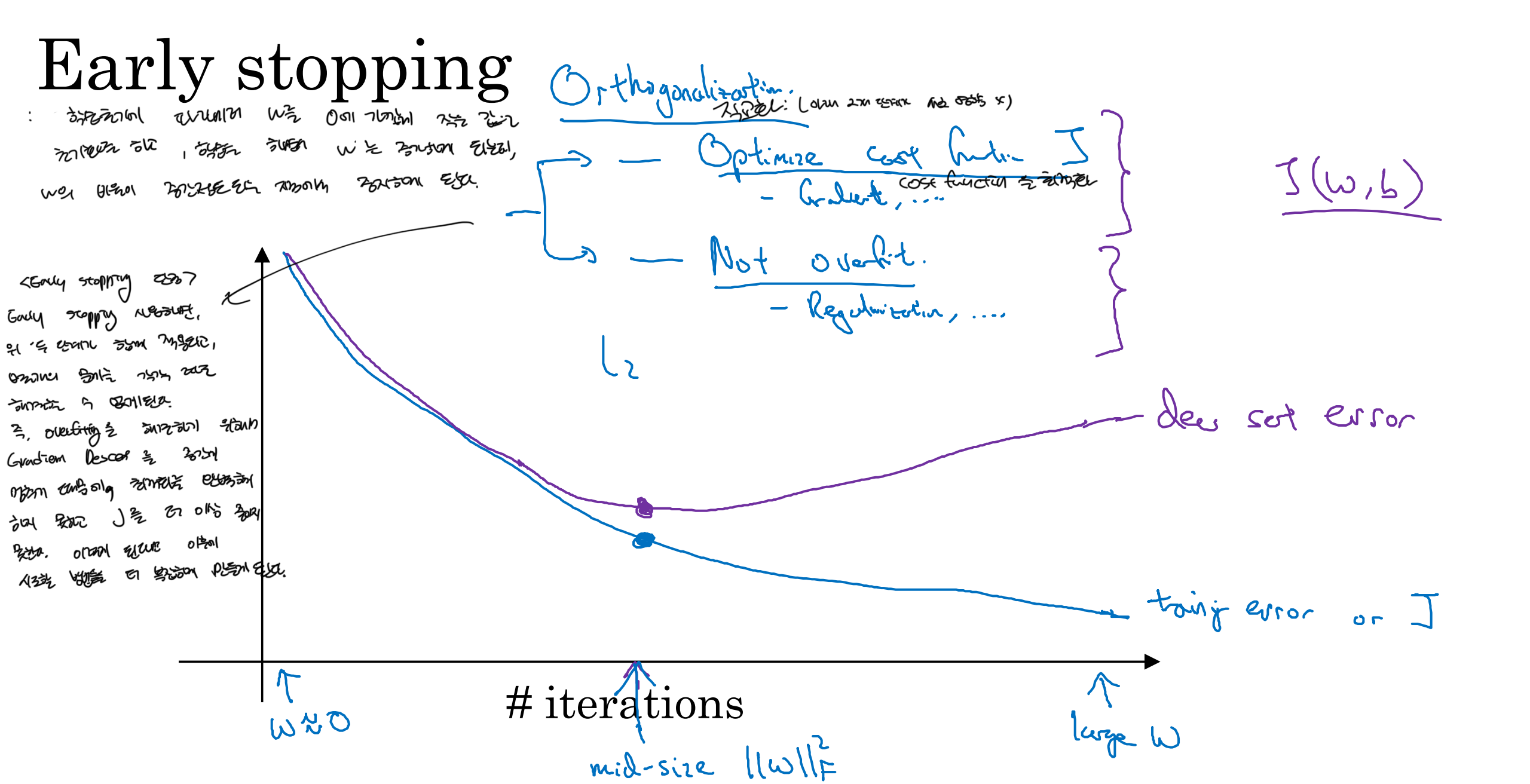
4





Andrew Ng

Early stopping



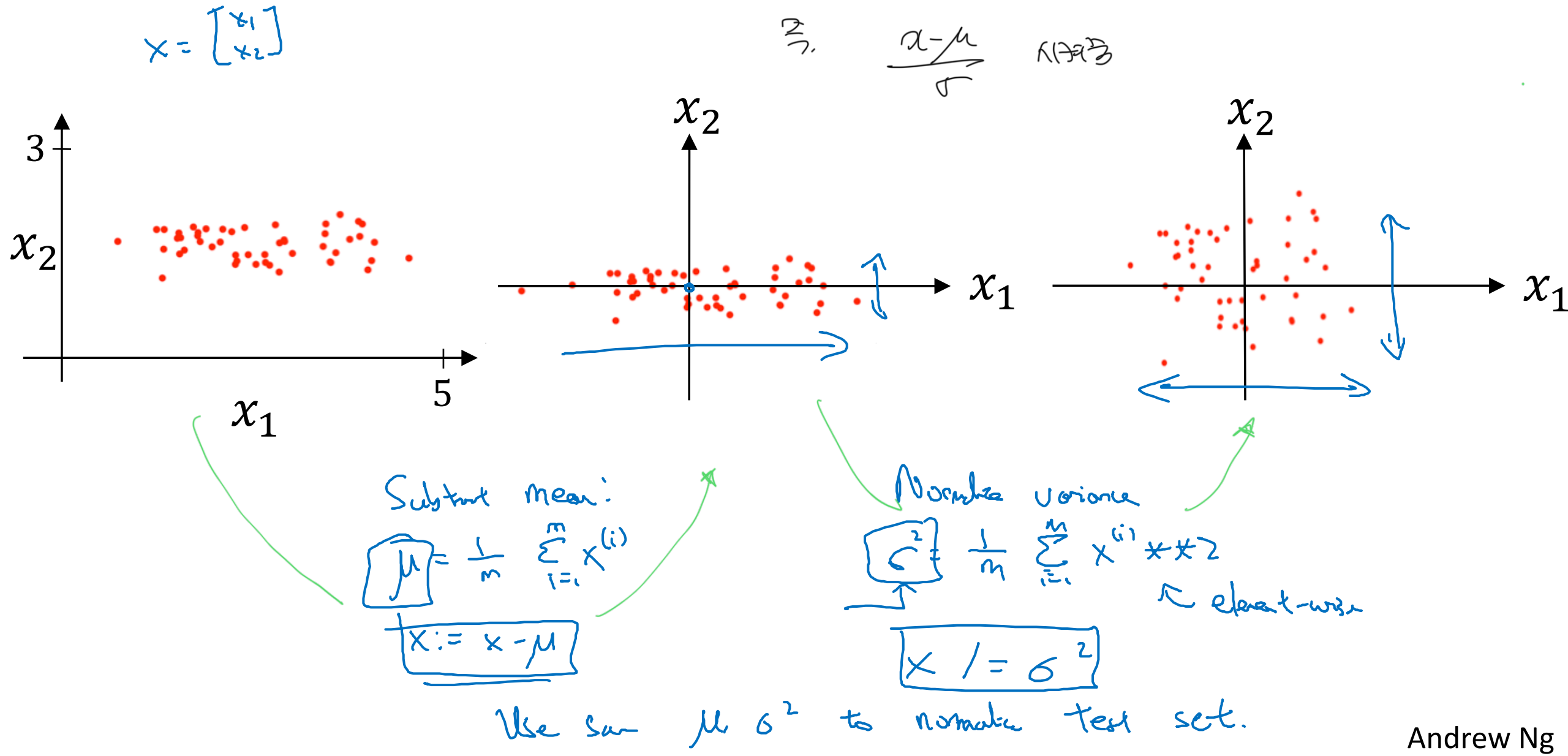
Andrew Ng

deeplearning.ai

Setting up your optimization problem

Normalizing inputs

Normalizing training sets

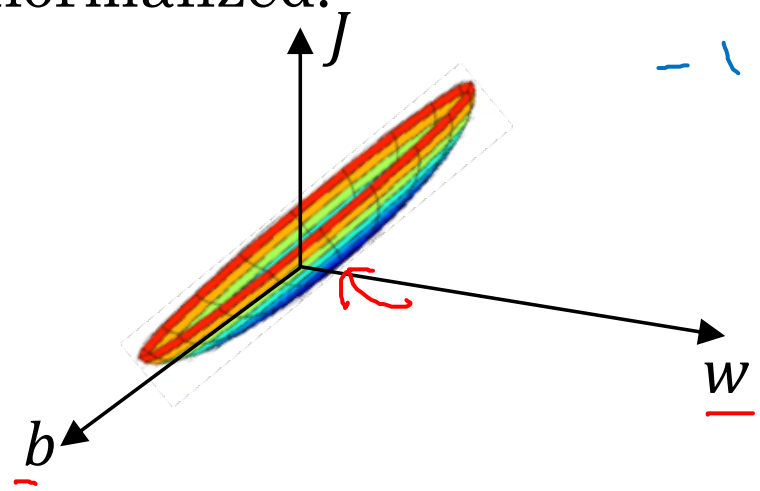


Andrew Ng

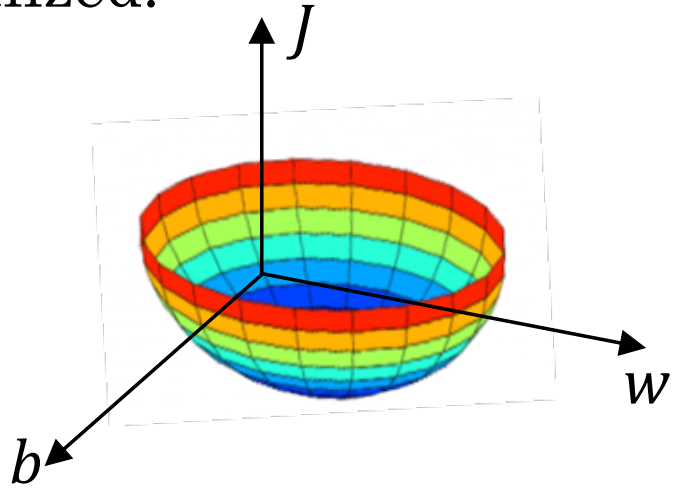
Why normalize inputs?

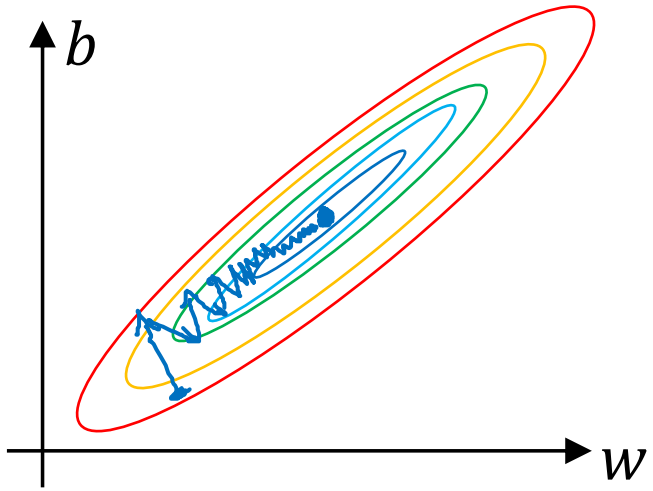
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

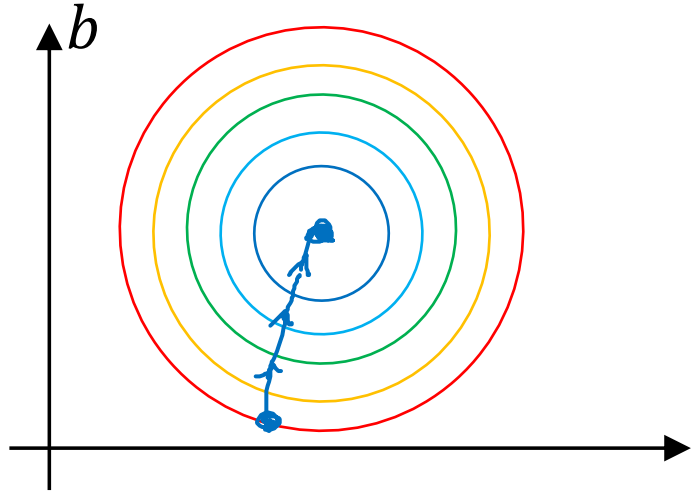
Unnormalized:



Normalized:







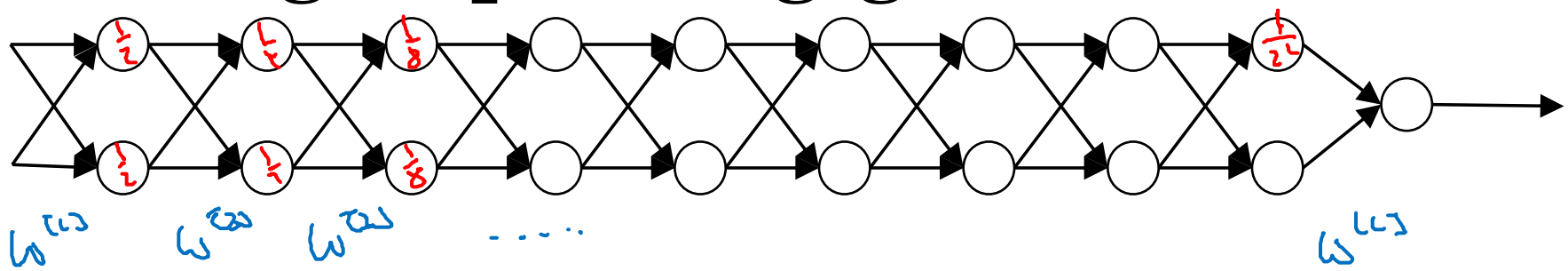
Andrew Ng

deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients



$L = 150$

$g(z) = z$: linear activation function

$b^{[l]} = 0$

$\hat{y} = W^{[L]} W^{[L-1]} W^{[L-2]} \cdots W^{[3]} W^{[2]} W^{[1]} x$

$a^{[3]}$

1.5^L

0.5^L

$W^{[l]} > I$

$W^{[l]} < I \quad \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}$

$z^{[1]} = W^{[1]} x$

$a^{[1]} = g(z^{[1]}) = z^{[1]}$

$a^{[2]} = g(z^{[2]}) = g(W^{[2]} a^{[1]})$

$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \rightarrow \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$

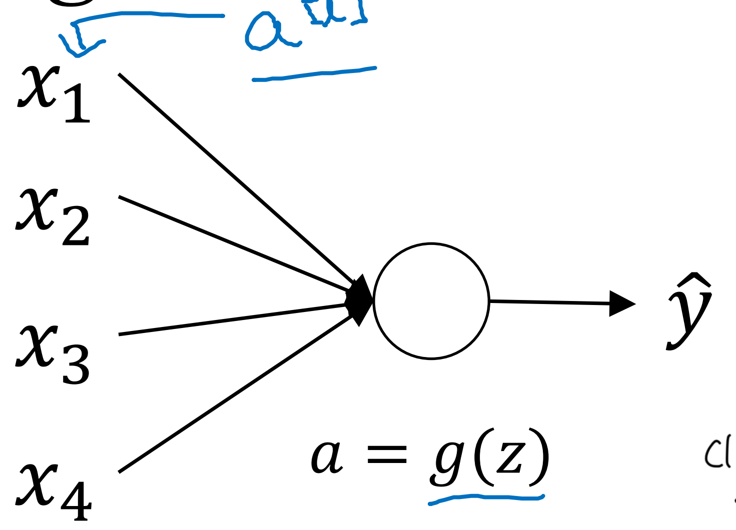
$\hat{y} = W^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x \rightarrow \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1}$

$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example



$w^{[l]}$ < 가중치 초기화 >
: 가중치를 (너무 너무) 큰 수나 너무 작은 수로 하지 않음으로 해서 폭발이나 소실을 막아준다. 즉, 기울기 값이 폭발하거나 너무 작은 값을 가지지 않기 위해서는 좋은 초기화 (large n) 경우에는 작은 가중치 (smaller w)가 필요하다.

$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n$ ~~$+b$~~

large n → smaller w_i

$Var(w_i) =$ ~~$\frac{1}{n}$~~ $\frac{2}{n}$

(w_i 분산은 $1/n$으로 설정)
n: 입력 특성 개수

$w^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

ReLU $\quad g^{[l]}(z) = ReLU(z)$

ReLU 사용하는 경우, w_i의 분산: $\frac{2}{n^{[l-1]}}$

Other variants:

tanh: $\sqrt{\frac{1}{n^{[l-1]}}}$

Xavier initialization

$\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

$f(\theta) = \theta^3$

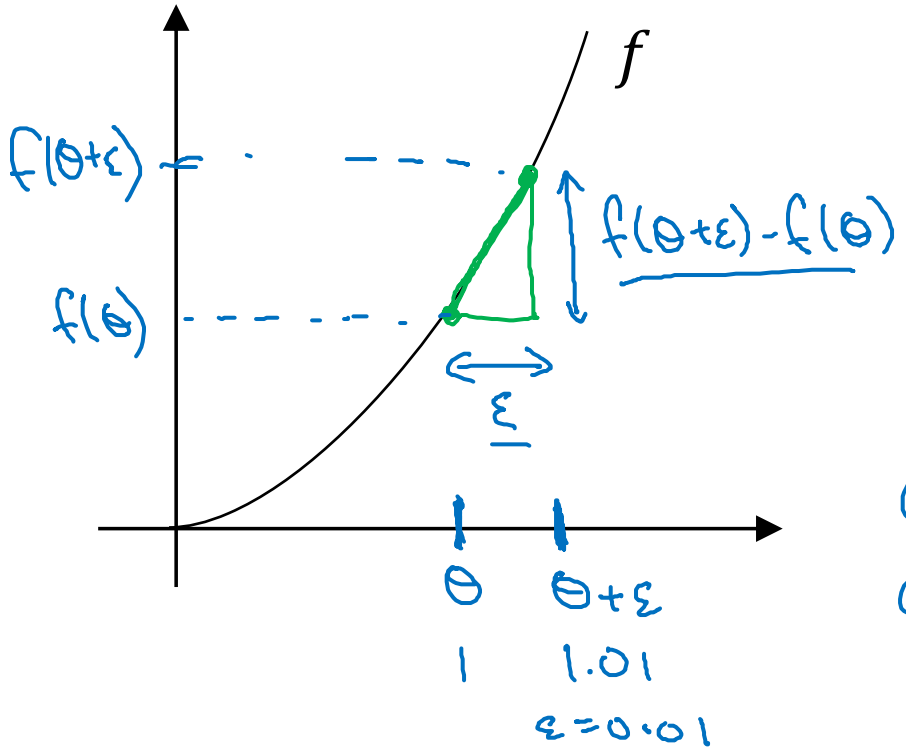
$\theta \in \mathbb{R}$.

$g(\theta) = \frac{d}{d\theta} f(\theta) = f'(\theta)$

$g(\theta) = 3\theta^2$.

$g(\theta) = 3 \cdot (1)^2 = 3$ when $\theta = 1$

$\frac{dw}{db}$



$\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} \approx g(\theta)$

$\theta = 1$

$\theta + \varepsilon = 1.01$

$\frac{(1.01)^3 - 1^3}{0.01} = 3.0301 \approx 3$

0.0301

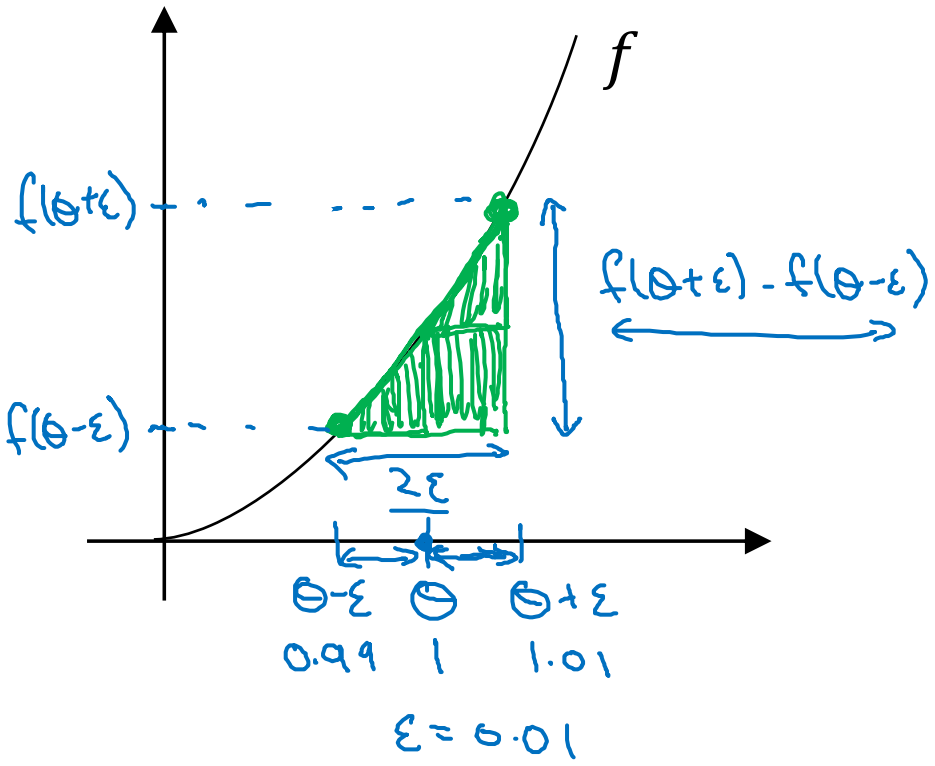
3.1

3.2

Checking your derivative computation

$f(\theta) = \theta^3$

Gradient checking in ...



$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)} = 3.0001 \approx 3$$

$g(\theta) = 3\theta^2 = 3$

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

$f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$

$O(\varepsilon^2)$

0.01

0.0001

$\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon}$ error: $O(\varepsilon)$

0.01

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking

Gradient check for a neural network

Take $W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}$ and reshape into a big vector θ.

concatenate

$J(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}) = J(\theta)$

1. 모든 파라미터를 벡터 형태로 연결시켜 하나의 vector로 만들어줌.

Take $dW^{[1]}, db^{[1]}, \ldots, dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

concatenate

Is $d\theta$ the gradient of $J(\theta)$?

2. dW, db를 $d\theta$로 만들어 $J(\theta)$의 기울기와 비교함.

Gradient checking (Grad check)

$J(\theta) = J(\theta_1, \theta_2, \theta_3, \dots)$

for each i:

$d\theta_{approx}[i] = \dfrac{J(\theta_1, \theta_2, \dots, \theta_i + \varepsilon, \dots) - J(\theta_1, \theta_2, \dots, \theta_i - \varepsilon, \dots)}{2\varepsilon}$

$d\theta_{approx}[i]$: 양쪽에서 계산한 값

$\approx d\theta[i] = \dfrac{\partial J}{\partial \theta_i}$

$d\theta_{approx} \overset{?}{\approx} d\theta$

Check $\dfrac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$ ($d\theta_{approx}$와 $d\theta$의 차이 / 양쪽에서 계산한 값과 그래디언트 값의 차이)

$\varepsilon = 10^{-7}$

$\approx 10^{-7}$ — great! ←

10^{-5}

$\rightarrow 10^{-3}$ — worry. ←

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking implementation notes

Gradient checking implementation notes

- Don't use in training – only to debug

- If algorithm fails grad check, look at components to try to identify bug.

- Remember regularization.

- Doesn't work with dropout.

- Run at random initialization; perhaps again after some training.

Andrew Ng