deeplearning.ai

Setting up your ML application

Train/dev/test sets

Applied ML is a highly iterative process

layers

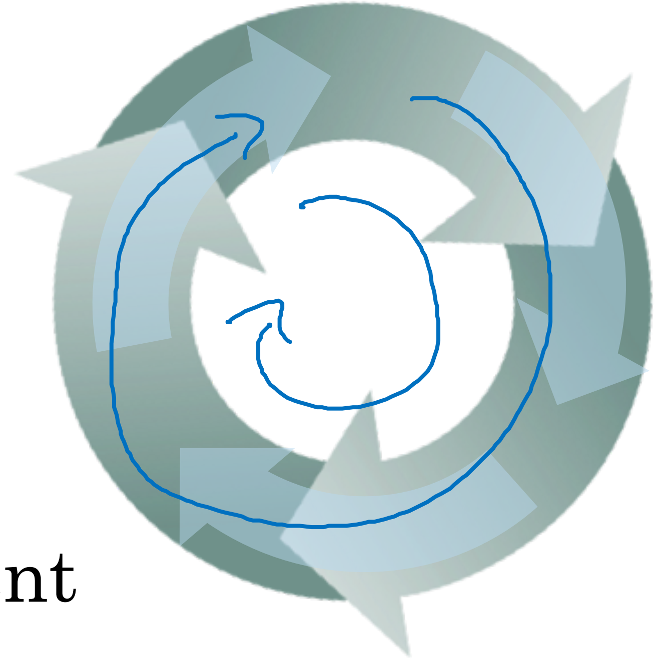
hidden units

learning rates

activation functions

…

Idea



Experiment

Code

NLP, Vision, Speech, Structural data

Ads search Security logistic ....

Andrew Ng

Train/dev/test sets

Data | training set | dev | test

- Hold-out cross validation
- Development set "dev"

test

train → test

Previous era: 70/30 %

100 - 1000 - 10000

60/20/20 %

Big data: 1,000,000 | 10,000 | 10,000

98/1/1 %

99.5 / ~~.25~~ .4 / ~~.25~~ .1 %

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

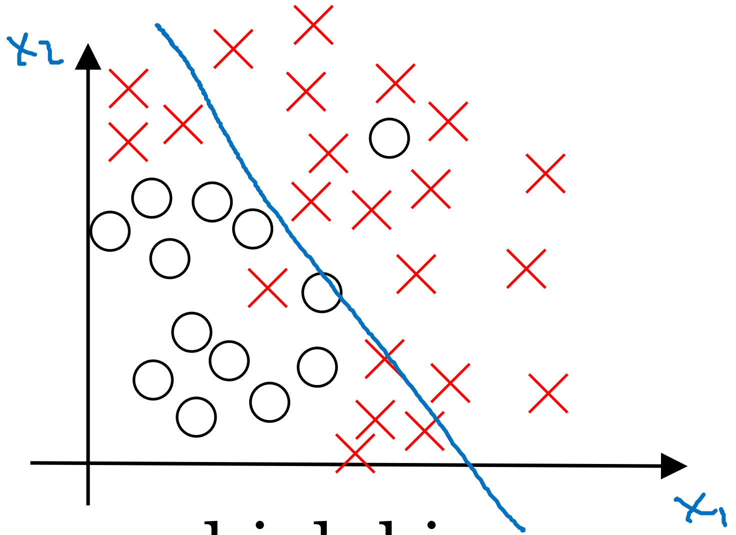
Not having a test set might be okay. (Only dev set.)

deeplearning.ai

Setting up your ML application

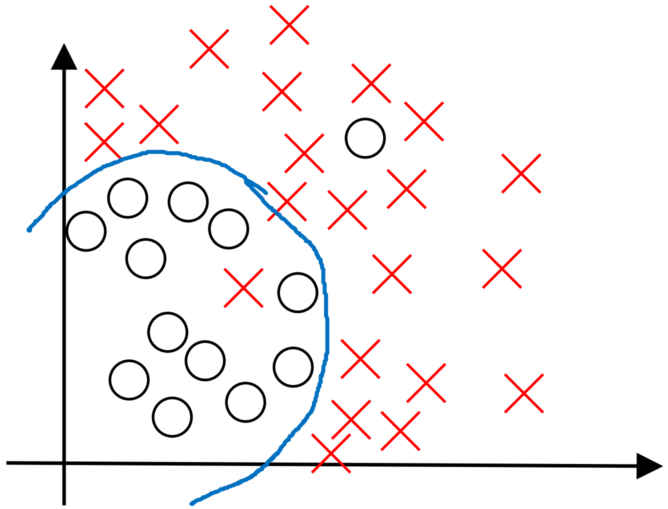
Bias/Variance

Bias and Variance

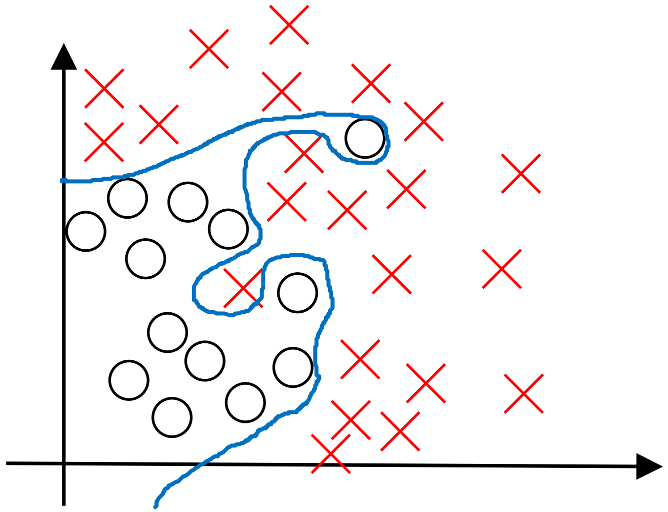


high bias

Underfitting



"just right"



high variance

Overfitting

Andrew Ng

Bias and Variance

Cat classification

y=1

y=0

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

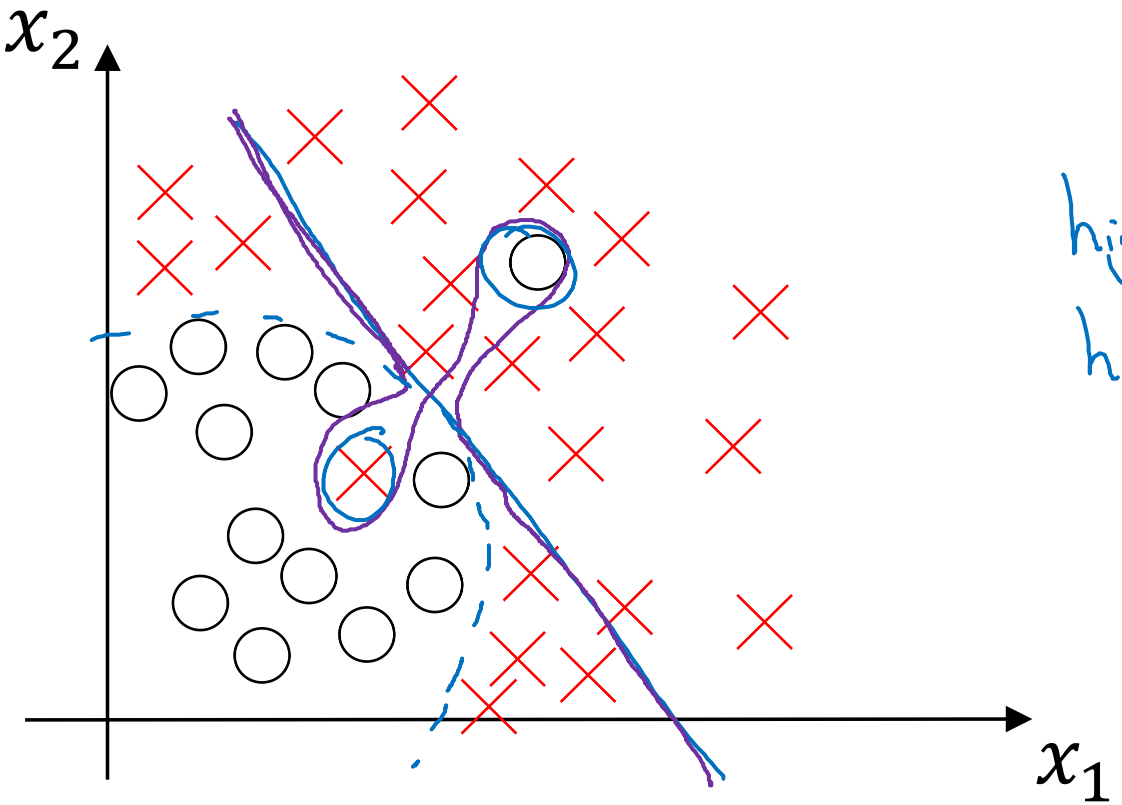
Human: ≈0%

Optimal (Bayes) error: ≈ ~~0%~~ 15%

Blurry images

Andrew Ng

High bias and high variance



high bias
high variance

Andrew Ng

deeplearning.ai

Setting up your ML application

Basic "recipe" for machine learning

Basic recipe for machine learning

High bias?
(training data performance)

↓ N

High variance?
(dev set performance)

↓ N

Done

Y → Bigger network
→ Train longer.
(NN architecture search)

Y → More data
→ Regularization
(NN architecture search)

"Bias variance tradeoff"

deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

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

λ = regularization parameter

lambda lambd

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

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

omit

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

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

w will be sparse

Andrew Ng

Neural network

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

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

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

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

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

"Weight decay"

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

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

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

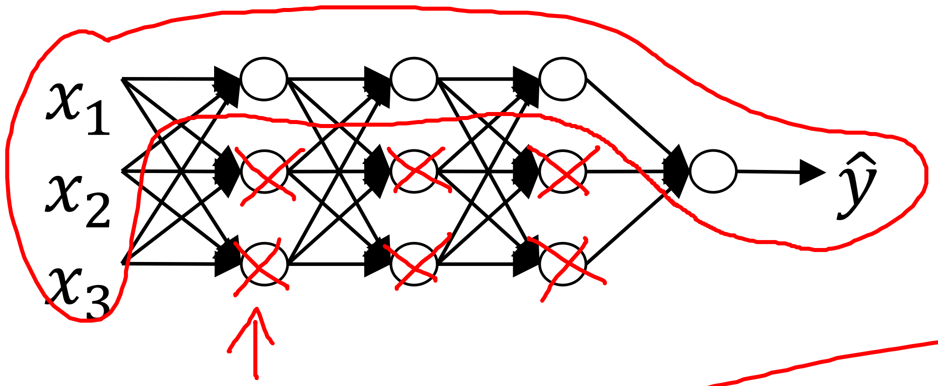
Andrew Ng

deeplearning.ai

Regularizing your neural network

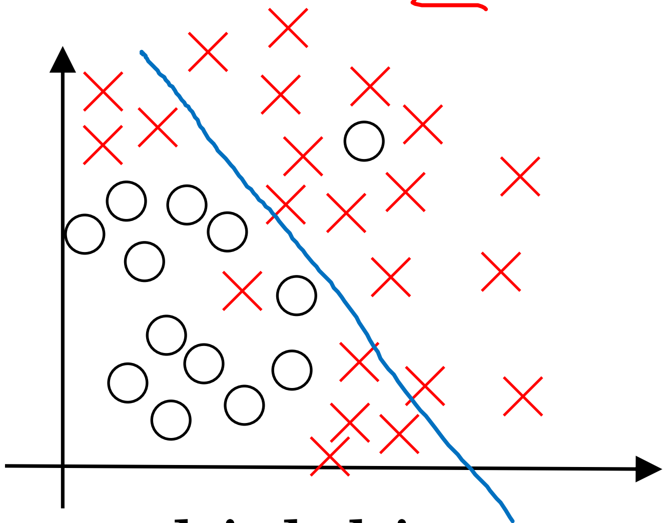
Why regularization reduces overfitting

How does regularization prevent overfitting?

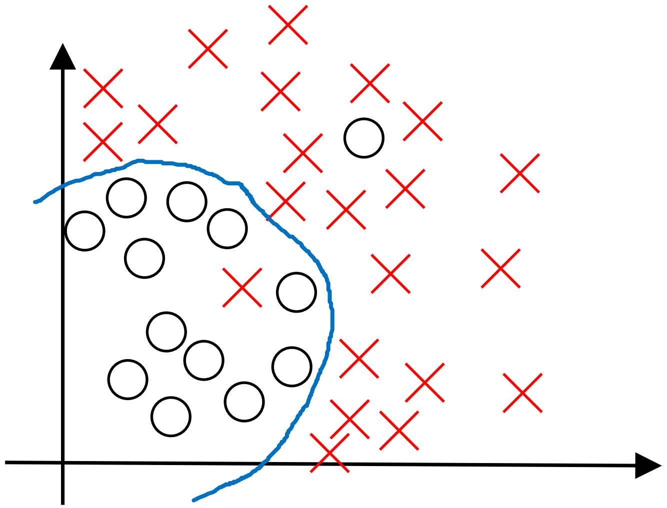


$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \underbrace{\|w^{[l]}\|_F^2}$$

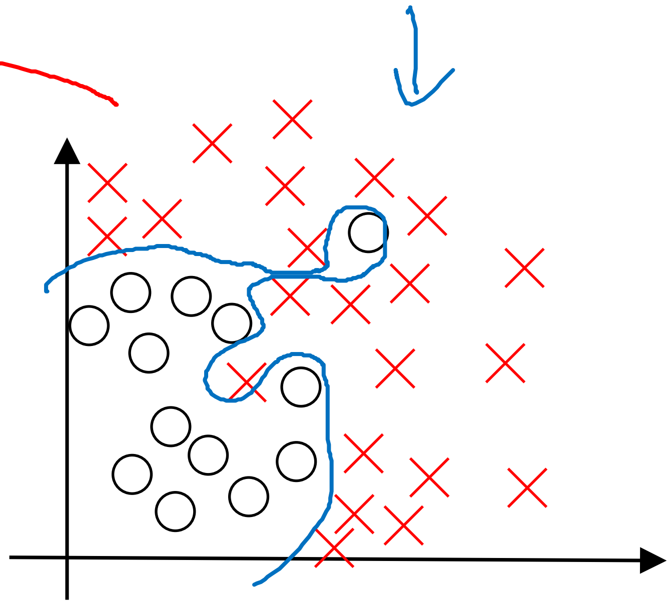
$$\underline{w^{[l]} \approx 0}$$



high bias

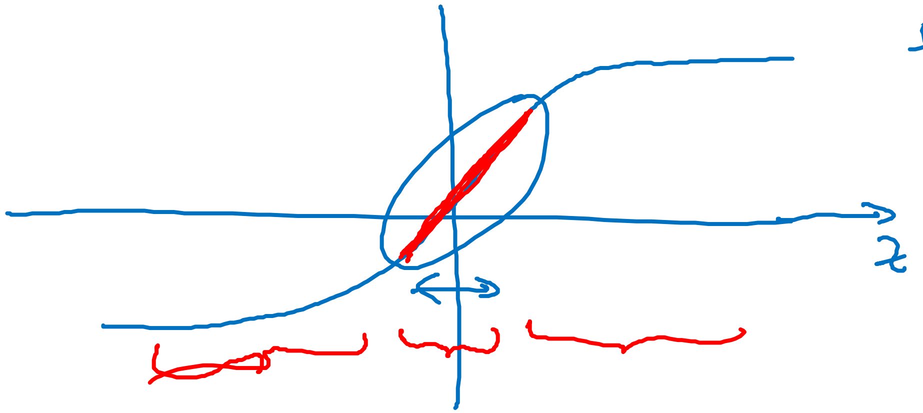


"just right"



high variance

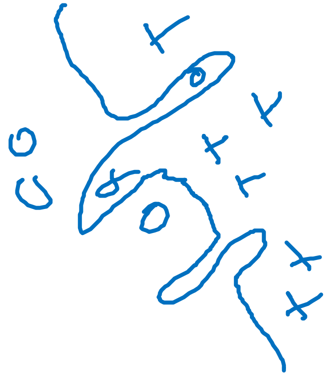
How does regularization prevent overfitting?



tanh

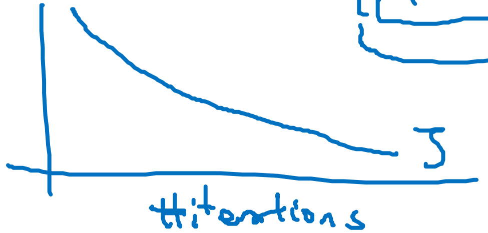
$g(z) = \tanh(z)$

$\lambda \uparrow \qquad W^{[l]} \downarrow \qquad z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$



Every layer $\approx$ linear.

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



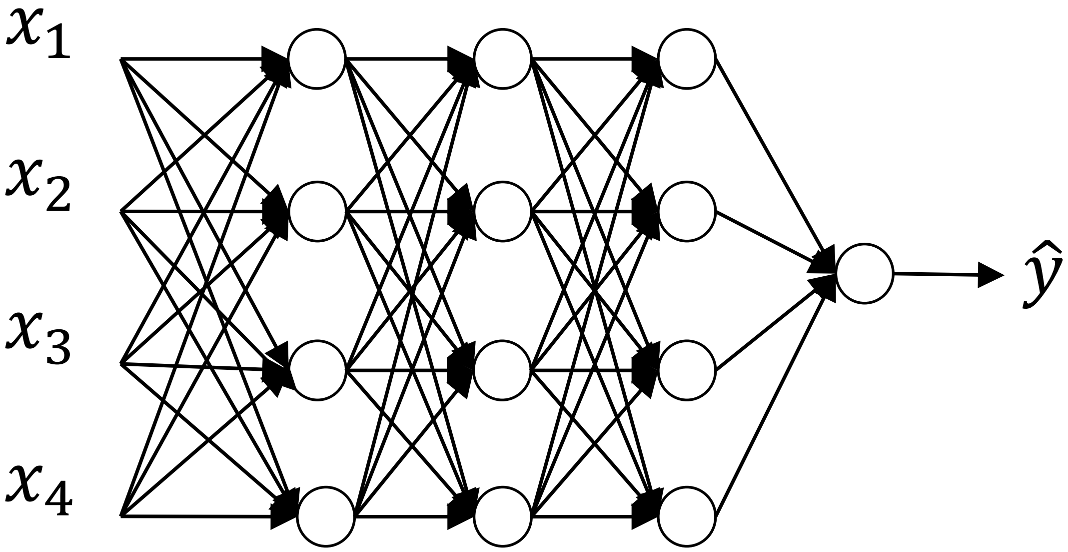
Andrew Ng

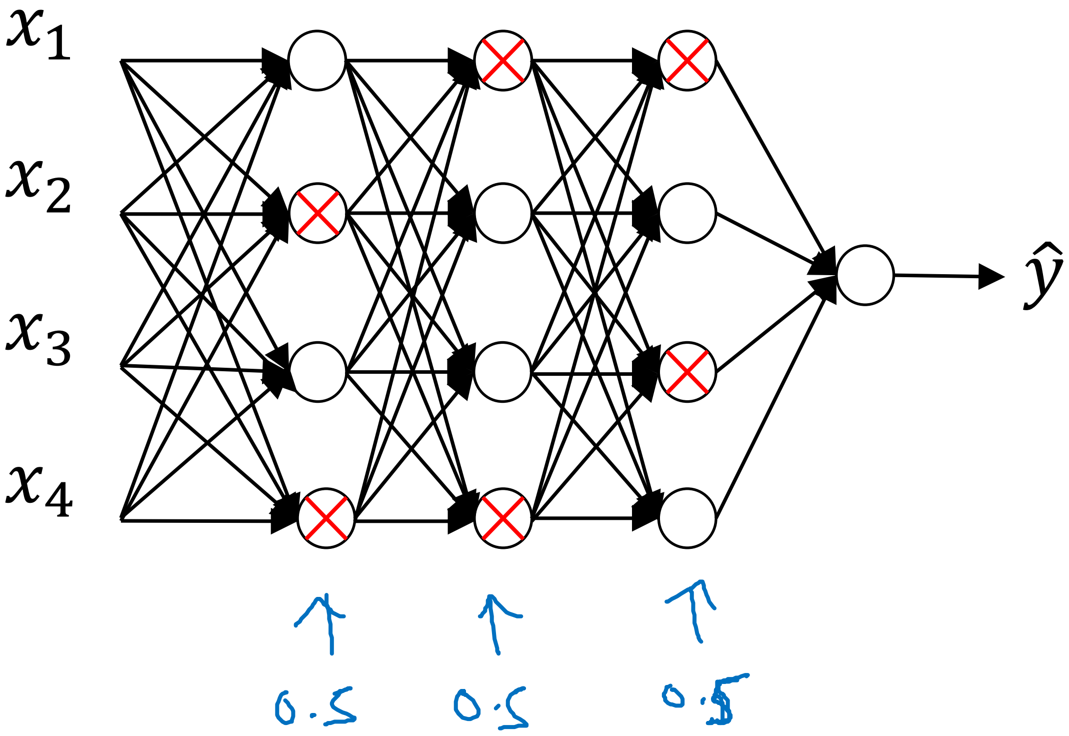
deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization





Andrew Ng

Implementing dropout ("Inverted dropout")

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

→ $\boxed{d3}$ = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob

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

→ $\boxed{\text{a3 /= } \cancel{0.8} \text{ keep-prob}}$ ←

50 units. ⇝ 10 units shut off

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

reduced by 20%.   Test

/= 0.8

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

<u>No drop out.</u>

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

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

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

$a^{[2]} = \dots$

$\downarrow$

$\hat{y}$

$/=$ keep-prob

Andrew Ng

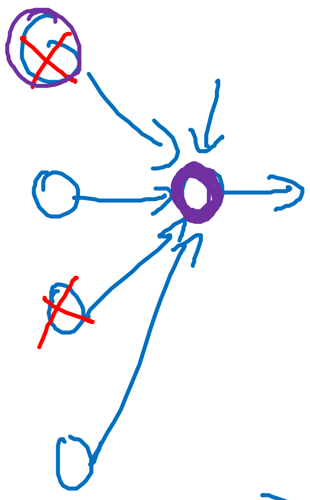
deeplearning.ai

Regularizing your neural network

Understanding dropout

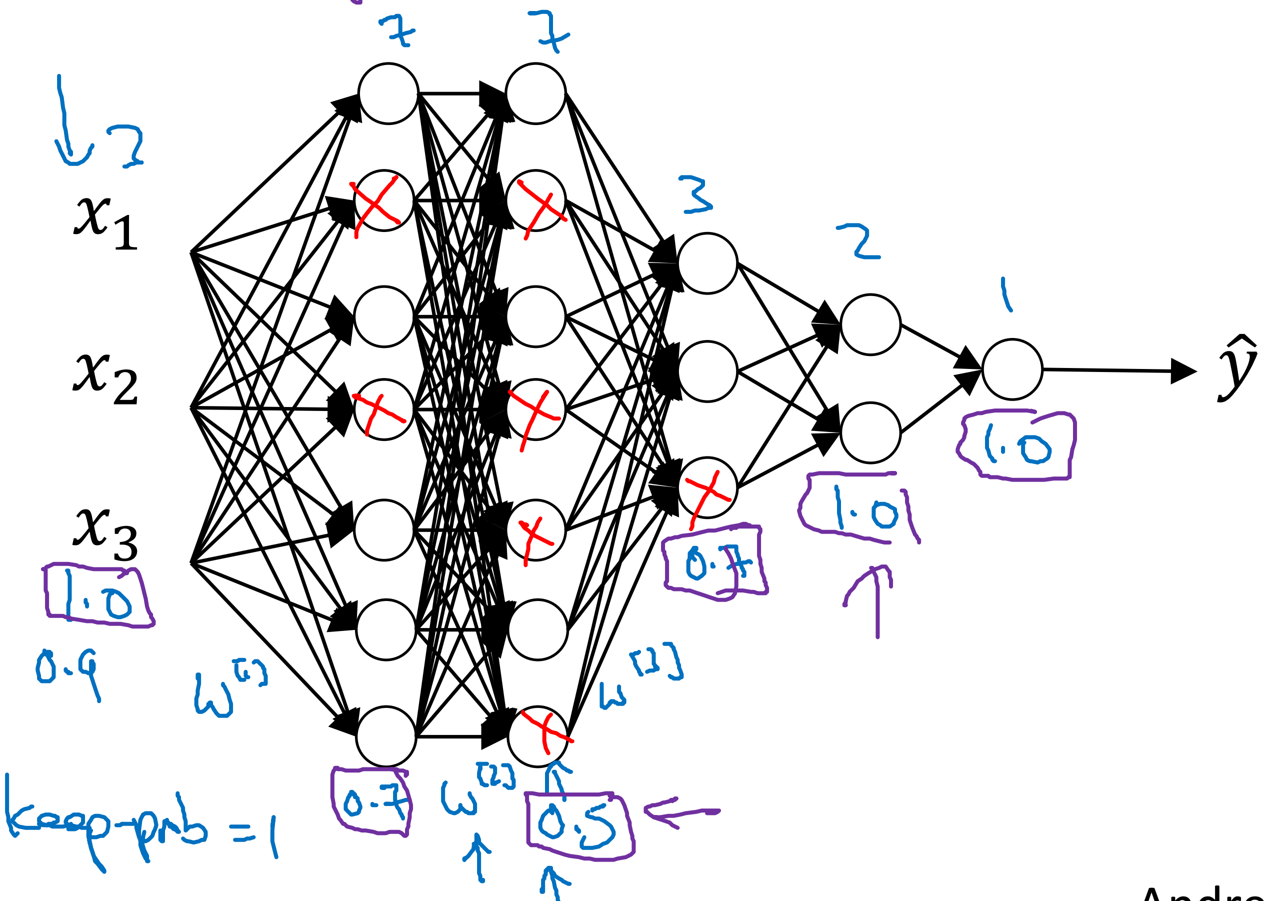
Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights. → Shrink weights. L_2



→ Computer Vision

Downside: J



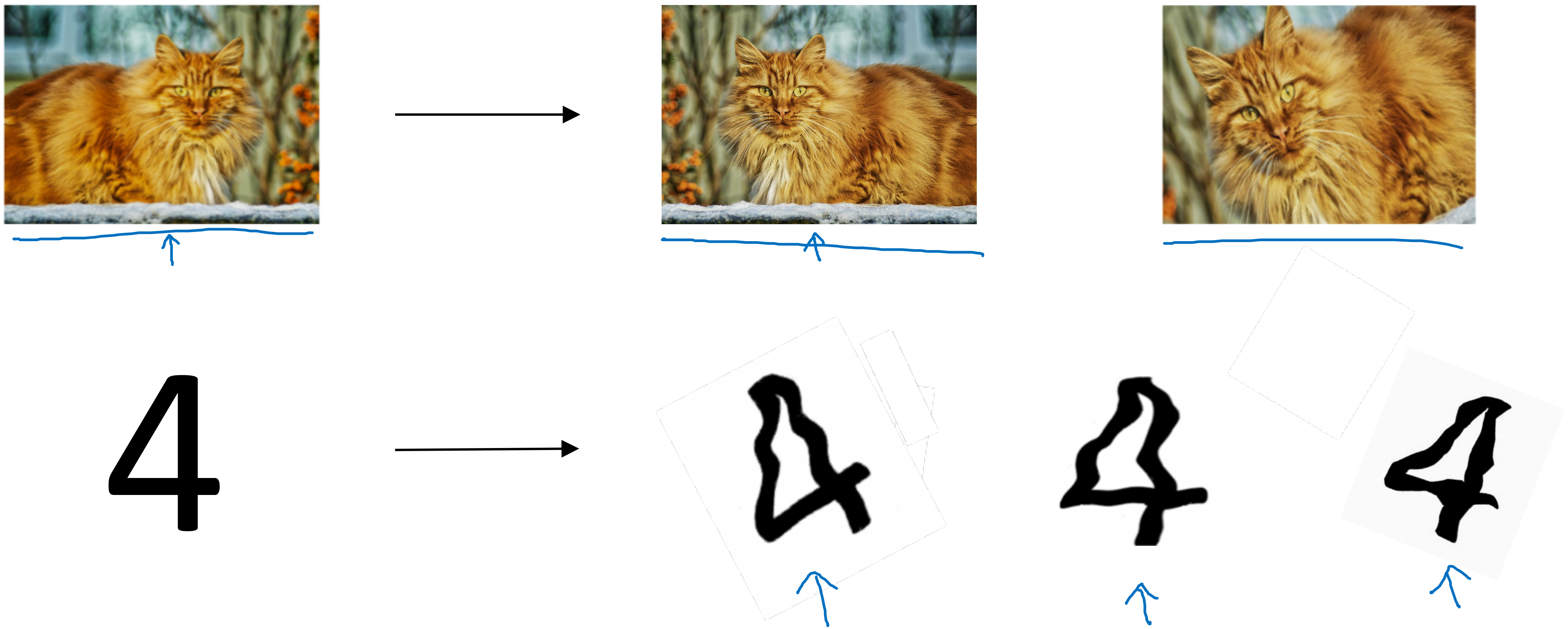
Andrew Ng

deeplearning.ai

Regularizing your neural network

Other regularization methods

Data augmentation



Andrew Ng

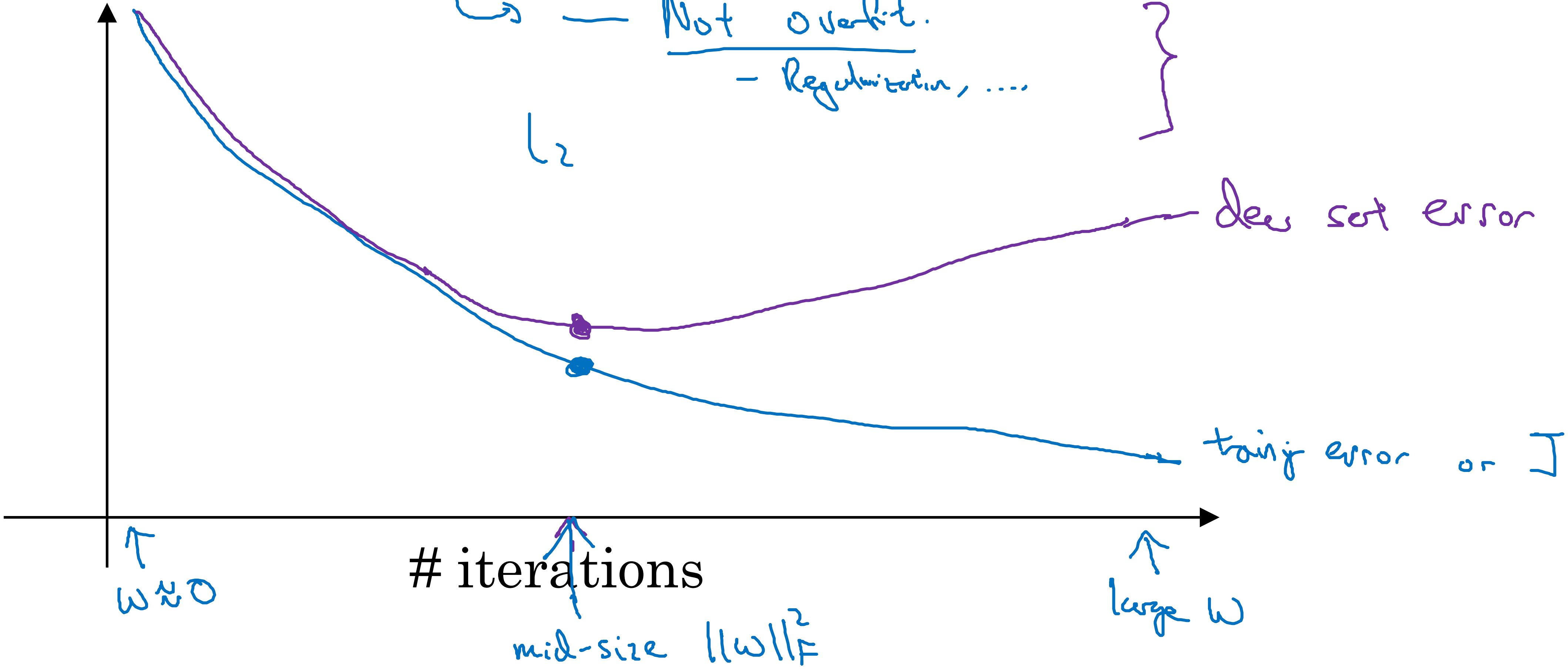
Early stopping

Orthogonalization.

- Optimize cost function J
  - Gradient, ...
- Not overfit.
  - Regularization, ...

$J(w,b)$

L_2



Andrew Ng

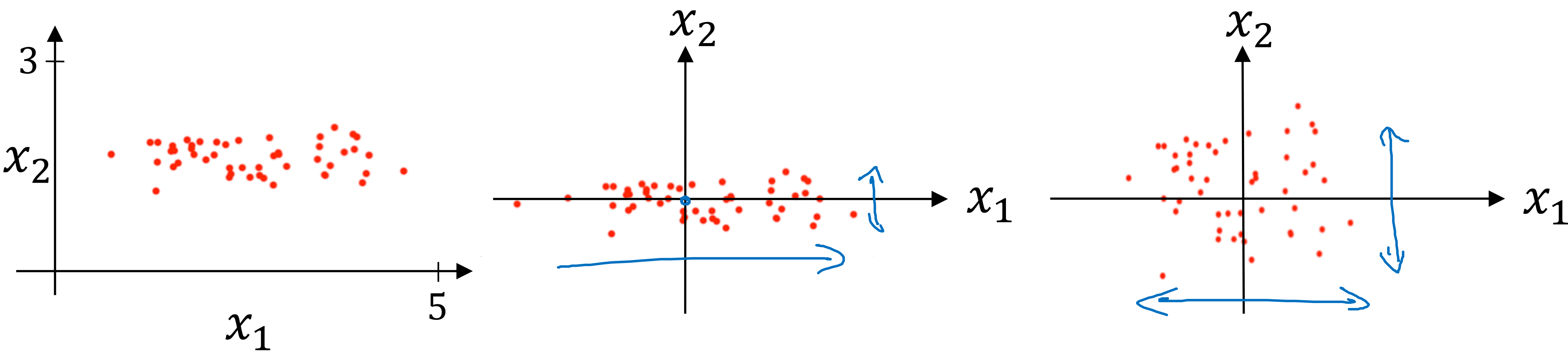
deeplearning.ai

Setting up your optimization problem

Normalizing inputs

Normalizing training sets

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



Subtract mean:

$$\boxed{\mu} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\boxed{x := x - \mu}$$

Normalize variance

$$\boxed{\sigma^2} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} {**} 2$$

← element-wise

$$\boxed{x \mathrel{/}= \sigma^2}$$

Use same μ, σ^2 to normalize test set.

Andrew Ng

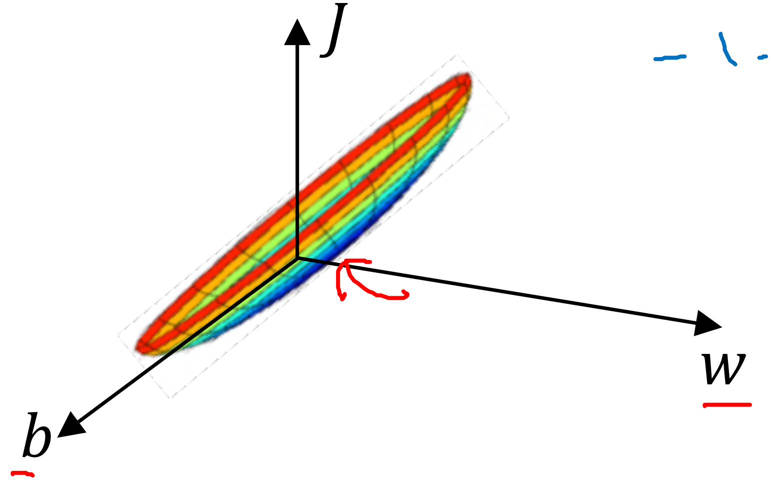
Why normalize inputs?

w_1 x_1 : $\underline{1 \ldots 1000}$ ←

w_2 x_2 : $\underline{0 \ldots 1}$ ←

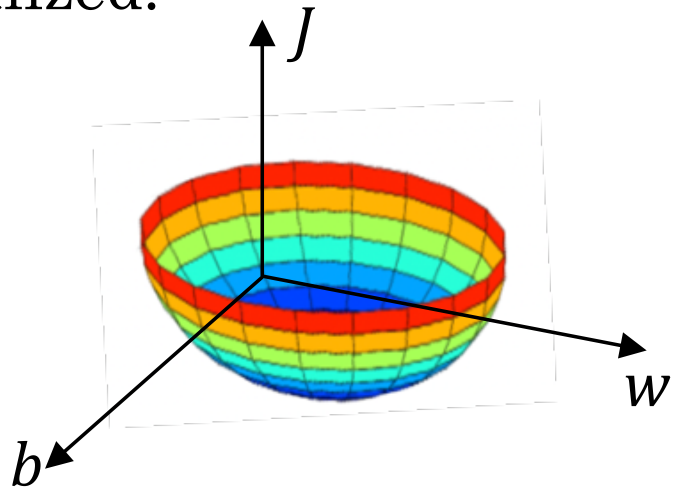
$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\,(\hat{y}^{(i)}, y^{(i)})$$

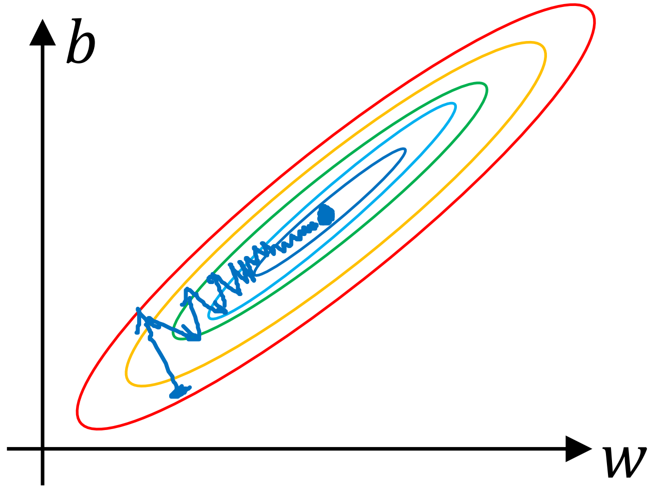
Unnormalized:



$-1 \ldots 1$

Normalized:

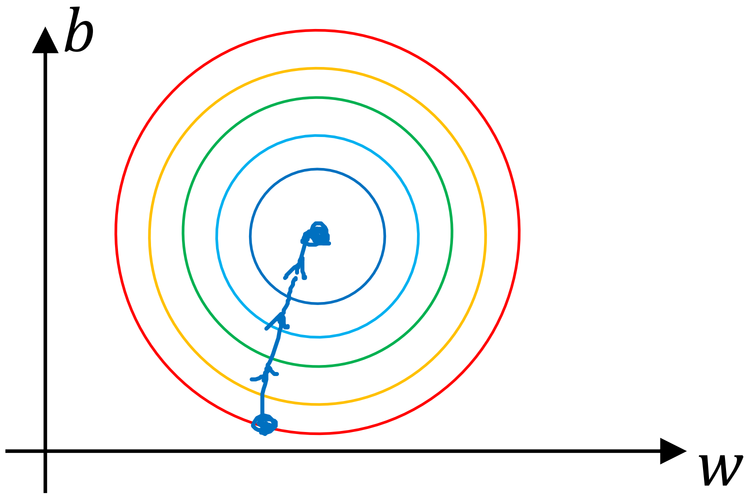




x_1 : $0 \ldots 1$

x_2 : $-1 \ldots 1$

x_3 : $1 \ldots 2$



Andrew Ng

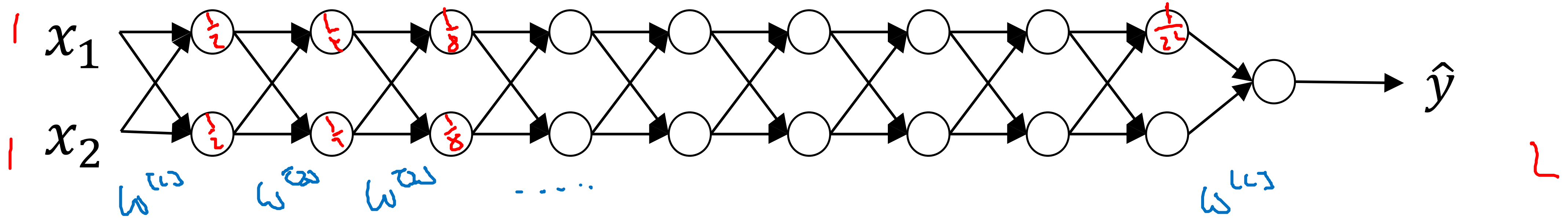
deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients

$L = 150$



$g(z) = z$. $\quad b^{[l]} = 0$.

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

1.5^L

0.5^L

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

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

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

$W^{[l]} > I$

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

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

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

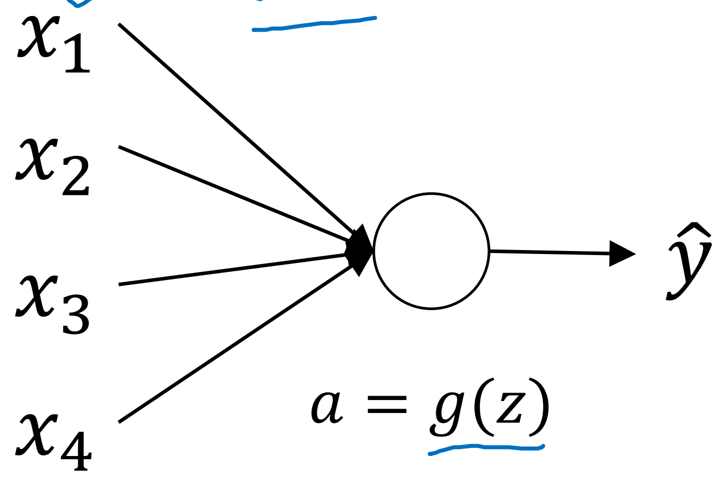
$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example

$a^{[l]}$



$w^{[l]}$

$a = g(z)$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \cancel{+b}$

Large n → Smaller w_i

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

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

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

Other variants:

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

Xavier initialization

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

deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

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

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

J

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

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

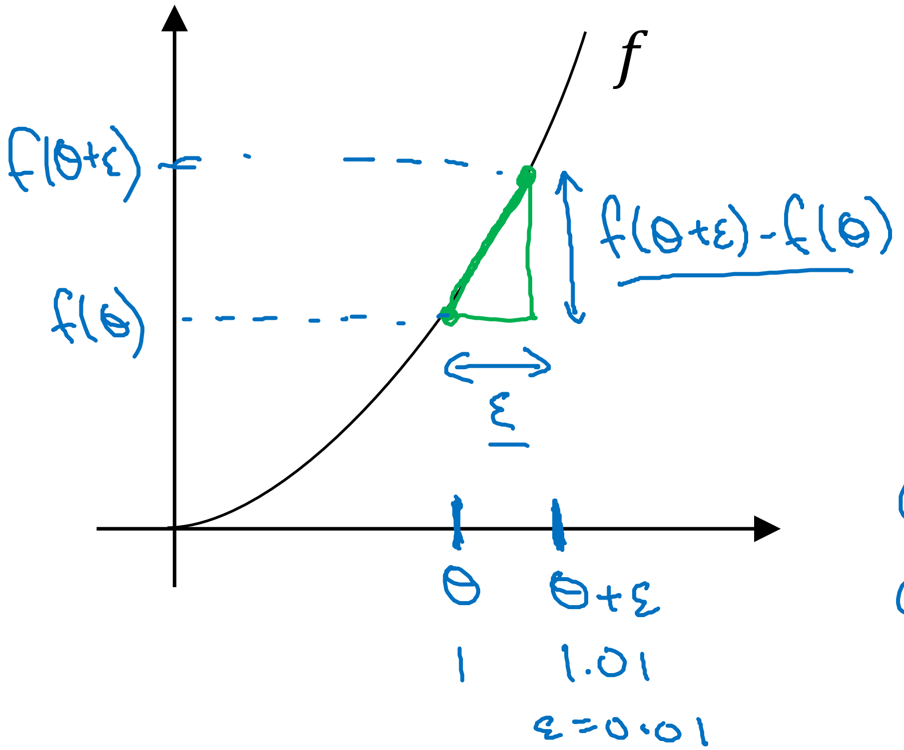
$g(\theta) = 3 \cdot (1)^2 = 3$

when $\theta = 1$

dw

db

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



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

$\theta = 1$

$\theta + \varepsilon = 1.01$

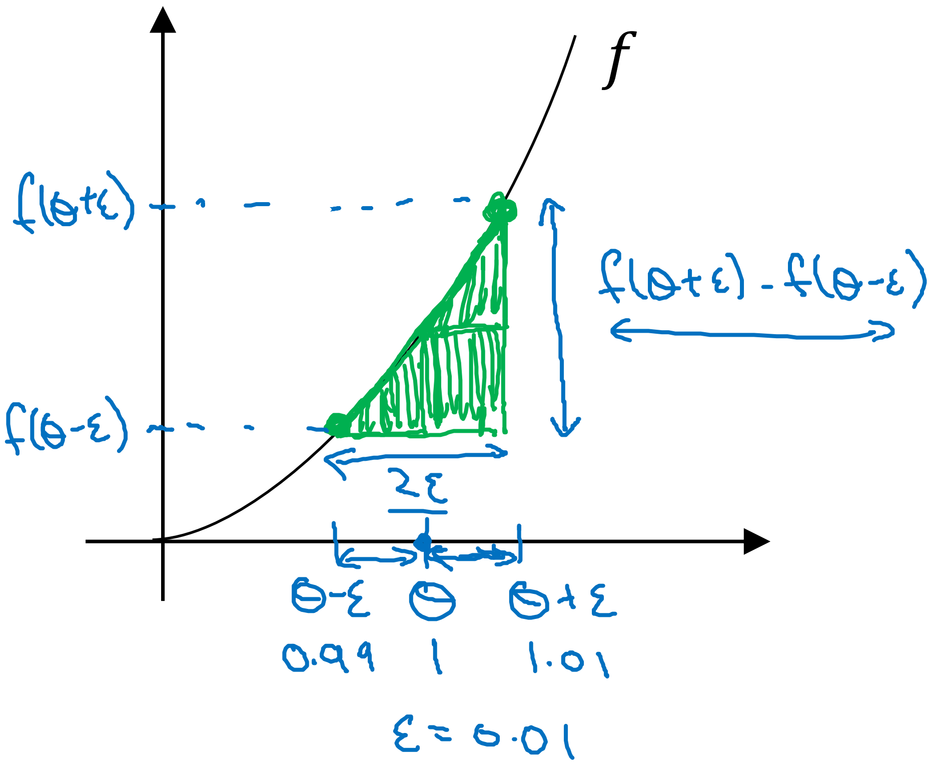
0.0301

3.1

3.2

Checking your derivative computation

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



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

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

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

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

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

0.01

0.0001

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

0.01

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking

Gradient check for a neural network

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

concatenate

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

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

concatenate

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

Andrew Ng

Gradient checking (Grad check)

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

for each i:

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

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

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

Check

$$\rightarrow \frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$$

$\approx \boxed{10^{-7} \text{ - great!}} \leftarrow$

10^{-5}

$\rightarrow 10^{-3}$ - worry. $\leftarrow$

$\varepsilon = 10^{-7}$

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

$d\theta_{approx}[i] \longleftrightarrow d\theta[i]$

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

$db^{[l]}$ $dW^{[l]}$

- Remember regularization.

$J(\theta) = \frac{1}{m} \sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_l \|W^{[l]}\|_F^2$

$d\theta$ = gradient of J wrt. θ

- Doesn't work with dropout.

J keep-prob = 1.0

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

$W, b \approx 0$

Andrew Ng