Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode



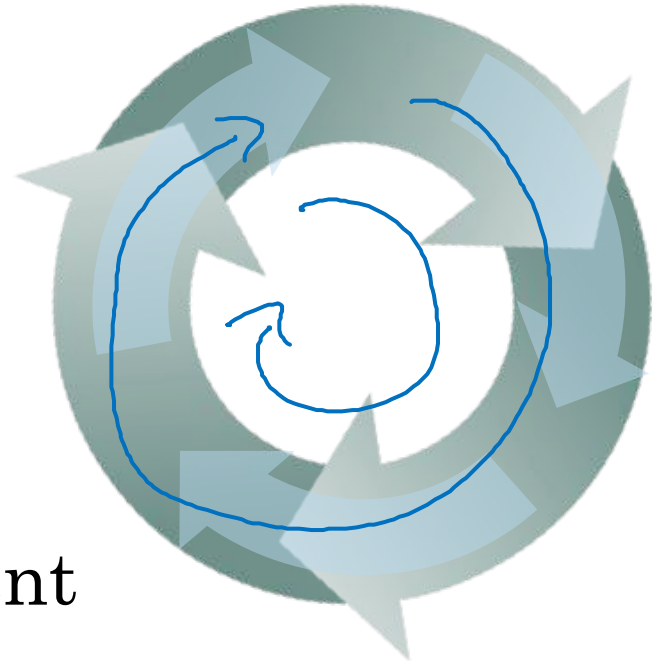
deeplearning.ai

Setting up your ML application

Train/dev/test sets

Applied ML is a highly iterative process

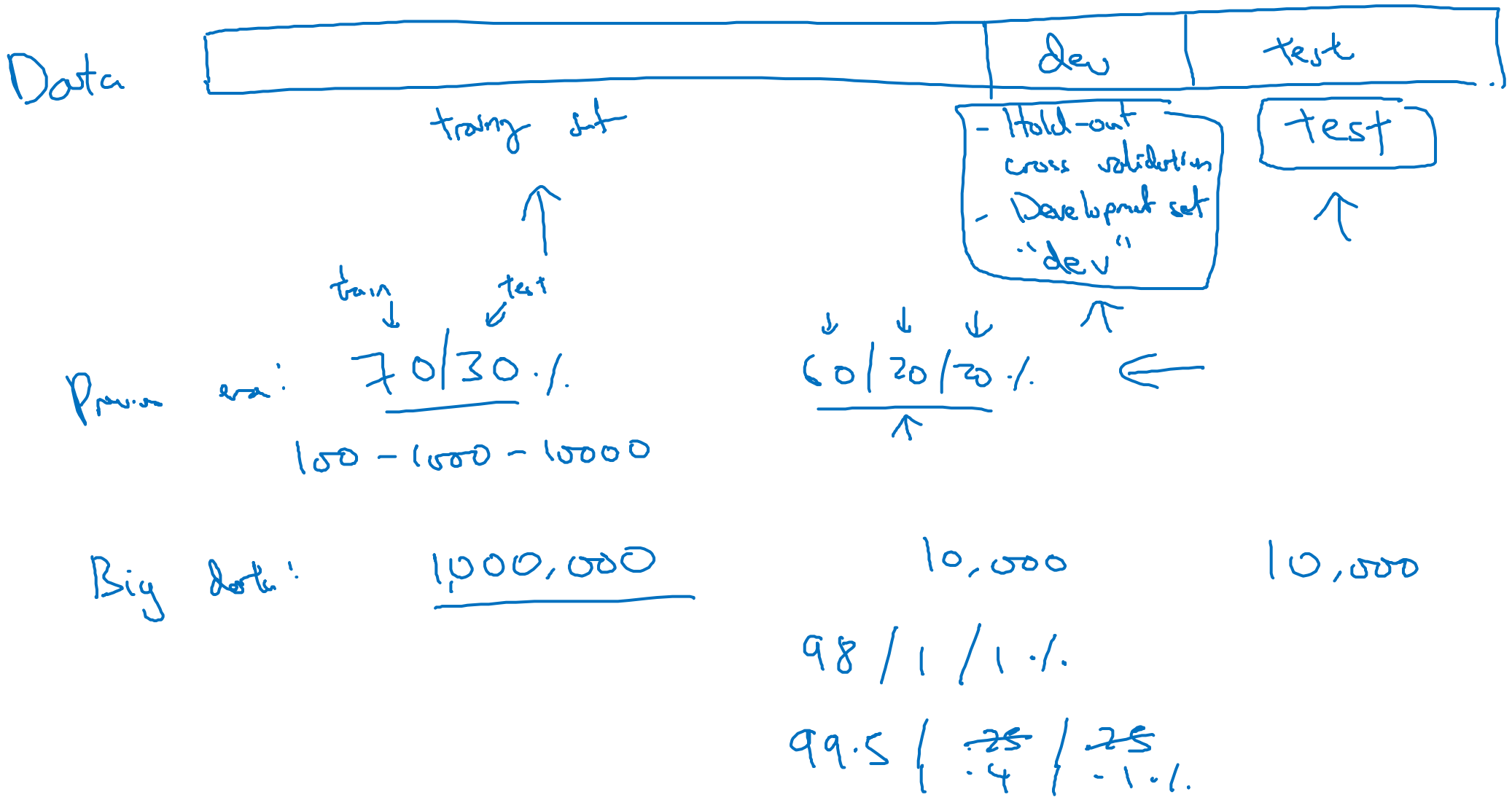
- # layers
- # hidden units
- learning rates
- activation functions
- ...



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.)

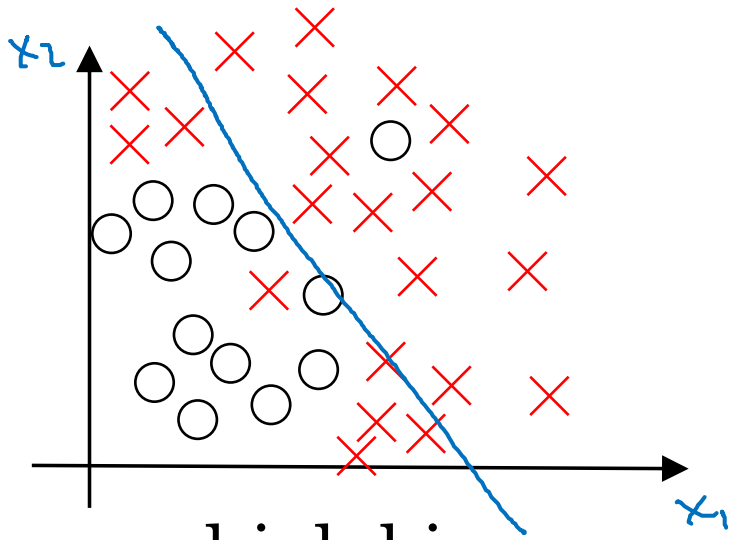


deeplearning.ai

Setting up your ML application

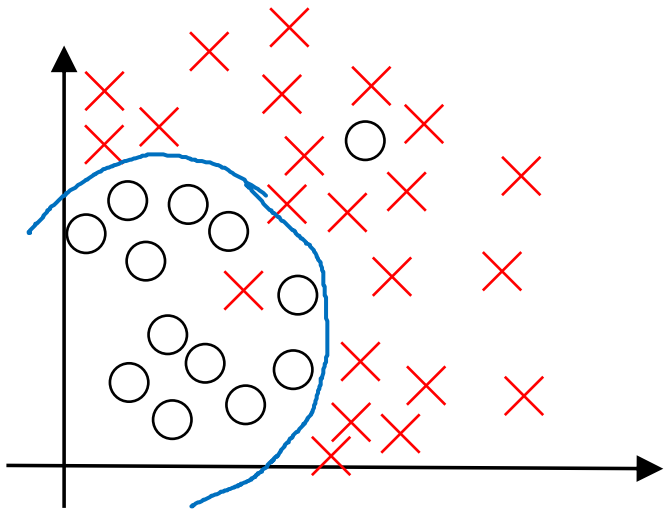
Bias/Variance

Bias and Variance

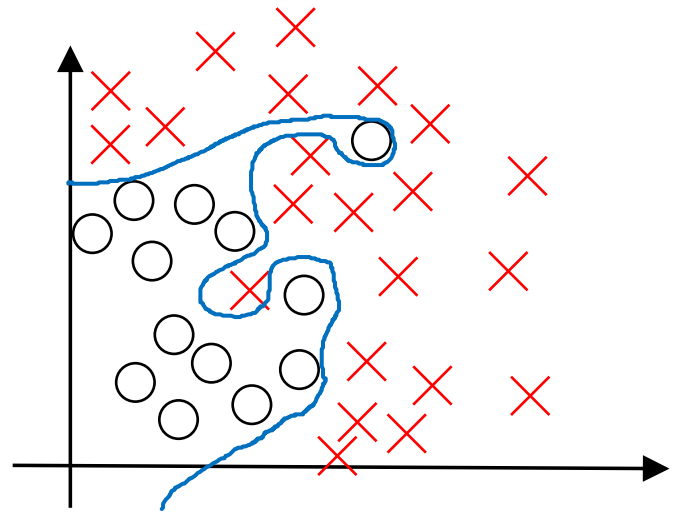


high bias

Underfitting



→ "just right"



high variance

Overfitting

Bias and Variance

Cat classification





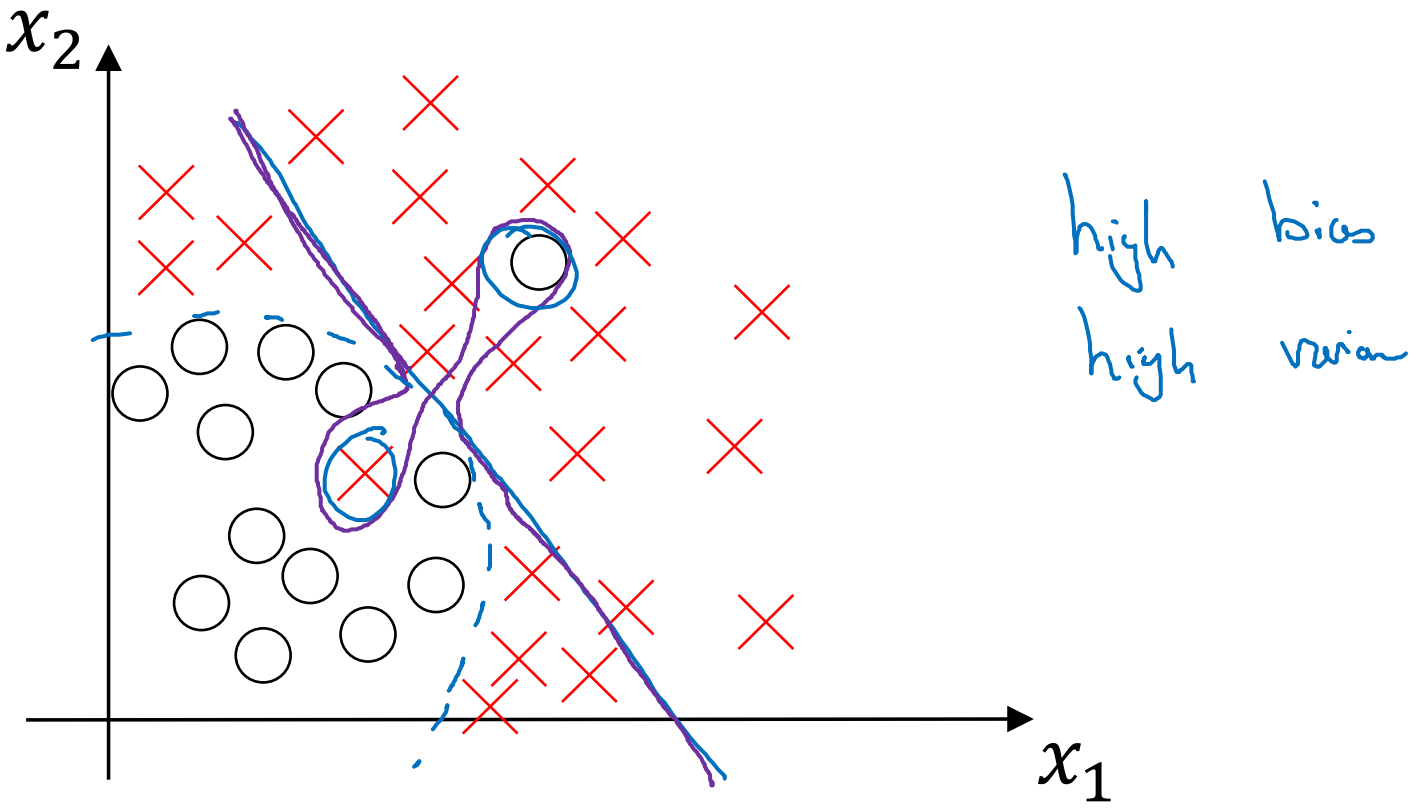
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human: ≈0%

Optimal (Bayes) error: ~~≈0%~~ 15%

Blurry images

High bias and high variance



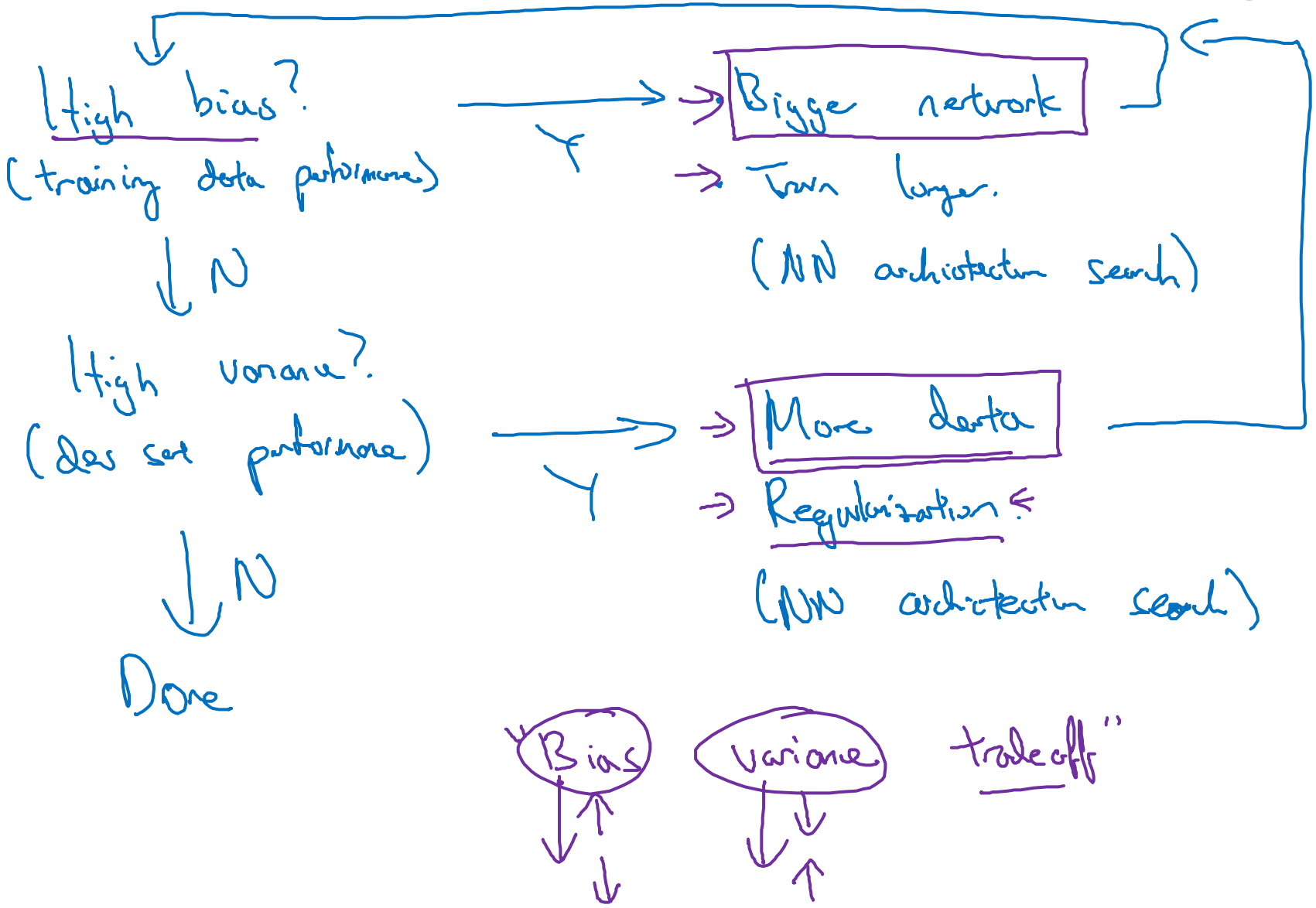


deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning





deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

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

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

λ = regularization parameter

lambda lambd

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

$+ \frac{\lambda}{2m} b^2$

omit

L_2 regularization $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L_1 regularization $\frac{\lambda}{2m}\sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m}\|w\|_1$

w will be sparse

Andrew Ng

Neural network

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

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

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

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

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

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

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

"Weight decay"

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

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

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

Andrew Ng

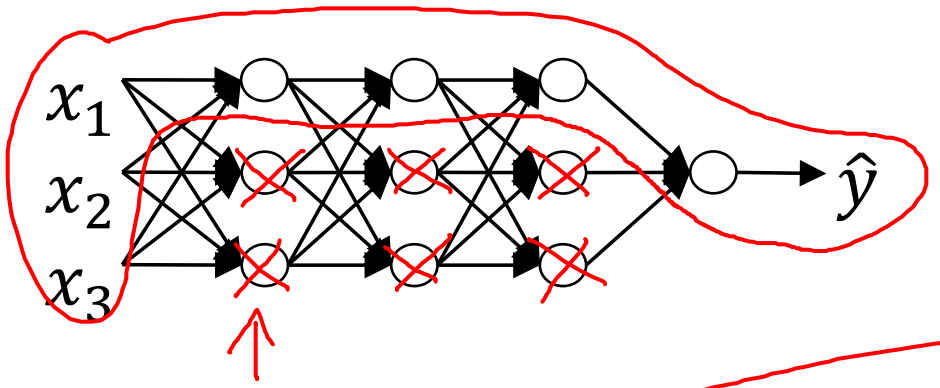


deeplearning.ai

Regularizing your neural network

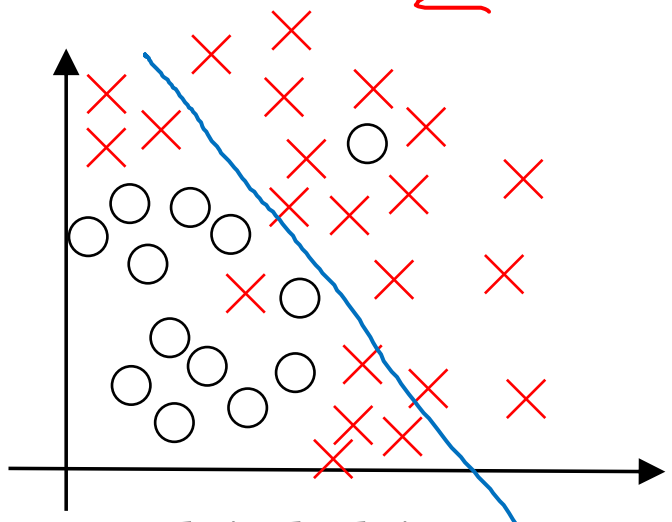
Why regularization reduces overfitting

How does regularization prevent overfitting?

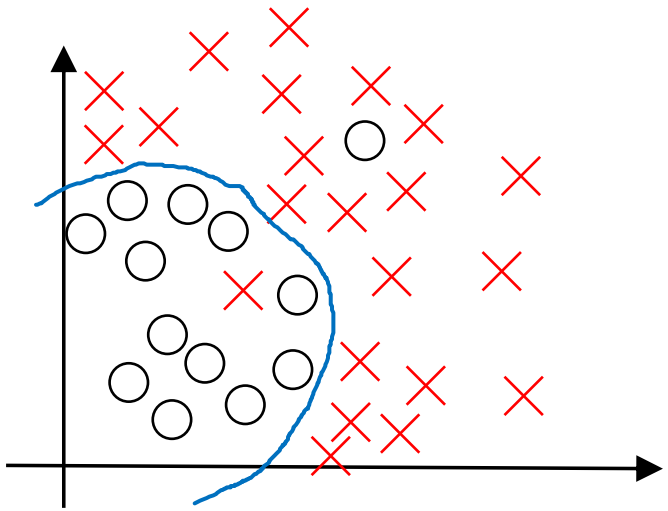


$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

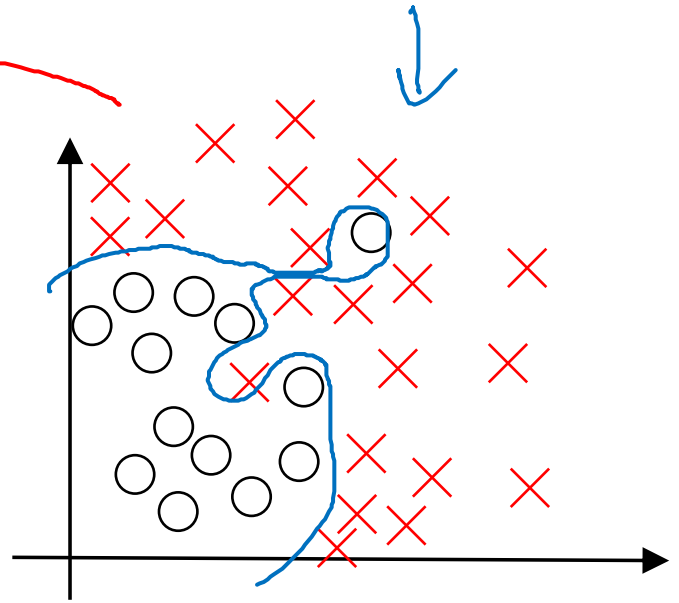
$w^{[l]} \approx 0$



high bias



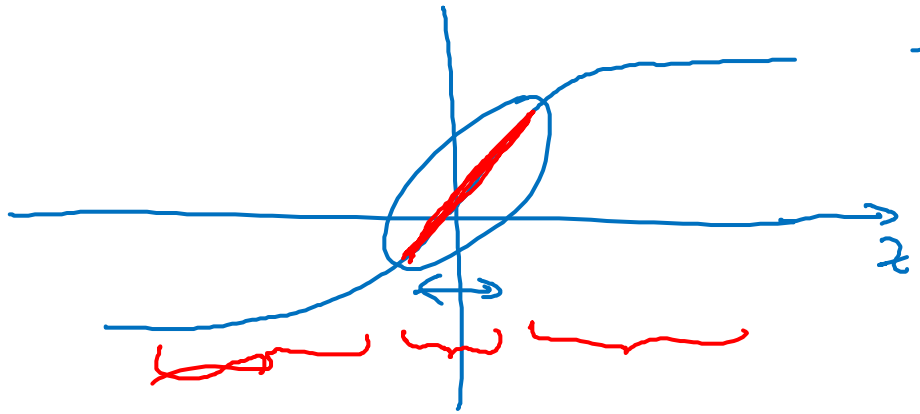
"just right"



high variance

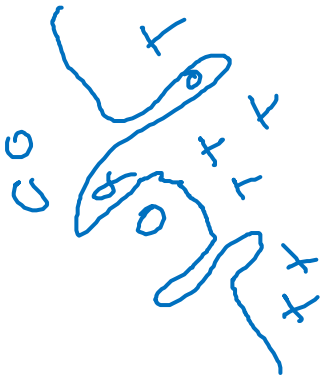
Andrew Ng

How does regularization prevent overfitting?



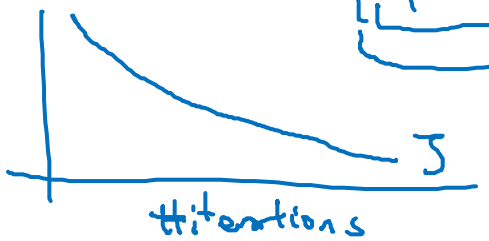
$g(z) = \tanh(z)$

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



Every layer $\approx$ linear.

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



Andrew Ng

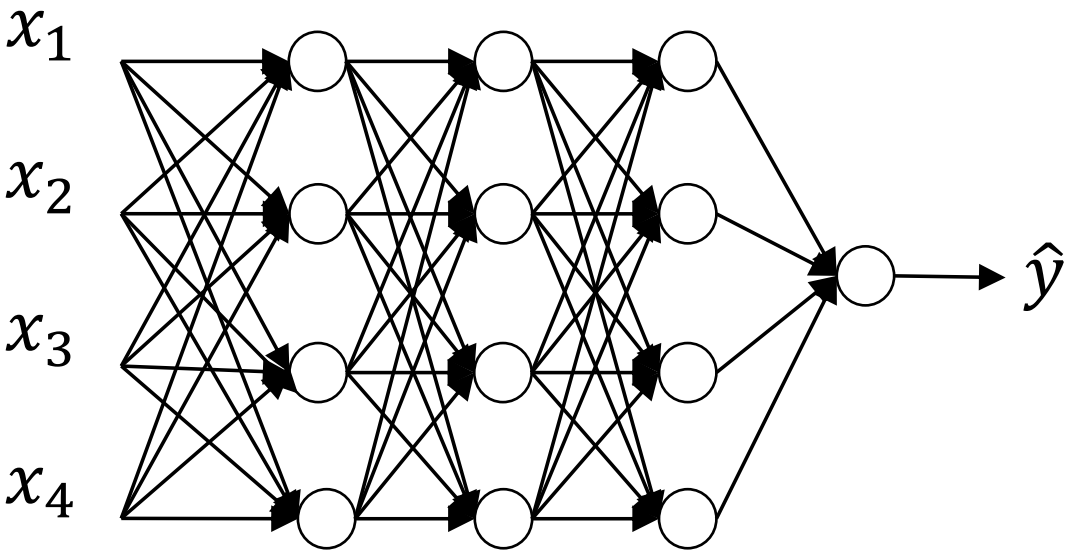


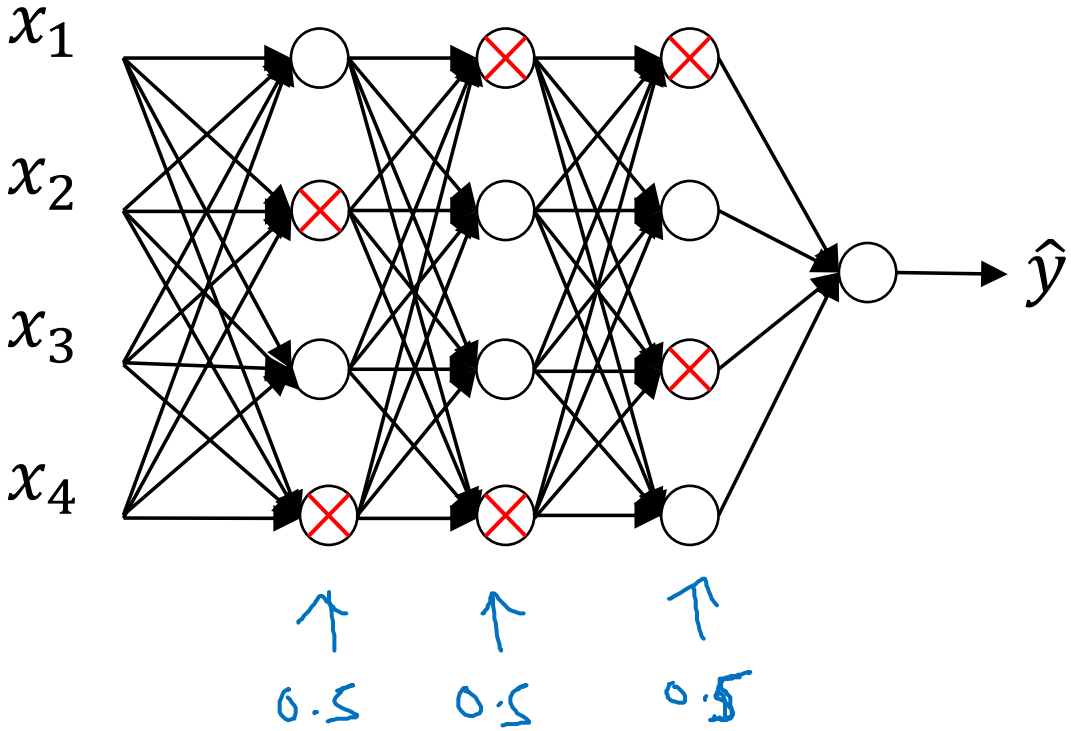
deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization





Implementing dropout ("Inverted dropout")

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

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

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

$a3 /= keep\text{-}prob$

50 units. ⇝ 10 units shut off

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

$a^{[3]}$ reduced by 20%.

$/= 0.8$

Test

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



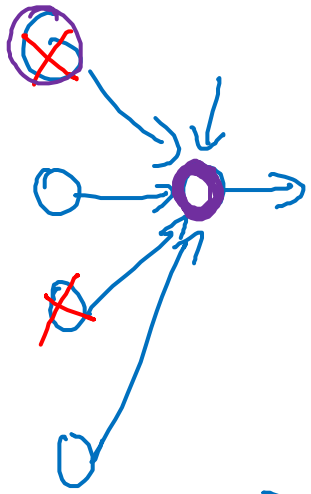
deeplearning.ai

Regularizing your neural network

Understanding dropout

Why does drop-out work?

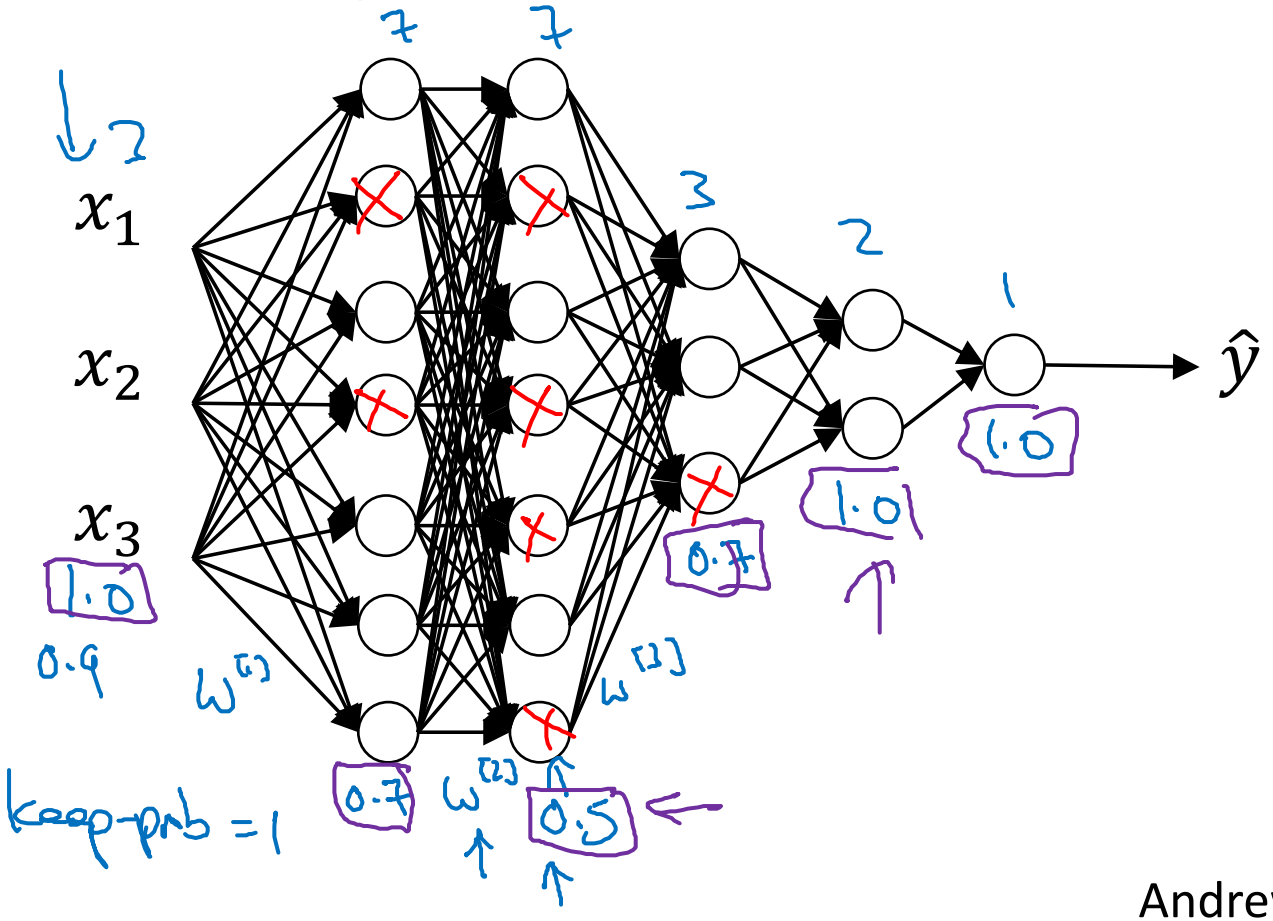
Intuition: Can't rely on any one feature, so have to spread out weights. ⇝ Shrink weights. b_2



→ Computer Vision

Downside: J





Andrew Ng

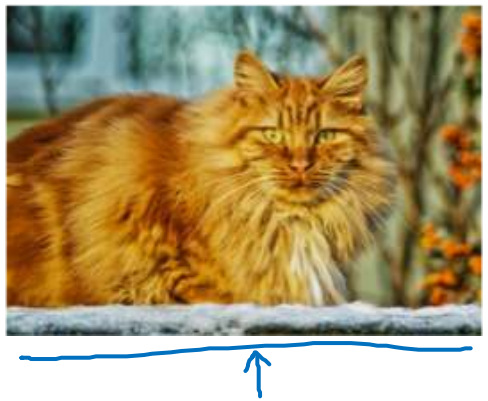


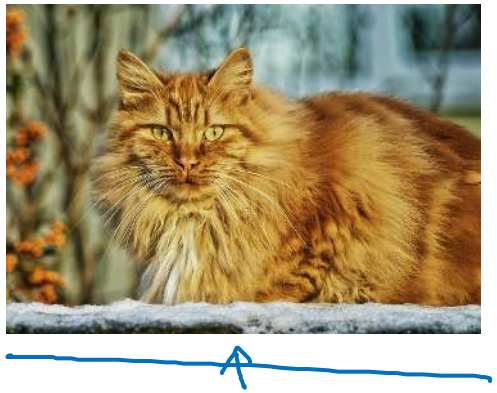
deeplearning.ai

Regularizing your neural network

Other regularization methods

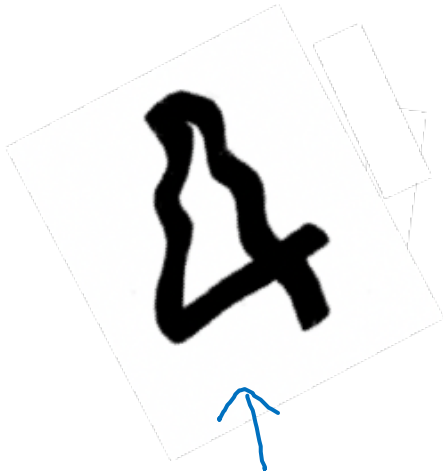
Data augmentation



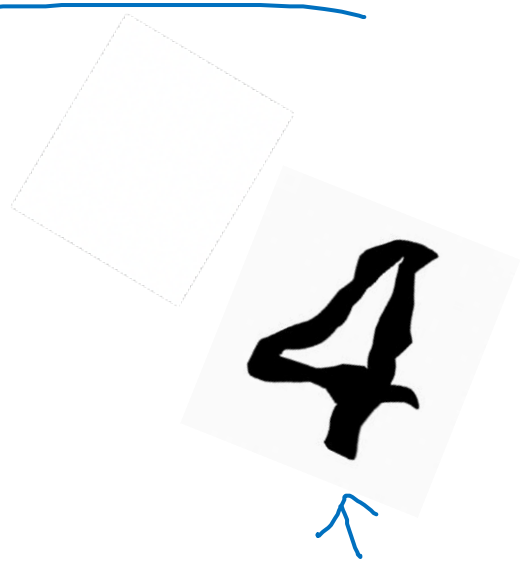




4

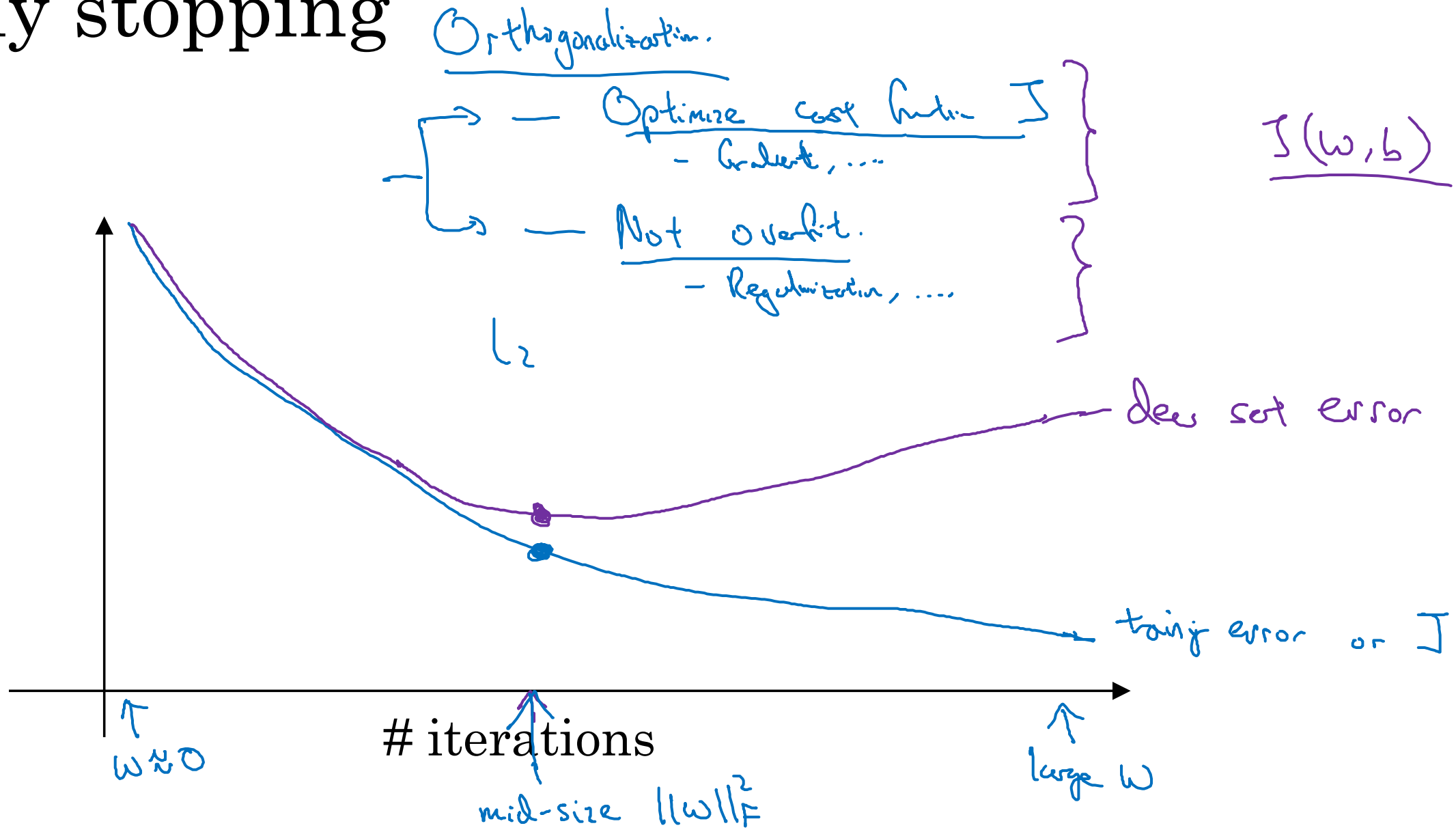






Andrew Ng

Early stopping





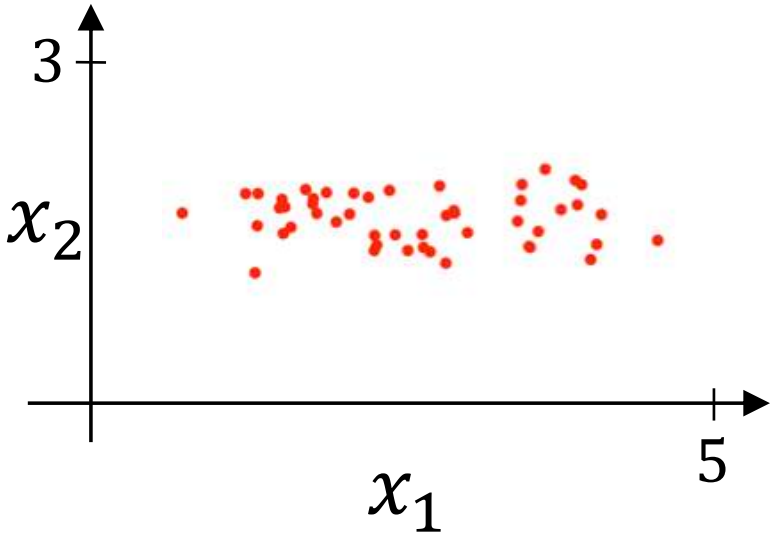
deeplearning.ai

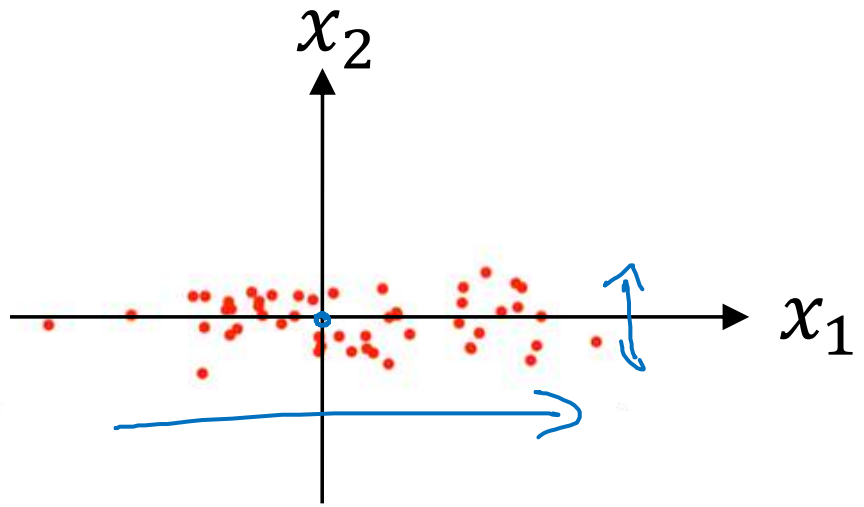
Setting up your optimization problem

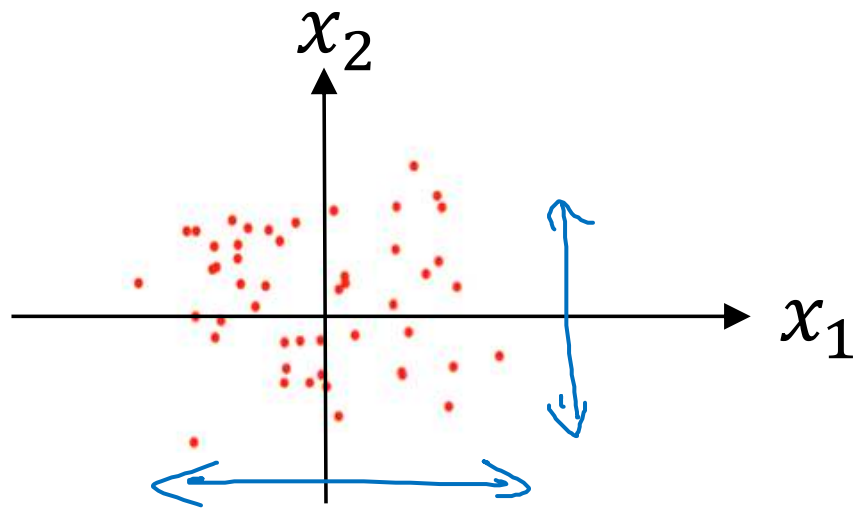
Normalizing inputs

Normalizing training sets

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







Subtract mean:

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

$x := x - \mu$

Normalize variance

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

↑ element-wise

$x /= \sigma^2$

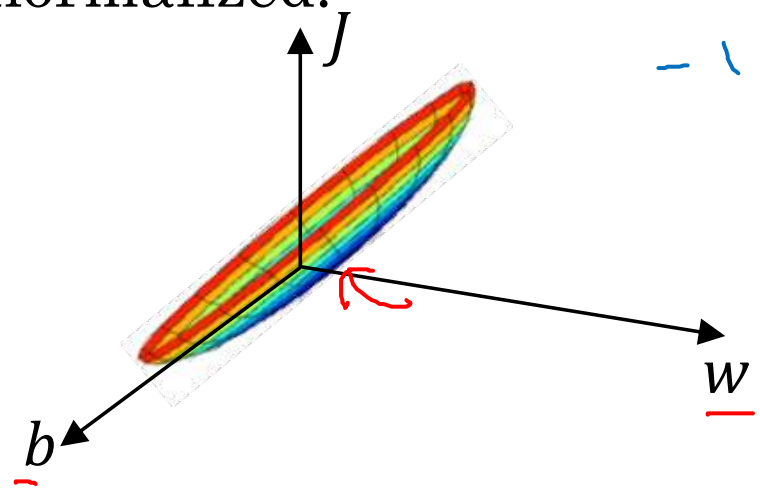
Use same μ, σ^2 to normalize test set.

Andrew Ng

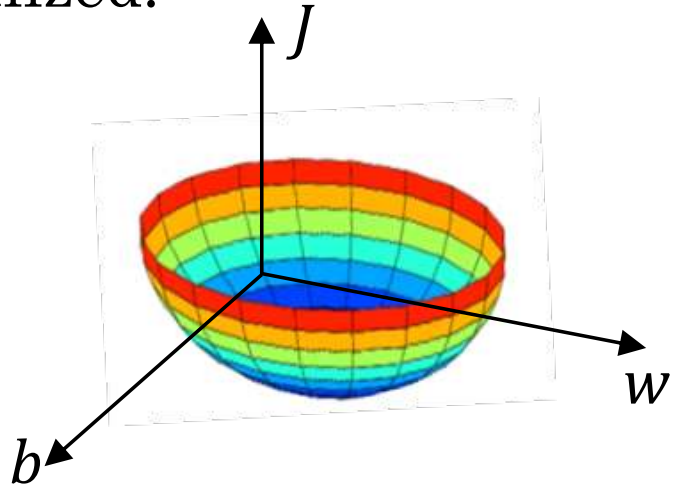
Why normalize inputs?

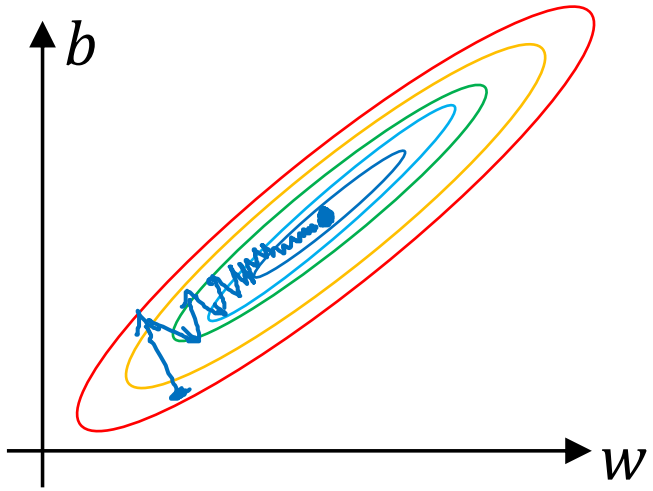
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

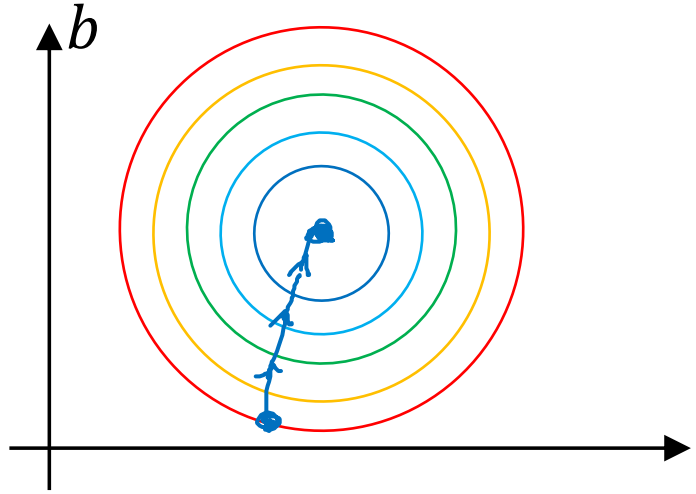
Unnormalized:



Normalized:







Andrew Ng

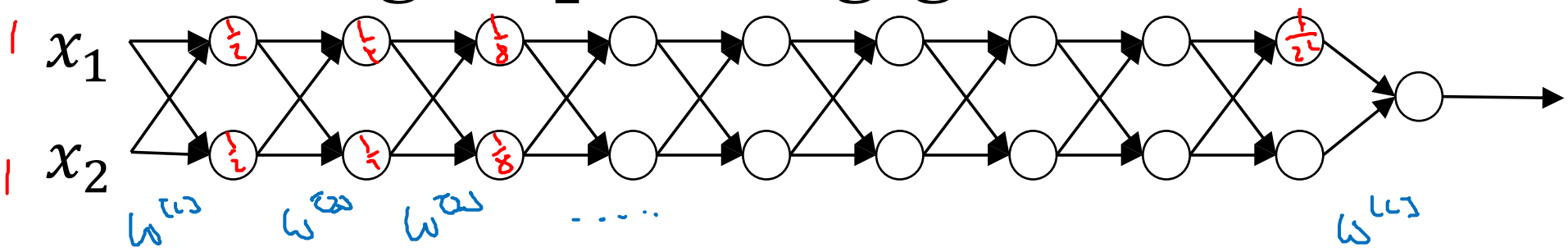


deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients



$L = 150$

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

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

1.5^L

0.5^L

$w^{[l]} > I$

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

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

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

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

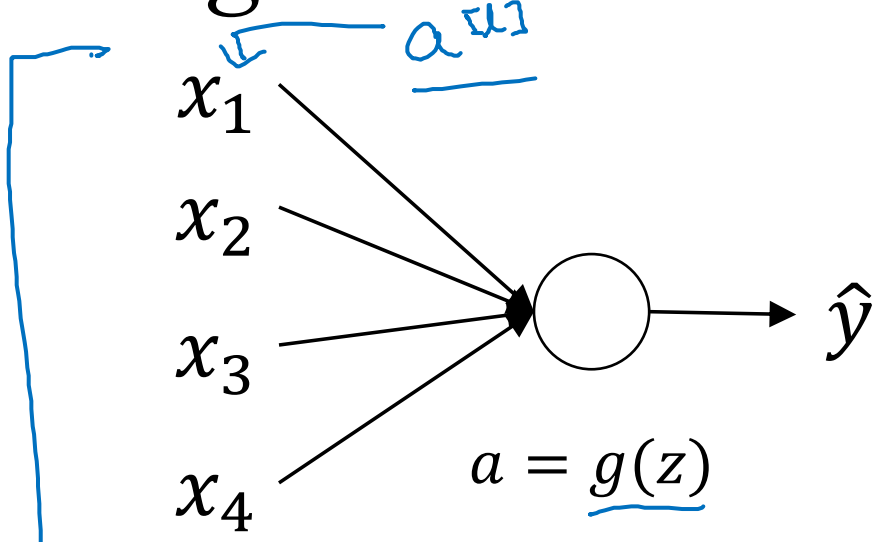
$w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ $\quad 0.5$ $\quad 0.5$

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

$1.5^{L-1} x$

$0.5^{L-1} x$

Single neuron example



$w^{[l]}$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$

large n → smaller w_i

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

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

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

Other variants:

tanh

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

Xavier initialization

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

Andrew Ng



deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

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

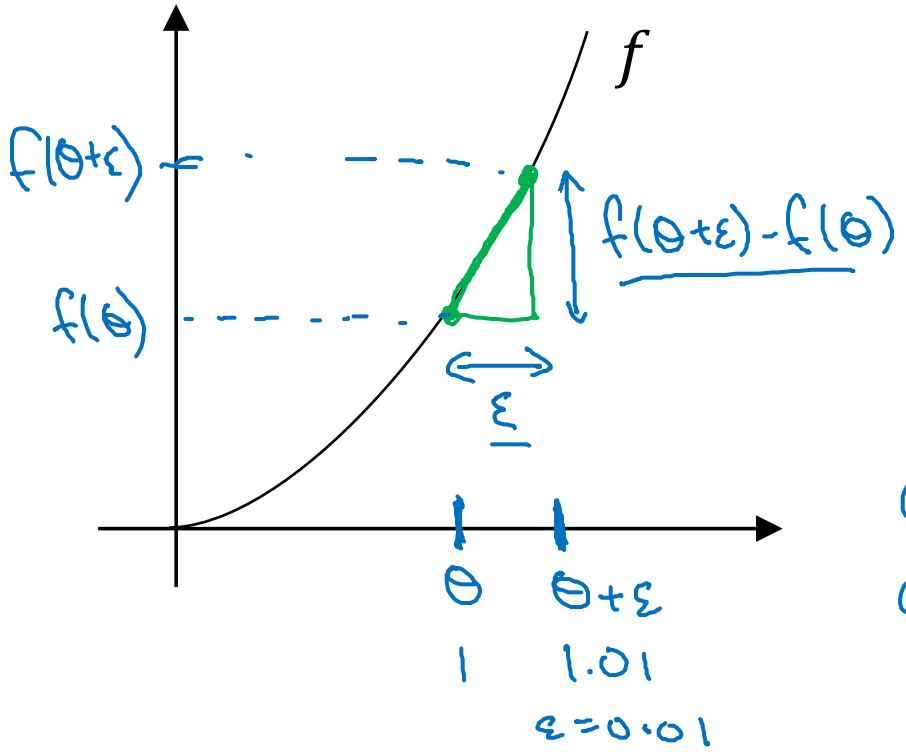
$\theta \in \mathbb{R}$.

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

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

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

$\frac{dw}{db}$



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

$\theta = 1$

$\theta + \varepsilon = 1.01$

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

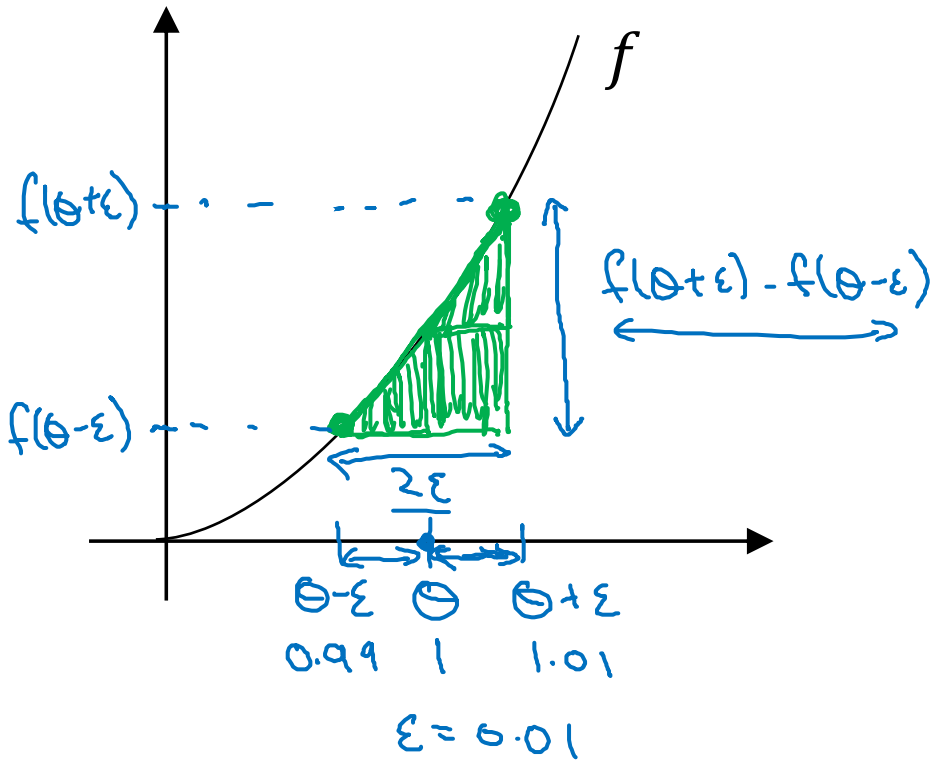
0.0301

3.1

3.2

Checking your derivative computation

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



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

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

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

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

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

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 $J(\theta)$?

Gradient checking (Grad check)

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

for each i:

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

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

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

Check $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$ $\varepsilon = 10^{-7}$

$\approx 10^{-7}$ — 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

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

- 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.

$W, b \approx 0$