Copyright Notice

PA-1 : Parameters Initialisation 1-In general, initializing all the weights to zero results in the network failing to break symmetry. This means that every neur

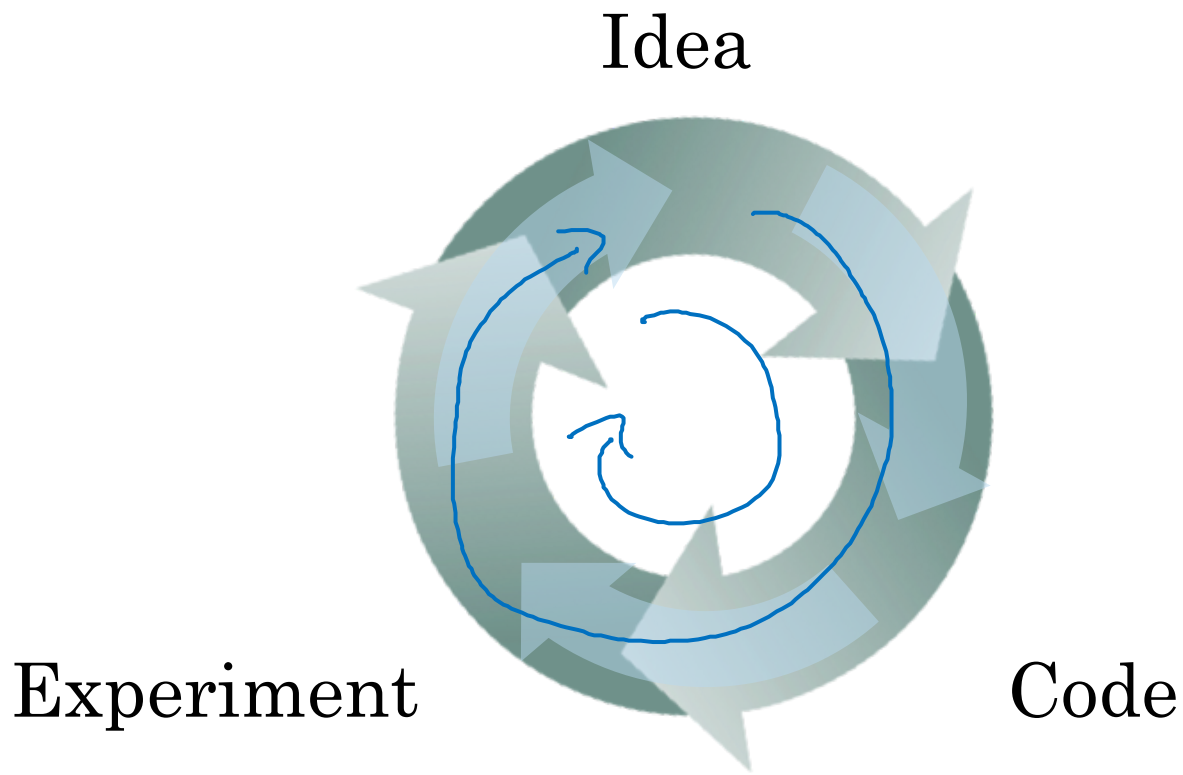
deeplearning.ai

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 logistics ....

Andrew Ng

Train/dev/test sets

Data | training set | dev | test

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

test

Previous era: train / test
70/30 %
60/20/20 %
100 - 1000 - 10000

If you have small dataset then it is OK to spililt in the ratio of 60/20/20 and something like

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

98/1/1 %

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

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.)

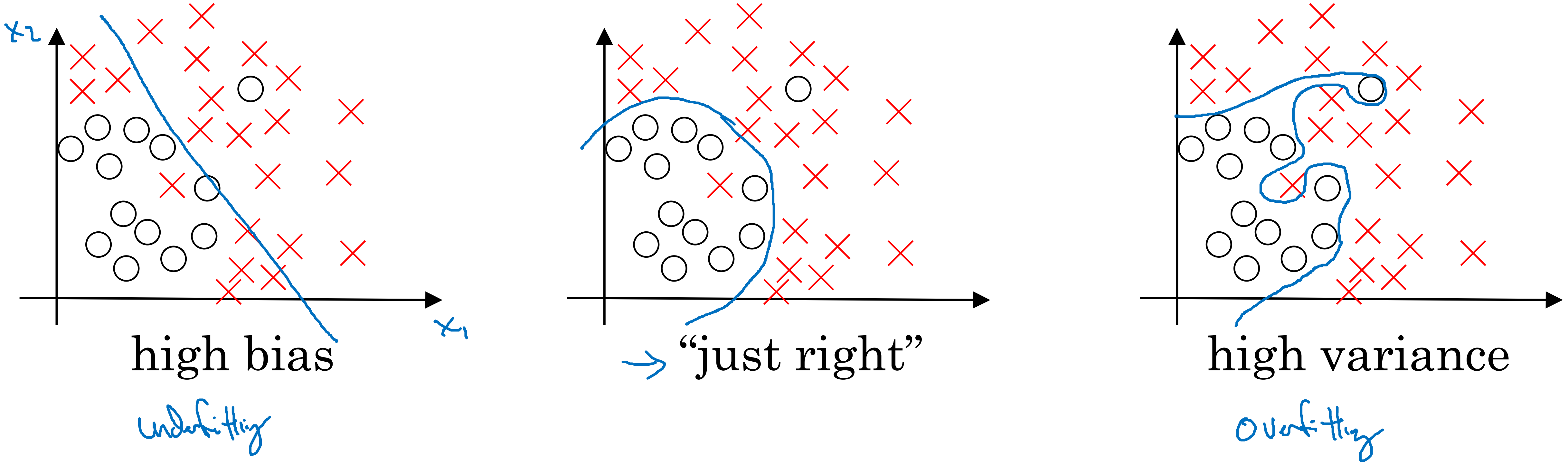
Andrew Ng

deeplearning.ai

Setting up your ML application

Bias/Variance

Bias and Variance



O and V paas paas hain

Andrew Ng

Bias and Variance

Cat classification

$y=1$

$y=0$

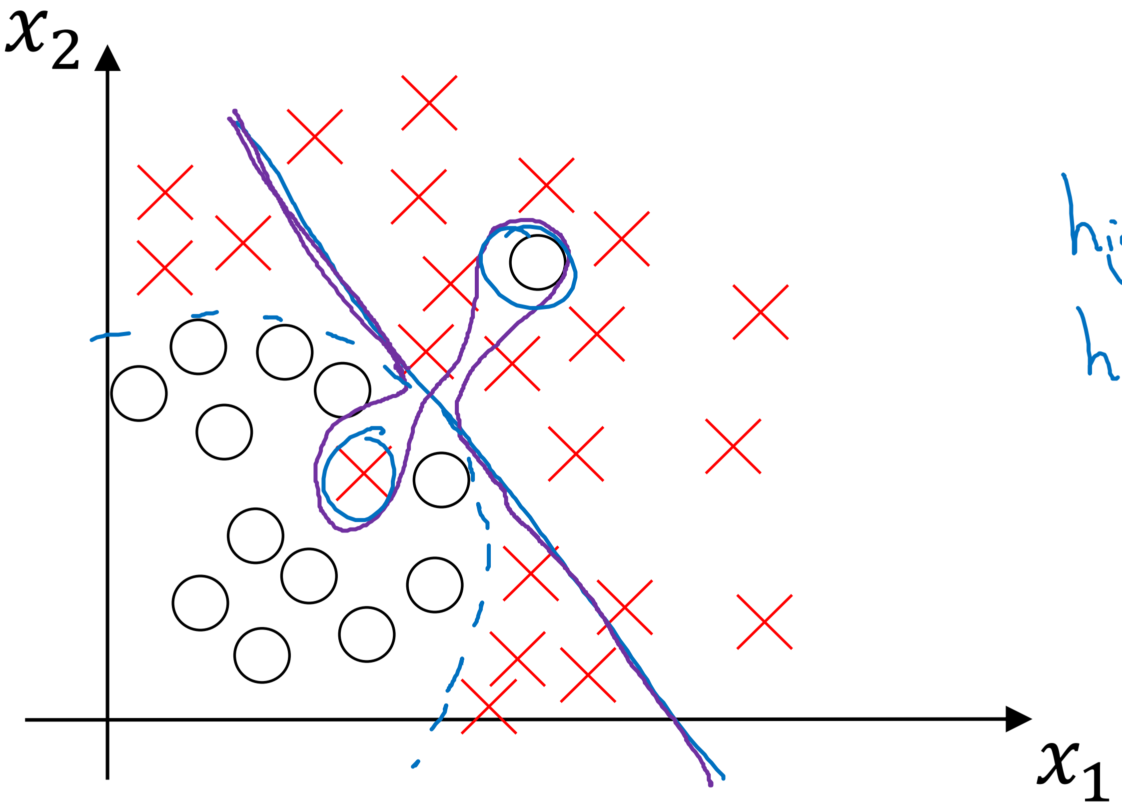
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high varian	low bias low variance

Human : ≈ 0%

Optimal (Bayes) error : ~~≈ 0%~~ 15%   Blurry images

Andrew Ng

High bias and high variance



high bias
high varian

deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning

High bias?
(training data performance)

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

↓ N

High variance?
(dev set performance)

Y → More data
→ Regularization
→ (NN architecture search)

↓ N

Done

Bias variance "tradeoff"

In earlier times there used to be trade-off but now w

Regularizing your neural network

Regularization

deeplearning.ai

Logistic regression

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

$\boxed{\lambda = \text{regularization parameter}}$

lambda lambd

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

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

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

omit

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

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

w will be sparse

Neural network

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

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

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

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

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

"Weight decay"

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

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

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

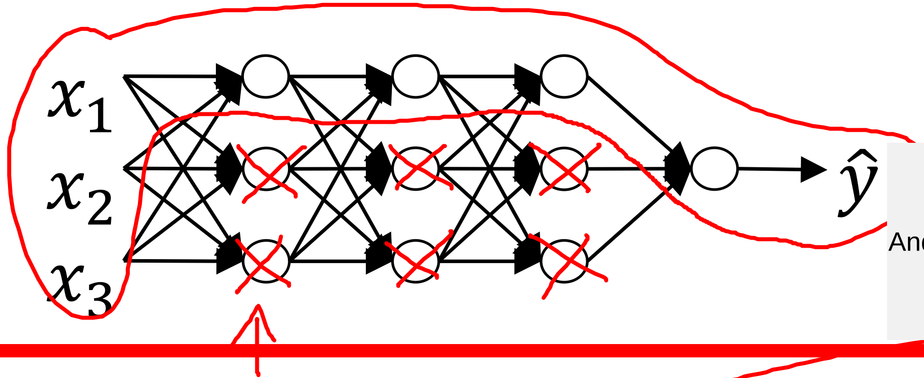
Andrew Ng

deeplearning.ai

Regularizing your neural network

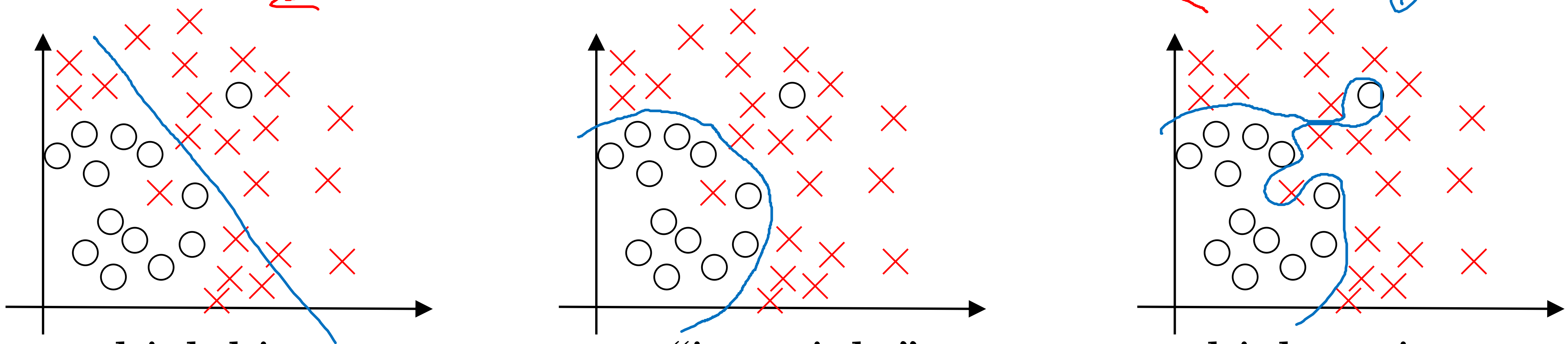
Why regularization reduces overfitting

How does regularization prevent overfitting?



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

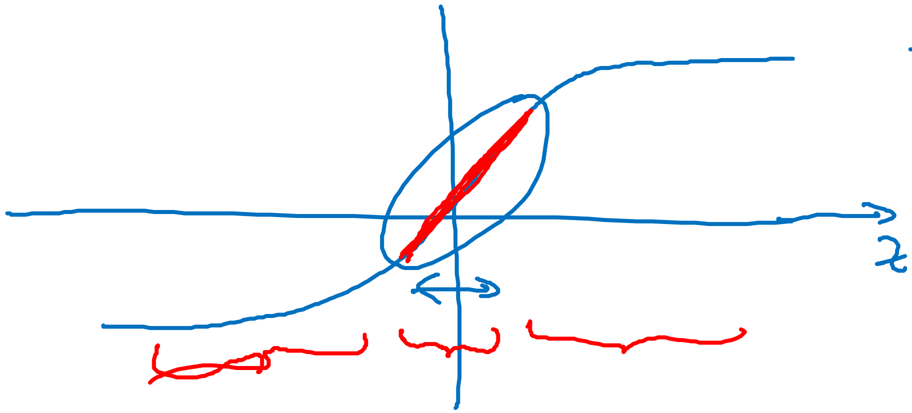
And so what we did for regularization was add this extra term thatpenalizes the weight matrices from being too large.And we said that v



high bias "just right" high variance

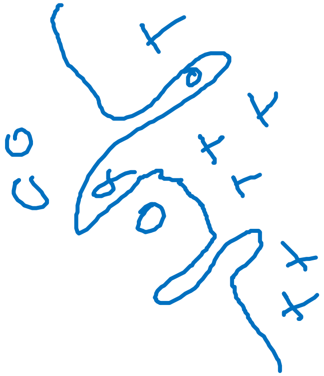
Andrew Ng

How does regularization prevent overfitting?



tanh

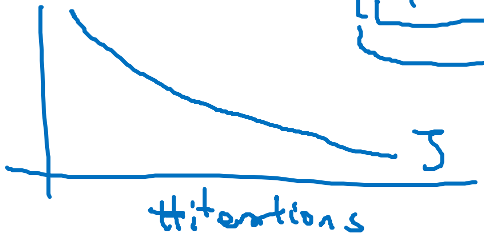
$g(z) = \tanh(z)$



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

Every layer $\approx$ linear.

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



Andrew Ng

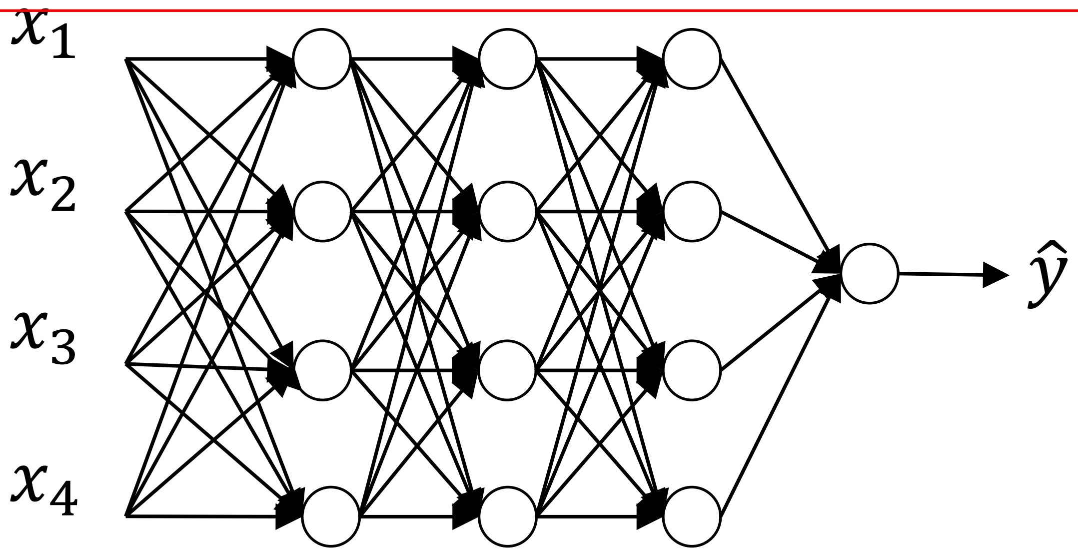
deeplearning.ai

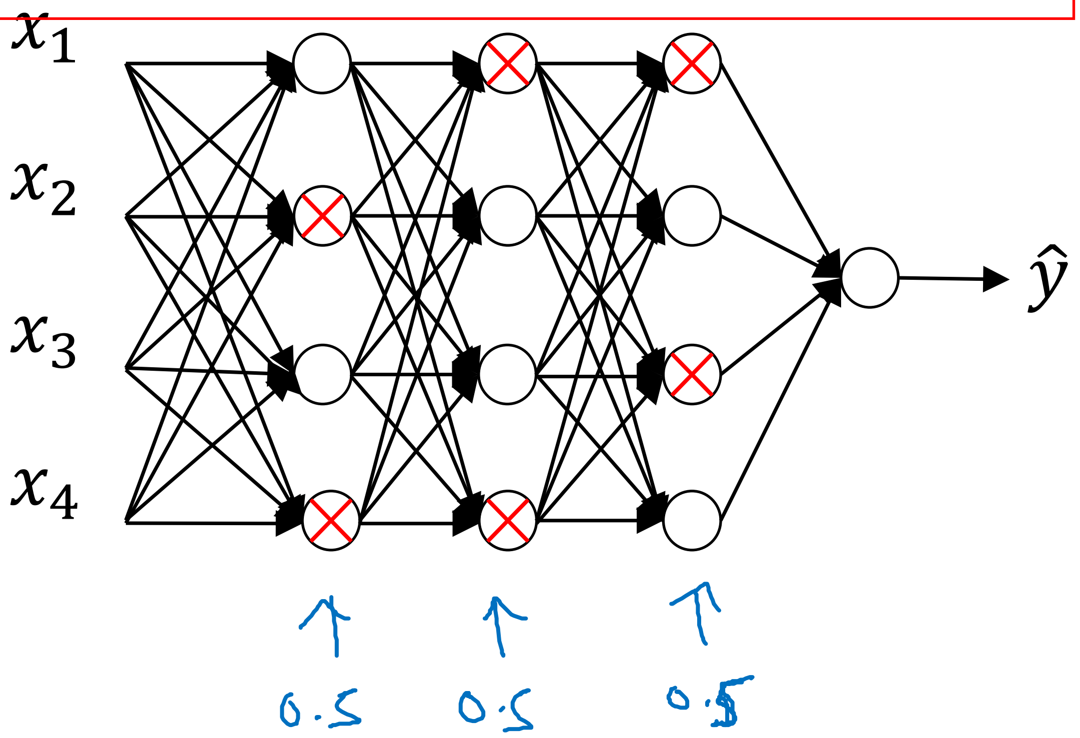
Regularizing your neural network

Dropout regularization

Dropout regularization

what we do we assoctiate a probability with each node denoting its probability of being dropped out. then we actiallly drop those nodes and do our traini





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

$a^{[0]} = X$

$\boxed{\text{No drop out.}}$

No drop out at the testong time.

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

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

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

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

$\downarrow$

$\hat{y}$

$/= \text{keep-prob}$

deeplearning.ai

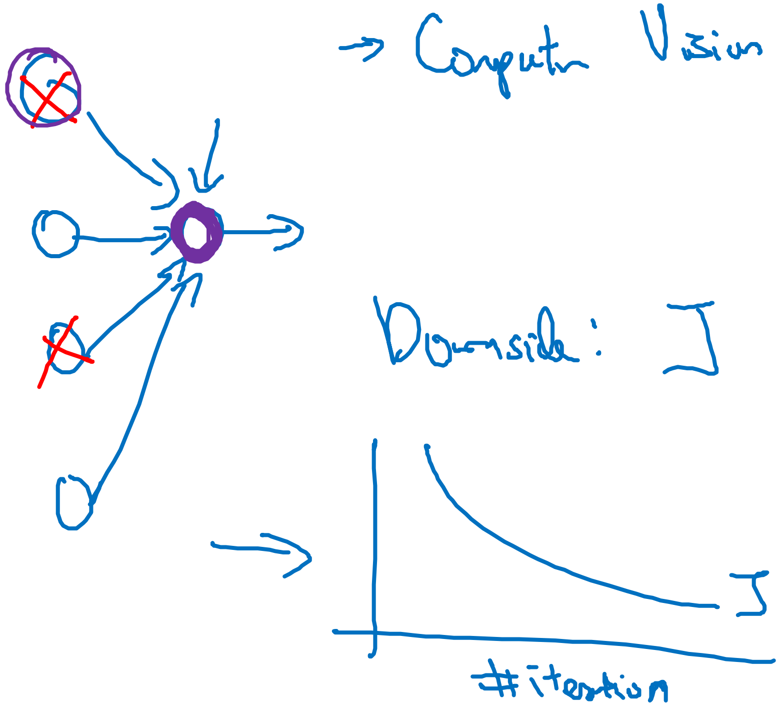
Regularizing your neural network

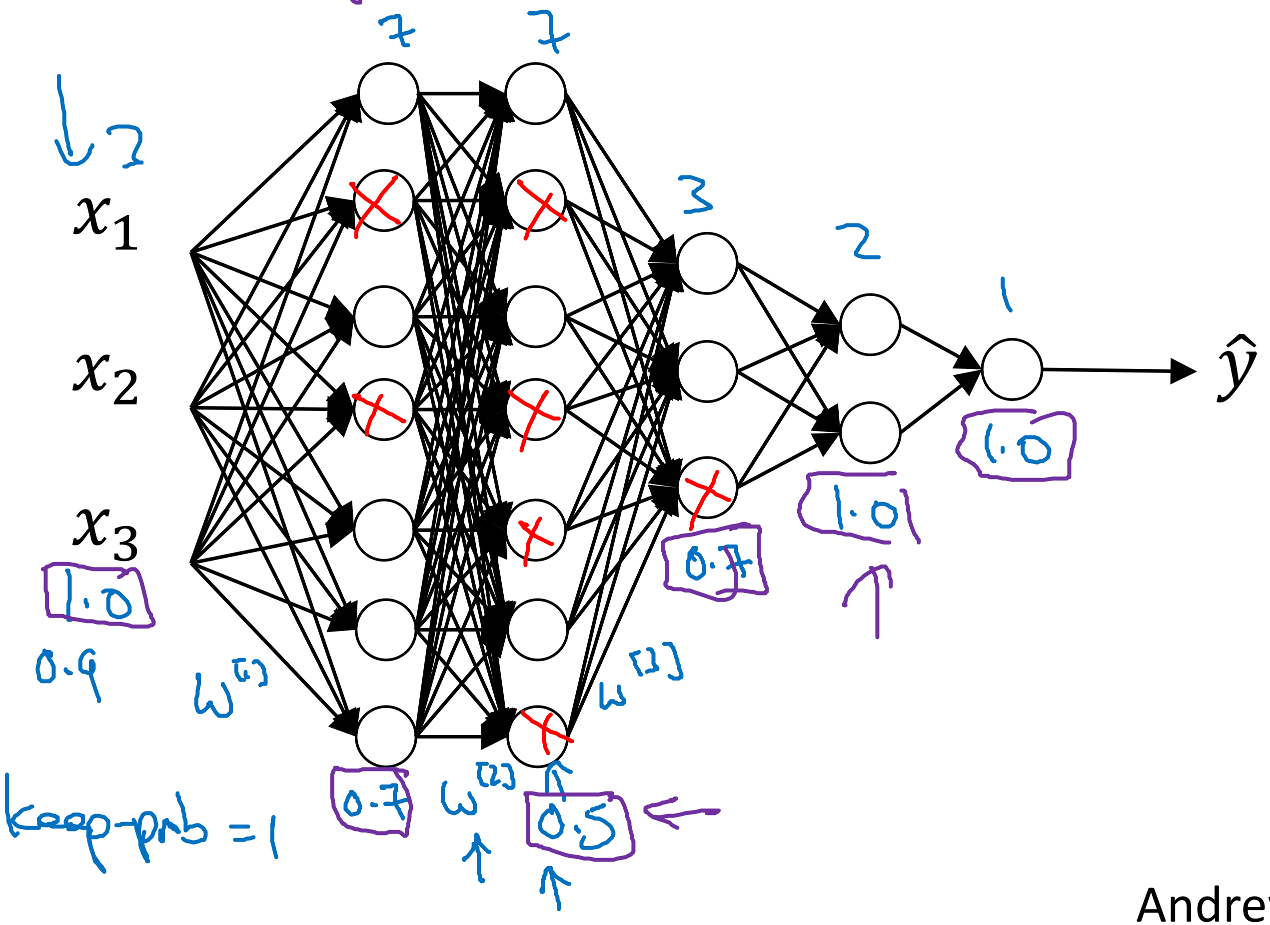
Understanding dropout

Why does drop-out work?

Jahan par overfitting ka dar lage, ya parameters jyada hon waha

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





Andrew Ng

deeplearning.ai

Regularizing your neural network

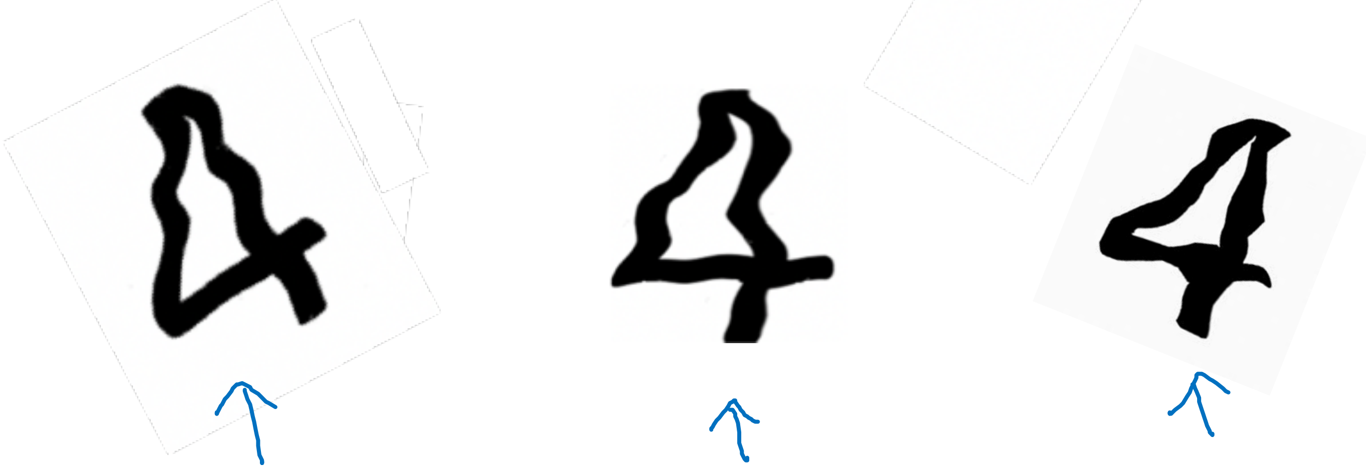
Other regularization methods

Data augmentation

Jo data hai usi ko ghuma phira ke naya data bana do. Kahin se aur dataa mil jaay

4

Augument karo



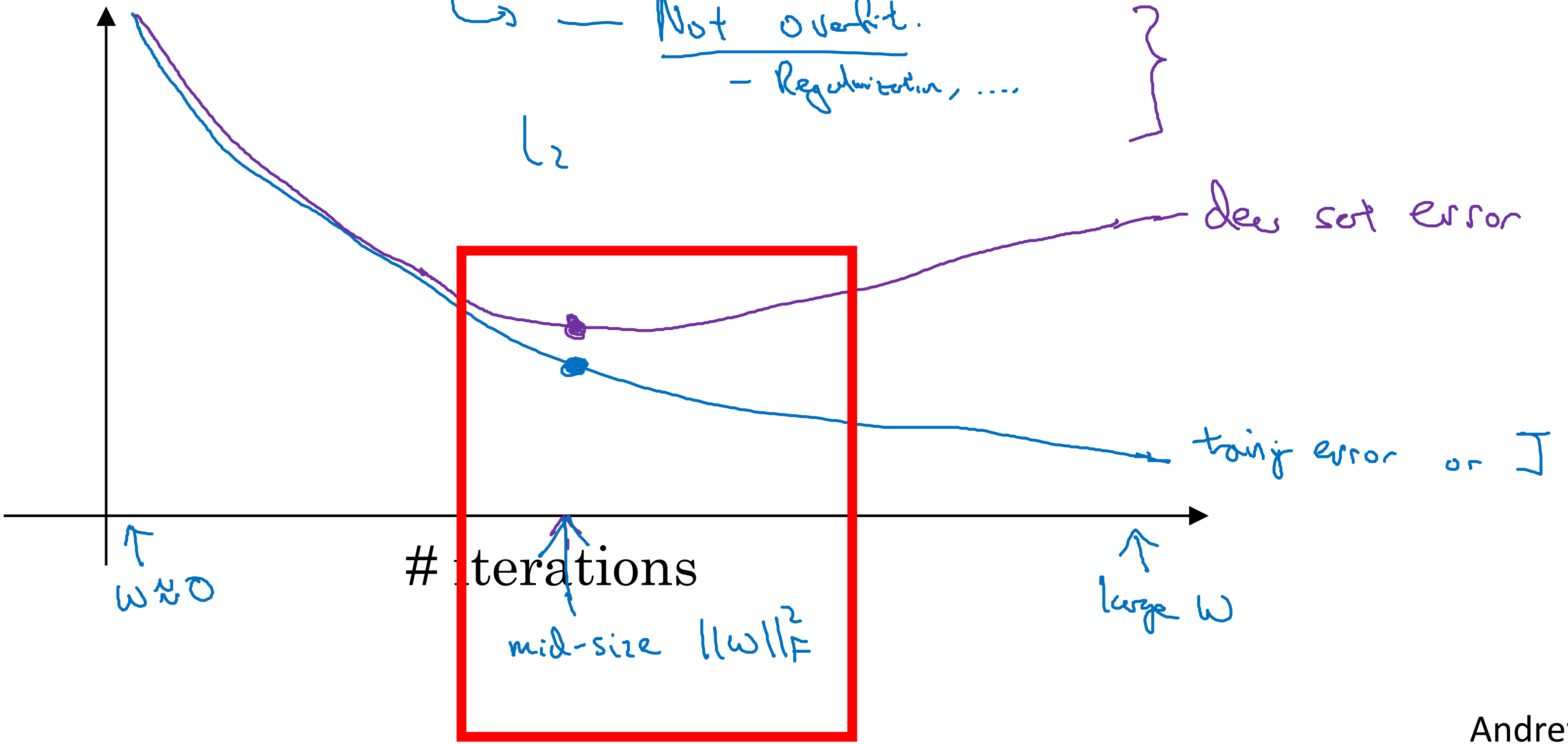
Early stopping

Orthogonalization.

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

$J(w,b)$

L_2



Andrew Ng

deeplearning.ai

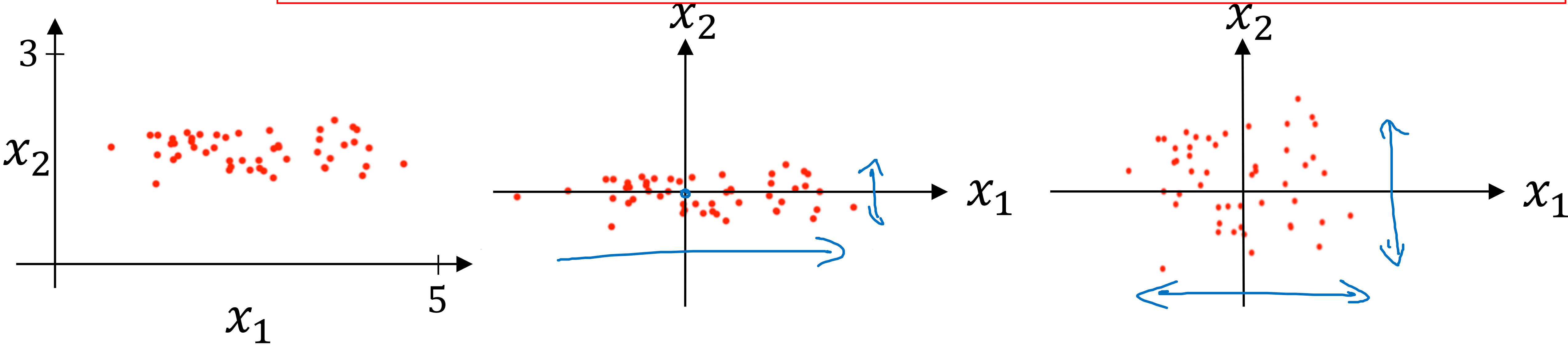
Setting up your optimization problem

Normalizing inputs

Normalizing training sets

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

Normalizing involves two steps. The first is two is subtract out means from every training set and then to normalize the variance



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

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

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

Andrew Ng

Why normalize inputs?

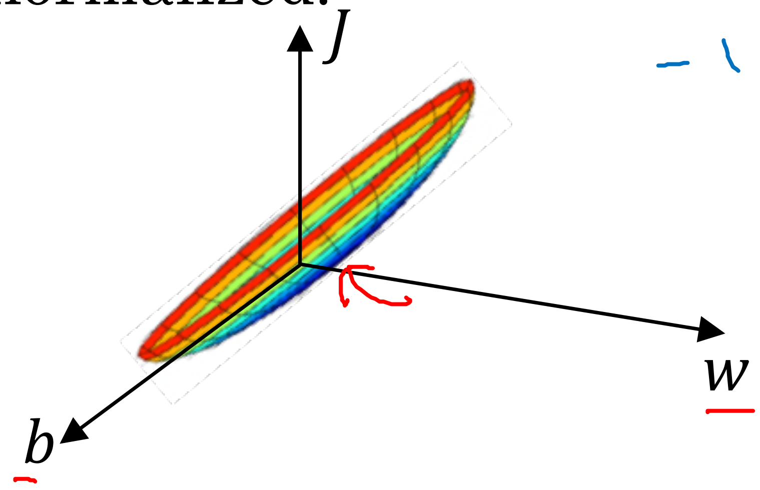
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

w_1 x_1 : 1 . . . . 1000 ←

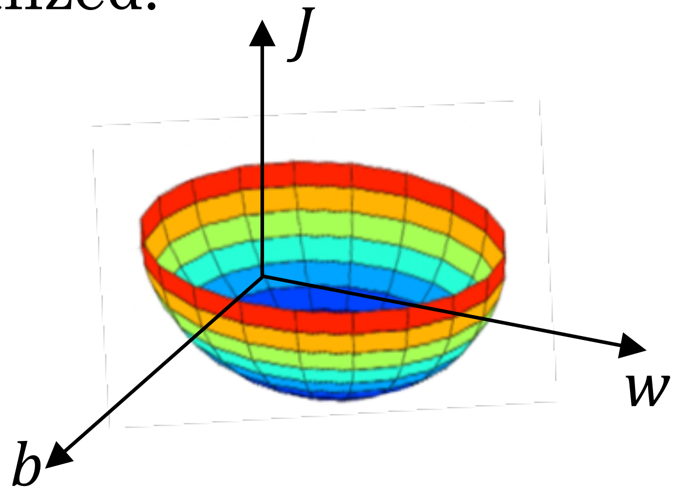
w_2 x_2 : 0 . . . 1 ←

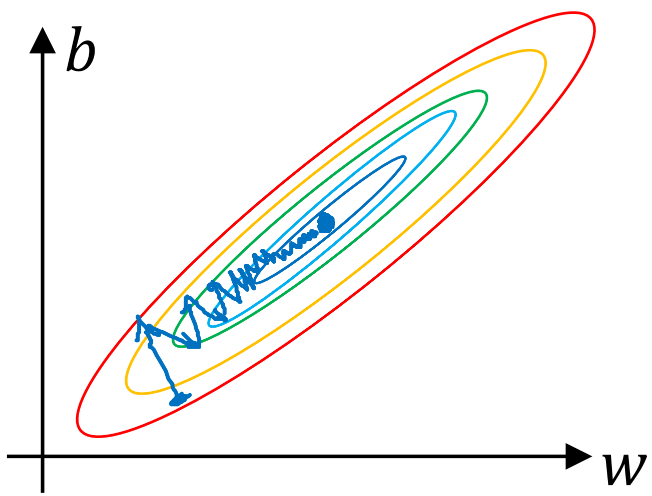
−1 . . . 1

Unnormalized:



Normalized:

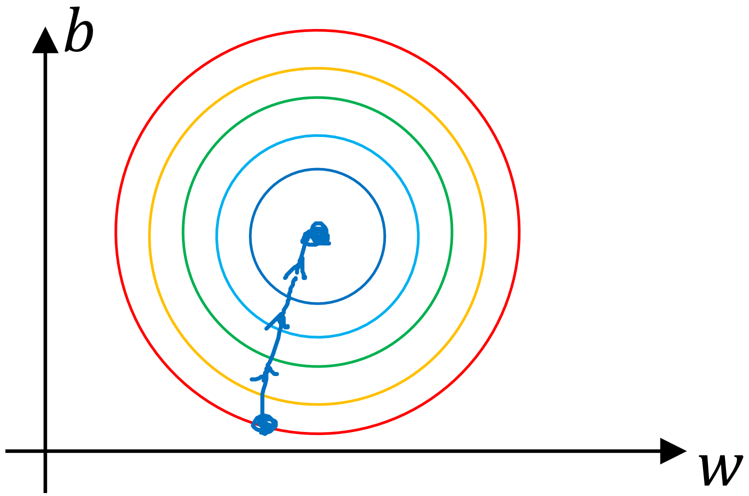




x_1 : 0 . . . 1

x_2 : −1 . . 1

x_3 : 1 . . 2



Andrew Ng

What that means is that when you're training a very deep network your derivatives or your slopes can sometimes get either very, very big or very, very sm

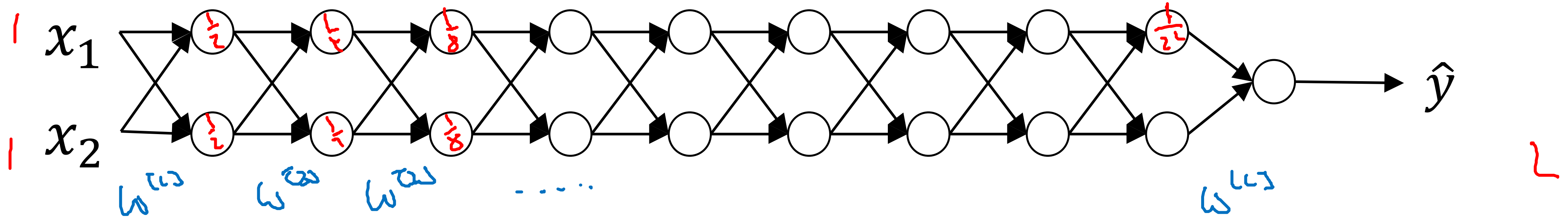
deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients

$L = 150$



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

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

1.5^L

0.5^L

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

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

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

$W^{[l]} > I$

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

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

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

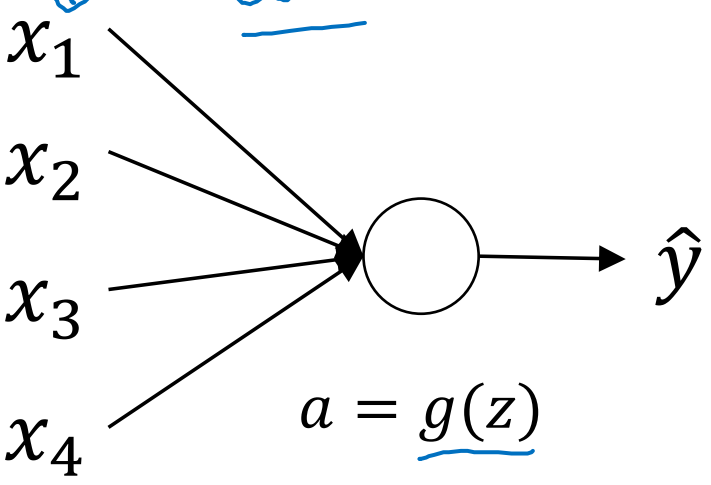
$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example

$a^{[l]}$



$w^{[l]}$

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

Large n → Smaller w_i

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

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

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

Other variants:

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

$\boxed{\text{Xavier initialization}}$

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

for other than relu, numerator is 1

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

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

J

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

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

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

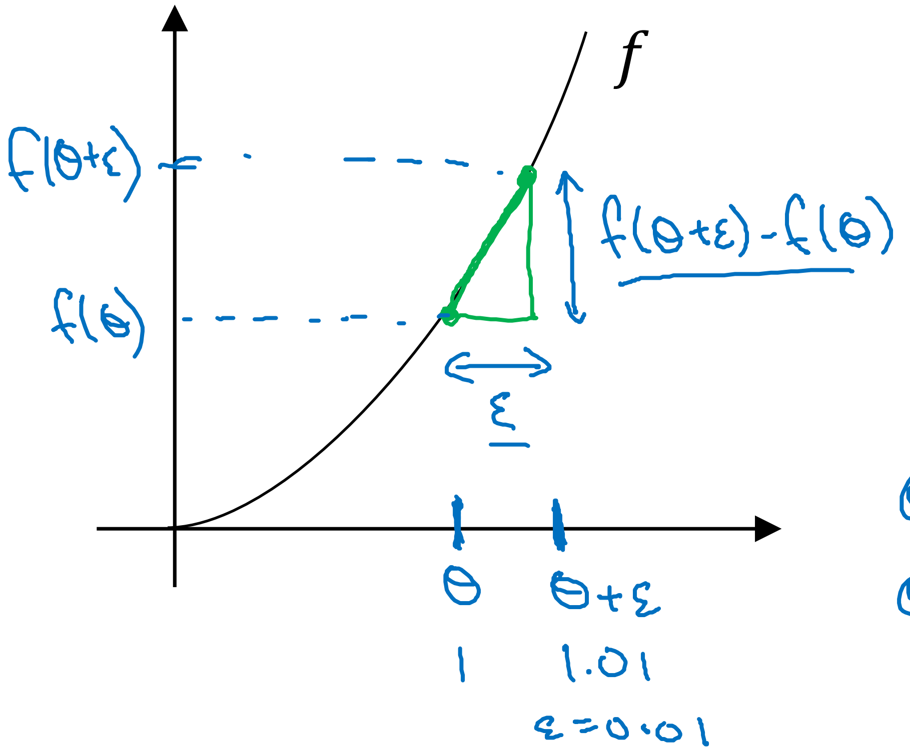
dw

db

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

when $\theta = 1$

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



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

$\theta = 1$

$\theta + \varepsilon = 1.01$

0.0301

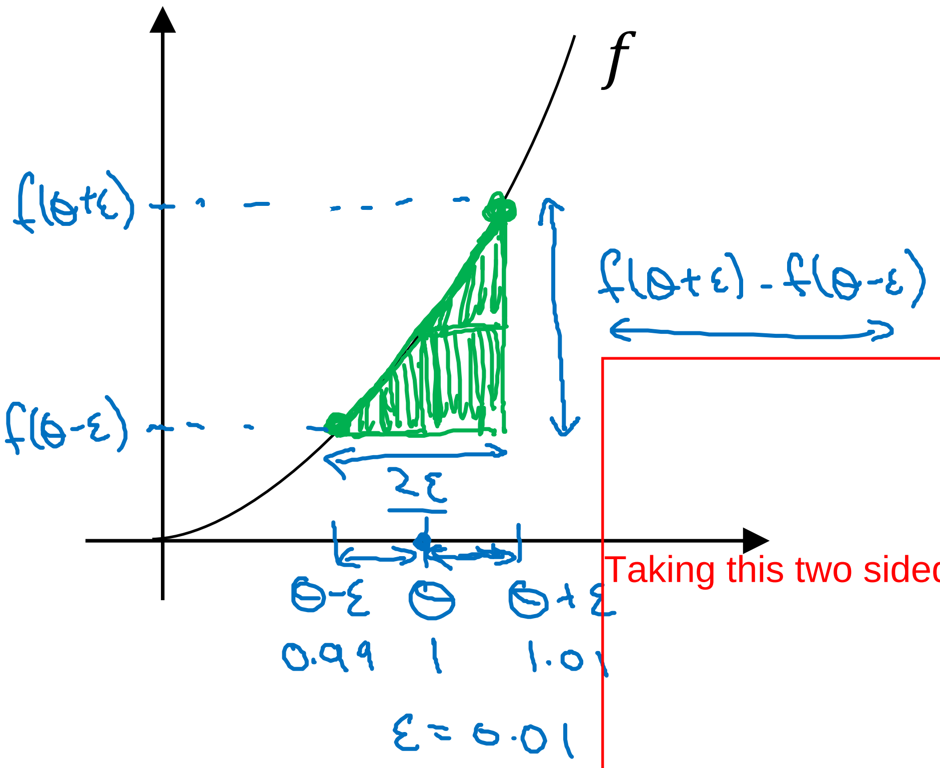
3.1

3.2

Andrew Ng

Checking your derivative computation

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



Taking this two sided approx arou

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

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

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

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

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

0.01

0.0001

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

0.01

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking

To check correctness of gradient descent and to find bug in back propo case it is useful.

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 $J(\theta)$?

Andrew Ng

Gradient checking (Grad check)

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

for each i :

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

Only change theta_i

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

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

Check $\dfrac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$ $\approx$ $\boxed{10^{-7} - \text{great!}}$ $\leftarrow$

10^{-5}

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

$\boxed{\varepsilon = 10^{-7}}$

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 w.r.t. 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