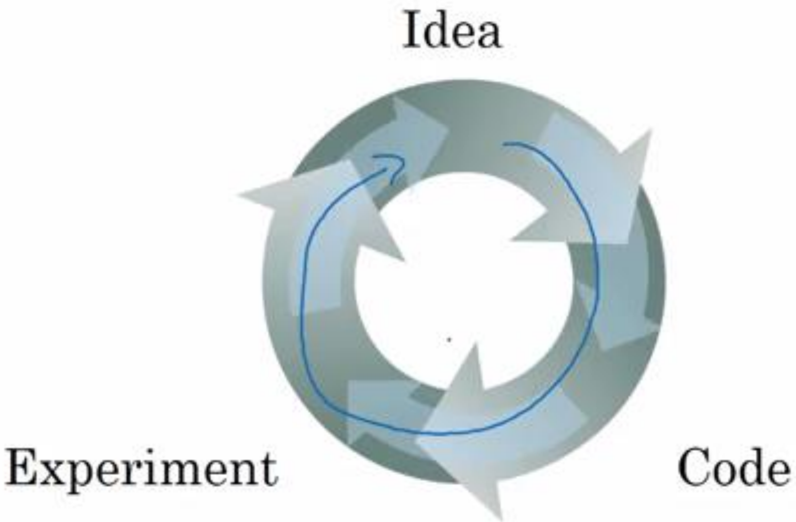
deeplearning.ai

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, Structural data

Ads search Security logistic ....

Andrew Ng

Train/dev/test sets

training set

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

test

Previous era: 70/30% (train / test)   60/20/20%

100 – 1000 – 10000

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.)

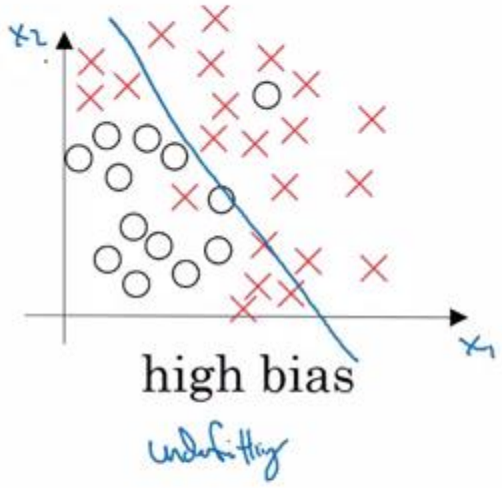
Andrew Ng

deeplearning.ai

Setting up your ML application

Bias/Variance

Bias and Variance



Andrew Ng

Bias and Variance

Cat classification

$y=1$

$y=0$

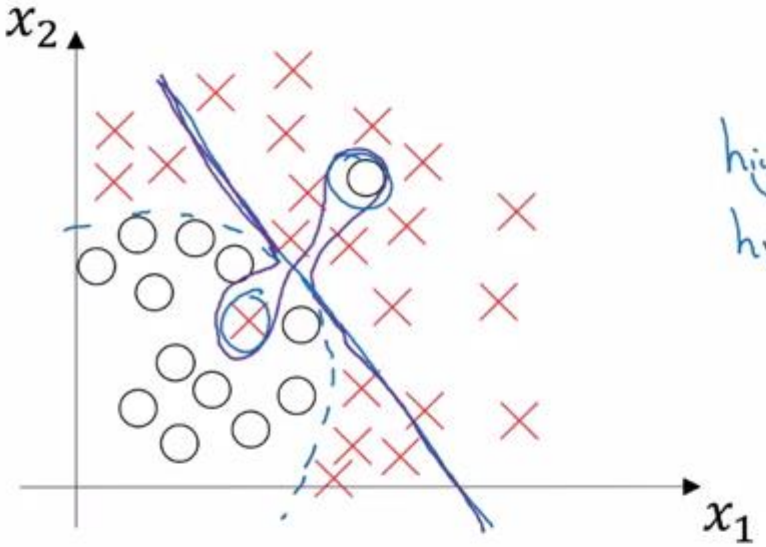
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human: ≈ 0%

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

Andrew Ng

High bias and high variance



high bias
high varian

Andrew Ng

deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

0:21 / 6:21

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"

Andrew Ng

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

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

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

omit

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

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

w will be sparse

Neural network

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

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

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

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

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

"Weight decay"

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

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

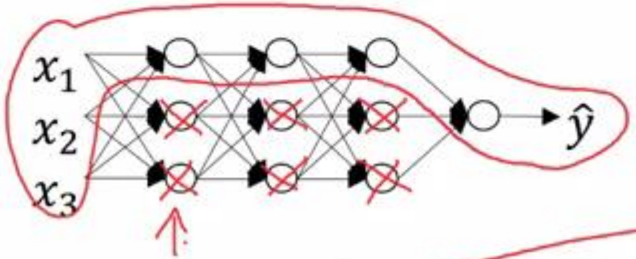
Andrew Ng

deeplearning.ai

Regularizing your neural network

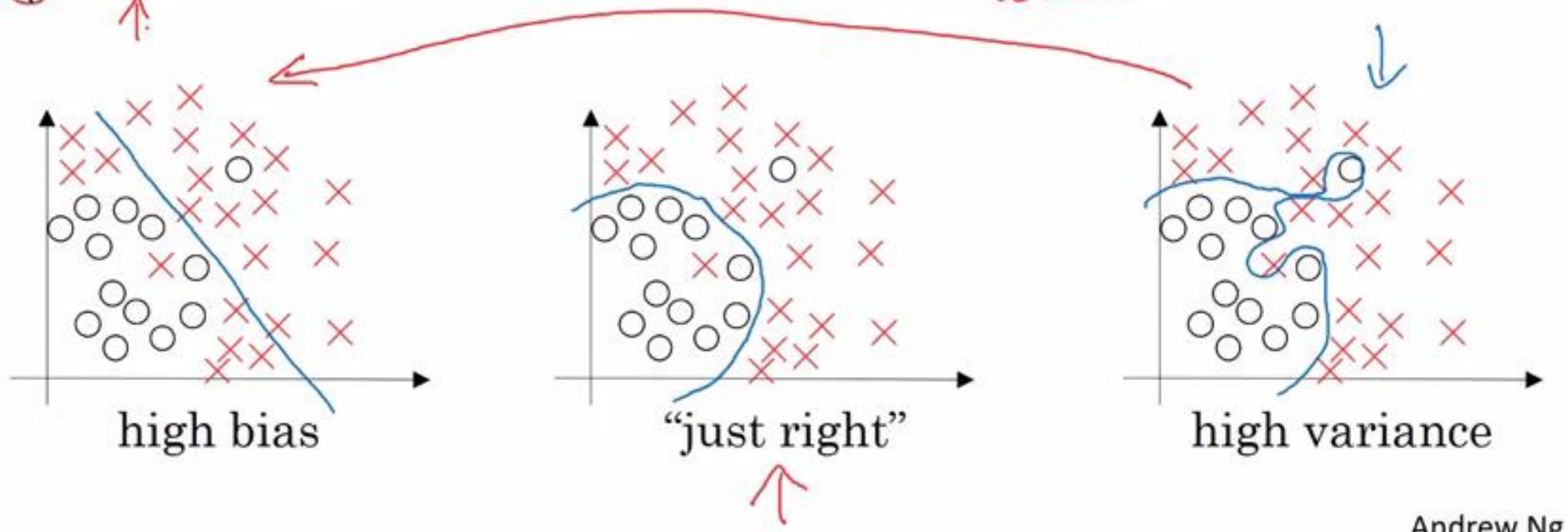
Why regularization reduces overfitting

How does regularization prevent overfitting?



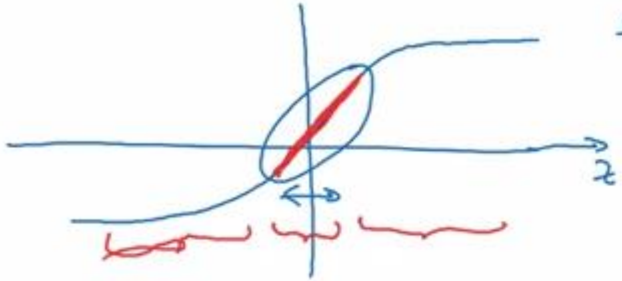
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \underbrace{\|w^{[l]}\|_F^2}$$

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



Andrew Ng

How does regularization prevent overfitting?



tanh

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

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

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

Every layer ≈ linear.

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

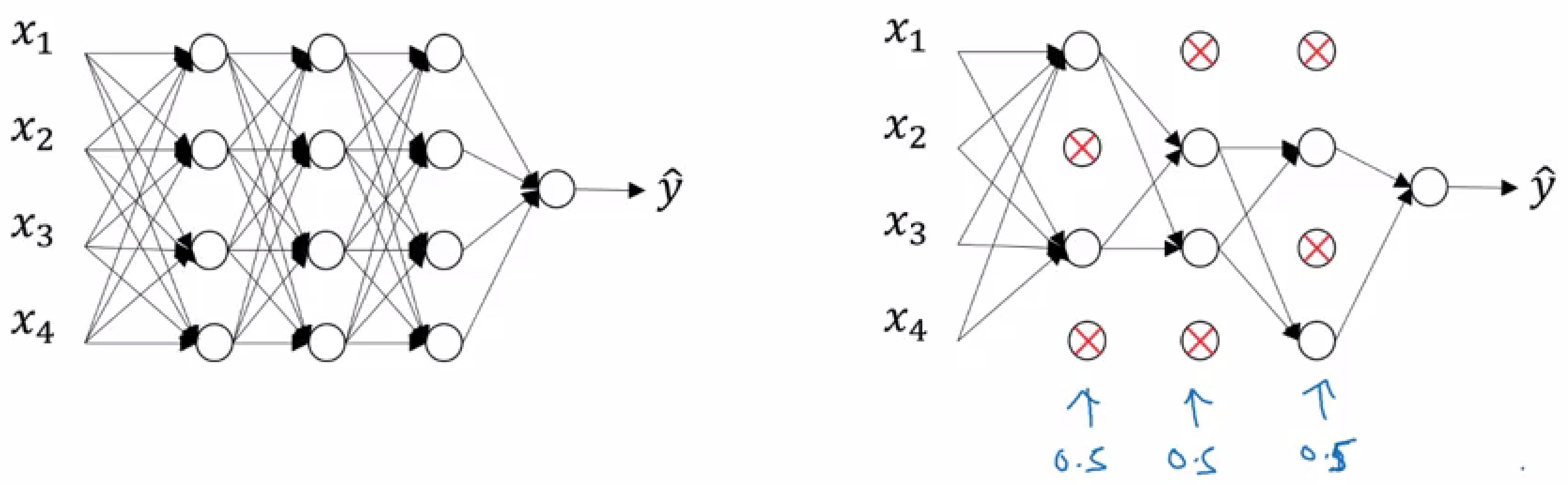
Andrew Ng

deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization



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.

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

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

$a^{[0]} = X$

No drop out.

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

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

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

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

$\downarrow$

$\hat{y}$

/= keep-prob

Andrew Ng

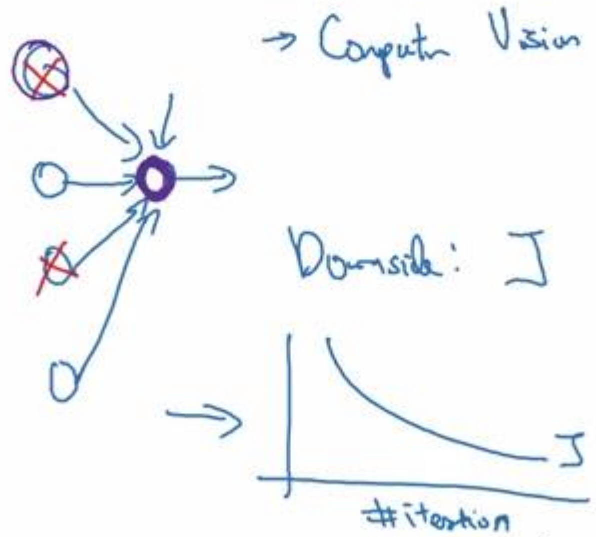
deeplearning.ai

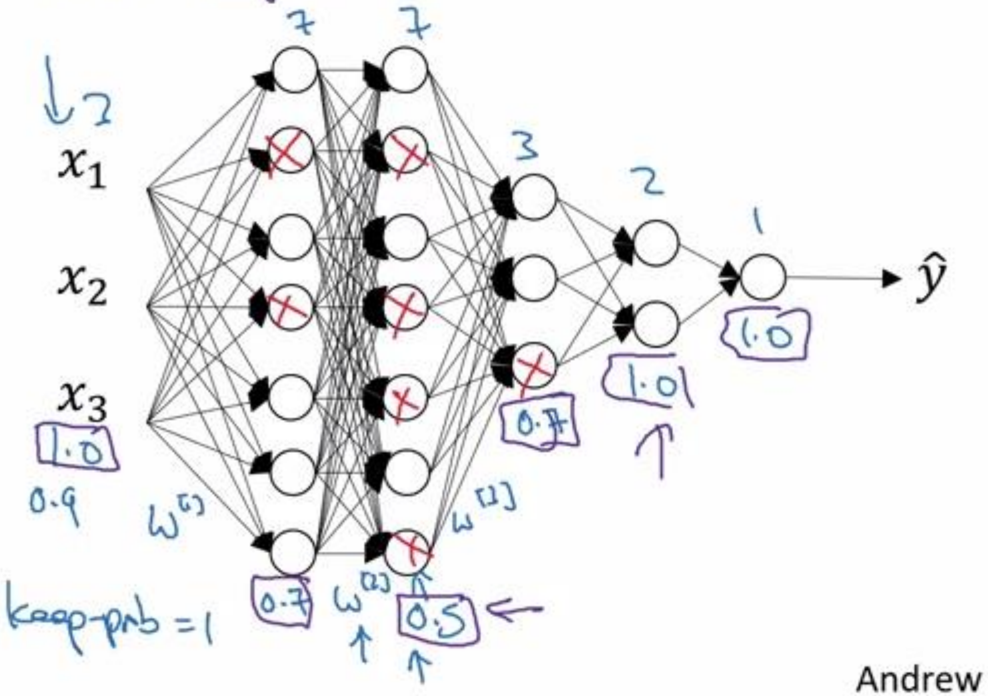
Regularizing your neural network

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





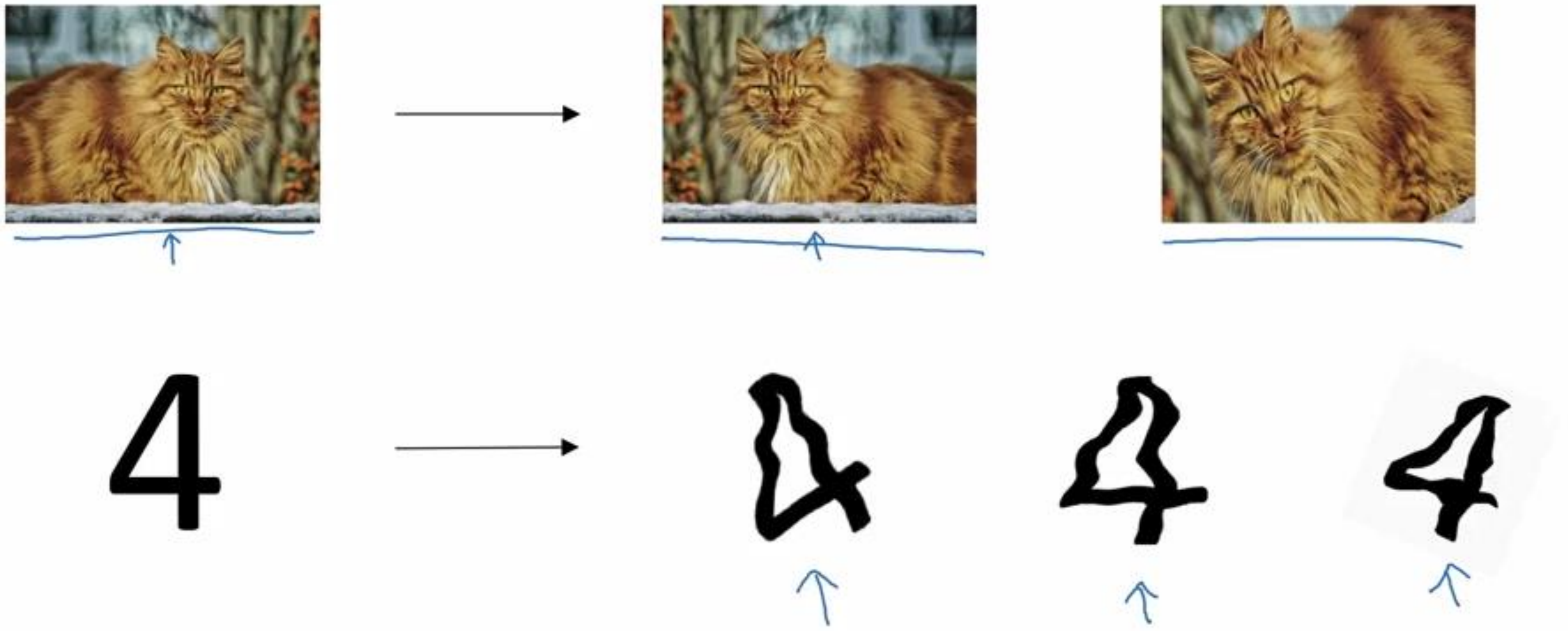
Andrew Ng

deeplearning.ai

Regularizing your neural network

Other regularization methods

Data augmentation



Andrew Ng

Early stopping

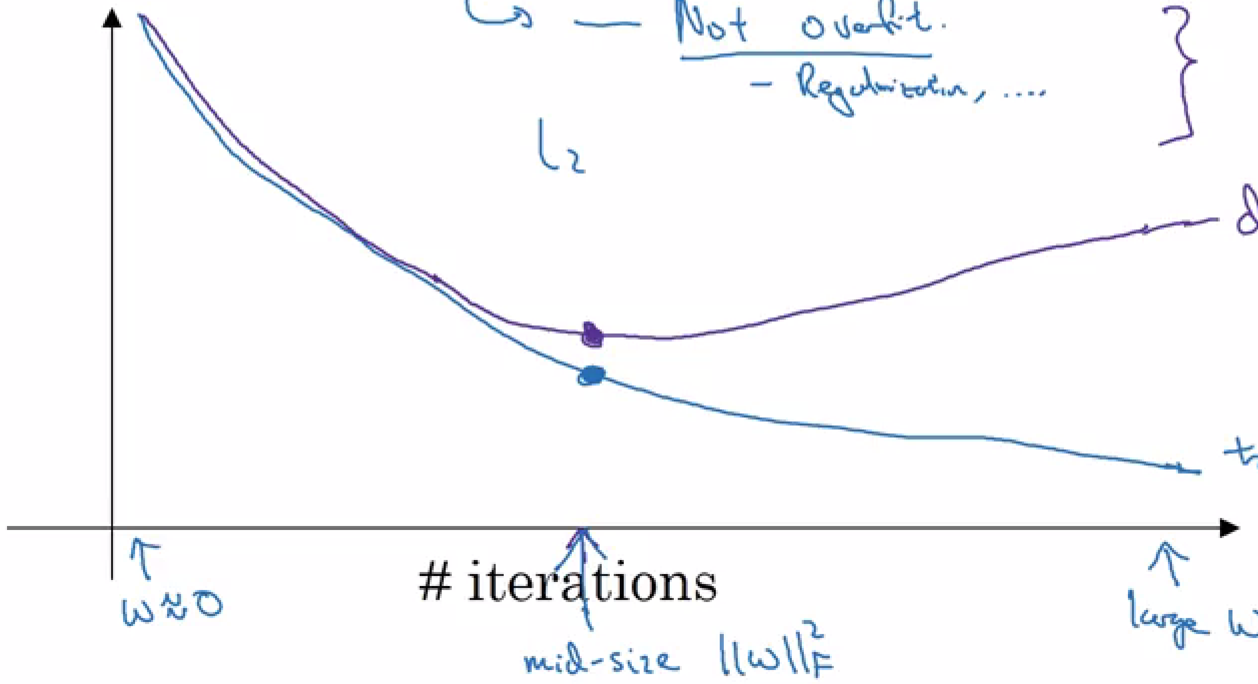
Orthogonalization

→ — Optimize cost function J
  - Gradient, ...

→ — Not overfit.
  - Regularization, ...

$J(w,b)$

L_2



Andrew Ng

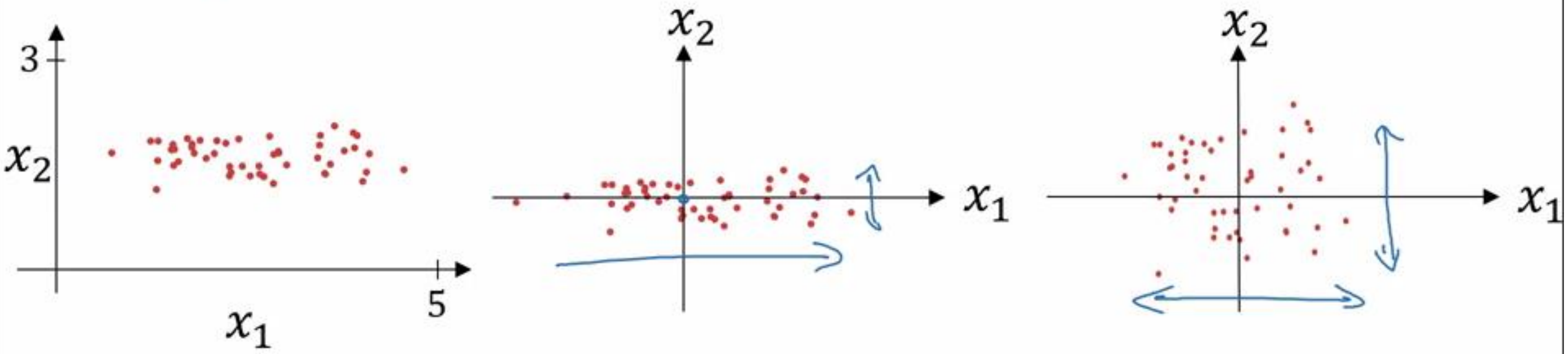
deeplearning.ai

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?

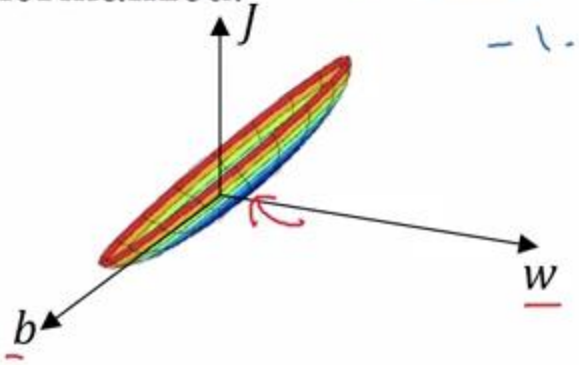
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

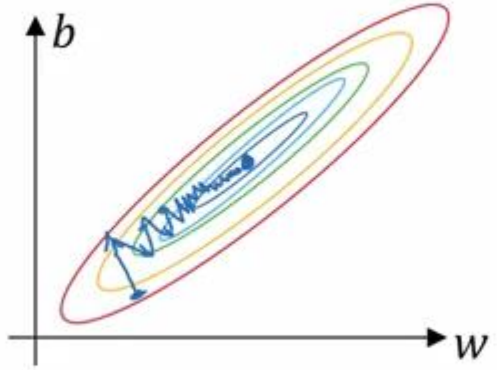
w_1 x_1: 1....1000 ←

w_2 x_2: 0...1 ←

−1...1

Unnormalized:



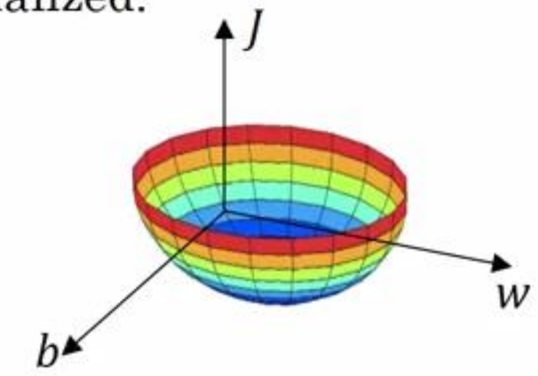


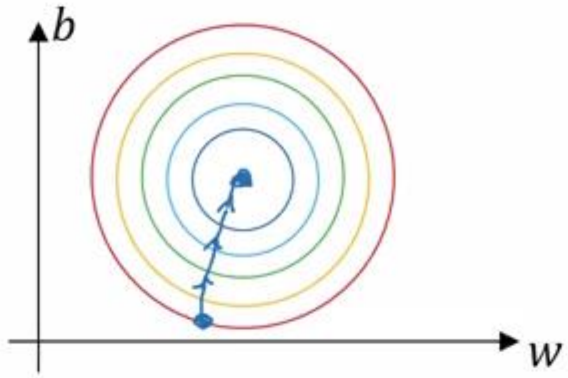
x_1: 0...1

x_2: −1..1

x_3: 1..2

Normalized:





Andrew Ng

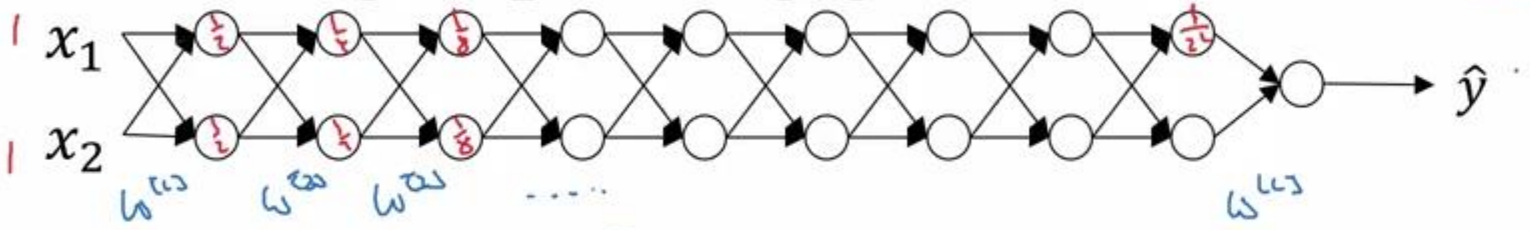
deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients

$L = 150$



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

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

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

deeplearning.ai

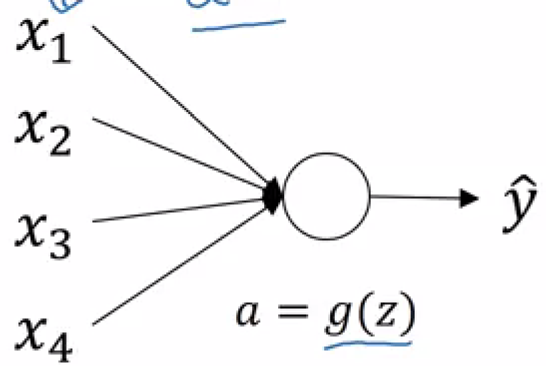
Setting up your optimization problem

Weight initialization for deep networks

Single neuron example

$a^{[l-1]}$

$w^{[l]}$



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

Large $n \rightarrow$ Smaller w_i

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

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

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

Other variants:

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

Xavier initialization

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

Andrew Ng

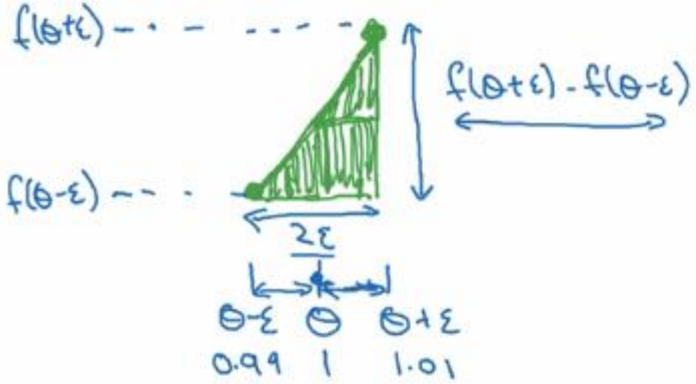
deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

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



$\varepsilon = 0.01$

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

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

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

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

deeplearning.ai

Setting up your optimization problem

Gradient Checking

Gradient check for a neural network

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

concatenate

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

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

concatenate

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

Andrew Ng

Gradient checking (Grad check)

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

for each i:

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

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

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

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

10^{-5}

$\varepsilon = 10^{-7}$

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

6:18 / 6:34

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.

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

- Remember regularization.

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

$d\theta$ = gradient of J wrt. θ

- Doesn't work with dropout.

J $\quad$ keep-prob = 1.0

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

$W, b \approx 0$

Andrew Ng