deeplearning.ai

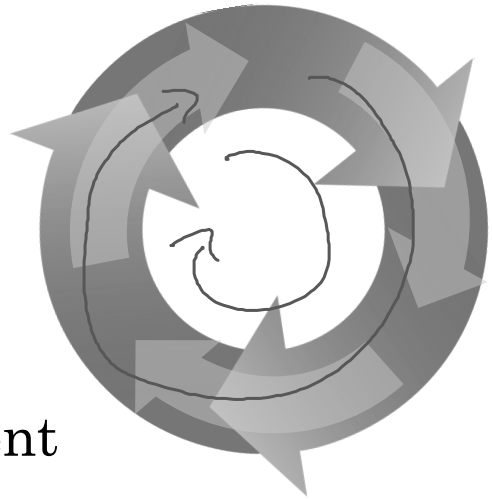
Setting up your ML application

Train/dev/test sets

structured data
- advertising
- search
- security
- logistics

Applied ML is a highly iterative process

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



NLP, Vision, Speech, Structured data

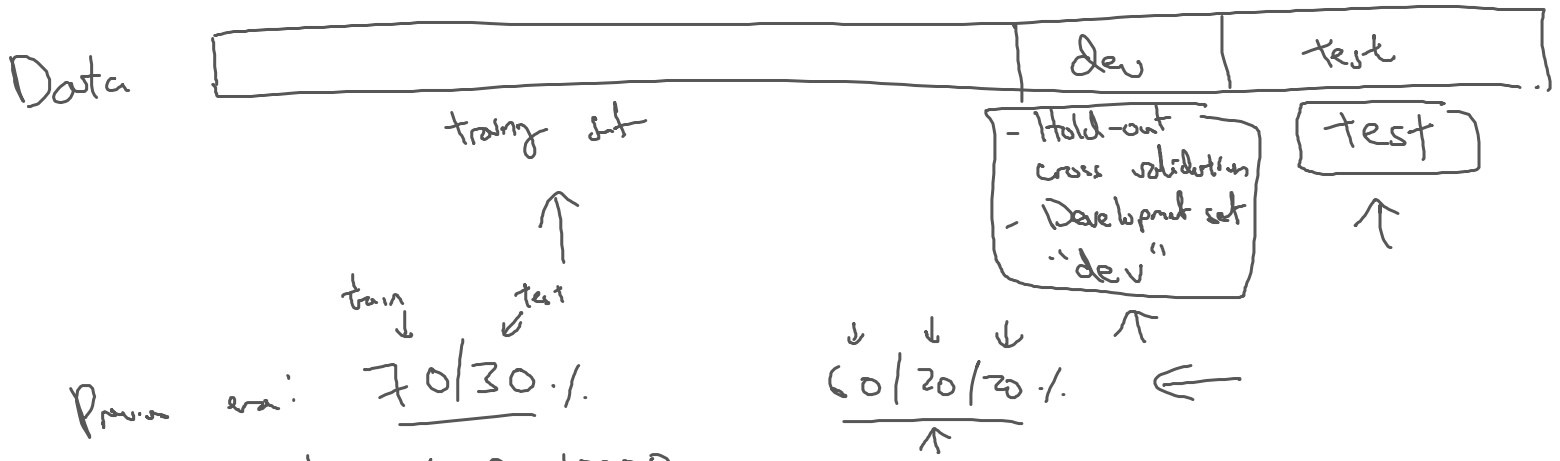
Ads Search Security logistic ....

Data : ① Training ② Development ③ Testing

Dev set = Hold-out = Cross validation = development

برای تطابیس عملکرد چندین الگوریتم جامعه است.

Train/dev/test sets



Mismatched Distributions اگر منبع Train متفاوت از dev/Test باشد ⇐ Solution: حداقل dev/test منبع مشابه باشند.

Mismatched train/test distribution

Carts

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

→ Make sure dev and test come from same distribution.

train / dev ("test")

train / test → train / dev

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

خصوصاً اگر unbiased estimate مهم نباشد (چون ماشین قبلاً dev set را دیده است، Test کردن با dev دقیق نیست)

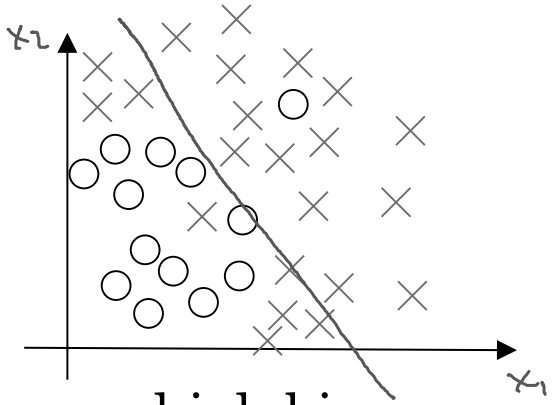
← وقتی می‌گویند Train/Test منظور Train/dev است.

deeplearning.ai

Setting up your ML application

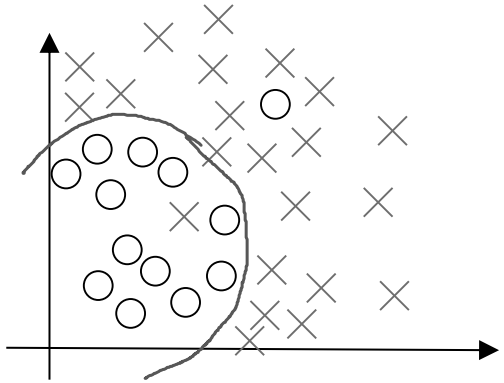
Bias/Variance

Bias and Variance

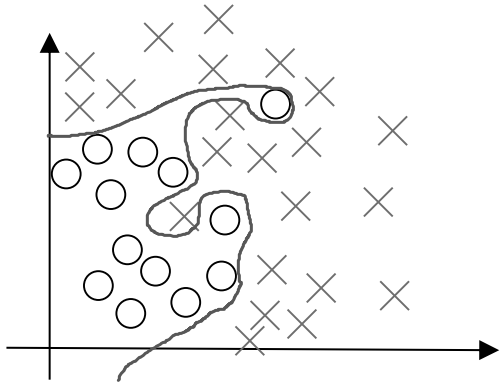


high bias

underfitting



→ "just right"



high variance

overfitting

آدم یک انسان بخواهد بین سگ/گربه classification کند شاید با خطای 5% بکند ← این عدد معیاری برای عملکرد بالا یا پایین optimal (Bayes) Error

تا بین خطای Train, Dev است. مثلا Train خطای 6% بد نیست.

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

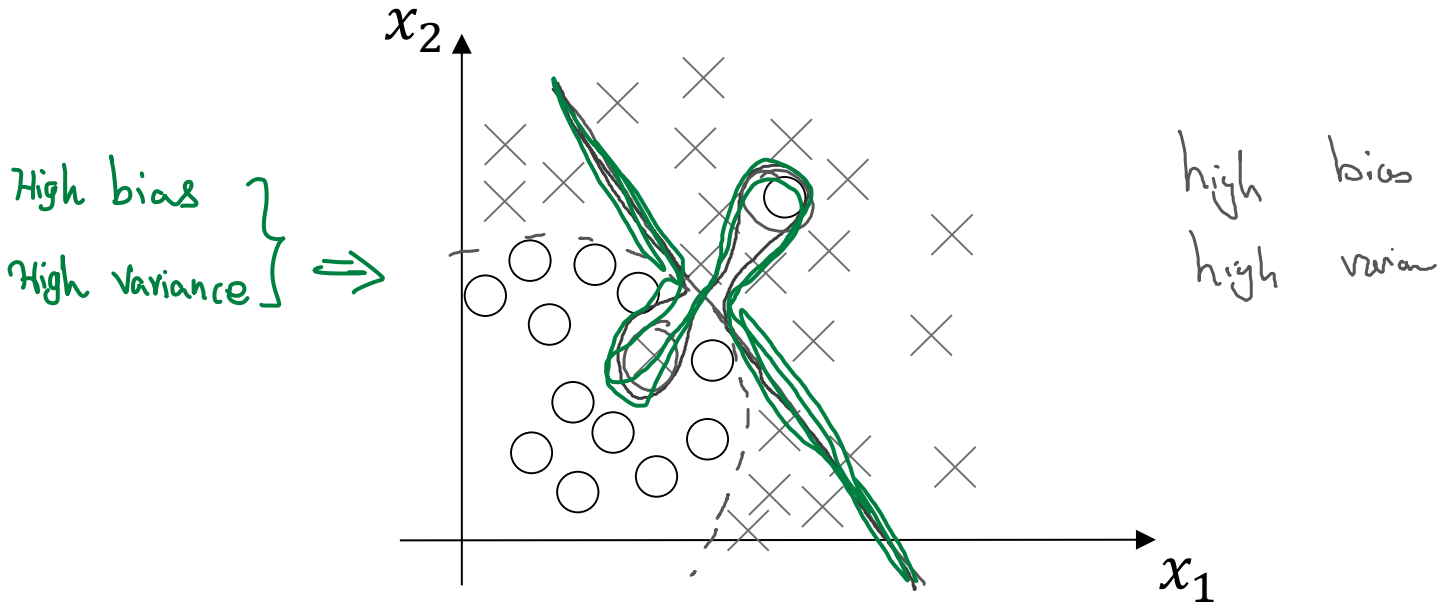
بهترین حالت زمانی است که Bias, Variance هر دو minimal باشند و بدترین وقتی هر دو maximal باشند.

High Bias: Training error ≫ Bayes error ⇒ underfitting

High Variance: Dev error ≫ Training error ⇒ overfitting

Can we have under/over fitting simultaneously?

High bias and high variance

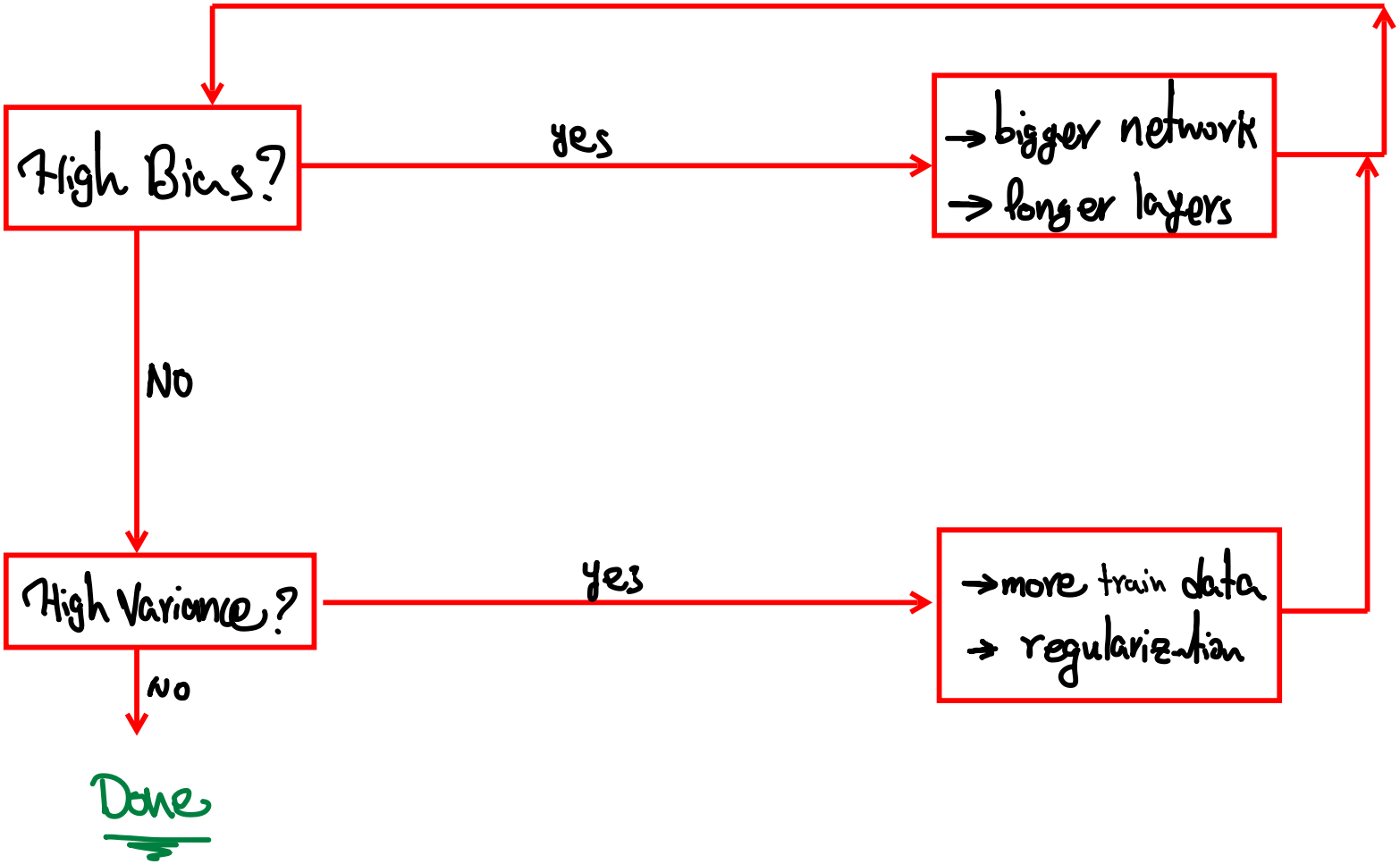


deeplearning.ai

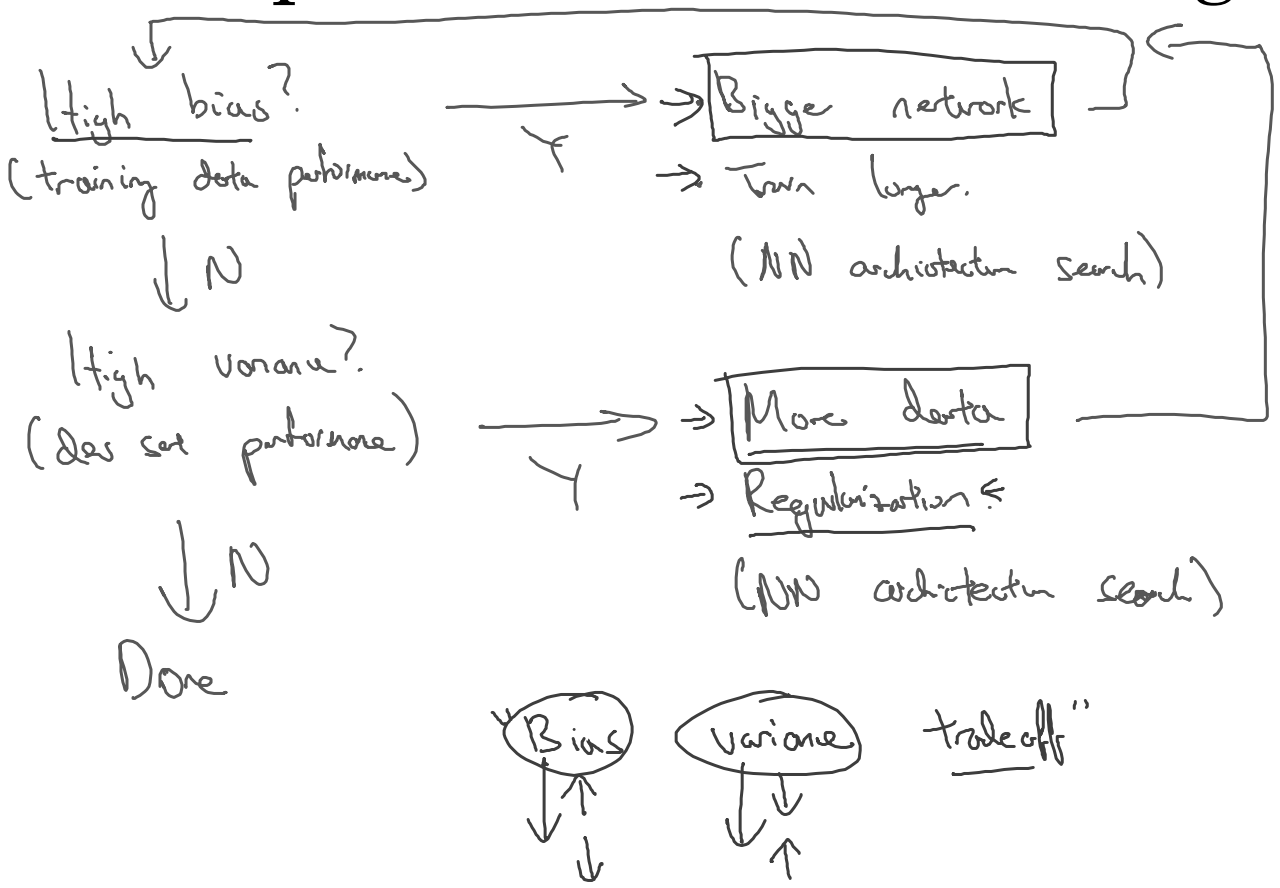
Setting up your ML application

Basic “recipe” for machine learning

Basic "recipe" for machine learning



Basic recipe for machine learning



deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

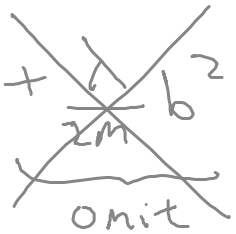
$\min_{w,b} J(w,b)$

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

λ = regularization parameter

lambda lambd

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



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

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

w will be sparse

L_2 Regularization $\frac{\lambda}{2m}\|w\|_2^2$ (weight decay)

L_1 Regularization $\frac{\lambda}{2m}\|w\|_1 \Rightarrow$ Sparse w (more zero components)

For vectors

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

Frobenius norm (matrix) $\|W^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(W_{ij}^{[l]})^2$ where $W^{[l]} : (n^{[l]}, n^{[l-1]})$

Neural network

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

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

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

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

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

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

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

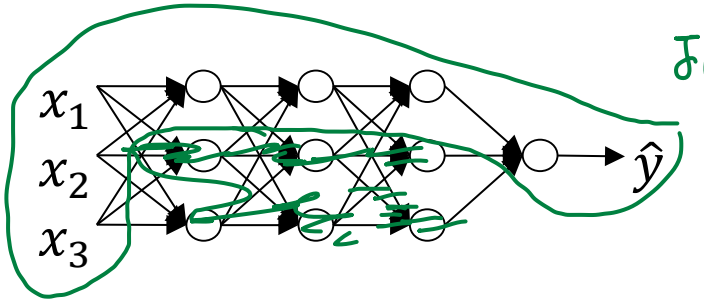
$$= \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{<1} w^{[l]} - \alpha(\text{from backprop})$$

L_2 Regularization $\equiv$ "Weight decay"

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

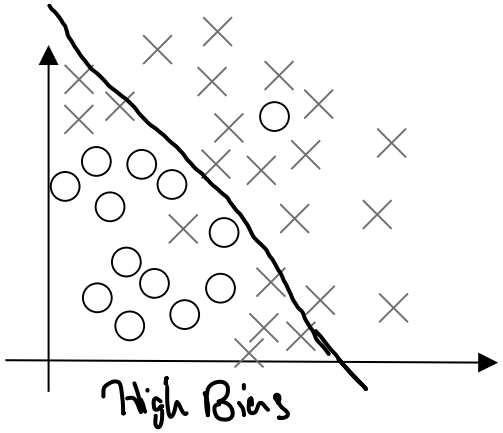
$w^{[l]} = w^{[l]} - \alpha(\text{from backprop})$ $\rightsquigarrow$ without regularization

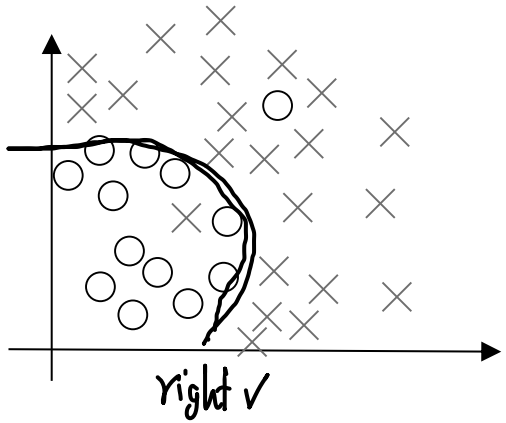
How does regularization prevent overfitting?

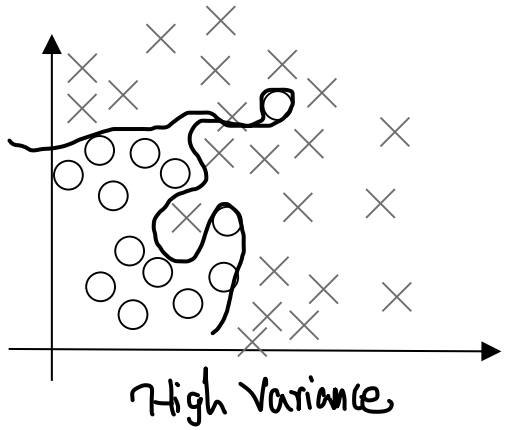


$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum \mathcal{L}(\hat{y}_i, y_i) + \frac{\lambda}{2m}\sum_{l=0}^{L} \|w^{[l]}\|_F^2$$

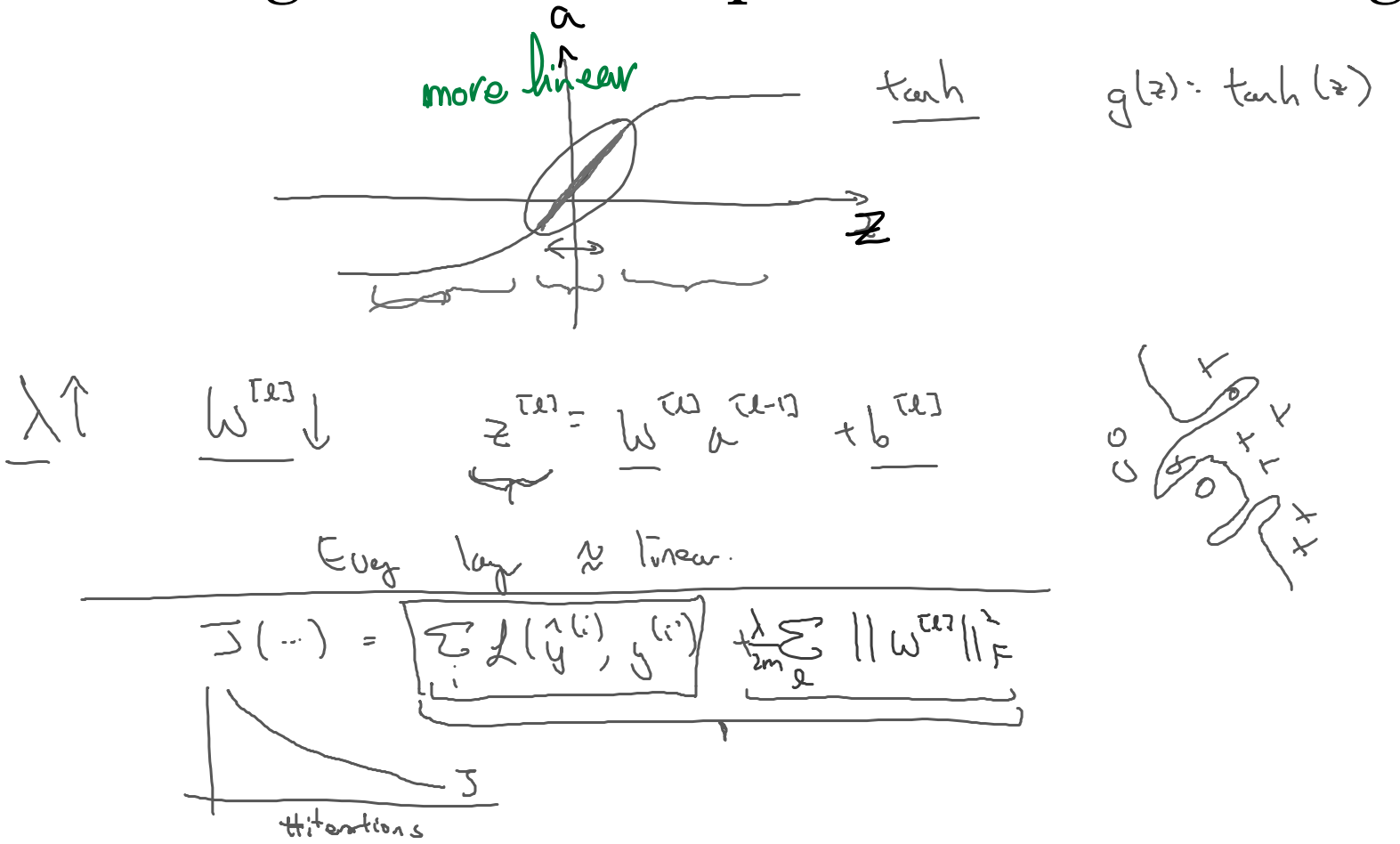
If $\lambda \uparrow \Rightarrow J_{min}$ leads to $w^{[l]} \simeq 0 \Rightarrow$ simpler model $\Rightarrow$ overfitting $\downarrow$







How does regularization prevent overfitting?



deeplearning.ai

Regularizing your neural network

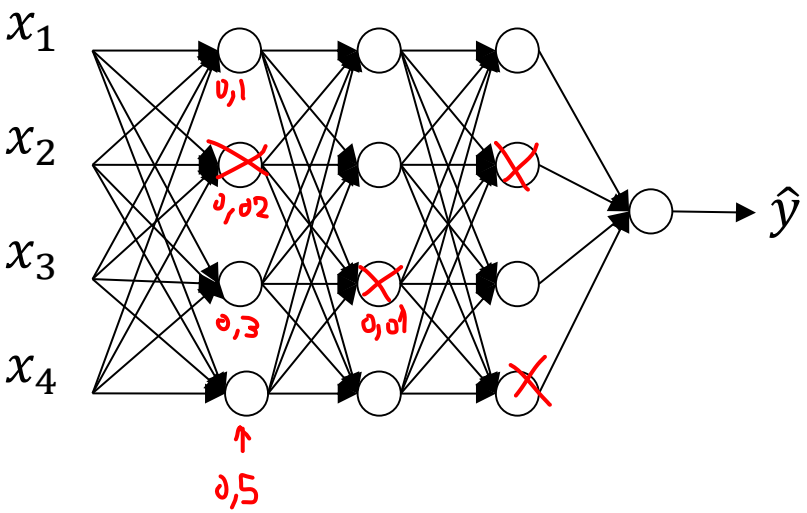
Dropout regularization

Dropout regularization

برای جلوگیری از overfitting

keep_prob ← احتمال صفر شدن یک hidden node به صورت تصادفی به هر

← هر بار در هر iteration این نوع zero-assignment را انجام می‌دهیم



0.5 0.5 0.5

معروف‌ترین نوع از regularization technique، dropout است.

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

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

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

$/= 0.8$

1. $d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_Prob$ ⇒ d3 برداری از T و F با احتمال 0,8 و 0,2 می‌شود.
2. $a3 = np.multiply(a3, d3)$
3. $a3 = a3 / keep\_prob$

در مرحله Test set دیگر انجام نمیدهیم dropout .

Making predictions at test time

$a^{[0]} = X$

No drop out.

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

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

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

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

$\hat{y}$

/= keep-prob

Andrew Ng

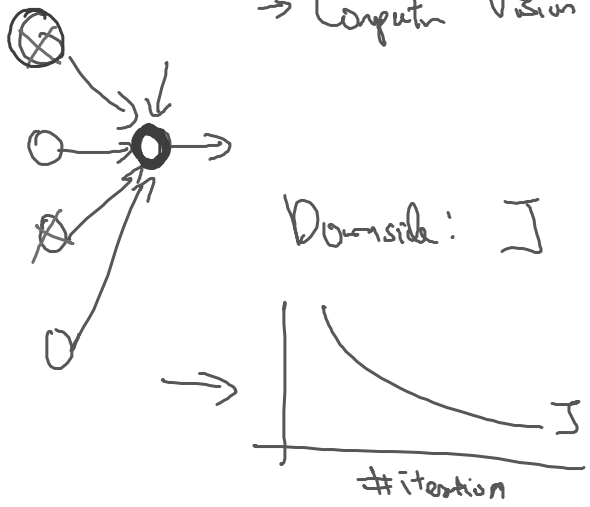
deeplearning.ai

Regularizing your neural network

Understanding dropout

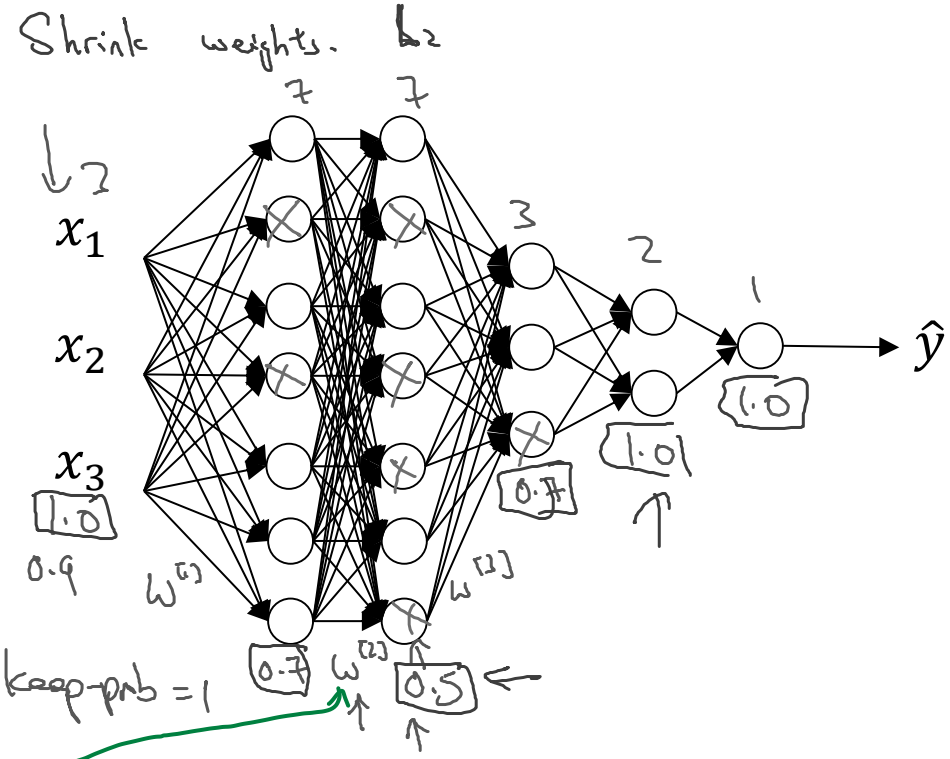
Why does drop-out work? ⇒ Similar to L_2 & shrinking the weights!

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



→ Computer Vision

Downside: J



برای هر لایه می‌توان keep-prob خاص خودش را تعریف کرد ⇐ مخصوصاً برای لایه‌هایی که بزرگ‌ترند. حتی می‌توان برای Input features این کار کرد.

البته مشکل این است که cost function را دیگر نمی‌توان Consistantly تعریف کرد.

Regularization: ① L_2 ② dropout ③ data augmentation ④ early stopping

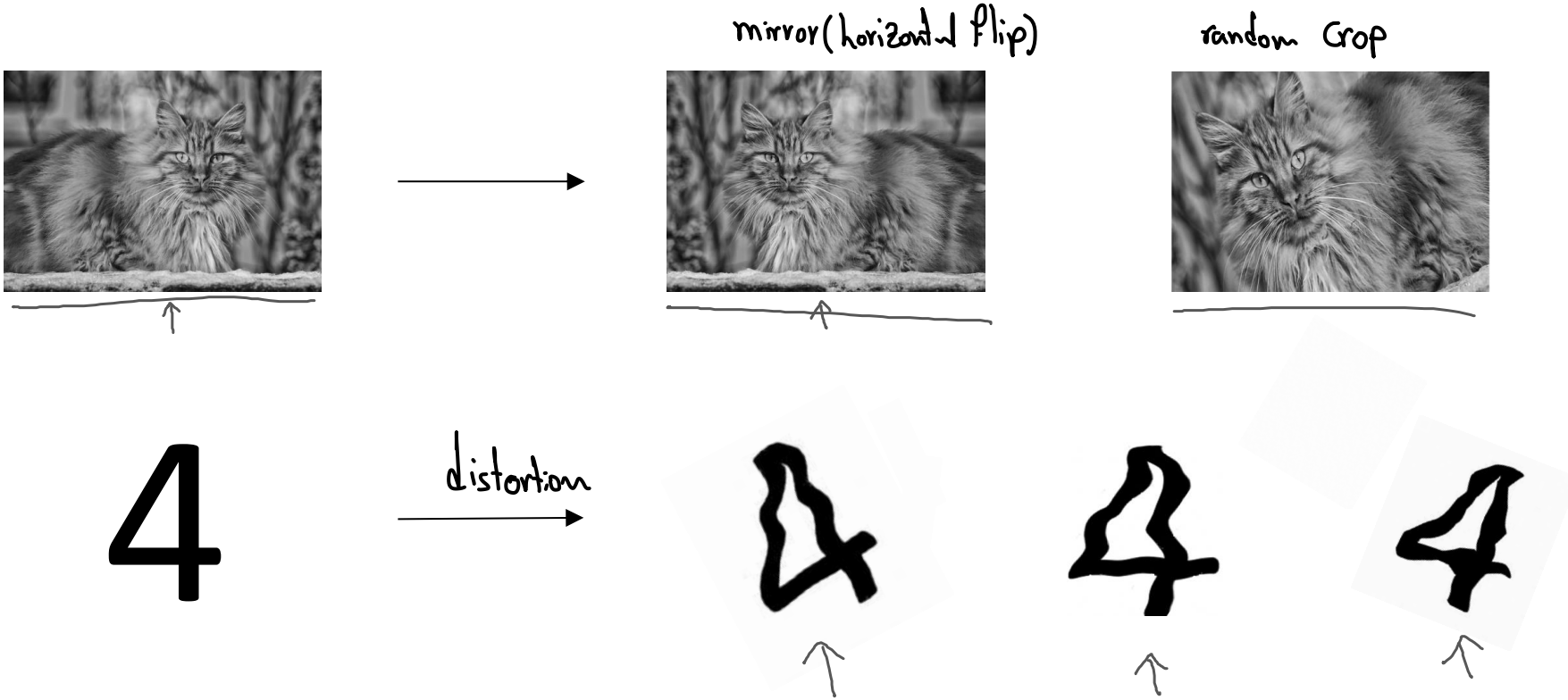
deeplearning.ai

Regularizing your neural network

Other regularization methods

بجای اینکه more training data جمع کنیم ← یک حیله augmentation میکنیم که regularization است.

Data augmentation



orthogonalization ⇐ to independently do the two tasks.

⇐ عیب early stopping: orthogonalization را رعایت نمی‌کند.

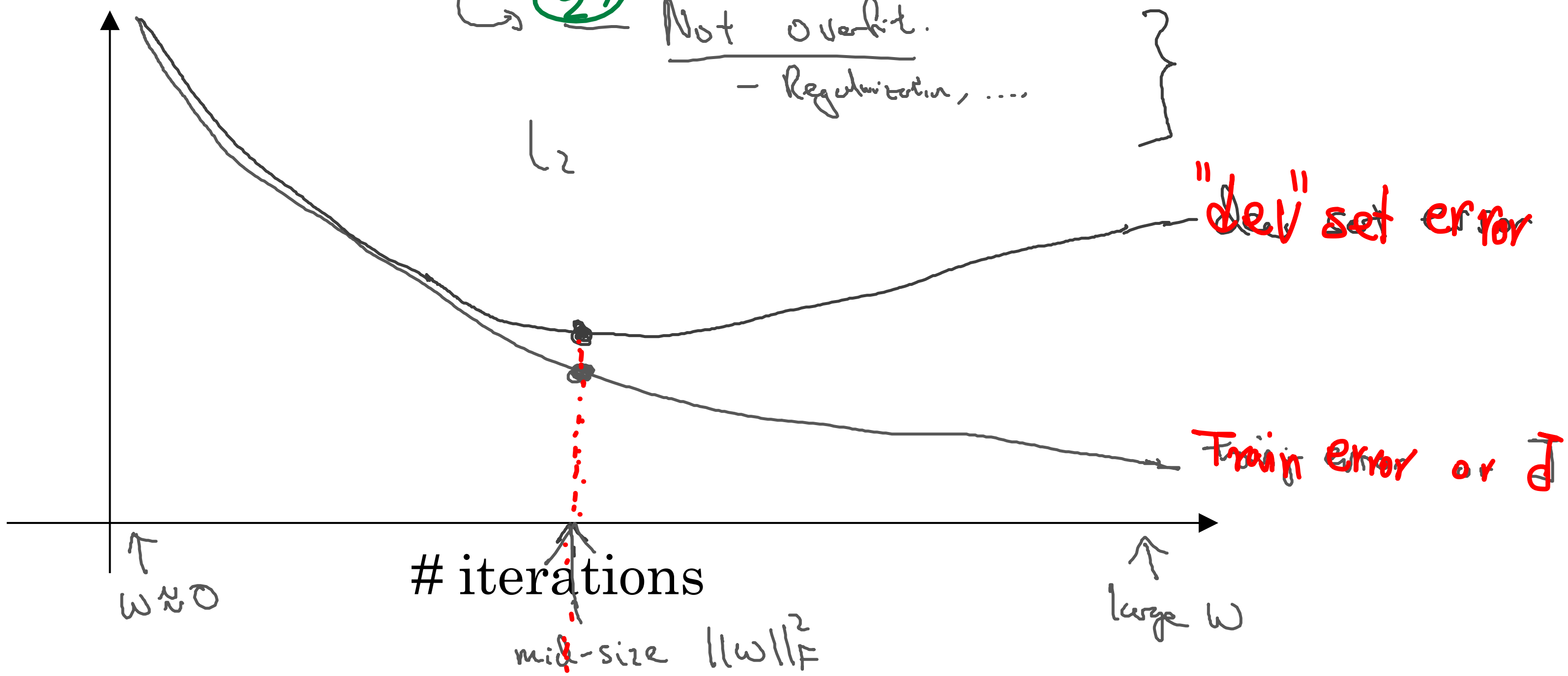
Early stopping

Orthogonalization.

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

L_2

$J(w, b)$



early-stop (زود train می‌کنه)

deeplearning.ai

Setting up your optimization problem

Normalizing inputs

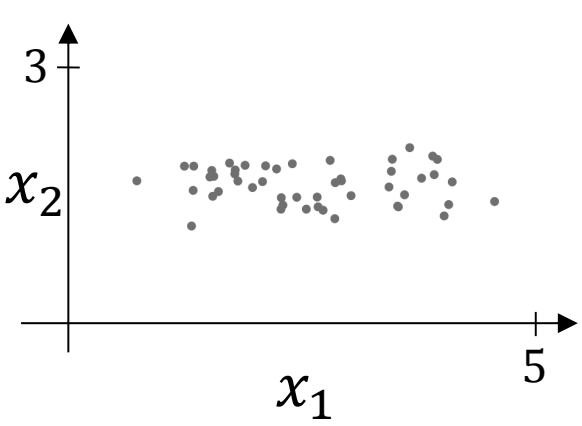
میانگین و واریانس داده‌ها صفر نمی‌شود.

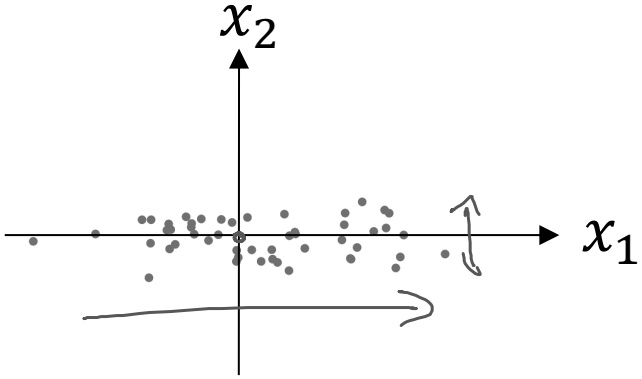
برای داده‌های normalization باید Train و dev مثل هم باشد و جداگانه نباشد

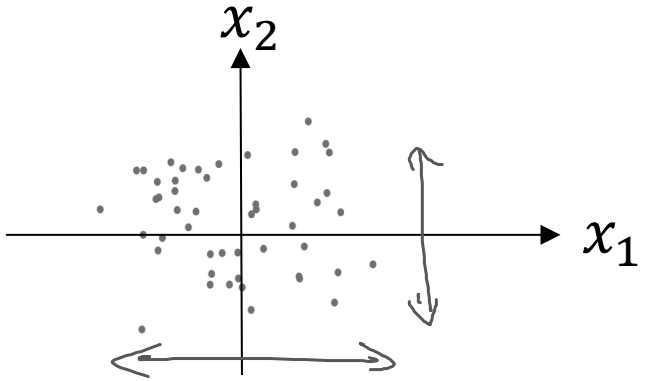
Normalizing training sets

$$X = \frac{X - \mu}{\sigma}$$

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







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

$x /= \sigma$

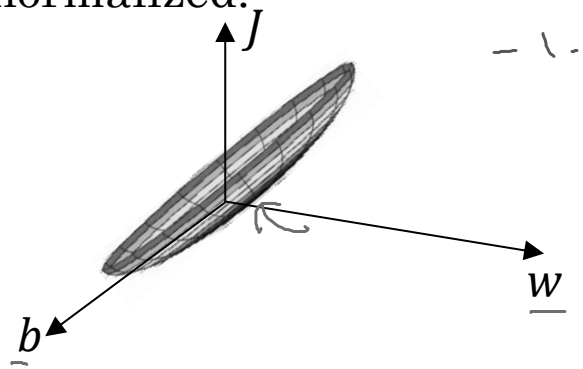
Use same μ, σ^2 to normalize test set.

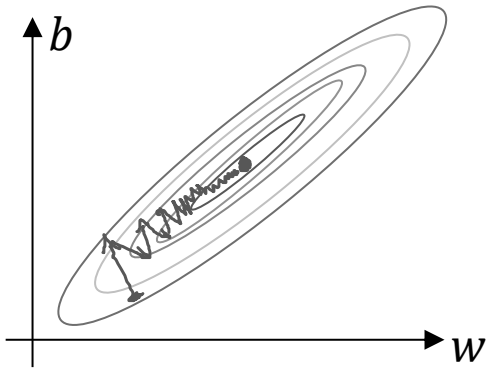
Andrew Ng

Why normalize inputs?

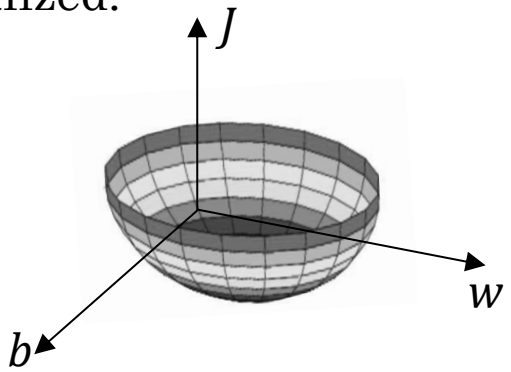
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

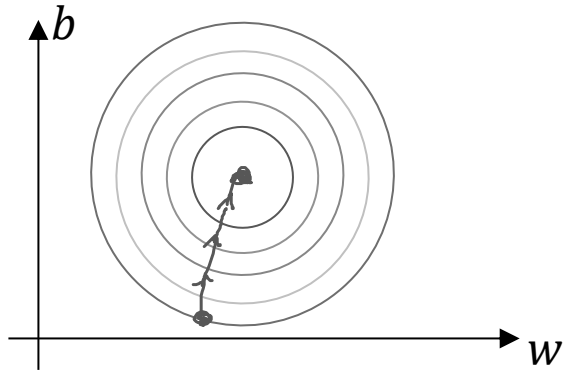
Unnormalized:





Normalized:





Andrew Ng

deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

$\frac{\partial J}{\partial w} \to 0$ or $\to \infty$ ، deep های خیلی NN در ⇐ $(w^{[l]} < 1$ or $> 1)$ اگر مقادیر اولیه $w^{[l]}$ ها کمتر یا بیشتر از یک باشد

می‌رود ⇐ باید initial values را هوشمند انتخاب کرد.

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

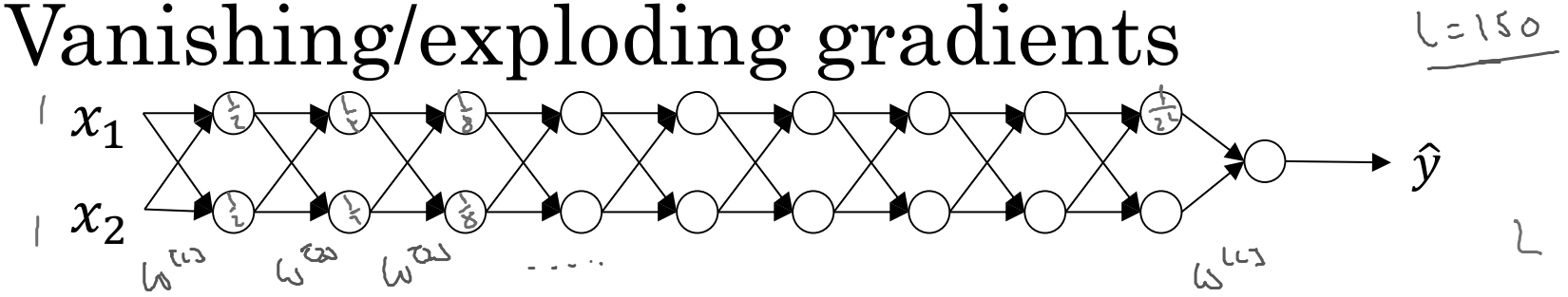
Tanh: $\frac{1}{\sqrt{n^{[l-1]}}}$

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

این جا گفته می‌شود که $Var(w^{[l]})$ بصورت $\frac{2}{n}$ شود ⇐ $w^{[l]}$ خیلی کمتر یا بیشتر از یک نیست.

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

Vanishing/exploding gradients



$L = 150$

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

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

$a^{[3]} = w^{[3]} w^{[2]} w^{[1]} x$

1.5^{L}

0.5^{L}

$w^{[l]} > I$

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

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

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

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

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

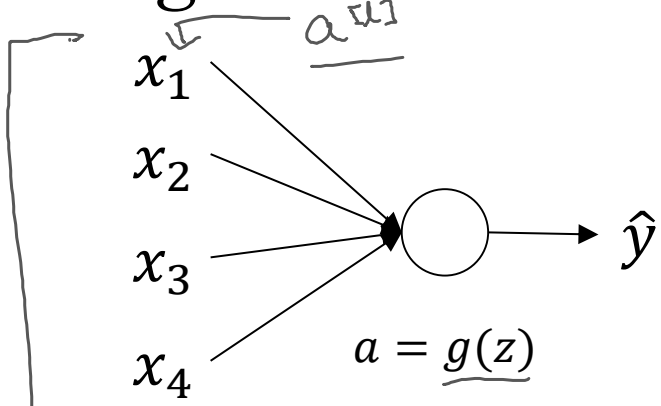
$\hat{y} = w^{[L]} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1} x$

$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example



$w^{[l]}$

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

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

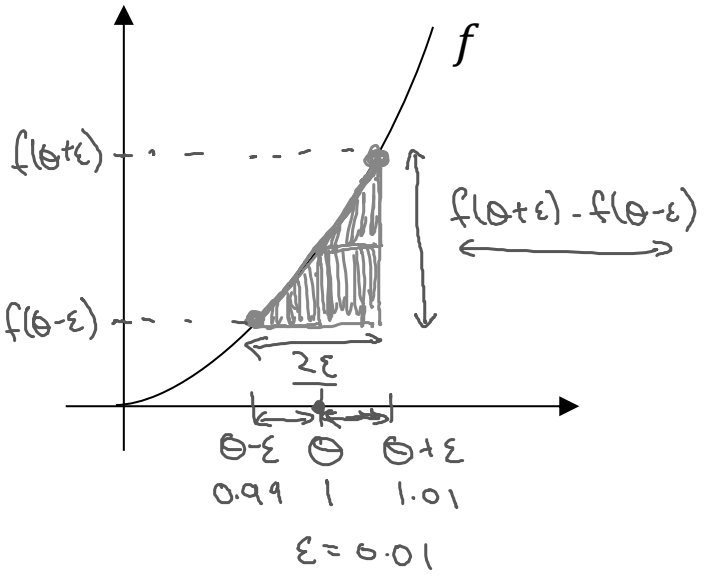
deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

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



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

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

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

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

Gradcheck

Gradient check for a neural network

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

concatenate

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

② 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)

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

(3) for each i:

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

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i} \quad | \quad d\theta_{approx} \overset{?}{\approx} d\theta$$

(4) Check

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

$\varepsilon = 10^{-7}$

$\approx$ 10^{-7} — great! ←

10^{-5}

$\rightarrow 10^{-3}$ — worry. ←

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.

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

$d\theta$ = grad 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