

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

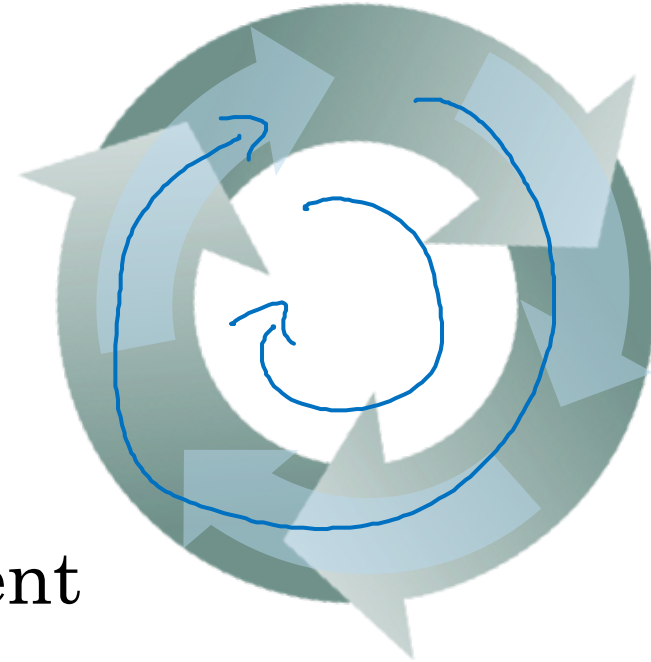
hidden units

learning rates

activation functions

...

Idea



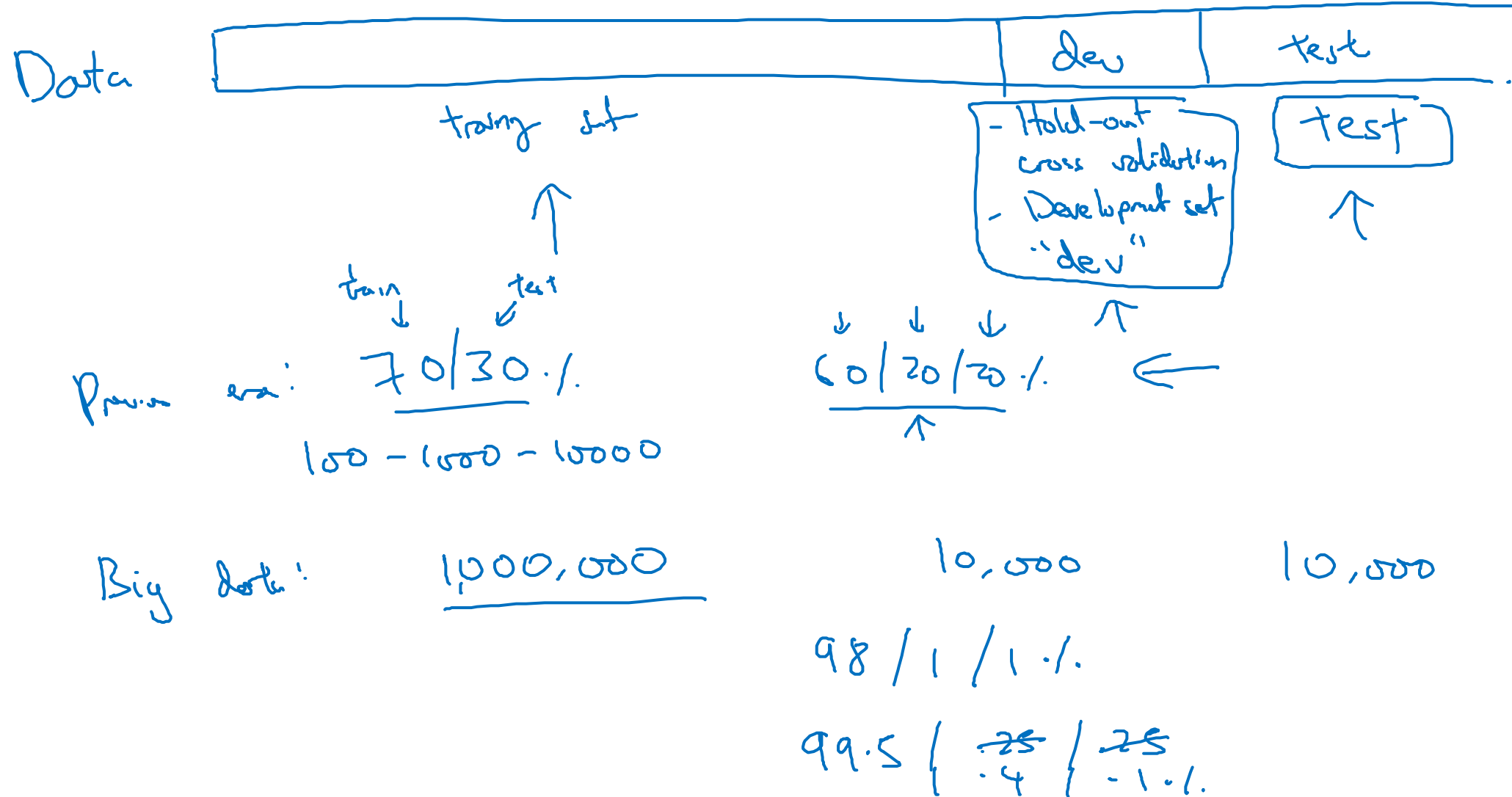
Experiment

Code

NLP, Vision, Speech, Structured data

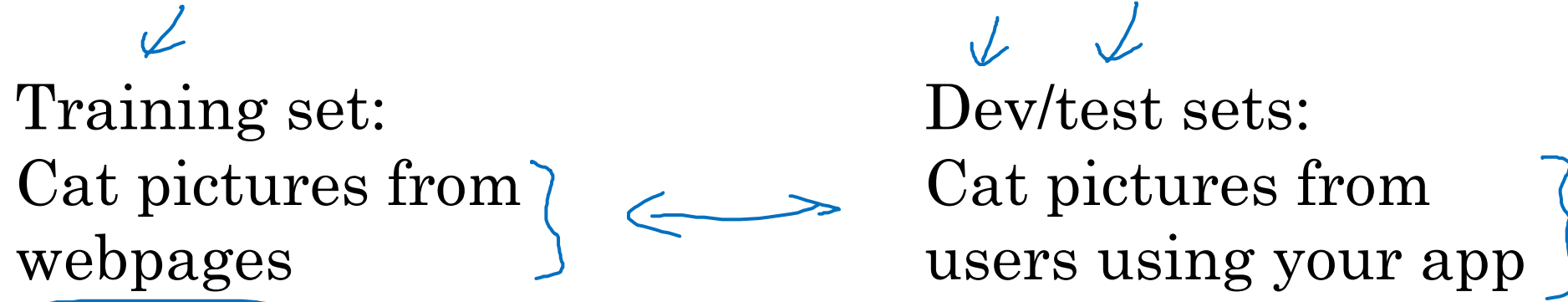
```
graph TD; A["NLP, Vision, Speech"] --> B["Structured data"]; B --> C["Ads"]; B --> D["Search"]; B --> E["Security"]; B --> F["logistic ..."]; E --> G[" "]; F --> G;
```

Train/dev/test sets

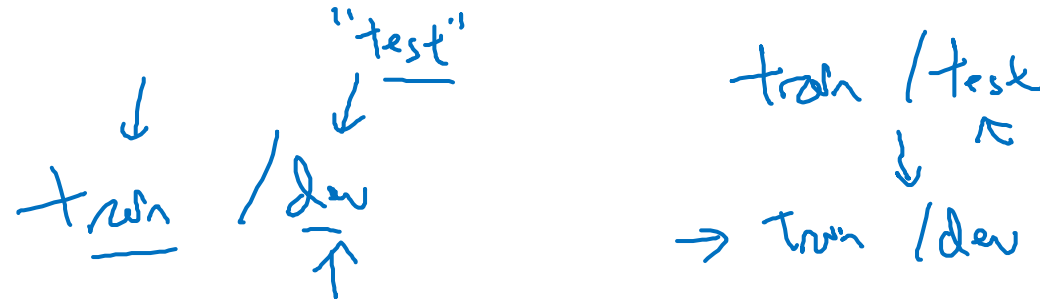


Mismatched train/test distribution

Certs



→ Make sure dev and test come from same distribution.



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

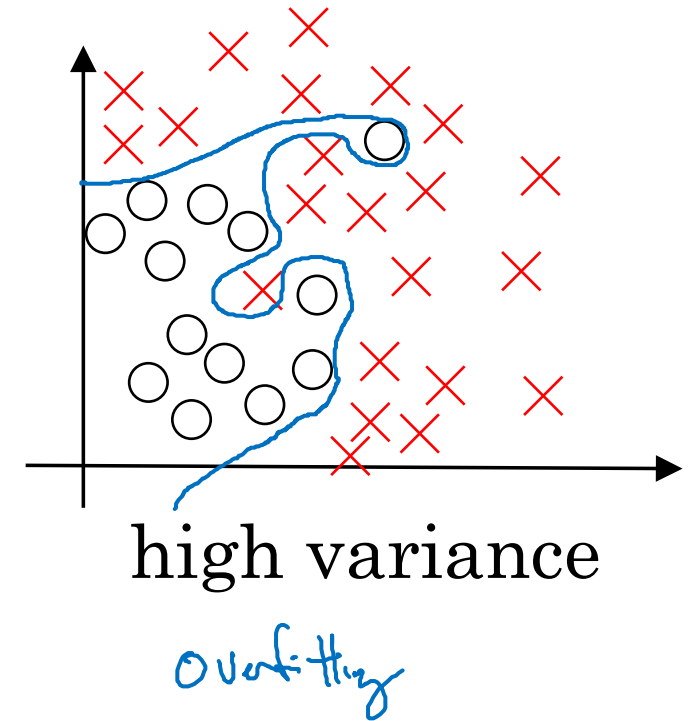
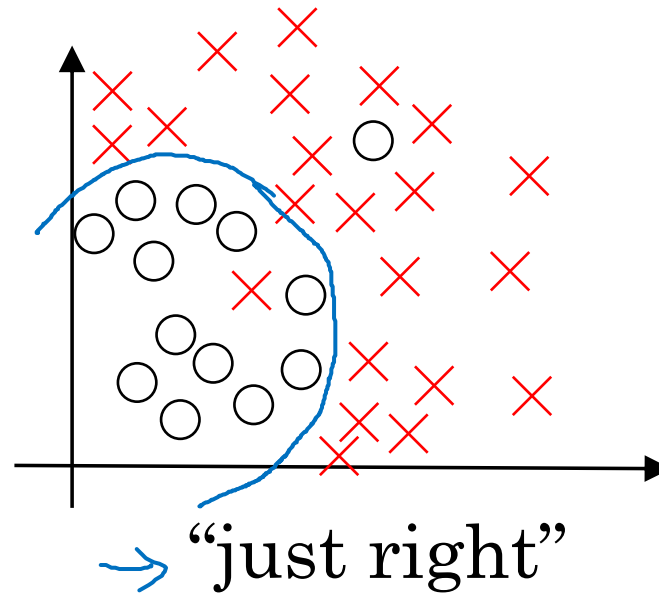
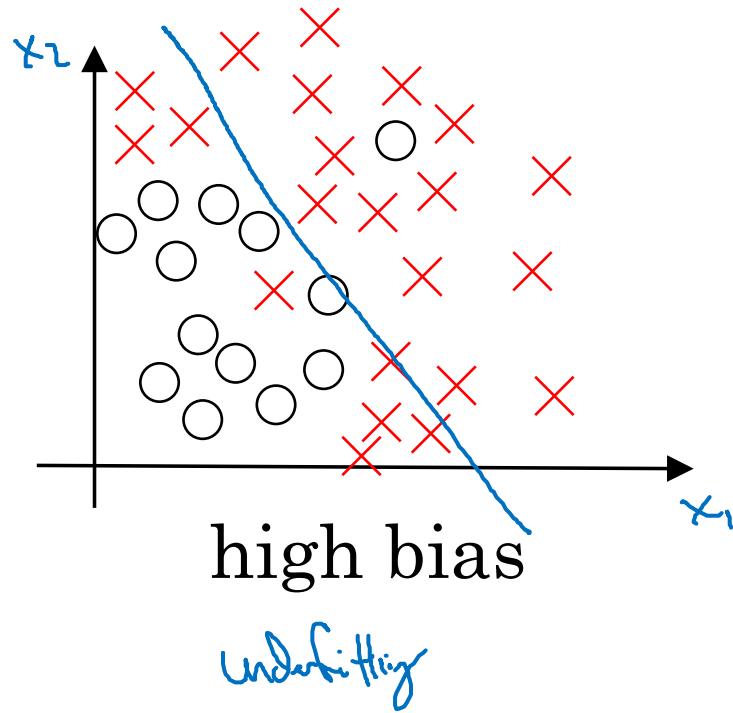


deeplearning.ai

Setting up your ML application

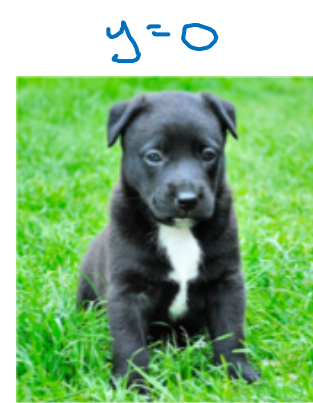
Bias/Variance

Bias and Variance



Bias and Variance

Cat classification



Train set error:

1%

15% \swarrow

15%

0.5%

Dev set error:

11%

16% \swarrow

30%

1%

high variance
 \uparrow

high bias
 \uparrow

high bias
& high variance

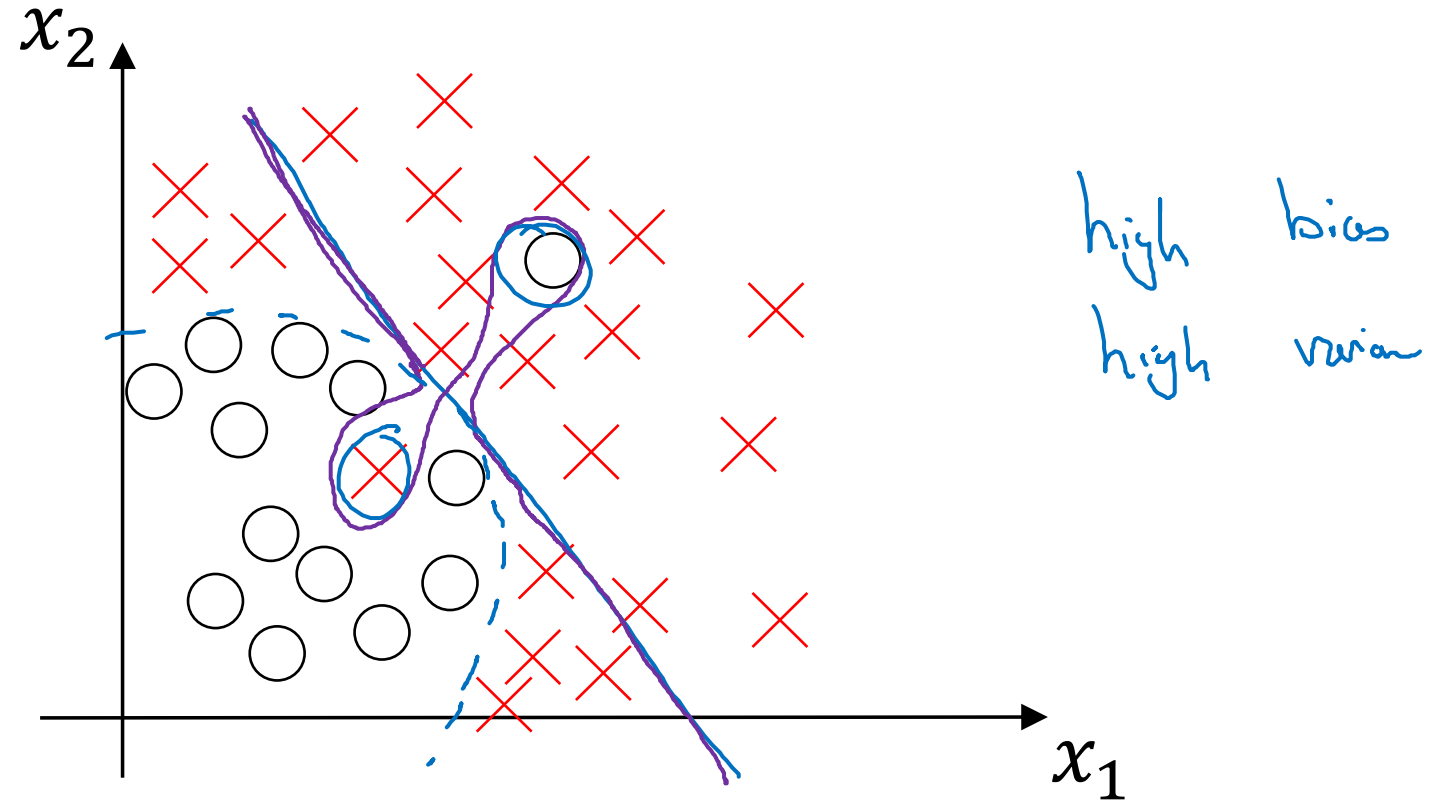
low bias
low variance
 \uparrow

Human: ~0%

Optimal (Bayes) error: ~~~0%~~ 15%

Blurry images

High bias and high variance



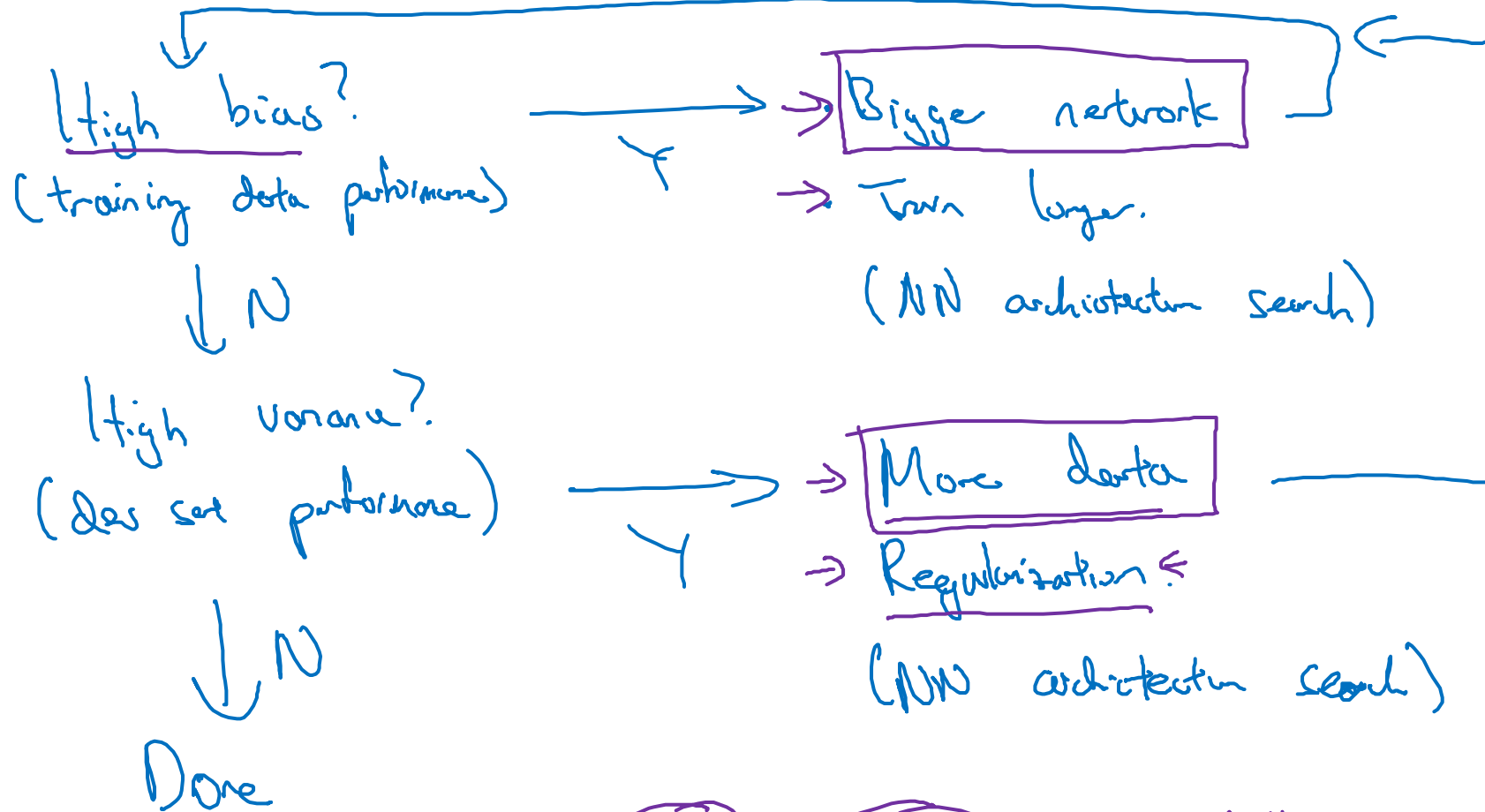


deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning



Analizzando l'errore sul set di addestramento e sul set di sviluppo, è possibile identificare se il modello ha bias elevato (underfitting) o varianza elevata (overfitting) e adottare strategie mirate per migliorarlo.

Bias Elevato (Underfitting)

Se il modello non si adatta bene ai dati di addestramento:

Aumentare la complessità della rete (più livelli o unità nascoste)

Addestrare più a lungo

Utilizzare algoritmi di ottimizzazione avanzati

Provare diverse architetture di rete

Varianza Elevata (Overfitting)

Se il modello si adatta bene al training set ma non generalizza:

Aumentare i dati di addestramento

Applicare la regolarizzazione

Provare un'architettura più adatta





deeplearning.ai

Regularizing your neural network

Regularization

Riassunto sulla Regularizzazione in Regressione Logistica e Reti Neurali

La regolarizzazione è una tecnica utilizzata per prevenire l'overfitting nei modelli di machine learning, migliorando la generalizzazione su nuovi dati.

Regularizzazione in Regressione Logistica

- L2 (Ridge Regression o Penalità L2): aggiunge un termine $\frac{\lambda}{2} \|w\|^2$ alla funzione di costo, penalizzando pesi elevati.
- L1 (Lasso Regression o Penalità L1): aggiunge un termine $\lambda \|w\|_1$, che può portare alla sparsità (eliminazione di alcune feature).

Regularizzazione in Reti Neurali

- L2 Regularization (Weight Decay): penalizza pesi grandi per evitare modelli troppo complessi.
- Dropout: disattiva casualmente neuroni durante il training per evitare dipendenze eccessive tra unità.
- Batch Normalization: normalizza gli input nei livelli nascosti per rendere l'allenamento più stabile.

L'uso della regolarizzazione aiuta a bilanciare bias e varianza, migliorando le prestazioni del modello su dati nuovi.

Logistic regression

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

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

Lambda
 $\lambda =$ regularization parameter
lambda lambda

$$J(w,b) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(y^{(i)}, \hat{y}^{(i)})}_{\text{cost function}} + \frac{\lambda}{2m} \underbrace{\|w\|_2^2}_{\text{L2 regularization}}$$

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

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

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

w will be sparse

Neural network

$$\rightarrow J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \underbrace{\frac{1}{n} \sum_{i=1}^n \ell(y^{(i)}, \hat{y}^{(i)})}_{\text{loss}} + \underbrace{\frac{\lambda}{2n} \sum_{l=1}^L \|w^{[l]}\|_F^2}_{\text{weight decay}}$$

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

$w^{[l]}: \begin{matrix} n^{[l]} & n^{[l-1]} \\ \uparrow & \uparrow \end{matrix}$

"Frobenius norm"

$\|\cdot\|_2^2$

$\|\cdot\|_F^2$

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

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

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

"Weight decay"

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

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

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

Regolarizzazione e la Riduzione della Varianza

La regolarizzazione aiuta a prevenire l'overfitting riducendo la varianza del modello. L'overfitting si verifica quando il modello è troppo complesso e si adatta troppo ai dati di training, catturando anche il rumore.

Perché la regolarizzazione aiuta?

Aggiunge un termine alla funzione di costo che penalizza i pesi troppo grandi, forzando il modello a mantenere valori più piccoli e quindi meno flessibili. Questo riduce la capacità del modello di adattarsi eccessivamente ai dati di training, migliorando la generalizzazione sui nuovi dati.

Regolarizzazione L2 (Norma di Frobenius)

L2 penalizza i pesi elevati aggiungendo il termine $\|W\|^2$ alla funzione di costo. Se $\|W\|^2$ è molto grande:

- I pesi W saranno molto piccoli, riduce l'effetto di molte unità nascoste.
- La rete neurale diventa più semplice, simile a una regressione logistica con più livelli.
- Il modello diventa meno capace di creare limiti decisionali complessi, riducendo l'overfitting.

Effetto su una rete con attivazione tanh

- Se W è piccolo, allora $z = Wx + b$ sarà piccolo.
- Con z piccolo, $\tanh(z)$ è quasi lineare, la rete diventa più simile a una rete lineare, quindi meno complessa e meno incline a sovradimensionare i dati.

Conclusione:

Aumentando λ , la rete si comporta come una versione semplificata, riducendo la varianza e migliorando la generalizzazione. Tuttavia, non deve essere troppo grande, altrimenti il modello diventa troppo semplice e rischia di soffrire di underfitting.

Nel deep learning, oltre alla regolarizzazione L2, esistono altre tecniche come dropout, che verranno esplorate successivamente.

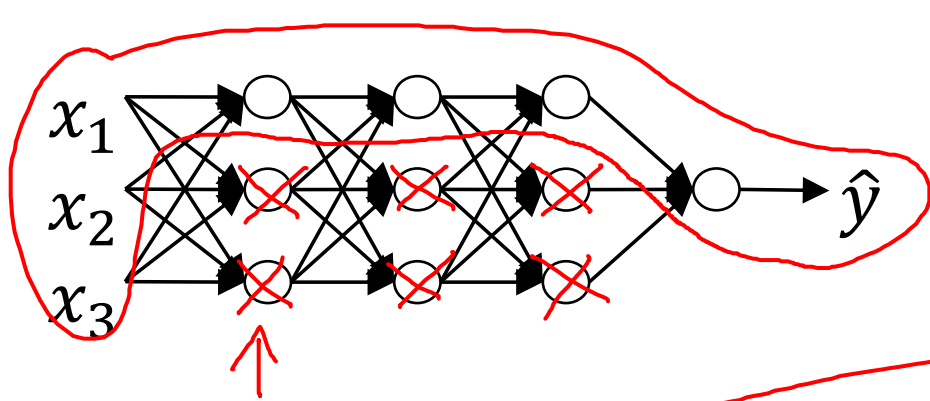
Regularizing your neural network



deeplearning.ai

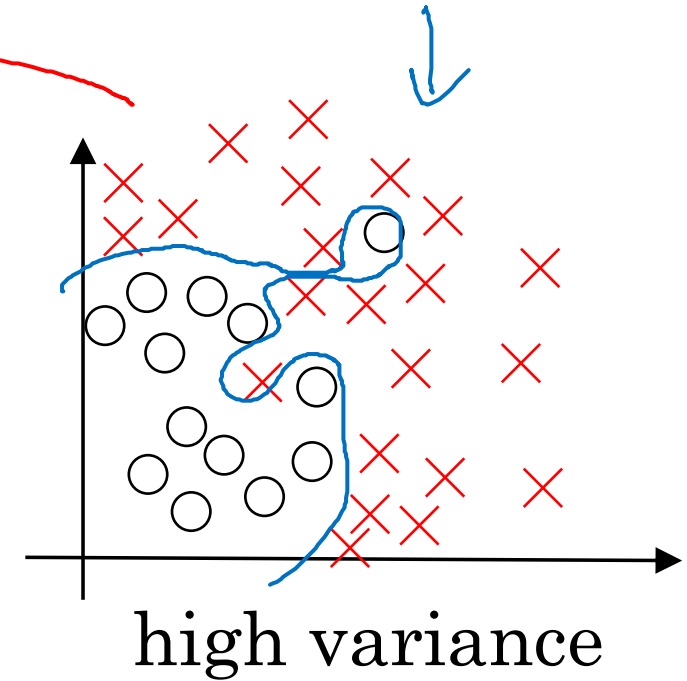
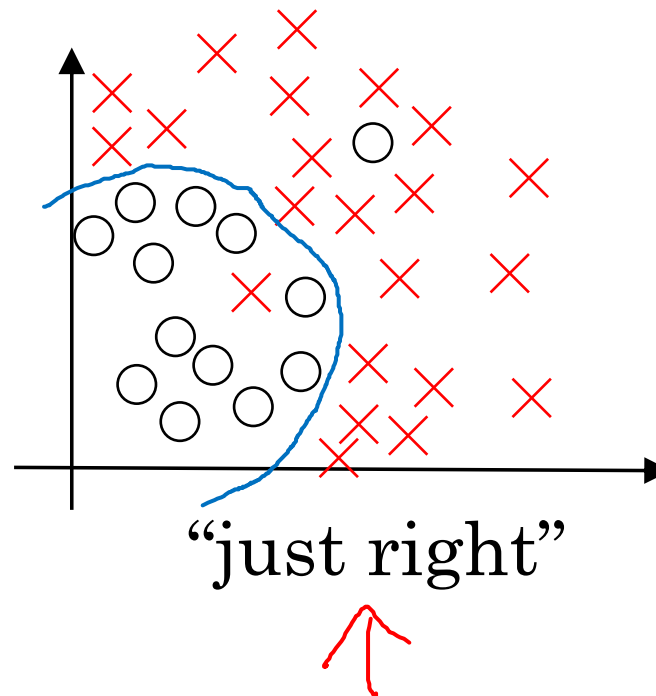
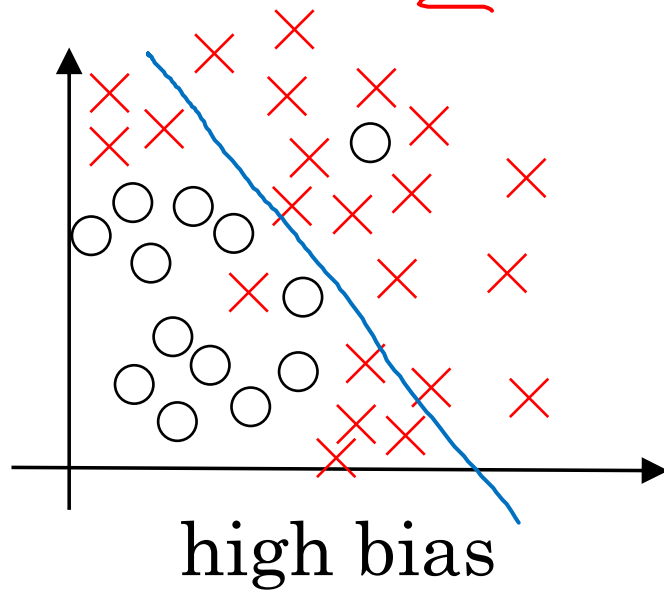
Why regularization reduces overfitting

How does regularization prevent overfitting?

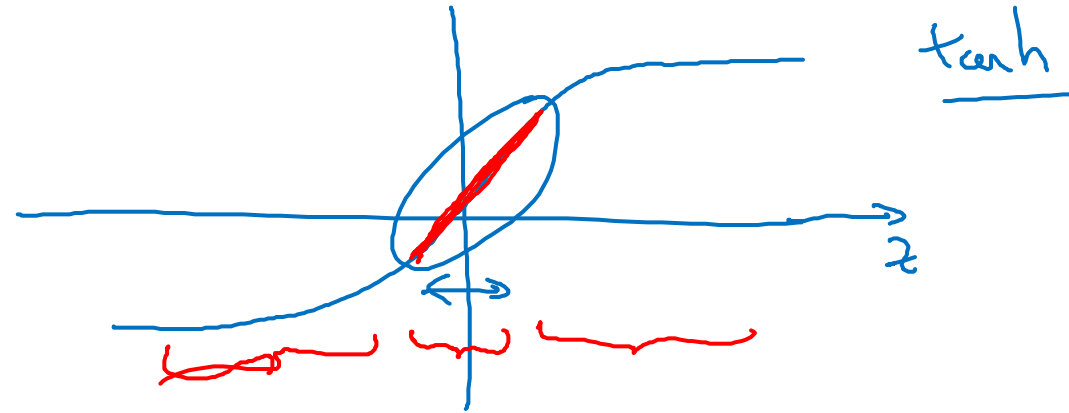


$$J(\mathbf{w}^{(L)}, \mathbf{b}^{(L)}) = \frac{1}{n} \sum_{i=1}^n \ell(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2n} \sum_{l=1}^L \underbrace{\|\mathbf{w}^{(l)}\|_F^2}_{\text{regularization}}$$

$$\mathbf{w}^{(L)} \approx 0$$



How does regularization prevent overfitting?



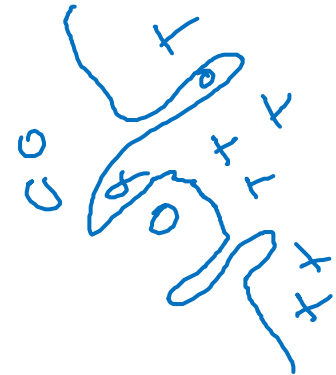
$$g(z) = \tanh(z)$$

$\lambda \uparrow$

$W^{[L]} \downarrow$

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

Every layer \approx linear.



$$J(\dots) = \underbrace{\sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)})}_{\text{training loss}} + \underbrace{\frac{\lambda}{2m} \sum_L \|W^{[L]}\|_F^2}_{\text{regularization term}}$$



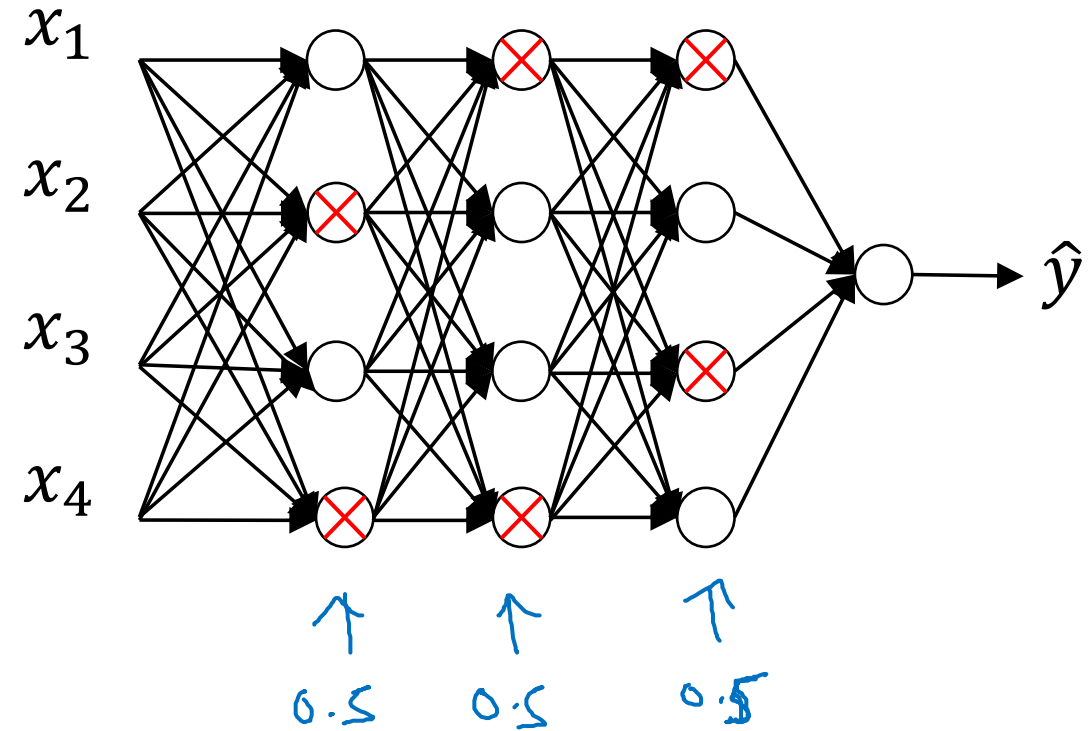
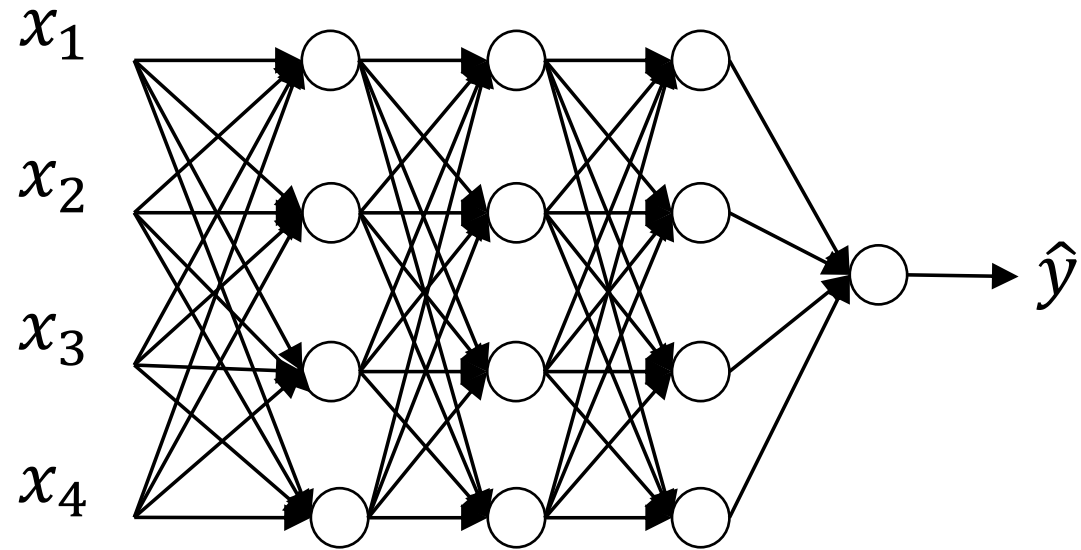


deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization



Implementing dropout ("Inverted dropout")

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

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

$a3 = \text{np.multiply}(a3, d3)$ # $a3 \neq d3$.

→ $a3 /= \text{keep-prob}$ ←

50 units. \leadsto 10 units shut off

$$z^{[4]} = w^{[4]} \cdot \underbrace{a^{[3]}}_{\text{reduced by } 20\%} + b^{[4]}$$

\uparrow

reduced by 20%

$$/= \underline{0.8}$$

Test

Making predictions at test time

$$a^{[0]} = X$$

No drop out.

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

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

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

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

↓
↑
 \hat{y}

\neq keep-prob



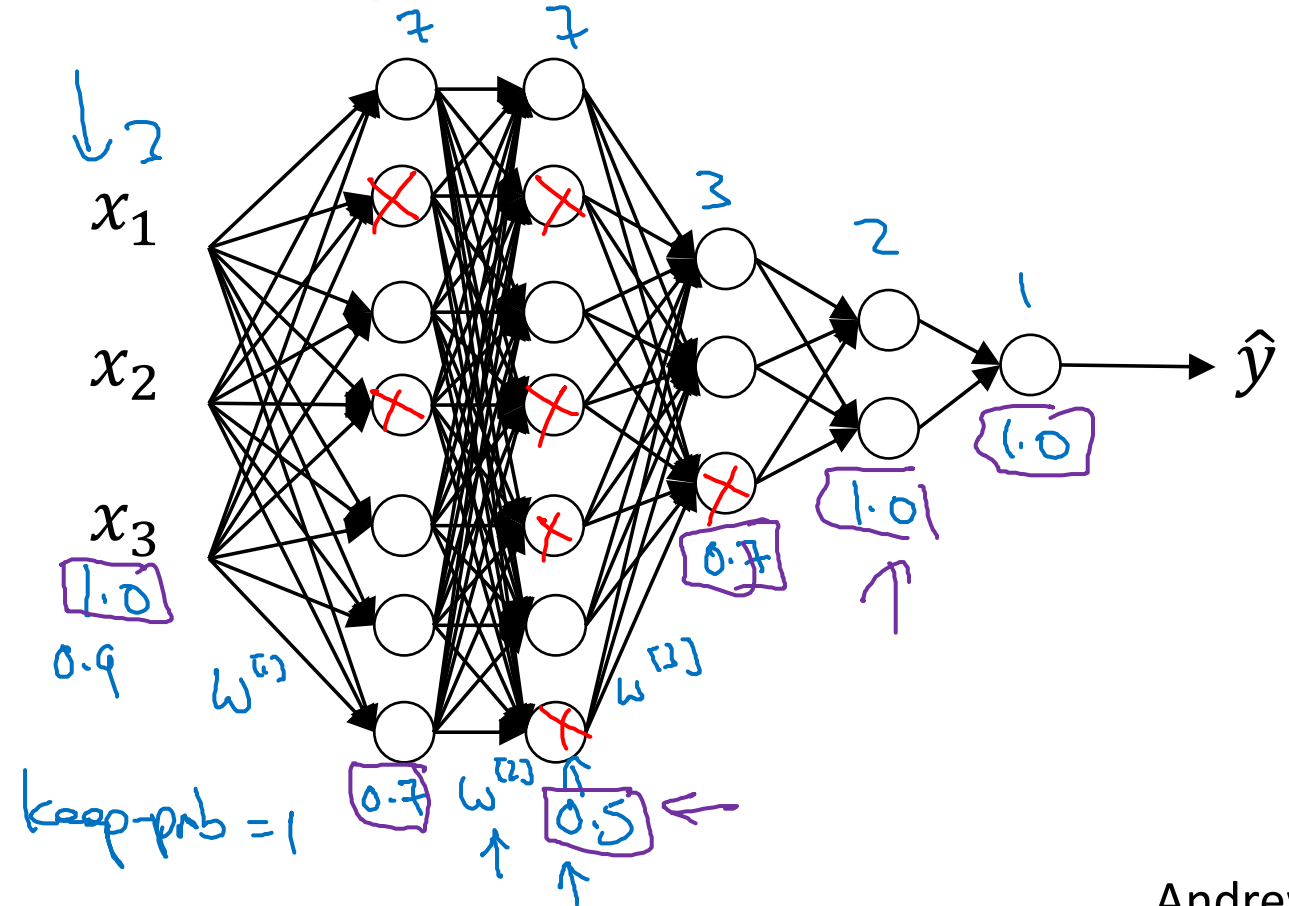
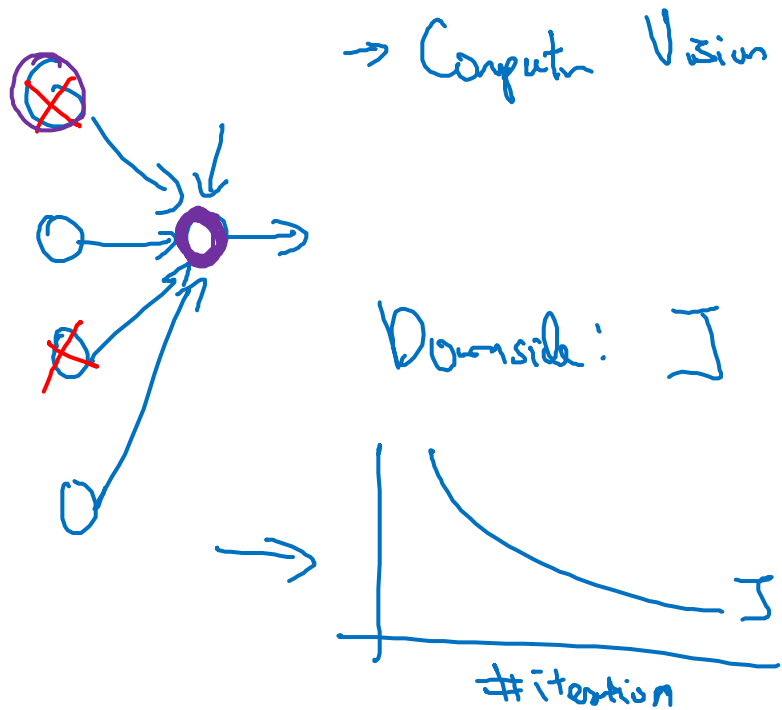
deeplearning.ai

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. \leadsto Shrink weights. b_2





deeplearning.ai

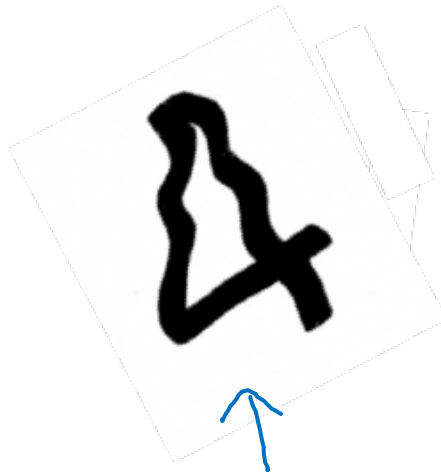
Regularizing your neural network

Other regularization methods

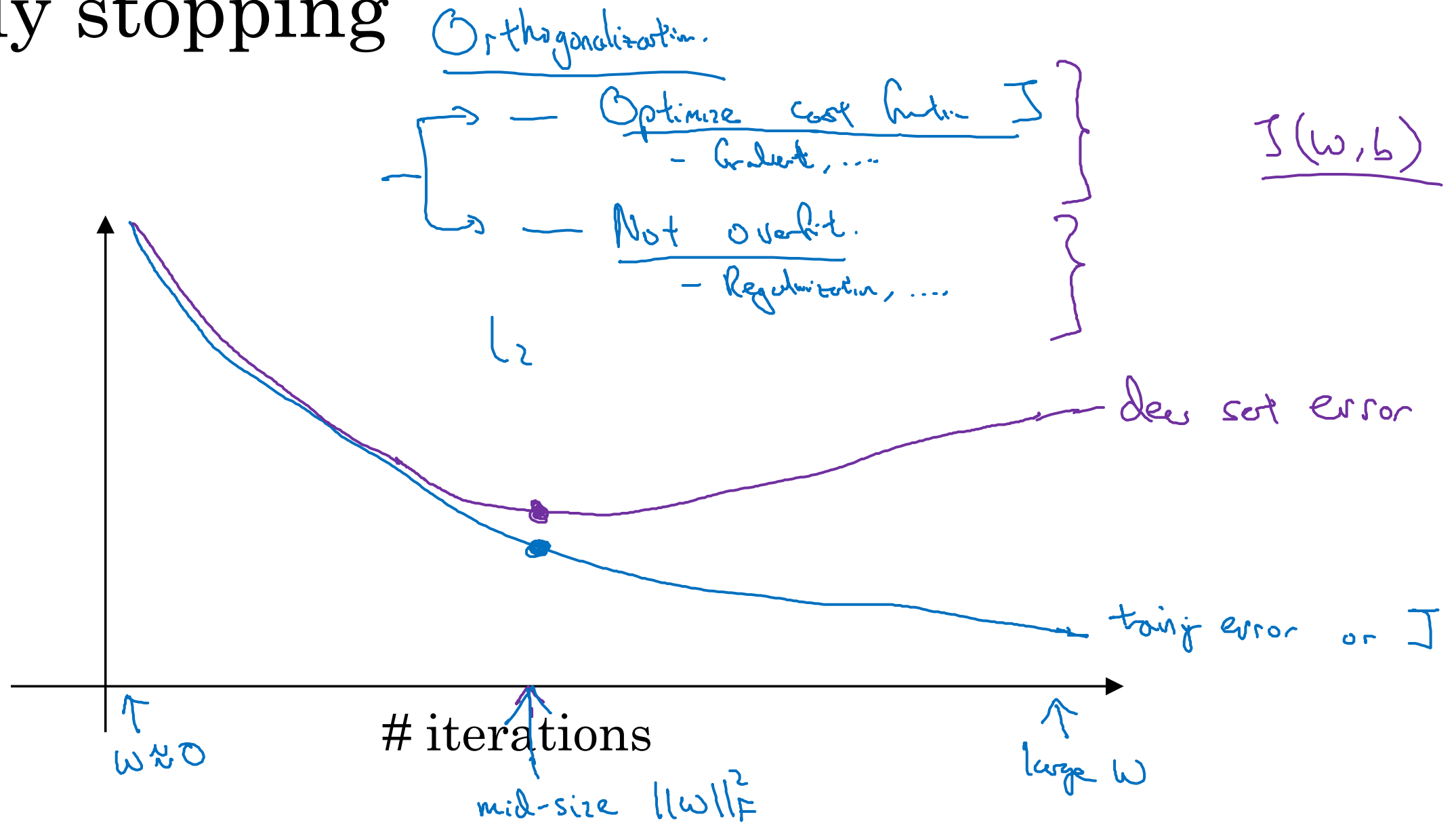
Data augmentation



4



Early stopping





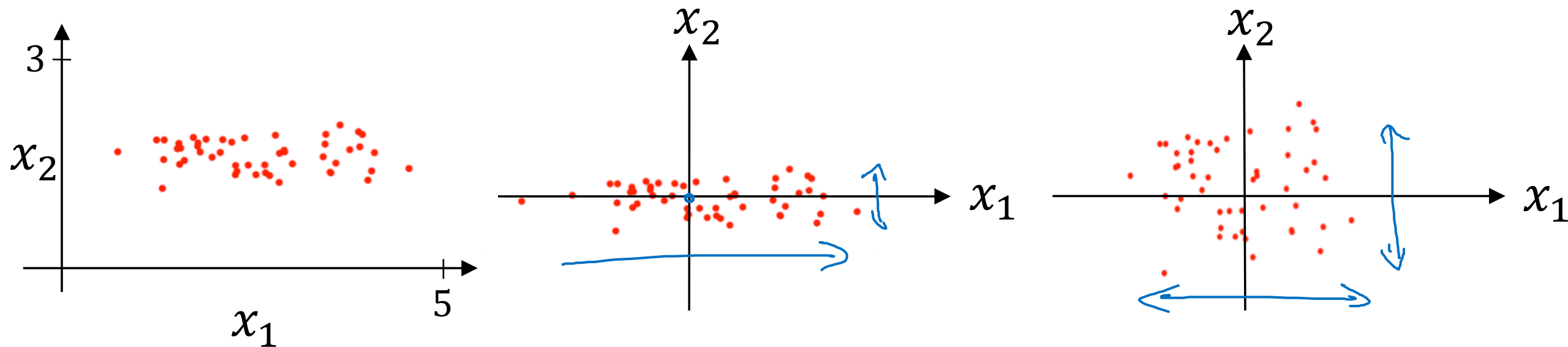
deeplearning.ai

Setting up your
optimization problem

Normalizing inputs

Normalizing training sets

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



Subtract mean:

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

$$x := x - \mu$$

Normalize variance

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

← element-wise

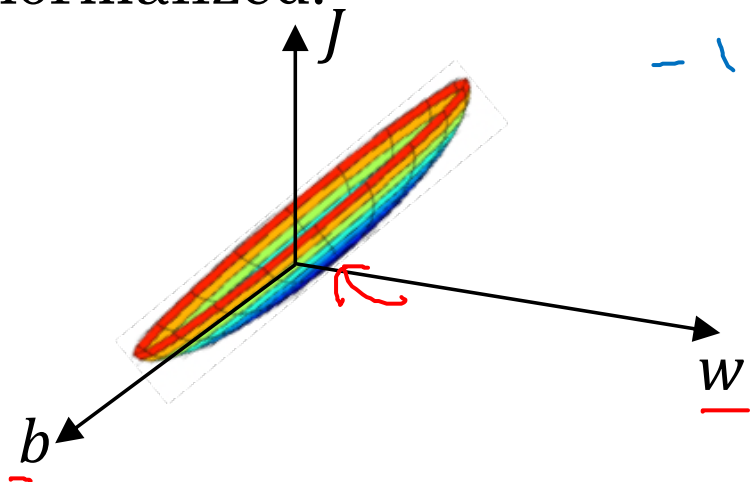
$$x /= \sigma^2$$

Use same μ σ^2 to normalize test set.

Why normalize inputs?

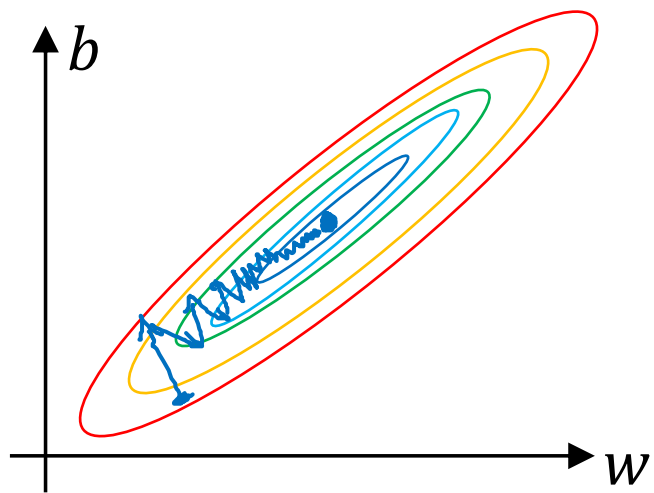
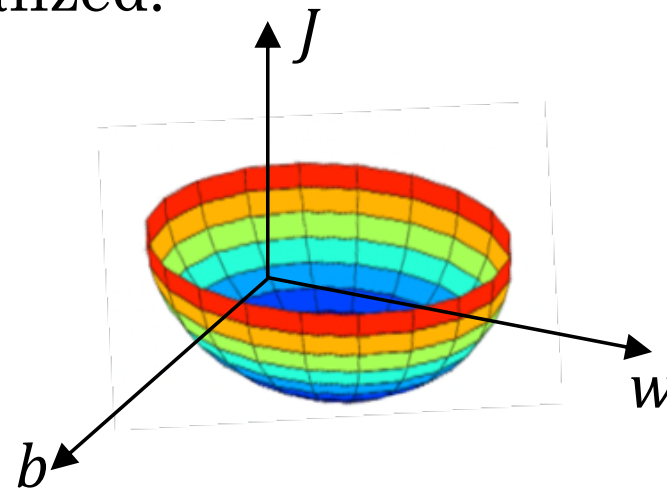
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Unnormalized:

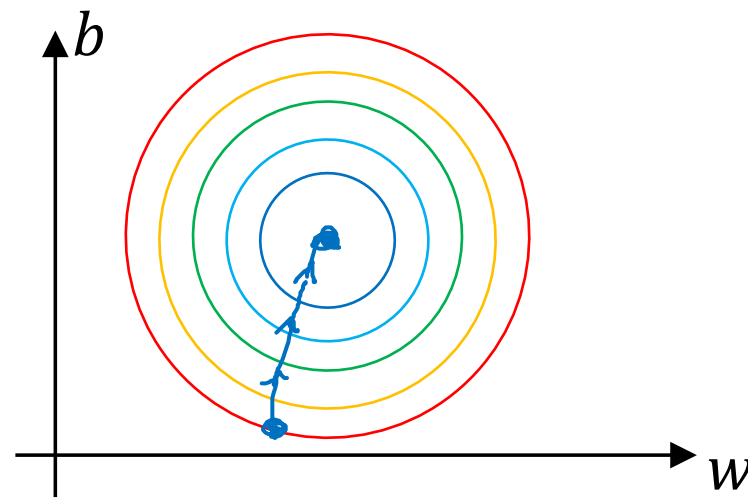


$w_1: x_1: \underline{1 \dots 1000} \leftarrow$
 $w_2: x_2: \underline{0 \dots 1} \leftarrow$
 $\quad \quad \quad -1 \dots 1$

Normalized:



$x_1: 0 \dots 1$
 $x_2: -1 \dots 1$
 $x_3: 1 \dots 2$



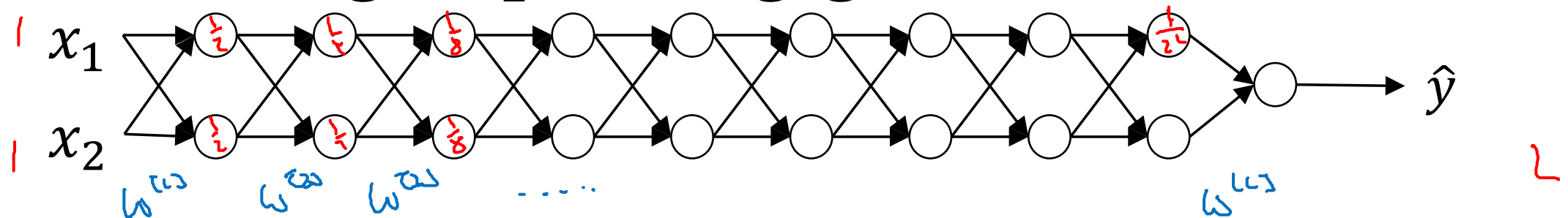


deeplearning.ai

Setting up your
optimization problem

Vanishing/exploding
gradients

Vanishing/exploding gradients



$$g(z) = z \quad b^{(L)} = 0$$

$$\hat{y} = w^{(L)} \left(w^{(L-1)} w^{(L-2)} \dots \left(w^{(2)} w^{(1)} x \right) \right)$$

$$w^{(1)} > I$$

$$w^{(2)} < I \quad \begin{bmatrix} 0.9 & \\ & 0.9 \end{bmatrix}$$

$$w^{(2)} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

0.5 (above 1.5)
6.5 (below 1.5)

$$z^{(1)} = w^{(1)} x$$

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

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

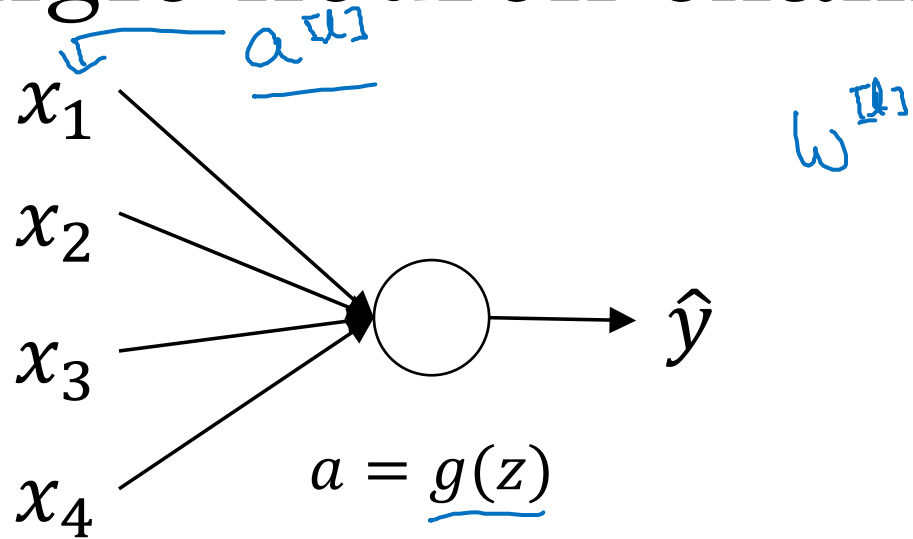
$$\hat{y} = w^{(L)} \left[\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x \right]$$

6.5 (above 1.5)
0.5 (below 1.5)

$$1.5^{L-1} x$$

$$0.5^{L-1} x$$

Single neuron example



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

large $n \rightarrow$ Smaller w_i

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

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

ReLU $g^{[2]}(z) = \text{ReLU}(z)$

Other variants:

tanh

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

Xavier initialization ↑

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

↑



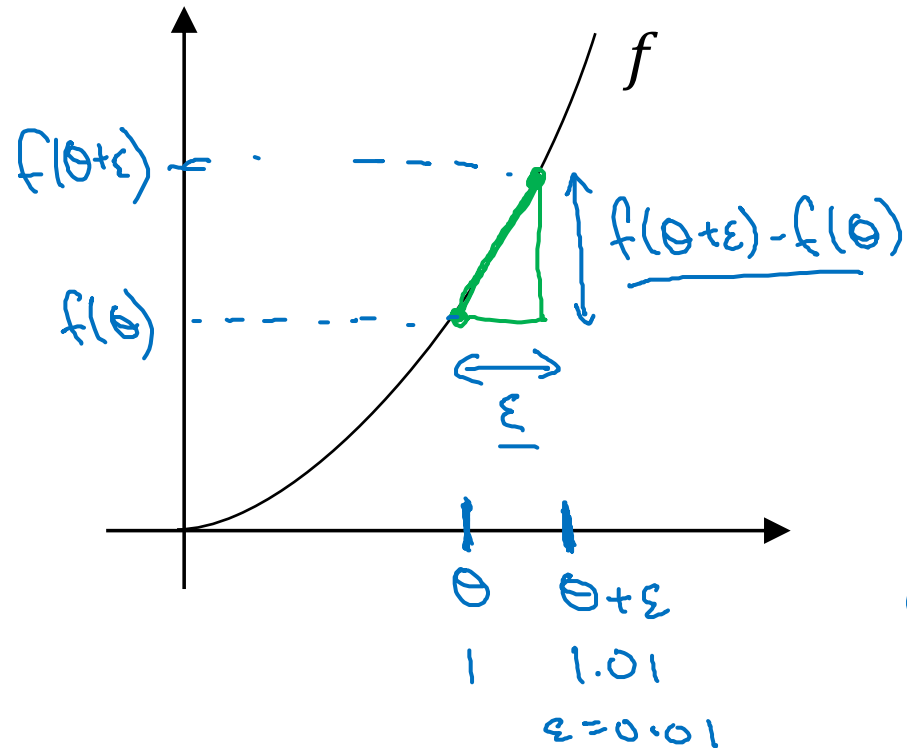
deeplearning.ai

Setting up your
optimization problem

Numerical approximation
of gradients

Checking your derivative computation

I $f(\theta) = \theta^3$
 $\theta \in \mathbb{R}.$



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

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

$\frac{dw}{db}$

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

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

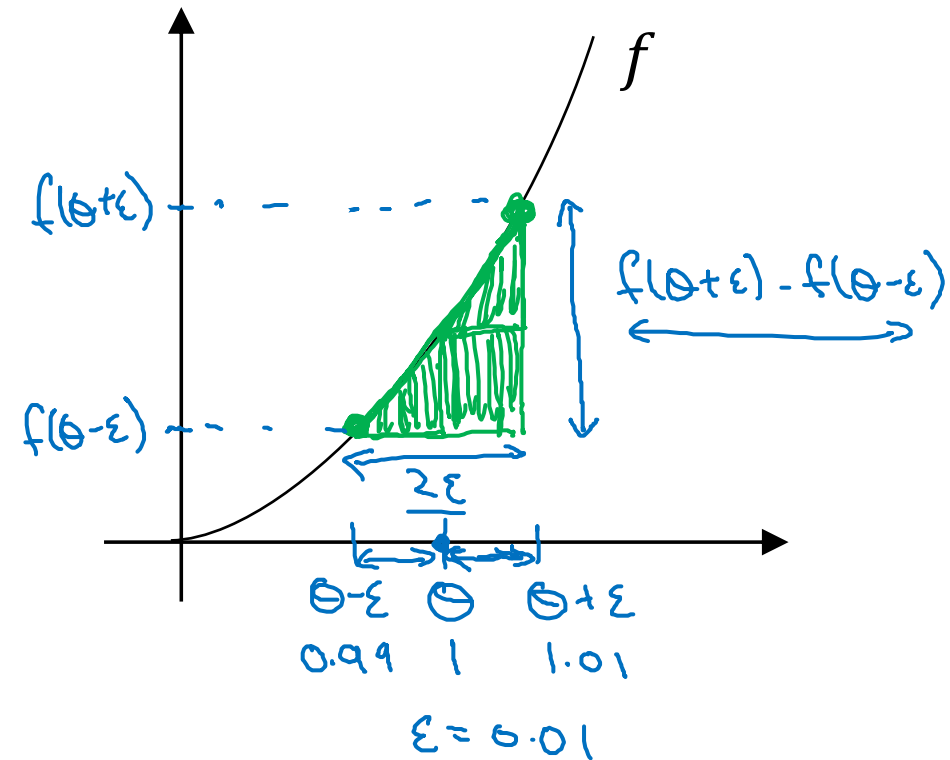
$$\frac{(1.01)^3 - 1^3}{0.01} = \frac{1.030301 - 1}{0.01} = \frac{0.0301}{0.01} = 3.0301 \approx 3$$

Annotations: 0.0301 is the numerator, 3.1 and 3.2 are the integer parts of the division, and 3 is the final result.

$\theta = 1$
 $\theta + \epsilon = 1.01$

Checking your derivative computation

$$\underline{f(\theta) = \theta^3}$$



$$\left[\frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon} \approx \underline{g(\theta)} \right]$$

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

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

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

$$\left\{ \begin{array}{l} f'(\theta) = \lim_{\epsilon \rightarrow 0} \frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon} \quad \begin{array}{l} \mathcal{O}(\epsilon^2) \\ 0.01 \\ \underline{0.0001} \end{array} \quad \left| \quad \begin{array}{l} \frac{f(\theta + \epsilon) - f(\theta)}{\epsilon} \quad \text{error: } \mathcal{O}(\epsilon) \\ \uparrow \quad \uparrow \\ 0.01 \end{array} \end{array} \right.$$



deeplearning.ai

Setting up your
optimization problem

Gradient Checking

Gradient check for a neural network

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

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

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

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

Gradient checking (Grad check)

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

for each i :

$$\rightarrow \underline{d\theta_{\text{approx}}[i]} = \frac{J(\theta_1, \theta_2, \dots, \overset{\downarrow}{\theta_i + \epsilon}, \dots) - J(\theta_1, \theta_2, \dots, \overset{\downarrow}{\theta_i - \epsilon}, \dots)}{2\epsilon}$$

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

Checks

$$\rightarrow \frac{\|d\theta_{\text{approx}} - d\theta\|_2}{\|d\theta_{\text{approx}}\|_2 + \|d\theta\|_2}$$
$$\underline{\epsilon = 10^{-7}}$$

$$\approx \frac{10^{-7}}{10^{-5}} - \text{great!} \leftarrow$$
$$\rightarrow 10^{-3} - \text{worry.} \leftarrow$$



deeplearning.ai

Setting up your
optimization problem

Gradient Checking
implementation notes

Gradient checking implementation notes

- Don't use in training – only to debug

$$\frac{d\theta_{\text{approx}}[\vec{i}]}{\uparrow \uparrow} \longleftrightarrow \frac{d\theta[\vec{i}]}{\uparrow}$$

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

$$\frac{db^{[L]}}{\uparrow} \quad \frac{dW^{[L]}}{\uparrow}$$

- Remember regularization.

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

$d\theta = \text{gradient of } J \text{ wrt. } \theta$

- Doesn't work with dropout.

J

$$\underline{\text{keep-prob} = 1.0}$$

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

$$\underline{W, b \approx 0}$$