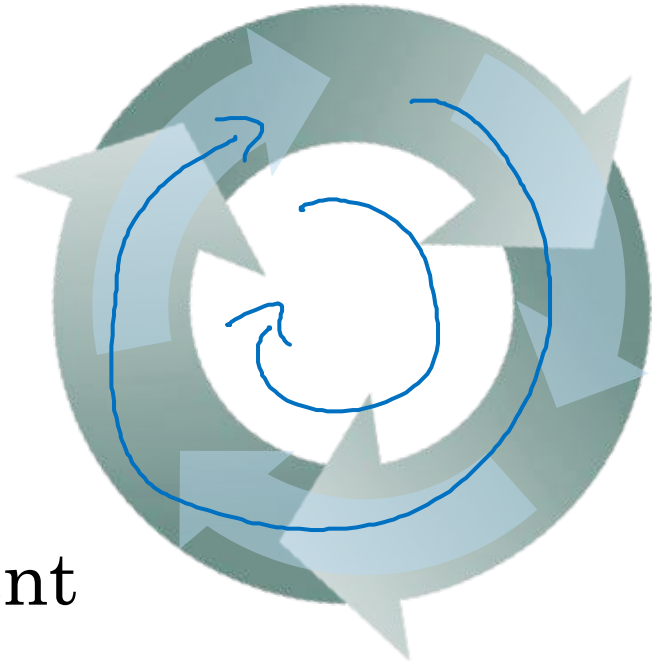
deeplearning.ai

Setting up your ML application

Train/dev/test sets

Applied ML is a highly iterative process

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

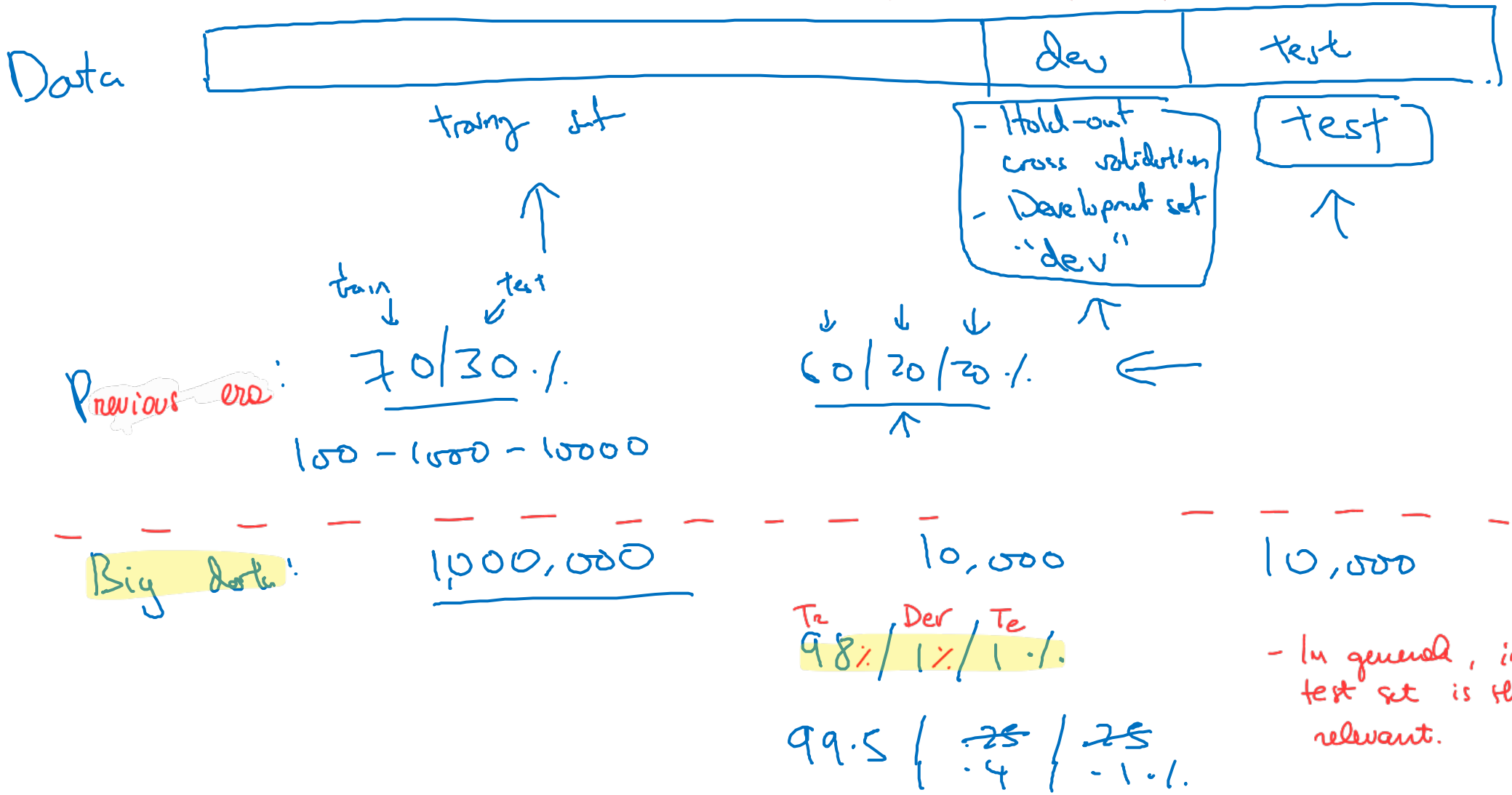


NLP, Vision, Speech, Structured data

Ads Search Security Logistic ....

Train/dev/test sets

- In a traditional "small data" setting a typical train/test split is 70/30 %.
- In a big data setting it makes more sense to have the test & development (= validation) sets much smaller ≤ 1% each.



- In general, in big data if test set is the least relevant.

Mismatched train/test distribution

Training set:
Cat pictures from webpages

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

- It is common to have non i.d. TRAIN & TEST SETS. This is fine as long as the validation set is i.d. to the test set (because the network is optimized, using the dev set, to work on the target distribution).

→ Make sure dev and test come from same distribution.

"test"

train / dev

train / test

→ train / dev

- WEIRD. Some groups not always use a separate validation (dev) and test set. They only have train & dev (called "hold out dev. set" or, improperly, test set) and to train & test on those, although this is improper because they are overfitting on the test set.

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

IF ONE DOESN'T NEED AN UNBIASED ESTIMATE OF THE MODEL!!

↳ Sounds like BS to me ...

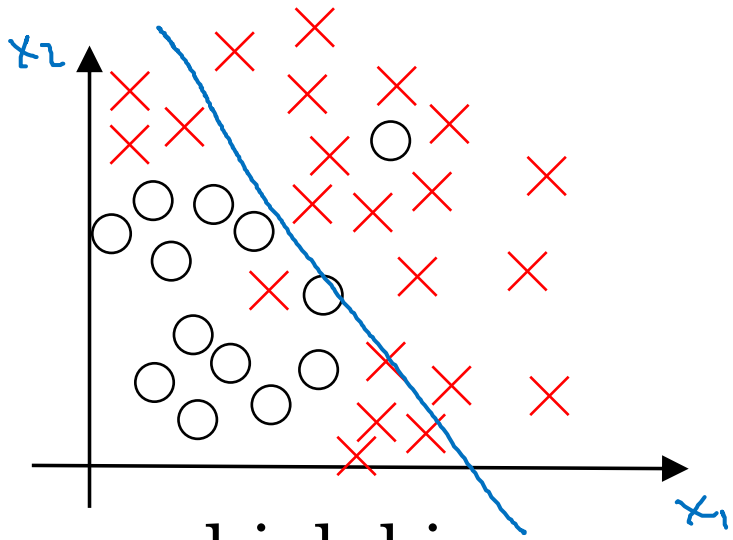
Andrew Ng

deeplearning.ai

Setting up your ML application

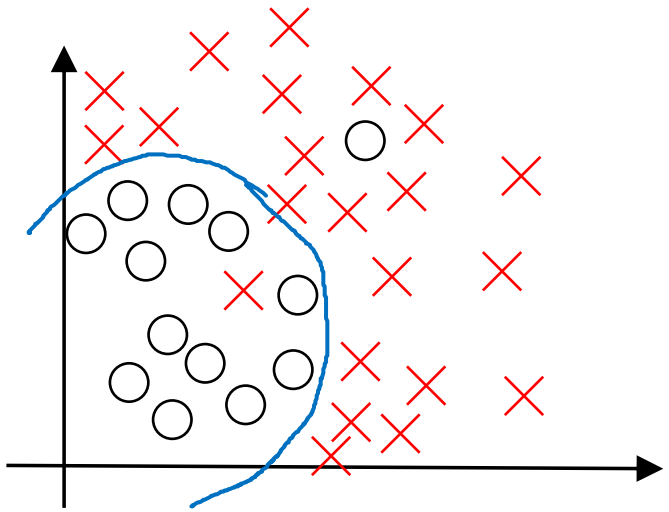
Bias/Variance

Bias and Variance

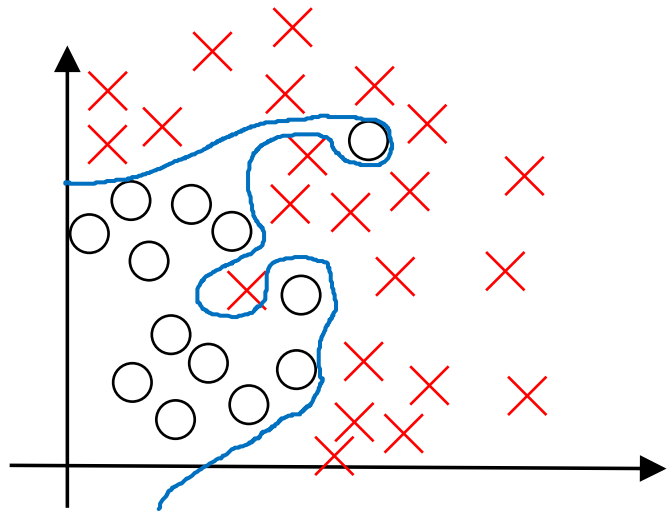


high bias

Underfitting



→ "just right"



high variance

Overfitting

Bias and Variance

→ Metrics to evaluate BIAS & VARIANCE are the TRAIN & DEV SET ERRORS.

→ We want both low train error and dev set error.

Cat classification

Train set error:	1% very good	15% not good	15% not good	0.5% very good
Dev set error:	11% very bad	16% not good	30% even worse...	1% still very good
	high variance	high bias	high bias & high variance	low bias low variance

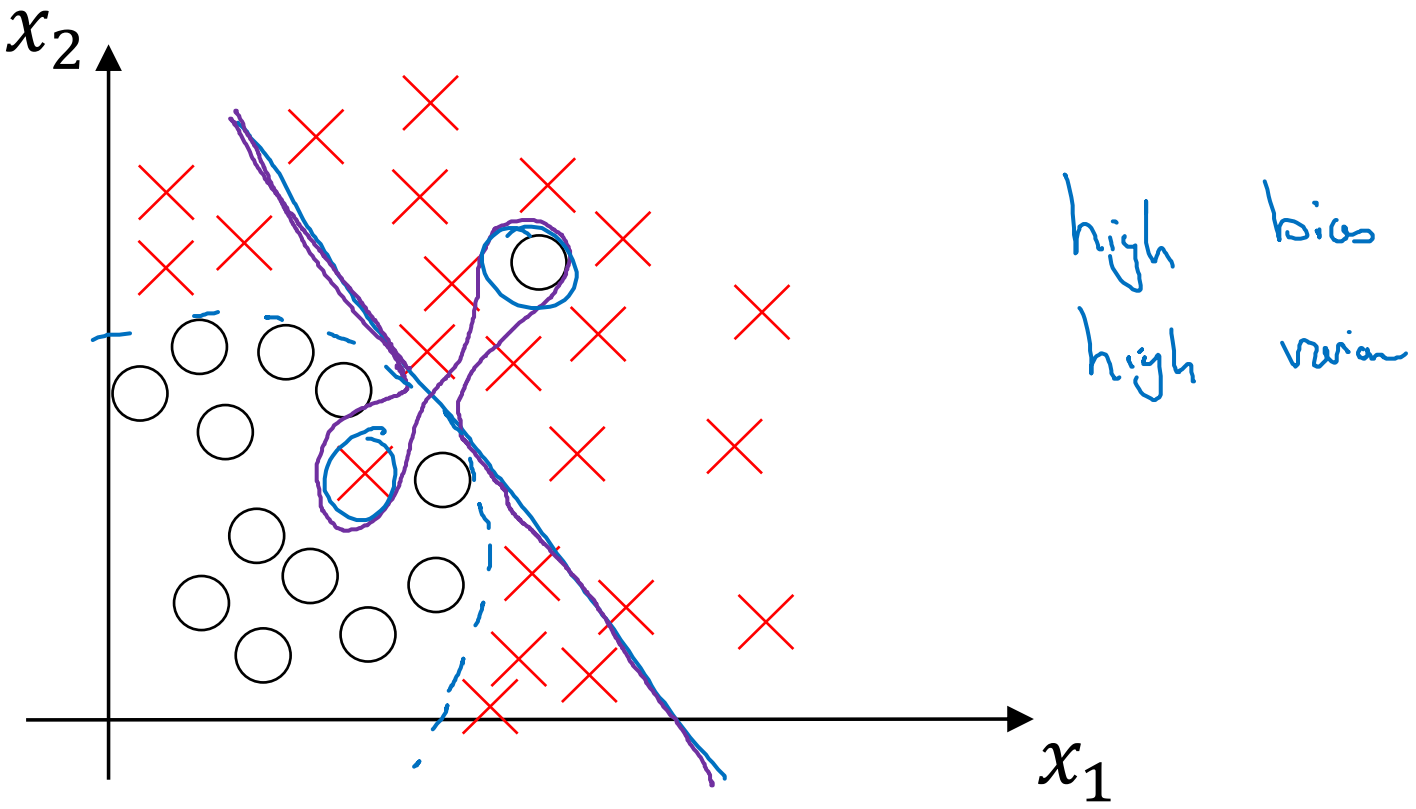
Human: ~0%

Optimal (Bayes) error: 15%

Blurry images

Andrew Ng

High bias and high variance



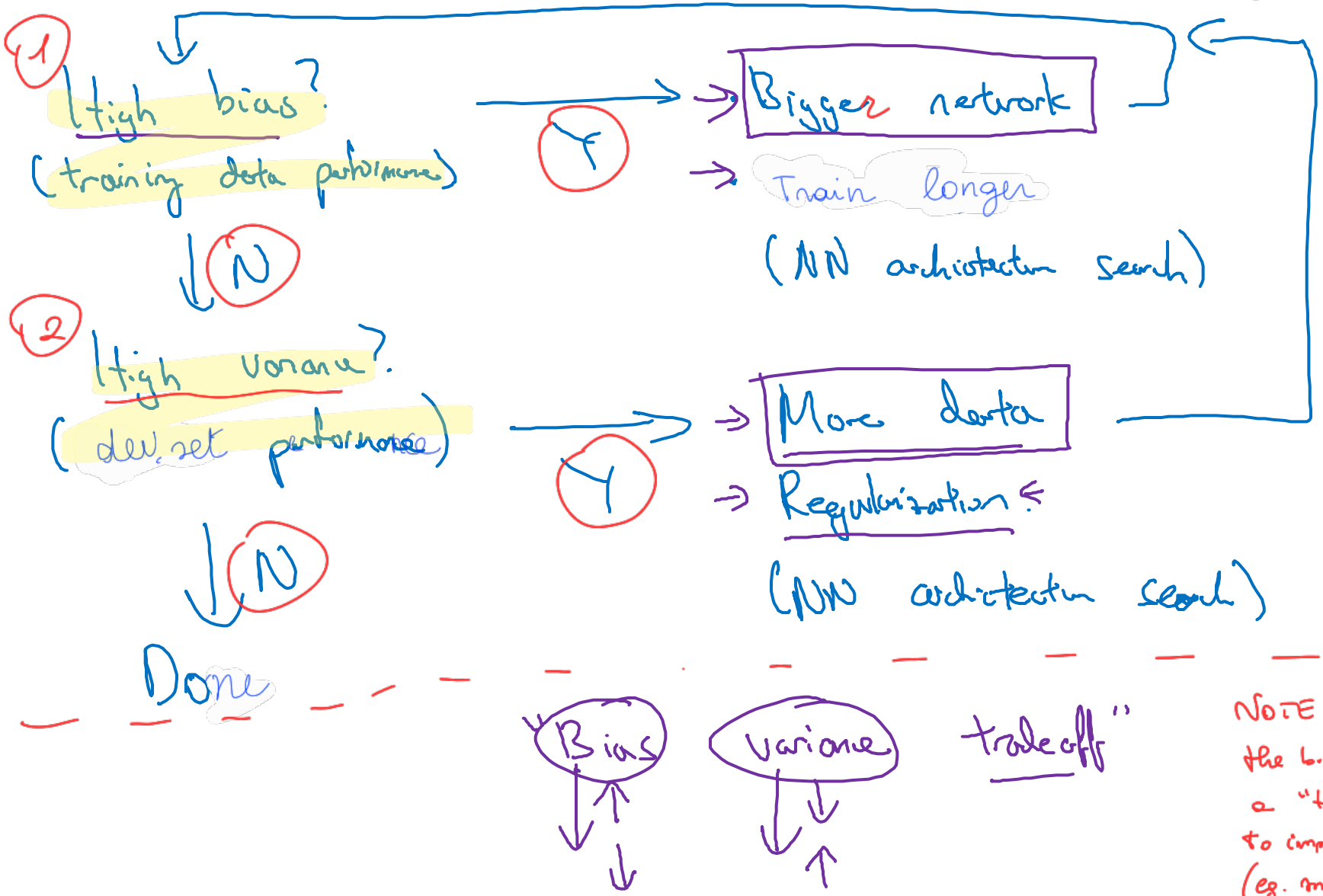
deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning

How to reduce both BIAS AND VARIANCE



NOTE: for deep learning applications, the bias/variance tradeoff is less of a "tradeoff" because there are options to improve (reduce) one without hurting the other.
(eg. more depth → reduce bias but does not impact variance that much.
more data → reduces variance without impacting bias that much)

deeplearning.ai

Regularizing your neural network

Regularization

Is another way to reduce variance (often than increasing the data)

example: regularization in logistic regression

Logistic regression

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

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

λ = regularization parameter

lambda lambd

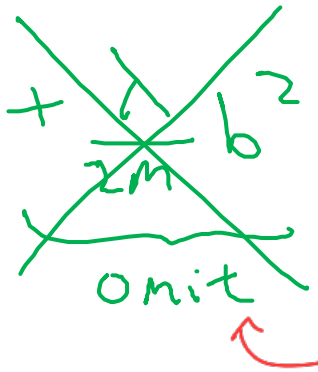
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

L_2 regularization

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

L_1 regularization

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



often the bias is left unregularized because it is only a scalar, while w can be high dimensional and contribute more to the variance.

w will be sparse

Neural network

- If we apply the Frob. regularization to the weights (new term in the cost function) the gradients of the cost function are: $-$SAME $\frac{\partial J}{\partial b^{[l]}}$, $\frac{\partial J}{\partial W^{[l]}} = \frac{\partial J}{\partial W^{[l]}} + \frac{\lambda}{m} W^{[l]}$.

$$\rightarrow J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \underbrace{\frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})}_{\text{average loss}} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^{L} \|W^{[l]}\|_F^2}_{\text{regularization term}}$$

($L \rightarrow$ # samples)

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

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

① "Frobenius norm" $\|\cdot\|_2^2$ $\|\cdot\|_F^2$

② GRADIENT DESCENT with regularized weights

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

$$\rightarrow W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$$

$$\frac{\partial J}{\partial W^{[l]}} = dW^{[l]}$$

③ a.k.a. "Weight decay"

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

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

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

intuition for naming "weight decay": at each update the weights are updated by multiplying by a number < 1.

Andrew Ng

deeplearning.ai

Regularizing your neural network

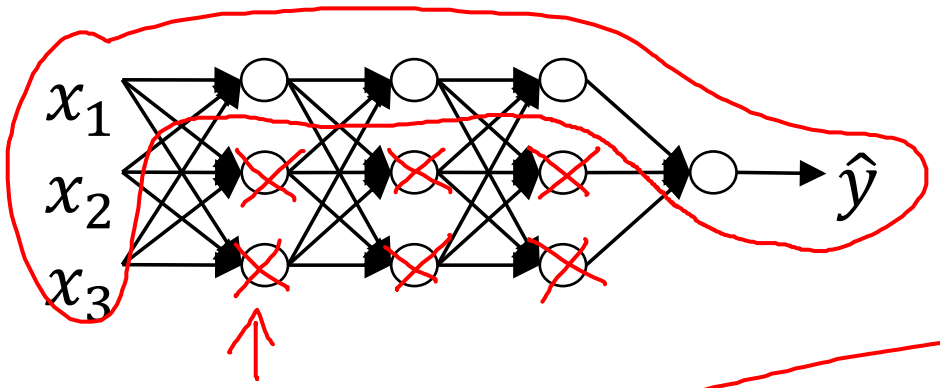
Why regularization reduces overfitting

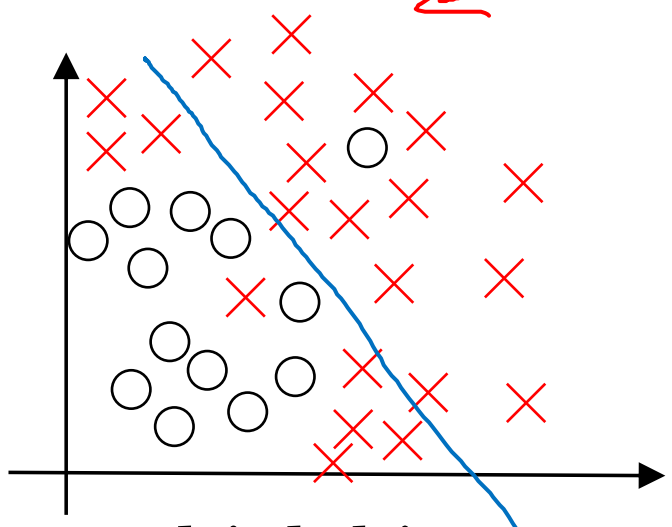
How does regularization prevent overfitting?

- INTUITION 1: regularization reduces nonlinearity by dampening or killing some neurons.

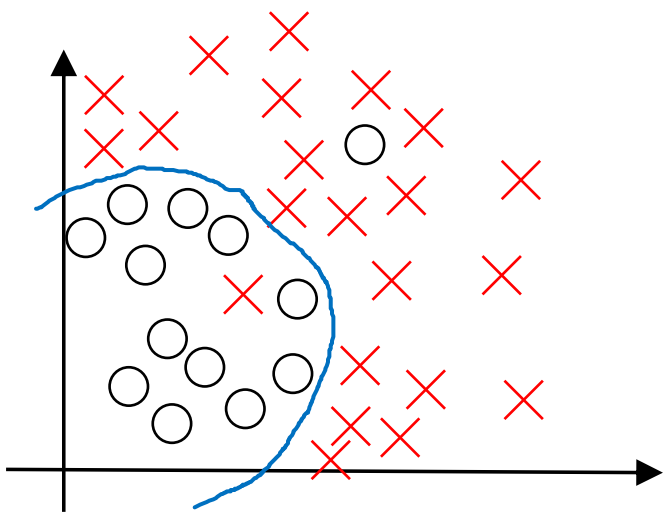
$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

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

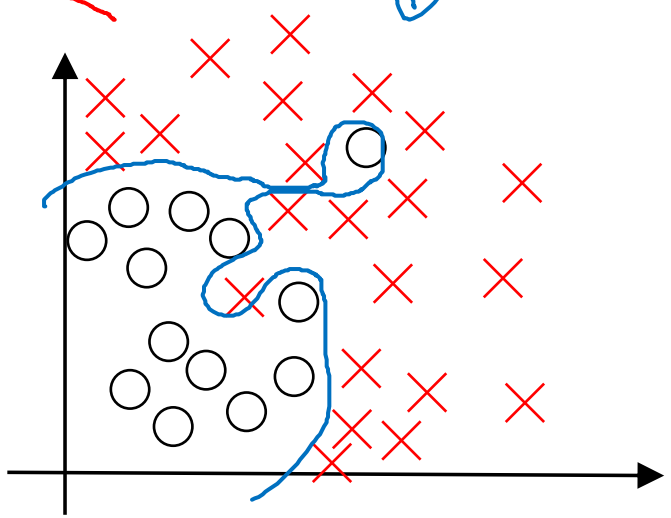




high bias



"just right"

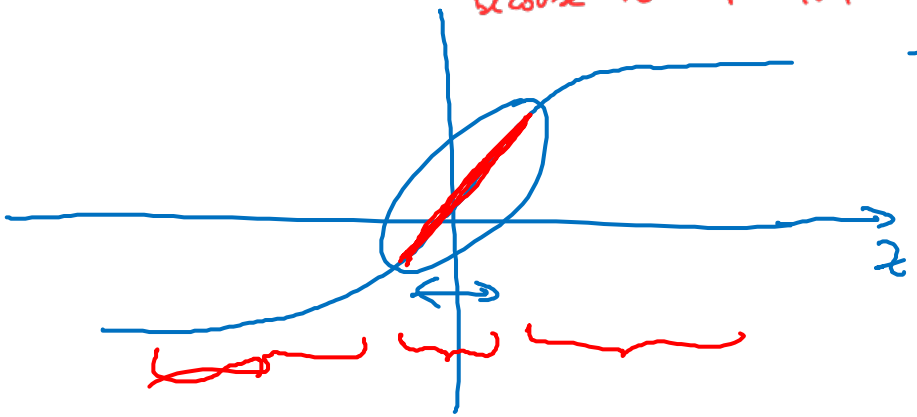


high variance

Andrew Ng

How does regularization prevent overfitting?

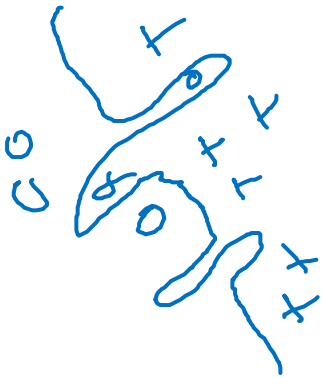
INTUITION 2: Regularization reduces nonlinearity in the case of sigmoidal activations because it keeps $|W|$ low $\Rightarrow$ $|z|$ low $\Rightarrow$ sigmoidal activations remain in the "linear regime".



tanh

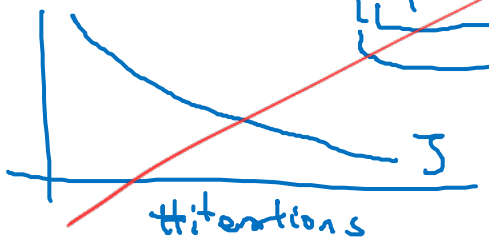
$g(z) = \tanh(z)$

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



$\rightarrow$ Every layer $\approx$ linear

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



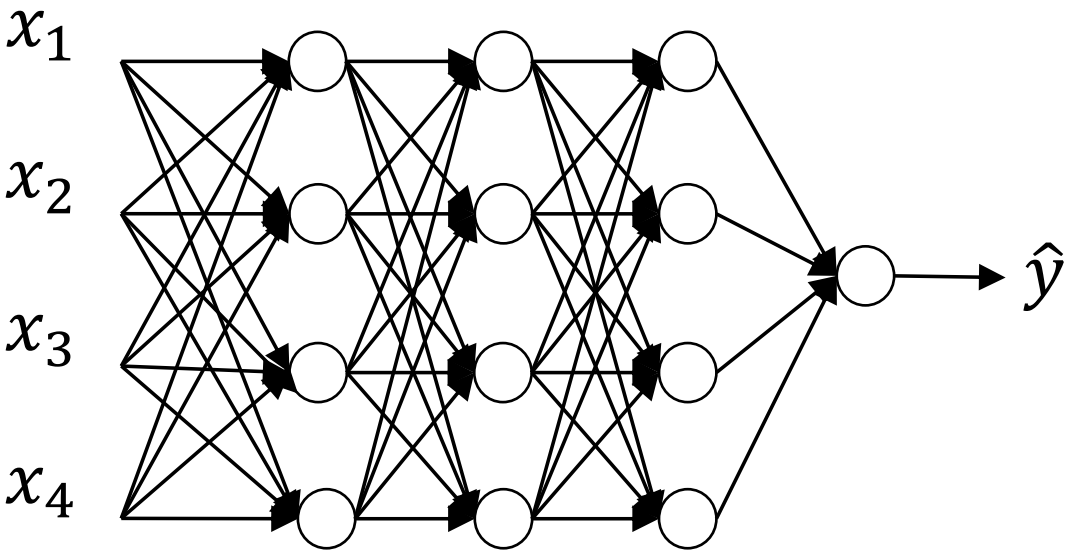
Andrew Ng

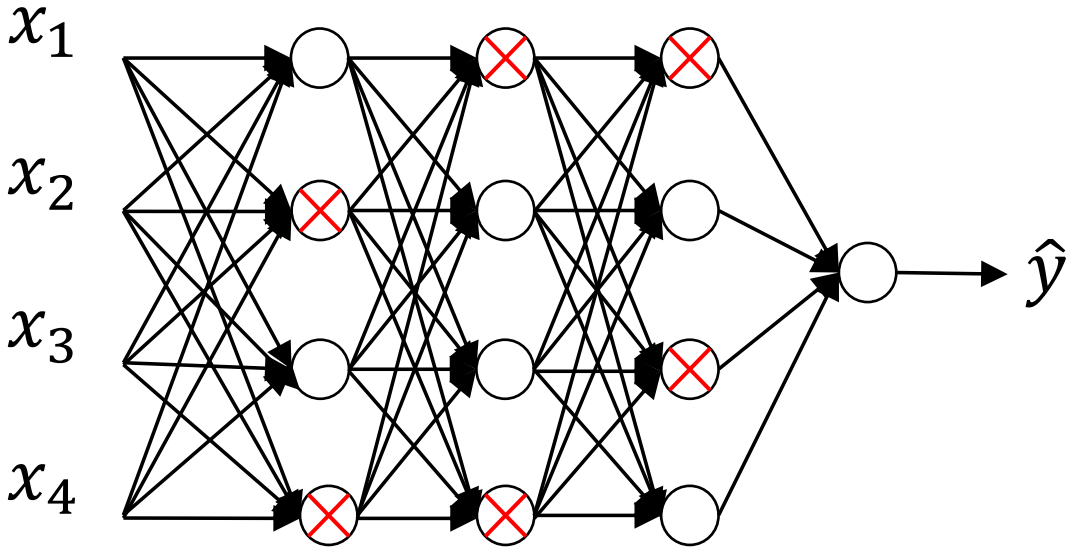
deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization





- For each layer, define the probability that any node of the layer could be deactivated at training time (for one minibatch).

Keep-prob = 0.5 0.5 0.5

- At training time for each sample/minibatch randomly select nodes to deactivate, conduct model update, reactivate the nodes and continue with the next minibatch.

Andrew Ng

MORE USED IN PRACTICE THAN STANDARD DROPOUT

Implementing dropout ("Inverted dropout")

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

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

(d3: zeroing mask)

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

$\# \; a3 \mathrel{*}= d3$.

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

CHARACTERISTIC OF THE INVERTED DROPOUT: during training, the activations of a layer are scaled by the inverse of the keep_prob so to keep the expected value (average magnitude) of the layer's activations unchanged.

50 units. ⇝ 10 units shut off

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

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

$/= 0.8$

Test

This technique reduces scaling problems at test time. (If we didn't scale at training time, we should add an extra scaling at test time).

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

→ Dropout is NOT used at test time.

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~~

no scaling is needed at test time if we used inverted dropout.

deeplearning.ai

Regularizing your neural network

Understanding dropout

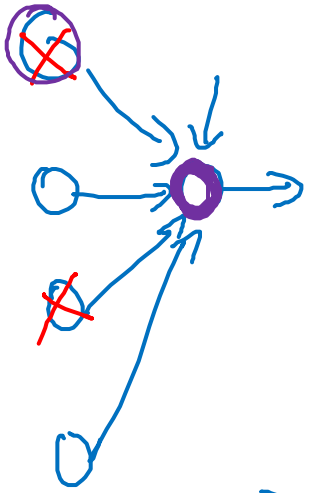
Why does drop-out work?

1 → "DROPOUT SPREADS OUT THE WEIGHTS"

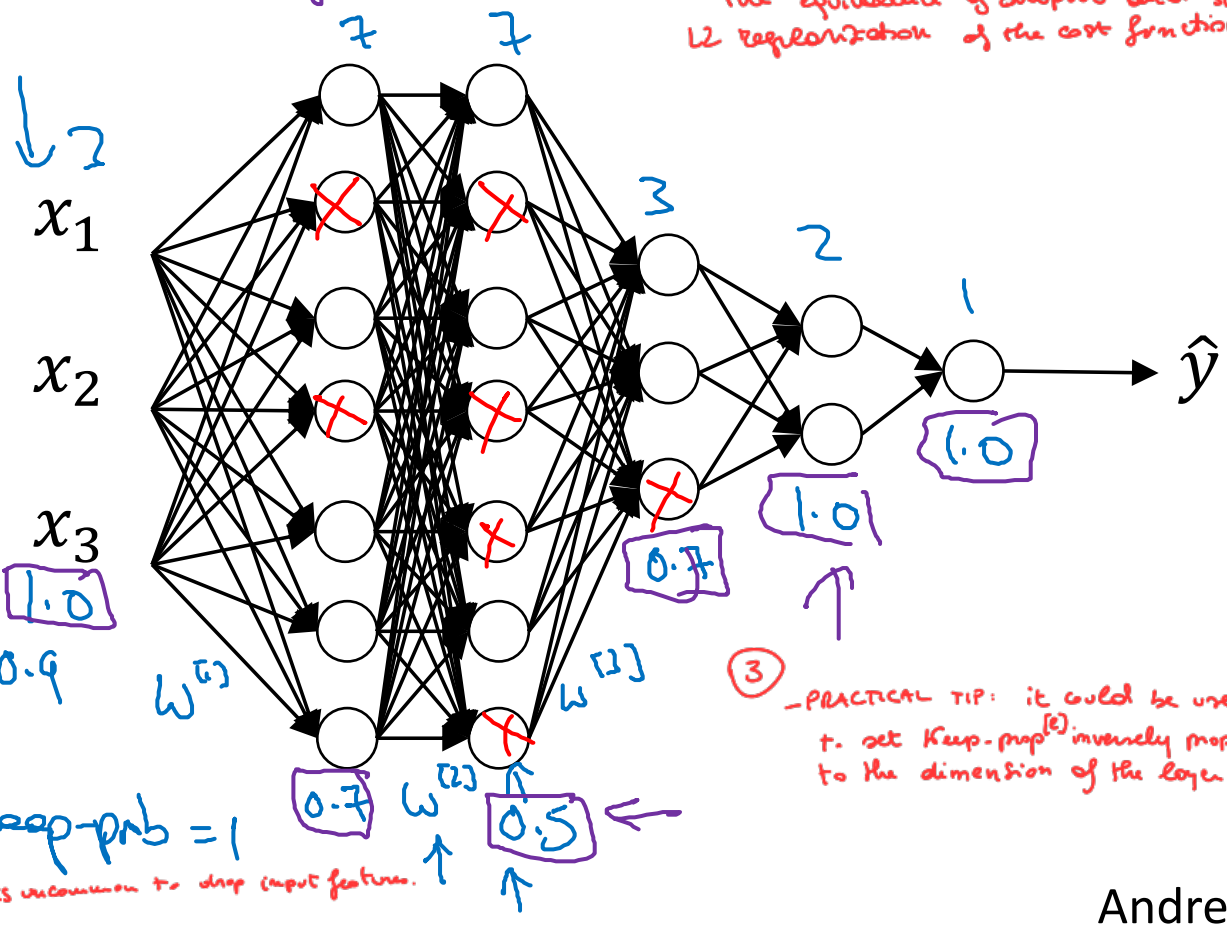
=> intuitively, by spreading the weights it shrinks the weights (it prevents that any weight becomes too big)
=> it has a regularizing effect

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

2 - It is possible to formally demonstrate the equivalence of dropout and specific L2 regularization of the cost function.

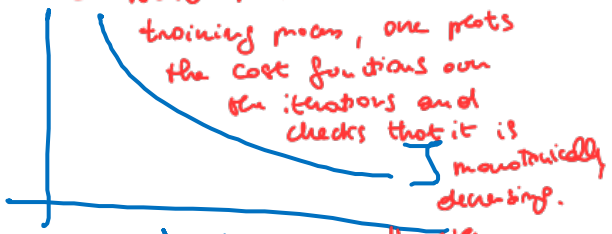


→ Computer Vision



Downside: J

5 NOTE: to monitor the training mean, one plots the cost function over the iterations and checks that it is monotonically decreasing.



However, monotonicity may not be observed if dropout is used. The solution is to periodically deactivate dropout, compute/plot the loss, then reactivate it and continue the training.

4 it is uncommon to drop input features.

3 PRACTICAL TIP: it could be useful to set keep-prop[l] inversely proportional to the dimension of the layer l.

Andrew Ng

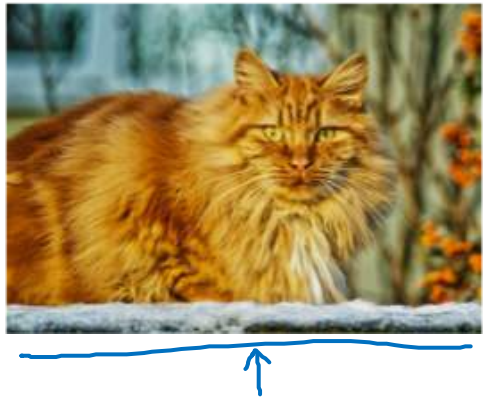
deeplearning.ai

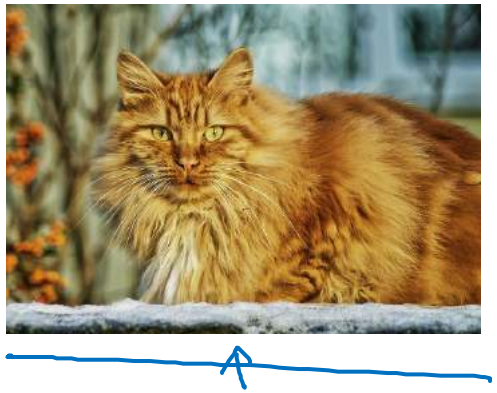
Regularizing your neural network

Other regularization methods

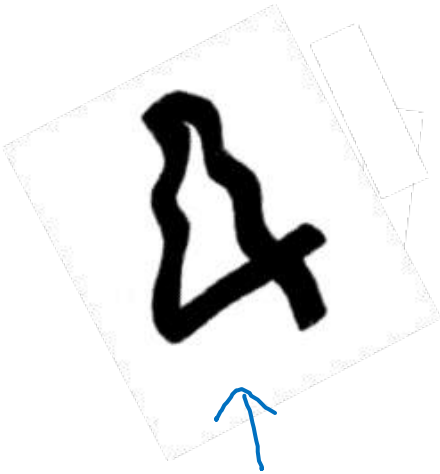
Data augmentation

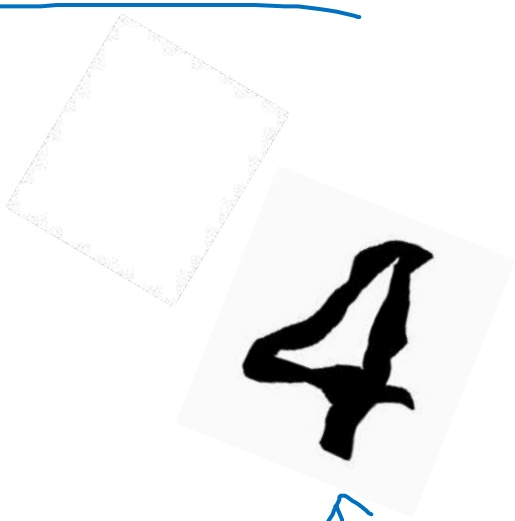
- eg for images: rotations, translations, scaling, flipping, zooming



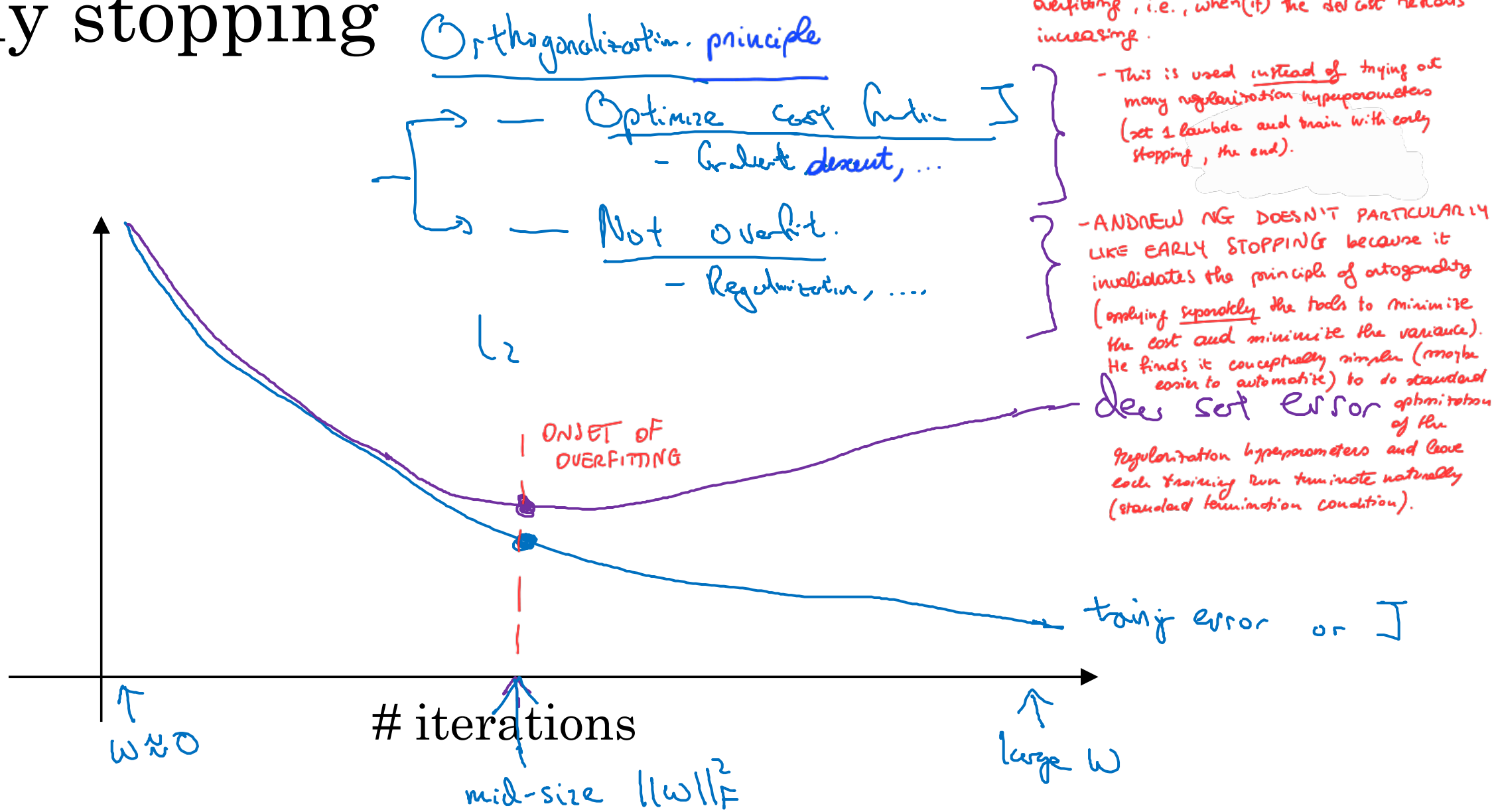


4





Early stopping



- IDEA: stop the NN training at the onset of overfitting, i.e., when(if) the dev cost restarts increasing.

- This is used instead of trying out many regularization hyperparameters (set 1 lambda and train with early stopping, the end).

- ANDREW NG DOESN'T PARTICULARLY LIKE EARLY STOPPING because it invalidates the principle of ortogonality (applying separately the tools to minimize the cost and minimize the variance). He finds it conceptually simpler (maybe easier to automatize) to do standard optimization of the regularization hyperparameters and have each training run terminate naturally (standard termination condition).

deeplearning.ai

Setting up your optimization problem

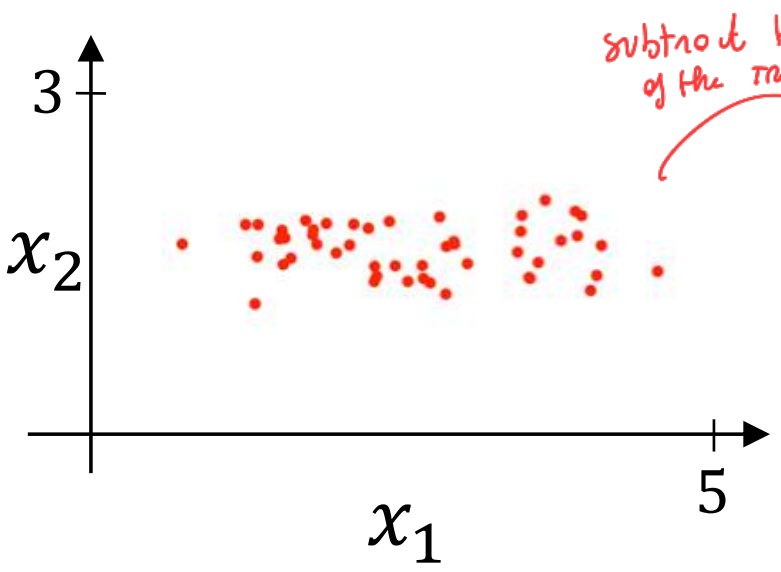
Normalizing inputs

Normalizing training sets

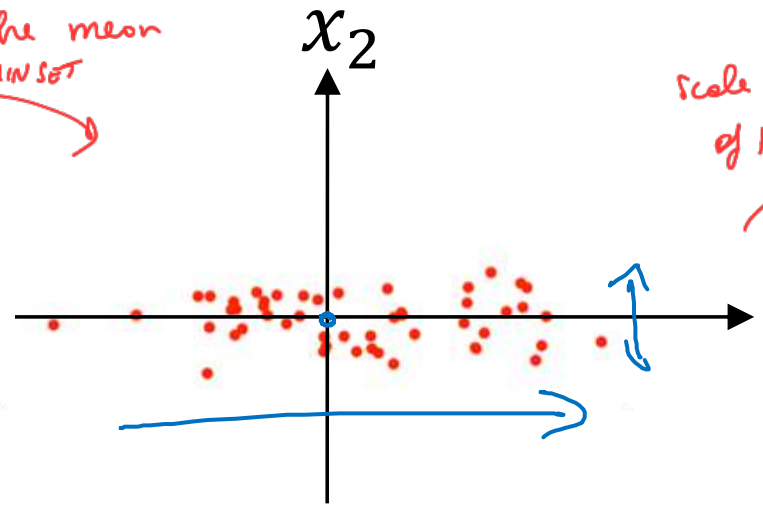
→ STANDARDIZING

- Technique to speed up training
- Scale the training set so it has $\mu = 0$, $\sigma = \sigma^2 = 1$.
- obtained by computing μ_{TRAIN}, σ_{TRAIN} and scaling BOTH training AND test sets by them $\left(\frac{x_i - \mu_{TR}}{\sigma_{TR}}\right)$.
- NOTE! This operation removes the magnitude information from the data and it can be used ONLY when the magnitude of the variables is not an interesting information!

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



subtract the mean of the TRAIN SET



scale by the variance of the TRAIN SET

Subtract mean:

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

$x := x - \mu$

Normalize variance

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

↖ element-wise

$x /= \sigma^2$

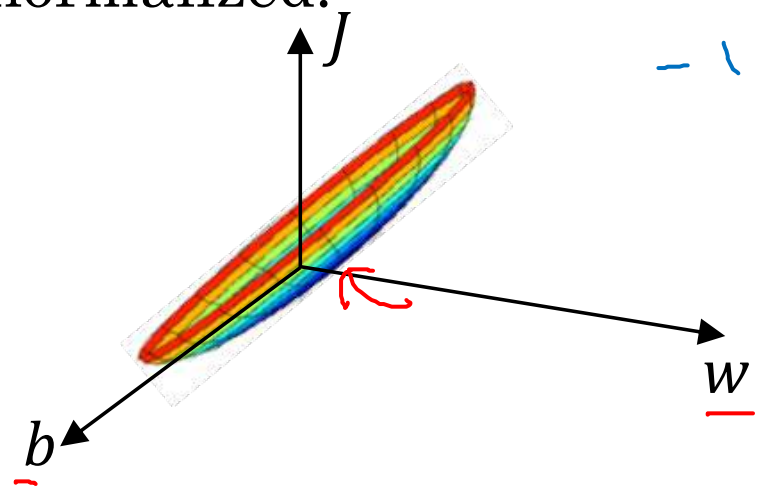
Use same μ, σ^2 to normalize test set.

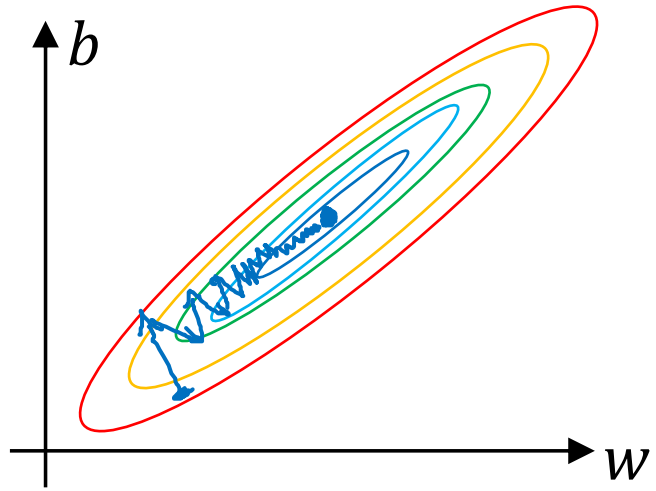
Andrew Ng

Why normalize inputs?

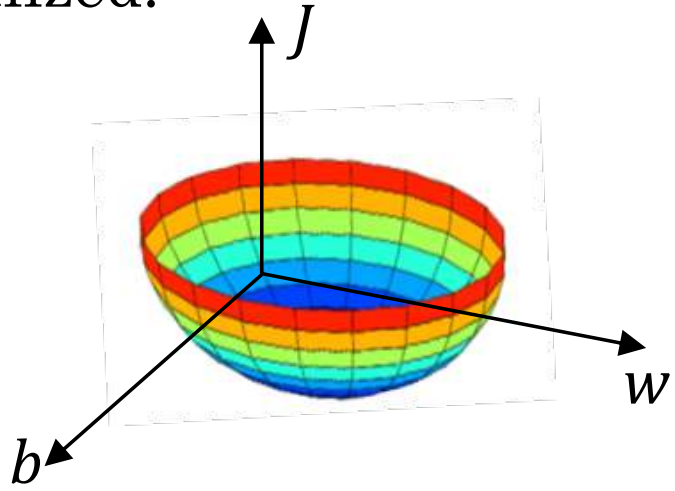
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

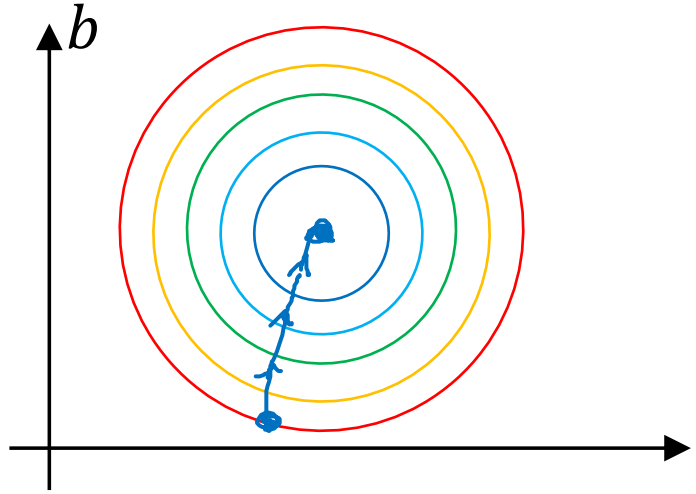
Unnormalized:





Normalized:





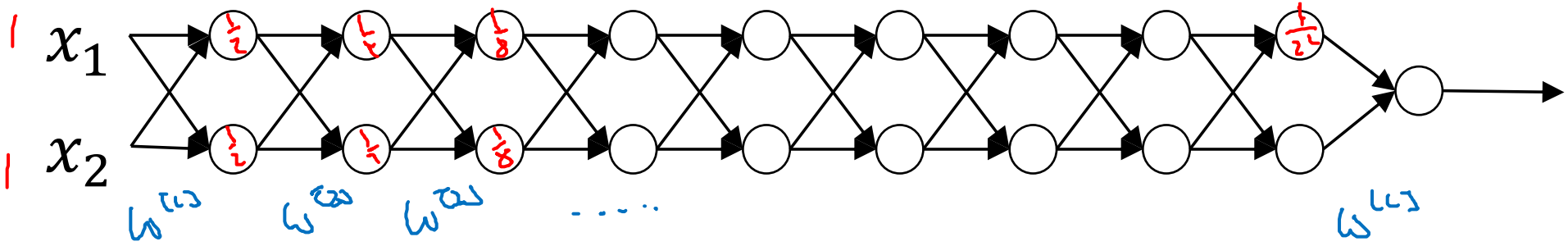
Andrew Ng

deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients



$L = 150$

INTUITION: for very deep networks the activations in later layers (or similarly the gradients in earlier layers) can increase/decrease exponentially if the weights are >1 / <1.

- This is because the activation of one layer is, in practice, multiplied by the activation of the next one.
- This can be prevented with an intelligent initialization of the weights.

1.5

0.5^L

assume linear activation and $b=0$ $g(z) = z$. $b^{[l]} = 0$.

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

$w^{[l]} > I$

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

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

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

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

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

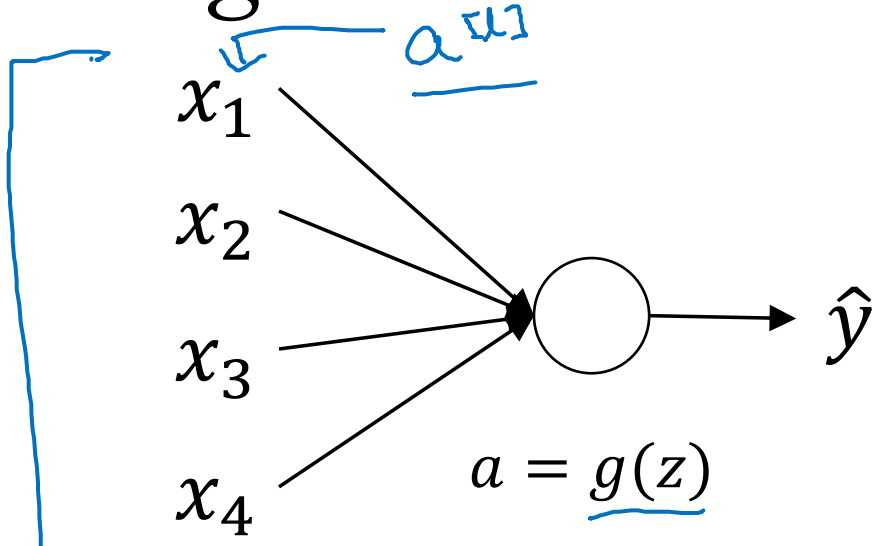
$\hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x$ $\to$ $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1}$

$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example



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

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

How to initialize the weights

$w^{[l]}$ "HE INITIALIZATION"

- commonly used for neurons with ReLU activations.
- assuming that consecutive layers are densely connected (each neuron on layer l has $l-1$ inputs), it makes sense (to limit the output of that neuron) to set each and entrante's weight to $\frac{1}{n\_inputs^{[l]}} = \frac{1}{n^{[l-1]}}$. In practice one initializes the weights of neurons on layer l to

$w^{[l]} \sim N\left(\emptyset, \frac{2}{\sqrt{n^{[l-1]}}}\right)$

Other variants:

tanh

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

Xavier initialization

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

- In practice, this is not one of the most influential hyperparameters.

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

One can approximate a gradient numerically as

$$f'(\theta) = \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$$

. This approximation has an error of order $O(\varepsilon^2)$, and is more precise than the standard "one sided" derivative, which has an error with $O(\varepsilon)$.

Checking your derivative computation

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

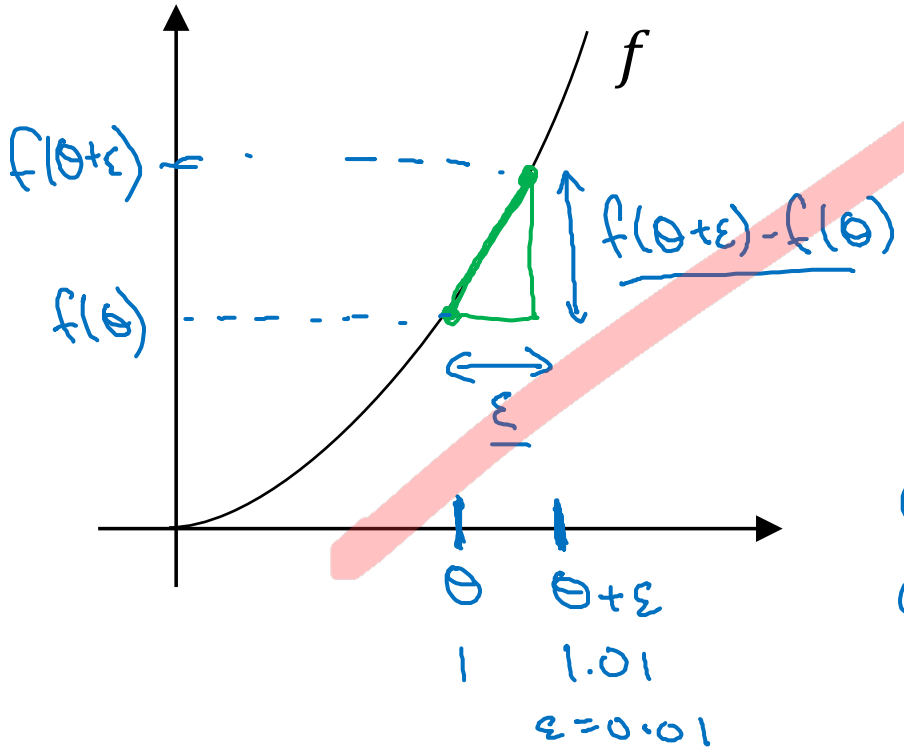
$\theta \in \mathbb{R}$.

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

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

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

$\frac{dJ}{db}$



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

$\theta = 1$

$\theta + \varepsilon = 1.01$

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

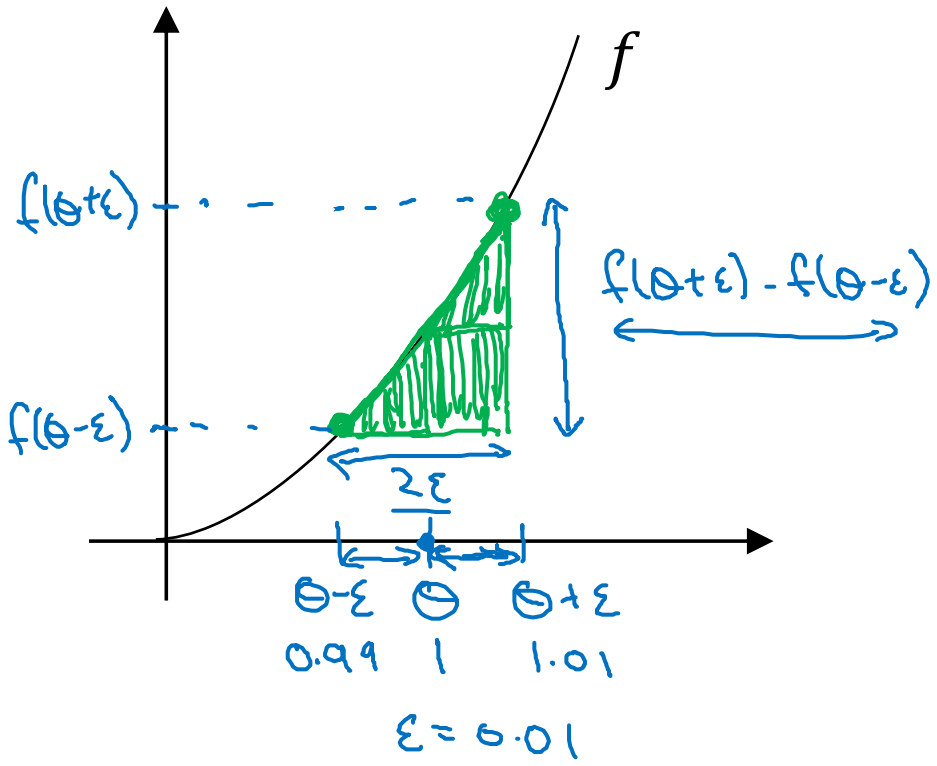
0.0301

3.1

3.2

Checking your derivative computation

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



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

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

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

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 $\dfrac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$

$\varepsilon = 10^{-7}$

$\approx 10^{-7}$ — great! ←

10^{-5}

$\rightarrow 10^{-3}$ — worry. ←

- Compute $d\theta_{approx}[i]$ independently for each $\theta[i]$.
- verify that $d\theta_{approx}$ and $d\theta$ (analytical) are not too different
- In practice: - use $\varepsilon = 10^{-7}$
  - if error $\approx 10^{-7}$ ok
    $\approx 10^{-5}$ maybe ok
    $\approx 10^{-3}$ problem

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

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

$\Leftarrow$ check if $|d\theta_{approx}[i] - d\theta[i]|$

- If algorithm fails grad check, look at which 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.

$W, b \approx 0$

- it's possible that the backprop implementation works well for some values of the parameters and badly for others (e.g. $W, b \approx 0$ ... e.g. W, b >> 0.
- Therefore you could run the gradcheck at the beginning of the training (params are in a certain range), then update for a bit, and then repeat the gradient check (params in different range).

Andrew Ng