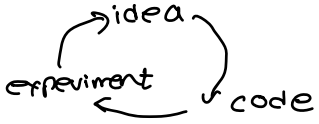
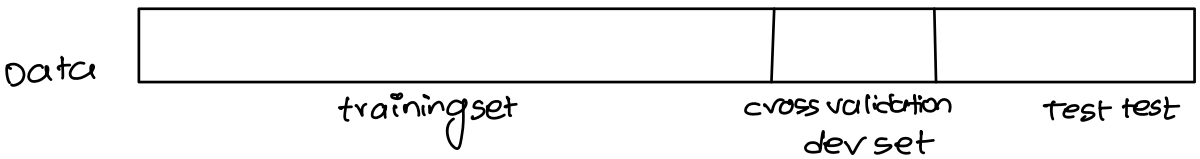
dec1: Train/Dev/Test sets:

ML highly iterative process

- # layers
- # hidden units
- learning rate
- activation function





previous : 70% – train
30% – test

60 – train
20 – dev
20 – test

m = 10,000 to 10,000

Big data : m = 1,000,000

98% train
1% dev
1% test

99.5 % train
0.4 % dev
0.1 % test

Mismatched train/test distribution:-

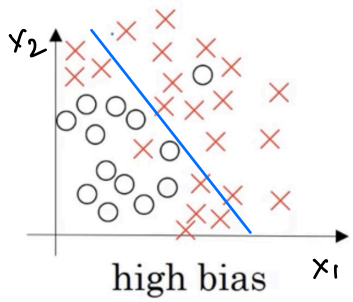
Training set:
cat pictures from webpage
- high res
- good

dev/test sets:
cat pictures from phone app
- low res
- blurry

Make sure dev, test come from same distribution

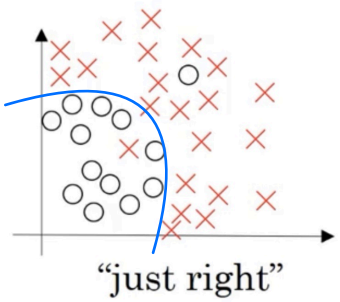
- Not having a test set might be okay. (only train, dev test)
  ↓ test

dec2: Bias and Variance:

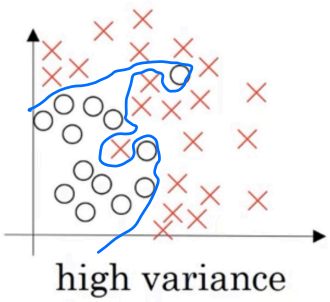


high bias

underfitting



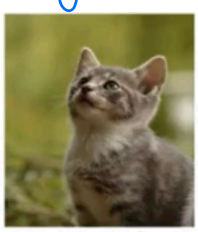
"just right"

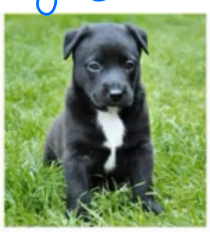


high variance

overfitting

Cat classification



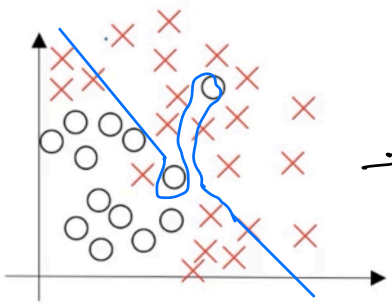


Human error ≈ 0%.
Optimal Bayes error ≈ 0%.

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

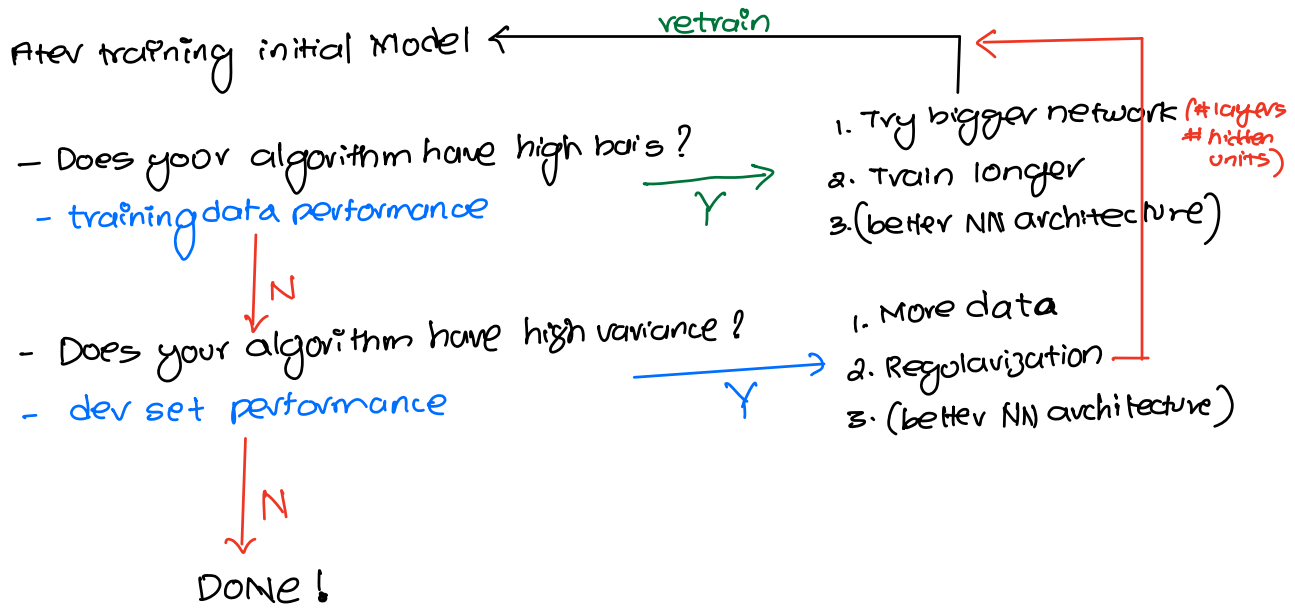
if optimal / Bayes error is 15%, such as blurry images.

then may train 15%, dev 16% - low bais, low variance.



→ high bais, high variance.

dec3. Basic Recipe for Machine learning:



Bais Variance Tradeoff:-

previous era : ↑ bais ↓ variance ↓bais ↑ variance

Big data era : Deeplearning

Move data, ↓ reduces variance

Bigger Network ↓ reduces bais → works inderendetly

Lec4: Regularization:

- Use when model is overfitting or high variance problem.
- use get more training data

Logistic Regression:-

$$\min_{w,b} J(w,b) \qquad w \in \mathbb{R}^{n_x}, b \in \mathbb{R}$$

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

α_2 regularization $= \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

α_1 regularization $= \frac{\lambda}{2m}\|w\|_1 = \frac{\lambda}{2m}\sum_{j=1}^{n_x} |w_j|$

λ = regularization parameters = lambd

Neural Network Reg:-

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

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

- Frobenius Norm.

Before

$dw^{[l]}$ = from backprop

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

After reg

$dw^{[l]}$ = (from backprop) $+ \frac{\lambda}{m} w^{[l]}$

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

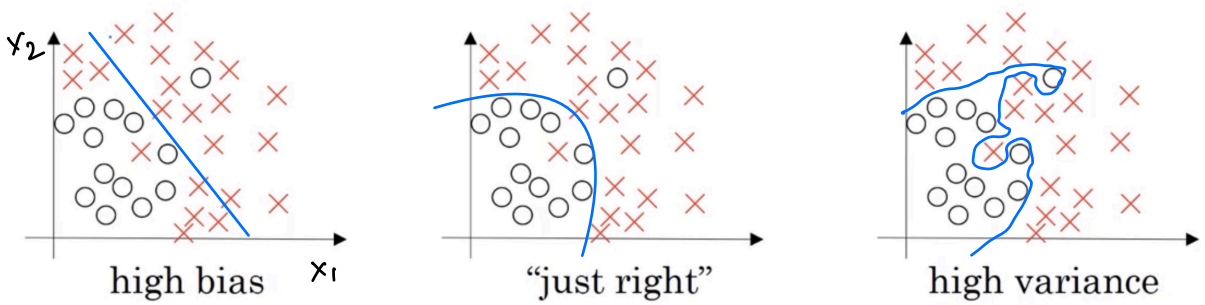
weight decay $w^{[l]} = w^{[l]} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]} \right]$

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

$= \left[1 - \alpha \lambda/m\right] w^{[l]} - \alpha\, dw^{[l]}$

weight decay. < 1

Lec 5: Why does Regularization reduces overfitting:-

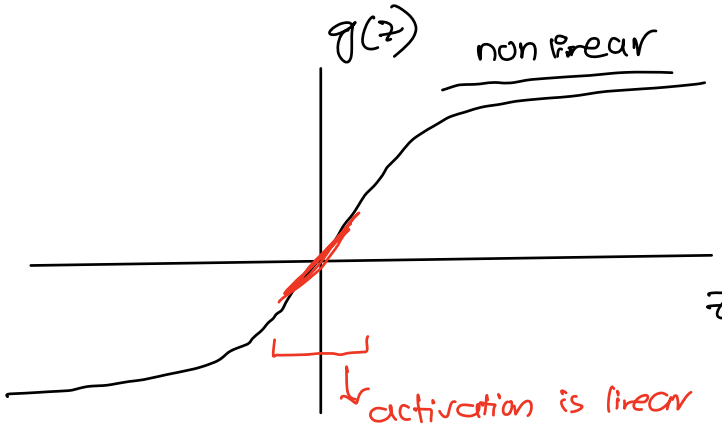


Cost with reg:-

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

if $\lambda = \infty$ then to min J $w^{[l]} \approx 0$

$\Rightarrow$ Neural Network becomes much simpler network.



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

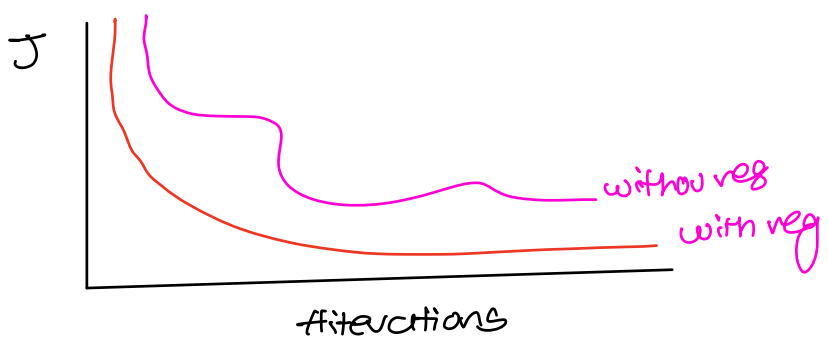
$\lambda \uparrow \quad W^{[l]} \downarrow \qquad z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \Rightarrow z^{[l]} \downarrow$

$\downarrow g(z^{[l]})$ ⇒ roughly linear regression.

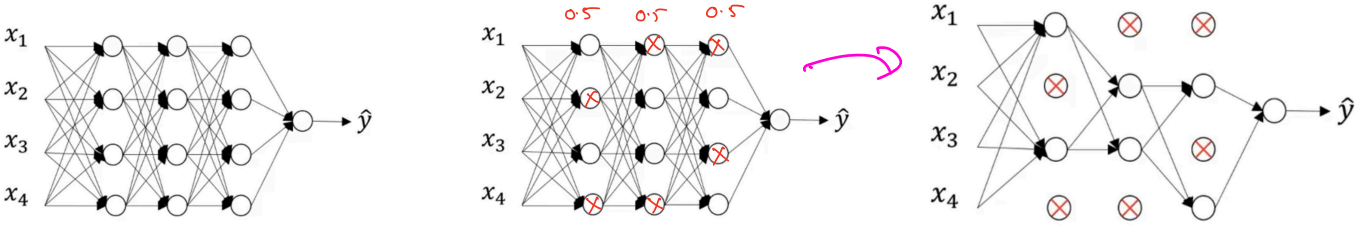
⇒ deep network ⇒ linear regression

⇒ model became simpler.

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



dec 6: Dropout Regularization!



NN-1 NN-2 smaller Network

if NN-1 overfits your data, then we can change following

- do through each of layers in NN-2
- set some probability of eliminating a Node in NN-2

⇒ smaller Network will generalize data better, and reduce the overfitting.

Implementing dropout ("Inverted dropout") = /keep drop

illustrate with layer $l=3$

keepprob = 0.8

d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob

a3 = np.multiply(a3*d3) (elementwise)

a3 /= keep-prob → makes test time easier (less scaling problem).

if a3.shape = (50, m) = $[n^{[3]}, m]$

with dropout: keep prob = 0.8

⇒ on avg layers has 10 units shut off = 50*0.2

$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$

in order not to reduce the Expected value of $z^{[4]}$, we $a^{[3]}$ /= keep prob

Predictions at test time

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

No dropout at test time ⇒ No random predictions.

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

$$a^{[1]} = g^{[0]}(z^{[1]})$$

$$\vdots$$

$$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$$

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

Lec7: Why does dropout work:-

Intuition: can't rely on any one feature, so have to spread out weights

dropout randomly knock out hidden units

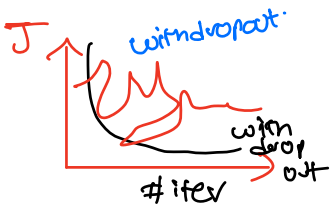
↓

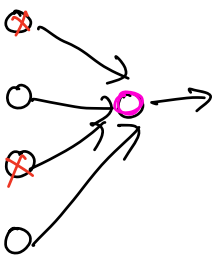
every iteration working with smaller network

↓

have regularizing effect

} previous intution.





- can't rely on any one feature
  - ⇒ hence spreds weights outs
  - ⇒ this will shrink weights → L_2 regularization.
  - → prevents overfitting

dropout formaly can be shown to be adaptive form L_2 regularization.

$w^{[1]} = (7,3)$ 0.7 $\quad w^{[2]} = (7,7)$ keepprob = 0.5, $\quad w^{[3]} = (3,7)$ 0.7, $\quad w^{[4]} = (2,3)$ 1.0 $\quad w^{[5]} = (1,2)$ 1.0

→ less with more parameters

Lec8: Other Regularization:-

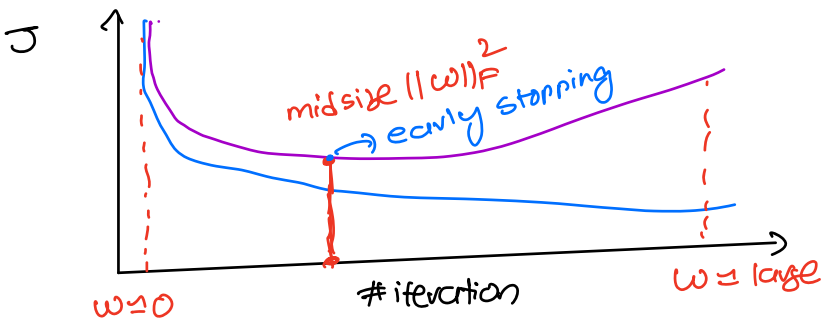
(i) Data Augmentation:-

(i) flip horizontaly images (double training set)

(ii) random crops

(iii) random rotation, distortion

(ii) Early stopping:- (overfitting)



problem? (orthogonalization is not possible)

→ mixed

① optimize cost function J
- gradient descent
- momentum
} min J(w,b) w,b

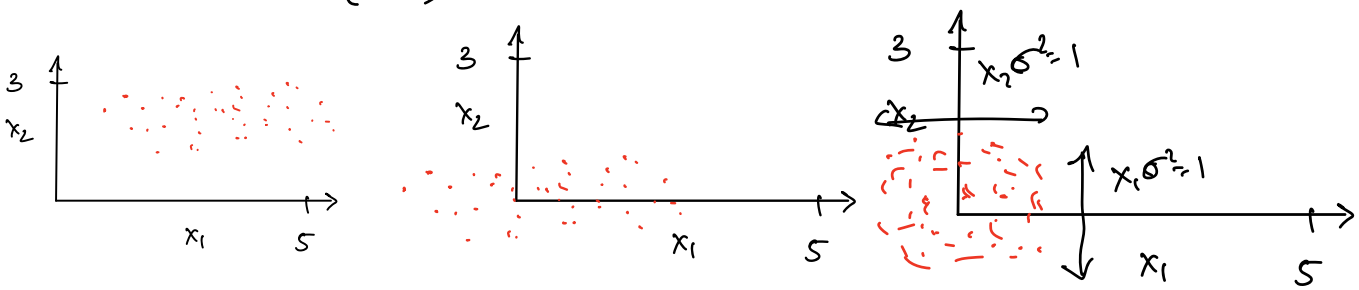
② Not overfit
- regularization
- more data
} reduce variance

separate task. orthogonalization.

Setting up your optimization Problem:

decq: Normalizing inputs:-

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



① subtract mean

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

$$x := x - \mu$$

② Normalize variance

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

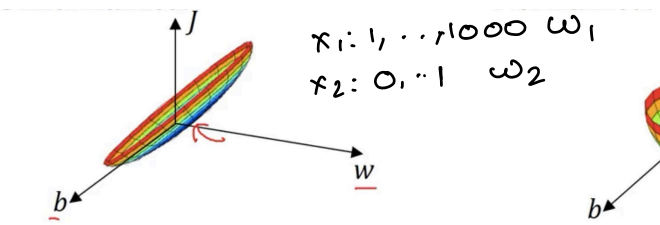
$$x \mathrel{/}= \sigma$$

use same μ, σ for test set same as training set

Why Normalize inputs:-

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

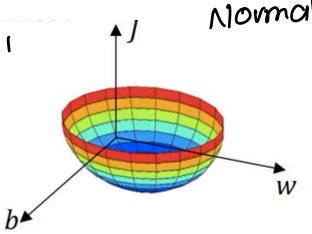
unnormalized

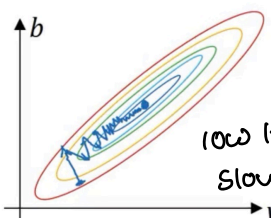


x_1: 1, . . ,1000 w_1

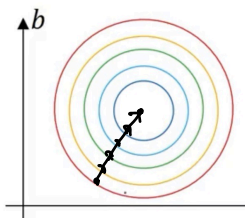
x_2: 0, . . 1 w_2

Normalized.



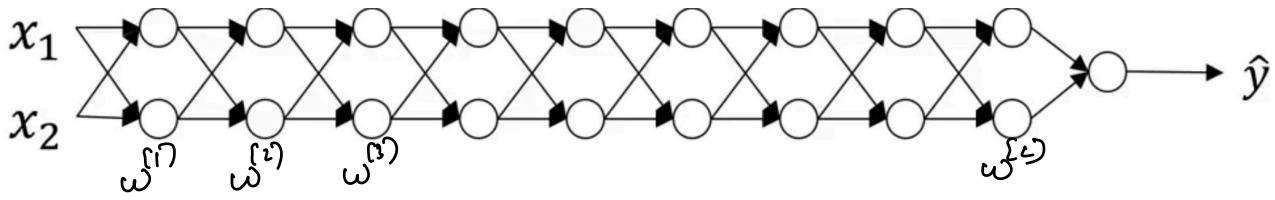


low learning rate
slow training



high learning rate
faster training

dec10: Vanishing / Exploding gradients: (DeepNN problem)



$g(z) = z$ (linear activation)

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

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

$\hat{y} = w^{[L]} w^{[L-1]} \dots w^{[3]} w^{[2]} w^{[1]} x$

$a^{[1]}$

$a^{[2]}$

$a^{[3]}$

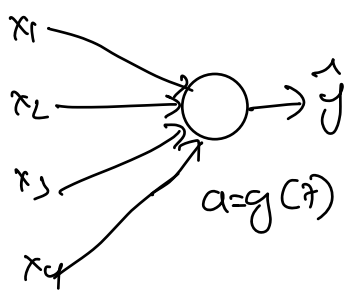
$a^{[L]}$

$L-1$, $l=1$ $\quad w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ $\quad \hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x \simeq$ Very large

L, $l=1$ $\quad w^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ $\quad \hat{y} = w^{[L]} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1} x \simeq$ Very low

→ same logic $dw^{[l]}$ applys

Lec11: Weight Initialization for Deep Networks:-



$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$ ①

large n → smaller w_i

$var(w) = \frac{1}{n}$ if $g(z)$ - relu

$var(w) = 2/n$

$$w^{[l]} = np.random.randn(shap) * np.sqrt\left(\frac{1}{n^{[l-1]}}\right)$$

if $g(z)$ = relu

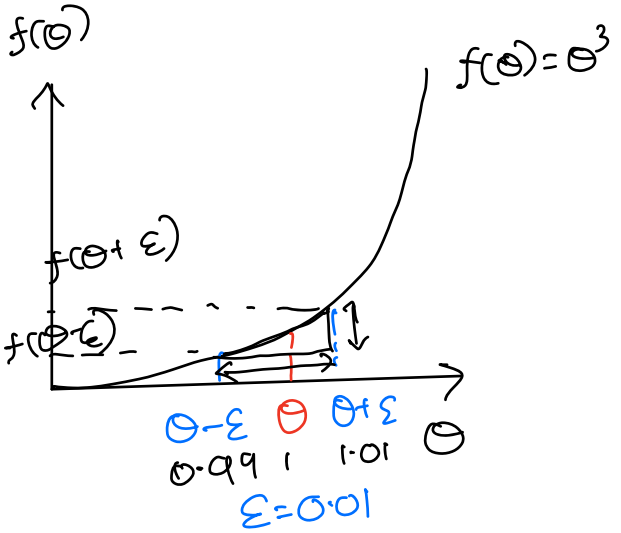
$$w^{[l]} = np.random.rand(shap) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$$

if $g(z)$ = tanh

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

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

dec12: Numerical approximation of gradients:



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

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

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

approx error = 0.0001

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

dec13: Gradient checking for NN:-

take $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \dots w^{[L]}, b^{[L]}$ and convert in to-

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

big vector $d\theta$

is $d\theta$ gradent of $J(\theta)$?

for each i:

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

$$\simeq d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

$d\theta_{approx} \simeq d\theta$? $\varepsilon = 10^{-7}$

check $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} \simeq 10^{-7}$ - great!

if 10^{-5} - check

if 10^{-3} = bug

dec 14: Gradient checking implementation:-

1. don't use in training - only to debug
2. if algorithm fails grad check, look at components to try to identify bug
3. remember regularization

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

4. doesn't work with dropout. (keep_prob = 1.0) then check.
5. run at random initialization; perhaps again after some training