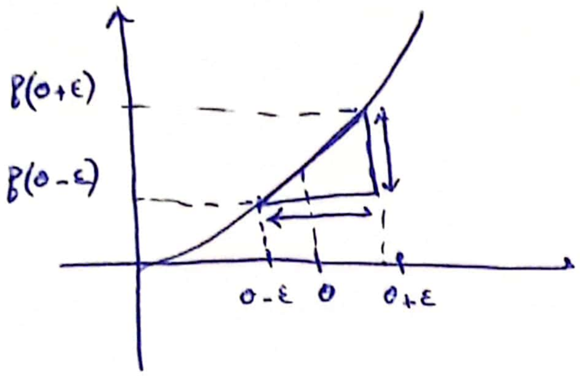
numerical approximation of gradients

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



$\varepsilon = 0.01$

approximation of gradient $\simeq \dfrac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$

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

the error of this approximation is $O(\varepsilon^2)$

$\dfrac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} \longrightarrow$ error is $O(\varepsilon)$

gradient checking of a NN

① take $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \dots$

and reshape into a big vector θ

$W^{[1]}, W^{[2]}, \dots, W^{[L]}$: reshape them into vectors

concatenate them with $b^{[1]}, b^{[2]}, \dots, b^{[L]}$

② take $dW^{[1]}, db^{[1]}, \dots$

reshape them into a big vector $d\theta$

③ $J(\theta) = J(\theta_1, \theta_2, \theta_3, \dots \theta_i \dots)$; eps $= 10^{-7}$

for each i

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

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

---

practical tips

1/ don't use gradient checking in training
use it only to debug

2/ if algorithm fails grad check, look at components
to identify the bug

3/ remember to add the derivative of the regularization
term

4/ doesn't work with dropout

~~normalizing tra~~

normalizing training sets

mean = $\mu = \frac{1}{m} \sum_{i=1}^{m} x^{[i]}$

variance = $\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{[i]} ** 2$

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

NB: use same μ and σ to normalize the test set

why normalization ?

we will need less # iterations to converge to the minimum

$\Rightarrow$ it will easier and faster to train

⚠ if the feature are in the same scale, no need to ~~do~~ this step

Vanishing gradients / exploding gradients

suppose that: $g^{[l]}(z^{[l]}) = z^{[l]}$

$b^{[l]} = 0$ for all l

$\Rightarrow \hat{y} = w^{[L]} \cdots\cdots w^{[1]} X$

if we suppose $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$ for l in $1 \dots L-1$

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

$\simeq 1.5^{L-1} X$

if we have a very deep NN :

$\hat{y}$ will explode

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

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

$\simeq 0.5^{L-1} X$

if we have a very deep NN

activations will decrease exponentially

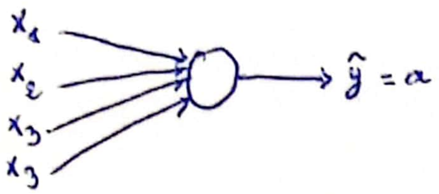
Conclusion

if weights > 1 : we will increase exponentially

if weights < 1 : we will decrease exponentially solution for vanishing/exploding

better choice for parameter initialization

1/ single neuron



suppose $b=0$

$$z = \sum_{i=1}^{n} w_i x_i \ .$$

we want z not too large not too small

if n is large : we want small w_i

$\Rightarrow$ set $Var(w_i) = \frac{1}{n}$ or $\frac{2}{n}$ if we are using a relu activation

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

relu activation : $var(w_i) = \sqrt{\frac{2}{n^{[l-1]}}}$: He initialization

or

$$var(w_i) = \sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$$

this variance can be another hyperparameter to fine tune

① Random initialization is used to break symmetry and make sure different hidden ~~layers~~ units can learn different things

② He initialization works well with ReLU activations