Week 1c - Setting up your optimization problem

Friday, August 14, 2020 11:25 PM

① Normalizing inputs

↳ $x := \frac{x - \mu}{\sigma}$

i. Subtract mean to set mean to zero
& center data

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

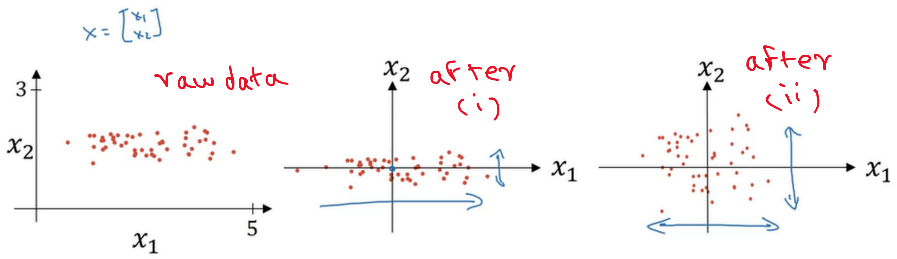
$$x := x - \mu$$

ii. Normalize variance

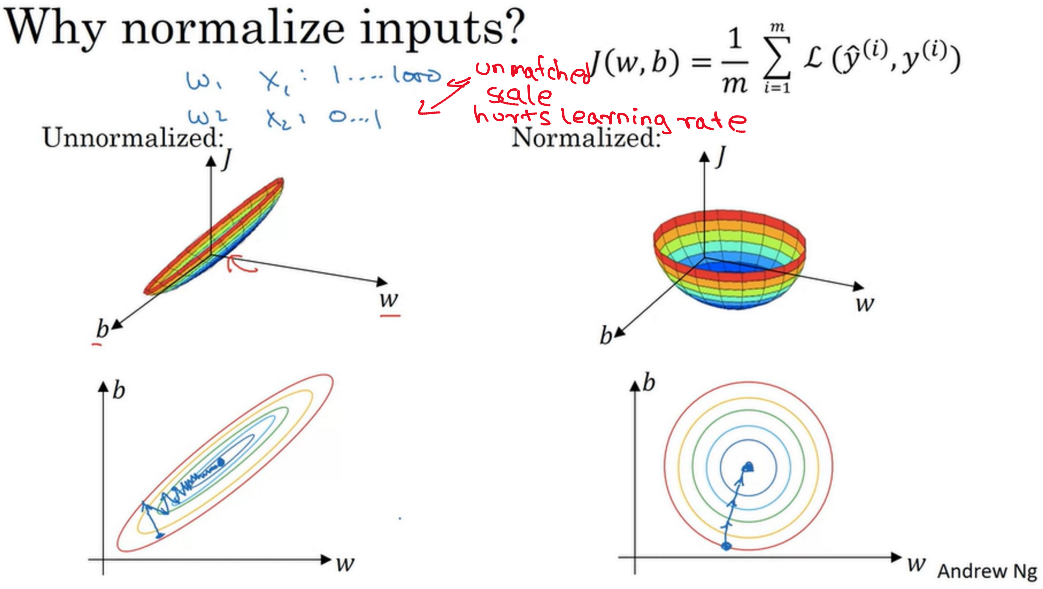
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} **2$$

$$x /= \sigma$$

iii. use same μ & σ to normalize test set too



→ why normalize inputs



② Vanishing / Exploding gradients

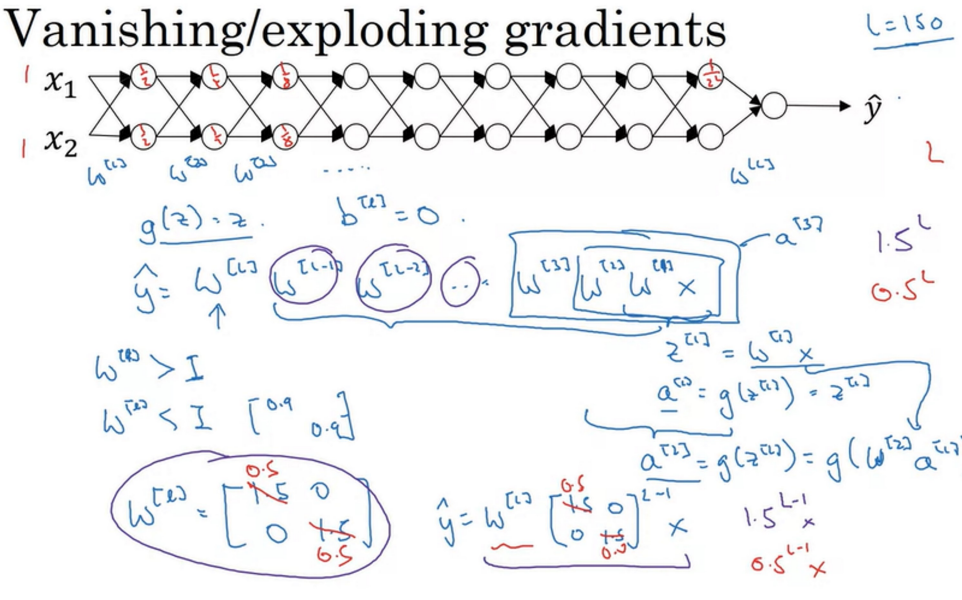
→ When training deep NNs, gradients might become too small or very large

→ if $W^{[l]} > I$ (Identity matrix),

the final value $\hat{y}$ is in the proportion of $W^{(L-1)}$ [exponentially high]

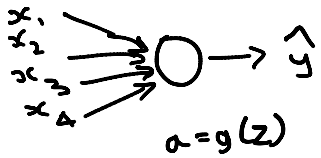
if $W^{[l]} < I$

the final value $\hat{y}$ is very low



Andrew Ng

③ Weight Initialization for Deep Neural Networks



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

large n → small w_i

→ we could set $\text{var}(w_i) = 1/n$ — $\frac{2}{n}$ works better for ReLU

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

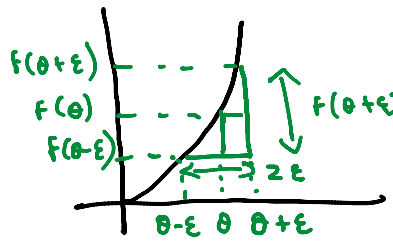
→ other variants

- tanh → $\sqrt{\frac{1}{n^{[l-1]}}}$ works better (Xavier initialization)
- $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$ is also used

④ Numerical approximation of gradients

→ Checking your derivative computation

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



2 sided approximation

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

$3.0001 = \frac{(1.01)^3-(0.099)^3}{2(0.01)}$

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

$\therefore$ error $= 0.0001$

1-sided approx

$\Rightarrow g(\theta) \approx \frac{f(\theta+\varepsilon)-f(\theta)}{\varepsilon}$ → order of error is ε

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

- 2 sided approximation runs slower but is worth the improvement in accuracy

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

↓

order of error is in order of ε^2

⑤ Gradient Checking

i. Take $w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}$ and reshape it into θ

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

ii. Take $dw^{[1]}, db^{[1]}, \ldots, dw^{[L]}, db^{[L]}$ and reshape it into $d\theta$

Is $d\theta$ gradient of θ?

→ Gradient checking (grad check)

- for each i: $J(\theta) = J(\theta_1, \theta_2 \ldots)$

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

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

- check if $d\theta_{approx} \approx d\theta$

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

$\varepsilon = 10^{-7}$ is generally used

↳ if o/p is 10^{-7} → great
10^{-5} → double check
10^{-3} → worry

⑥ Gradient checking Implementation Notes

- → Use grad check only in testing / debug as it is very slow to be used in training
- → If algorithm fails grad check, look at components to try
- → Remember regularization
- → Doesn't work with dropout
  Use keep-prob = 1.0 then do grad-check & then set keep-prob to desired value
- → Run at random initialization ; perhaps again after some training