Lecture 12 Backprop & Improving Neural Networks

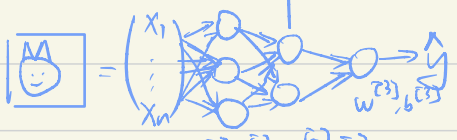
1. Backpropagation

cost function: $J(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} L^{(i)}(\hat{y}, y)$ → for vectorization

with $L^{(i)} = -[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$

update: $w^{[l]} = w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}}$

The cat example:



- $\frac{\partial J}{\partial w^{[3]}}$ ✗, instead: $\frac{\partial L}{\partial w^{[3]}} = -[y^{(i)} \frac{\partial L}{\partial w^{[3]}} \log \sigma(w^{[3]} a^{[2]} + b^{[3]}) + (1-y^{(i)}) \frac{\partial L}{\partial w^{[3]}} \log[1-\sigma(\cdot)]]$

$$= -y^{(i)} \frac{1}{a^{[3]}} a^{[3]}(1-a^{[3]}) a^{[2]T} + (1-y^{(i)}) \frac{1}{1-a^{[3]}} a^{[3]}(1-a^{[3]}) a^{[2]T}$$

$$= -y^{(i)}(1-a^{[3]}) a^{[2]T} + (1-y^{(i)}) a^{[3]} a^{[2]T}$$

$$= -y^{(i)} a^{[2]T} + y^{(i)} a^{[3]} a^{[2]T} + a^{[3]} a^{[2]T} - y^{(i)} a^{[3]} a^{[2]T}$$

$$= \boxed{-(y^{(i)} - a^{[3]}) a^{[2]T}}$$

$\because \frac{\partial}{\partial w^{[3]}}(w^{[3]} a^{[2]} + b^{[3]})$ $\{(1,2)$

$(1,2)$ $(1,1)$ $a^{[2]T}$ $(2,1)$

$$\Rightarrow \frac{\partial J}{\partial w^{[2]}} = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - a^{[2]})\,a^{[2]T}$$

- $$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

$\therefore \frac{\partial \mathcal{L}}{\partial w^{[2]}} = (a^{[2]} - y)\cdot a^{[1]T} \Leftarrow a^{[2]} - y$; $w^{[2]}$; $a^{[1]}\cdot(1-a^{[1]})$; $a^{[0]T}$

$a^{[1]T} = \frac{\partial z^{[2]}}{\partial w^{[2]}}$

$\therefore \frac{\partial \mathcal{L}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial w^{[1]}} \div \frac{\partial z^{[1]}}{\partial w^{[1]}}$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w^{[1]}} = (a^{[2]} - y)\cdot w^{[2]} \cdot a^{[1]}(1-a^{[1]})\cdot a^{[0]T}$$

(2,3) ; (1,1) ; (2,1) ; (2,1) ; (1,3)

element wise product:

when same matrix size,

$\begin{bmatrix}1\\2\end{bmatrix} * \begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}1\\4\end{bmatrix}$

$$= w^{[2]T} \boxed{*} a^{[1]}(1-a^{[1]}) \cdot (a^{[2]} - y)\,a^{[0]T}$$

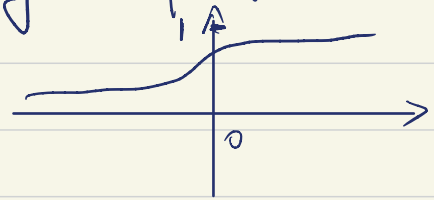
(2,1) ; (1,3) ; (2,3)

2. Improve NNs

1) Activation functions

① sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$

$\sigma'(z) = \sigma(z) \cdot (1-\sigma(z))$

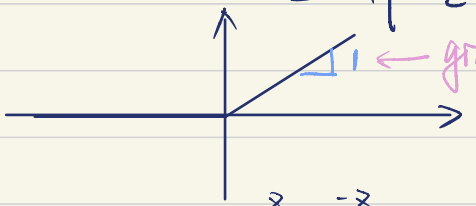


② ★ $\mathrm{ReLU}(z) = \begin{cases} 0 & \text{if } z \le 0 \\ z & \text{if } z > 0 \end{cases}$

Most used.

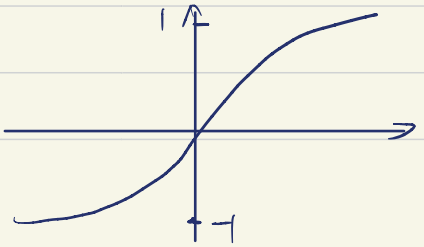
The other two has low gradients when z is $\ll 0$ or $\gg 0$

$\mathrm{ReLU}'(z) = \mathbb{I}\{z > 0\}$

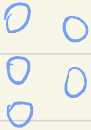


③ $\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

$\tanh'(z) = 1 - \tanh(z)^2$



- Why do we need activation function?

example:  activations: Id. $z \longmapsto z$

$$\Rightarrow \hat{y} = a^{[3]} = z^{[3]} = w^{[3]} a^{[2]} + b^{[3]}$$
$$= w^{[3]}(w^{[2]} a^{[1]} + b^{[2]}) + b^{[3]} = w^{[3]} w^{[2]} (w^{[1]} x + b^{[1]}) + w^{[3]} b^{[2]} + b^{[3]}$$
$$= Wx + B$$

with: $W = w^{[3]} w^{[2]} w^{[1]}$

$B = w^{[3]} \cdot w^{[2]} \cdot b^{[1]} + w^{[3]} b^{[2]} + b^{[3]}$

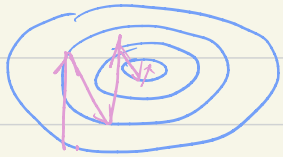
$\Rightarrow$ without activation function $\rightarrow$ linear combination

2) Initialization Method

- Normalizing your input

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})^2 \qquad \Rightarrow x = \frac{x - \mu}{\sigma}$$

Before:



After:

directly towards the middle

* The μ and σ are computed on the training data. And these μ and σ have to be used on the test set as well.

- Vanishing / Exploding gradients

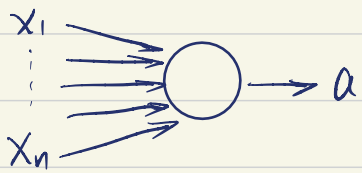
example:

$$\hat{y} = w^{[L]} \cdot a^{[L-1]} = w^{[L]} \cdot w^{[L-1]} \cdot a^{[L-2]} = w^{[L]} \cdot \underbrace{w^{[L-1]} \cdots w^{[1]}}_{(2,2)} \cdot x$$

$$w^{[L]} = \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix} \qquad \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}^L$$

very large

example with one neuron:



$a = g(z)$

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

large n → small w_i ⇒ $w_i \sim \frac{1}{n}$

⇒ for sigmoid: $w^{(l)} = np.random.randn(shape) * np.sqrt(\frac{1}{n^{(l-1)}})$

for ReLU: $w^{(l)} = np.random.randn(shape) * np.sqrt(\frac{2}{n^{(l-1)}})$

for tanh: $w^{(l)} \sim \sqrt{\frac{1}{n^{(l-1)}}}$ (Xavier Initialization)

He Initialization: $w^{(l)} \sim \sqrt{\frac{2}{n^{(l)} + n^{(l-1)}}}$ (often used)

3) Optimization

$X = (x^{(1)}, x^{(2)}, \cdots x^{(m)})$

$Y = (y^{(1)}\ y^{(2)} \cdots y^{(m)})$

$X = (X^{\{1\}}\ X^{\{2\}} \cdots X^{\{T\}})$ $\quad Y = (Y^{\{1\}} \cdots Y^{\{m\}})$

$X^{\{1\}} \cdots X^{(1000)}$

Algorithm for mini-batch gradient descent:

For iteration $t = 1 - \cdots$:

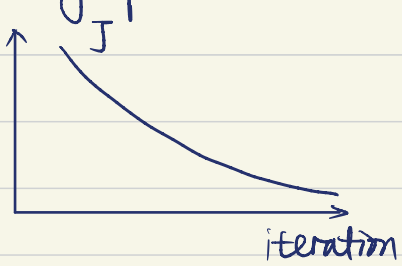
Select batch $(X^{\{t\}}, Y^{\{t\}})$

Do Forward propagation $\Rightarrow J = \frac{1}{1000} \sum_{i=1}^{1000} L^{(i)}$

Backprop Batch

Update $w^{(l)}, b^{(l)}$

Show in graph:



- Gradient Descent + Momentum algorithm.

weight → velocity : tracks the direction that we should take regarding the current update and also the past update

$$\begin{cases} v = \beta v + (1-\beta)\dfrac{\partial L}{\partial w} \\ w = w - \alpha v \end{cases}$$

look at past updates

⇓

the avg. of vertical is smaller (different direction)

the avg. of horizontal is larger (same direction)