DLSG Week 3: Neural Network

Thuong Nguyen Canh

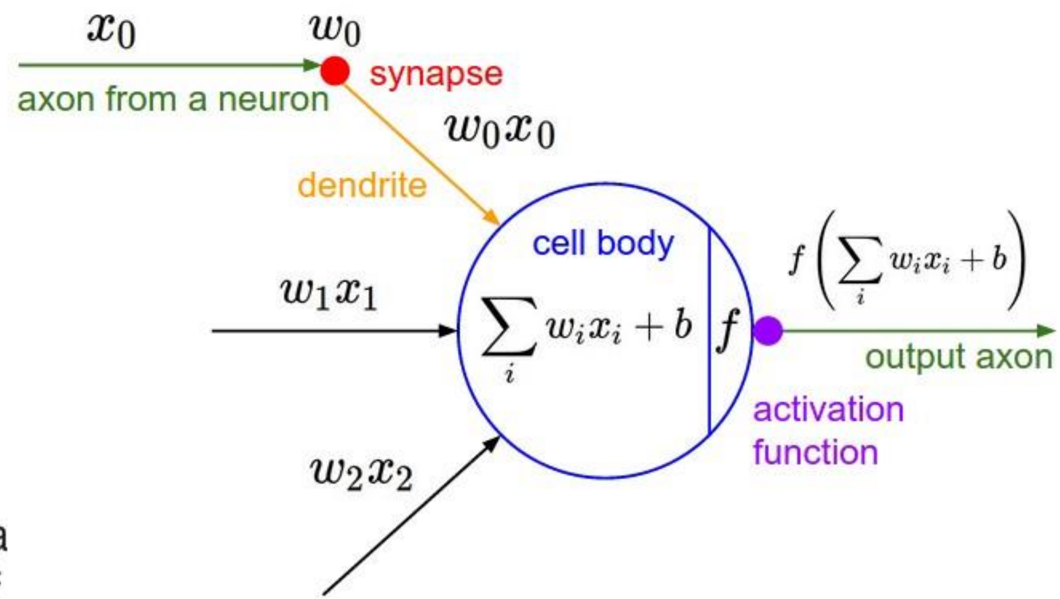
August 2019

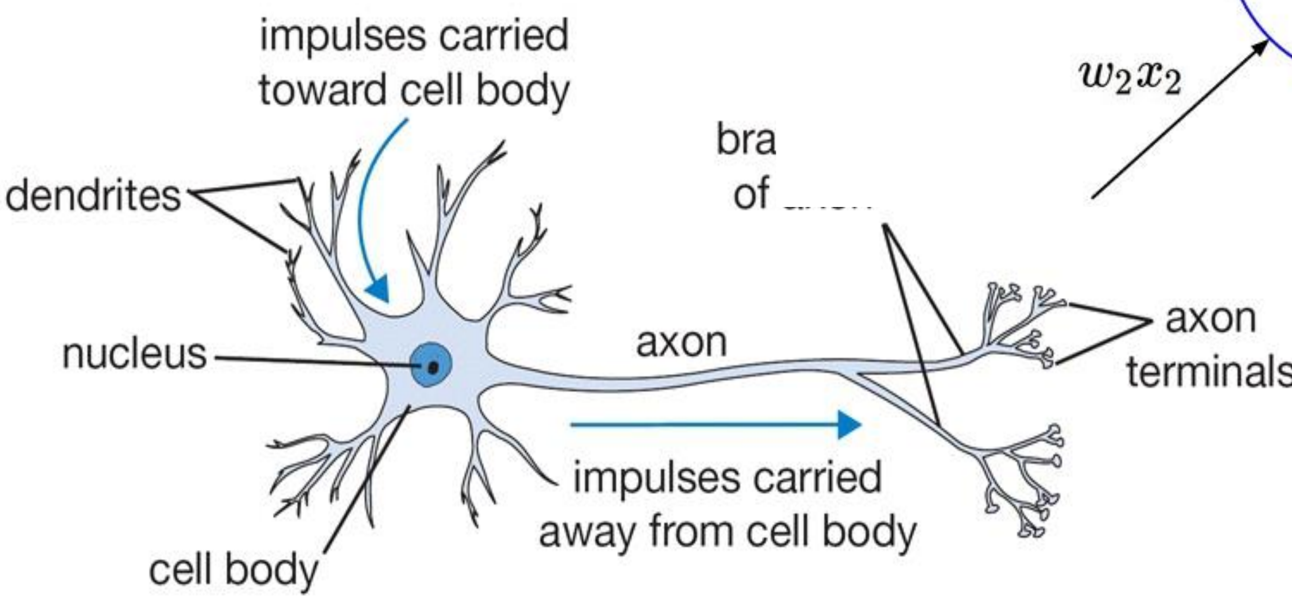
This material is based on Coursera Deep Learning Specialization and Stanford CS231n Course

Neural Network Definition

- Biological motivations

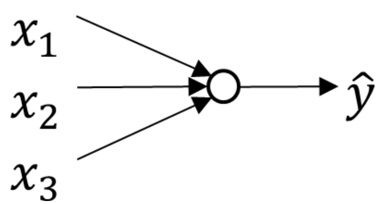
This is a simplified connection.
Neural network does not represent how human brain works

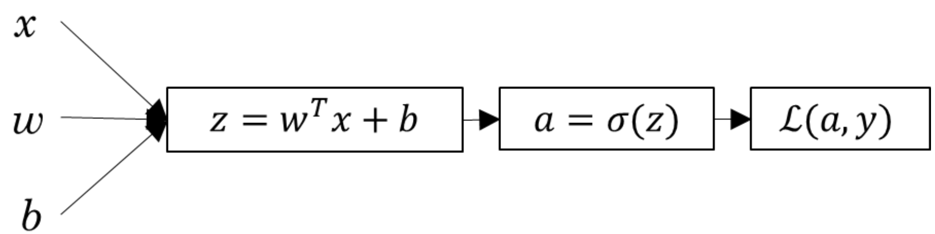




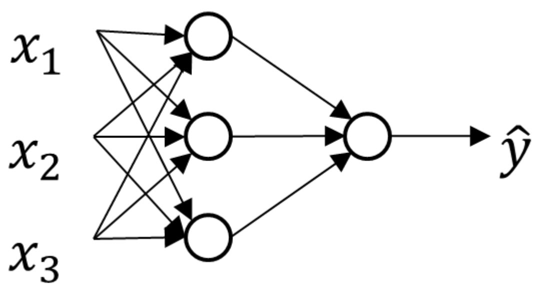
Neural Network Definition (3)

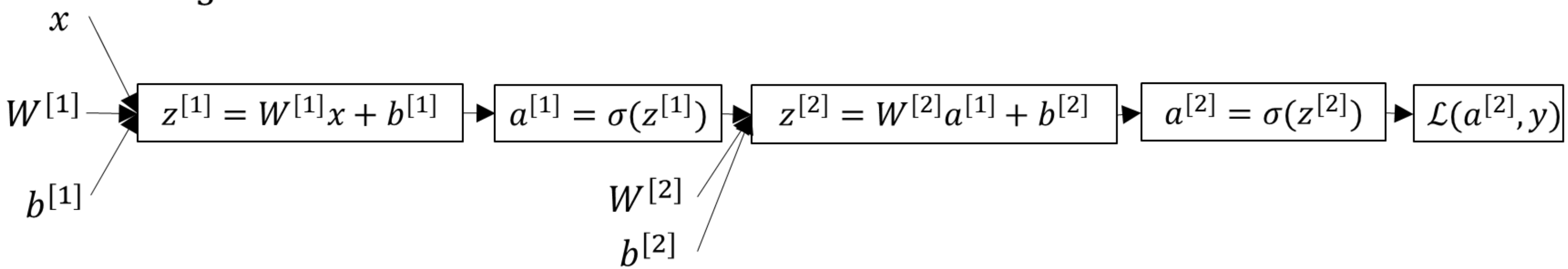
- Single neuron as a linear classifier



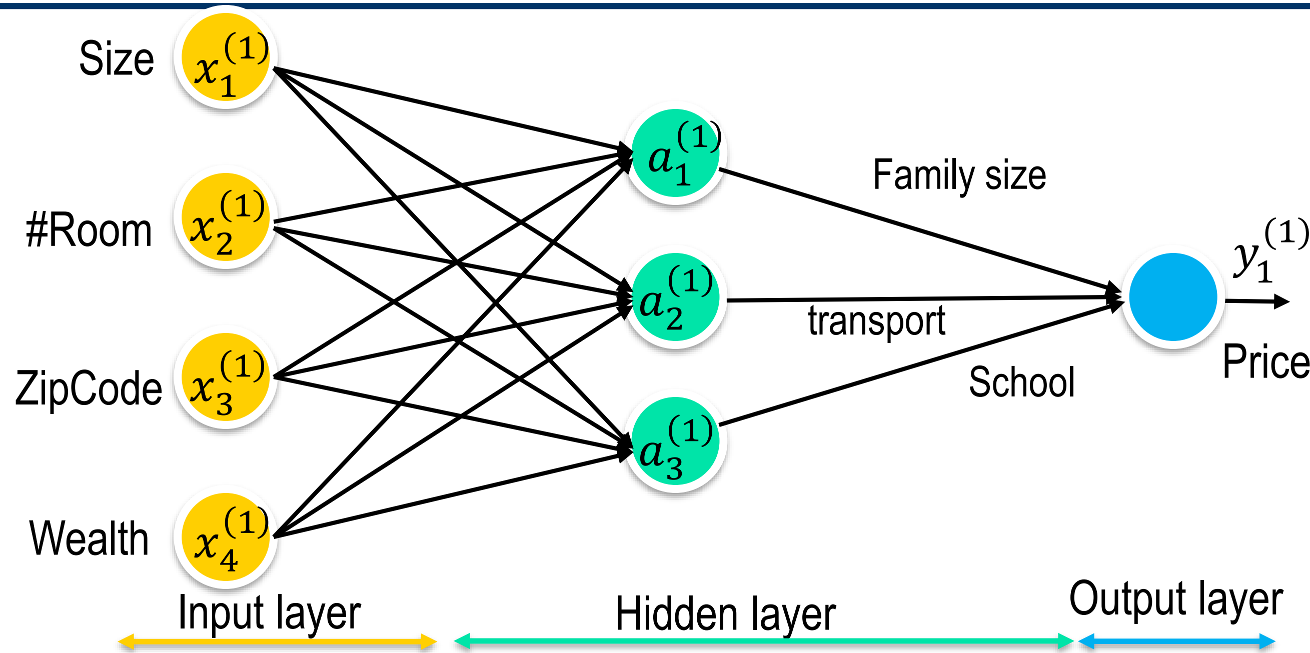


- Multiple neurons as a classifier





Neural Network Representation



$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} \rightarrow \boxed{\mathcal{F}(\cdot)} \rightarrow y$$

$\boldsymbol{x} \in \mathbb{R}^{n_x}$ $\qquad$ $y \in \{0, 1\}$

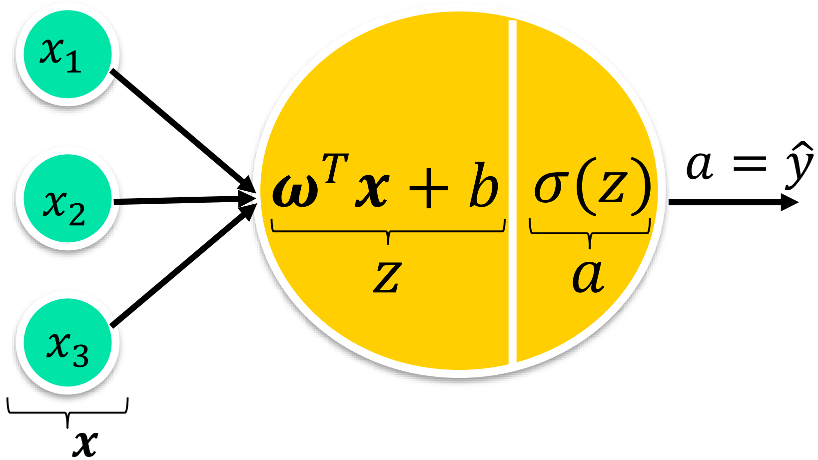
Training pairs $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \dots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$

In matrix form

$$\boldsymbol{X} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \dots & \boldsymbol{x}^{(m)} \\ | & | & & | \end{bmatrix} \qquad \boldsymbol{y} = [y^{(1)}, \dots, y^{(m)}]$$

Inputs $\qquad$ Labels

Neural Network Representation



$$z = \boldsymbol{\omega}^T\boldsymbol{x} + b$$

$$a = \sigma(z)$$

$$\boldsymbol{W}^{(\mathbf{1})} = \begin{bmatrix} \text{——} & \boldsymbol{\omega}_1^{(1)^T} & \text{——} \\ & \cdots & \\ \text{——} & \boldsymbol{\omega}_4^{(1)^T} & \text{——} \end{bmatrix}$$

$$\begin{aligned}
z_1^{(1)} &= \boldsymbol{\omega}_1^{(1)^T}\boldsymbol{x} + b_1^{(1)}, & a_1^{(1)} &= \sigma(z_1^{(1)}) \\
z_2^{(1)} &= \boldsymbol{\omega}_2^{(1)^T}\boldsymbol{x} + b_2^{(1)}, & a_2^{(1)} &= \sigma(z_2^{(1)}) \\
z_3^{(1)} &= \boldsymbol{\omega}_3^{(1)^T}\boldsymbol{x} + b_3^{(1)}, & a_3^{(1)} &= \sigma(z_3^{(1)}) \\
\underbrace{z_4^{(1)}}_{\boldsymbol{z}^{(\mathbf{1})}} &= \underbrace{\boldsymbol{\omega}_4^{(1)^T}}_{\boldsymbol{W}^T}\boldsymbol{x} + \underbrace{b_4^{(1)}}_{\boldsymbol{b}}, & \underbrace{a_4^{(1)}}_{\boldsymbol{a}^{(\mathbf{1})}} &= \sigma(z_4^{(1)})
\end{aligned}$$

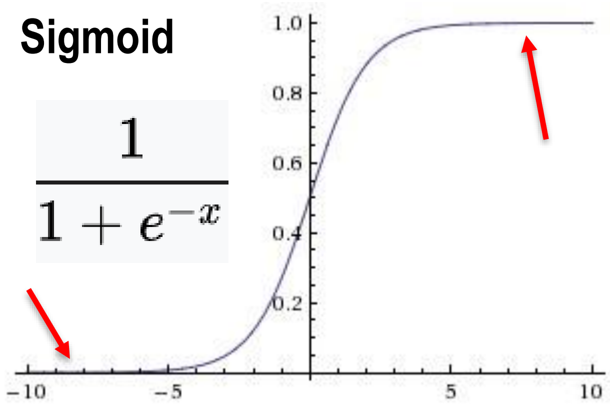
$$\boldsymbol{z}^{(\mathbf{1})} = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{a}^{(\mathbf{1})} = \sigma\left(\boldsymbol{z}^{(\mathbf{1})}\right)$$

Activation Functions (1)

- Activation function defines the output of given input

Sigmoid

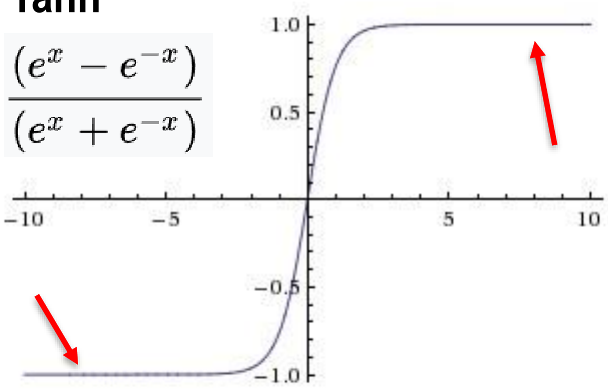
$$\frac{1}{1+e^{-x}}$$



- Enforce output range to [0, 1]
- Saturation kills the gradients
- Outputs are not zero means

Tanh

$$\frac{(e^{x}-e^{-x})}{(e^{x}+e^{-x})}$$



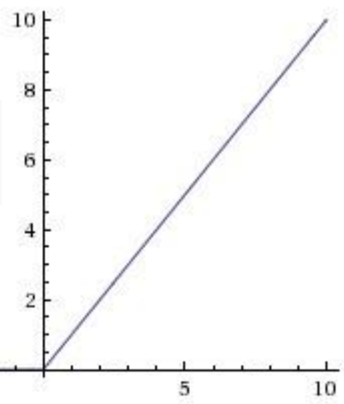
- Enforce output range to [-1, 1]
- Saturation kills the gradients

Activation Functions (2)

- Activation function defines the output of given input

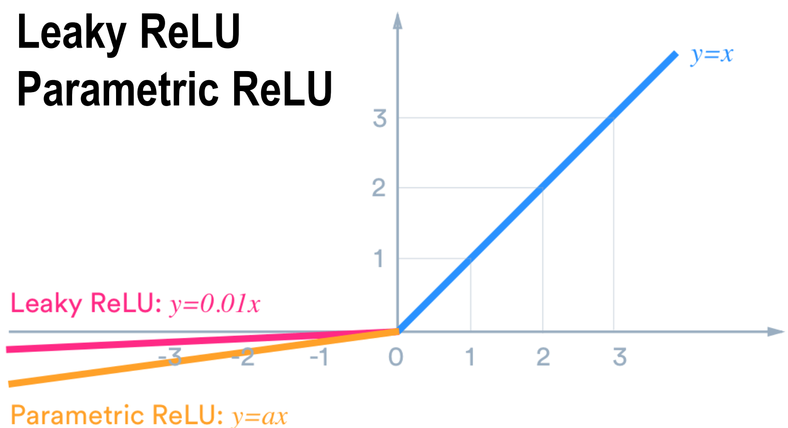
ReLU

$$\begin{cases} 0 & \text{for } x \le 0 \\ x & \text{for } x > 0 \end{cases}$$



- Not saturate
- Computation efficient
- Faster convergence
- Dead ReLU

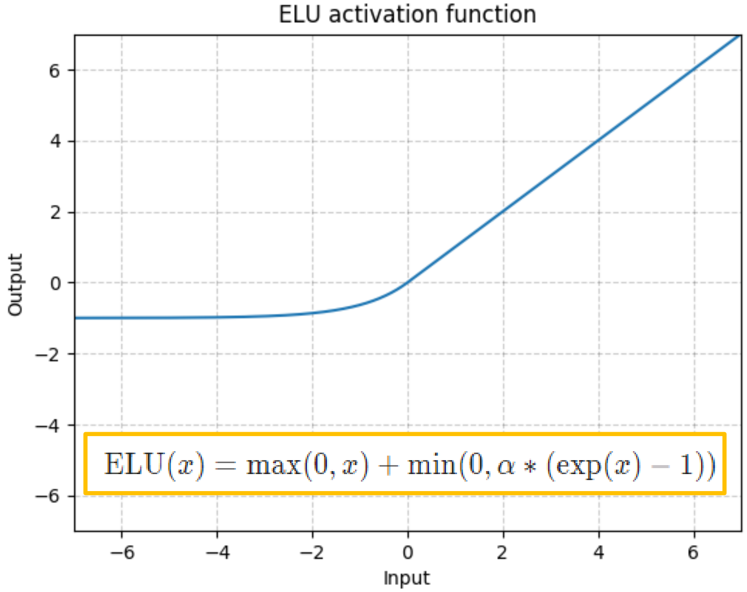
Leaky ReLU
Parametric ReLU

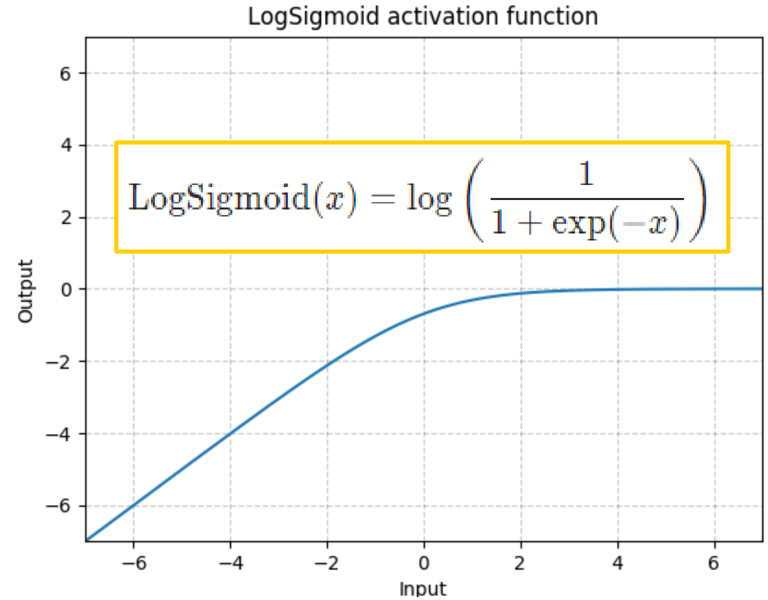


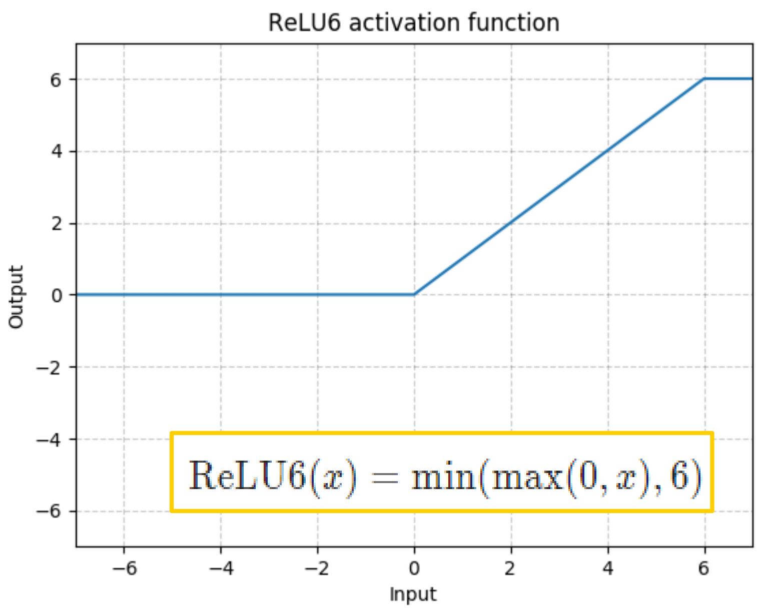
- Not saturated
- Computation efficient
- Faster convergence
- Will not "die"

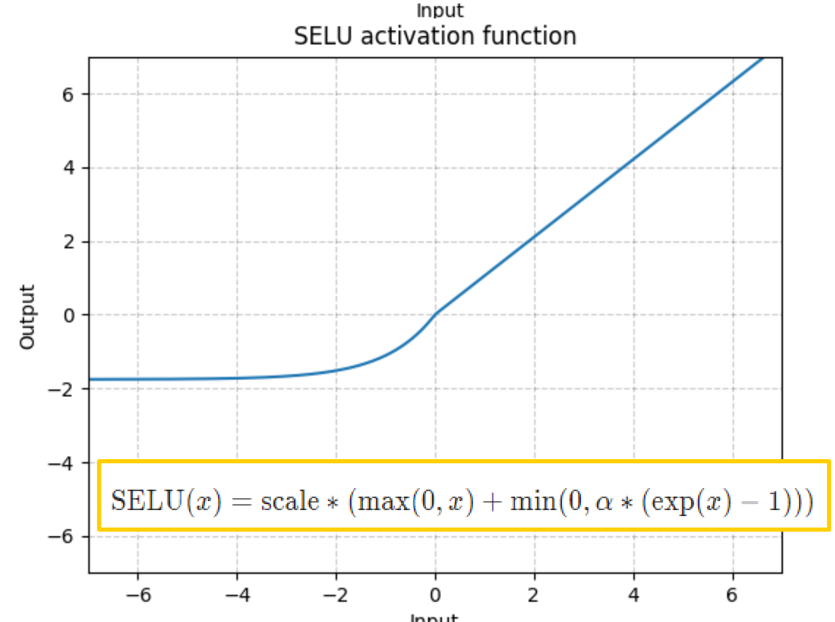
A. Karpathy et al, Backprop and intro to neural nets, presentation 2015.

Activation Function (3)









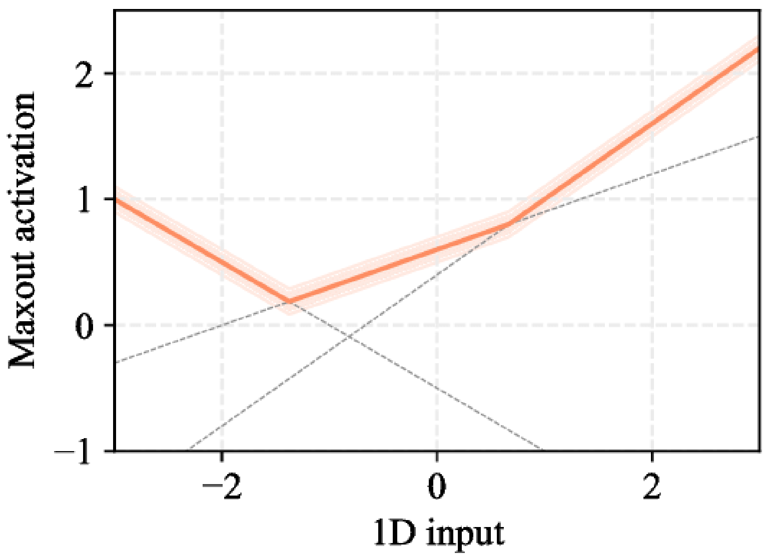
https://pytorch.org/docs/stable/_modules/torch/nn/modules/activation.html

Activation (4)

- Maxout "activation"
  - Generalized Relu & Leaky ReLU

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

    - Not saturate
    - Faster convergence
    - Not die
    - Double parameters



- Rule of thump
  - Use ReLU. Be careful with your learning rates
  - Try out Leaky ReLU / Maxout
  - Try out tanh but don't expect much
  - Never use sigmoid except for the desired output range.

A. Karpathy et al, Backprop and intro to neural nets, presentation 2015.

Back Propagation

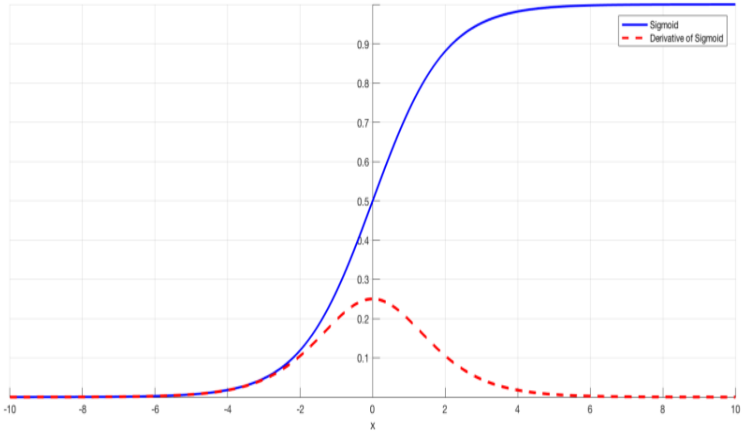
- See the Lecture note from CS231n note

Random Initialization

- Gradient vanishing (deep network)
  - Certain activation change very slow

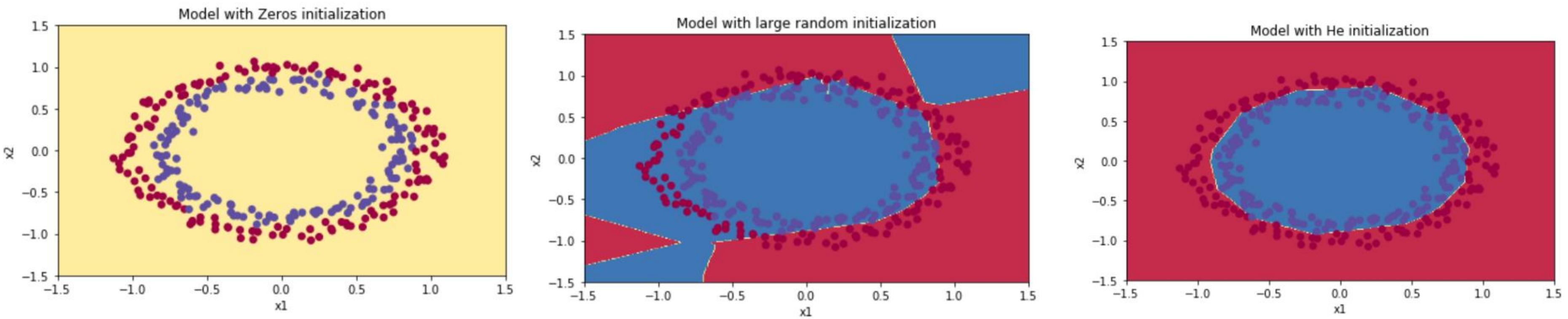
  - Gradient get smaller via back propagation
    - Too small gradients,
    - Network changing slowly
    - Slow to converge → cannot train more

- Gradient Exploding
  - Very large gradient can jump to far away
  - Objective function can be much bigger
  - Undo all previous training effort



Random Initialization

- Zero initialization or very small random initialization
  - Network is more likely to stuck at local minimal
  - All neural would follow the same directions due to same weights
- Large random initialization
  - Gradient will converse slowly if using sigmoid
  - Potential of gradient exploding
- Random initialization with a scale (depend on size of parameters)



https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78
https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94