UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/SIOC 228 Machine Learning for Physical Applications
Lecture 3: Neural Networks and Backpropagation

Yuanyuan Shi

Assistant Professor, ECE

University of California, San Diego

Reference & Acknowledgement: Stanford CS230 Deep Learning Lecture Note, Andrew Ng

Logistics & Survey Results

Homework

UC San Diego

- Assignment 1 released
- Due date: April 21st Thursday, 2:00pm (before class)
- Late submission: Late homework within 2 days of the due date is acceptable but the grades of the late homework will be 25% discounted each late day (50% discounted for 2 days). No late homework after 2 days will be accepted without a note from the health center or the student's Resident Dean. In order to ensure fairness, we will be STRICT on the late homework policy.

Project Website

UC San Diego

- Project website: https://sites.google.com/eng.ucsd.edu/ecesioc228/home

Survey Statistics

UC San Diego

What is degree program are you pursuing and which year are you? (Master, Ph.D., Junior, Senior)

Master 1st Year	124 respondents	78 %	
Master 2nd Year	12 respondents	8 %	
Ph.D. (Year 1-2)	19 respondents	12 %	
Ph.D. (>2 years)	3 respondents	2 %	
Undergraduate		0 %	

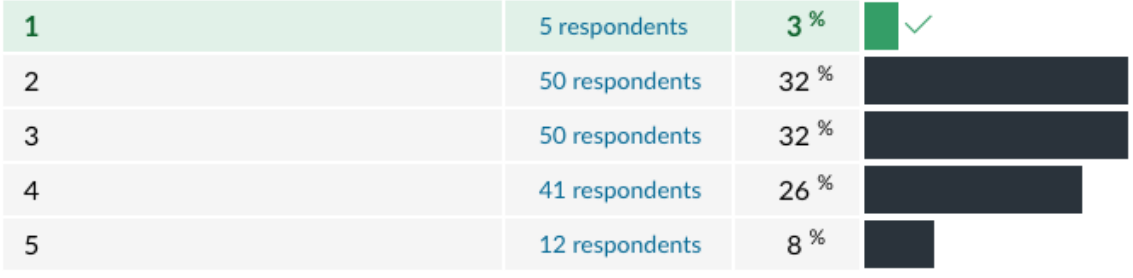
What is your major?

SIOC	2 respondents	1 %	
ECE	126 respondents	80 %	
CSE	1 respondent	1 %	
MAE	14 respondents	9 %	
BioEngineering	2 respondents	1 %	
NanoEngineering	2 respondents	1 %	
Structural Engineering	3 respondents	2 %	
Others	8 respondents	5 %	

Topic Familiarity

UC San Diego

1. Deep learning basics: supervised learning, neural networks, gradient descent, linear regression, logistic regression



1	5 respondents	3 %
2	50 respondents	32 %
3	50 respondents	32 %
4	41 respondents	26 %
5	12 respondents	8 %

(1) Never heard about it (2) Know a little bit, (3) Manageable, (4) Familiar, (5) Very good at it.

Topic Familiarity

UC San Diego

2. Advanced deep learning models: convolutional neural networks, recurrent neural networks, graph neural networks

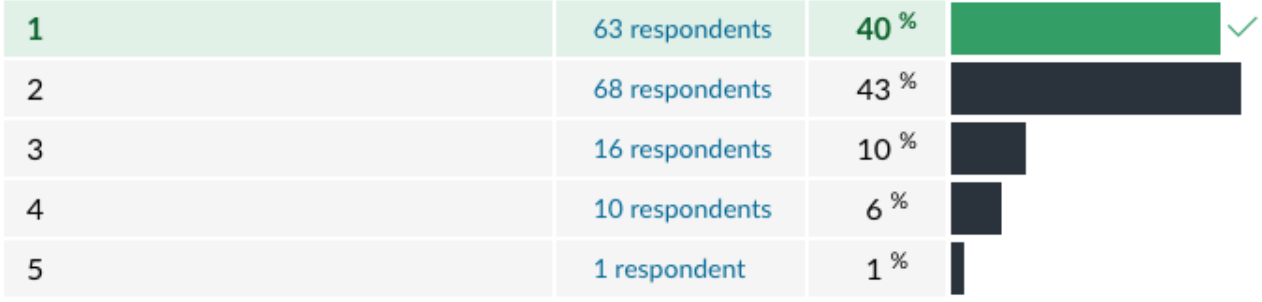
1	27 respondents	17 %	
2	67 respondents	42 %	
3	43 respondents	27 %	
4	18 respondents	11 %	
5	3 respondents	2 %	

(1) Never heard about it (2) Know a little bit, (3) Manageable, (4) Familiar, (5) Very good at it.

Topic Familiarity

UC San Diego

3. Reinforcement learning: Markov decision process, policy iteration, value iteration, Q-learning, policy gradient, actor-critic



(1) Never heard about it (2) Know a little bit, (3) Manageable, (4) Familiar, (5) Very good at it.

Topic Familiarity

UC San Diego

4. Physics-informed machine learning, partial differential equations

1	53 respondents	34 %	
2	63 respondents	40 %	
3	27 respondents	17 %	
4	13 respondents	8 %	
5	2 respondents	1 %	

(1) Never heard about it (2) Know a little bit, (3) Manageable, (4) Familiar, (5) Very good at it.

Topic Familiarity

UC San Diego

5. Coding in Python

1	2 respondents	1 %
2	16 respondents	10 %
3	29 respondents	18 %
4	52 respondents	33 %
5	59 respondents	37 %

(1) Never heard about it (2) Know a little bit, (3) Manageable, (4) Familiar, (5) Very good at it.

Today's Lecture

UC San Diego

- Forward and Backward Computation of Logistic Regression (1-layer Neural Network)
- One Hidden Layer Neural Network (or called 2-layer NN)
- Activation Functions
- Forward and Backward Computation of Neural Network with L Hidden Layers

# of hidden layer + output layer

- Universal Approximation Theorem & Why Deep?

Forward and Backward Computation of Logistic Regression

Forward Computation of Logistic Regression

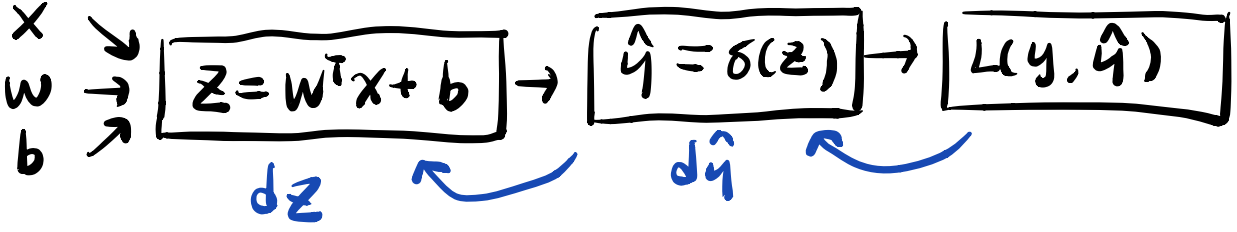
UC San Diego

Logistic regression with 1 sample (x, y), $x \in R^n$, $y \in \{0, 1\}$

$\hat{y} = \sigma(w^T x + b)$, $z = w^T x + b$

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

Computation graph



Backward Computation of Logistic Regression

$$\text{"}da\text{"} = \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

"chain rule"

$$\text{"}dz\text{"} = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = \left[-\frac{y}{a} + \frac{1-y}{1-a}\right] \frac{\sigma(z)[1-\sigma(z)]}{a(1-a)}$$

$$= \left[-\frac{y}{a} + \frac{1-y}{1-a}\right] a(1-a)$$

$$= -y(1-a) + (1-y)a = a - y$$

Backward Computation of Logistic Regression

$$\text{"}dw\text{"} = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial z}\cdot\frac{\partial z}{\partial w} = x\,dz$$

dz

$$\text{"}db\text{"} = dz$$

Gradient update:

$$w \leftarrow w - \eta\, dw$$

$$b \leftarrow b - \eta\, db$$

η: learning rate

★ $w \in R^n$ is a vector, so $\frac{\partial z}{\partial w}$ is is a vector: $\frac{\partial z}{\partial w} = \left(\frac{\partial z}{\partial w_1}, \cdots, \frac{\partial z}{\partial w_n}\right)$ ← also called gradient of z w.r.t. w

Clarification on Notations

UC San Diego

☆ Both $\frac{df}{dx}$, $\frac{\partial f}{\partial x}$ are used to denote derivatives.

- $\frac{df}{dx}$ is more commonly used for function of single variable.
- If $f(x_1, x_2, x_3)$ is a multi-variate function, $\frac{\partial f}{\partial x_1}$ is partial derivative of f w.r.t. x_1, given x_2, x_3 fixed
- $\frac{\partial f}{\partial x} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}\right)$; also called gradient of f

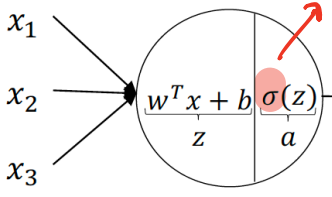
☆ We introduce a short-hand notation in this course

$\boxed{dx} = \frac{df}{dx}$ (single-variate); $dx = \frac{\partial f}{\partial x} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}\right)$ ← multi-variate

One Hidden Layer Neural Network

One Hidden Layer Neural Network

UC San Diego

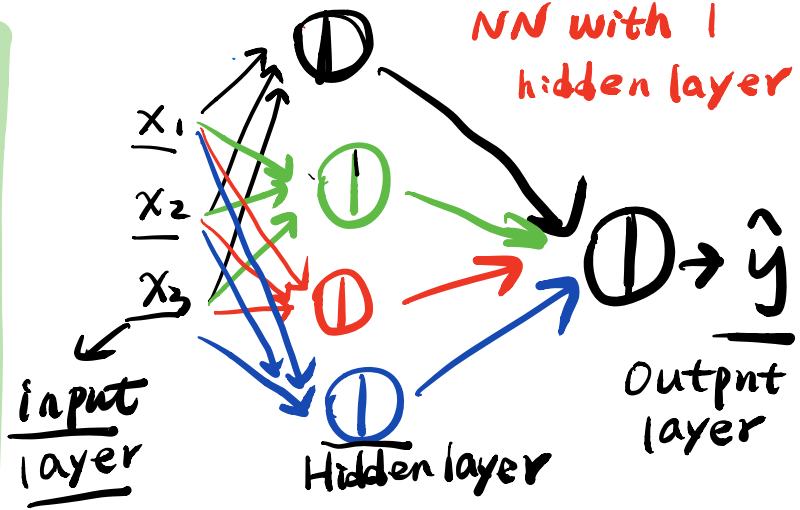


• activation function (talk more later)

$$z = w^T x + b$$

$$a = \sigma(z)$$

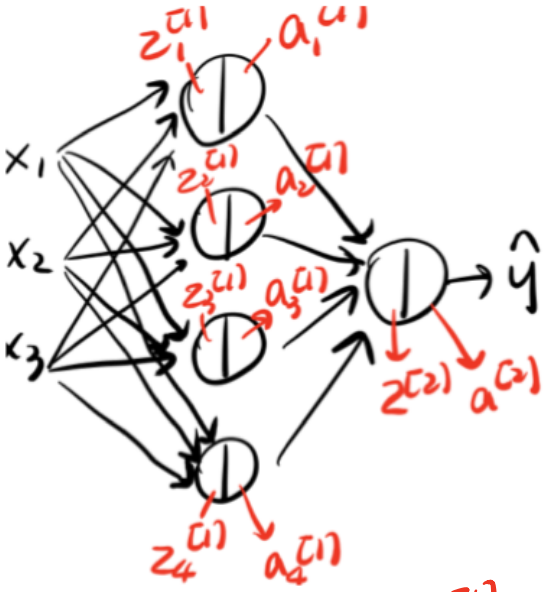
⇒ building block for neural network, neuron (perceptron)



• multiple neurons form a layer

• Multiple layers form a neural network (multi-layer perceptron)

Forward Computation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]} \qquad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]} \qquad a_2^{[1]} = \sigma(z_2^{[1]})$$

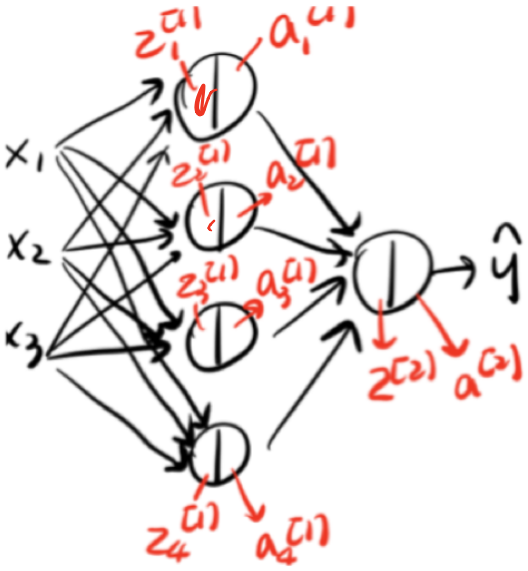
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]} \qquad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]} \qquad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} -w_1^{[1]T}- \\ -w_2^{[1]T}- \\ -w_3^{[1]T}- \\ -w_4^{[1]T}- \end{bmatrix}}_{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}}$$

Forward Computation

UC San Diego



Given input $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

$$z^{[1]} = W^{[1]} x + b^{[1]}, \quad a^{[1]} = \sigma(z^{[1]})$$

(4×1) $(4\times3)\times(3\times1)$ (4×1)

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

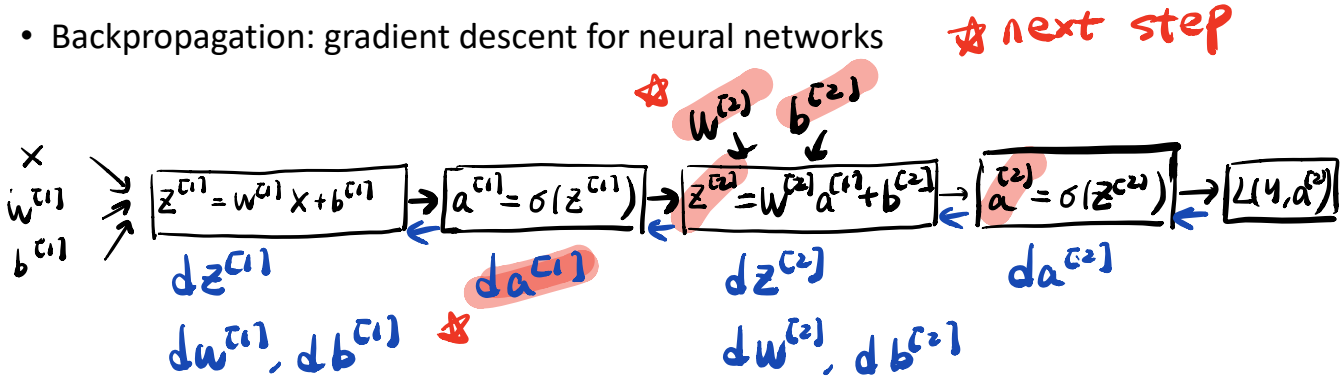
(1×1) (1×4) (4×1) (1×1)

$$\hat{y} = a^{[2]} = \sigma(z^{[2]})$$

Backward Computation

UC San Diego

- Backpropagation: gradient descent for neural networks

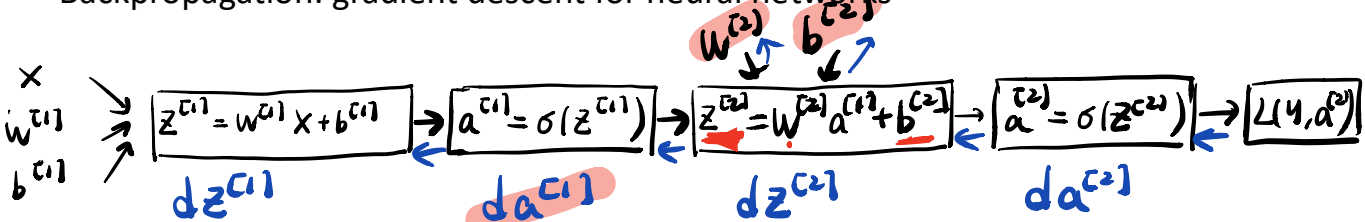


- Suppose: $L(y, a^{[2]}) = -[y \log a^{[2]} + (1-y)\log(1-a^{[2]})]$
- $da^{[2]} = \frac{\partial L(y, a^{[2]})}{\partial a^{[2]}} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$, $dz^{[2]} = a^{[2]} - y$

Backward Computation

UC San Diego

- Backpropagation: gradient descent for neural networks



- $dw^{[2]} = \frac{\partial L(y, a^{[2]})}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}}$ "chain rule"

$= dz^{[2]} \cdot a^{[1]T}$

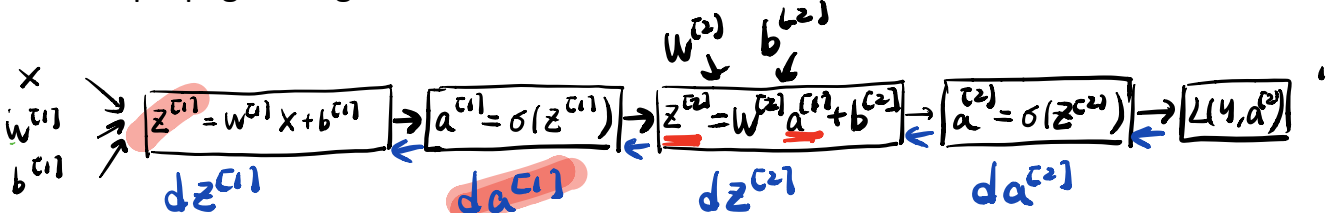
"Computed"

- $db^{[2]} = dz^{[2]}$ "Computed"

Backward Computation

UC San Diego

- Backpropagation: gradient descent for neural networks



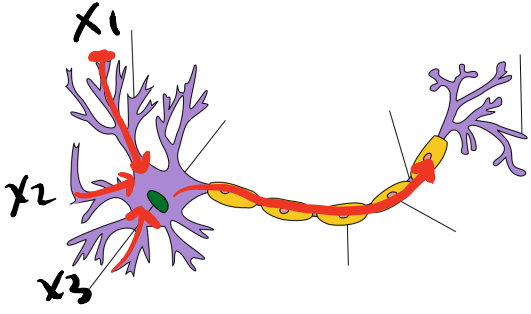
★ next step

- $da^{[1]} = W^{[2]T} \cdot dz^{[2]}$
- $dz^{[1]} = da^{[1]} * g^{[1]\prime}(z^{[1]})$, $*$: element-wise multiplication
  $(n^{[1]}, 1)$ $(n \times 1)$ $(n \times 1)$
- $dW^{[1]} = dz^{[1]} \cdot x^T$, $db^{[1]} = dz^{[1]}$

Activation Functions

What is an Activation Function?

UC San Diego



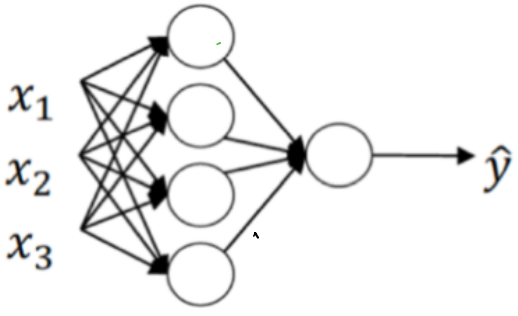
☆ Inspired by neuron activities in the brain

* In "Neural Network", it refers to

how weighted sum → is tranformed to (often nonliner) → output

Why Do We Need Nonlinear Activation Functions?

UC San Diego



No activation:

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

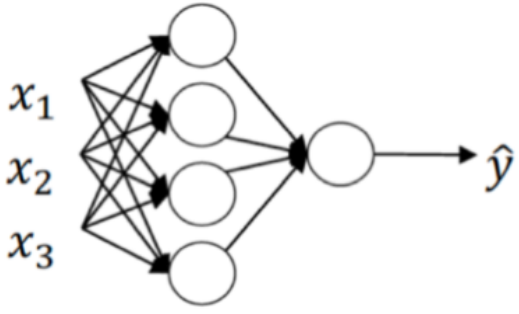
$$a^{[2]} = z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

$$= w^{[2]}\left(w^{[1]}x + b^{[1]}\right) + b^{[2]}$$

$$= \underbrace{w^{[2]}w^{[1]}}_{w'}x + \underbrace{w^{[2]}b^{[1]} + b^{[2]}}_{b'}$$

- Without activation func, Can only represent linear relationship.

Why Do We Need Nonlinear Activation Functions?



With activation

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

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

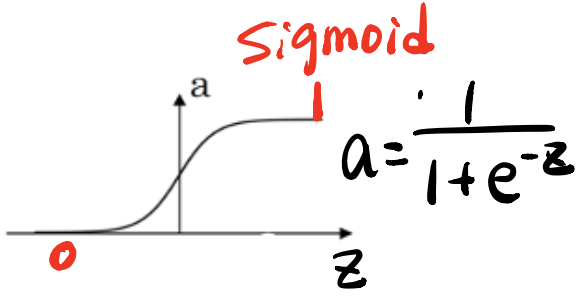
$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$

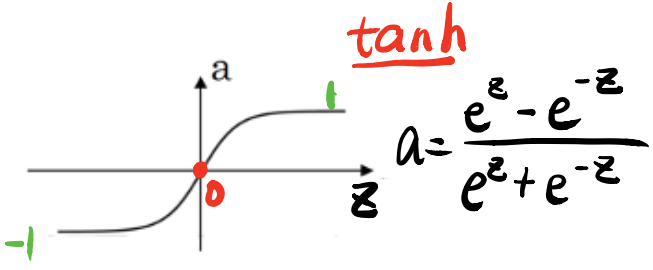
$$= w^{[2]} \sigma(w^{[1]} x + b^{[1]}) + b^{[2]}$$

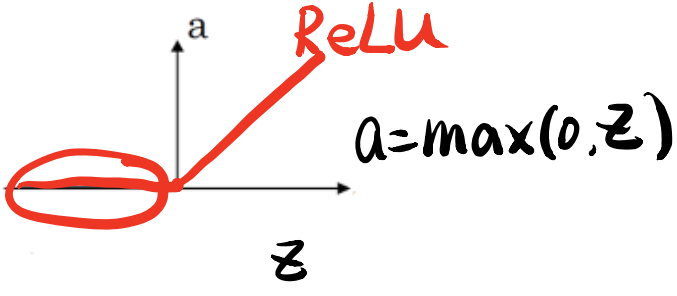
$$a^{[2]} = \sigma(z^{[2]})$$

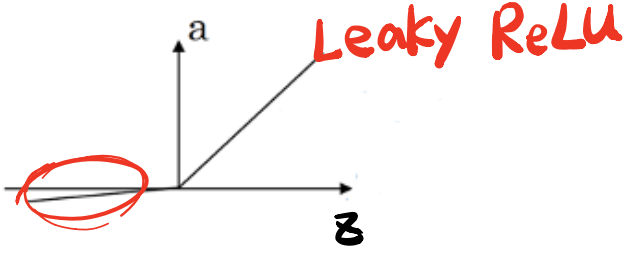
Different Activation Functions

UC San Diego



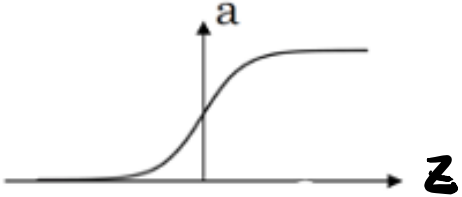






Derivatives of Activation Functions

UC San Diego



Sigmoid function

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

$$\Rightarrow \sigma'(z) = \sigma(z)[1-\sigma(z)]$$

- $z = 100$, $\sigma(z) \approx 1$

$$\sigma'(z) \approx 1 \cdot (1-1) = 0$$

- $z = -100$, $\sigma(z) \approx 0$

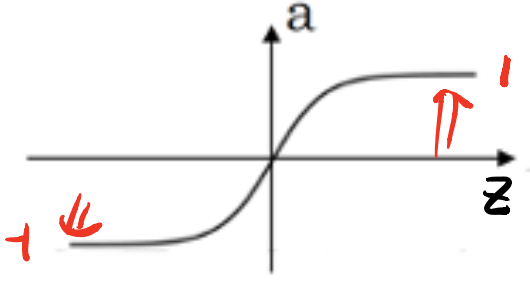
$$\sigma'(z) \approx 0[1-0] = 0$$

- $z = 0$, $\sigma(z) = \frac{1}{2}$, $\sigma'(z) = \frac{1}{4}$

Derivatives of Activation Functions

UC San Diego

Tanh function



Tanh

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

$$g'(z) = \frac{d}{dz} g(z)$$

$$= \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2}$$

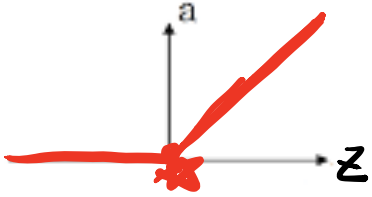
$$= 1 - \tanh^2(z)$$

- $z = 100$, $\tanh(z) \approx 1$, $g'(z) \approx 0$
- $z = -100$, $\tanh(z) \approx -1$, $g'(z) \approx 0$
- $z = 0$, $\tanh(0) = 0$, $g'(z) = 1$

Derivatives of Activation Functions

UC San Diego

ReLU

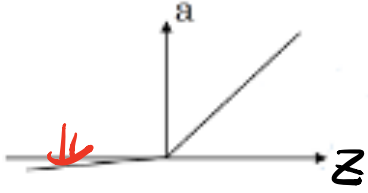


$g(z) = \max(0, z)$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \\ \text{undefined} & \text{if } z = 0 \end{cases}$$

$z=0$ not differentiable; (empirically), Set it as 0 or 1

Leaky-Relu

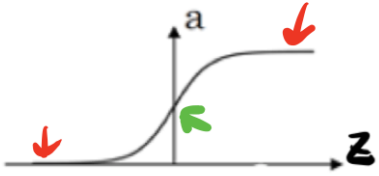


$g(z) = \max(0.01z, z)$

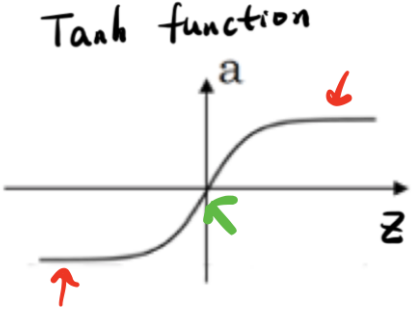
$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$$

Pros and Cons of Different Activations

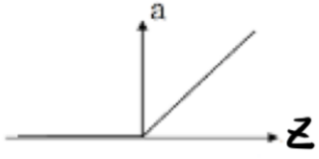
UC San Diego



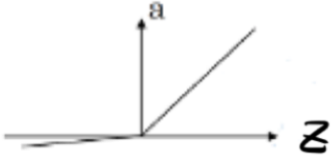
Sigmoid function



ReLU



Leaky-Relu

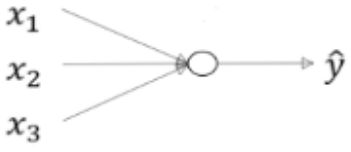


Forward and Backward

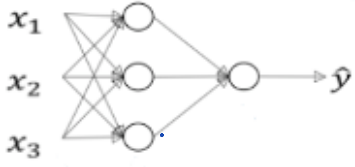
Computation of Neural Network

Neural Networks with Different Number of Layers

UC San Diego

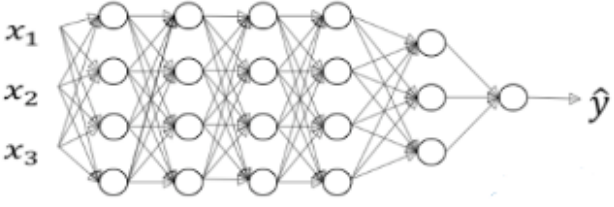


logistic regression



1 hidden layer

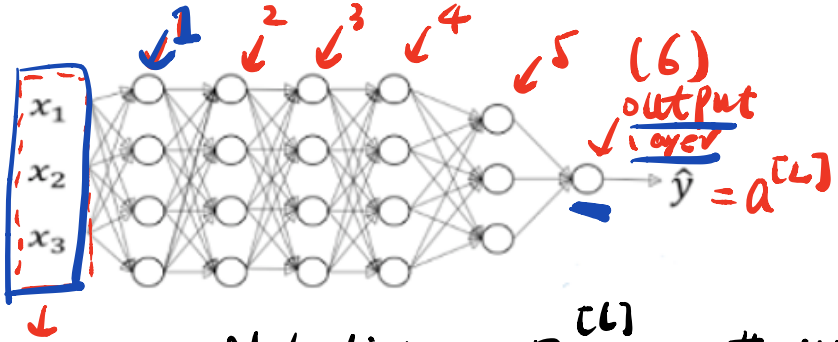
2 hidden layers



5 hidden layers

Notation

UC San Diego



$L = 6$ (6-layer NN)

5 hidden layers + 1 output layer

Notations: $n^{[l]}$ = # units in layer l

$a^{[l]}$ = activations in layer l

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

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$, $a^{[0]} = x$

weights & bias in layer l

$n^{[0]} = n_x = 3$

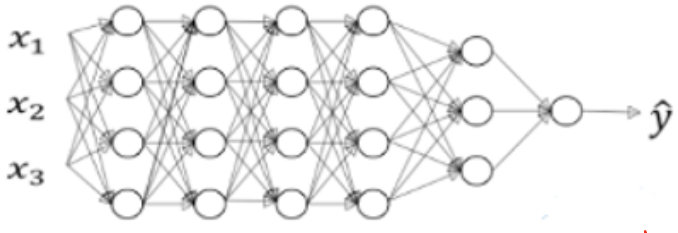
$n^{[1]} = 4$, $n^{[2]} = 4$

$n^{[3]} = 4$, $n^{[4]} = 4$

$n^{[5]} = 3$, $n^{[6]} = n_y = 1$

Forward Computation

UC San Diego



$$W^{[l]} : (n^{[l]}, n^{[l-1]})$$

$$b^{[l]} : (n^{[l]}, 1)$$

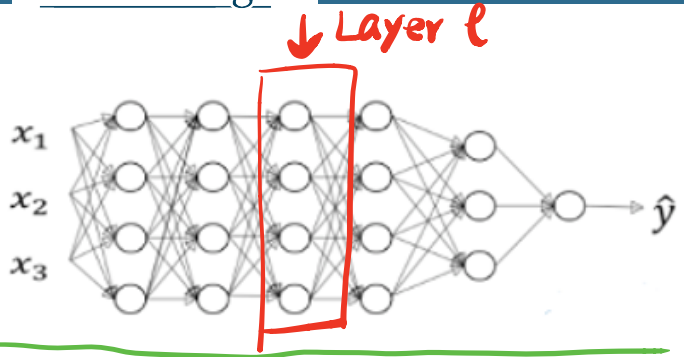
weights & bias in layer l

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}, \quad a^{[0]} = x$$

$$a^{[l]} = \sigma^{[l]}(z^{[l]})$$

Backward Computation

UC San Diego



backward, elementwise

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$, $db^{[l]} = dz^{[l]}$

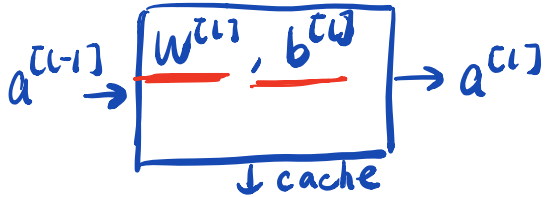
$da^{[l-1]} = w^{[l]T} dz^{[l]}$

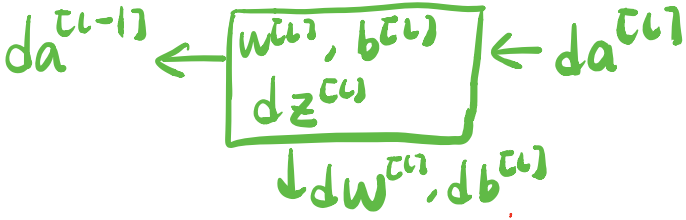
Layer ℓ

forward

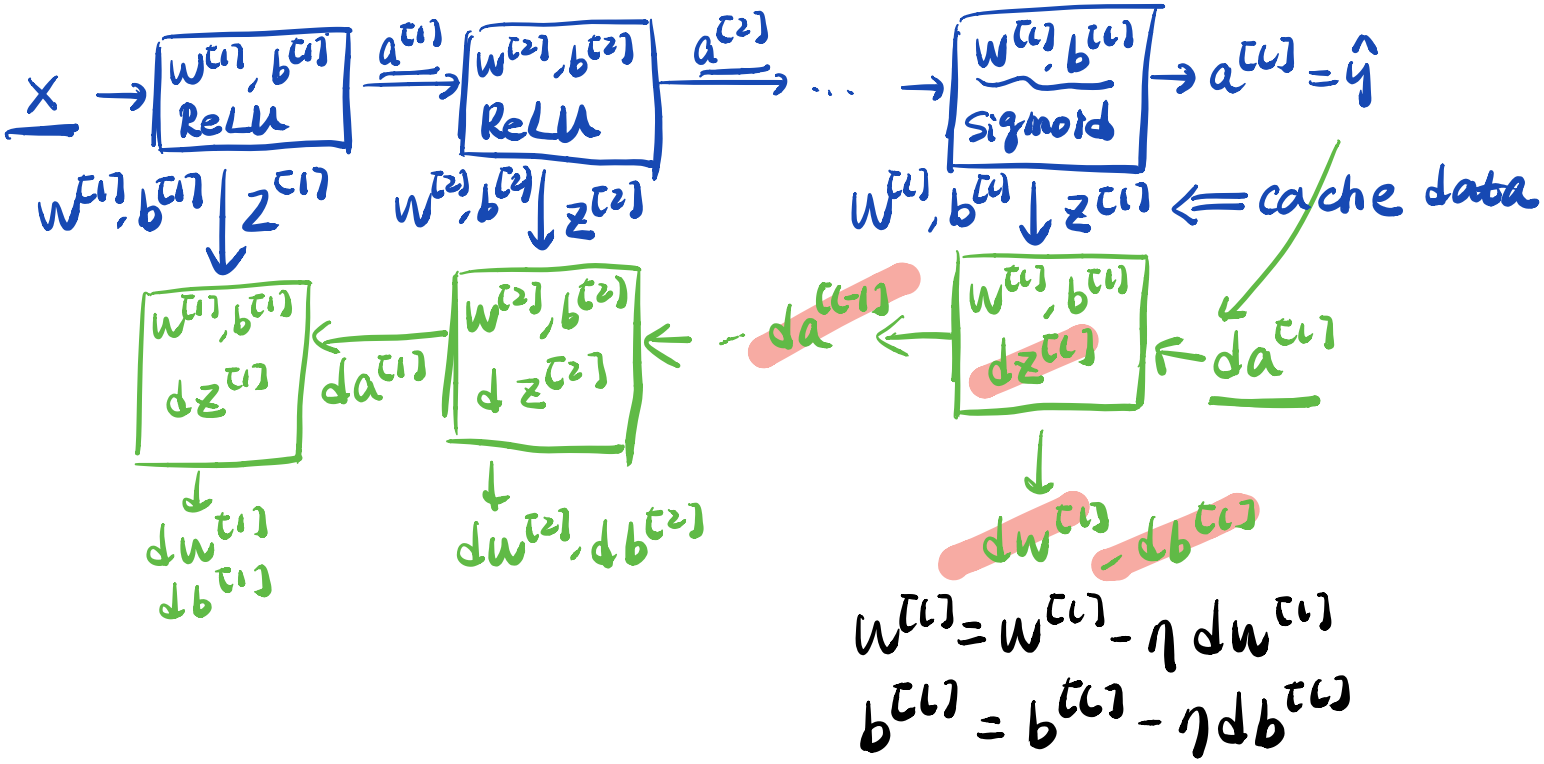
$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = \sigma(z^{[l]})$





Forward and Backward Computation



Universal Approximation Theorem & Why Deep?

Universal Function Approximation

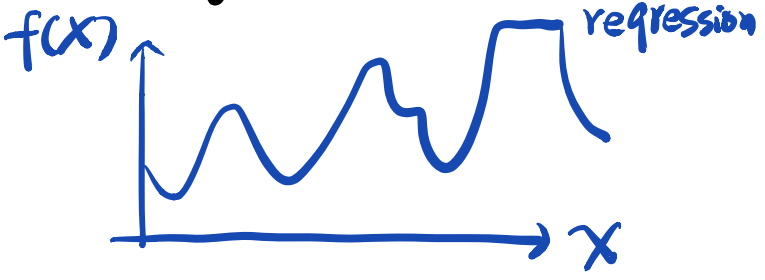
UC San Diego

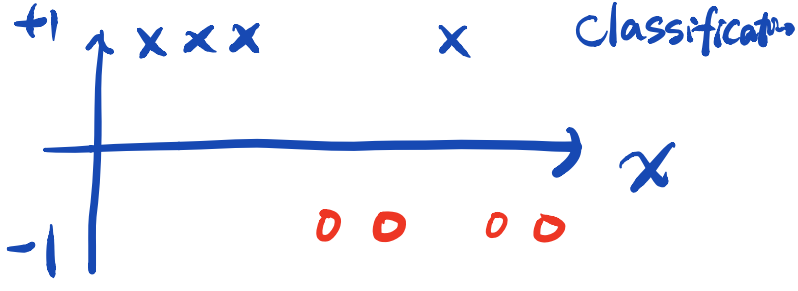
- Informal theorem:

  "a wide enough 1 hidden layer NN can fit any function"

- Emperical findings:

  "my NN can fit any labelling of the train data"

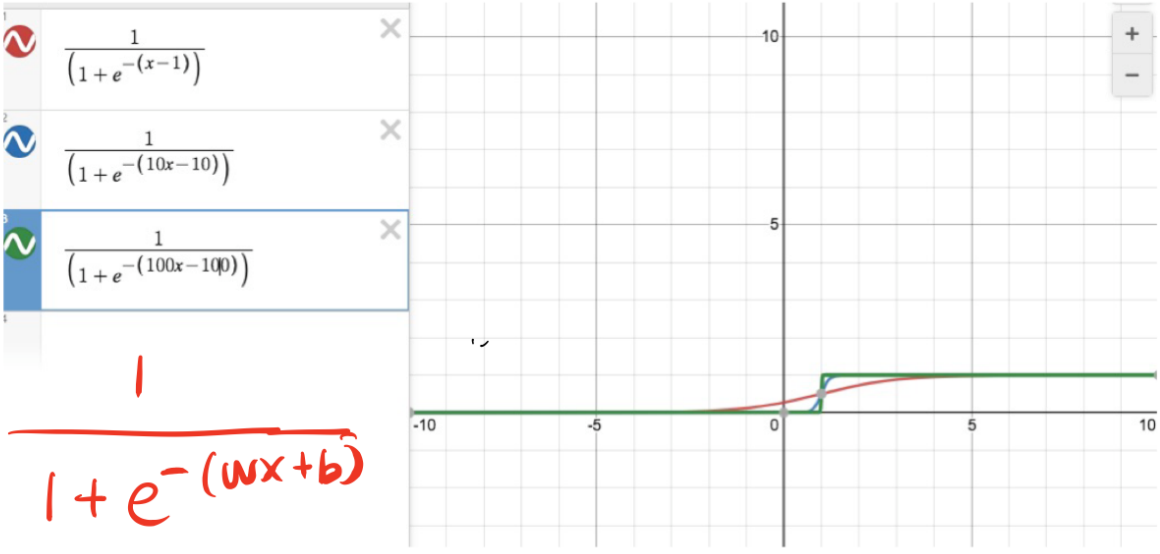




Universal Function Approximation

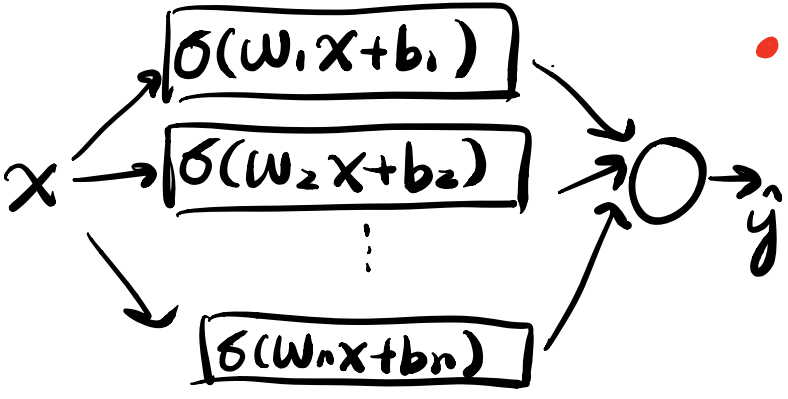
UC San Diego

Consider a 1d input $x \in \mathbb{R}$ & sigmoid activation



$$\frac{1}{1+e^{-(wx+b)}}$$

- When w is very large, Sigmoid function look like step functions

- 

- Sum of step functions can approximate any function shape!

Universal Function Approximation

UC San Diego

deep networks are *universal function approximators* (Hornik, 1991)

→ with enough units & layers, can approximate *any* function

Classic Results

Theorem: For one layer neural networks, the error between the estimated network and a target function f is shown to be bounded by

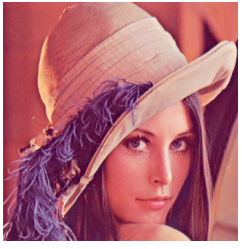
$$O(\frac{C_f^2}{n}) + O(\frac{nd}{N}\log N)$$

here n is the number of nodes, d is the input dimension of the function, N is the number of training data, and C_f is the first absolute moment of the Fourier magnitude distribution of f.

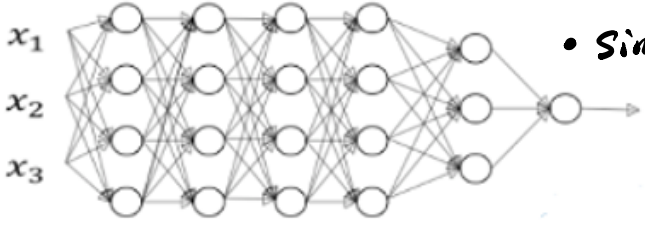
Barron, Andrew R. "Approximation and estimation bounds for artificial neural networks." *Machine learning* 14.1 (1994): 115-133.

Why Deep Representation?

UC San Diego



- hierarchy of concepts
  - simpler concepts → more abstract



pixels and edges

more complex shapes: eyes & mouths

faces