
Neural Networks and Deep Learning

박시은, 오영빈, 최유진,
한정호



Project Proposal

Topic Neural Networks, Deep Learning

Description We will learn basic concepts of neural network & deep learning and discuss as a team every friday 3-5pm (75 min: lecture / 45 min: discussion).

Expected Duration 9 weeks (whole semester)

https://docs.google.com/spreadsheets/d/1UGbADy_LJ_xxtNhIWLFJKB1_Z8BFpPHPizxV6r5JNnE/edit#gid=0

Team Member 박시은, 오영빈, 최유진, 한정호

Timeline

Course 1: Neural Networks and Deep Learning (4-5 weeks)



Course 2 : Improving Deep Neural Networks: Hyperparameter Tuning, Regularization, and Optimization (4-5 weeks)

Coursera Deep Learning Specialization

COURSE	Neural Networks and Deep Learning
1	4.9 115,284 ratings In the first course of the Deep Learning Specialization, you will study the foundational concept of neural networks and deep learning. By the end, you will be familiar with the significant technological trends driving the rise of deep learning: build, train, and apply fully connected deep neural networks; implement
SHOW ALL	
COURSE	Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization
2	4.9 61,140 ratings In the second course of the Deep Learning Specialization, you will open the deep learning black box to understand the processes that drive performance and generate good results systematically.
SHOW ALL	
COURSE	Structuring Machine Learning Projects
3	4.8 48,450 ratings In the third course of the Deep Learning Specialization, you will learn how to build a successful machine learning project and get to practice decision-making as a machine learning project leader.
SHOW ALL	
COURSE	Convolutional Neural Networks
4	4.9 40,742 ratings In the fourth course of the Deep Learning Specialization, you will understand how computer vision has evolved and become familiar with its exciting applications such as autonomous driving, face recognition, reading radiology images, and more.
SHOW ALL	
COURSE	Sequence Models
5	4.8 28,367 ratings In the fifth course of the Deep Learning Specialization, you will become familiar with sequence models and their exciting applications such as speech recognition, music synthesis, chatbots, machine translation, natural language processing (NLP), and more.
SHOW ALL	

<https://www.coursera.org/specializations/deep-learning>

Table of Contents

- ❖ Our Progress
- ❖ What is a Neural Network?
- ❖ Logistic regression as neural network
- ❖ Gradient Descent
- ❖ Intuition about Derivatives
- ❖ Next Steps

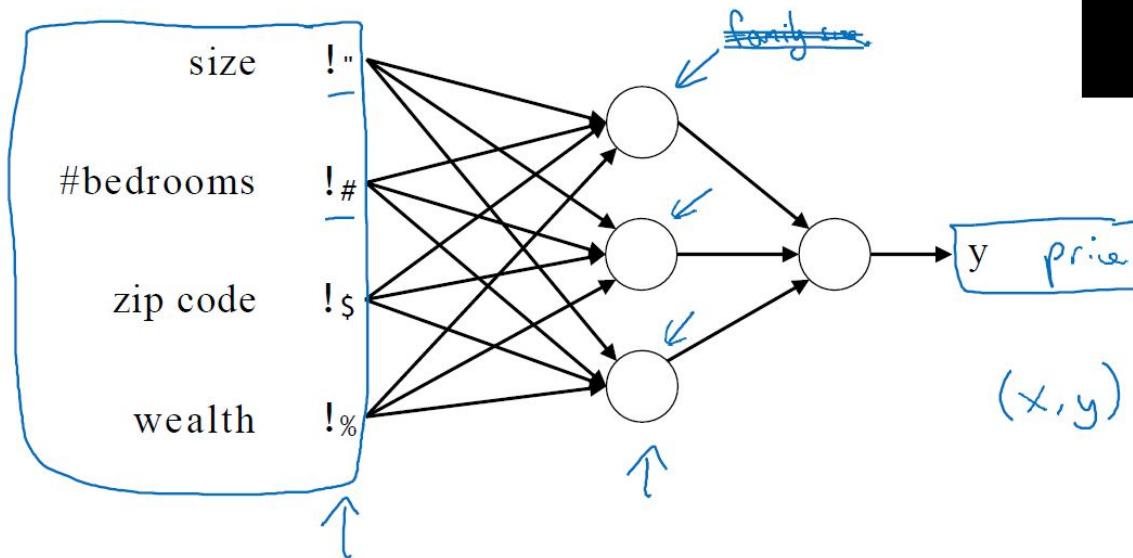
Our Progress

Series 1 Neural Networks and Deep Learning		
Week1	<u>Welcome</u>	5
	What is a Neural Network?	7
	Supervised Learning with Neural Networks	8
	Why is Deep Learning taking off?	10
	About this Course	2
Week2	<u>Binary Classification</u>	8
	Logistic Regression	5
	Logistic Regression Cost Function	8
	Gradient Descent	11
	Derivatives	7
	More Derivative Examples	10

What is a Neural Network?

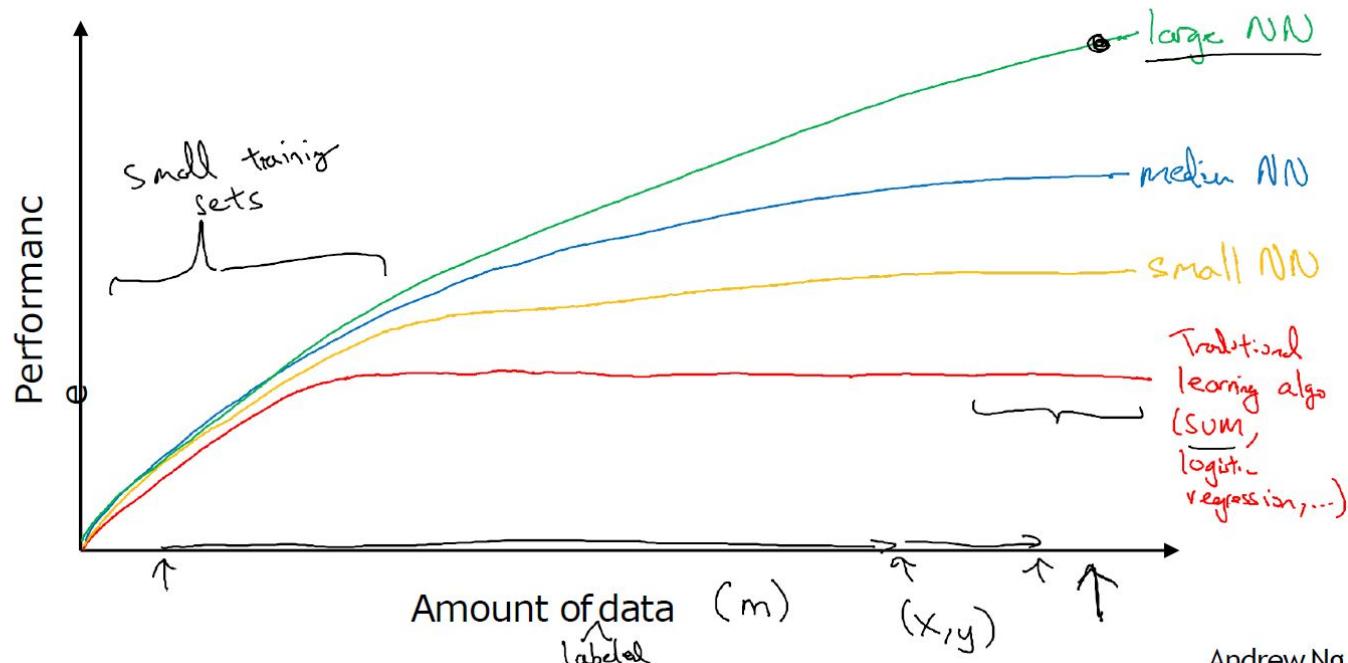
Housing Price Prediction

Drawing of
previous Image



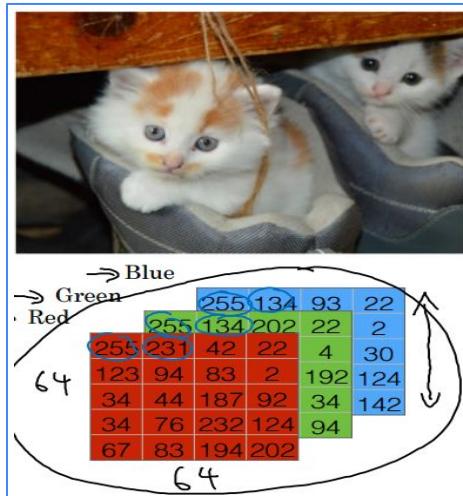
What is a Neural Network?

Scale drives deep learning progress



Binary classification

- Basics of neural network problem
- Supervised learning problem when the output y are all either zero or one.

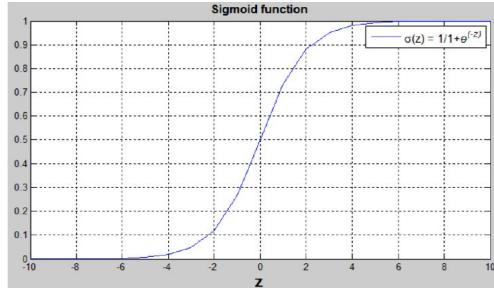


→ 1 (cat) vs 0 (non cat)

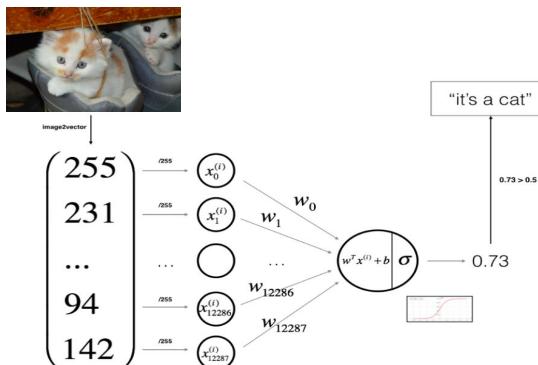
y

Logistic regression

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$$



Logistic Regression with a Neural Network Mindset



Cost function

- Loss (error) function: computes the error for a single training example.

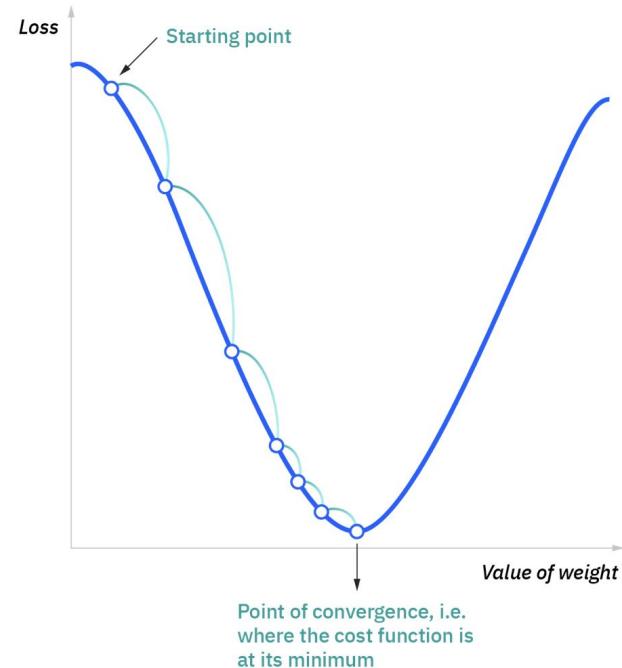
$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

- Cost function: the average of the loss function of the entire training set.
- Goal: find the parameters w and b that minimize the overall cost function.

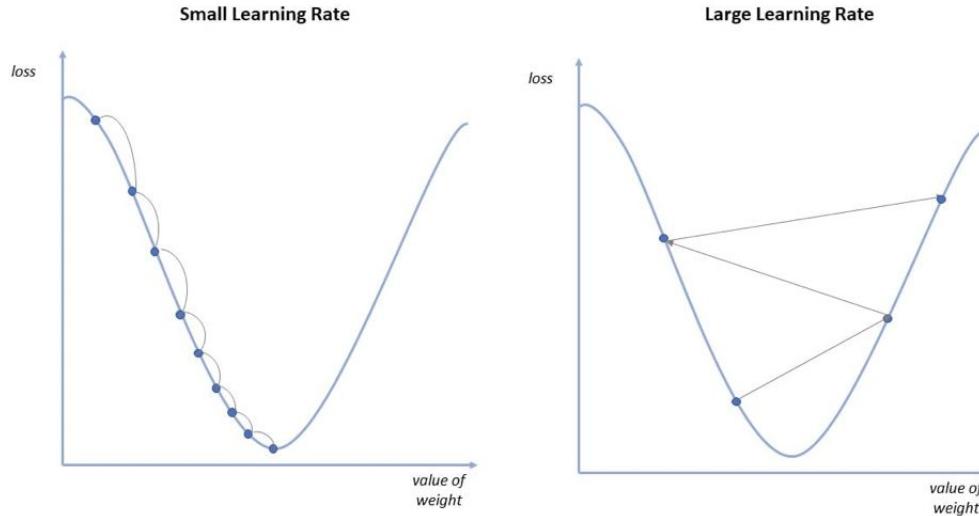
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Gradient Descent

- Optimization algorithm to minimize cost function (convex)
- Take repeated steps in the opposite direction of the gradient of the function at the current point
- Repeat $x \leftarrow x - \alpha * f'(x)$
(α : learning rate, small positive number)
- Justification: The gradient points in the direction of the steepest ascent

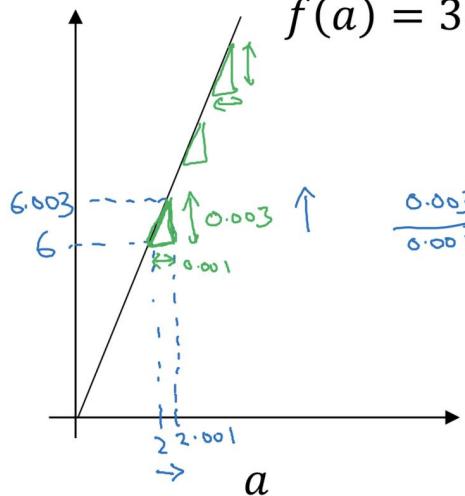


Gradient Descent



- Importance of making initial guess of optimal learning rate

Intuition about derivatives



$$f(a) = 3a$$

$$\frac{0.003}{0.001}$$

height
width

$$\rightarrow a = 2$$

$$f(a) = 6$$

$$a = 2.001$$

$$f(a) = 6.003$$

slope (derivative) of $f(a)$
at $a=2$ is 3

$$\rightarrow a = 5$$

$$f(a) = 15$$

$$a = 5.001$$

$$f(a) = 15.003$$

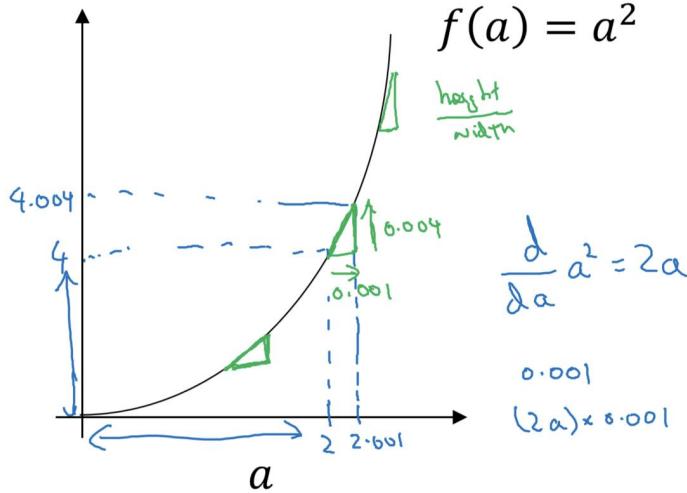
slope at $a=5$ is also 3

$$\frac{d f(a)}{d a} = 3 = \frac{d}{d a} f(a)$$

0.001
0.00000001
0.0000000001

Andrew Ng

Intuition about derivatives



$$a=2 \quad f(a)=4$$
$$a=2.001 \quad f(a) \approx 4.004$$
$$0.001 \leftarrow$$
$$0.00000\dots01 \leftarrow$$

$\frac{d}{da} f(a) = 4$ when $a=2$.
slope (derivative) of $f(a)$ at $a=2$ is 4.

$$\frac{d}{da} f(a) = 4$$
$$f(a) = 25$$
$$a=5$$
$$a=5.001$$
$$f(a) \approx 25.010$$

$$\frac{d}{da} f(a) = 10$$
$$f(a) = 25$$
$$a=5$$
$$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a$$

Andrew Ng

Next Steps

- Python Implementation of Machine Learning
- Introduction to Neural Network

Table of Contents

- ❖ Our Progress
 - ❖ What is Vectorization?
 - ❖ Broadcasting
 - ❖ Next Steps
-

Our Progress

Series 1 Neural Networks and Deep Learning		
Week	Topic	Score
Week 2	Computation Graph	3
	Derivatives with a Computation Graph	14
	Logistic Regression Gradient Descent	6
	Gradient Descent on m Examples	8
	Vectorization	8
	More Vectorization Examples	6
	Vectorizing Logistic Regression	7
	Vectorizing Logistic Regression's Gradient Output	9
	Broadcasting in Python	11
	A Note on Python/Numpy Vectors	6
Week 3	Quick tour of Jupyter/iPython Notebooks	3
	Explanation of Logistic Regression Cost Function (Optional)	7
	Neural Networks Overview	4
Week 3	Neural Network Representation	5
	O(10/7)	O(10/14)
O(10/14)		

What is Vectorization?

What is vectorization?

$$z = \underbrace{w^T x + b}$$
$$w = \begin{bmatrix} : \\ : \\ : \end{bmatrix} \quad x = \begin{bmatrix} : \\ : \\ : \end{bmatrix} \quad \begin{array}{l} w \in \mathbb{R}^{n_x} \\ x \in \mathbb{R}^{n_x} \end{array}$$

Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

Vectorized

$$z = \underbrace{\text{np.dot}(w, x)}_{w^T x} + b$$

\rightarrow GPU } SIMD - single instruction
 \rightarrow CPU } multiple iteration.

Andrew Ng

- Iterative for loop vs vectorization
- Enables parallel processing
- Minimizes the running & execution time of code
- Vectorization Example:

<https://colab.research.google.com/drive/1np5oDUVOMFPx8M-NScuEakZgDWjufuLe#scrollTo=nMUIJO0cjmPF>

What is Vectorization?

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

$\Rightarrow u = np.zeros((n,1))$
 $\Rightarrow \text{for } i \text{ in range}(n):$ \leftarrow
 $\Rightarrow u[i] = \text{math.exp}(v[i])$

non-vectorized

`import numpy as np
u = np.exp(v) ←
np.log(v)
np.abs(v)
np.maximum(v, 0)
v ** 2 ← v/v`

Andrew Ng

- Whenever possible, remove explicit for loops
- Speed up data processing
- Implement by Python Numpy
- ex) `np.exp(v)`, `np.log(v)`, `np.abs(v)`, `np.square(v)`, `np.maximum(v, 0)`

What is Broadcasting?

- numpy가 다른 모양의 행렬을 처리하는 방법

General Principle

$$\begin{array}{c} (m, n) \\ \text{matrix} \\ \hline + \\ * \\ \diagup \\ (1, n) \\ (m, 1) \end{array} \rightsquigarrow \begin{array}{c} (m, n) \\ (m, n) \end{array}$$

$$\begin{array}{ccc} (m, 1) & + & \mathbb{R} \\ \left[\begin{smallmatrix} 1 \\ 2 \\ 3 \end{smallmatrix} \right] & + & 100 \\ [1 2 3] & + & 100 \end{array} = \begin{bmatrix} 101 \\ 102 \\ 103 \end{bmatrix} = [101 \quad 102 \quad 103]$$

Matlab/Octave: bsxfun

Broadcasting

- Python vs R

```
1 a = np.array([[1,4,7],  
2 [2,5,8],  
3 [3,6,9]])  
4 print(a)  
  
[[1 4 7]  
[2 5 8]  
[3 6 9]]  
  
1 b = np.array([[1],  
2 [2],  
3 [3]])  
4 print(b)  
  
[[1]  
[2]  
[3]]  
  
1 a+b  
  
array([[ 2,  5,  8],  
       [ 4,  7, 10],  
       [ 6,  9, 12]])
```

Python

```
> a <- matrix(data = 1:9, nrow = 3, ncol = 3)  
> a  
[,1] [,2] [,3]  
[1,] 1 4 7  
[2,] 2 5 8  
[3,] 3 6 9  
> b <- matrix(data = 1:3, nrow = 3, ncol = 1)  
> b  
[,1]  
[1,] 1  
[2,] 2  
[3,] 3  
> c <- a + b  
Error in a + b : non-conformable arrays
```

R

Broadcasting

- 각 음식별 탄수화물, 단백질, 지방 칼로리 함량 (%)

Broadcasting example

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes	
Carb	156.0	0.0	4.4	68.0	
Protein	1.2	104.0	52.0	8.0	= A
Fat	1.8	135.0	99.0	0.9	

- 백분율 계산 (matrix division)

```
1 # 총 칼로리 계산
2 # axis=0은 세로로 값을 더하라는 의미 (axis=1: 가로 합)
3 cal = A.sum(axis=0)
4 print(cal)
```

[59. 239. 155.4 76.9]

```
1 # 각 음식별 탄수화물, 단백질, 지방 칼로리 비율 (%) Matrix division
2 percentage = 100*A/cal.reshape(1,4) # python broadcasting
3 # M(3,4) / M(1,4)
4 percentage.round(2)
```

```
array([[94.92, 0., 2.83, 88.43],
       [2.03, 43.51, 33.46, 10.4],
       [3.05, 56.49, 63.71, 1.17]])
```

- 주의할 점: 아주 감지하기 힘든 버그나 또는 매우 이상하게 생긴 버그

Next Steps

- Neural Networks Overview
- Describe hidden units and hidden layers
- Use units with a non-linear activation function, such as tanh
- Implement forward and backward propagation
- Apply random initialization to your neural network
- Increase fluency in Deep Learning notations and Neural Network Representations
- Implement a 2-class classification neural network with a single hidden layer
- Compute the cross entropy loss

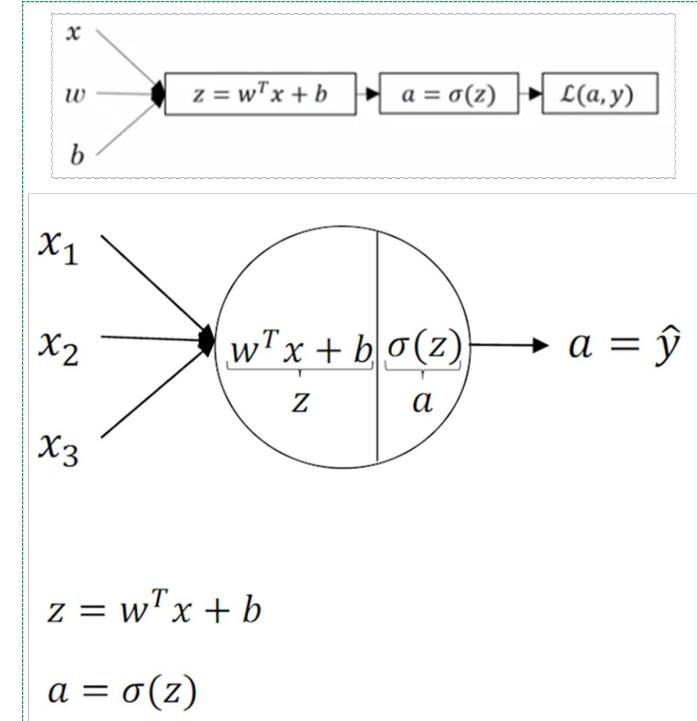
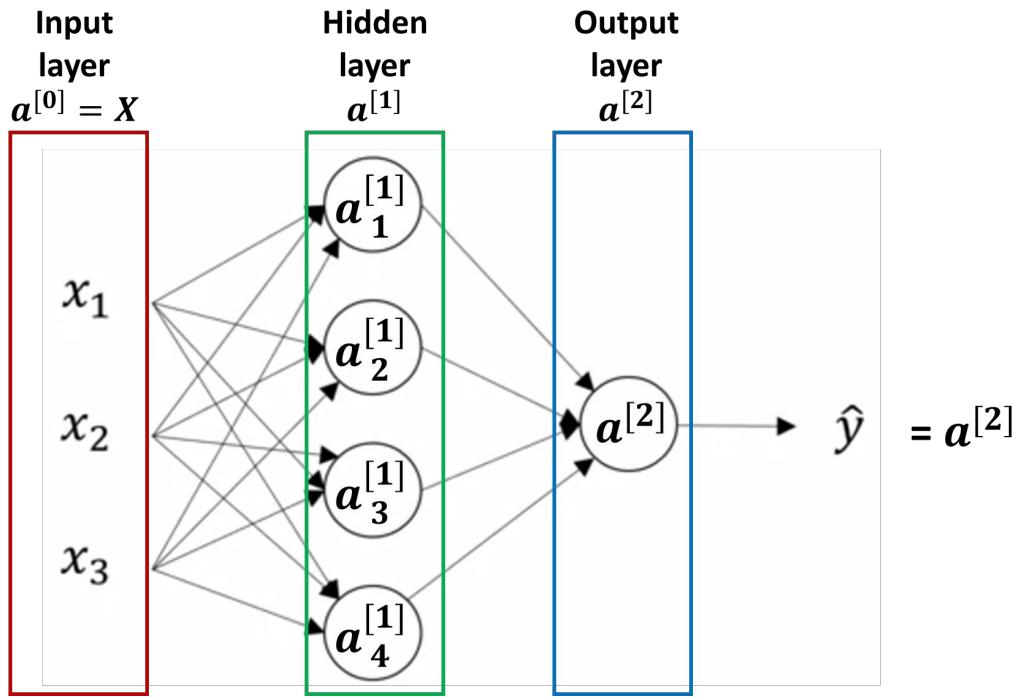
Table of Contents

- ❖ Our Progress
 - ❖ Neural Network Representation
 - ❖ What is Activation Function?
 - ❖ Random initialization
 - ❖ Next Steps
-

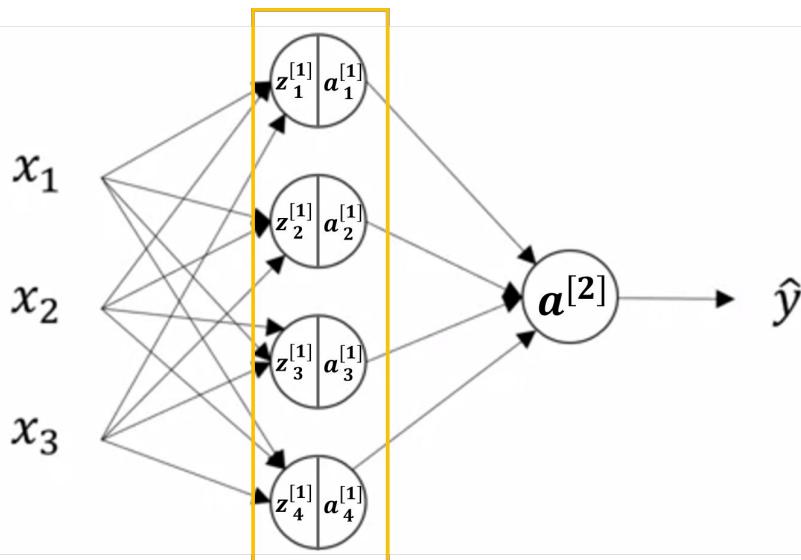
Our Progress (3 / 4)

Week3	<u>Neural Networks Overview</u>	4	O(10/14)
	Neural Network Representation	5	
	Computing a Neural Network's Output	9	
	Vectorizing Across Multiple Examples	9	
	Explanation for Vectorized Implementation	7	
	Activation Functions	10	
	Why do you need Non-Linear Activation Functions?	5	
	Derivatives of Activation Functions	7	
	Gradient Descent for Neural Networks	9	
	Backpropagation Intuition (Optional)	15	
	Random Initialization	7	O(10/28)

Neural Network Representation



Neural Network Representation



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

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

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

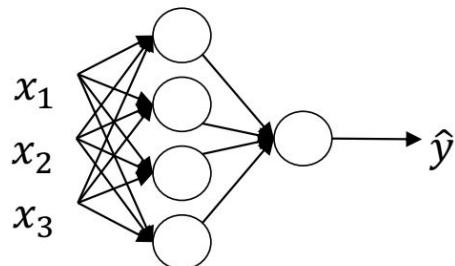
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

Vectorizing

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

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \sigma(z_4^{[1]}) \end{bmatrix}$$

Vectorizing across multiple examples



$x \rightarrow a^{[2]} = y$
 $x^{(1)} \rightarrow a^{[2](1)} = y^{(1)}$
 $x^{(2)} \rightarrow a^{2} = y^{(2)}$
 \vdots
 $x^{(n)} \rightarrow a^{[2](n)} = y^{(n)}$

$a^{[2](i)}$ example i
layer 2

$$\left\{ \begin{array}{l} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{array} \right.$$

for $i = 1 \dots n$,

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

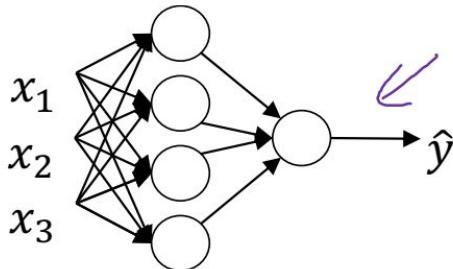
Andrew Ng

Explanation for Vectorized Implementation

$$z^{(1)(1)} = w^{(1)} x^{(1)} + b^{(1)}, \quad z^{(1)(2)} = w^{(1)} x^{(2)} + b^{(1)}, \quad z^{(1)(3)} = w^{(1)} x^{(3)} + b^{(1)}$$
$$w^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad w^{(1)} x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad w^{(1)} x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad w^{(1)} x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$
$$z^{(1)} = w^{(1)} X + b^{(1)}$$
$$X = \begin{bmatrix} 1 & 1 & 1 \\ x^{(1)} & x^{(2)} & x^{(3)} \\ 1 & 1 & 1 \end{bmatrix}$$
$$w^{(1)} x^{(1)} = z^{(1)(1)}$$
$$w^{(1)} x^{(2)} = z^{(1)(2)}$$
$$w^{(1)} x^{(3)} = z^{(1)(3)}$$
$$z^{(1)(1)} + b^{(1)} = z^{(1)(1)}$$
$$z^{(1)(2)} + b^{(1)} = z^{(1)(2)}$$
$$z^{(1)(3)} + b^{(1)} = z^{(1)(3)}$$
$$z^{(1)} = \begin{bmatrix} z^{(1)(1)} \\ z^{(1)(2)} \\ z^{(1)(3)} \end{bmatrix}$$

Andrew Ng

Recap for Vectorized Implementation



$$X = \begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

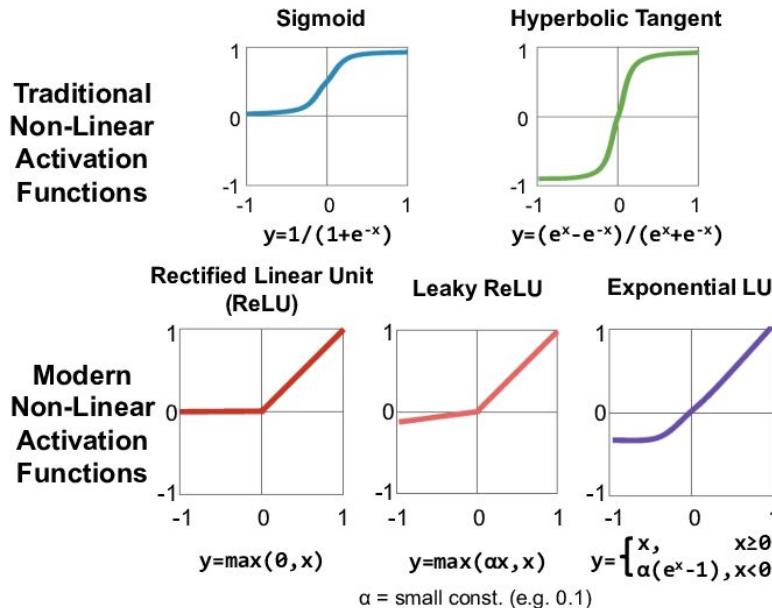
$$A^{[1]} = \begin{bmatrix} | & | & | & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

for $i = 1$ to m
 $\quad z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$
 $\quad \rightarrow a^{[1](i)} = \sigma(z^{[1](i)})$
 $\quad \rightarrow z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$
 $\quad \rightarrow a^{[2](i)} = \sigma(z^{[2](i)})$

$Z^{[1]} = W^{[1]}X + b^{[1]} \leftarrow w^{[1]T} A^{[1]} + b^{[1]}$
 $A^{[1]} = \sigma(Z^{[1]})$
 $Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
 $A^{[2]} = \sigma(Z^{[2]})$

Andrew Ng =

What is Activation Function?



Activation Function(활성 함수)란?

- decides whether a neuron should be activated or not
- decides whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations
- to derive output from a set of input values fed to a node (or a layer)
- Many types of activation functions (Sigmoid, tanh, ReLU, Leaky ReLU..)
- ReLU is the most used activation function nowadays (gradient vanishing issues of sigmoid and tanh functions)

What is Activation Function?

Why do we need non-linear activation functions?

Activation function이 linear라면 hidden layer를 쓰는 이유가 사라지며, gradient descent를 쓰지 못하게 됨

ex) if activation function is $h(x)=cx$ and $y=h(h(h(x)))$ which is $y=c^3x$

$c^3=a \rightarrow y=ax$: return to linear function, one-layer network 와 동일하게 됨

Gradient Descent for Neural Networks

1. Compute Loss function & Cost function with computed predictions
2. Compute derivative of cost function with respect to each parameter
3. Update parameters
4. Repeat this until they are converging.

Parameters: $w^{(0)}, b^{(0)}, w^{(1)}, b^{(1)}, \dots, w^{(L)}, b^{(L)}$ $n_x = n^{(0)}, n^{(1)}, \dots, n^{(L)} = 1$

Cost function: $J(w^{(0)}, b^{(0)}, w^{(1)}, b^{(1)}, \dots, w^{(L)}, b^{(L)}) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}_i, y_i)$

Gradient descent:

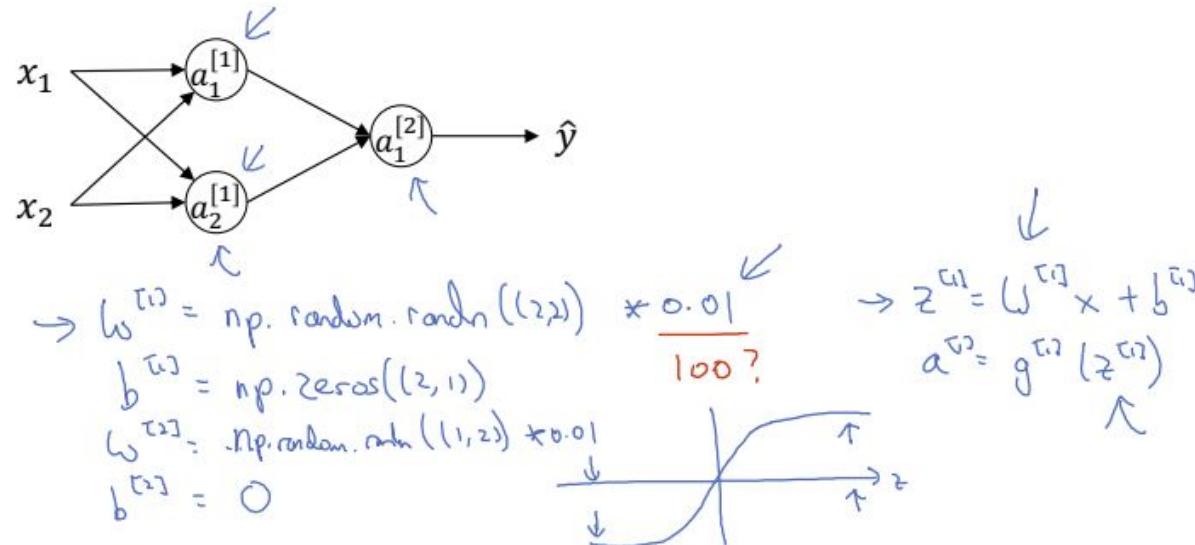
Repeat {

- Compute predict $(\hat{y}^{(i)}, i=1 \dots m)$
- $\frac{\partial J}{\partial w^{(l)}} = \frac{\partial J}{\partial w^{(l)}} = \frac{\partial J}{\partial b^{(l)}} = \dots$
- $w^{(l)} := w^{(l)} - \alpha \frac{\partial J}{\partial w^{(l)}}$
- $b^{(l)} := b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$
- \vdots
- $b^{(L)} := \dots$
- $b^{(0)} := \dots$

Andrew Ng

Random initialization

- What happens if you initialize weights to zero?
 - If a weight for w is initialized to zero, gradient descent won't work.
 - Solution: Initialize weights randomly
 - Usually use very small number, such as 0.01.



Next Steps

- Week 4 마무리 & review

Week4	Deep L-layer Neural Network	5
	Forward Propagation in a Deep Network	7
	Getting your Matrix Dimensions Right	11
	Why Deep Representations?	10
	Building Blocks of Deep Neural Networks	8
	Forward and Backward Propagation	10
	Parameters vs Hyperparameters	7
	What does this have to do with the brain?	3

- 프로그래밍 과제 진행

Table of Contents

- ❖ Motivation
 - ❖ Week 1
 - ❖ Week 2
 - ❖ Week 3
 - ❖ Week 4
 - ❖ Summary
-

Motivation

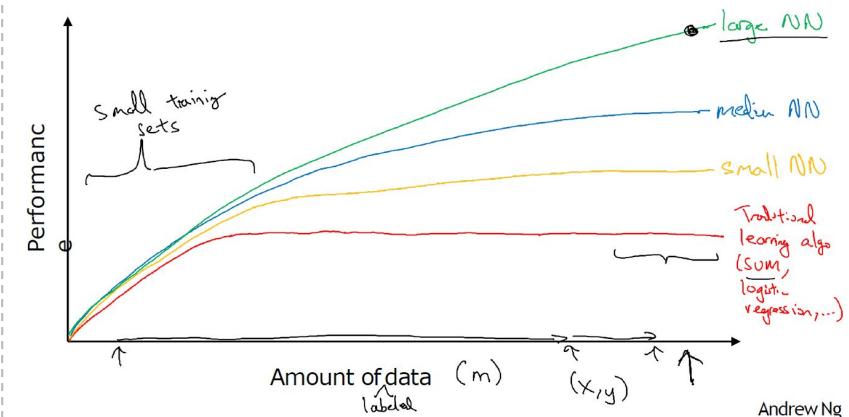
- Introduction to neural network and deep learning
- Learn fundamentals of machine learning
- Hone Python programming skills

Week 1

Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving Custom/Hybrid

Scale drives deep learning progress

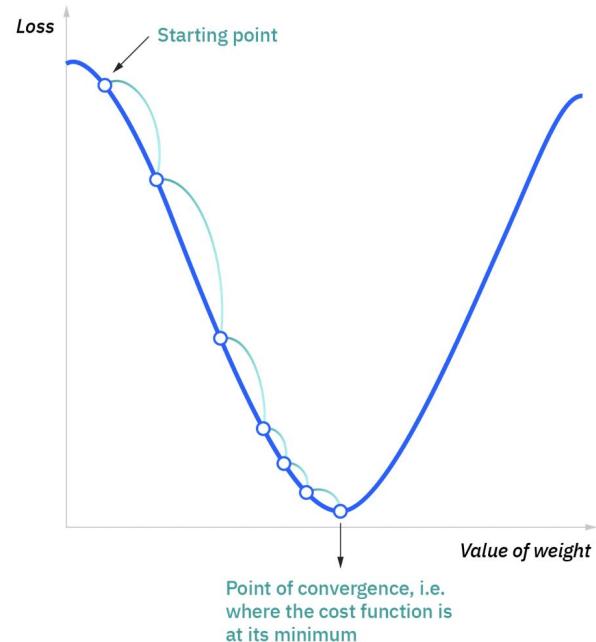


Andrew Ng

Week 2

Gradient Descent

- Optimization algorithm to minimize cost function (convex)
- Take repeated steps in the opposite direction of the gradient of the function at the current point
- Repeat $x \leftarrow x - \alpha f'(x)$
(α : learning rate, small positive number)
- Justification: The gradient points in the direction of the steepest ascent



Week 2

Vectorization

- Type of parallel processing—used to speed up the Python code without using loop
- Whenever possible, remove explicit for loops
- Minimizes the running & execution time of code
- Use built-in functions in Numpy
ex) `np.exp(v)`, `np.log(v)`, `np.abs(v)`, `np.square(v)`,
`np.maximum(v,0)`

What is vectorization?

$$z = \underline{w^T x + b}$$

$$w = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

$$w \in \mathbb{R}^{n_x} \quad x \in \mathbb{R}^{n_x}$$

Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i]*x[i]
```

$z += b$

Vectorized

$$z = \underbrace{\text{np.dot}(w, x)}_{w^T x} + b$$

\rightarrow GPU } SIMD - single instruction
 \rightarrow CPU } multiple data.

Andrew Ng

Week 2

Broadcasting in Python

- How numpy treats arrays with different dimension during arithmetic operations
- Unique function in Python numpy

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0]) Two arrays in same shape
>>> a * b
array([2.,  4.,  6.])
```

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = 2.0
>>> a * b
array([2.,  4.,  6.])
```

Two arrays in different shapes

```
> a <- matrix(data = 1:9, nrow = 3, ncol = 3)
> a
 [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> b <- matrix(data = 1:3, nrow = 3, ncol = 1)
> b
 [,1]
[1,]    1
[2,]    2
[3,]    3
> c <- a + b
Error in a + b : non-conformable arrays
```

Week 3

Explanation for Vectorized Implementation

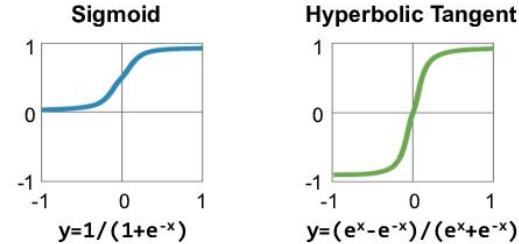
$$\begin{aligned} z^{(1)(1)} &= w^{(1)} x^{(1)} + b^{(1)}, & z^{(1)(2)} &= w^{(1)} x^{(2)} + b^{(1)}, & z^{(1)(3)} &= w^{(1)} x^{(3)} + b^{(1)} \\ w^{(1)} &= \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix} & w^{(1)} x^{(1)} &= \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} & w^{(1)} x^{(2)} &= \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} & w^{(1)} x^{(3)} &= \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} \\ w^{(1)} \begin{bmatrix} 1 & X^{(1)} & X^{(2)} & X^{(3)} \dots \end{bmatrix} &= \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} & = \begin{bmatrix} 1 & 1 & 1 & \dots \\ z^{(1)(1)} & z^{(1)(2)} & z^{(1)(3)} & \dots \end{bmatrix} &= z^{(1)} \\ z^{(1)} &= w^{(1)} X + b^{(1)} & w^{(1)} x^{(1)} &= z^{(1)(1)} & + b^{(1)} & & + b^{(1)} & & + b^{(1)} \end{aligned}$$

Andrew Ng

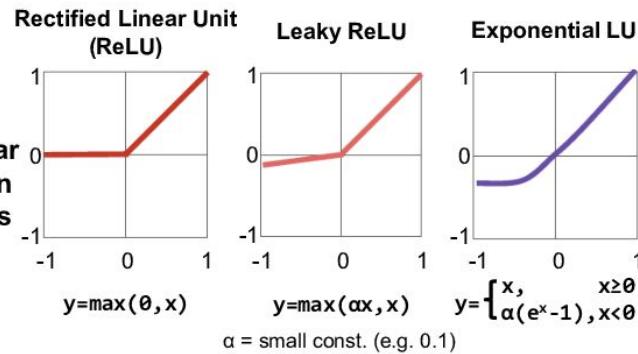
Week 3

Non-linear Activation Function

Traditional Non-Linear Activation Functions



Modern Non-Linear Activation Functions

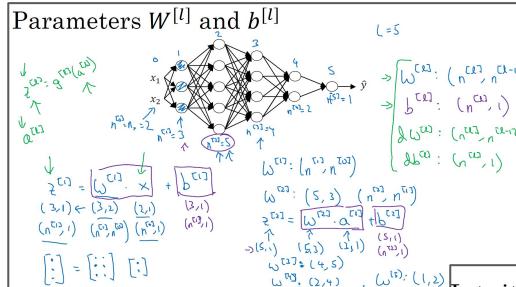


Activation Function(활성함수)란?

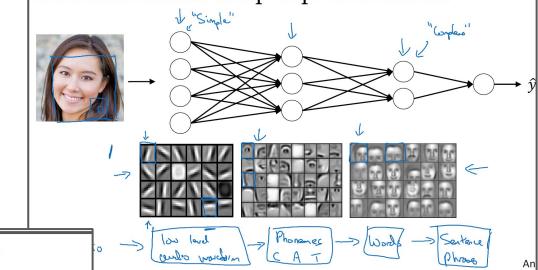
- decides whether a neuron should be activated or not
- decides whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations
- to derive output from a set of input values fed to a node (or a layer)
- Many types of activation functions (Sigmoid, tanh, ReLU, Leaky ReLU..)
- ReLU is the most used activation function nowadays (gradient vanishing issues of sigmoid and tanh functions)

Week 4

- Forward propagation in a Deep Network
- Getting your matrix dimensions right
- Why deep representation?
- Parameters vs Hyperparameters
- What does this have to do with the brain?



Intuition about deep representation



What are hyperparameters?

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \dots$

Hyperparameters:

- Learning rate α $\propto \frac{1}{\# \text{iterations}}$
- # hidden layers L
- # hidden units $n^{[1]}, n^{[2]}, \dots$
- Choice of activation function

Optimizer: Momentum, minibatch size, regularization, ...

Summary

- Familiarized with technological trends
- Built & trained deep neural networks
- Implemented neural network algorithms

Thank you