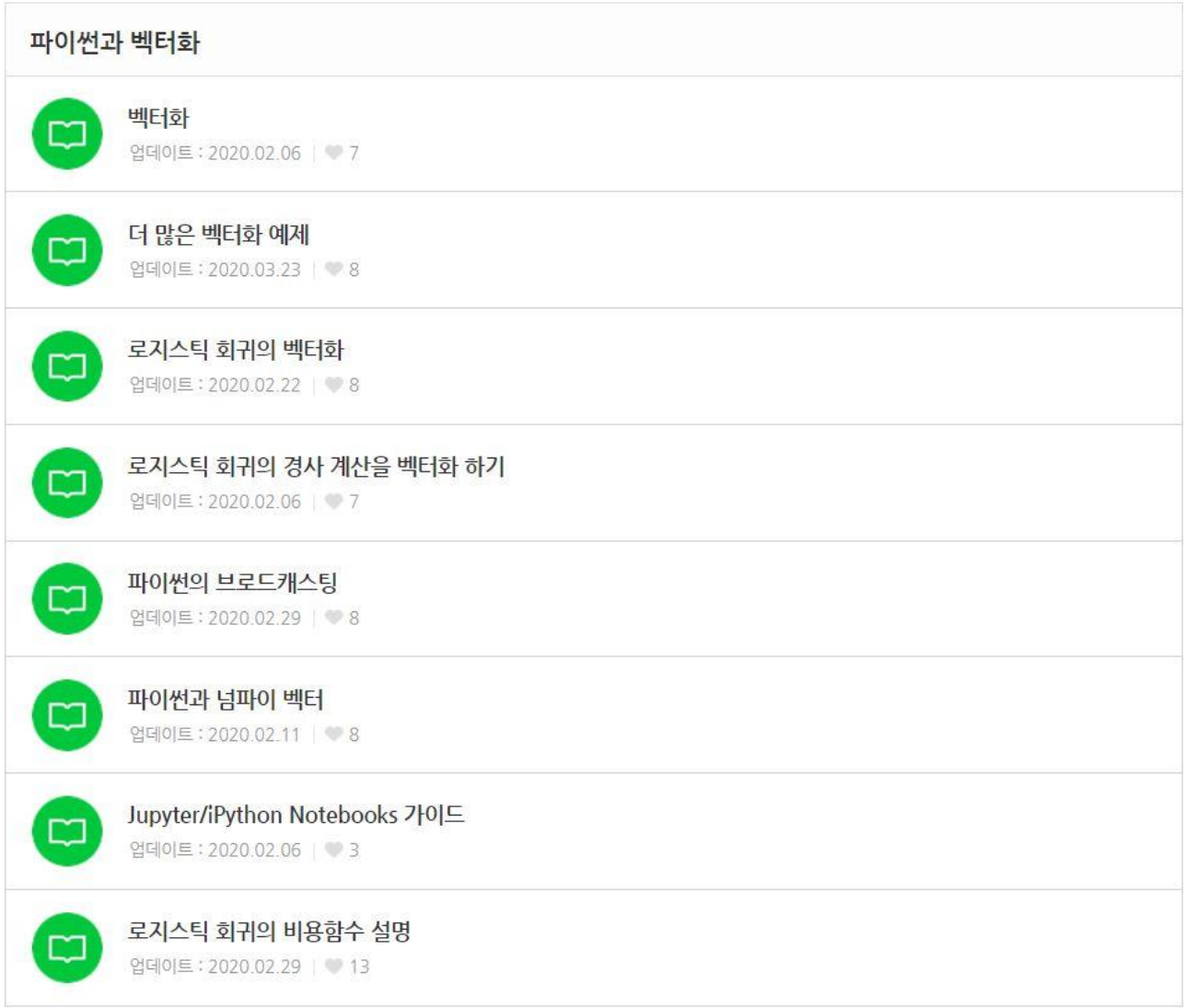
# Deep Learning Study

2020/04/28

ByungChan Lee

# 강의 목록


파이썬과 벡터화
벡터화
업데이트 : 2020.02.06 | 7
더 많은 벡터화 예제
업데이트 : 2020.03.23 | 8
로지스틱 회귀의 벡터화
업데이트 : 2020.02.22 | 8
로지스틱 회귀의 경사 계산을 벡터화 하기
업데이트 : 2020.02.06 | 7
파이썬의 브로드캐스팅
업데이트 : 2020.02.29 | 8
파이썬과 넘파이 벡터
업데이트 : 2020.02.11 | 8
Jupyter/iPython Notebooks 가이드
업데이트 : 2020.02.06 | 3
로지스틱 회귀의 비용함수 설명
업데이트 : 2020.02.29 | 13

# Jupyter/iPython Notebooks

- https://www.edwith.org/deeplearningai1/lecture/34817/


Quick Tour of Jupyter/iPython Notebooks 원본보기

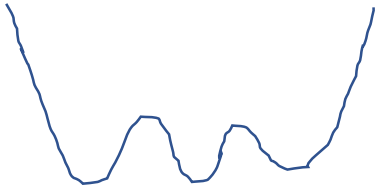

[WINDOWS]쥬피터노트북설치하기 PDF

윈도우 운영체제 Jupyter notebook 튜토리얼입니다.


[MAC]쥬피터노트북설치하기 PDF

MAC 운영체제 Jupyter notebook 튜토리얼입니다.

# Review



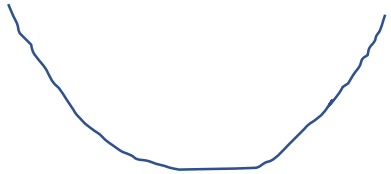
- Loss function

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

→ 경사하강법을 사용할 때, local optimum에 갇혀서 global optimum을 찾지 못할 수 있다.

→ Cross entropy

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



- Cost fucntion

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

# Logistic Regression : Cost function

- Interpret $\hat{y} = P(y = 1|x)$

  if $y = 1$, $P(y|x) = \hat{y}$

  if $y = 0$, $P(y|x) = 1 - \hat{y}$

- $P(y|x) = \hat{y}^{y}(1 - \hat{y})^{1-y}$

  $\log P(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \equiv -\mathcal{L}(\hat{y}, y)$ maximum P → minimum L

- $P(labels\ in\ training\ set) = \prod_{i=1}^{m} P\left(y^{(i)}|x^{(i)}\right)$
- $\log P = -\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$

- $\mathrm{J}(w, b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$ average of loss functions

# Vectorization

$$z = w^T x + b \qquad x = \begin{bmatrix} .. \\ .. \\ .. \\ .. \end{bmatrix} \qquad w = \begin{bmatrix} .. \\ .. \\ .. \\ .. \end{bmatrix}$$

- for loop

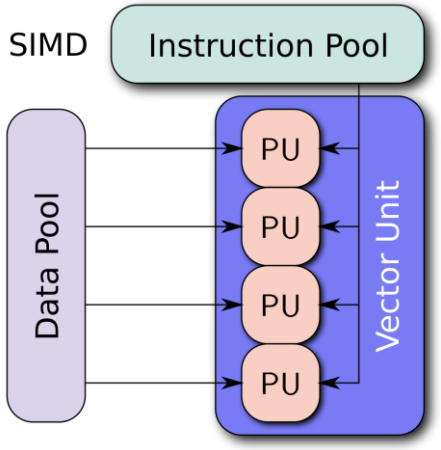
```
for i in range(1000000):
    c += w[i]*x[i]
```

- Vectorization

```
import numpy as np
c = np.dot(w.T, x)
```

- for loop을 최소화하고 벡터와 함수를 적극적으로 활용하면 연산속도가 훨씬 빨라진다.

- SIMD; Single instruction, multiple data



# Vectorization

In [12]:
```
1  import numpy as np
2  import time
3
4  a = np.random.rand(1000000)
5  b = np.random.rand(1000000)
6
7  tic = time.time()
8  c = np.dot(a,b)
9  toc = time.time()
10
11 print(c)
12 print("Vectorized Version : " + str(1000*(toc-tic)) + "ms")
13
14 c = 0
15 tic = time.time()
16 for i in range(1000000):
17     c += a[i]*b[i]
18 toc = time.time()
19
20 print(c)
21 print("for-loop   Version : " + str(1000*(toc-tic)) + "ms")
```

```
249965.8632635132
Vectorized Version : 0.9999275207519531ms
249965.86326351386
for-loop   Version : 660.2320671081543ms
```

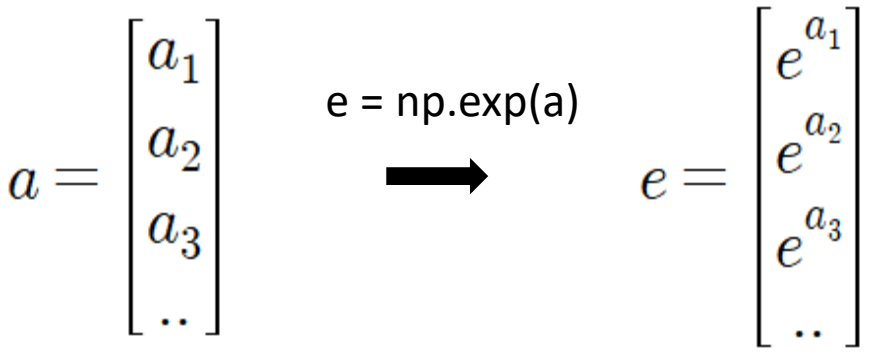
- 660 times faster
- 1 min vs 11 hours

In [ ]:
```
1
```

# Vectorization

Numpy 함수

- exp
- log
- abs
- maximum
- **
- /
- zeros




* tip : 구현하고 싶은 연산이 for 문을 필요로 하면, 내장 함수를 먼저 살펴봐라

# Vectorization

$$X = \begin{bmatrix} | & | \\ x_1 & x_2 .. \\ | & | \end{bmatrix} \quad n \times m \qquad Z = [z_1, z_2, ..]$$

1

- Z = np.dot(w.T , X) + b
- A = σ(z)

2

- dw = np.zeros( (n, 1) )
- dw += x[i]*dz[i]
- dw /= m

## Logistic regression derivatives

```
J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to m:
    z(i) = w^T x(i) + b
    a(i) = σ(z(i))
    J += -[y(i) log ŷ(i) + (1 - y(i)) log(1 - ŷ(i))]
    dz(i) = a(i)(1 - a(i))
    dw1 += x1(i) dz(i)
    dw2 += x2(i) dz(i)
    db += dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m, db = db/m
```

1 → `for i = 1 to m:` block

2 → `dw1 += x1(i) dz(i)`, `dw2 += x2(i) dz(i)`

# Vectorization

$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$

for i = 1 to m:

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m$

$db = db/m$

$Z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dZ^T$

$db = \frac{1}{m} np.sum(dZ)$

$w := w - \alpha dw$

$b := b - \alpha db$

- 경사하강법 전체를 for loop 없이 구현 가능

# Broadcasting

In [27]:
```
a = np.array([1, 2, 3])
b = 100
c = a+b
print(c)
```

```
[101 102 103]
```

In [30]:
```
a = np.array([[1, 2, 3],
              [4, 5, 6]])
b = np.array([[100, 200, 300]])
c = a+b
print(c)
```

```
[[101 202 303]
 [104 205 306]]
```

In [34]:
```
a = np.array([[100, 200, 300],
              [450, 520, 664]])
b = np.array([[100, 200, 300]])
c = a/b
print(c)
```

```
[[1.         1.         1.        ]
 [4.5        2.6        2.21333333]]
```

- 코드를 짧고 빠르게 만들어 준다.

# Reshape

In [49]:

```
a = np.array([[100, 200, 300],
              [450, 520, 664]])
b = a.reshape(2, 3)
c = a.reshape(6)
d = a.reshape(3, 2)
print(b)
print(c)
print(d)
```

```
[[100 200 300]
 [450 520 664]]
[100 200 300 450 520 664]
[[100 200]
 [300 450]
 [520 664]]
```

- 차원이 불분명할 때 reshape를 사용하여 확실히 의도하는 차원으로 만들어 줄 수 있다.
- reshape 함수를 호출할 때 상수시간이 소요되기 때문에 전체 속도에 큰 영향을 미치지 않는다.

# Broadcasting

In [51]:
```
a = np.array([[10],
              [20],
              [30]])
b = np.array([[1, 2, 3]])
c = a+b
print(c)
```

```
[[11 12 13]
 [21 22 23]
 [31 32 33]]
```

- Numpy의 유연성은 장점이자 단점이 될 수 있다.
- 브로드캐스팅의 작동 원리를 잘 모르면 원인을 알기 힘든 버그가 발생하기 쉽다.

# Broadcasting

In [52]:
```
a = np.random.rand(5)
print(a)
print(a.T)
b = np.random.rand(5)
c = np.dot(a.T,b)
print(c)
```

```
[0.94279148 0.63942625 0.73003016 0.12247464 0.6993223 ]
[0.94279148 0.63942625 0.73003016 0.12247464 0.6993223 ]
0.9119557383994352
```

In [54]:
```
a = np.random.rand(1,5)
print(a)
print(a.T)
b = np.random.rand(1,5)
c = np.dot(a.T,b)
print(c)
```

```
[[0.95640169 0.88210857 0.69889984 0.26178702 0.20102956]]
[[0.95640169]
 [0.88210857]
 [0.69889984]
 [0.26178702]
 [0.20102956]]
[[0.84794601 0.24909704 0.67584544 0.74139408 0.00835518]
 [0.78207771 0.22974723 0.62334588 0.68380272 0.00770615]
 [0.61964479 0.18203009 0.49388063 0.54178094 0.00610563]
 [0.23210045 0.06818304 0.18499294 0.20293497 0.00228699]
 [0.17823286 0.05235862 0.14205842 0.15583632 0.00175621]]
```

# Broadcasting

In [61]:

```
a = np.random.rand(5)
print(a.shape)
b = np.random.rand(1,5)
print(b.shape)

a = a.reshape(1,5)
assert(b.shape == (1,5))
print(a.shape)
```

```
(5,)
(1, 5)
(1, 5)
```

- a = np.random.rand(5)  : do not use
- a = np.random.rand(1,5)  : a row vector
- a = np.random.rand(5,1)  : a column vector

assert와 reshape를 적극적으로 활용하라

- assert(a.shape == (1,5))
- a = a.reshape(1,5)

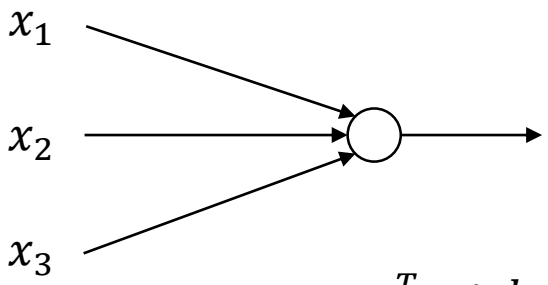
# 강의 목록

얕은 신경망 네트워크

- 신경망 네트워크 개요
  업데이트 : 2020.02.06 | ♥ 8
- 신경망 네트워크의 구성 알아보기
  업데이트 : 2020.02.06 | ♥ 5
- 신경망 네트워크 출력의 계산
  업데이트 : 2020.02.06 | ♥ 9
- 많은 샘플에 대한 벡터화
  업데이트 : 2020.03.01 | ♥ 6
- 벡터화 구현에 대한 설명
  업데이트 : 2020.03.01 | ♥ 9
- 활성화 함수
  업데이트 : 2020.02.06 | ♥ 7
- 왜 비선형 활성화 함수를 써야할까요?
  업데이트 : 2020.03.02 | ♥ 6
- 활성화 함수의 미분
  업데이트 : 2020.03.05 | ♥ 6
- 신경망 네트워크와 경사 하강법
  업데이트 : 2020.03.07 | ♥ 8
- 역전파에대한 이해
  업데이트 : 2020.03.09 | ♥ 9
- 랜덤 초기화
  업데이트 : 2020.02.29 | ♥ 7

# Neural Network

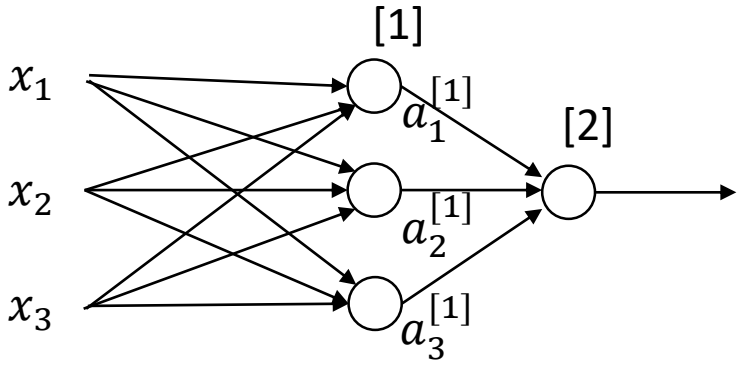
- Logistic regression




$$z = w^T x + b \rightarrow a = \sigma(z) \rightarrow \mathcal{L}(a, y)$$

da, dz

- Neural network




[ i ] : i'th layer

( i ) : i'th training example

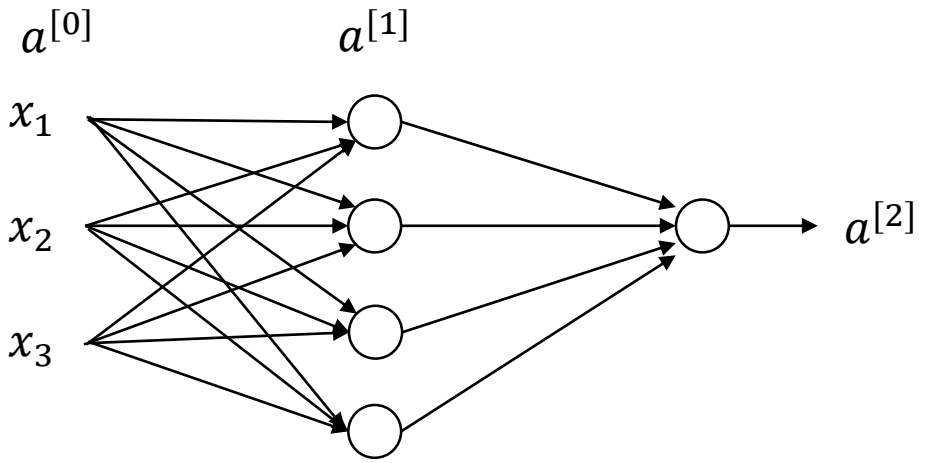
layer → $z_i^{[1]} = W_i^{[1]} x_i + b_i^{[1]} \rightarrow a_i^{[1]} = \sigma\left(z_i^{[1]}\right) \rightarrow z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} \rightarrow a^{[2]} \rightarrow \mathcal{L}\left(a^{[2]}, y\right)$

node →

$da^{[1]},\ dz^{[2]}$ …

# Neural Network

- 2 Layer Neural Network




input layer hidden layer output layer

- Input layer는 세지 않는다. (convention)
- hidden layer의 값은 알 수 없다. training example은 (x, y)로 주어진다.
- a : activation ($a^{[0]} = x$)

# Neural Network

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

$$z^{[i]} = \begin{bmatrix} - & w_1^{[i]\,T} & - \\ - & w_2^{[i]\,T} & - \\ & \vdots & \end{bmatrix} \begin{bmatrix} a_1^{[i-1]} \\ a_2^{[i-1]} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^{[i]} \\ b_2^{[i]} \\ \vdots \end{bmatrix}$$

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

$$n_i \times n_{i-1}$$

- 2-Layer NN, given input x

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

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

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

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

← Vectorized

# Neural Network

- Vectorizing across Multiple examples

a single example

$$z^{[i]} = \begin{bmatrix} - & w_1^{[i]T} & - \\ - & w_2^{[i]T} & - \\ & \vdots & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[i]} \\ b_2^{[i]} \\ \vdots \end{bmatrix}$$

multiple examples

for i in range(m)

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

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

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

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

Vectorizing

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

$$Z^{[i]} = \begin{bmatrix} | & | & \\ z^{[i](1)} & z^{[i](2)} & .. \\ | & | & \end{bmatrix} \qquad A^{[i]} = \begin{bmatrix} | & & | \\ a^{[i](1)} & .. & a^{[i](m)} \\ | & & | \end{bmatrix}$$

# Neural Network

- 2-Layer NN, given input X

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

# Back up