AI 프로그래밍

- 2024

5주차

오늘 수업 순서

- Numpy 복습
- Numpy 1,2차 배열
- Numpy new axis, reshape, scikit learn input,
- 농어 예제로 error 수정
- 경사 하강법
- 선형회귀 code 경사 하강법

Numpy


```
>>> import numpy as np

>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])

>>> a[0]
1
```

참고자료 : 딥러닝 express

Numpy


```
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> b[0][2]
3
```

b[1,2] ?
b[2,0] ?

참고자료 : 딥러닝 express

Numpy


```
>>> import numpy as np
>>> a = np.array([[ 0, 1, 2],
          [ 3, 4, 5],
          [ 6, 7, 8]])
>>> a.shape                    # 배열의 형상
(3, 3)
>>> a.ndim                     # 배열의 차원 개수
2
>>> a.dtype                    # 요소의 자료형
dtype('int32')
>>> a.itemsize     # 요소 한개의 크기
4                              # 오타
>>> a.size                     # 전체 요소의 개수
9
```

```
>>> import numpy as np
>>> a = np.array([[ 0, 1, 2],
                  [ 3, 4, 5],
                  [ 6, 7, 8]])
```

참고자료 : 딥러닝 express

Numpy


```
>>> np.zeros( (3, 4) )                    # (3, 4)는 배열의 형상(행의 개수, 열의 개수)
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])

>>> np.ones((3, 4))
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])

>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
>>> import numpy as np
>>> a =np.zeros( (3, 4) )
      a.shape ?
      a.size?
```

참고자료 : 딥러닝 express

Numpy-arange


```
>>> np.arange(5)
array([0, 1, 2, 3, 4])

>>> np.arange(1, 6)
array([1, 2, 3, 4, 5])

>>> np.arange(1, 10, 2)
array([1, 3, 5, 7, 9])
```

>>> np.arange(1, 11 2) ?

참고자료 : 딥러닝 express

Numpy-배열 합치기


```
>>> x = np.array([[1, 2], [3, 4]])
>>> y = np.array([[5, 6], [7, 8]])

>>> np.concatenate((x, y), axis=1)
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

```
>>> np.vstack((x, y))
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

참고자료 : 딥러닝 express

Numpy-배열 합치기


```
>>> x = np.array([[1, 2], [3, 4]])
>>> y = np.array([[5, 6], [7, 8]])

>>> np.concatenate((x, y), axis=1)
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

```
>>> np.vstack((x, y))
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

```
np.hstack((x, y))
```

```
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

참고자료 : 딥러닝 express

Numpy


```
>>> a = np.arange(12)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

# a에 대하여 reshape(3, 4)를 호출하면 1차원 배열이 2차원 배열로 바뀌게 된다.
>>> a.reshape(3, 4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>> a.reshape(6, -1)
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

참고자료 : 딥러닝 express

Numpy


```
>>> array = np.arrange(30).reshape(-1, 10)
>>> array
Array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])

>>> arr1, arr2 = np.split(array, [3], axis=1)
>>> arr1
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22]])
>>> arr2
array([[ 3,  4,  5,  6,  7,  8,  9],
       [13, 14, 15, 16, 17, 18, 19],
       [23, 24, 25, 26, 27, 28, 29]])
```

참고자료 : 딥러닝 express

Numpy


```
a = np.array([1, 2, 3, 4, 5, 6])

a.shape

(6,)

a1 = a[np.newaxis, :]

a1

array([[1, 2, 3, 4, 5, 6]])

a1.shape

(1, 6)
```

```
a2 = a[:, np.newaxis]

a2.shape

(6, 1)

a2
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6]])
```

참고자료 : 딥러닝 express

Numpy-인덱싱과 슬라이싱


```
>>> ages = np.array([18, 19, 25, 30, 28])
>>> ages[1:3] # 인덱스 1에서 인덱스 2까지
array([19, 25])
>>> ages[:2] # 인덱스 0에서 인덱스 1까지
array([18, 19])

# 논리적인 인덱싱(logical indexing)
>>> y = ages > 20
>>> y
array([False, False,  True,  True,  True])
>>> ages[ ages > 20 ]
array([25, 30, 28])
```

참고자료 : 딥러닝 express

조건에 맞는 인덱스 찾기


```
import numpy as np
a=np.array([3,6,0,3,2,7,3,0,0,2])
print(np.where(a!=0))
```

```
(array([0, 1, 3, 4, 5, 6, 9]),)
```

slicing

배열[start : end : step]
-start는 시작 인덱스, end는 끝 인덱스, step은 증가폭/감소폭이다.
-마지막 인덱스 : -1

slicing


```
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

print(a[0:3]) =>   [0 1 2]
print(a[3:6]) =>[3 4 5]
print(a[1: 8: 2]) => [1 3 5 7]
print(a[ : : 2]) => [ 0 2 4 6 8 10]
print(a[3 : 8: 2]) => [3 5 7]
print(a[3: -1: 2])  => [3 5 7 9]
print(a[ : :-1])  =>[10 9 8 7 6 5 4 3 2 1 0]
```

Numpy-2차원 배열의 인덱싱


```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> a[0, 2]
3
>>> a[0, 0] = 12
>>> a
array([[12,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9]])
```

참고자료 : 딥러닝 express

Numpy-2차원 배열의 슬라이싱


```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> a[0:2, 1:3]
array([[2, 3],
       [5, 6]])
```

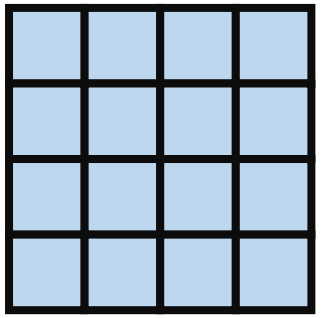
참고자료 : 딥러닝 express

Numpy

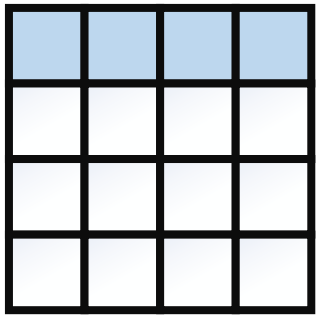

```
a[::2, ::2]
```

참고자료 : 딥러닝 express

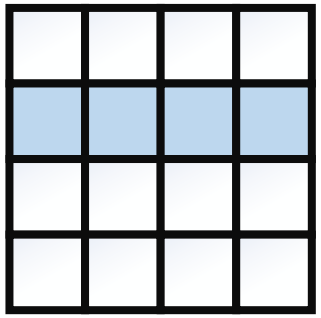
a



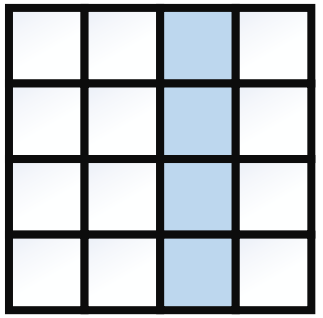
a[0]



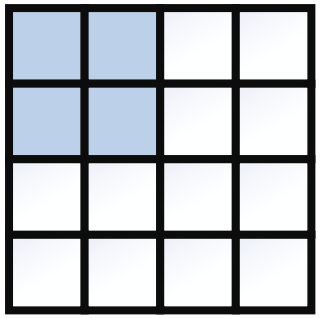
a[1,:]



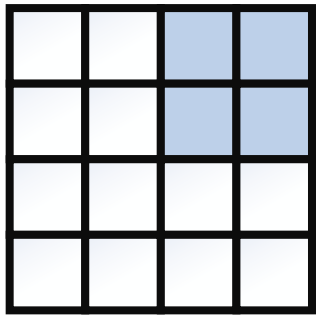
a[:,2]



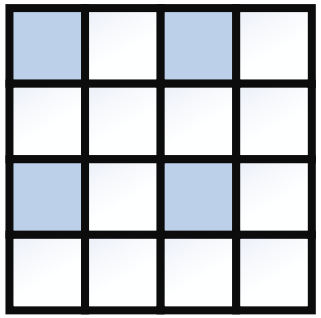
a[0:2,0:2]



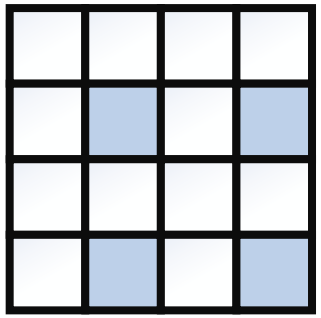
a[0:2,2:4]



a[::2,::2]



a[1::2,1::2]



Numpy


```
a[::2, ::2]
```

Numpy view 와 deep copy

view

```
>>a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>b=a[0,:]
```

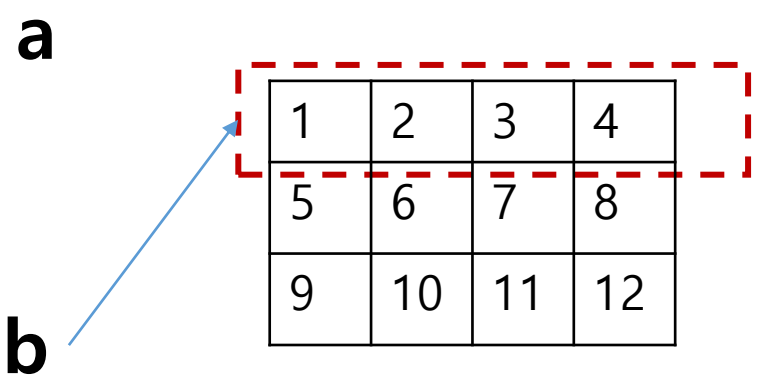
a

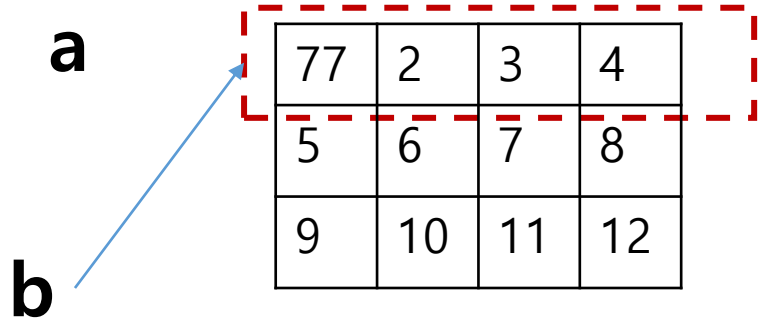
1	2	3	4
5	6	7	8
9	10	11	12

Numpy

view

```
>>a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>b=a[0,:]
>>b
array([1,2,3,4])
b[0]=77
b
array([77,2,3,4])
a
array ([[77,2,3,4],
        [5,6,7,8],
        [9,10,11,12]])
```





Numpy

deep copy

```
>>a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>b2=a.copy()
>>b2[0][0]=55
>>a[0][0] ?
```

a

1	2	3	4
5	6	7	8
9	10	11	12

b2

1	2	3	4
5	6	7	8
9	10	11	12

Numpy-배열의 연산


```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]])
>>> arr2 = np.array([[1, 1], [1, 1], [1, 1]])
>>> result = arr1 + arr2                    # 넘파이 배열에 + 연산이 적용된다.
>>> result
array([[2, 3],
       [4, 5],
       [6, 7]])
```

Numpy-배열의 연산


```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]])
>>> arr2 = np.array([[1, 1], [1, 1], [1, 1]])
>>> result = arr1 + arr2                    # 넘파이 배열에 + 연산이 적용된다.
>>> result
array([[2, 3],
       [4, 5],
       [6, 7]])
```

참고자료 : 딥러닝 express

Numpy


```
>>> A = np.array([0, 1, 2, 3])
>>> 10 * np.sin(A)
array([0.        , 8.41470985, 9.09297427, 1.41120008])
```

참고자료 : 딥러닝 express

Numpy-특정한 행과 열을 이용한 연산


```
>>> scores = np.array([[99, 93, 60], [98, 82, 93],
  ...:                 [93, 65, 81], [78, 82, 81]])
>>> scores.mean(axis=0)
array([92.  , 80.5 , 78.75])
```

참고자료 : 딥러닝 express

참고자료 : 딥러닝 express

Numpy-난수 생성


```
>>> np.random.seed(100)
>>> np.random.rand(5)
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])

>>> np.random.rand(5, 3)
array([[0.12156912, 0.67074908, 0.82585276],
       [0.13670659, 0.57509333, 0.89132195],
       [0.20920212, 0.18532822, 0.10837689],
       [0.21969749, 0.97862378, 0.81168315],
       [0.17194101, 0.81622475, 0.27407375]])
```

참고자료 : 딥러닝 express

Numpy


```
>>> np.random.randn(5)
array([ 0.78148842, -0.65438103,  0.04117247, -0.20191691, -0.87081315])

>>> np.random.randn(5, 4)
array([[ 0.22893207, -0.40803994, -0.10392514,  1.56717879],
       [ 0.49702472,  1.15587233,  1.83861168,  1.53572662],
       [ 0.25499773, -0.84415725, -0.98294346, -0.30609783],
       [ 0.83850061, -1.69084816,  1.15117366, -1.02933685],
       [-0.51099219, -2.36027053,  0.10359513,  1.73881773]])

>>> m, sigma = 10, 2
>>> m + sigma*np.random.randn(5)
array([ 8.56778091, 10.84543531,  9.77559704,  9.09052469,  9.48651379])
```

참고자료 : 딥러닝 express

Numpy – 전치 행렬


```
arr=np.array([[1,2],[3,4],[5,6]])
print(arr.T)
```

```
[[1 3 5]
 [2 4 6]]
```

참고자료 : 딥러닝 express

브로드캐스팅


```
>>> miles = np.array([1, 2, 3])
>>> result = miles * 1.6
>>> result
array([1.6, 3.2, 4.8])
```

참고자료 : 딥러닝 express

넘파이 곱셈


```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]])
>>> arr2 = np.array([[2, 2], [2, 2], [2, 2]])
>>> result = arr1 * arr2
>>> result
array([[ 2,  4],
       [ 6,  8],
       [10, 12]])
```

참고자료 : 딥러닝 express

행렬 곱셈


```
>>> arr1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> arr2 = np.array([[2, 2], [2, 2], [2, 2]])
>>> result = arr1 @ arr2                    # arr1.dot(arr2)로 하여도 된다.
array([[12, 12],
       [30, 30],
       [48, 48]])
```

참고자료 : 딥러닝 express

Numpy


```
>>> A = np.array([0, 1, 2, 3])
>>> 10 * np.sin(A)
array([0.        , 8.41470985, 9.09297427, 1.41120008])
```

참고자료 : 딥러닝 express

넘파이 배열 메소드


```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> a.sum()
45
>>> a.min()
1
>>> a.max()
9
```

참고자료 : 딥러닝 express

Numpy-특정한 행과 열을 이용한 연산


```
>>> scores = np.array([[99, 93, 60], [98, 82, 93],
  ...:                 [93, 65, 81], [78, 82, 81]])
>>> scores.mean(axis=0)
array([92.  , 80.5 , 78.75])
```

참고자료 : 딥러닝 express

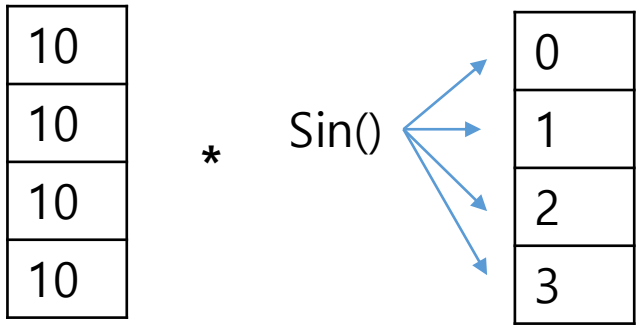
Numpy-브로드 캐스팅


```
>>> miles = np.array([1, 2, 3])
>>> result = miles * 1.6
>>> result
array([1.6, 3.2, 4.8])
```

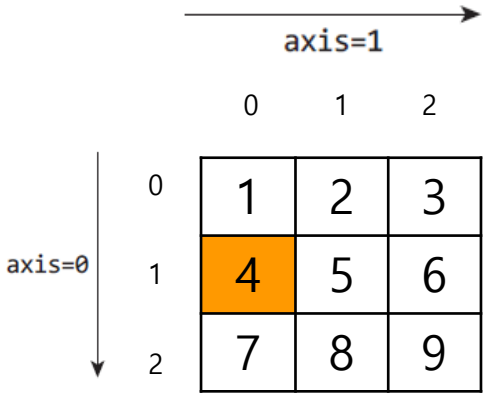
참고자료 : 딥러닝 express

Numpy-함수 적용


```
>>a=np.array([0,1,2,3])
>>10*np.sin(A)
>>array([0.,8.41470985,9.09297427,1.4112008])
```



참고자료 : 딥러닝 express




```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
a1=a[1:2,0:1]
a2=a[1:2,0]
a3=a[1,0:1]
a4=a[1,0]
print('a1',a1,a1.shape)
print('a2',a2,a2.shape)
print('a3',a3,a3.shape)
print('a4',a4,a4.shape)
```

```
a1 [[4]]   (1, 1)
a2 [4]   (1,)
a3 [4]   (1,)
a4 4    ()
```

numpy


```
import numpy as np
a = np.array([1, 2, 3])
a1 = a[np.newaxis, :]
a2 = a[:, np.newaxis]
print('a',a,a.shape)
print('a1',a1,a1.shape)
print('a2',a2,a2.shape)
```

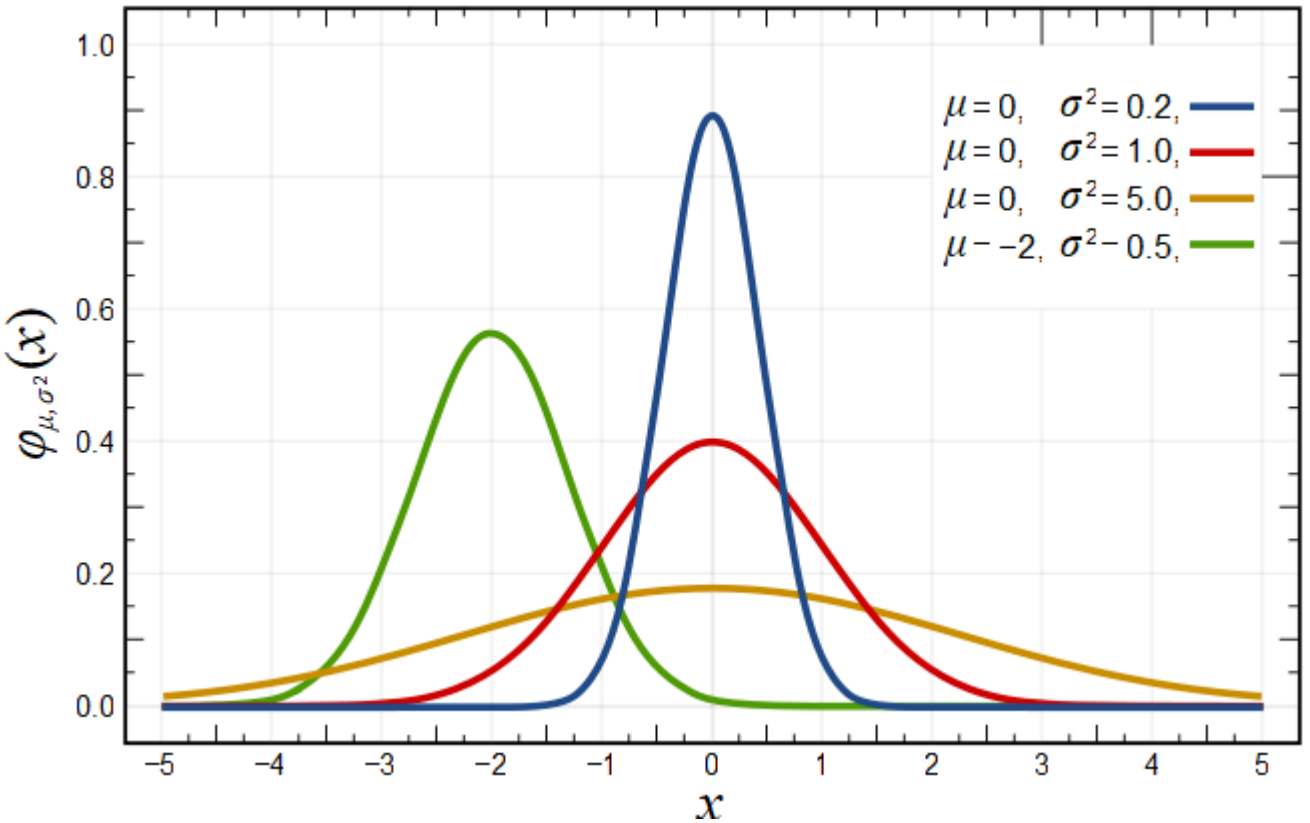
```
a  [1 2 3]          (3,)
a1 [[1 2 3]]        (1, 3)
a2 [[1]
    [2]
    [3]] (3, 1)
```

Numpy-난수 생성


```
>>> np.random.seed(100)
>>> np.random.rand(5)
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])

>>> np.random.rand(5, 3)
array([[0.12156912, 0.67074908, 0.82585276],
       [0.13670659, 0.57509333, 0.89132195],
       [0.20920212, 0.18532822, 0.10837689],
       [0.21969749, 0.97862378, 0.81168315],
       [0.17194101, 0.81622475, 0.27407375]])
```

참고자료 : 딥러닝 express

Numpy


1.0
0.8
0.6
0.4
0.2
0.0
$\varphi_{\mu,\sigma^2}(x)$
−5
−4
−3
−2
−1
0
1
2
3
4
5
x
$\mu = 0, \quad \sigma^2 = 0.2,$
$\mu = 0, \quad \sigma^2 = 1.0,$
$\mu = 0, \quad \sigma^2 = 5.0,$
$\mu - -2, \quad \sigma^2 - 0.5,$

Numpy


```
>>> np.random.randn(5)
array([ 0.78148842, -0.65438103,  0.04117247, -0.20191691, -0.87081315])

>>> np.random.randn(5, 4)
array([[ 0.22893207, -0.40803994, -0.10392514,  1.56717879],
       [ 0.49702472,  1.15587233,  1.83861168,  1.53572662],
       [ 0.25499773, -0.84415725, -0.98294346, -0.30609783],
       [ 0.83850061, -1.69084816,  1.15117366, -1.02933685],
       [-0.51099219, -2.36027053,  0.10359513,  1.73881773]])

>>> m, sigma = 10, 2
>>> m + sigma*np.random.randn(5)
array([ 8.56778091, 10.84543531,  9.77559704,  9.09052469,  9.48651379])
```

참고자료 : 딥러닝 express

Numpy – 전치 행렬


```
import numpy as np
arr=np.array([[1,2],[3,4],[5,6]])
print(arr.T)
```

```
[[1 3 5]
 [2 4 6]]
```

Numpy= 평탄화


```
x = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
x.flatten()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

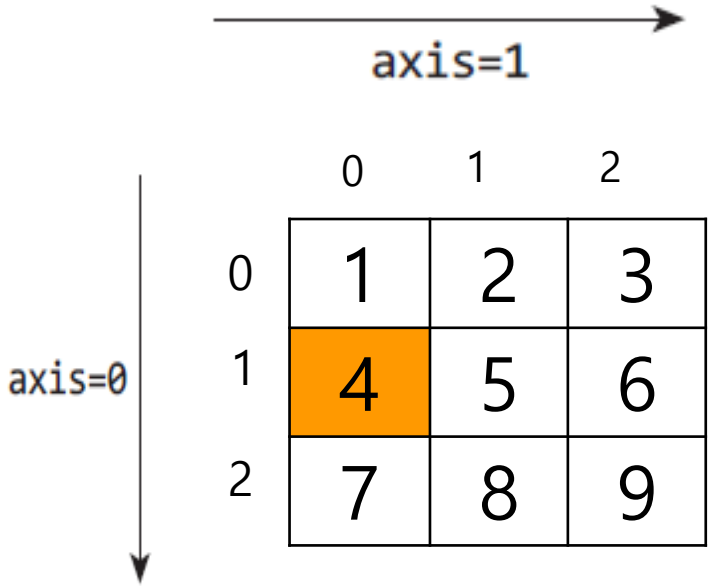
1 차원 배열


```
import numpy as np
a=np.array([1,2,3,4,5])
print(a.shape)
```

```
(5,)
```

```
import numpy as np
a=np.array([1,
            2,
            3,
            4,
            5])
print(a.shape)
```

```
(5,)
```




```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6], [7,
 8, 9]])
a1=a[1:2,0:1]
a2=a[1:2,0]
a3=a[1,0:1]
a4=a[1,0]
print('a1',a1,a1.shape)
print('a2',a2,a2.shape)
print('a3',a3,a3.shape)
print('a4',a4,a4.shape)
```

```
a1 [[4]]    (1, 1)
a2 [4]    (1,)
a3 [4]    (1,)
a4 4     ()
```

numpy


```
import numpy as np
a = np.array([1, 2, 3])
a1 = a[np.newaxis, :]
a2 = a[:, np.newaxis]
print('a',a,a.shape)
print('a1',a1,a1.shape)
print('a2',a2,a2.shape)
```

```
a   [1 2 3]          (3,)
a1 [[1 2 3]]         (1, 3)
a2 [[1]
    [2]
    [3]] (3, 1)
```

numpy


```
import numpy as np

a=np.array([1,2,3])
a3=a.reshape(1,-1)
a4=a.reshape(-1,1)
print(a3,a3.shape)
print(a4,a4.shape)
```

numpy


```
import numpy as np

a=np.array([1,2,3])
a3=a.reshape(1,-1)
a4=a.reshape(-1,1)
print(a3,a3.shape)
print(a4,a4.shape)
```

```
[[1 2 3]] (1, 3)
[[1]
 [2]
 [3]] (3, 1)
```

numpy


```
a1 [[1 2 3]]        (1, 3)
a2 [[1]
    [2]
    [3]] (3, 1)
```

numpy


```
a1 [[1 2 3]]    shape->  (1, 3)
a2 [[1]
    [2]
    [3]] shape-> (3, 1)
```

```
arr=np.array(
   [ [ [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10 ,11] ],
     [ [12, 13, 14, 15],
       [16 ,17 ,18, 19],
       [20 ,21 ,22, 23] ] ])

print(arr,arr.shape)
```

Scikit learn parameter

https://scikit-learn.org/

- **fit (X,y) :학습**

Parameters:
X{array} of shape (n_samples, n_features).

- **Predict (X) :예측**

Parameters:
X{array} of shape (n_samples, n_features).

붓꽃 데이터 세트 (n_samples, n_features).

순번	sepal length (꽃받침 길이)	sepal width (꽃받침 너비)	petal length (꽃잎 길이)	petal width (꽃잎 너비)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.3	0.2
2	4.7	3.2	1.3	0.2
...				
149				

data

3개를 학습 데이터로 사용할때

X= [[5.1, 3.5, 1.4, 0.2],
[4.9, ,3.0, 1.3, 0.2],
[4.7, 3.2, 1.3, 0.2]]

(3, 4)

키(단위: cm)
174
152
138
128
186

5개를 학습 데이터로 사용할때

X= [[174], [152], [138], [128], [186]]

(5, 1)

농어의 무게 예측

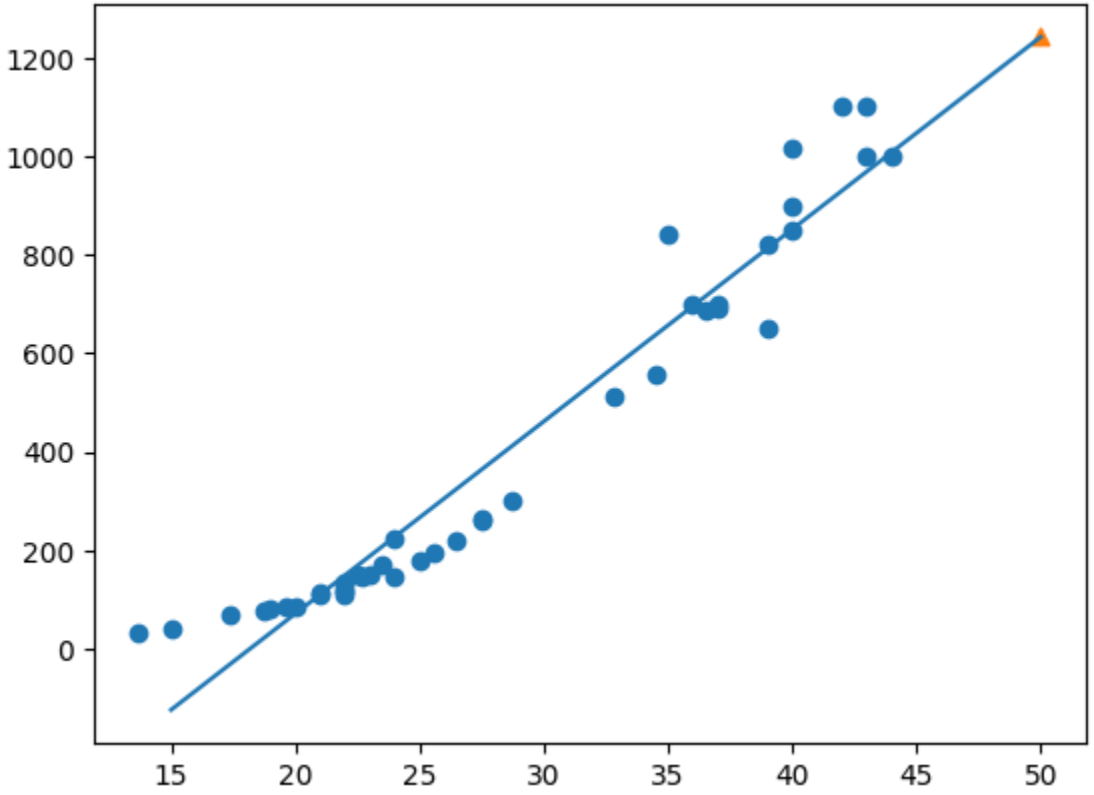
- **입력 데이터 : 길이**
- **출력 데이터 무게**

```
import numpy as npperch_length = np.array(     [8.4, 13.7, 15.0, 16.2,
17.4, 18.0, 18.7, 19.0, 19.6, 20.0,       21.0, 21.0, 21.0, 21.3, 22.0, 22.0,
22.0, 22.0, 22.0, 22.5,       22.5, 22.7, 23.0, 23.5, 24.0, 24.0, 24.6, 25.0,
25.6, 26.5,       27.3, 27.5, 27.5, 27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0,
36.5, 36.0, 37.0, 37.0, 39.0, 39.0, 39.0, 40.0, 40.0, 40.0,       40.0, 42.0,
43.0, 43.0, 43.5, 44.0]      )
```

```
perch_weight = np.array(     [5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0,
85.0, 85.0,       110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0,
110.0,       130.0, 150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0,
197.0, 218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0,       514.0,
556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0,       820.0, 850.0,
900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0,       1000.0, 1000.0]      )
```

참고자료 : 혼자 공부하는 머신 러닝 + 딥러닝

```
from sklearn.model_selection import train_test_split# 훈련 세트와 테스트 세트로 나눕니다

rain_input, test_input, train_target, test_target = train_test_split(     perch_length, perch_weight,
random_state=42)
from sklearn.linear_model import LinearRegression

  lr = LinearRegression() # 선형 회귀 모델 훈련
  lr.fit(train_input, train_target)


# 50cm 농어에 대한 예측 p
rint(lr.predict([[50]]))

print(lr.coef_, lr.intercept_)

plt.scatter(train_input, train_target)

plt.plot([15, 50], [15*lr.coef_+lr.intercept_, 50*lr.coef_+lr.intercept_])
# 50cm 농어 데이터
plt.scatter(50, 1241.8, marker='^')
plt.show()
```


1200
1000
800
600
400
200
0
15
20
25
30
35
40
45
50

과제

- 과제1) numpy_연습.ipynb

- 과제2. 농어.ipynb code가

  1) error가 생기지 않고 2) 길이가 25일때의 무게를 예측 하도록

Code 를 수정 한 후 *.ipynb 파일과 *.py 파일 업로드

선형 회귀 예제 실습

x	Y
0	3
1	3.5
2	5.5

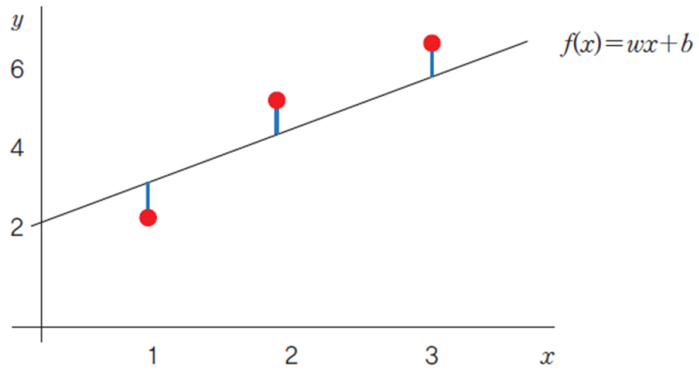


그림 4–5 데이터와 직선 간의 거리

```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([0.0, 1.0, 2.0])
y = np.array([3.0, 3.5, 5.5])

w = 0       # 기울기
b = 0       # 절편

lrate = 0.01  # 학습률
epochs = 1000  # 반복 횟수

n = float(len(X)) # 입력 데이터의 개수

# 경사 하강법
for i in range(epochs):
    y_pred = w*X + b                          # 선형 회귀 예측값
    dw = (2/n) * sum(X * (y_pred-y))          # 넘파이 배열간의 산술 계산은 요소별로 적용
    db = (2/n) * sum(y_pred-y)                # sum()은 모든 요소들의 합을 계산하는 내장 함수
    w = w - lrate * dw                        # 기울기 수정
    b = b - lrate * db                        # 절편 수정

# 기울기와 절편을 출력한다.
print (w, b)

# 예측값을 만든다.
y_pred = w*X + b

# 입력 데이터를 그래프 상에 찍는다.
plt.scatter(X, y)

# 예측값은 선그래프로 그린다.
plt.plot([min(X), max(X)], [min(y_pred), max(y_pred)], color='red')
plt.show()
```

참고자료 : 딥러닝 express

오차 계산하기 (2)

- 평균 제곱 오차(Mean Squared Error, MSE) :

  오차의 합에 이어 각 x 값의 평균 오차를 이용함

  위에서 구한 값을 n으로 나누면 오차 합의 평균을 구할 수 있음

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum (\hat{y}_i - y_i)^2$$

$\hat{y}_i$: 예측값

y_i: 정답

- 선형 회귀란 :

  임의의 직선을 그어 이에 대한 평균 제곱 오차를 구하고, 이 값을 가장 작게 만들

  어 주는 a와 b 값을 찾아가는 작업임

평균 제곱 오차 (Mean Squared Error: MSE)

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum (\hat{y}_i - y_i)^2$$

$\hat{y}_i$: 예측값

y_i: 정답

y_i: 정답

$\hat{y}_i$: 예측값

공부한 시간	2	4	6	8
성적	80	84	92	94
예측 값	70	80	90	100

평균 제곱 오차 (Mean Squared Error: MSE)

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum (\hat{y}_i - y_i)^2$$

$\hat{y}_i$: 예측값

y_i: 정답

y_i: 정답

$\hat{y}_i$: 예측값

공부한 시간	2	4	6	8
성적	80	84	92	94
예측 값	70	80	90	100

$$\text{MSE}=1/4\ (\ (70-80)^2+(80-84)^2+(90-92)^2+(100-94)^2\ )$$

$$= \tfrac{1}{4}(100+16+4+36)=1/4(156)= 39$$

평균 제곱 오차 (Mean Squared Error: MSE)

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum (\hat{y}_i - y_i)^2$$

$\hat{y}_i$: 예측값

y_i: 정답

y_i: 정답

$\hat{y}_i$: 예측값

공부한 시간	2	4	6	8
성적	80	84	92	94
예측 값	78	82	88	96

$$\text{MSE}=1/4\ (\ (78-80)^2+(82-84)^2+(88-92)^2+(96-94)^2\ )$$

$$= \tfrac{1}{4}(4+4+16+4)=1/4(28)=\ 7$$

오차 수정하기 : 경사 하강법

- **Y=ax**
- **오차가 가장 작은 지점은?**

경사하강법

- 그래프에서 오차를 작은 방향으로 이동. 미분 기울기 이용
- 미분을 하면 순간 기울기가 구해짐.
- 미분값이 0인 지점이 오차가 가장 작은 지점

경사하강법

- 미분 값으로 기울기를 구한후,
- 기울기 반대 방향으로 학습률*기울기 만큼 움직임

학습률

- 학습률이 낮을 때 학습이 너무 오래 걸림. 정답을 찾을 확률은 높음
- 학습률이 높을때. 학습 속도가 빠르지만 정답을 놓칠 확률이 좋음,

경사 하강법 실습

- 손실 함수 $y = (x-3)^2 + 10$
- 그래디언트(미분된 함수): y' = 2x-6
- 학습률 : 0.2

X의 초기 위치가 10일때 두번째 위치는?

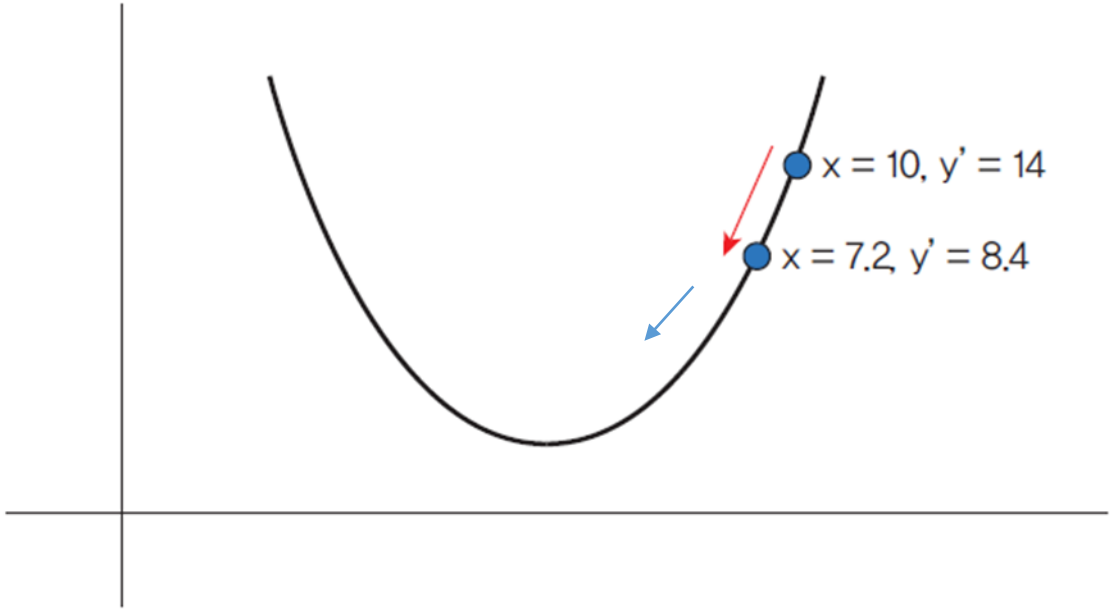


그림 6-12 그래디언트의 계산

- X=10, y'=14, 0.2*14=2.8,

Gradient의 반대 방향 => -2.8

10-2.8=7.2

- X=7.2, Y' =8.4, 0.2*8.4=

참고자료 : 딥러닝 express

경사하강법

손실 함수가 $y = (x-2)^2 + 5$ 이고

그래디언트(미분된 함수) 는 $y' = 2x - 4$,

x의 초기 위치는 7.0, 학습률은 0.3일때 경사 하강법이 적용될 때
x 의 두 번째 위치를 구하시오.

경사 하강법 실습

- 과제3

- colab_03_Linear_Regression.ipynb 실행
- 학습률 lr=0.0003 일때의 첫번째와 두번째 loss 제출
- 학습률 lr=0.0009 일때의 첫번째와 두번째 loss 제출

경사 하강법 실습

과제4. grad2_exe.ipynb

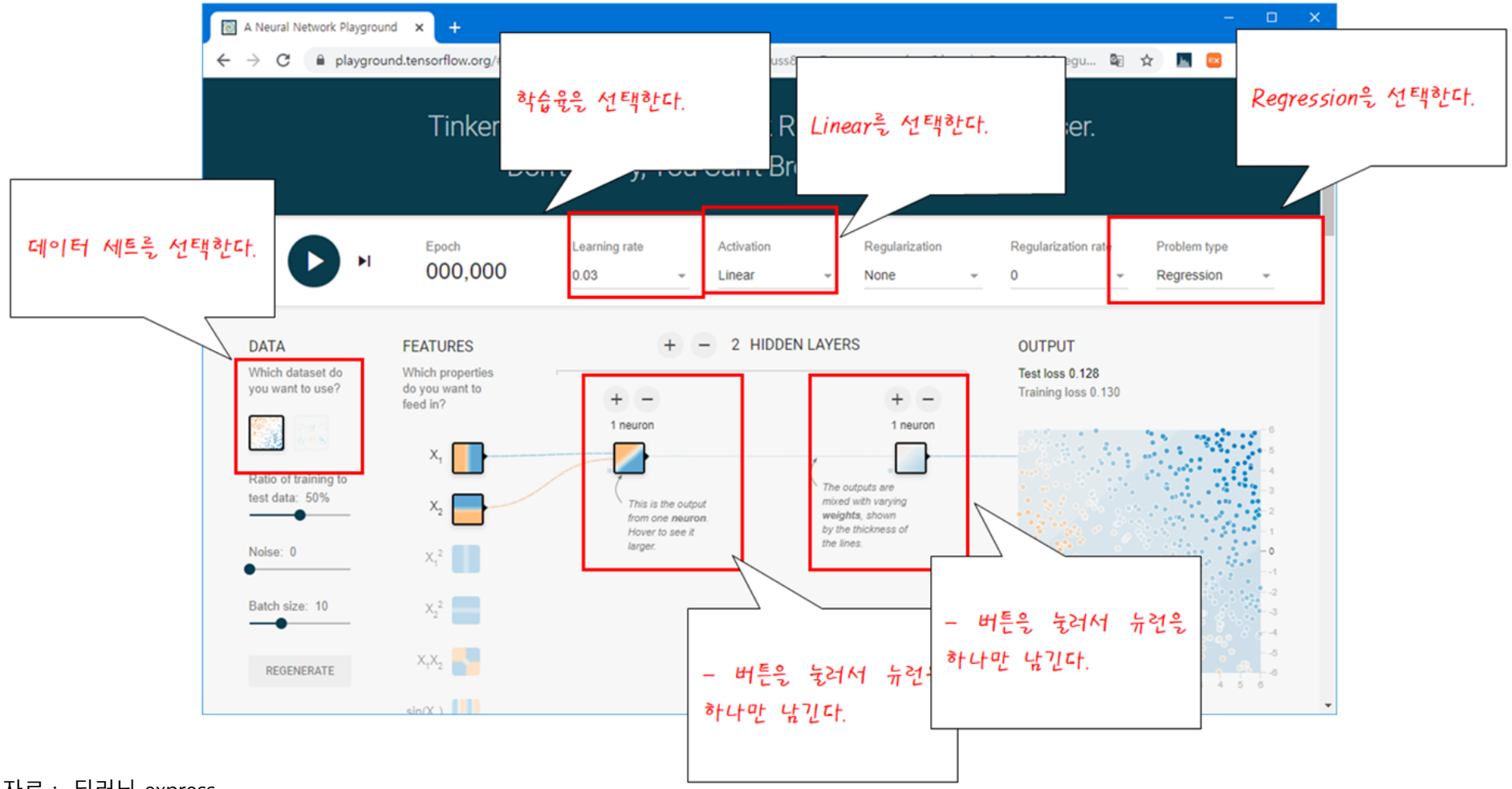
3번째 x값과 loss값 제출

```python
import numpy as np
import matplotlib.pyplot as plt

x = 10
learning_rate = 0.2

max_iterations = 100

# 손실함수를 람다식으로 정의한다.
loss_func = lambda x: (x-3)**2 + 10

# 그래디언트를 람다식으로 정의한다. 손실함수의 1차 미분값이다.
gradient = lambda x: 2*x-6

list1 = []
list2 = []

# 그래디언트 강하법
for i in range(max_iterations):
    x = x - learning_rate * gradient(x)
    list1.append(x)
    list2.append(loss_func(x))
    print("X=", x, "loss", loss_func(x))

print("최소값 = ", x)

x1 = np.linspace(0.0, 10.0)
y1 = loss_func(x1)
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot(x1,y1)  # Plot some data on the axes.
ax.plot(list1,list2, '*')  # Plot some data on the axes.
```

참고자료 : 딥러닝 express

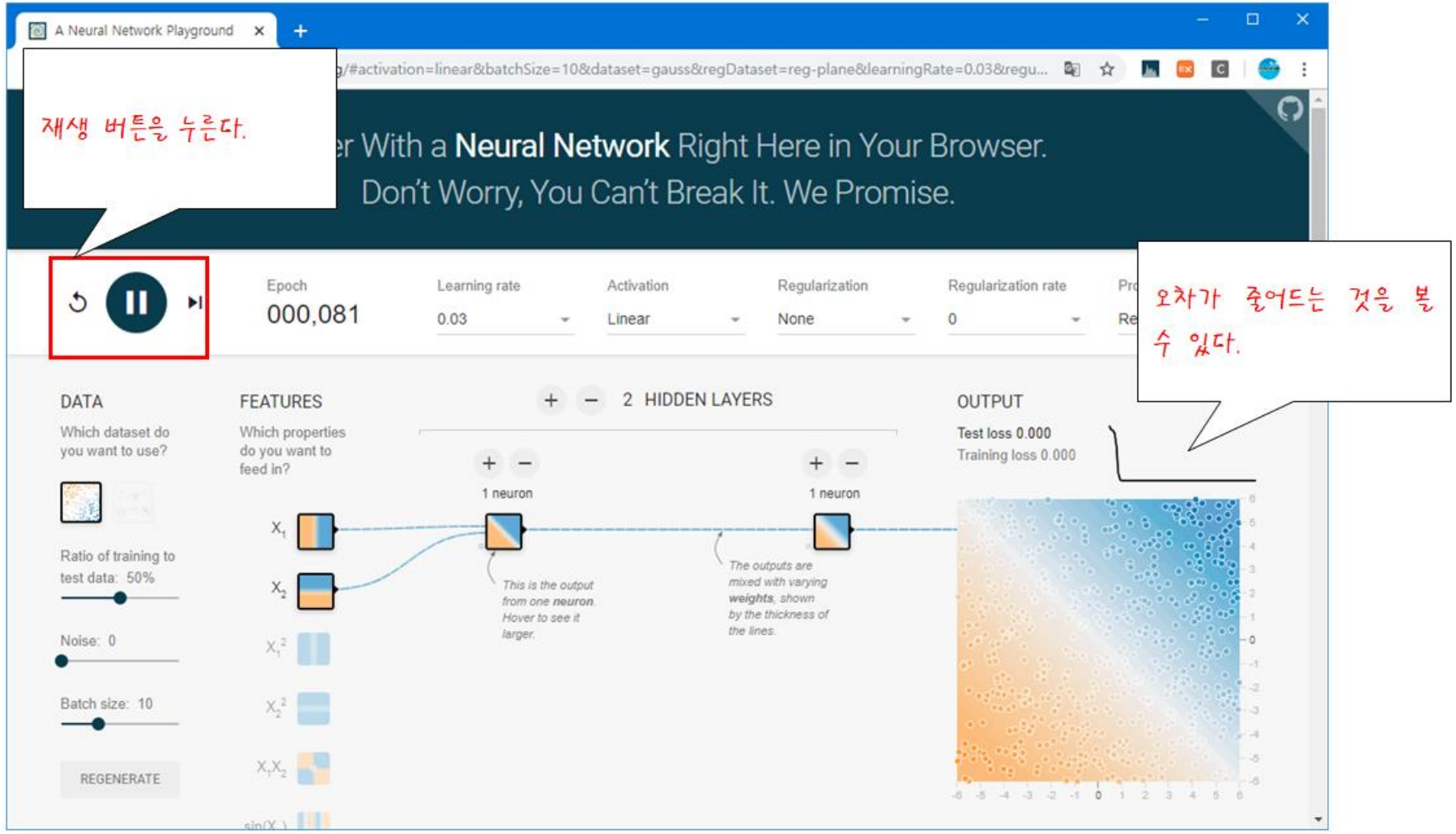
학습률 실습

- 구글의 텐서 플로우 플레이그라운드는 이주 유용한 사이트 (https://playground.tensorflow.org)이다.



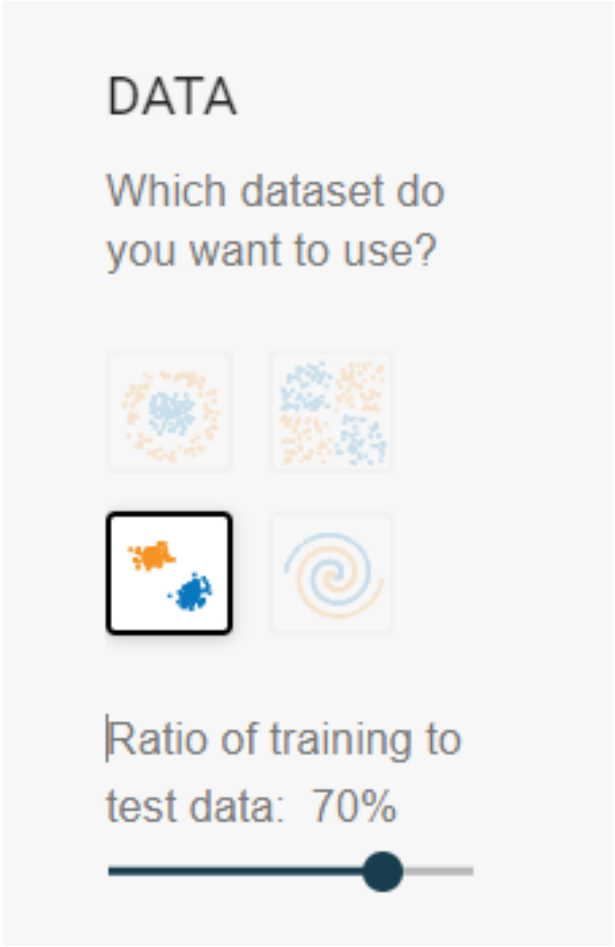
참고자료 : 딥러닝 express

학습률 실습



참고자료 : 딥러닝 express

학습률 실습



참고자료 : 딥러닝 express

수업 내용 요약

- **손실 함수 (오차=loss)는 경사 하강법으로 감소 시킴**
- **경사 하강법**
  - 손실 함수(오차)를 미분한 값(기울기)의 반대 방향으로 진행 하여 최소값을 찾아감
  - 학습률에 비례 하여 진행
  - 학습률에 따라 최소값에 도달하는 시간과 정확도가 정해짐

수고하셨습니다

jhmin@inhatc.ac.kr