(AI Math 1강) 벡터가 뭐에요?

```
import numpy as np
x = np.array([1,7,2])
y = np.array([5,2,1])


# 성분곱(Hadamard product)
x * y
```

```
array([ 5, 14,  2])
```

```
# L1 노름 : 각 성분의 변화량의 절대값
def l1_norm(x):
  x_norm = np.abs(x)
  x_norm = np.sum(x_norm)
  return x_norm

# L2 노름 : 유클리드 거리
def l2_norm(x):
  x_norm = x*x
  x_norm = np.sum(x_norm)
  x_norm = np.sqrt(x_norm)
  return x_norm
```

(AI Math 2강) 행렬이 뭐에요?

```
x = np.array([[1,-2,3],
              [7,5,0],
              [-2,-1,2]])

y = np.array([[0,1],
              [1,-1],
              [-2,1]])
# 행렬 곱셈
x @ y
```

```
array([[-8,  6],
       [ 5,  2],
       [-5,  1]])
```

```
x = np.array([[1,-2,3],
              [7,5,0],
              [-2,-1,2]])

y = np.array([[0,1,-1],
              [1,-1,0]])
```

```
# 행렬 내적 : i번째 행 벡터와 j번째 행 벡터 사이의 내적
np.inner(x,y)
```

```
array([[-5,  3],
       [ 5,  2],
       [-3, -1]])
```

```
# 역행렬
x = np.array([[1,-2,3],
             [7,5,0],
             [-2,-1,2]])
np.linalg.inv(x)
```

```
array([[ 0.21276596,  0.0212766 , -0.31914894],
       [-0.29787234,  0.17021277,  0.44680851],
       [ 0.06382979,  0.10638298,  0.40425532]])
```

```
np.linalg.inv(x) @ x
```

```
array([[ 1.00000000e+00,  1.11022302e-16, -1.11022302e-16],
       [-1.11022302e-16,  1.00000000e+00,  1.11022302e-16],
       [-1.11022302e-16,  5.55111512e-17,  1.00000000e+00]])
```

```
# 유사역행렬, 무어-펜로즈 역행렬
y = np.array([[0,1],
              [1,-1],
              [-2,1]])
np.linalg.pinv(y)
```

```
array([[ 5.00000000e-01,  4.09730229e-17, -5.00000000e-01],
       [ 8.33333333e-01, -3.33333333e-01, -1.66666667e-01]])
```

```
np.linalg.pinv(y) @ y
```

```
array([[ 1.00000000e+00, -2.22044605e-16],
       [ 5.55111512e-17,  1.00000000e+00]])
```

```
# 사이킷런을 활용한 회귀분석
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X,y)
y_test = model.predict(x_test)

# 무어-펜로즈 역행렬
X_ = np.array([np.append(x,[1]) for x in X])
beta = np.linalg.pinv(X_) @ y
y_test = np.append(x, [1]) @ beta
```

(AI Math 3강) 경사하강법 - 순한맛

```python
import sympy as sym
from sympy.abc import x

sym.diff(sym.poly(x**2+2*x+3), x)
```

$$\text{Poly}\left(2x + 2, x, domain = \mathbb{Z}\right)$$

```python
var = init
grad = gradient(var)


while(abs(grad) > eps): # 컴퓨터에서 미분이 정확히 0이 되는 것은 불가하기 때문에 eps보다 작을 때
  var = var - lr * grad # 학습률로 미분 업데이트 속도 조절
  grad = gradient(var) # 미분값 업데이트
```

```python
import numpy as np
import sympy as sym
from sympy.abc import x
from sympy.plotting import plot

def func(val):
    fun = sym.poly(x**2 + 2*x + 3)
    return fun.subs(x, val), fun

def func_gradient(fun, val):
    ## TODO
    _, function = fun(val)
    diff = sym.diff(function, x)
    return diff.subs(x, val), diff

def gradient_descent(fun, init_point, lr_rate=1e-2, epsilon=1e-5):
    cnt = 0
    val = init_point
    ## Todo
    diff, _ = func_gradient(fun, init_point)
    while np.abs(diff) > epsilon:
      val = val - lr_rate * diff
      diff, _ = func_gradient(fun, val)
      cnt += 1

    print("함수: {}₩n연산횟수: {}₩n최소점: ({}, {})".format(fun(val)[1], cnt, val, fun(val)[0]))
```

```python
import sympy as sym
from sympy.abc import x, y

# 다변수 함수(입력: 벡터) 편미분
sym.diff(sym.poly(x**2 + 2*x*y + 3) + sym.cos(x + 2*y), x)
```

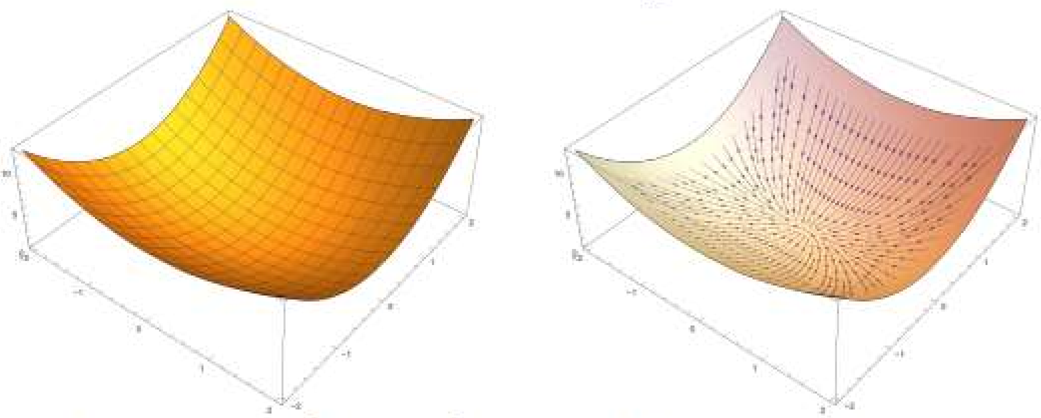
```
/usr/local/lib/python3.7/dist-packages/sympy/polys/polytools.py:79: SymPyDeprecationWarning:

Mixing Poly with non-polynomial expressions in binary operations has
been deprecated since SymPy 1.6. Use the as_expr or as_poly method to
convert types instead. See https://github.com/sympy/sympy/issues/18613
for more info.

  useinstead="the as_expr or as_poly method to convert types").warn()
```

$2x + 2y - \sin(x + 2y)$

그래디언트 벡터 : 각 변수 별로 편미분을 계산

기호는 nabla라고 부른다.

$$\nabla f = (\partial_{x_1} f, \partial_{x_2} f, \cdots, \partial_{x_d} f)$$



$f(x, y) = x^2 + 2y^2$ $\quad$ $-\nabla f = -(2x, 4y)$

= 극소점 방향으로 이동

코드

↳ 숨겨진 셀 1개

(AI Math 4강) 경사하강법 - 매운맛

```
import numpy as np

X = np.array([[1,1], [1,2], [2,2], [2,3]])
y = np.dot(X, np.array([1,2])) + 3

beta_gd = [10.1, 15.1, -6.5] # 정답 [1, 2, 3]
X_ = np.array([np.append(x, [1]) for x in X]) # 절편 항 추가

# 학습률 작게 : 수렴이 늦음
# 학습률 크게 : 발산 되거나 수렴이 이상하게 됨
# 학습횟수 작으면 : 수렴이 잘 안되거나 되다 말수도 있다.
```

```
# 학습횟수 크면 :
for t in range(5000):
  error = y - X_ @ beta_gd
  grad = - np.transpose(X_) @ error
  beta_gd = beta_gd - 0.01 * grad

print(beta_gd)
```

```
[1.00000367 1.99999949 2.99999516]
```

비선형회귀에는 SGD 사용

- SGD는 데이터 한개 또는 일부(미니 배치) 파라미터를 업데이트해 연산자원을 좀 더 효율적인 사용이 가능하다.
- 딥러닝에서 SGD가 경사하강법보다 낫다고 검증됨
  - Non-convex 함수에도 적용가능하기 때문에 GD보다 SGD가 머신러닝 학습에 더 효율적이다.
  - 미니배치 사이즈가 작으면 GD보다 SGD가 느려질 수도 있음

(AI Math 5강) 딥러닝 학습방법 이해하기

더블클릭 또는 Enter 키를 눌러 수정

소프트맥스 함수는 모델의 출력을 확률로 해설할 수 있게 변환해준다.

분류 문제를 풀 때 선형 모델과 소프트맥스 함수를 결합하여 예측한다.

$$\mathrm{softmax}(\mathbf{o}) = \mathrm{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

코드

[ ] ↳ 숨겨진 셀 2개

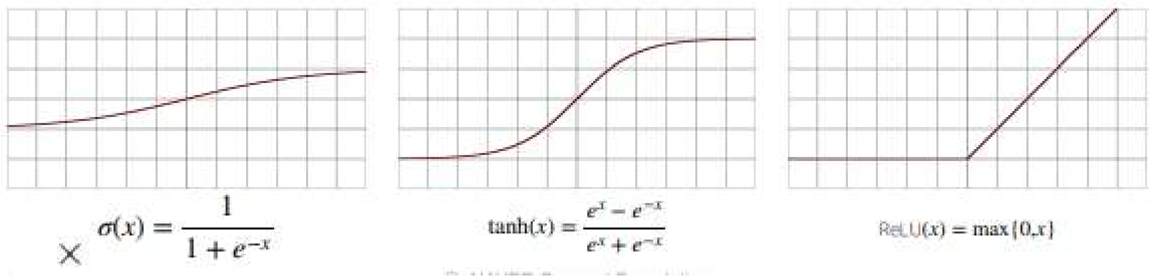
신경망

신경망 함수 = 선형 모델 + 활성 함수를 합성

MLP = 신경망이 여러층 합성된 함수

활성 함수 그래프

- 딥러닝에서는 ReLU를 많이 쓴다.



universal approximation theorem : 이론적으로는 2층 신경망으로도 임의의 연속함수를 근사할 수 있다.

그러나 층이 깊을수록 목적함수를 근사하는데 필요한 뉴런(노드)의 숫자가 훨씬 빨리 줄어들어 좀 더 효율적으로 학습이 가능하다.

- But 층이 깊어지면 최적화가 어려워진다.

층이 얇으면 필요한 뉴런의 숫자가 기하급수적으로 늘어나서 넓은(wide) 신경망이 되어야 한다.

역전파 알고리즘을 이용해서 파라미터 학습

- 역전파 알고리즘은 합성함수 미분법인 연쇄 법칙(chain-rule) 기반 자동 미분 (auto-differentiation)을 사용
- 각 노드의 텐서 값을 컴퓨터가 기억해야 미분 계산이 가능

(AI Math 6강) 확률론 맛보기

- 딥러닝은 확률론 기반의 기계학습 이론에 바탕을 둔다
- 기계학습에서 사용되는 손실함수(loss&function)들의 작동 원리는 데이터 공간을 통계적으로 해석해서 유도한다.
- 확률변수는 확률분포에 따라 이산형, 연속형 확률변수로 구분한다.
- 물류 회귀에서 사용했던 선형모델과 소프트맥스 함수의 결합은 데이터에서 추출된 패턴을 기반으로 확률을 해석하는데 사용됩니다

다양한 기대값들

$$\mathbb{V}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim P(\mathbf{x})}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])^2] \qquad \mathrm{Skewness}(\mathbf{x}) = \mathbb{E}\left[\left(\frac{\mathbf{x} - \mathbb{E}[\mathbf{x}]}{\sqrt{\mathbb{V}(\mathbf{x})}}\right)^3\right]$$

$$\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2) = \mathbb{E}_{\mathbf{x}_1,\mathbf{x}_2\sim P(\mathbf{x}_1,\mathbf{x}_2)}[(\mathbf{x}_1 - \mathbb{E}[\mathbf{x}_1])(\mathbf{x}_2 - \mathbb{E}[\mathbf{x}_2])]$$

몬테카를로 샘플링 방법 : 확률분포를 모를 때 데이터를 이용해 기대값을 계산할 때 사용한다.

- 기계학습의 많은 문제들은 확률분포를 명시적으로 몰라서 사용한다.

- 이산/연속 상관 없이 사용 가능한 방법이다.
- 몬테카를로 샘플링은 독립추출만 보장되면 대수의 법칙에 의해 수렴성을 보장한다.

$$\mathbb{E}_{\mathbf{x}\sim P(\mathbf{x})}[f(\mathbf{x})] \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}^{(i)}), \quad \mathbf{x}^{(i)} \overset{\text{i.i.d.}}{\sim} P(\mathbf{x})$$

코드

[ ] ↳ 숨겨진 셀 *1*개

(AI Math 7강) 통계학 맛보기

통계적 모델링은 적절한 가정 위에서 확률 분포를 추정(inference)하는 것이 목표이며, 기계학습과 통계학이 공통으로 추구하는 목표

- 예측모형의 목적은 분포를 정확하게 맞추는 것보다는데이터와 추정 방법의 불확실성을 고려해서 위험을 최소화하는 것

모수적(parametric) 방법론 : 데이터가 특정확률분포를 따른다고 선험적으로(apriori) 가정한 후 그 분포를 결정하는 모수(parameter)를 추정하는 방법

비모수(nonparametric) 방법론 : 특정확률분포를 가정하지 않고 데이터에 따라 모델의 구조 및 모수의 개수가 유연하게 바뀔 때 사용

- 기계학습의 많은 방법론은 비모수 방법론이다.

PPT 필기

↳ 숨겨진 셀 *14*개

(AI Math 8강) 베이즈 통계학 맛보기

통계학 : 빈도주의

베이즈 통계학 : 주관주의

PPT 필기

↳ 숨겨진 셀 *11*개

(AI Math 9강) CNN 첫걸음

커널 신호

continuous $$[f * g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)\mathrm{d}z = \int_{\mathbb{R}^d} f(x-z)g(z)\mathrm{d}z = [g * f](x)$$

discrete $$[f * g](i) = \sum_{a \in \mathbb{Z}^d} f(a)g(i-a) = \sum_{a \in \mathbb{Z}^d} f(i-a)g(a) = [g * f](i)$$

CNN은 커널을 이용해 정보를 확대, 감소, 추출하는 연산을 수행한다.

- 커널 : 정의역 내에서 움직여도 변하지 않고 신호에 국소적으로 적용한다,
- CNN에서 사용하는 연산은 +로 사실은 convolution이 아닌 엄밀하게 따지면 cross-correlation이다. 즉 원래는 CCNN이여야 한다는 것.
- 데이터의 성격에 따라 사용하는 커널이 달라진다.
- 커널 개수에 따라 출력도 달라진다.

채널 여러개 인 2차원 입력은 2차원 Convolution * 채널 개수로 연산을 적용한다.

- 3차원은 텐서

PPT 필기

↳ 숨겨진 셀 *11*개

(AI Math 10강) RNN 첫걸음

시계열 데이터 : 소리, 문자열, 주가 등의 데이터

- 시퀀스 데이터는 독립동등분포(i.i.d.) 가정을 잘 위배한다. 따라서 순서를 바꾸거나 과거 정보에 손실이 발생하면 데이터의 확률 분포도 바뀐다. => 맥락이 중요하다.
  - 개가 사람을 물었다
  - 사람이 개를 물었다.
- 시퀀스 정보를 가지고 미래 발생할 데이터의 확률분포를 다루기 위해 조건부확률을 이용한다.
  - 시퀀스 데이터를 분석할 때 과거의 모든 정보가 필요한 것은 아니다.
  - 조건부에 들어가는 데이터 길이는 가변적이다.

그래디언트 배니싱(기울기 소실 == 그래디언트 -> 0)되면 과거 정보 유실 위험

- 문맥적으로 이전 시점이 중요한 텍스트 분석, 긴 시퀀스 분석해야 하는 경우 기울기가 0으로

PPT 필기

↳ 숨겨진 셀 9개