(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

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

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

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

```
 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
```

```
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: {}Wn연산횟수: {}Wn최소점: ({}, {})".format(fun(val)[1], cnt, val, fun(val)[0]))
```

```
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)
```

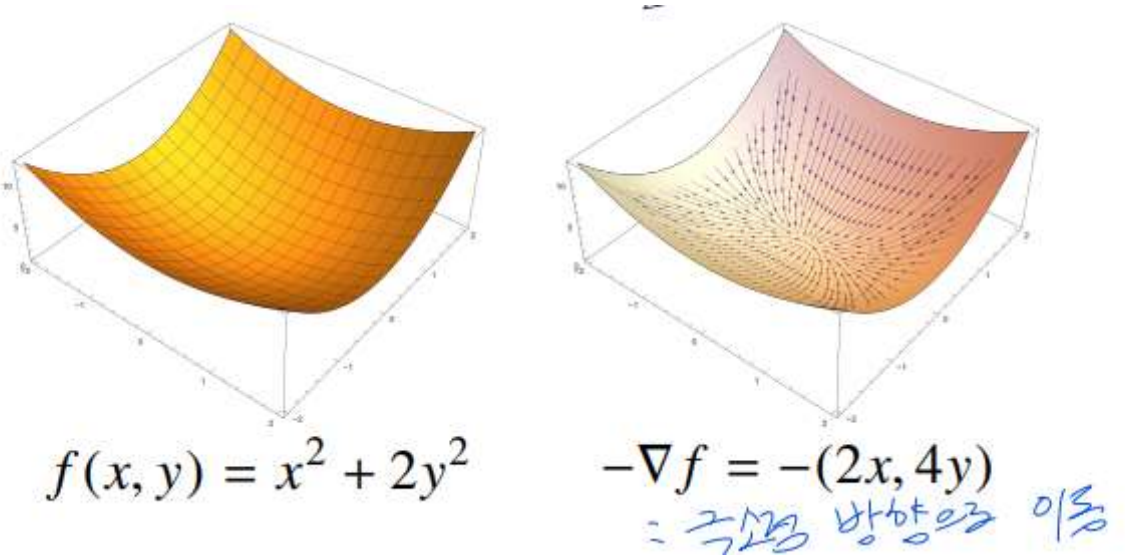
```
/usr/local/lib/python3.7/dist-packages/sympy/polys/polytools.py:79: SymPyDeprecationWarning:

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.

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



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.

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

코드

```
# 소프트맥스 함수
def softmax(vec):
  # overflow 방지를 위해 np.max 를 뺀다.
  denumerator = np.exp(vec - np.max(vec, axis=-1, keepdims=True))
  numerator = np.sum(denumerator, axis=1, keepdims=True)
  val = denumerator / numerator
  return val

vec = np.array([[1,2,0],[-1,0,1],[-10,0,10]])
softmax(vec)
```

```
array([[2.44728471e-01, 6.65240956e-01, 9.00305732e-02],
       [9.00305732e-02, 2.44728471e-01, 6.65240956e-01],
       [2.06106005e-09, 4.53978686e-05, 9.99954600e-01]])
```

```
# 원핫 함수 : 추론에만 사용
def one_hot(val, dim):
  return [np.eye(dim)[_] for _ in val]

def one_hot_encoding(vec):
  vec_dim = vec.shape[1]
  vec_argmax = np.argmax(vec, axis=-1)
  return one_hot(vec_argmax, vec_dim)

# 소프트맥스 함수 : 학습시에만 필요
def softmax(vec):
  denumerator = np.exp(vec - np.max(vec, axis=-1, keepdims=True))
  numerator = np.sum(denumerator, axis=1, keepdims=True)
  val = denumerator / numerator
  return val

vec = np.array([[1,2,0],[-1,0,1],[-10,0,10]])
print(one_hot_encoding(vec))
print(one_hot_encoding(sotfmax(vec)))

      [array([0., 1., 0.]), array([0., 0., 1.]), array([0., 0., 1.])]
      [array([0., 1., 0.]), array([0., 0., 1.]), array([0., 0., 1.])]
```

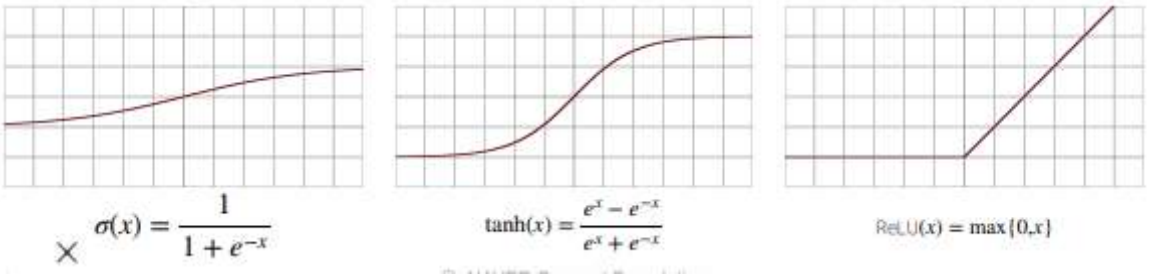
신경망

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

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

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

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 생플링은 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

```
import numpy as np

def mc_int(fun, low, high, sample_size=100, repeat=10):
  int_len = np.abs(high-low)
  stat = []
  for _ in range(repeat):
```

```
    # 균등 분포에서 데이터 샘플링
    x = np.random.uniform(low=low, high=high, size=sample_size)

    # 함수에 값 대입
    fun_x = fun(x)

    # 구간 길이 * 대입된 함수들의 산술평균
    int_val = int_len * np.mean(fun_x)

    stat.append(int_val)
  return np.mean(stat), np.std(stat)

def f_x(x):
  return np.exp(-x**2)

# 샘플 사이즈가 적으면 오차 범위가 커진다.
print(mc_int(f_x, low=-1, high=1, sample_size=10000, repeat=100))
```

```
(1.4936098869590668, 0.0038028201675268498)
```

(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

조건부 확률이란?

- 베이즈 통계학을 이해하기 위해선 조건부확률의 개념을 이해해야 합니다
- 베이즈 정리는 조건부확률을 이용하여 정보를 갱신하는 방법을 알려줍니다

$$P(A \cap B) = P(B)P(A|B)$$

$$P(B|A) = \frac{P(A \cap B)}{P(A)} = P(B)\frac{P(A|B)}{P(A)}$$

A 라는 새로운 정보가 주어졌을 때 $P(B)$ 로부터 $P(B|A)$ 를 계산하는 방법을 제공한다

베이즈 정리: 예제

$$P(\theta|\mathcal{D}) = P(\theta)\frac{P(\mathcal{D}|\theta)}{P(\mathcal{D})}$$

- $P(\theta|\mathcal{D})$: 사후확률 (posterior)
- $P(\theta)$: 사전확률 (prior)
- $P(\mathcal{D}|\theta)$: 가능도 (likelihood)
- $P(\mathcal{D})$: Evidence

- COVID-99 의 발병률이 10% 로 알려져있다. COVID-99 에 실제로 걸렸을 때 검진될 확률은 99%, 실제로 걸리지 않았을 때 오검진될 확률이 1% 라고 할 때, 어떤 사람이 질병에 걸렸다고 검진결과가 나왔을 때 정말로 COVID-99 에 감염되었을 확률은?

사전확률, 민감도(Recall), 오탐율(False alarm)을 가지고 정밀도(Precision)를 계산하는 문제이다

베이즈 정리: 예제

$$P(\theta|\mathcal{D}) = P(\theta)\frac{P(\mathcal{D}|\theta)}{P(\mathcal{D})} \qquad P(\theta) = 0.1 \qquad \begin{matrix} P(\mathcal{D}|\theta) = 0.99 \\ P(\mathcal{D}|\neg\theta) = 0.01 \end{matrix}$$

$$P(\mathcal{D}) = \sum_{\theta} P(\mathcal{D}|\theta)P(\theta) = 0.99 \times 0.1 + 0.01 \times 0.9 = 0.108$$

$$P(\theta|\mathcal{D}) = 0.1 \times \frac{0.99}{0.108} \approx 0.916$$

- COVID-99 의 발병률이 10% 로 알려져있다. COVID-99 에 실제로 걸렸을 때 검진될 확률은 99%, 실제로 걸리지 않았을 때 요검진될 확률이 1% 라고 할 때, 어떤 사람이 질병에 걸렸다고 검진결과가 나왔을 때 정말로 COVID-99 에 감염되었을 확률은?

만일 오검진될 확률(1종 오류)이
1% 가 아닌 10% 면 어떻게 될까?

베이즈 정리: 예제

$$P(\theta|\mathcal{D}) = P(\theta)\frac{P(\mathcal{D}|\theta)}{P(\mathcal{D})} \qquad P(\theta) = 0.1 \qquad \begin{matrix} P(\mathcal{D}|\theta) = 0.99 \\ P(\mathcal{D}|\neg\theta) = 0.1 \end{matrix}$$

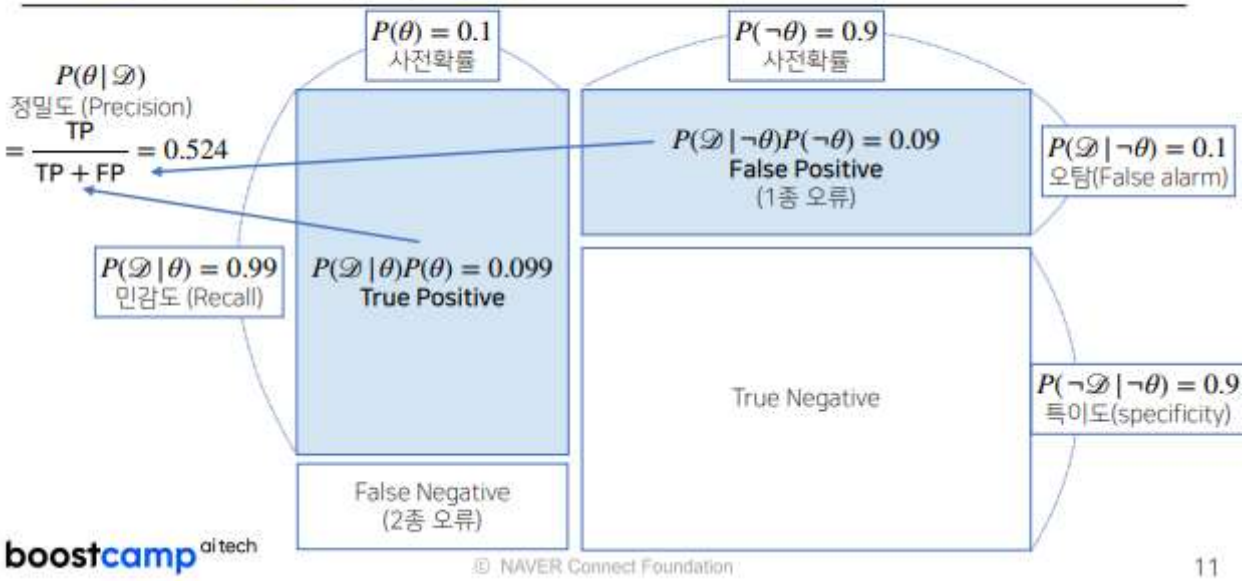
$$P(\mathcal{D}) = \sum_{\theta} P(\mathcal{D}|\theta)P(\theta) = 0.99 \times 0.1 + 0.1 \times 0.9 = 0.189$$

$$P(\theta|\mathcal{D}) = 0.1 \times \frac{0.99}{0.189} \approx 0.524$$

- COVID-99 의 발병률이 10% 로 알려져있다. COVID-99 에 실제로 걸렸을 때 검진될 확률은 99%, 실제로 걸리지 않았을 때 오검진될 확률이 1% 라고 할 때, 어떤 사람이 질병에 걸렸다고 검진결과가 나왔을 때 정말로 COVID-99 에 감염되었을 확률은?

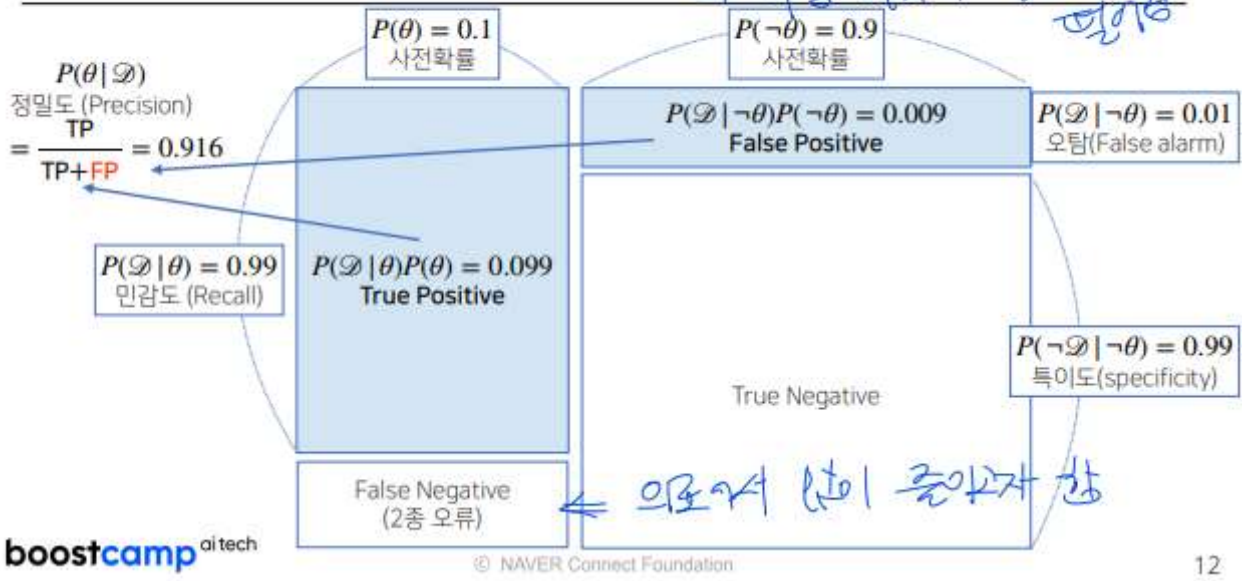
오탐율(False alarm)이 오르면 테스트의
정밀도(Precision)가 떨어진다

조건부 확률의 시각화



조건부 확률의 시각화

사전확률 보면 양쪽 가정하나 → 추정 결과의 성능의 한계점



베이즈 정리를 통한 정보의 갱신

데이터 들어올 때 사후 업데이트 가능

- 베이즈 정리를 통해 새로운 데이터가 들어왔을 때 앞서 계산한 사후확률을 사전확률로 사용하여 갱신된 사후확률을 계산할 수 있습니다

$$P(\theta|\mathcal{D}) = 0.1 \times \frac{0.99}{0.189} \approx 0.524$$

$$P(\mathcal{D}|\theta) = 0.99$$
$$P(\mathcal{D}|\neg\theta) = 0.1$$

$$P(\mathcal{D}^*) = 0.99 \times 0.524 + 0.1 \times 0.476 \approx 0.566$$

갱신된 사후확률 (posterior) $P(\theta|\mathcal{D}^*) = 0.524 \times \frac{0.99}{0.566} \approx 0.917$

- 앞서 COVID-99 판정을 받은 사람이 두 번째 검진을 받았을 때도 양성이 나왔을 때 진짜 COVID-99 에 걸렸을 확률은?

세번째 검사해도 양성이 나오면 정밀도가 99.1% 까지 갱신된다

조건부 확률 → 인과관계?

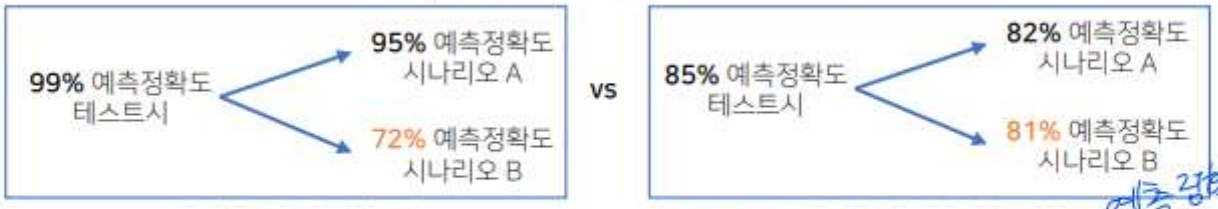
- 조건부 확률은 유용한 통계적 해석을 제공하지만 인과관계(causality)를 추론할 때 함부로 사용해서는 안 됩니다

데이터가 많아져도 조건부 확률만 가지고 인과관계를 추론하는 것은 불가능합니다

조건부 확률 → 인과관계?

- 조건부 확률은 유용한 통계적 해석을 제공하지만 인과관계(causality)를 추론할 때 함부로 사용해서는 안 됩니다
- 인과관계는 데이터 분포의 변화에 강건한 예측모형을 만들 때 필요합니다

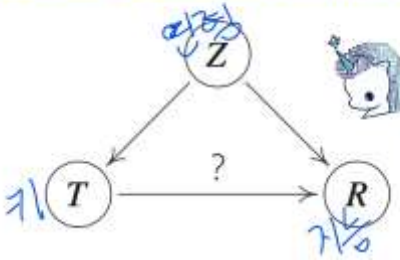
단, 인과관계만으로는 높은 예측 정확도를 담보하기는 어렵습니다



조건부확률 기반 예측모형 | 인과관계 기반 예측모형

↑ 데이터 분포 변화에 (예: 시대별 변화) ↓ 예측 강건하다

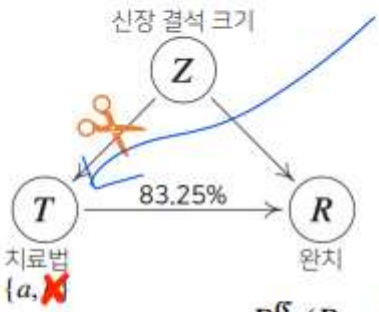
- 인과관계를 알아내기 위해서는 중첩요인(confounding factor)의 효과를 제거하고 원인에 해당하는 변수만의 인과관계를 계산해야 합니다



만일 Z 의 효과를 제거하지 않으면 가짜 연관성(spurious correlation)이 나온다

→ 예측 모델 정확도 떨어짐

인과관계 추론: 예제



	Overall	Z = 0: Patients with small stones	Z = 1: Patients with large stones
Treatment *a*: Open surgery	78% (273/350)	**93%** (81/87)	**73%** (192/263)
Treatment *b*: Percutaneous nephrolithotomy	**83%** (289/350)	87% (234/270)	69% (55/80)

출처: Elements of Causal Inference, Peters et al.

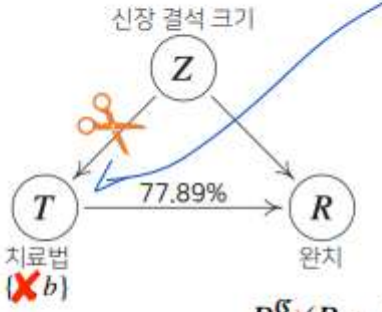
$$P^{\mathfrak{G}_a}(R=1) = \sum_{z\in\{0,1\}} P^{\mathfrak{G}}(R=1 \mid T=a, Z=z)P^{\mathfrak{G}}(Z=z)$$

do(T = a) 라는 조정(intervention) 효과를 통해 Z 의 개입을 제거한다

$$= \frac{81}{87} \times \frac{(87+270)}{700} + \frac{192}{263} \times \frac{(263+80)}{700} \approx 0.8325$$

boostcamp ai tech

인과관계 추론: 예제



	Overall	Z = 0: Patients with small stones	Z = 1: Patients with large stones
Treatment *a*: Open surgery	78% (273/350)	**93%** (81/87)	**73%** (192/263)
Treatment *b*: Percutaneous nephrolithotomy	**83%** (289/350)	87% (234/270)	69% (55/80)

출처: Elements of Causal Inference, Peters et al.

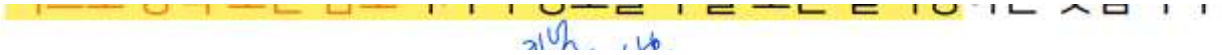
$$P^{\mathfrak{G}_b}(R=1) = \sum_{z\in\{0,1\}} P^{\mathfrak{G}}(R=1 \mid T=b, Z=z)P^{\mathfrak{G}}(Z=z)$$

조건부확률로 계산한 치료효과와 정반대의 결과가 나오게 된다

$$= \frac{234}{270} \times \frac{(87+270)}{700} + \frac{55}{80} \times \frac{(263+80)}{700} \approx 0.7789$$

boostcamp ai tech

▾ (AI Math 9강) CNN 첫걸음



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

- 커널 : 정의역 내에서 움직여도 변하지 않고 신호에 국소적으로 적응한다,
- 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

Convolution 연산 이해하기

고정된 가중치 행렬

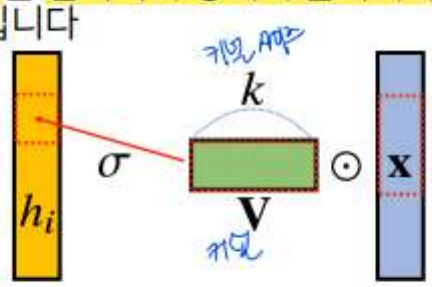
- Convolution 연산은 이와 달리 커널(kernel)을 입력벡터 상에서 움직여가면서 선형모델과 합성함수가 적용되는 구조입니다

$$h_i = \sigma\left(\sum_{j=1}^{k} V_j x_{i+j-1}\right)$$

커널 사이즈 k

활성함수 σ

가중치 행렬 V_j



모든 i 에 대해 적용되는 커널은 $\mathbf{V}$ 로 같고 커널의 사이즈만큼 $\mathbf{x}$ 상에서 이동하면서 적용합니다

Convolution 연산 이해하기

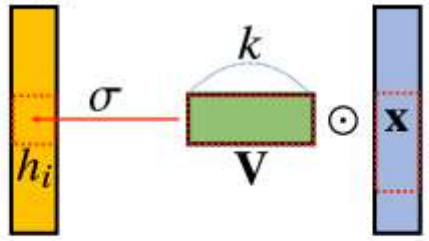
- Convolution 연산은 이와 달리 커널(kernel)을 입력벡터 상에서 움직여가면서 선형모델과 합성함수가 적용되는 구조입니다

$$h_i = \sigma\left(\sum_{j=1}^{k} V_j x_{i+j-1}\right)$$

커널 사이즈 k

활성함수 σ

가중치 행렬 V_j



활성화 함수를 제외한 Convolution 연산도 선형변환에 속합니다

Convolution 연산 이해하기

- Convolution 연산의 수학적인 의미는 신호(signal)를 커널을 이용해 국소적으로 증폭 또는 감소시켜서 정보를 추출 또는 필터링하는 것입니다

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

Convolution 을 수식으로만 이해하는 것은 매우 어렵습니다

Convolution 연산 이해하기

- Convolution 연산의 수학적인 의미는 신호(signal)를 커널을 이용해 국소적으로 증폭 또는 감소시켜서 정보를 추출 또는 필터링하는 것입니다

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

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

CNN 에서 사용하는 연산은 사실 convolution 이 아니고 cross-correlation 이라 부릅니다

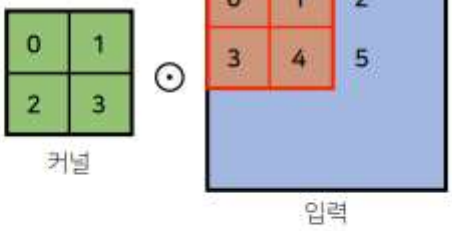
2차원 Convolution 연산 이해하기

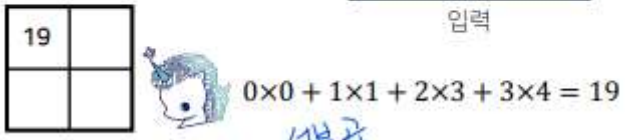
- 2D-Conv 연산은 이와 달리 커널(kernel)을 입력벡터 상에서 움직여가면서 선형모델과 합성함수가 적용되는 구조입니다

$$[f * g](i,j) = \sum_{p,q} f(p,q)g(i+p, j+q)$$

커널

입력





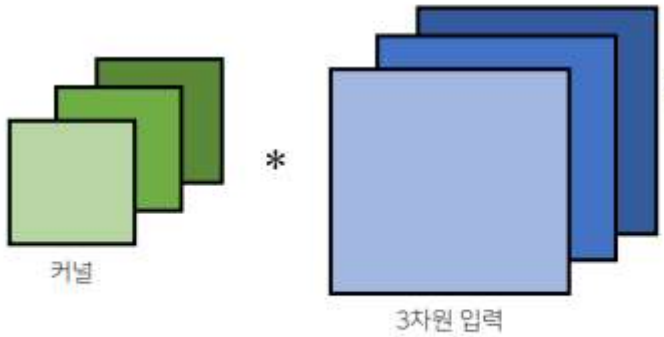
- 입력 크기를 (H, W), 커널 크기를 (K_H, K_W), 출력 크기를 (O_H, O_W) 라 하면 출력 크기는 다음과 같이 계산합니다

$$O_H = H - K_H + 1$$
$$O_W = W - K_W + 1$$

가령 28x28 입력을 3x3 커널로 2D-Conv 연산을 하면 26x26 이 된다

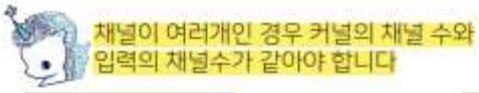
2차원 Convolution 연산 이해하기

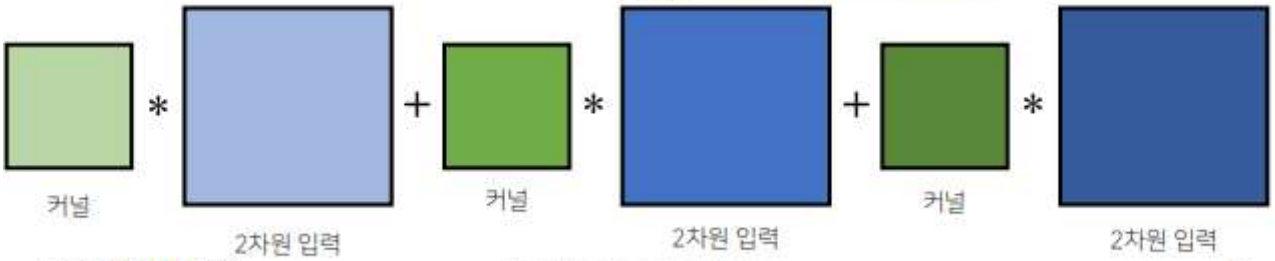
- 채널이 여러개인 2차원 입력의 경우 2차원 Convolution 을 채널 개수만큼 적용한다고 생각하면 됩니다



2차원 Convolution 연산 이해하기

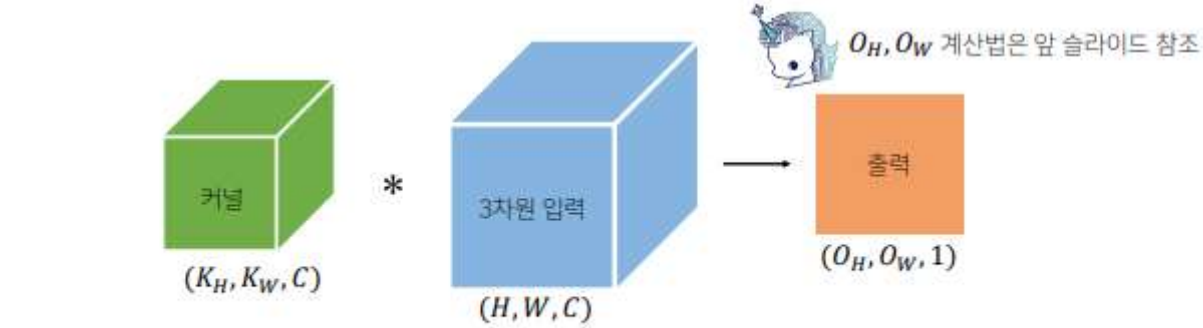
- 채널이 여러개인 2차원 입력의 경우 2차원 Convolution 을 채널 개수만큼 적용한다고 생각하면 됩니다





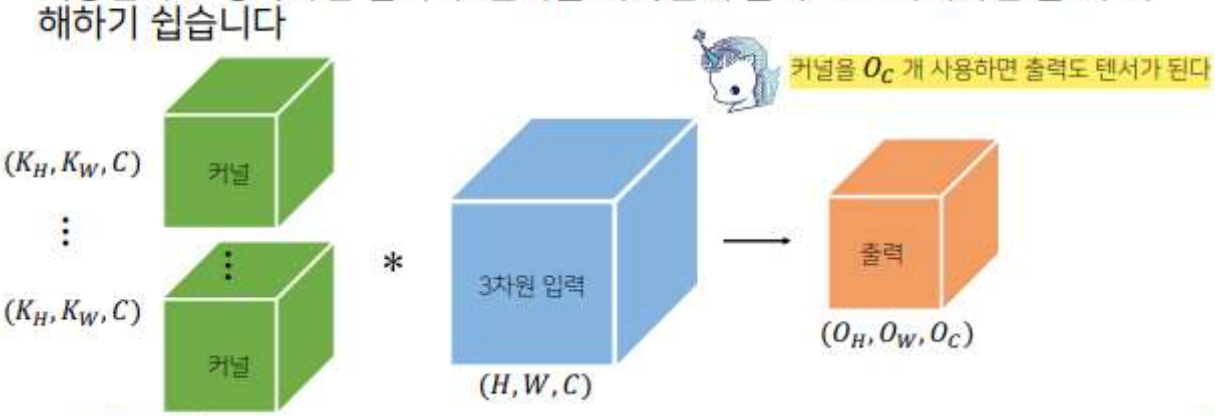
2차원 Convolution 연산 이해하기

- 채널이 여러개인 2차원 입력의 경우 2차원 Convolution 을 채널 개수만큼 적용한다고 생각하면 됩니다. 텐서를 직육면체 블록으로 이해하면 좀 더 이해하기 쉽습니다



2차원 Convolution 연산 이해하기

- 채널이 여러개인 2차원 입력의 경우 2차원 Convolution 을 채널 개수만큼 적용한다고 생각하면 됩니다. 텐서를 직육면체 블록으로 이해하면 좀 더 이해하기 쉽습니다



Convolution 연산의 역전파 이해하기

선형변환 임

- Convolution 연산은 커널이 모든 입력데이터에 공통으로 적용되기 때문에 역전파를 계산할 때도 convolution 연산이 나오게 됩니다

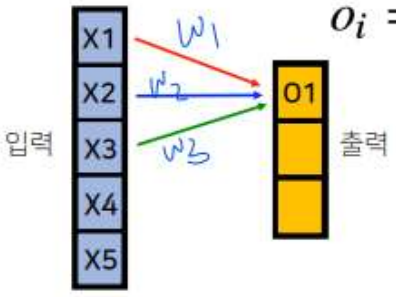
$$\frac{\partial}{\partial x}[f * g](x) = \frac{\partial}{\partial x}\int_{\mathbb{R}^d} f(y)g(x-y)\mathrm{d}y$$
$$= \int_{\mathbb{R}^d} f(y)\frac{\partial g}{\partial x}(x-y)\mathrm{d}y$$
$$= [f * g'](x)$$

Discrete 일 때도 마찬가지로 성립한다

Convolution 연산의 역전파 이해하기

- Convolution 연산은 커널이 모든 입력데이터에 공통으로 적용되기 때문에 역전파를 계산할 때도 convolution 연산이 나오게 됩니다

$$o_i = \sum_j w_j x_{i+j-1}$$



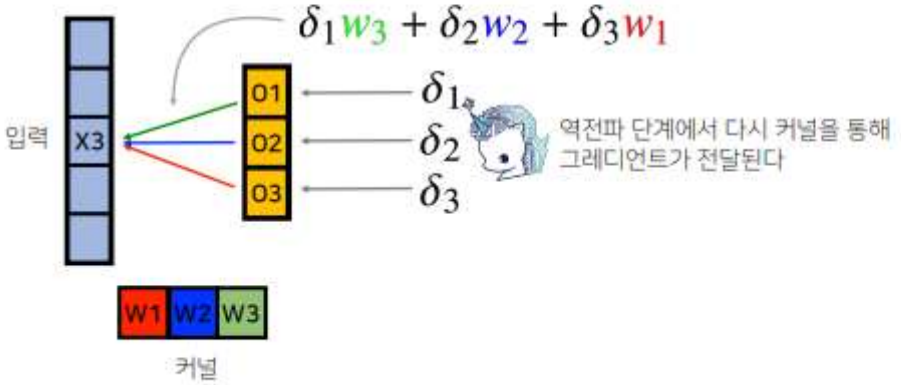
그림으로 이해하면 훨씬 쉽습니다

커널

= 5-3+1 개수

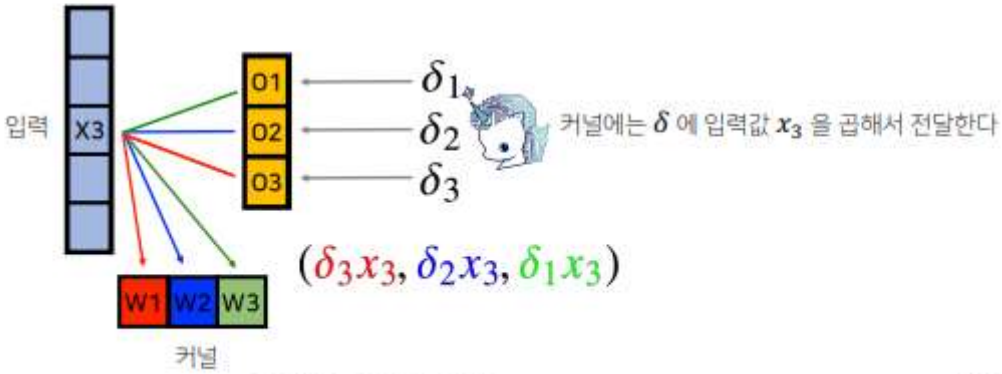
Convolution 연산의 역전파 이해하기

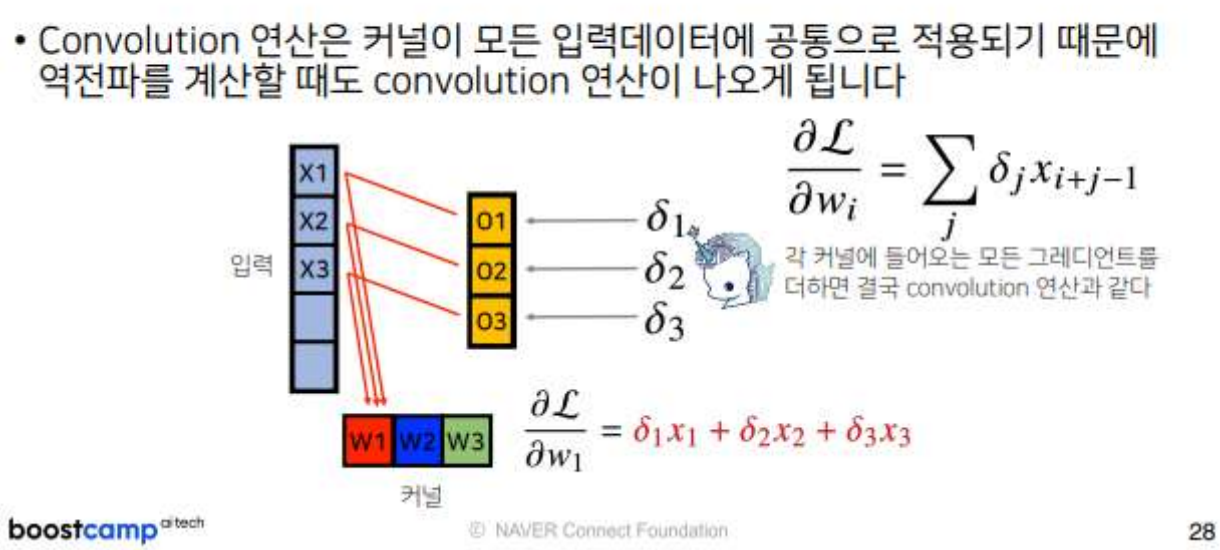
- Convolution 연산은 커널이 모든 입력데이터에 공통으로 적용되기 때문에 역전파를 계산할 때도 convolution 연산이 나오게 됩니다



Convolution 연산의 역전파 이해하기

- Convolution 연산은 커널이 모든 입력데이터에 공통으로 적용되기 때문에 역전파를 계산할 때도 convolution 연산이 나오게 됩니다





(AI Math 10강) RNN 첫걸음

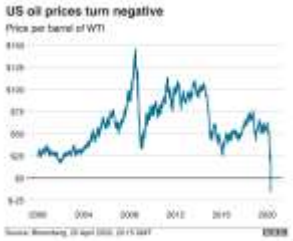
시계열 데이터 : 소리, 문자열, 주가 등의 데이터

- 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.

PPT 필기

시퀀스 데이터 이해하기

- 소리, 문자열, 주가 등의 데이터를 시퀀스(sequence) 데이터로 분류합니다
- 시퀀스 데이터는 독립동등분포(i.i.d.) 가정을 잘 위배하기 때문에 순서를 바꾸거나 과거 정보에 손실이 발생하면 데이터의 확률분포도 바뀌게 됩니다



$$X_1, \ldots, X_t, \ldots$$

과거 정보 또는 앞뒤 맥락 없이 미래를 예측하거나 문장을 완성하는 건 불가능하다

시퀀스 데이터를 어떻게 다루나요?

- 이전 시퀀스의 정보를 가지고 앞으로 발생할 데이터의 확률분포를 다루기 위해 조건부확률을 이용할 수 있습니다

$$\begin{aligned} P(X_1, \ldots, X_t) &= P(X_t | X_1, \ldots, X_{t-1}) P(X_1, \ldots, X_{t-1}) \\ &= P(X_t | X_1, \ldots, X_{t-1}) P(X_{t-1} | X_1, \ldots, X_{t-2}) \times \\ &\qquad \times P(X_1, \ldots, X_{t-2}) \\ &= \prod_{s=1}^{t} P(X_s | X_{s-1}, \ldots, X_1) \end{aligned}$$

요 기호는 $s = 1, \ldots, t$ 까지 모두 곱하라는 기호입니다

시퀀스 데이터를 어떻게 다루나요?

- 이전 시퀀스의 정보를 가지고 앞으로 발생할 데이터의 확률분포를 다루기 위해 조건부확률을 이용할 수 있습니다
- 시퀀스 데이터를 다루기 위해선 길이가 가변적인 데이터를 다룰 수 있는 모델이 필요합니다

τ : 적당한 데이터 크기의 고정된 값

$$X_t \sim P(X_t|X_{t-1}, \ldots, X_1)$$

$$X_{t+1} \sim P(X_{t+1}|X_t, X_{t-1}, \ldots, X_1)$$

고정된 길이 τ 만큼의 시퀀스만 사용하는 경우 $AR(\tau)$
(Autoregressive Model) 자기회귀모델이라고 부릅니다

시퀀스 데이터를 어떻게 다루나요?

- 이전 시퀀스의 정보를 가지고 앞으로 발생할 데이터의 확률분포를 다루기 위해 조건부확률을 이용할 수 있습니다
- 시퀀스 데이터를 다루기 위해선 길이가 가변적인 데이터를 다룰 수 있는 모델이 필요합니다

$$X_t \sim P(X_t|X_{t-1}, \ldots, X_1) \rightarrow H_t$$

$$X_{t+1} \sim P(X_{t+1}|X_t, X_{t-1}, \ldots, X_1) \rightarrow H_{t+1}$$

또 다른 방법은 바로 이전 정보를 제외한 나머지 정보들을
H_t 라는 잠재변수로 인코딩해서 활용하는 잠재 AR 모델입니다

시퀀스 데이터를 어떻게 다루나요?

- 이전 시퀀스의 정보를 가지고 앞으로 발생할 데이터의 확률분포를 다루기 위해 조건부확률을 이용할 수 있습니다
- 시퀀스 데이터를 다루기 위해선 길이가 가변적인 데이터를 다룰 수 있는 모델이 필요합니다

길이 고정

$$X_t \sim P(X_t|X_{t-1}, H_t)$$

$$X_{t+1} \sim P(X_{t+1}|X_t, H_{t+1})$$

$$H_t = \text{Net}_\theta(H_{t-1}, X_{t-1})$$

잠재변수 H_t 를 신경망을 통해 반복해서 사용하여
시퀀스 데이터의 패턴을 학습하는 모델이 RNN 입니다

Recurrent Neural Network 을 이해하기

- 가장 기본적인 RNN 모형은 MLP 와 유사한 모양입니다

$\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ 은 시퀀스와 상관없이 불변인 행렬입니다

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$
$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

잠재변수 / 활성화함수 / 가중치행렬 / bias

Recurrent Neural Network 을 이해하기

- 가장 기본적인 RNN 모형은 MLP 와 유사한 모양입니다

이 모델은 과거의 정보를 다룰 수 없습니다

$W^{(1)}, W^{(2)}$: 불변 행렬

$$\mathbf{O}_t = \mathbf{H}_t\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$
$$\mathbf{H}_t = \sigma(\mathbf{X}_t\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

잠재변수 / 활성화함수 / 가중치행렬 / bias

$\mathbf{O}_t$

$\mathbf{H}_t$

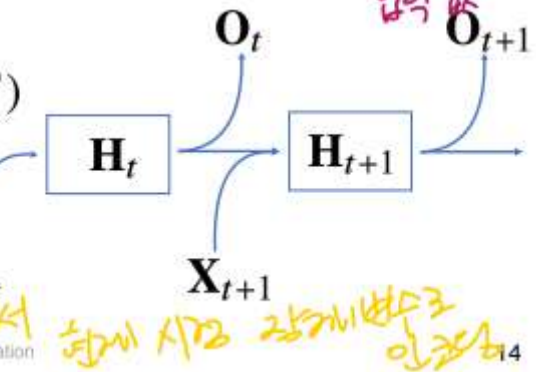
현재 시점 데이터만 포함 → $\mathbf{X}_t$

Recurrent Neural Network 을 이해하기

- 가장 기본적인 RNN 모형은 MLP 와 유사한 모양입니다
- RNN 은 이전 순서의 잠재변수와 현재의 입력을 활용하여 모델링합니다

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$
$$\mathbf{H}_t = \sigma(\mathbf{X}_t \mathbf{W}_X^{(1)} + \mathbf{H}_{t-1} \mathbf{W}_H^{(1)} + \mathbf{b}^{(1)})$$

잠재변수인 $\mathbf{H}_t$ 를 복제해서 다음 순서의 잠재변수를 인코딩하는데 사용합니다



Recurrent Neural Network 을 이해하기

- 가장 기본적인 RNN 모형은 MLP 와 유사한 모양입니다
- RNN 은 이전 순서의 잠재변수와 현재의 입력을 활용하여 모델링합니다
- RNN 의 역전파는 잠재변수의 연결그래프에 따라 순차적으로 계산합니다

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$
$$\mathbf{H}_t = \sigma(\mathbf{X}_t \mathbf{W}_X^{(1)} + \mathbf{H}_{t-1} \mathbf{W}_H^{(1)} + \mathbf{b}^{(1)})$$

이를 Backpropagation Through Time (BPTT) 이라 하며 RNN 의 역전파 방법이다