

## 데이터 마이닝(data mining)

데이터베이스로부터 새로운 통찰력을 만드는 것

Data + Mining

용어

Data: 데이터

Mining: 채굴, 채광

즉, 데이터 마이닝이란 광산에서 광석을 캐내는 것에 비유한 것으로, 금광석에 극히 미량으로 포함된 금을 여러 단계를 거쳐 추출하듯이 수많은 데이터의 산에서 가치있는 유용한 정보를 찾아내는 것이다.

절차

데이터 추출 -> 데이터 정제 -> 데이터 변경 -> 데이터 분석 -> 데이터 해석  
-> 보고서 작성

## 인공지능(artificial intelligence)

컴퓨터가 인간과 같은 지능적인 행동을 할 수 있게 해주는 기법들을 연구

## 딥러닝(deep learning)

인공신경망(Artificial Neural Networks(ANN))을 여러 층 쌓아 올린 기법을 의미한다.

딥러닝은 인공 신경망을 단계적으로 깊게 쌓아 올린다.

깊게 쌓아 올림으로써 얻는 효과는 데이터의 특징을 단계별로 학습할 수 있다는 점이다.

딥러닝 주목 이유

1. 빅데이터를 구할 수 있게 된 환경 조성
2. GPU를 필두로한 컴퓨팅 파워의 발전
3. 새로운 딥러닝 알고리즘의 개발

## 머신러닝(machine learning)

톰 마첼 교수는 머신러닝을 어떤 문제(Task) T에 관련된 경험(Experience) E로부터 성과 측정 지표(Performance Measure) P를 가지고 학습(learn)을 진행하는 컴퓨터 프로그램을 말한다고 정의한다.

머신러닝은 인공지능 AI라는 연구분야의 방법론 중의 하나이다.

뛰어난 분야

기존 솔루션으로는 많은 수동 조정과 규칙이 필요한 문제: 하나의 머신러닝 모델이 코드를 간단하고 더 잘 수행되도록 할 수 있다.

유동적인 환경: 머신러닝 시스템은 새로운 데이터에 적응할 수 있다.

복잡한 문제와 대량의 데이터에서 통찰 얻기

종류

Label에 의한 구분

즉 데이터에 대한 정답(Label)이 있는가

지도 학습(supervised learning)

훈련 데이터에 Label이 존재

분류와 회귀 작업이 전형적인 지도학습이다.

일부 회귀 알고리즘은 분류에 사용할 수도 있다 - 로지스틱 회귀

회귀는 왜 지도학습의 예인가?

시스템을 훈련시키려면 예측 변수와 레이블이 포함된 데이터가 필요하므로

ex) 중고차의 주행거리와 중고차 가격

### 비지도 학습(unsupervised learning)

훈련 데이터에 Label이 없음

군집, 시각화, 차원축소, 연관 규칙 학습이 대표적인 예이다.

### 준지도 학습(semisupervised learning)

훈련 데이터에 Label이 일부 존재

심층 신경망 DBN이 대표적인 예이다.

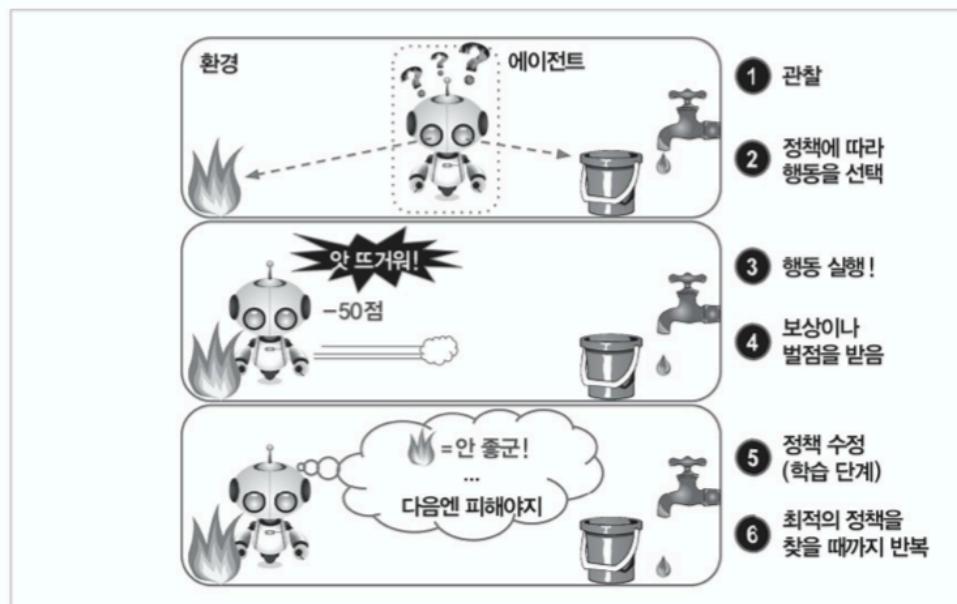
### 강화 학습(Reinforcement Learning)

학습 하는 시스템을 에이전트라 부름

환경을 관찰해서 행동을 실행하고 그 결과로 보상 또는 부정적 보상에 해당하는 벌점을 얻는다.

시간이 지나면서 가장 큰 보상을 얻기 위해 정책(policy)이라고 부르는 최상의 전략을 스스로 학습한다.

그림 1-12 강화 학습



### 점진적으로 학습할 수 있는가에 의한 구분

입력 데이터의 스트림(stream)으로부터 점진적으로 학습할 수 있는가?

#### 배치 학습(batch learning)

가용한 데이터를 모두 사용해 훈련시키는 학습

시스템이 점진적으로 학습할 수 없다.

이 방식은 시간과 자원을 많이 소모하므로 보통 오프라인에서 수행된다. 그래서 오프라인 학습(offline learning)이라고도 한다.

먼저 시스템을 학습시키고 그런 다음 제품 시스템에 적용하면 더 이상의 학습없이 실행된다.

#### 점진적 학습(incremental learning), 온라인 학습(online learning)

미니 배치라 부르는 작은 묶음 단위로 시스템을 훈련시킨다.

매 학습 단계가 빠르고 비용이 적게 들어 시스템은 데이터가 도착하는 대로 즉시 학습할 수 있다.(이래서 점진적 학습(incremental learning)이라고도 한다)

온라인 학습은 연속적으로 데이터를 받고 빠른 변화에 스스로 적응해야 하는 시스템에 적합하다.

### 문제점

온라인 학습에서 가장 큰 문제점은 시스템에 나쁜 데이터가 주입되었을 때 시스템 성능이 점진적으로 감소한다는 것이다.

이런 위험을 줄이려면 시스템을 면밀히 모니터링하고 성능 감소가 감지되면 즉각 학습을 중지시켜야 한다.

가능하면 이전 운영 상태로 되돌린다.

### 중요한 점

온라인 학습 시스템에서 중요한 파라미터 하나는 변화하는 데이터에 얼마나 빠르게 적응할 것인지이다.

이를 학습률이라고 한다.

학습률을 높게 하면 시스템이 데이터에 빠르게 적응하지만 예전 데이터를 금방 잊어버릴 것이다.

반대로 학습률이 낮으면 시스템의 관성이 더 커져서 더 느리게 학습된다.

하지만 새로운 데이터에 있는 잡음이나 대표성 없는 데이터 포인트에 덜 민감해진다.

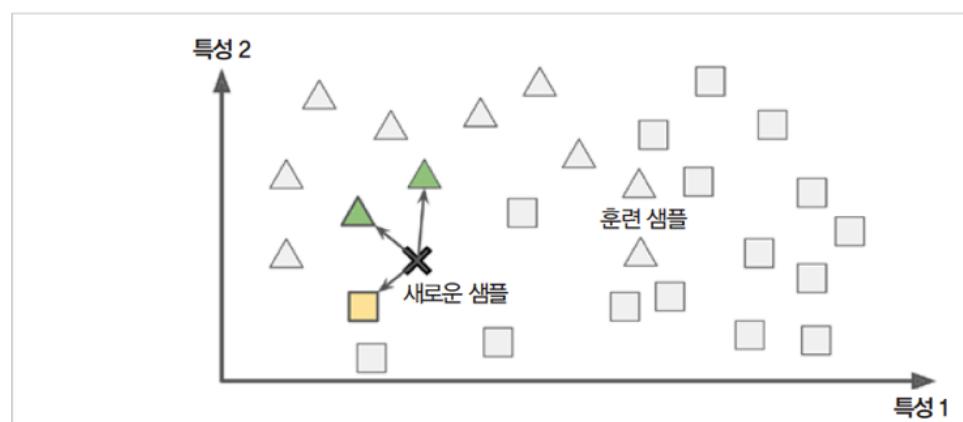
### 일반화 과정의 차이에 의한 구분

머신러닝 시스템은 어떻게 일반화되는가에 따라 분류할 수 있다.

#### 사례기반학습(instance-based learning)

가장 단순한 형태의 학습은 단순히 기억하는 것이다.

시스템이 사례를 기억하고 학습한다. 그리고 유사도 측정을 사용해 새로운 데이터에 일반화한다.



#### 모델기반학습(model based learning)

샘플로부터 일반화시키는 다른 방법은 이 샘플들의 모델을 만들어 예측에 사용하는 것이다.

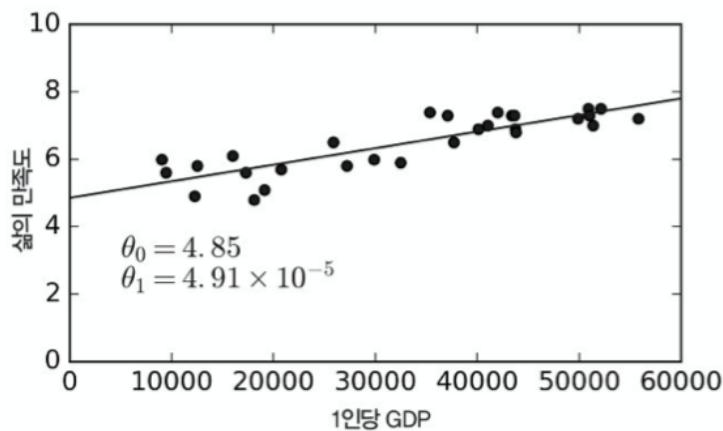
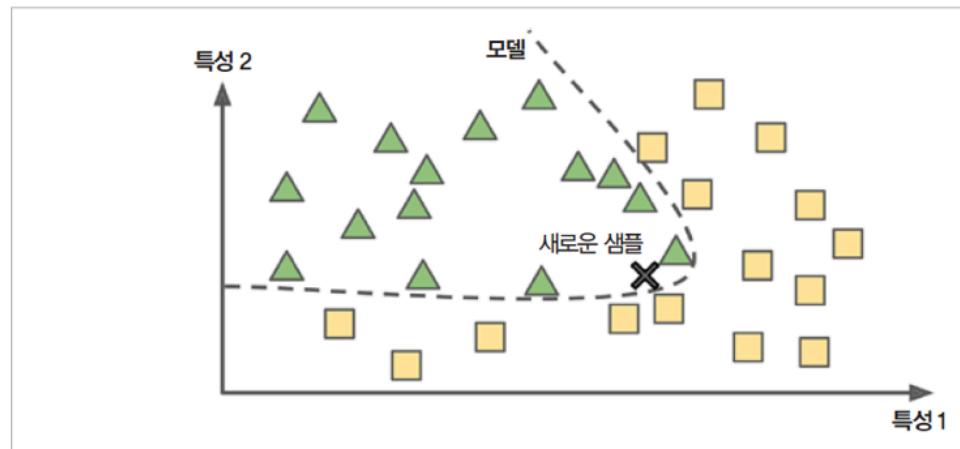


그림 1-16 모델 기반 학습



### 도전과제

#### 충분하지 않은 양의 훈련 데이터

머신러닝을 하려면 충분한 데이터가 필요하다.

충분하지 않은 데이터 가지고는 아직 아무것도 할수 없음

#### 대표성이 없는 훈련 데이터

일반화가 잘되려면 우리가 일반화하기 원하는 새로운 사례를 훈련 데이터가 잘 대표 하는 것이 중요하다

#### 샘플링 잡음(Sampling noise)

우연에 의한 대표성이 없는 데이터

#### 샘플링 편향(Sampling bias)

표본 추출이 잘못되어 대표성을 띠지 못하는 데이터

#### 낮은 품질의 데이터

훈련 데이터가 이상치, 잡음으로 가득하다면 머신러닝 시스템이 내재된 패턴을 찾기 어려워 잘 작동하지 않을 것이다.

그렇기 때문에 정제에 힘을 써야 한다.

- NA값을 무시할지

- 이상치 고치기

#### 관련 없는 특성

훈련 데이터에 관련 없는 특성이 적고 관련 있는 특성이 충분해야 시스템이 학습할 수 있을 것이다.

성공적인 머신러닝 프로젝트의 핵심요소는 훈련에 사용할 좋은 특성들을 찾는것이며 이 과정을 특성 공학(Feature Engineering)이라 한다.

과적합  
과소적합

## Feature Engineering

일반화가 잘되려면 우리가 일반화하기 원하는 새로운 사례를 훈련 데이터가 잘 대표하는 것  
이 중요하다.

## Performance Measure로 왜 손실함수를 설정하는가?

'정확도'라는 지표를 놔두고 왜 손실 함수값을 사용하는가?

정확도는 매개변수의 미소한 변화에는 거의 반응을 보이지 않고, 반응이 있더라도 그 값이  
불연속적으로 값자기 변화하기 때문이다.

정확도가 32%일때 만약 정확도가 지표였다면 가중치 매개변수의 값을 조금 바꾼다고 해  
도 정확도는 그대로 32%일것이다.

즉 매개변수를 약간만 조정해서는 정확도가 개선되지 않고 개선된다 하더라도 33%나 34%  
처럼 띄엄띄엄한 값으로 바뀌어버린다.

## 기본적인 과정

데이터 수집 -> 데이터 탐색 및 준비 -> 데이터에 대한 모델 훈련 -> 모델 성  
능 평가 -> 모델 성능 개선

### 데이터 수집

학습 자료를 수집

### 데이터 탐색과 준비

데이터의 미묘한 차이를 파악

엉망인 데이터를 교정하거나 정리하고, 필요 없는 데이터는 제거하고, 학습  
자가 기대하는 입력에 맞춰서 데이터를 다시 코드화

### 모델 훈련

적절한 알고리즘 선택

알고리즘이 모델 형태로 데이터를 표현한다.

### 모델 평가

자신의 경험으로부터 학습을 얼마나 잘하는지 평가

사용되는 모델의 종류에 따라 테스트 데이터셋으로 모델의 정확도를 평가하거  
나 대상 응용에 특화된 성능 척도를 개발

### 모델 개선

추가 데이터로 보충하거나, 2단계로 회귀 혹은 다른 모델로 전환

## 데이터 마이닝과의 차이점

데이터 마이닝은 문제를 해결하기 위해 컴퓨터에게 사람이 사용할 패턴을 찾게 가  
르치는데 집중하는 반면,

머신러닝은 문제를 해결하기 위해 컴퓨터에게 데이터의 사용법을 가르친다.

포함관계는 머신러닝이 더 큼

## 한계

학습한 엄격한 파라미터의 범위 밖에서는 추정의 유연성이 거의 없고 상식조차 없  
다.

즉, 자신이 학습한 데이터 정도 밖에 우수하지 않다.

### 학습과정

#### 데이터 저장소(data storage)

관찰(observation), 기억(memory), 회상(recall)을 활용해 향후 추론을 위한 사실적 기반을 제공한다.

데이터를 저장만 하는것의 문제점

- 지식들이 어떻게 관련이 있는지, 저장된 정보를 어떻게 사용할지에 대한 전략을 세우고 대표적인 지식을 몇 개 암기하며 시간을 선택적으로 사용하는 것이 자료를 암기하는 것보다 더 나은 방법이다. (높은 수준의 이해가 없다면 지식은 오직 기억에 한정된다.)

#### 추상화(abstraction)

저장된 데이터를 넓은 표현과 개념으로 변환한다.

데이터에 의미를 부여하는 작업

추상화된 연결은 원시 센서 정보가 의미 있는 통찰로 변환되게 도와주는 논리 구조가 형성된 상태인 지식표현(knowledge representation)의 기반이다.

기계의 지식 표현 과정에서 컴퓨터는 저장된 원시 데이터를 모델(model)을 이용해서 요약한다.

#### 모델(model)

모델은 데이터 안의 패턴을 명시적으로 표현한 것이다.

학습된 모델 자체로 새로운 데이터를 제공하지는 않지만, 새로운 지식을 산출한다.

- 기저 데이터(underlying data)에 가정된 구조를 도입하는 것은 데이터 요소 간의 연관 관계에 대한 개념을 가정하므로써 보이지 않는 것에 통찰을 부여한다.
- 즉, 가정이 통찰을 부여한다.

#### 훈련(training)

모델을 데이터셋에 맞추는 과정

모델이 훈련됐을 때 데이터는 원래의 정보를 요약한 추상화된 형태로 변환된다.

#### 모델의 선택

일반적으로 모델의 선택을 기계에 맡기지 않는다.

학습할 작업과 보유한 데이터가 모델의 선택에 영향을 미친다.

#### 추상화 과정의 문제점

추상화의 생성이 제한되지 않는다면 학습자는 진행을 할 수 없다.

실행 가능한 통찰력 없이 정보량만 풍부한 학습자는 시작 지점에 멈춰 있게 된다.

#### 일반화(generalization)

#### 평가(evaluation)

학습된 지식의 효율성을 측정하고 잠재적인 개선 사항을 알려주는 피드백 매커니즘을 제공한다.

학습자는 각자 약점을 갖고 있으며 모두를 지배하는 단일 학습 알고리즘은 없다.

그러므로 일반화 과정의 최종단계는 편향이 있음에도 불구하고 학습자의 성공을 평가하거나 측정하고 필요하다면 추가 훈련을 통지하는 것이다.

### 잡음(noise)

데이터에서 설명되지 않거나 설명할 수 없는 변형을 나타내는 용어이다.

잡음을 모델링하려고 하면 과적합 문제가 발생한다.

잡음이 있는 데이터는 정의로 설명되지 않기 때문에 잡음을 설명하려고 하면 새로운 경우에 일반화를 잘하지 못하는 잘못된 결과가 만들 어진다.

또한 잡음을 설명하려고 노력하면 일반적으로 학습자가 식별하려는 진짜 패턴은 빠져있는 좀 더 복잡한 모델이 생성될 것이다.

## 입력 데이터 타입

머신러닝 작업에는 가용한 방법의 편향에 입력 데이터의 특성을 맞추는 것이 포함된다.

### 관측 단위(unit of observation)

학습을 하기 위해 관심 있는 측정 속성을 갖는 가장 작은 엔티티(entity)를 설명하는데 사용

여러 관측 단위와 관측 단위의 속성을 저장하고 있는 데이터셋은 다음의 요소로 구성되는 데이터의 모음으로 생각해볼 수 있다.

ex)

암환자 식별 –

관측 단위 – 환자

예시 – 암환자의 무작위 표본

특징 – 조직 검사 세포의 이름 마커와 몸무게, 키, 혈압과 같은 환자의 특성

### 예시(Example)

속성이 기록돼 있는 관측 단위의 인스턴스(instance)

### 특징(Features)

학습에 유용할 수 있는 예시의 기록된 수정

feature를 가지고 고차함수의 model을 만들고, 특정 차수의 feature만 뽑아낸다고 할때,  $O(n^2)$ 으로 feature가 늘어난다. ( $n = \text{feature의 개수}$ )

이때 보통  $n^2 / 2$  만큼 뽑아진다.

### 수치(numeric)

특징이 숫자로 측정된 특성을 나타내는 것

### 범주(categorical), 명목(nominal)

특징이 범주의 집합으로 이뤄진 속성

### 순위(ordinal)

순서 목록(ordered list)에 속하는 범주를 갖는 명목 변수

## 머신러닝 알고리즘 타입

### 예측 모델(predictive model)

데이터셋의 값을 이용해서 새로운 값을 예측할 때 사용

### 지도학습(supervised learning)

예측 모델은 학습 대상과 학습 방법에 대한 명확한 지침이 주어지기 때문에 예측모델을 훈련하는 과정을 지도학습(supervised learning)이라 한다.

지도 학습 알고리즘은 데이터셋이 주어지면 함수(모델)를 최적화해 목표 출력을 만드는 특징 값의 조합을 찾는다.

#### 분류(classification)

예제가 속하는 범주를 예측하는 머신러닝 작업

#### 클래스(class)

분류에서 예측해야 할 목표 특징

클래스는 두 개 이상의 레벨을 가질 수 있다.

#### level

범주

레벨은 순위이거나 아닐수 있다.

#### 수치 예측(numeric prediction)

수치 데이터를 예측

선형 회귀 모델을 입력 데이터에 맞추는 것

### 서술 모델(descriptive model)

새롭고 흥미로운 방식으로 데이터를 요약해서 얻은 통찰로 이득을 얻는 작업에 사용된다.

관심 있는 목표를 예측하는 예측 모델과 달리 서술모델에서는 한 특징이 다른 특징들보다 중요하지 않다.

### 자율학습(unsupervised learning)

실제 학습해야 할 목표가 없기 때문에 서술 모델을 훈련하는 과정을 자율학습(unsupervised learning)이라 한다.

#### 패턴 발견(pattern discovery)

데이터 내에서 유용한 연관을 찾아내는 데 사용

#### 군집화(clustering)

데이터셋을 동질 그룹(homogeneous group)으로 분리하는 서술 모델링 작업

### 메타 학습자(meta-learning)

효과적으로 학습하는 방법을 학습하는데 주력한다.

### 준 지도 학습(Semi-Supervised Learning)

기계 학습의 한 범주로 목표값이 표시된 데이터와 표시되지 않은 데이터를 모두 훈련에 사용하는 것을 말한다.

대개의 경우 이러한 방법에 사용되는 훈련 데이터는 목표값이 표시된 데이터가 적고 표시되지 않은 데이터를 많이 갖고 있다.

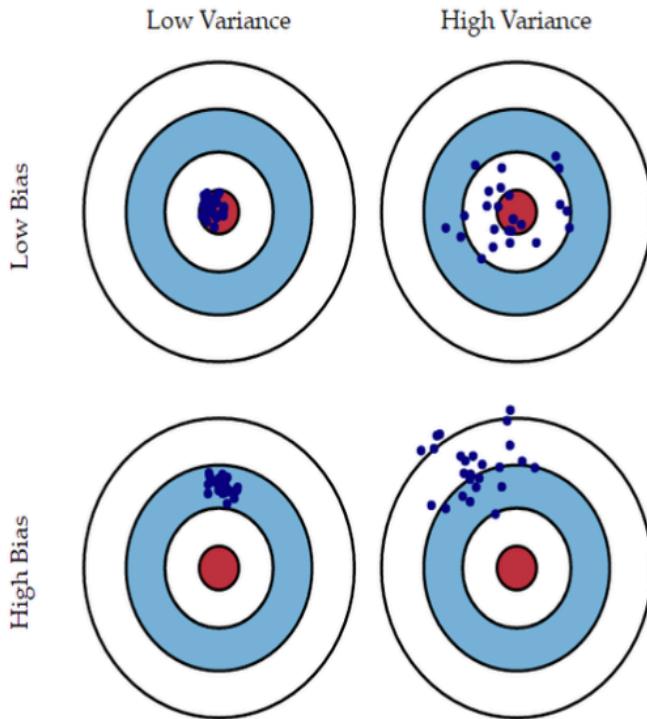
많은 기계 학습 연구자들이 목표값이 없는 데이터에 적은 양의 목표값을 포함한 데이터를 사용할 경우 학습 정확도가 상당히 좋아짐을 확인했다.

이러한 훈련 방법이 사용되는 이유는 목표값을 포함한 데이터를 얻기 위해서는 훈련된 사람의 손을 거쳐야 하기 때문이고 그 비용이 감당할 수 없

을만큼 클 수 있기 때문이다.

### 편향과 분산

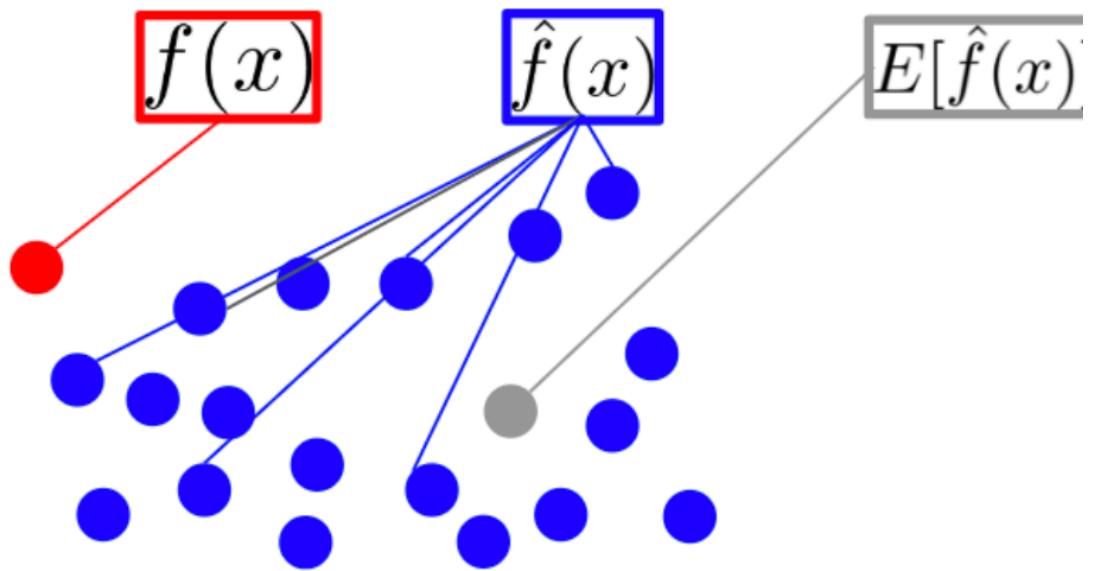
예측값들과 정답이 대체로 멀리 떨어져 있으면 결과의 편향(bias)이 높다고 말하고, 예측값들이 자기들끼리 대체로 멀리 떨어져 있으면 결과의 분산(variance)이 높다고 말한다.



가운데 빨간점을 사람이 정해준 '정답'이라고 한다면, 파랗게 여러 번 찍힌 점이 컴퓨터가 예측한 값이라 했을때

1. 왼쪽 상단 과녁은 -> 엄청 잘맞춘다!
  - 예측값들이 대체로 정답 근방에서 왔다갔다 한다. -> 편향이 낮다.
  - 예측값들끼리 서로 몰려 있다. -> 분산이 낮다.
2. 오른쪽 상단 과녁은 -> 맞추긴 하는데 덜 잘맞춘다.
  - 예측값들이 대체로 정답 근방에서 왔다갔다 한다 -> 편향이 낮다.
  - 예측값들끼리 서로 흩어져 있다. -> 분산이 높다.
3. 왼쪽 하단 과녁은 -> 잘 못맞추는데 경향성이 있다?
  - 예측값들이 대체로 정답으로부터 멀어져 있다. -> 편향이 높다.
  - 예측값들끼리 서로 몰려 있다. -> 분산이 낮다.
4. 오른쪽 하단 과녁은 -> 못맞추면서 경향성도 없다.
  - 예측값들끼리 대체로 정답으로부터 멀어져 있다. -> 편향이 높다.
  - 예측값들끼리 서로 흩어져 있다. -> 분산이 높다.

### 수식표현



### 편향의 수식 표현

빨간 점과 회색 점에 대한 이야기이다.

편향은 예측값이 정답과 얼마나 다른지를 표현한다.

$f(x)$ 는 실제 값이고  $\hat{f}(x)$ 는 예측값이고,  $x$ 는 데이터이다.

Bias: How much predicted value differ from true values

$$\left( \underbrace{E[\hat{f}(x)]}_{\text{average predicted value}} - \underbrace{f(x)}_{\text{true value}} \right)^2$$

### 분산의 수식 표현

빨강 점과 회색점에 대한 이야기이다.

분산은 예측값들이 얼마나 서로 흩어져 있는가를 표현한다.

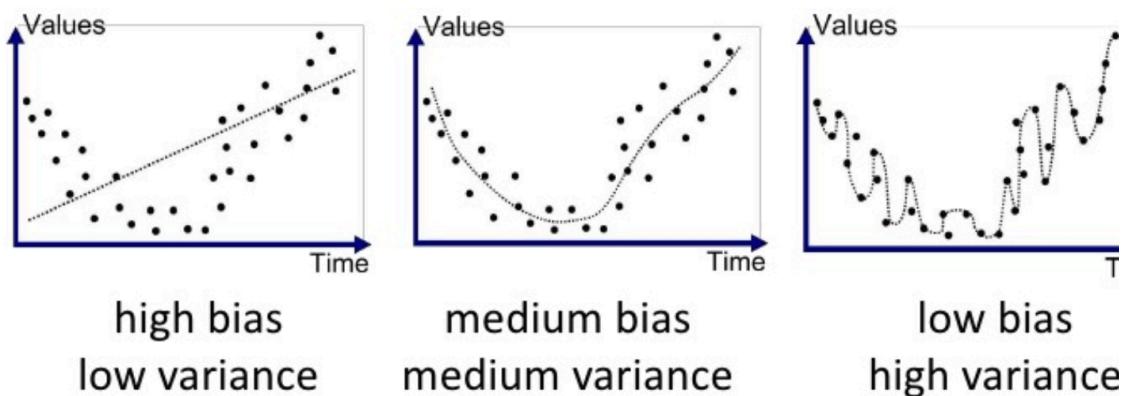
Variance: How predictions made on the same value vary on different realizations of the model

$$\left| \frac{E[\hat{f}(x)] - E[\hat{f}(x)]}{\text{predicted value}} \right|^2$$

첫번째 그림은 데이터들이 모델과 멀어져 있으므로 편향이 높고, 모델이 내놓는 값들끼리는 별로 떨어져 있지 않게 되므로 분산이 낮다

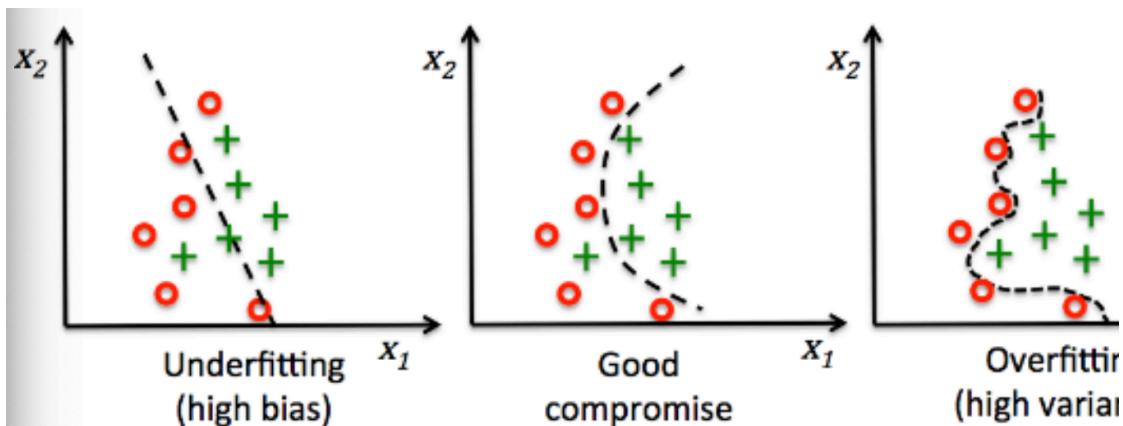
세번째 그림은 정답들이 모델과 아주 붙어 있으므로 편향이 낮고, 모델이 내놓는 값들끼리는 매우 흩어져 있으므로 분산이 높다.

두번째는 적당하다.



첫번째 그림은 데이터들이 모델과 멀어져 있으므로 편향이 높다.  
모델이 내놓는 값을 끼리는 별로 떨어져 있지 않게되므로 분산은 낫다.

세번째 그림은 정답이 모델과 아주 불어있으므로 편향이 낮고, 모델이 내놓는 값들끼리는 매우 흩어져 있게 되므로 분산이 높다.



모델을 학습시킬 때, 우리의 목표는 주어진 dataset  $D = \{(x_1, t_1), (x_2, t_2), \dots, (x_n, t_n)\}$ 에 대해 이 데이터를 완벽히 표현하는 true function  $f$ 를 찾는 것이다.

이때  $x$ 는 데이터를  $t$ 는 target value 또는 label을 의미한다.

target value  $t$ 는 true function  $f$ 와 noise  $\epsilon$ 로 표현된다.

$$t = f(\mathbf{x}) + \epsilon$$

이때  $\epsilon$ 는 평균이 0이고 분산이  $\sigma^2$ 인 normal distribution을 따른다.

만약 loss function으로 MSE를 사용한다면 loss의 기대값은 아래와 같이 bias, variance, noise로 분해된다.

이때  $y$ 는 학습시키고자 하는 모델을 의미한다.

$$\begin{aligned} \mathbb{E}\{(t - y)^2\} &= \mathbb{E}\{(t - f + f - y)^2\} \\ &= \mathbb{E}\{(t - f)^2\} + \mathbb{E}\{(f - y)^2\} + 2\mathbb{E}\{(f - y)(t - f)\} \\ &= \mathbb{E}\{\epsilon^2\} + \mathbb{E}\{(f - y)^2\} + \underbrace{2[\mathbb{E}\{ft\} - \mathbb{E}\{f^2\} - \mathbb{E}\{yt\} + \mathbb{E}\{yf\}]}_{=0} \\ &= \mathbb{E}\{(f - \mathbb{E}\{y\} + \mathbb{E}\{y\} - y)^2\} + \mathbb{E}\{\epsilon^2\} \\ &= \mathbb{E}\{(f - \mathbb{E}\{y\})^2\} + \mathbb{E}\{(\mathbb{E}\{y\} - y)^2\} + \underbrace{2\mathbb{E}\{(\mathbb{E}\{y\} - y)(f - \mathbb{E}\{y\})\}}_{=0} + \mathbb{E}\{ \\ &= \underbrace{\mathbb{E}\{(f - \mathbb{E}\{y\})^2\}}_{\text{bias}^2} + \underbrace{\mathbb{E}\{(\mathbb{E}\{y\} - y)^2\}}_{\text{variance}} + \underbrace{\mathbb{E}\{\epsilon^2\}}_{\text{noise}} \end{aligned}$$

노이즈 항은 우리가 학습시키고자 하는 모델  $y$ 와는 독립적이기 때문에 학습을 통해 최소화 하는 것이 불가능하다.

결국 loss를 최소화 하기 위해서는 bias와 variance 항을 최소화 해야 한다.

그러나 bias와 variance 사이에는 tradeoff가 존재한다.

bias 항을 최소화하기 위해  $E(y) = f$ 가 되도록 모델을 학습한다면, variance 항은 noise 항과 같이 된다. ( $f - y = \epsilon$  이므로)

반대로 variance 항을 최소화하기 위해 모델  $y$ 가 입력 데이터에 상관없이 특정 상수  $a$ 만을 반환한다면

$$\mathbb{E}\{(\mathbb{E}\{y\} - y)^2\} = \mathbb{E}\{(a - a)^2\} = 0$$

이 되겠지만, bias 항은 크게 증가할 것이다.

### 성능 측정 지표

#### 평균 제곱근 오차(RMSE, Root Mean Square Error)

$$RMSE(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

제곱항을 합한 것의 제곱근 계산은 유clidean norm(Euclidian norm)에 해당한다.

#### 평균 절대 오차(Mean Absolute Deviation, MAE)

이상치가 많이 보인다면 이 지표를 사용하는 것도 좋다.

$$MAE(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

RMSE와 MAE 모두 예측값의 벡터와 타깃값의 벡터 사이의 거리를 재는 방법이다.

거리 측정에는 여러 가지 방법(또는 노름)이 가능하다.

절대값의 합을 계산하는 것은 맨해튼 노름(Manhattan norm)이라고 한다.

노름의 지수가 클수록 큰 값의 원소에 치우치며 작은 값은 무시된다.  
그래서 RMSE가 MAE보다 조금 더 이상치에 민감하다.  
하지만 이상치가 매우 드물면 RMSE가 잘 맞아 일반적으로 널리 사용된다.

## kNN(k – Nearest Neighbors)

사회적인 관계를 관찰해보면 대략적으로 비슷한 사람끼리 모이는 성질이 있다.  
이를 공간적인 관계로 가져와보면 거리가 가까울수록 모임

### 수치형 데이터에 대해서 수행됨(거리를 계산해야 하므로)

거리를 계산한다음, 순위를 매기고 다수결에 의해 평가  
이때 순위를 기준으로 뽑아오는 표본의 개수를 k라 한다.  
k값은 홀수가 좋다. 왜냐하면, 결정을 내려야 하므로(ex: 평가가 2대 2로 나뉘었다고  
하면 어느것을 선택해야 할지 모름)

#### 거리 유사도 측정

##### 유클리드 거리(Euclidean distance)

#### 다수결

순위를 기준으로 데이터를 정렬해서 k개의 표본을 뽑아옴  
k개의 표본의 label을 확인해서 label이 가장 큰 녀석이 최근접 이웃이라고 판  
단

#### kNN과 표준화, 정규화

scale이 다른 데이터 셋을 그대로 training set으로 두고 거리를 계산하면,  
scale이 큰 데이터셋에 의존하는 편향된 결과가 나올수 있다.  
따라서, 각 dataset을 표준화, 정규화하여 scaling 한다음 거리를 계산하는  
방식도 이용된다.

## 표준화(standardization)

평균을 기준으로 얼마나 떨어져있는지를 나타내는 값으로, 이 방법을 적용하려는 때는 2  
개 이상의 대상이 단위가 다를 때 대상 데이터를 같은 기준으로 볼 수 있게 한다.  
이 방법은 데이터를 다소 평평하게 하는 특성을 가지므로 이 방법을 적용하면 간극이 줄  
어드는 효과가 발생하여 간극이 큰 데이터간의 간극을 줄이는 결과를 얻게 된다.

x\_stdardization = (x\_sample - x\_평균) / x\_표준편차  
ex)

```
x_sample_set = (np.arange(9, dtype=np.float) - 3)
x_sample_set = x_sample_set.reshape(-1, 1)
x_stdardization_set = (x_sample_set -
    np.mean(x_sample_set)) / np.std(x_sample_set)
```

#### PYTHON

##### StandardScaler

```
from sklearn.preprocessing import StandardScaler
표준화를 해줌
ex)
X = (np.arange(9, dtype=np.float) - 3).reshape(-1, 1)
    # -3부터 5까지의 분포
pd.DataFrame(X).describe()
scaler = StandardScaler()
```

```

scaler.fit(X)
X_scaled = scaler.transform(X)
np.mean(X_scaled), np.std(X_scaled)
    fit()

transform()

scale()
from sklearn.preprocessing import scale
scale함수를 쓰면 표준화된 데이터가 도출됨
ex) scale(bmi.iloc[:, :2])

```

### 표준화한 평균과 분산

데이터 집합을 표준화하면 평균이 0, 분산이 1이 된다.

#### IN R

```

scale()
ex) scale(c(1,2,3,4,5))

```

### 정규화(normalization)

전체 구간을 설정하여 데이터를 관찰하는 방법

이는 데이터 군 내에서 특정 데이터가 가지는 위치를 볼때 유용하다

```
x_new = (x - x_min) / (x_max - x_min)
```

ex)

```
x_sample_set = (np.arange(9, dtype=np.float) - 3)
x_sample_set = x_sample_set.reshape(-1, 1)
```

```
x_normalization_set = (x_sample_set - x_sample_set.min()) /
    (x_sample_set.max() - x_sample_set.min())
```

#### IN PYTHON

##### MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
```

정규화를 해줌

ex)

```
min_max_scaler = MinMaxScaler()
```

```
min_max_scaler.fit_transform(x_sample_set)
```

### 정규화는 왜 0~1의 범위를 가질까?

```
x_new = (x - x_min) / (x_max - x_min)
```

위 공식의 x에 x\_min을 넣어보라, 0이 나온다.

위 공식의 x에 x\_max를 넣어보라, 1이 나온다.

즉, sample의 값이 최소, 최대가 될때 0, 1이 나오고, 그 중간에 있는 데이터가 들어갈때 0~1의 사이값이 도출된다.

따라서, 0~1의 범위를 가진다.

$0 \leq x_{\text{new}} \leq 1$

### feature scaling

서로 다른 변수 값의 범위를 일정한 수준으로 맞추는 작업

#### 방법

##### 표준화(standardization)

## 정규화(normalization)

### 자료

#### 자료의 중심

관찰된 자료들이 어디에 집중되어 있는지 나타낸다.

#### 대표값

평균, 중앙값, 최빈값

### 데이터분류

#### IN PYTHON

##### train\_test\_split

데이터를 분류해준다.

```
from sklearn.model_selection import train_test_split
훈련 데이터, 테스트 데이터, 훈련 라벨, 테스트 라벨 =
    train_test_split(data_set, labels, test_size = 테스트 데이터의 비율)
ex)
data_train, data_test, label_train, label_test =
    train_test_split(data_set, labels, test_size = 0.2)
```

#### IN R

##### createDataPartition()

createDataPartition(y = [기준 데이터 set], p = 전체 데이터 중 training 할 데이터의 비율, list = list로 뽑아낼지)

리턴값은 training 을 위한 인덱스 벡터

ex)

```
x <- c(1,1,1,1,1,1,2,2,2,2)
idx <- createDataPartition(y= c(1,1,1,1,1,1,2,2,2,2),
                           p=0.4, list=FALSE)
x[idx] <- 훈련할 데이터 set
x[-idx] <- test를 위한 데이터 set
```

random sampling 매뉴얼하게 하기

R의 sample 함수를 이용하여 데이터를 numbering하고 랜덤 샘플링할수 있음

ex)

```
iris_sample = sample(2, nrow(iris), replace = TRUE,
                     prob=c(0.7,0.3))
```

```
iris_training <- iris[iris_sample == 1, 1:4]
iris_train_label <- iris[iris_sample == 1, 5]
```

```
iris_test <- iris[iris_sample == 2, 1:4]
iris_test_label <- iris[iris_sample == 2, 5]
```

### 혼동행렬(confusion matrix)

contingency table, an error matrix 라고도 표현되는데, 이는 주로 알고리즘의 성능을 평가할 때 평가하는 지표로 많이 사용된다.

예측값이 실제 관측값에 대해 얼마나 일치하는지를 보여주는 행렬

행 - 실제 (p, n)  
열 - 예측 (p', n')

	p'	n'
p	True Positive(TP)	False Negative(FN)
n	False Positive(FP)	True Negative(TN)

값

예측과 실제가 일치하는지 여부에 따라 True or False가 결정된다.  
예측한 값에 따라 Positive/Negative 가 결정된다.

### 모델 평가

#### Precision - 정밀도

$$TP / (TP + FP) \Leftrightarrow TP / \text{sum}(p')$$

관련 있는 것으로 분류된 결과물의 비율

positive로 예측한 것 중 실제로 positive한것의 비율

1에 가까울수록 좋음

범위 0~1

#### Recall(=sensitivity) - 재현율

$$TP / (TP + FN) \Leftrightarrow TP / \text{sum}(p)$$

관련 있는 것으로 분류된 항목들 중 실제 일치하는 항목들의 비율

실제로 positive 한 것중 정확히 예측한 것의 비율

1에 가까울수록 좋음

범위 0~1

#### Accuracy - 정확도

전체 중에서 올바르게 예측한 것이 몇개인지

$$(TP + TN) / (TP + TN + FP + FN) \Leftrightarrow \text{trace}(C) / \text{sum}(p, n)$$

전체 경우에서 정확하게 예측한것의 비율

1에 가까울수록 좋음

범위 0~1

#### IN PYTHON

```
from sklearn.metrics import accuracy_score  
accuracy_score(실제값, 예측값)  
ex)  
sklearn.metrics.accuracy_score(iris_test_label,  
y_pred)
```

#### F1-score

F1 score는 Precision과 Recall의 조화평균이다.

Precision과 Recall이 서로 다른 모델들을 비교하기 위해 사용한다.

$$(F1\text{-score}) = 2 * (1 / (1/\text{precision}) + (1/\text{recall})) = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$$

F1-score 범위

0~1 (1에 가까울수록 좋음)

#### IN PYTHON

모델 평가 기준 출력

ex) `classification_report(Y_test, y_predict)`

IN PYTHON 혼동행렬 생성

ex)

```
from sklearn.metrics import classification_report,  
    confusion_matrix  
confusion_matrix(Y_test, y_predict)
```

### 분류결과표

	예측클래스0	예측클래스1	예측클래 스2
정답클래스 0	정답이0, 예측이0인 표본의 수 의 수정답이0, 예측이2인 표본의 수	정답이0, 예측이1인 표본	
정답클래스 1	정답이1, 예측이0인 표본의 수 의 수정답이1, 예측이2인 표본의 수	정답이1, 예측이1인 표본	
정답클래스 2	정답이2, 예측이0인 표본의 수 의 수정답이2, 예측이2인 표본의 수	정답이2, 예측이1인 표본	
	...		

IN R

gmodels 라이브러리의 `CrossTable()` 함수 이용

```
CrossTable( y_test, y_predict, prop.chisq = FALSE, dnn =  
    축표현)
```

ex) `CrossTable(iris_test_label, iris_predict, prop.chisq =  
 FALSE, dnn = c('예측', '실제'))`

## 정밀도(Precision)와 재현율(Recall, sensitivity)

이진 분류 기법(binary classification)을 사용하는 패턴 인식과 정보 검색 분야에서,

정밀도는 검색된 결과들 중 관련 있는 것으로 분류된 결과물의 비율이고,

재현율은 관련 있는 것으로 분류된 항목들 중 실제 검색된 항목들의 비율이다.

따라서 정밀도와 재현율 모두 관련도(Relevance)의 측정 기준 및 지식을 토대로 하고 있다.

## 평균(Mean)

데이터의 개별성은 버리고, 또한 우연성은 줄이면서, 집단의 경향성 만을 고려하는 자료 요약에 대한 대표적인 개념

데이터 집단에서 중심의 경향(Central Tendency)을 나타내는 수학적 척도

### 산술평균(arithmetic mean)

주어진 수의 합을 수의 개수로 나눈 값이다.

### 조화평균(harmonic mean)

수학에서 조화 평균은 주어진 수들의 역수의 산술 평균의 역수를 말한다.

평균적인 변화율을 구할 때에 주로 사용된다.

$$H = n / (1/a_1) + (1/a_2) + \dots + (1/a_n)$$

### 왜 조화평균을 사용하는가?

기하학적으로 봤을 때, 단순 평균이라기보다는 작은 길이 쪽으로 치우치게

된, 그러면서 작은 쪽보다도 작은 평균이 도출된다.  
이렇게 조화평균을 이용하면 산술평균을 이용하는 것보다, 큰 데이터가  
끼치는 bias가 줄어든다고 볼 수 있다.

### 가중 산술 평균(weighted mean)

각 항의 수치에 그 중요도에 비례하는 계수를 곱한 다음 산출한 평균  
정밀도나 들어온 양이 같지 않은 물품의 평균 가격처럼 원래의 수치가 동등하지 않  
다고 생각되는 경우 사용  
자료의 평균을 구할 때 자료 값의 중요도나 영향 정도에 해당하는 가중치를 반영하  
여 구한 평균값

#### 제한

가중 평균은 집단의 변량에 부의 값이 나타나지 않을 경우에 한해서 이용되  
며,  
다소 계산이 복잡

#### 장점

변량의 극단적인 값에 영향을 덜 받게됨  
가중평균은 결국 비율의 평균법으로서 산술평균보다 훨씬 합리적

### 기하평균(Geometric mean)

n개의 양수값을 모두 곱한 것의 n제곱근 이다.  
비율의 평균 계산에 많이 사용된다.  
산술평균이 item을 add하는 반면, 기하평균은 항목을 곱한다.  
또한 양수만 기하평균을 얻을수 있다.  
ex) 물가상승률, 인구변동률, 연평균증가률

#### 기하평균의 필요성

곱셈으로 계산하는 값에서의 평균을 계산하고자 할 때 산술평균이 아닌 기하  
평균을 사용한다.

ex)  
어떤 값이 처음에 1000이고, 첫 해에 10% 증가하고, 그 다음 해에 20%  
증가하고, 그 다음 해에 15% 감소했다고 할 때,  
결과 값은 처음의 값 1000에 1.1, 1.2, 0.85의 기하평균을 세번 곱한  
값이 된다.  
 $(1.1 * 1.2 * 0.85)^{(1/3)}$

### 절단평균, 절사평균(Trimmed mean)

자료중에서 큰 관측값이나 작은 관측값을 각각 몇 만큼 버린 나머지 관측값들로부  
터 구한 평균  
스포츠 경기에서 많이 사용

### 자료의 퍼진 정도

대표값을 중심으로 얼마나 자료들이 퍼져있는지를 나타냄  
범위(range): [min, max]

### 분산(variance)

합((개별관측값 - 개별평균)<sup>2</sup>)

$$\text{분산} = \frac{\text{합}((\text{개별관측값} - \text{개별평균})^2)}{n - 1}$$

### 공분산(covariance)

2개의 확률변수의 상관정도를 나타내는 값이다. (1개의 변수의 이산정도를 나타내는 분산과는 별개임)

- $\text{cov}(x, y) > 0$  : x와 y가 같은 방향으로 변화한다.
- $\text{cov}(x, y) < 0$  : x와 y가 다른 방향으로 변화한다.
- $\text{cov}(x, y) = 0$  : x와 y의 값이 서로 상관없이 움직이는 경우  
합  $[ (x - \bar{x}) * (y - \bar{y}) ]$

$$\text{공분산}(x, y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

### 표준편차(standard deviation)

#### 결정 경계(Decision Boundary)

두 개의 계층을 가지고 있는 통계적인 분류 문제에서, 결정 경계는 기본 벡터공간을 각 클래스에 대하여 하나씩 두 개의 집합으로 나누는 초표면이다.

분류기는 결정 경계의 한쪽에 있는 모든 점을 한 클래스, 다른 한쪽에 있는 모든 점을 다른 클래스에 속하는 것으로 분류한다.

2차원에서는 결과 값 y들을 분류할 수 있는 기준선을 의미한다.

#### 통계적 결정 이론(Statistical Decision Theory)

불확실한 상황에서 최적의 결정을 하기 위한 이론

#### 확률(Probability)

경험 혹은 실험결과로 특정한 사건이나 결과가 발생할 가능성

관심있는 사건이 발생할 가능성을 0~1사이 숫자로 표현한 값

어떤 사건이 일어날 것인지(가능성, possibility) 혹은 일어났는지(상대빈도)에 대한 믿음 혹은 지식을 표현하는 방법

possibility의 measure 혹은 relative frequency의 표현

#### 한계확률, 주변확률(marginal probability)

둘 이상의 사건이 동시에 일어날 수 있을 때에 하나의 특정 사건에 주목하여 그것이 일어날 확률

#### 결합확률(Joint Probability)

둘 이상의 사건이 모두, 즉 동시에 일어날 확률(동시성)

즉 조건이 동시에 적용되는 사건에 대한 확률이다. (and 조건)

#### 조건부확률(Conditional Probability)

이미 하나의 사건이 발생한 상태에서 또 다른 사건이 발생할 가능성을 나타내는 확률

사건 B가 일어나는 경우에 사건 A가 일어날 확률

즉, B가 발생하여, 표본공간이 B 사건으로 변화되었을 때 A사건이 일어날 경우이다.

다시 말해서, 특정 조건을 적용시킨 후, 다시 특정 조건을 적용한 사건에 대한 확률이다. <- 합성함수 느낌

#### 사건(Event)

특정 조건을 만족시키는 집합(조건 + 집합)

사건은 공간에 제한을 두는 것이다.

표본공간의 부분집합이다.

### 동시 사건, 결합 사건(Joint Events)

동시에 함께 고려하는 여러 확률적 사건들

### 표본공간(Sample Space)

통계적 실험에서 일어날 수 있는 모든 가능한 실험결과들의 집합

나이브 베이즈(Naive Bayes) – 우도 확률을 이용해서 클래스를 예측해보자.

데이터를 나이브(단순)하게 독립적인 사건으로 가정하고 이 독립사건들을 베이즈이론에 대입시켜 가장 높은 확률의 레이블로 분류하는 알고리즘

특징값이 제공하는 증거를 기반으로 결과가 관측될 확률을 계산한다.

나중에 분류기가 레이블이 없는 데이터에 적용될 때 결과가 관측될 확률을 이용해서 새로운 특징에 가장 유력한 클래스를 예측한다.

베이지안 분류기는 결과에 대한 전체 확률을 추정하기 위해 동시에 여러 속성 정보를 고려해야만 하는 문제에 적합하다.

많은 머신 러닝 알고리즘이 영향력이 약한 특징은 무시하지만, 베이지안 기법은 모든 증거를 활용해 예측을 절묘하게 바꾼다.

특징이 아주 많아서 상대적으로 영향이 작다면 모두 합쳤을 때 결합된 영향은 꽤 클 수 있다.

### 사용 분야

- 스팸 이메일 필터링과 같은 텍스트 분류
- 컴퓨터 네트워크에서 침입이나 비정상행위 탐지
- 일련의 관찰된 증상에 대한 의학적 질병 진단

### 가정

데이터셋의 모든 특징이 동등하게 중요하고 독립적이라고 가정한다.

### IN R

e017 패키지 사용

library(e1071)

naiveBayes(x, y, laplace = 0, ...)

ex)

mail <- read.csv('c:/data/mail.csv', header = T)

mail\_cols = colnames(mail)

mail\_label = mail[5]

nm <- naiveBayes(mail[1:4], mail\$분류, laplace = 1)

predict(m, test)

model을 적용하여 예측

m: model

test: 테스트 데이터

ex)

test <- data.frame('YES', 'NO', 'NO', 'YES')

colnames(test) <- mail\_cols[1:4]

predict(nm, test)

## IN PYTHON

```
nltk 라이브러리 사용
from nltk.tokenize import word_tokenize
ex)
train = [
    ('홍길동은 좋아', '긍정'),
    ('강아지는 무지 좋아', '긍정'),
    ('수업이 재미없어', '부정'),
    ('홍길동은 이쁜 강아지야', '긍정'),
    ('민형이는 너무 멋있어', '긍정'),
    ('은진이는 무지 무지 많이 이뻐', '긍정'),
    ('난 수업을 마치고 희주랑 클럽갈꺼야', '긍정'),
    ('오늘 하루는 너무 짜증스러운 날이야', '부정'),
    ('날이 맑아서 좋아', '긍정'),
    ('비가 오니 짜증난다.', '부정'),
    ('지하철에서 질서가 너무 없어 짜증이 난다', '부정'),
    ('공기가 맑아서 좋다', '긍정'),
    ('밝게 인사해주니 행복하다', '긍정'),
    ('친구가 짜증을 낸다', '부정')
]
allword = {word for sentence in train for word in
    word_tokenize(sentence[0])}
t = [ ({word : (word in word_tokenize(x[0])) for word in
    allword}, x[1]) for x in train ]
model
    NaiveBayesClassifier
        nltk.NaiveBayesClassifier.train()
        nltk.NaiveBayesClassifier.train( 훈련 데이터 셋 )
        훈련 데이터셋 구성 =   <- 각 리스트당 row 하나
        [
            (
                {
                    description_variable1 : value1,
                    description_variable2 : value2,
                    ...
                    description_variablek : valuek
                },
                label1
            ),
            (
                {
                    description_variable1 : value1,
                    description_variable2 : value2,
                    ...
                    description_variablek : valuek
                },
                label2
            )
        ]
```

```

        )
    ]
ex) model = nltk.NaiveBayesClassifier.train(t)

show_most_informative_features()
가장 성격이 뚜렷한 feature를 추출한다.
ex) model.show_most_informative_features()

test_data
ex) test_f = {word : (word in word_tokenize(test)) for
word in allword }

분류
classify()
classify( test_data )
ex) model.classify(test_f)

```

### 베이지안 확률(Bayesian Probability)

확률을 지식 또는 믿음의 정도를 나타내는 양으로 해석하는 확률론이다.

확률을 발생 빈도나 어떤 시스템의 물리적 속성으로 여기는 것과는 다른 해석이다.

통계학자 토마스 베이즈의 이름을 따서 명명되었다.

사전에 발생한 일들을 가지고 사후확률을 예측하는것!

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{class Prior}}{\text{evidence}}$$

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

### 수식

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

$$P(B \cap A) = P(B|A)P(A) \text{ 이다. 그런데 } P(A \cap B) = P(B \cap A) \text{ 이기 때문에}$$

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

두 확률 변수의 사전 확률과 사후 확률 사이의 관계를 나타내는 정리이다.

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

$$\Leftrightarrow \text{Posterior Probability} = (\text{Likelihood} * \text{Class Prior Probability}) / \text{Predictor Prior Probability}$$

사전확률(Prior Probability)

결과가 나타나기 전에 결정되어 있는 A(원인)의 확률

현재 가지고 있는 정보를 기초로하여 정한 초기 확률

확률 시행 전에 이미 가지고 있는 지식을 통해 부여한 확률

$$P(A)$$

### 우도확률(likelihood Probability)

A(원인)가 발생하였다는 조건하에서 B(결과)가 발생할 확률  
나타난 결과에 따라 여러 가능한 가설들을 평가할 수 있는 측도임  
우도는 확률로 표현되나 각 가설에 대한 가능성/지지도 등의 의미가 강함  
즉, 각 가설에 대한 우도는, 그 가설을 지지하는 정도라고 볼수 있음  
각각의 원인으로부터 결과가 나타날 것이라는 가설에 대해 지지하는 정도  
나타난 결과  $B_j$  마다 다른 값을 갖는 여러 가설  $A_i$ 들을 평가할 수 있는  
조건부확률  
각각의 원인  $A_i$ 은 분류 범주/ 분류 영역/ 카테고리 라고도 함  
 $P(B|A)$

### 사후확률(posterior Probability)

B(결과)가 발생하였다는 조건하에서 A(원인)가 발생하였을 확률  
사건 발생 후에 어떤 원인으로부터 일어난 것이라고 생각되어지는 확률  
추가된 정보로부터 사전 정보를 새롭게 수정한 확률(수정확률)  
 $P(A|B)$

### 주변우도(marginal likelihood)

Marginal Likelihood는 두 가지 관점에서 이야기할 수 있는데, 첫번째  
는 말 그대로 marginal을 하여 가능도를 구한다는 개념  
두번째로 베이지안 관점에서는 Evidence에 해당하는 즉, 데이터에 대한 확  
률 부분을 일컫는 말이다.

### 클래스 조건부 독립(class conditional independence)

주어진 클래스의 한 속성 값이 다른 속성의 값과 상호독립

### 조건부독립(conditional independence)

일반적인 독립과 달리 조건이 되는 별개의 확률변수 C가 존재해야 한다.  
조건이 되는 확률변수 C에 대한 A, B의 결합조건부확률이 C에 대한 A, B의 조건부확률  
의 곱과 같으면 A와 B가 C에 대해 조건부독립이라고 한다.  
 $P(A \cap B | C) = P(A | C)P(B | C)$

A, B가 C에 대해 조건부독립이면 다음도 만족한다.

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

#### 주의

주의할 점은 조건부독립과 무조건부독립은 관계가 없다는 점이다.  
즉, 두 확률변수가 독립이라고 항상 조건부독립이 되는 것도 아니고 조건부독립이  
라고 꼭 독립이 되는것도 아니다.

$$P(A \cap B) = P(A)P(B) \text{는 } P(A \cap B | C) = P(A | C)P(B | C) \text{ 를 의미하지 않는다.}$$
$$P(A, B | C) = P(A | C)P(B | C) \text{ 는 } P(A, B) = P(A)P(B) \text{를 의미하지 않는다.}$$

### 무조건부 독립(unconditional independence)

조건부독립과 비교하여 일반적인 독립은 무조건부독립이라고 한다.

### 설명 변수(Explanatory variable)

설명 변수는 독립변수의 유형이다.

이 두 용어는 종종 교환가능하게 사용된다.

그러나 이 두 용어 사이에는 미묘한 차이가 있다.

변수가 독립이라는 것은, 다른 변수에 영향을 미치지 않음을 의미한다.

변수가 독립이 아니라는 것은 설명 변수이다.

### 희소행렬(sparse matrix)

행렬의 값이 대부분 0인 경우를 가리키는 표현이다.

### 결정 트리 학습법(decision tree learning)

어떤 항목에 대한 관측값과 목표값을 연결시켜주는 예측 모델

몇몇 입력 변수를 바탕으로 목표 변수의 값을 예측하는 모델을 생성하는 것을 목표로 한다.

각 내부 노드들은 하나의 입력 변수에, 자녀 노드들로 이어지는 가지들은 입력 변수의 가능한 값에 대응된다.

잎 노드는 각 입력 변수들이 루트 노드로부터 잎 노드로 이어지는 경로에 해당되는 값들을 가질 때의 목표 변수 값에 해당된다.

결정 트리 학습법은 지도 분류 학습에서 가장 유용하게 사용되고 있는 기법 중 하나이다.

결정 트리 또는 분류 트리의 모든 내부 노드에는 입력 속성이 일대일로 대응된다.

트리의 내부 노드에 연결된 가지에는 속성이 가질 수 있는 값들이 표시되며, 잎노드에는 클래스 또는 클래스의 확률 분포가 표시된다.

결정 트리의 '학습'은 학습에 사용되는 자료 집합을 적절한 분할 기준 또는 분할 테스트에 따라 부분 집합들로 나누는 과정이다.

이러한 과정은 순환 분할이라 불리는 방식으로 각각의 나눠진 자료 부분 집합에 재귀적으로 반복되며, 분할로 인해 더이상 새로운 예측 값이 추가되지 않거나 부분 집합의 노드가 목표 변수와 같은 값을 지닐 때까지 계속된다.

데이터 마이닝에서 결정 트리는 주어진 데이터의 일반화와 범주화를 돋기 위해 수학적 표현으로 기술된다.

$(x, Y) = (x_1, x_2, \dots, x_k, Y)$  – 종속 변수  $Y$ 는 분류를 통해 학습하고자 하는 목표 변수이며, 벡터  $x$ 는  $x_1, x_2, x_3$ 등의 입력 변수로 구성된다.

### 결정 트리(decision tree)

여러가지 규칙을 순차적으로 적용하면서 독립 변수 공간을 분할하는 분류 모형이다.

#### 분류법

- 여러가지 독립 변수 중 하나의 독립 변수를 선택하고 그 독립 변수에 대한 기준값(threshold)을 정한다. 이를 분류 규칙이라고 한다.
- 전체 학습 데이터 집합(부모 노드)을 해당 독립 변수의 값이 기준값보다 작은 데이터 그룹(자식 노드1)과 해당 독립 변수의 값이 기준값보다 큰 데이터 그룹(자식 노드2)으로 나눈다.
- 각각의 자식 노드에 대해 1~2 단계를 반복하여 하위의 자식 노드를 만든다. 단, 자식노드에 한가지 클래스의 데이터만 존재한다면 더이상 자식 노드를 나누지 않고 중지한다.

#### 목표 변수

scklearn에서는 숫자여야함

R에서는 범주형이어야함

#### 결정 노드(decision node)

속성에 따라 선택이 이루어지는 노드

### 노드에서의 분할

p값이 낮은 변수를 최적의 변수로 선택  
엔트로피 정보이득은 큰값을 가진 변수를 최적의 변수로 선택(즉, 분할 되었을때 엔트로피가 작은 question을 던짐)  
지니값을 계산하고 최저값을 가진 변수를 최적의 변수로 선택(즉, 분할 되었을때 순도가 높은 question을 던짐)

decision node에서 사용하는 measure

- p값
- entropy
- gini coefficient

#### 정보이득

분할을 위해 엔트로피를 사용한다면, 각 특징별로 분할로 인해 생기는 동질성의 변화를 측정해야 한다.

이것이 정보 이득이라는 척도이다.

$\text{InfoGain}(F) = \text{Entropy}(S) - \text{Entropy}(S\text{의 하위에서의 클래스 } 1 = \text{파티션}1) \dots - \text{Entropy}(S\text{의 하위에서의 클래스 } n = \text{파티션}n)$

#### 분할되었을 때 엔트로피가 작은 question?

정보이득은 분할한 후 분류가 잘되었는지를 파악하는 측정값이다.

분류가 잘되었다는 것은 상위노드 보다 하위 노드의 엔트로피가 작아졌다는 것을 의미한다.

따라서 더 잘 분류하려면 하위노드의 엔트로피가 작은 값, 즉 정보 이득이 큰 분류기준을 선택해야한다.

### 목표

잘 분류하자!

잘 분류하려면? 잘 분할해야한다.

잘 분할하려면? 좋은 질문을 던져야 한다.

좋은 질문이란? 분할 이후 데이터셋이 순수해지는 질문이다.

### 잎 노드(leaf node)

일련의 결정이 이뤄진 결과

### 분기(branches)

결정의 잠재적 결과를 나타내는 분기로 데이터를 분할

### 분류 알고리즘

CART: 지니지수

C5.0: 엔트로피 지수

CHAID: 카이제곱 통계량

### 순도(purity)

부분집합이 단일 클래스를 포함하는 정도

### 순수(pure)

단일 클래스로 이루어진 부분집합

### 루트( root )

전체 데이터 셋을 표현함(분할되지 않았기에)

### 분할정복(divide and conquer)

의사결정 트리는 재귀분할(recursive partitioning)이라는 휴리스틱을 사용해서 만든다.

각 분기별로 처리하면서 알고리즘은 종료 기준에 도달할 때까지 데이터를 분할하고 정복해 나가며, 결정 노드를 생성할 때마다 최고의 후보 특징을 선택한다.

### 멈추는 조건

노드에 있는 모든 예시가 같은 클래스를 가진다.

예시를 구별하는 특징이 남아있지 않다.

미리 정의된 크기 한도까지 트리가 자랐다.

### 과적합

오분류된 값이 각자 자신의 작은 파티션으로 들어가 정확하게 분류될 때까지 점차 세부적으로 분할 정복을 이어갈 수도 있다.

그러나 이런 방식으로 의사결정 트리를 과적합 시키는 것은 바람직하지 않다.

데이터가 무한정 나뉘는 것을 막을 수는 없지만, 너무 세부적으로 결정이 만들어지면 더 폭넓게 일반화하지 못한다.

+ 의사결정 트리는 모델을 훈련 데이터에 과적합시키는 경향이 있는 것으로 알려져 있다.

### 트리에서 발견되는 비논리

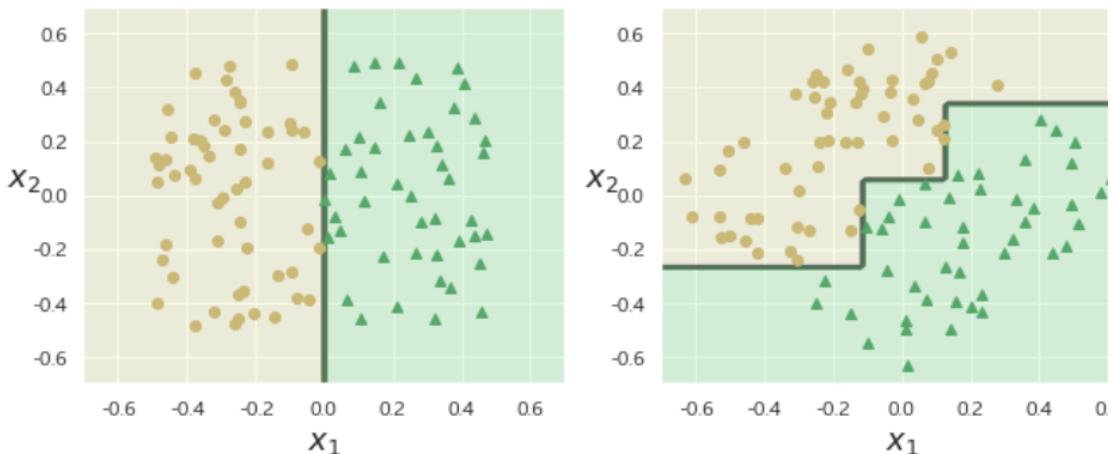
가끔 트리는 논리적으로 이해가 안되는 결정을 만든다.

그런 규칙들은 데이터에 실제 패턴을 반영하기도 하고 통계적 이상치일 수도 있다.

두 경우 모두, 트리의 논리가 비즈니스 용도로 합당한지 확인하기 위해 이런 이상한 결정을 조사해보는 것이 중요하다.

### 불안정성

의사결정나무 모델은 계단 모양의 결정경계를 만든다. 그래서 학습 데이터 셋의 회전에 민감하다.



## CART(Classification And Regression Tree) 알고리즘

의사결정나무분석을 형성하는데 있어서 가장 보편적인 알고리즘이다.

1984년 L.Breiman에 의해 발표했다.

### 분리기준찾기

CART 알고리즘은 이진 트리 구조로 모형을 형성하는데,

첫번째 과제는 목표변수를 가장 잘 분리하는 설명변수와 그 분리시점을 찾는 것이다.

이 측도의 하나를 다양성(diversity)이라고 하는데, 노드의 다양성을 가장 많이 줄이는 설명변수를 선택한다.

그리고 그 분리 기준은 다음 값을 가장 크게 하는 곳을 선택한다.

diversity(before split) - ( diversity(left child) + diversity(right child) )

### Growing the Full tree

처음 분리기준으로 두 개의 마디를 형성하면, 목표 변수를 한쪽의 값으로 분류시킬수 있는 기준을 찾아서 계속 나무를 형성해나간다.

그래서 더이상 분리가 이루어지지 않고 다양성이 효과적으로 줄었을 때 끝 노드를 형성한다.

이렇게 완전히 Full Tree를 형성하는 것은 주어진 데이터는 잘 맞추겠지만, 새로운 데이터가 들어오면 잘 분류하지 못할 수 있다.

### 에러율

원래는 yes인데 no라고 분류할수도 있다. 이런것을 에러율이라고 하는데

확률적으로 정의하면, 11개중 9개를 맞추었을 경우 맞춘 확률은 0.818이 되고, 그것의 에러율은 1-0.818로 0.182가 된다.

### 가지치기(pruning)

Full 모형 형성은 주어진 데이터를 잘 분류하기 위해서 분리기준과 분리를 형성했기 때문에 주어진 데이터 분류에는 잘 맞는다.

하지만, 이렇게 형성된 모형은 새로운 데이터에서 잘 작동되지 않을 수 있다.

따라서 우리는 일반적으로, 즉 새로운 데이터 셋이 들어와도 그 예측을 일반적으로 잘 할수 있도록 적절한 가지치기를 해주어야 한다.

문제는 어느 정도까지 가지를 쳐줘야 하냐는 것이다.

불순도를 최소화하는 것이 통계적으로 효과가 없다면 리프 노드는 필요없는 노드라

할 수 있다.

이렇게 판단하는 기준은 카이제곱 검정을 사용하면 p-value 값이 0.05보다 높으면 해당 노드는 불필요한 노드로 추정하여 해당 노드의 자식 노드를 가지치기 한다.

### Identifying Candidate Subtrees

어느정도까지 가지치기를 해야 할까를 결정하기 위해 먼저 반복적인 가지치기 과정을 통해서 candidates subtress를 결정해야 한다.

우리의 목표는 각 잎 노드에서 예측에 가장 덜 영향을 끼치는 가지를 제거하는 것이다.

이러한 가지를 정의하기 위해 adjusted error rate라는 개념을 소개한다.

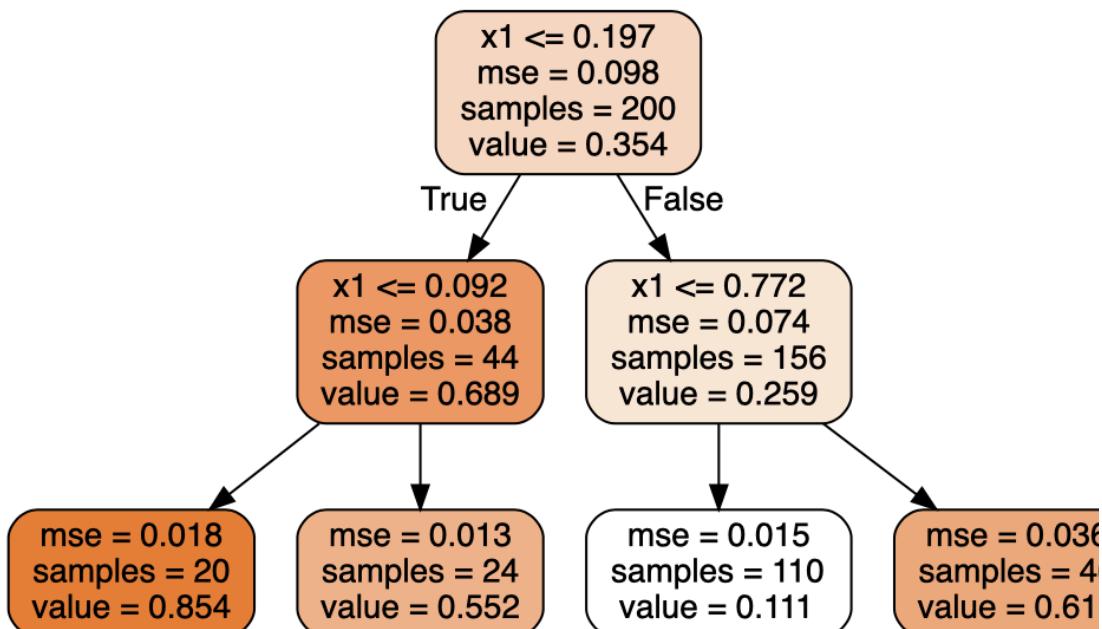
adjusted error rate

$$AE(T) = E(T) + \alpha * \text{leaf\_count}(T)$$

### 회귀

의사결정 나무는 회귀에서도 사용할수 있다.

CART 알고리즘은 학습 데이터셋을 평균제곱오차(MSE)를 최소화하도록 트리를 분할한다.



$$\left\{ \begin{array}{l} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y_i)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y_i \end{array} \right.$$

### 의사 결정 분석

결정 트리는 시각적이고 명시적인 방법으로 의사 결정 과정과 결정된 의사를 보여 주는데 사용된다

## 알고리즘

```
ex)  
hiking <- read.csv('c:/data/hiking.csv')
```

설명변수가 날씨일때 목표변수와의 교차표를 만들어서 관련성을 파악  
`CrossTable(hiking$날씨, hiking$등산, chisq = TRUE)`

다른 설명변수와의 관련성도 파악하기 위해

만약 테스트 데이터의 날씨 설명변수가 맑음일 경우, 어떤 설명변수로 분류할지 파악하기위해 각 설명변수로 분류표를 만들고 p-value값을 비교함  
p-value값이 낮은 설명변수가 관련성이 큰 설명변수이므로, 가장 낮은 설명변수를 택하여 다음 관련성을 파악

```
CrossTable(hiking[hiking$날씨 == '맑음', '기온'],  
          hiking[hiking$날씨=='맑음', '등산'], chisq = TRUE)  
CrossTable(hiking[hiking$날씨 == '맑음', '습도'],  
          hiking[hiking$날씨=='맑음', '등산'], chisq = TRUE)  
CrossTable(hiking[hiking$날씨 == '맑음', '바람'],  
          hiking[hiking$날씨=='맑음', '등산'], chisq = TRUE)
```

## 정보이득

정보이득이 가장 큰 설명변수가 가장 변별력이 좋은 설명 변수이다.  
따라서 정보이득을 계산하고, 좋은 설명변수를 선택한다.

## 모델로서 맞지 않는 시나리오

데이터에 다수의 명목 특징이 여러 레벨로 이뤄져 있거나 다수의 수치 특성이 있는 경우 결정이 아주 많이 생성되어 트리가 너무 복잡해진다.

## IN R

`rpart library` 사용

모델 생성 및 훈련

```
rpart()  
rpart( [어떻게 목표변수를 설명할것인지], data = [데이터],  
       control = [컨트롤 파라미터] )  
target <- 등산 ~ 날씨 + 기온 + 습도 + 바람 # 목표변수 d~ 설명변수1 + 설명변수2 ...
```

ex)

```
tree <- rpart(target, data=hiking,  
               control=rpart.control(minsplit=1))  
tree <- rpart(cupon_react ~ ., data=skin,  
               control=rpart.control(minsplit=4))  
control
```

`rpart 알고리즘의 세부 사항을 제어하는 옵션 목록`

`rpart.control()`

`minsplit`

분할을 시도하기 위해 노드에 존재해야하는 최소 관측치 수

## decision node의 결정

기본적으로 `rpart`는 분류를 수행할 때 gini impurity를 사용하여

분할을 선택합니다.  
params 매개 변수에 정보를 지정하여 대신 information gain을 사용할수 있습니다.  
<https://www.gormanalysis.com/blog/decision-trees-in-r-using-rpart/>

### 시각화

```
rattle library
fancyRpartPlot()
fancyRpartPlot(model, main= '')
RPart decision tree를 pretty rpart plotter를 이용해 표현

rpart.plot library <- 시각화
node 해석
맨위에 있는것 - class
중간에 있는것 - 부모노드에서 분기되었을때 branch에 해당되는 값으로 해당 노드로 오는 데이터 셋중 대표 class로 대답 할 확률
마지막에 있는것 - 전체중에 현재 노드에 해당되는 데이터의 비율 대표 클래스?
class 데이터셋의 factor 형태의 첫번째 값 해당 클래스의 빈도가 많은것
```

### 정보 획득량 계산

```
FSelector library
information.gain( labels , data_set)
ex)
install.packages('FSelector')
library(FSelector)

information.gain(cupon_react ~ ., skin)
```

### C50 library

결정트리를 만듦

<https://rpubs.com/cyobero/C50>

```
ex)
skin_model = C5.0(skin[-5], skin$cupon_react,
trials=1)
C5.0()
m <- C5.0(train, class, trials = 1, costs = NULL)
train: 훈련 데이터를 포함하는 데이터 프레임 또는 행렬
class: 훈련 데이터의 각 행에 대한 클래스를 갖는 팩터 벡터
trials: 부스팅 반복 횟수를 제어하는 숫자 옵션(디폴트 값은 1)
costs: 다양한 종류의 오류 관련된 비용을 지정하는 행렬 옵션
이 함수는 예측에 사용될 수 있는 C5.0 객체를 반환한다.
```

분류  
C5.0 은 purity 측정에 엔트로피 개념을 사용한다.

### 주의

독립변수에 feature가 1개 밖에 없는 데이터는 트리가 만들어  
지지 않음 - 분류가 안되니까..  
그러니까 이런 독립변수는 사용하면 안됨

### summary()

모델에 대한 요약정보를 보여줌

ex)

```
summary(skin_model)
```

### plot()

훈련된 model을 그림으로 보여준다.

ex)

```
plot(skin_model)
```

### predict()

p <- predict(m, test, type = 'class')

m: C5.0() 함수에 의해 훈련된 모델

test: 분류기를 구축하는 데 사용된 훈련 데이터와 같은 특징을 갖는  
테스트 데이터를 포함하는 데이터 프레임 또는 행렬

type: 'class'나 'prob', 예측 결과가 가장 확률이 높은 클래스  
인지 원시 예측 확률인지를 지정

이 함수는 type 파라미터의 값에 따라 예측 클래스 값이나 원시 예측  
확률의 벡터를 반환한다.

## 교차 분석(Crosstabulation)

교차 분석은 범주형으로 구성된 자료들간의 연관관계를 확인하기 위해 교차표를 만들어  
관계를 확인하는 방법이다.

변수들의 빈도를 확인하고 그 빈도를 이용하여 상호 연관성을 판단한다.

한 변수에 속한 빈도수와 다른 변수에 속한 빈도수를 함께 교차로 분석한다.

이때 통계량으로 카이제곱( $\chi^2$ ) 통계량을 이용하기 때문에 카이제곱검정이라고 한다.

카이제곱 검정을 하기 위해서 교차표, 관측빈도(행의 합과 열의 합을 가지고 주변확률을  
구하여 기대빈도를 만든다)

### 교차표(cross tabulation), 분할표(contingency table)

둘 이상의 독립 변수를 상호 관련시켜 한 눈에 보이게 한 표

- 어떤 분류기준(성별, 나이등)에 따라 행과 열을 분류하여 정리하여 놓은 표

ex)

구매의사

있음 없음 행의합

지역1 154 52 206

지역2 7 112 119

열의합 161 164 325

### 관측빈도(observed frequency)

교차표를 작성할 때에는 직접 수집한 데이터를 기준으로 빈도를 입력할 수 있  
는데, 이처럼 실제로 수집된 데이터의 빈도를 관측빈도라 한다.

(실제로 조사에 의해 관찰된 빈도)

### 기대빈도(expected frequency)

전체빈도  $n$ 에 대하여 행과 열의 합을 기준으로 보았을 때 각 교차되는 셀에 몇번의 빈도가 확인될수 있을지를 예상하는 기대값이다.

두 변수 사이에 연관성이 없다는 가정하에 예상되는 빈도를 기대빈도라 한다.

-> 기대빈도 = 행의합 \* 열의합 / 관측수

### 카이제곱

$$\text{카이제곱}(X^2) = \sum (\text{관측빈도} - \text{기대빈도})^2 / \text{기대빈도}$$

### 자유도

$$\text{교차표의 } (\text{행의수} - 1) * (\text{열의수} - 1)$$

### 표본 내 성능(in-sample testing)

학습 데이터 집합의 종속 변수값을 얼마나 잘 예측하였는지를 나타내는 성능이다.

### 표본 외 성능(out-of sample testing)

학습에 쓰이지 않은 표본 데이터 집합의 종속 변수 값을 얼마나 잘 예측하는가를 검사하는 것을 말한다.

### 교차검증(cross validation)

#### K-Fold Cross Validation

데이터의 수가 적은 경우에는 이 데이터 중의 일부인 검증 데이터의 수도 적기 때문에 검증 성능의 신뢰도가 떨어진다.

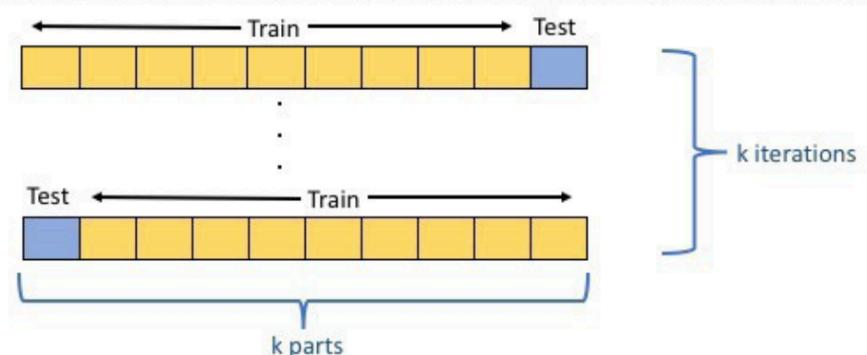
그렇다고 검증 데이터의 수를 증가시키면 학습용 데이터의 수가 적어지므로 정상적인 학습이 되지 않는다.

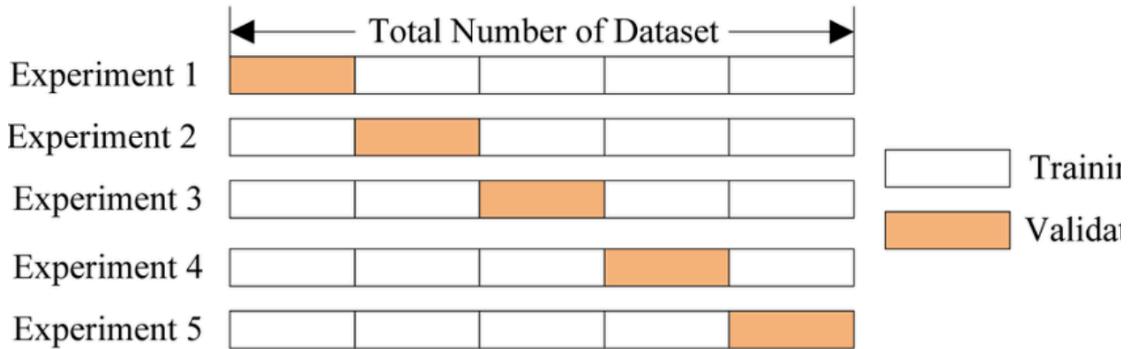
이러한 딜레마를 해결하기 위한 검증 방법이 K-폴드(K-fold) 교차검증 방법이다.

K 폴드 교차검증에서는 다음 처럼 학습과 검증을 반복한다.

## K Folds Cross Validation Method

1. Divide the sample data into  $k$  parts.
2. Use  $k-1$  of the parts for training, and 1 for testing.
3. Repeat the procedure  $k$  times, rotating the test set.
4. Determine an expected performance metric (mean square error, misclassification error rate, confidence interval, or other appropriate metric) based on the results across the iterations





이렇게 하면 총 K개의 모형과 K개의 교차검증 성능이 나온다.

이 K개의 교차검증 성능을 평균하여 최종 교차검증 성능을 계산한다.

### 사용이유

총 데이터 갯수가 적은 데이터 셋에 대하여 정확도를 향상시킬수 있음

이는 기존에 Training / Validation / Test 세 개의 집단으로 분류하는 것보다, Training과 Test로만 분류할 때 학습 데이터셋이 많기 때문

데이터 수가 적은데 검증과 테스트에 데이터를 더 뺏기면 underfitting 등 성능이 미 달되는 모델이 학습됨

### 단점

Training set / Test set을 통해 진행하는 학습법에 비해 시간 소요가 크다.

## 자유도(degrees of freedom)

통계학에서 자유도는 통계적 추정을 할 때 표본자료 중 모집단에 대한 정보를 주는 독립적인 자료의 수를 말한다.

## 유의수준(significance level)

허용가능한 1종 오류(=귀무가설이 True인데도 기각하는 오류)의 최대수준이라는 의미  
통계적인 가설검정과 추정에서 사용되는 기준값이다.

일반적으로 유의수준은  $\alpha$ 로 표시하고 95%의 신뢰도를 기준으로 한다면 ( $1 - 0.95 = 0.05$ )인 0.05값이 유의수준 값이 된다.

가설검정의 절차에서 유의수준 값과 유의확률 값을 비교하여 통계적 유의성을 검정하게 된다.

**유의확률(significance probability), p-값(p-value, probability value)**

귀무가설이 맞다고 가정할 때 얻은 결과보다 극단적인 결과가 실제로 관측될 확률  
p-값은 귀무가설이 맞다는 전제 하에, 표본에서 실제로 관측된 통계치와 같거나  
더 극단적인 통계치가 관측될 확률이다.

여기서 말하는 확률은 빈도주의(frequentist) 확률이다.

p-value는 관찰된 데이터가 귀무가설과 양립하는 정도를 0에서 1 사이의 수치로  
표현한 것이다.

p-value가 작을수록 그 정도가 약하다고 보며, 특정값보다 작을 경우 귀무가설  
을 기각하는 것이 관례이나 이에는 여러 문제가 있다.

### 극단적

유의확률은 귀무가설을 가정하였을 때 표본 이상으로 극단적인 결과를 얻을  
확률이다. -> 극단적인 결과에 대한 확률이기 때문에 확률분포의 꼬리  
에서 확률을 계산해야함

여기서 더 극단적이라는 것은 정의에 따라 다르다.

정규분포의 경우, 귀무가설을 가정한 실수 확률변수  $X:P \rightarrow R$ 와 표본  $x$   
에 대하여 다음 확률들을 정의할 수 있다.

만약 확률 변수가 단순한 실수가 아니라면, 더 복잡한 '극단성'을 정의하여야  
야 한다.

예를 들어, 표본이 노름공간에 있는 경우 노름함수를 이용하여 표본  $x$ 의 유  
의확률을 정의할수 있다.

$p(x) = P_r(||x|| \geq ||X||)$   
왼쪽 꼬리 확률(left-tail p-value)  
 $p_L(x) = P_r(x \leq X)$   
오른쪽 꼬리 확률(right-tail p-value)  
 $p_R(x) = P_r(x \geq X)$   
양쪽 꼬리 확률(double-tail p-value)  
 $p_D(x) = P_r(x \geq X \vee |x| \geq |X|)$

### p-value와 유의수준의 비교

p-value가 유의수준보다 작으면 귀무가설 기각

p-value가 유의수준보다 크면 대립가설 기각

p-value가 유의수준보다 크다는 의미는 sample을 통해 구한 통계량이 모  
평균으로부터 얼마 떨어져있지 않은 분포안에 있다는 의미이므로 귀무가  
설을 기각할수 없다.

p-value가 유의수준보다 작다는 의미는 sample을 통해 구한 통계량이 모  
평균으로부터 많이 떨어져있다는 의미이므로 귀무가설을 기각한다.

## 가설 검정(statistical hypothesis test)

상식적인 판단의 방법을 체계적으로 만들어 놓은것

### 통계적 가설(statistical hypothesis)

하나의 특정 주장을 모수를 이용해 나타낸 형태를 지칭한다.

ex) 미국 성인여자의 평균 신장은 180cm이다. - 평균신장은 여기서 모집단 특  
성을 나타내는 모수의 역할을 수행한다.

### 귀무가설, 영가설(Null Hypothesis, $H_0$ )

일반적으로 믿어온 사실을 가설로 설정한다.

귀무가설은 당연한 사실이나 연구할 의미가 없는 가설을 설정한다.

### 대립가설, 연구가설(Anti Hypothesis, $H_1$ )

공공연하게 사실로 받아들여진 현상에 대립되는 가설로 연구를 통한 대립가설  
의 조사는 의미가 있다.



며, female인 경우에 200의 sample이 있고 160명은 선호하고 40명은 선호하지 않습니다.

male과 female 모두에서 선호하는 경우가 많습니다.

남성가운데  $200/(200+50) = 80\%$ 가 선호하고 여성가운데  $160/(160+40) = 80\%$ 가 선호하므로 선호하는 비율이 같습니다.

따라서 검정통계량을 구한다음 Chisquare 분포의 임계치와 비교할 필요없이, 유의수준이 어떻게 되든 귀무가설이 기각되지 않습니다.

다시말해, male 그룹과 female 그룹에서 A라는 이슈에 대한 선호도(preference)에 차이가 있다라고 주장할수 없습니다.

즉 male과 female의 차이는 없는 것입니다.

이번에는 다음과 같이 sample이 추출되었다고 합시다.

남성	여성
----	----

--	--	--

선호	:	110명	90명
----	---	------	-----

선호하지 않음	:	90명	60명
---------	---	-----	-----

male 그룹에서 선호하는 비율은  $110/(110+90) = 55\%$ 이며 female 그룹에서는  $90/(90+60) = 60\%$ 인데 여기서는 5%의 차이가 있으며 매우 큰차이는 아니므로 가설검정을 해볼 필요가 있습니다.

### 검정통계량(test statistic)

카이제곱 통계량은

귀무가설이 맞다는 가정하에 변량과 기대값 사이 거리를 제곱하고 기대값으로 나눠 준 값을 합한다.

$$\text{SUM} \{ [(\text{실제빈도} - \text{기대빈도})^2] / \text{기대빈도} \}$$

즉, cell별로 식을 계산한 다음 모든 cell의 값을 합계하면 되는데, 기대빈도와 실제빈도의 차이가 클수록 귀무가설이 기각되게 됩니다.

### 기대빈도(expected frequency)

귀무가설이 맞다는 가정하에.. contingency table에서 각 셀에 대한 기대값을 구한 table

그룹별 차이가 없는 경우에 기대할 수 있는 빈도(frequency)인데, 그룹별로 구분하지 않고 전체를 한꺼번에 pooling해서 선호하는 비율과 선호하지 않은 비율을 구한 다음 이를 각 그룹의 sample size에서 가중해서 할당하면 됩니다.

row(1) / 전체 => 선호비율

row(2) / 전체 => 선호하지 않는 비율

ex)

남성	여성
----	----

선호	:	114명 (=200*57%)	85.5명
(=150*57%)			

선호하지 않음	:	86명 (=200*43%)	64.5명 (=150*43%)
---------	---	----------------	------------------

### 자유도(degree of freedom)

자유도는 자유롭게 움직이는 정도라는 의미인데 위의 분할표 같은 경우 1개 cell에 어떤 값이 주어지는 경우 나머지 cell의 값은 그에 따라 정해져야 하므로 자유롭게 움직일수 있는 cell은 1입니다.

#### 규칙

분할표에서, (행의개수 -1) \* (열의개수 -1)

### 휴리스틱(heuristics) 또는 발견법

시간이나 정보로 인하여 합리적인 판단을 할 수 없거나, 체계적이면서 합리적인 판단이 굳이 필요하지 않은 상황에서 사람들이 빠르게 사용할 수 있는 어림짐작의 방법이다.

'제한된 합리성'이란 다양한 의사결정 상황에서 인간의 인지적인 한계로 인해 발생하는 의사결정 문제를 인지적 한계 안에서 다룰 수 있는 범위로 축소시키고, 간단해진 과업의 수행에 한해 규범적 규칙을 이용한다는 것을 의미한다.

#### 개요

발견법은 인간과 기계에서 어떤 문제를 해결하거나 제어하기 위해 필요한 정보를 위해 느슨하게 적용시키는 접근을 시도하는 전략을 말한다.

가능한 가장 좋은 해답 혹은 최적의 해결법에 접근하기 위한 빠른 방법을 얻기 위해 특히 쓰인다.

### 검정통계량(test statistic)

통계적 가설을 검정할 목적으로 사용되는 통계량

검정 통계량은 표본 데이터에서 계산되어 가설 검정에 사용되는 랜덤 변수이다.

검정 통계량을 사용하여 귀무가설의 기각 여부를 확인할 수 있다.

검정 통계량은 데이터를 귀무 가설 하에서 기대되는 값과 비교한다.

검정 통계량은 P-값을 계산하기 위해 사용된다.

### 정보이론(information theory)

시그널에 존재하는 정보의 양을 측정하는 응용수학의 한 갈래이다.

정보이론은 무선전송을 통해 알파벳으로 된 메시지를 보내려는 연구에서 시작되었다.

정보이론은 최적의 코드를 디자인하고, 메세지의 기대 길이(expected length)를 계산하는 데 도움이 된다.

머신러닝에서는 해당 확률분포의 특성을 알아내거나 확률분포 간 유사성을 정량화하는데 쓰인다.

#### 핵심 아이디어

잘 일어나지 않는 사건(unlikely event)은 자주 발생하는 사건보다 정보량이 많다(informative)는 것이다.

- 자주 발생하는 사건은 낮은 정보량을 가진다. 발생이 보장된 사건은 그 내용에 상관없이 전혀 정보가 없다는 것을 뜻한다.

- 덜 자주 발생하는 사건은 더 높은 정보량을 가진다.
- 독립사건(independent event)은 추가적인 정보량(additive information)을 가진다. 예컨대 동전을 던져 앞면이 두번 나오는 사건에 대한 정보량은 동전을 던져 앞면이 한번 나오는 정보량의 두배이다.

### 정보량

위 세가지 조건을 만족하는 함수는 발생 가능한 사건이나 메세지의 확률분포에 음의 로그를 취한 수식이다.

확률변수  $X$ 의 값이  $x$ 인 사건의 정보량은 다음과 같다.

$$I(x) = -\log(P(x))$$

위 식에서 밑이 2인 경우의 정보량의 단위를 shannon(섀넌) 또는 비트(bit)라고 한다.

자연상수  $e$ 를 밑으로 하는 경우 내트(nat)라고 부른다.

머신러닝에서는 대개 밑을 자연상수로 사용한다.

### 섀넌 엔트로피(Shannon entropy)

각 메시지에 포함된 정보의 기대값이다.

섀넌 엔트로피(Shannon entropy)는 모든 사건 정보량의 기대값을 뜻한다.  
(정보량은 question의 수에 비례함)

일반적으로 엔트로피는 무질서도 또는 불확실성을 가리킨다. (엔트로피가 높다 = 무질서도가 높다 = 정보의 양이 크다)

전체 사건의 확률분포의 불확실성의 양을 나타낼 때 쓴다.

어떤 확률분포  $P$ 에 대한 섀넌 엔트로피  $H(P)$ 는 다음과 같다.

$$H(P) = H(x) = E_{X \sim P} [ I(x) ] = E_{X \sim P} [ -\log P(x) ]$$

<- 원래는  $\log(1/p)$ 로 계산하는 것이고, 로그 성질에 의해  $-$ 를 붙인 것이다.

### IDEA

만약 하나의 사건이 다른 사건보다 발생할 확률이 높다면 그 사건에 대한 관측이 제공할 수 있는 정보는 적다.

반대로 희귀한 사건을 관측하면 더 많은 정보를 얻을 수 있다.

그러므로 정보량(information content)은 확률에 반비례한다.

### 성질

서로 독립인 두 확률변수의 섀넌 엔트로피는 각 확률변수의 엔트로피의 합과 같다.

엔트로피는 어떤 기준으로 나누면 나누기 전의 엔트로피보다 작거나 같다.

### 엔트로피(entropy)의 이해

기호(symbol)를 알아내려면 평균적으로 몇번의 질문을 해야 하는가?

평균적으로 몇번의 질문을 해야하는지 알기위해서는 symbol이 나타날 확률분포에 근거해야함

#### 만약 등확률일 경우

ex)

symbol이 다음과 같이 존재한다면  $\rightarrow A, B, C, D // p(A) \dots p(D) = 0.25$

question의 수는 2개면 된다.

$(AB\text{인가?}) \rightarrow Y \text{ (A인가?)}$

$\rightarrow N \text{ (C인가?)}$

이를 평균으로 내도 2임

만약 확률분포가 다를경우

ex)

symbol이 다음과 같이 존재한다면  $\rightarrow A, B, C, D //$

$p(A) = 0.5, p(B) = 0.125, p(C) = 0.125,$

$p(D) = 0.25$

(A인가?)  $\rightarrow Y$

$\rightarrow N$  (D인가?)  $\rightarrow Y$

$\rightarrow N$  (C인가?)  $\rightarrow Y$

$\rightarrow N$

이를 평균으로 내면 1.75가 나옴

평균계산  $\sum p_i * (\text{symbol이 나타날수 있는 question의 개수} = \text{decision node의 개수})$

평균적인 질문의 수가 적다는 것의 의미

출력에 대한 불확실성이나 예상외의 일이 적다는 뜻

ex)

symbol의 종류는 4가지이고 symbol 100개가 입력되었다고 했을때

평균적인 question의 수가 2개라면 총 200개의 question이 필요

평균적인 question의 수가 1.75개라면 총 175개의 question이 필요  $\rightarrow$  다른 조합이 나올 일이 더 작다

### 공식

$H = \sum p_i * \log_2(\text{number of questions})$  이다.

이를 분해해보면

$(\text{number of questions}) \leftrightarrow (\text{number of outcomes}) \leftrightarrow (1/p_i)$

따라서 엔트로피는 모든 결과물이 동등하게 나오는 최대치이다.

### outcomes

symbol 100개가 주어질때 어떤 한 symbol이 나타날수 있는 빈도

이 빈도는 1/확률로 계산할수 있다.

왜 number of question이 number of outcomes와 대응 되는가?

number of outcomes에 의해 number of question이 결정되니까

즉, 결과의 개수에 의해 확률이 결정되고 이 확률은 question을 어디에 배치하느냐와 연관된다.

이 question을 어디에 배치하느냐에 의해 분류가 되는 question의 개수가 결정됨

### 엔트로피가 증감의 의미

엔트로피가 떨어진다는 것은 결과물에 대한 질문을 적게해도 된다는 것이다.

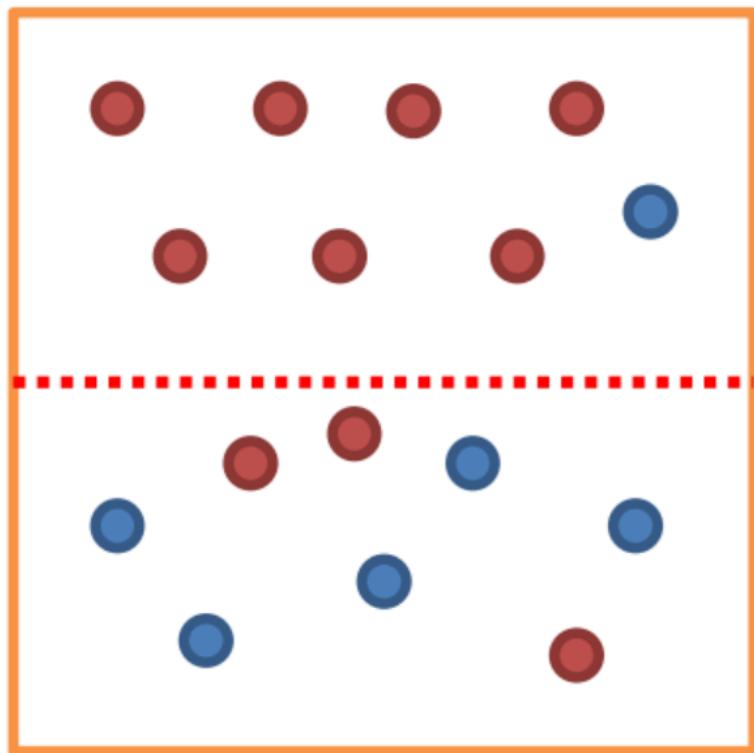
동확률의 결과로부터 멀어지거나 예측가능성을 만들어낼때, 엔트로피는

떨어진다.  
엔트로피가 증가하면 정보가 더 많다는 것을 의미

단위  
bit

log\_2는 왜 씌우는가?  
bit로 표현하기 위해

분할



오렌지색 박스로 둘러쌓인 영역 A의 엔트로피는 다음과 같다.

$$Entropy(A) = - \sum_{k=1}^m p_k \log_2 (p_k)$$

$$Entropy(A) = -\frac{10}{16} \log_2 \left(\frac{10}{16}\right) - \frac{6}{16} \log_2 \left(\frac{6}{16}\right) \approx 0.9$$

여기서 A 영역에 빨간색 점을 그어 두 개 부분집합으로 분할한다고 가정해보자  
두 개 이상 영역에 대한 엔트로피 공식은 다음과 같다.

$$Entropy(A) = \sum_{i=1}^d R_i \left( - \sum_{k=1}^m p_k \log_2(p_k) \right)$$

$$Entropy(A) = 0.5 \times \left( -\frac{7}{8} \log_2 \left( \frac{7}{8} \right) - \frac{1}{8} \log_2 \left( \frac{1}{8} \right) \right) + 0.5 \times \left( -\frac{3}{8} \log_2 \left( \frac{3}{8} \right) - \frac{5}{8} \log_2 \left( \frac{5}{8} \right) \right) \approx 0.92$$

$R_i$  = 분할 전 레코드 가운데 분할 후 i영역에 속하는 레코드의 비율

분기 전과 분기 후의 엔트로피가 변화되었다.

### 정보이득(Information Gain)

어떤 속성을 선택함으로 인해서 데이터를 더 잘 구분하게 되는 것을 말한다.

상위노드의 엔트로피에서 하위 노드의 엔트로피의 가중평균을 뺀 수치

$$Gain(A) = I(s1, s2, \dots, sm) - E(\text{속성}A)$$

정보이득의 계산은 상위노드의 엔트로피에서 하위노드의 엔트로피를 뺀 값이다.

$E(A)$ 는 A라는 속성을 선택했을 때 하위로 작은 m개의 노드로 나누어진다고 하면  
하위 각 노드의 엔트로피를 계산 한 후 노드의 속한 레코드의 개수를 가중치  
로 하여 엔트로피를 평균한 값이다.

여기서  $I(s1, s2, \dots, sm)$ 은 상위 노드 엔트로피

ex)

학생 데이터에서 수능등급을 구분하는데 있어 수학 점수가 체육 점수보다 변별력이  
높다고 하자.

그러면 수학 점수의 속성이 체육 점수 속성보다 정보이득이 높다고 말할수 있다.

하나의 바구니에 빨간색, 파란색 구슬이 혼합되어 있다고 하자  
이때 반반씩 혼합되어 있으면 엔트로피가 1이 된다.

이 바구니에 있는 구슬을 다른 바구니 2개로 나눌때 한쪽은 파란색으로 한쪽은 빨  
간색으로 나누고 싶다고 하자.

이때 잘 분류하려면 각 바구니의 엔트로피가 0이되어야함

-> 이때 분류전 바구니의 엔트로피(1)에서 분류후 바구니의 엔트로피들의 가중  
평균을 뺀 값이 정보이득이다.

### 기대값(expected value)

확률론에서, 확률 변수의 기대값은 각 사건이 벌어졌을 때의 이득과 그 사건이 벌어질  
확률을 곱한 것을 전체 사건에 대해 합한 값이다.

이것은 어떤 확률적 사건에 대한 평균의 의미로 생각할 수 있다.

### 정보(information)

하나가 다른 하나에게 영향을 미치도록 하는 것

정보는 기본적인 단위로 측정할 수 있다.

#### 엔트로피

정보의 측정방법

### 정보이론(information theory)

#### 현대 정보이론

속도 개선을 위해 기계를 만들 - 19세기

일차적 부호를 사용자가 입력하게해서 기계로 하여금 전기 펄스 같은 낮은

단계의 신호로 자동 변환시키는 기계 – 이를 이차적부호라 한다.  
기계는 시각 공급원으로 움직일수 있는데 정확하고 빠른 펄스 흐름을 생성한다 – 바로 다중 방식  
이는 셔터전신과 같은 개념으로 만들어짐 – 5개 키로 만들어짐, 각각의 조합은 고유한 메시지를 나타낸다. ( $2^5 = 32$ )  
가장 낮은 단계에서 이 시스템은 *clock*을 이용해 꺼져있거나 켜져있는 전류를 차례대로 서로 교환시킨다.  
신호 전달의 속도는 펄스 사이의 최소 공간에 의해 물리적으로 제한되었다.  
펄스가 너무 빨리 전달되면 신호 사이에 혼란

### symbol(부호)

보이는 어떤 상태가 오래 지속된 신호(*signal*)

### 신호(*signal*)를 보내는것

신호를 보내는 것은 단순히 한 상태에서 다른 상태로의 변화를 나타낸다.

### symbol rate(심벌 레이트)

1초에 여러가지를 나타낼수 있는 신호의 개수

너무 신호를 빨리보내면 혼동이 오므로 제한을 두어야함

### 통신용량(*channel capacity*)

초당 몇개의 신호를 전달할수 있는가?(n)

심볼당 몇 개의 차이점을 만들수 있는가?(s)

이러한 한도는 확률의 *decision tree*로 생각할수 있다.

각 심볼은 하나의 *decision*이라고 생각할 수 있고, 가지의 수는 차이의 수(s)로 볼수 있기 때문이다.

n개의 심볼 다음에 우리는 s의 n제곱의 잎이 있는 트리를 갖는다.

이 트리를 통하는 모든 경로가 메시지를 상징한다.

잎의 수를 메시지 공간의 크기라고 생각할 수 있다.

### 통신시스템의 용량을 증대시킬수 있는 방법 -> symbol에 차이를 두자 – 20 세기

서로 다른 신호 현상의 수를 증가시킬 수 있다.

약한 전자와 강한 전자를 이용해 세기가 다른 신호를 만들수 있다. – 토마스 에디슨

앞과 반대의 전류 두방향 사용 – 에디슨

### 보낼수 있는 서로다른 신호를 증가시켰을때의 문제점

잡음(*noise*)에 의해 제한됨

*noise*가 신호를 방해하지 않도록 크게 만들어야함

### 이를 통해 얻은 결과

차이가 0과 1일때, *symbol*을 여러개 전송 0과 1의 조합

으로 어떤 언어를 정의하는 것을 뛰어넘어

다양한 차이로 *symbol*을 보낼수 있게 됨

따라서 단위 시간에 표현할수 있는 언어가 많아짐

## 변량(variate)

주로 확률/통계학에서 많이 쓰이는 용어로써, 조사 대상의 특징, 성질을 숫자 또는 문자로 나타낸 것

## 지니계수(gini coefficient, gini index)

전체 Box안에서 특정 Class에 속하는 관측치의 비율을 모드 제외한 값

불순도(Impurity), 다양성(Diversity)를 계산하는 방법이다.

### 수학적 공식

J클래스를 지니는 요소들의 집합이 있다고 가정해보자.

$p_j$ ,  $j \in \{1, 2, \dots, J\}$  는 집합에서 클래스  $j$ 를 지닌 요소를 선택할 확률, 또는 집합내의 클래스  $j$ 를 가지는 요소의 비율로 정의하자.

그러면 지니 불순도는 다음과 같이 정의할 수 있다.

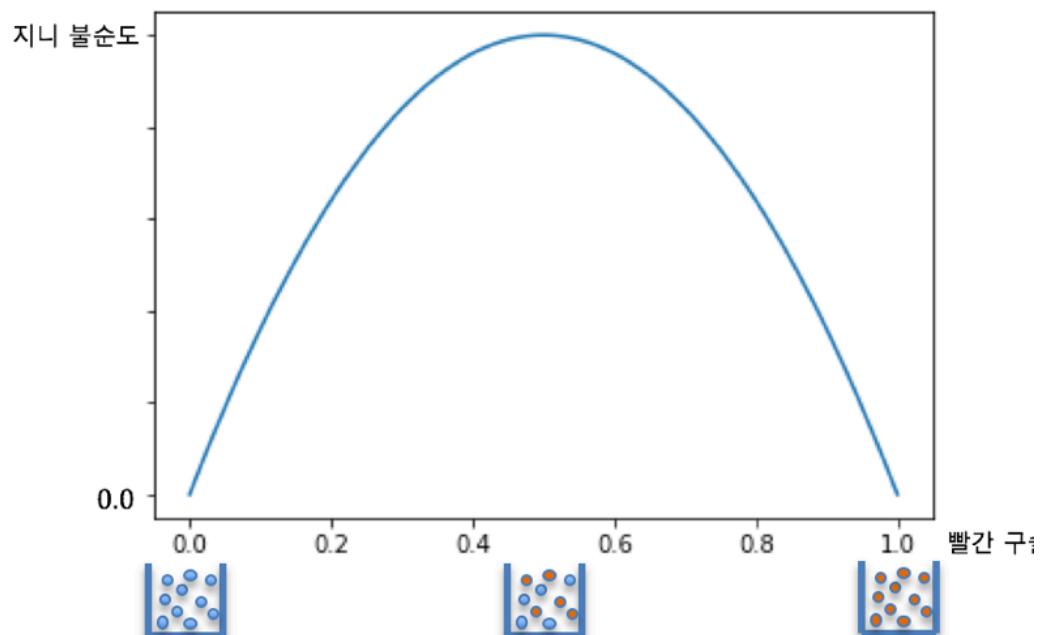
$$\sum_{j=1}^J p_j(1 - p_j)$$

ex)

만약  $j$ 가 빨간색 혹은 파란색에 속한다면 이경우  $J = 2$ 이다.

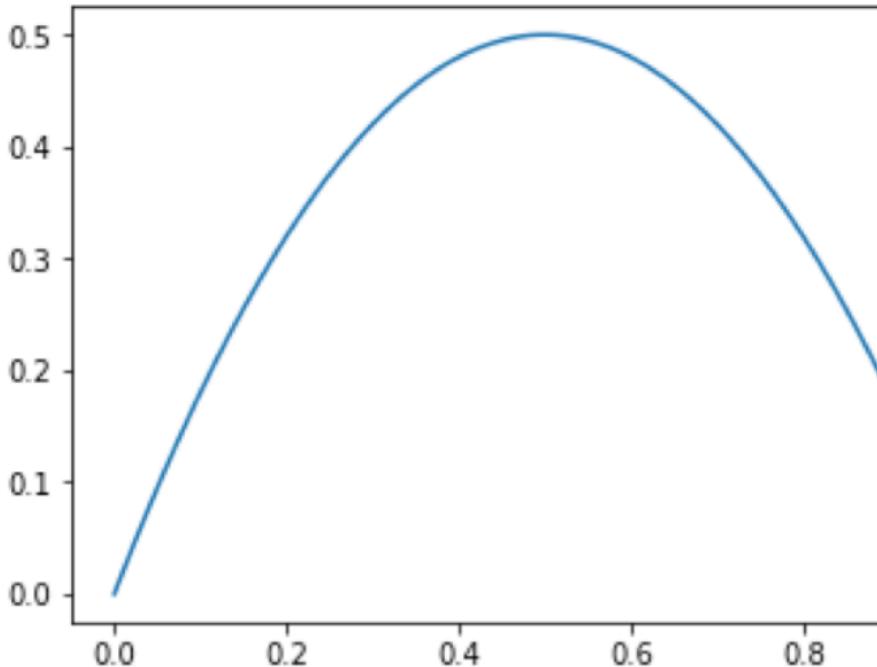
만약 구슬이 120개 들어있고 그중 37개가 빨간색 구슬이고, 나머지 73개가 파란색 구슬이라면 지니 불순도는 다음과 같이 계산된다.

$$(37/120) * (1 - 37/120) + (73/120) * (1 - 73/120) = 0.427$$



왜 지니 불순도 공식은 이런 수학적 형태를 요구하는가?

이 공식은 우리가 원하는 곡선의 모양을 그린다. (양쪽 극단이 0이고 중간이 두드러진 그래프)



확률론적 해석에 따라서도 동기부여를 할수 있다.

특정한 클래스 분포를 갖는 일련의 집합에 대한 지니 불순도는 동일한 또 다른 집합이 주어질 때 두 집합에서 서로 각기 다른 클래스의 요소를 선택하는 확률로 볼수있다.

또는 동일하게, 한 개의 집합에서 대체 샘플링(sampling with replacement)을 두번 할 경우 두 개의 요소가 서로 다른 클래스를 지닐 확률이다.

ex)

항아리에 빨간색 구슬과 파란색 구슬의 수가 거의 없는 경우 빨간색 구슬을 샘플링 한 후에 다음 샘플링이 파란색일 확률은 비교적 높지만,

항아리 속에 빨간색 구슬이 거의 없고 구슬 대부분이 파란색이라면 거의 항상 두번의 샘플링은 파란색 구슬만 골라내게 될것이다.

한가지 색이 대부분인 경우 이렇듯 두번의 샘플링에서 색이 바뀔 확률을 높지 않을것이다.

따라서 직관적으로 이 확률은 집합의 불순물을 정량화하는 합리적인 방법이라고 볼수 있다.

구슬의 예의 경우 색이 바뀌는 경우는 처음 샘플링이 빨간색이고 다음이 파란색일 경우와 처음이 파란색이고 다음이 빨간색일 경우 두 가지이다. 이 확률은 다음과 같이 나타낼 수 있다.

$\text{Prob}(\text{빨간색})\text{Prob}(\text{빨간색과 다른 색}) + \text{Prob}(\text{파란색})\text{Prob}(\text{파란색과 다른 색})$

이것은 2개 이상의 색으로도 쉽게 일반화 할 수 있다.

$\text{Prob}(\text{빨간색})\text{Prob}(\text{빨간색과 다른색}) + \text{Prob}(\text{파란색})\text{Prob}(\text{파란색과 다른 색}) + \text{Prob}(\text{초록색})\text{Prob}(\text{초록색과 다른 색}) \dots$   
etc

그리고 이 확률은 지니 불순도 공식이 설명하는 것과 정확히 같다.

$$\sum p_j * (1 - p_j)$$

### 분할된 경우

분할이 d개 만큼 된 경우 다음과 같은 공식을 이용한다.

$$G.I(A) = \sum_{i=1}^d \left( R_i \left( 1 - \sum_{k=1}^m p_{ik}^2 \right) \right)$$

이 공식은,

$$\sum_{j=1}^J p_j(1 - p_j)$$

여기에서  $\sum p_j(1 - p_j)$  를 풀면  $\sum p_j - \sum p_j^2$  이 되고

$\sum p_j = 1$  이 되서,  $1 - \sum p_j^2$  이 된다.

그리고 d 개 만큼 분할되었으므로 이에 대한 가중평균을 계산한다.

$R_i$  : 분할 후 i 영역에 속하는 레코드의 비율

$p_{ik}$ : i번째 분할된 영역 중, k클래스에 속하는 레코드의 비율

### 지니 불순도 측정(Gini Impurity Measure)

집합의 "순도"를 측정한다.

ex)

항아리에 10개의 구슬이 들어 있고 그 중 절반가량이 빨간색이고 나머지 절반가량이 파란색인 경우 그 구슬들의 집합은 빨간색과 파란색이 섞여있어 불순한 것으로 간주한다.

반면 항아리에 빨간색 또는 파란색 구슬만 있는 경우 그 구슬 집합은 완벽하게 순수한 것으로 간주한다.

양쪽 끝에서 지니 불순도 0값에 시작하여, 항아리의 빨간색 파란색 구슬의 수가 같은 경우 지니 불순도는 최댓값에 도달한다.

(구슬 모두가 빨간색 또는 파란색일때 모든 집합은 완벽하게 순수하다  $\Leftrightarrow$  지니 불순도 = 0)

### 범위

0~0.5 의 값을 갖는다.

0.5(가장 불순도가 큰 값)

0(가장 순수한 값)

### 원-핫 인코딩(one-hot encoding)

단 하나의 값만 True이고 나머지는 모두 False인 인코딩을 말한다.

#### 장점

카테고리를 숫자로 표현할때 문제가 있을수 있다.

예를 들어

10개의 카테고리가 있고 이를 0~9까지 표현하자

그런데 각각의 카테고리에 대해 똑같은 거리 값을 주고 싶다고 하자.

그런데 0과 4는 4만큼 거리가 떨어져있다.

이러한 문제는 일반적으로 카테고리별 이진 특성을 만들어 해결한다.

즉, 하나의 값이 1이고 나머지는 0이 되는 행렬을 만들어서 각각의 카테고리에 대해 똑같은

거리를 갖게 한다.

### 연관성 분석(association analysis)

데이터 내부에 존재하는 연관성, 즉 항목 간의 상호 관계 혹은 종속 관계를 찾아내는 분석

연관성 분석의 가장 잘 알려진 예로는 고객이 구매한 장바구니를 살펴봄으로써 거래되는 상품들의 연관성(규칙)을 발견하고 이를 분석함으로써 관련 상품끼리의 묶음 상품, 어떤 상품을 사기 위해 가는 동선에 관련 상품 진열한다는 등의 행위를 들 수 있다.

때문에 마케팅 분야에서는 연관성 분석을 장바구니 분석(basket analysis)라고도 한다.

#### 장바구니 분석(basket analysis)

기본 단위는 item

하나 이상의 item 그룹은 대괄호로 묶어서 아이템집합(itemset)을 이룬다.

장바구니 분석의 결과는 연관 규칙(association rule)의 모음으로, 아이템집합 사이의 관계에 존재하는 패턴을 명시한다.

#### 연관규칙(association rule)

아이템 집합 사이의 관계에 존재하는 패턴을 명시한다.

아이템 집합의 부분집합으로 구성된다.

ex)

{ peanut butter }  $\rightarrow$  {bread}

LHS(left-hand side)

왼쪽에 있는 아이템의 집합

규칙을 실행하기 위해 만족해야 하는 조건

#### RHS(right-hand side)

오른쪽에 있는 아이템의 집합

조건을 만족했을 때 기대하는 결과

#### 연관 규칙 학습자

자율적이기 때문에 알고리즘이 훈련될 필요가 없다.

다시말해 사전에 레이블될 필요가 없다.

단점은, 학습자에 대해 정성적으로 유용성을 평가하는 것 외에는 규칙 학습자의 성능을 객관적으로 측정하기 쉬운 방법이 없다는 점이다.

#### 강한 규칙(strong rules)

높은 지지도와 신뢰도를 갖는 규칙

#### 장점

대규모 거래 데이터에 대해 작업을 할 수 있다.

이해하기 쉬운 규칙을 생성한다.

데이터마이닝과 데이터베이스에서 예상치 못한 지식을 발굴하는데 유용하다.

#### 단점

작은 데이터셋에는 그다지 유용하지 않다.

진정한 통찰력과 상식을 분리하기 위한 노력이 필요하다.

#### 연관성을 일반화하거나 신뢰할 수 있는 판단 기준

### 지지도(support)

함께 구매가 일어나는 품목의 거래가 전체의 거래 수 중 얼마나 빈번히 일어난다는가를 측정하는 것

지지도 = A상품과 B상품을 같이 구입한 횟수 / 전체 구매 횟수

※ 특정사건에 대한 확률임

#### 목적

지지도를 구하는 목적은, 우리가 구한 추천 데이터가 과연 올바른가?  
라는 의문에서 시작한다

ex)

1000개의 물품이 있다. 총 1만번의 거래가 있었는데, 이때 A라는 상품은 단 한번밖에 거래가 안됐다고 가정을 해보자  
이때, A상품에 대한 데이터는 과연 효율성이 있을까?

### 신뢰도(confidence)

아이템 또는 아이템집합 X의 존재가 아이템 또는 아이템집합 Y의 존재를 유발하는 거래의 비율

신뢰도는 품목 A의 구매가 일어났을 때 품목 B가 함께 일어난 거래 수 즉, 품목 A를 구매한 전체 거래 건수 중 품목 A와 B가 동시에 일어난 구매 건수의 비율을 말한다.

A상품에 대한 B상품의 신뢰도 = A상품과 B상품의 동시출현 횟수 / A상품의 출현 횟수

신뢰도가 1에 가까울 수록 의미있는 연관성을 갖는다.

※ A에 대한 B의 조건부 확률임

$\text{support}(x, y)$

$$\text{confidence}(x \rightarrow y) = \frac{\text{support}(x, y)}{\text{support}(x)}$$

#### 목적

무언가의 관계를 구하기 위해서 어떻게 계산을 할것인가?

### 향상도(Lift, Improvement)

품목간의 의미를 파악하기 위해서 사용이 된다.

$$L = \frac{P(Y | X)}{P(Y)} = \frac{P(X \cap Y)}{P(X)P(Y)}$$

ex)

A라는 상품에서 신뢰도가 동일한 상품 B, C가 존재할때 우리는 어떤 상품을 더 추천해야 좋을까?

A가 총 5번 등장했다면, B와 C는 A와 동시에 2번 등장했음을 알수 있다.

전체 구매리스트는 총 50개이고, 여기서 B는 총 5번 등장했고, C는 총 10번 등장했다면, 어떤 상품을 추천해야 옳을 것인가?

$$L(B, A) = \text{신뢰도}(B \rightarrow A) / \text{지지도}(A)$$

### lhs(left hand side), rhs(right hand side)

$A \rightarrow B$  : A를 사면, B를 살것이다 와 같이 표현할때

이때 A는 lhs, B를 rhs라 부른다.

즉, 조건부 확률에서 |(given) 기호 오른쪽에 있는 것을 lhs <- given되는 조건, 왼쪽에 있는것을 rhs라 함

## apriori algorithm

집합의 크기가 1인 경우 부터 차례로 늘려가면서 처리한다.

집합의 크기가 k개인 항목을 구했다면 그 다음에는 k+1인 항목의 집합을 계산한다. (빈도수의 제한을 두며 계산)

그래서 총 최대개수를 가진 빈도 항목까지 반복한다.

### apriori 알고리즘이 필요한 이유

잠재적인 아이템 집합의 개수에 따라 기하급수적으로 증가한다.

만약 집합에 나타나거나 나타나지 않을 수 있는 k개 아이템이 있다면 잠재적인 규칙이 될 수 있는  $2^k$ 개의 가능한 아이템이 존재한다.

왜냐.. 아이템이 존재하거나 존재하지 않을 경우 = 2이고 그 경우가 k개 만큼 있으니까  $2^k$  만큼 존재할수 있음

좀더 똑똑한 알고리즘은 아이템 집합을 하나씩 평가하는 대신 많은 잠재적인 아이템 조합이 실제 드물게 발견된다는 사실을 이용한다.

흔치 않은 (아마 덜 중요한) 조합을 무시함으로써 규칙의 검색 범위를 좀 더 다루기 쉬운 크기로 제한한다.

### 이름의 유래

알고리즘이 빈번한 아이템집합의 속성에서 단순한 사전의(즉, 선형의 a priori) 믿음을 이용하기 문이다.

즉, 빈번한 아이템은 빈번한 아이템의 모든 부분집합도 빈번해야 한다. (빈 번한 아이템의 부분집합의 흥미도가 최소 임계치를 만족시켜야함)

### 흥미도(interestingness)

거래 데이터베이스에서 연관 규칙을 찾아내기 위한 통계 척도(지지도, 신뢰도, lift)

### wiki

Apriori는 frequent item set을 mining 하기 위한 알고리즘입니다.

그리고 관계형 데이터베이스에 대한 연관 규칙 학습을 위한 알고리즘입니다. 이는 데이터베이스에서 빈번한(frequent) 개별 항목을 식별하고, 해당 item set가 데이터베이스에서 충분히 자주 나타나는 한, item을 놀려갑니다.

Apriori가 결정한 빈번한 item set은 데이터베이스의 일반적인 경향을 강조하는 연관 규칙을 결정하는데 사용할 수 있습니다.

여기에는 market basket analysis와 같은 영역의 응용이 있습니다.

### 개요

Apriori 알고리즘은 1994년 Agrawal과 Srikant에 의해 제안되었습니다.

Apriori는 트랜잭션이 포함된 데이터베이스(ex: 고객이 구매한 품목 모음 또는 details of a website frequentation or IP addresses)에서 작동하도록 설계되었습니다.

다른 알고리즘은 트랜잭션이 없거나(Winepi and Minepi) 데이터에 타임 스탬프가 없는 (DNA sequencing), 데이터에서 연관 규칙을 찾기 위해 설계되었습니다.

각 transaction은 일련의 품목(itemset)으로 간주됩니다.

임계 값 C가 주어지면 Apriori 알고리즘은 데이터베이스에서 적어도 C를 만족하는 subset인 item set를 식별합니다.

Apriori는 frequent subset이 한번에 한 item씩 확장되고

(candidate generation으로 알려짐) 후보 그룹이 데이터에 대해 테스트 되는 'bottom up' 접근 방식(bottom up approach)을 사용합니다.  
확장이 더이상 발견되지 않으면 알고리즘이 종료됩니다.

Apriori는 너비 우선 검색과 해시 트리 구조를 사용하여 candidate item set를 효율적으로 계산합니다.  
후보 item set를 길이가 k-1인 item set에서 길이가 k인 candidate item set를 생성합니다.  
그런 다음 infrequent(빈번하지 않은) sub pattern을 제거 합니다.  
하향 폐쇄 정리(downward closure lemma)에 따르면, candidate set은 모든 빈번한 k길이 항목 세트가 포함됩니다.  
그런 다음, 트랜잭션 데이터베이스를 스캔하여 후보 중에서 빈번한 항목 세트를 판별합니다.

### 의사코드

알고리즘의 의사 코드는 트랜잭션 데이터베이스 T 그리고 임계값 업실론에 대해 제공  
일반적인 집합 이론 표기법이 사용되지만 T는 다중집합이다.  
C\_k는 k레벨에 설정된 후보이다.  
각 단계에서, 알고리즘은 이전 레벨의 큰 아이템 세트로부터 후보 세트를 생성하는 것으로 가정하여 하향 폐쇄 정리를 주의한다.  
count[c]는 후보 세트 c를 나타내는 데이터 구조의 필드에 액세스하며, 처음에는 0으로 가정된다.  
많은 세부 사항은 아래에서 생략된다.  
일반적으로 구현의 가장 중요한 부분은 후보세트를 저장하고 빈도를 계산하는데 사용되는 데이터 구조이다.

### IN R

```
arules library
ex)
library(arules)

buylist = list(c('우유', '버터', '시리얼'),
               c('우유', '시리얼'),
               c('우유', '빵'),
               c('버터', '맥주', '오징어'))

buylist = as(buylist, 'transactions')
inspect(buylist)
buyresulrt <- apriori(buylist)
apriori()
규칙을 생성한다.
apriori( [matrix 혹은 transaction형태] )
parameter
지지도, 신뢰도가 일정 수준 이상인 규칙을 필터링 할수 있
```

다.

ex) `buyresult <- apriori(buylist, parameter=list(conf=0.5))`

### `inspect()`

규칙을 상세히 볼수있다.

`inspect( arules )`

ex)

`inspect(buyresult)`

`inspect(subset(buyresult, subset=support>=0.5))`

`inspect(subset(buyresult, subset=lhs %in% c('버터', '시리얼')))`

`inspect(subset(buyresult, subset=lhs %ain% c('버터', '시리얼')))` <- %ain%는 모든 요소가 들어 있는것

`inspect(subset(buyresult, subset=lhs %oin% c('버터', '시리얼')))` <- %oin%는 한 요소만 순수하게 들어잇는것

`inspect(subset(buyresult, subset=lhs %pin% "우"))` <- %pin% 는 패턴 매칭

### `itemFrequencyPlot()`

`itemFrequencyPlot( arules )`

규칙에 대한 연관성을 표로 보여줌 (y축 지지도, x축 항목)

ex)

`itemFrequencyPlot(buylist)`

`itemFrequencyPlot(buylist, support = 0.5)`

`support`

지지도 제한

### `arulesViz`

연관성을 네트워크 그림으로 그려줌

ex)

`plot(buyresult, method='graph')`

`plot(buyresult, method='grouped')`

## IN PYTHON

ex)

`import pandas as pd`

`from mlxtend.preprocessing import TransactionEncoder`

`from mlxtend.frequent_patterns import apriori`

`from mlxtend.frequent_patterns import association_rules`

`buylist = [['우유', '버터', '시리얼'],`

`['우유', '시리얼'],`

`['우유', '빵'],`

`['버터', '맥주', '오징어']]`

```

t = TransactionEncoder()
t_ary = t.fit(buylist).transform(buylist)
df = pd.DataFrame(t_ary, columns=t.columns_)

item = apriori(df, min_support=0.5, use_colnames=True)

from mlxtend.frequent_patterns import
    association_rules

a = association_rules(item)
a.info()

#####
build = pd.read_csv('c:/data/building_utf8.csv')
build = build.iloc[:, 1:]
build = build.fillna(0)
build = build.astype(bool)

item = apriori(build, min_support=0.3,
    use_colnames=True)
association_rules(item)
    mlxtend library
        apriori()
            from mlxtend.frequent_patterns import apriori
apriori 알고리즘으로 itemset을 계산후 dataframe으로
    출력함
이때 입력값은 0과 1을 요소로 갖는 dataframe
apriori( [dataframe] )

    TransactionEncoder()
        from mlxtend.preprocessing import
            TransactionEncoder
apriori 함수를 적용하기 위해, transaction을 만드는 녀
   석
Python list의 트랜잭션 데이터에 대한 Encoder class
Python list의 list 형식의 트랜잭션 데이터를 Numpy 배열
   로 인코딩한
TransactionEncoder 객체를 이용해서 데이터 세트를 일반적
    인 머신 러닝 API에 적합한 array 형식으로 변환할수 있
    다.
fitting 한 후에는 columns_ 속성을 통해 위에 표시된 데이
   터 배열에 해당하는 고유한 열 이름에 액세스 할 수 있다.
ex)
dataset = [['Apple', 'Beer', 'Rice',
    'Chicken'],
    ['Apple', 'Beer', 'Rice'],
    ['Apple', 'Beer'],
    ['Apple', 'Bananas'],
    ['Milk', 'Beer', 'Rice'],

```

```

        'Chicken'],
        ['Milk', 'Beer', 'Rice'],
        ['Milk', 'Beer'],
        ['Apple', 'Bananas']])
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
s = pd.DataFrame(te_ary, columns=te.columns_)

```

### 시각화

#### 네트워크

```

ex)
build <- read.csv('c:/data/building.csv', header = T,
                  stringsAsFactors = FALSE)
b2 <- t(as.matrix(build)) %*% as.matrix(build)
b2.w <- b2 - diag(diag(b2))

library(sna)
library(rgl)
library(ggplot2)

gplot(b2.w,
      displaylabel = T,
      vertext.cex=1,
      vertext.col='green',
      edge.col='blue',
      edge.lty='dotted',
      edge.lwd=0.1,
      arrowhead.cex = 0.3,
      label.pos = 0)

```

### 군집화(cluster)

데이터에서 패턴을 발견하는 것(spotting patterns)

데이터 클러스터(cluster)로 자동 분리하는 비지도학습의 머신러닝이다.

군집화는 데이터 안에서 발견되는 자연스런 그룹에 대한 통찰력을 제공한다.

클러스터 안에 있는 아이템들은 서로 아주 비슷해야 하지만 클러스터 밖에 있는 아이템과는 아주 달라야 한다.

자율분류로 언급되는 것은 레이블이 없는 데이터를 분류하기 때문이다.

#### IDEA

군집화는 클러스터 안에 있는 아이템들은 서로 아주 비슷해야 하지만 클러스터 밖에 있는 아이템과는 아주 달라야 한다는 원칙을 따른다.

유사성(similarity)의 정의는 애플리케이션에 따라 달라질 수 있지만 군집화의 기본 아이디어는 언제나 같다.

다시 말하면, 클러스터 내의 차이를 최소화하고 클러스터 간의 차이를 최대화 하는 것!

#### 군집화를 위해서

예시들이 얼마나 밀접하게 연관돼 있는지를 측정해서 동질적인 그룹을 식별할 수 있다.

### 군집화 활용범위

찾고 있는 것이 무엇인지 모르기 때문에 군집화는 예측보다는 지식의 발견에 사용된다.

마케팅 캠페인을 위해 유사한 인구 통계나 구매 패턴을 가진 그룹으로 고객을 세분화한다.

알고 있는 클러스터 밖의 사용 패턴을 찾아 무단 네트워크 침입과 같은 이상행동을 탐지한다.

유사한 값을 갖는 특징을 적은 개수의 동질적인 범주로 그룹핑해 초대형 데이터 셋을 단순화 할 수 있다.

### 머신 러닝 작업으로서 군집화

개념적으로 모델은 데이터 내에 존재하는 패턴을 설명한다.

이와 대조적으로 군집화는 새로운 데이터를 생성한다.

데이터 내의 관계에서 전적으로 추론된 클러스터 레이블이 레이블이 없는 예시에 제공된다.

이런 이유로 가끔은 군집화 작업이 자율 분류(unsupervised classification)으로 언급되는 것을 볼 텐데, 어떤 의미에서 군집화는 레이블이 없는 예시를 분류하기 때문이다.

주목할 점은 자율 분류기에서 얻은 클래스 레이블은 본질적인 의미가 없다는 것이다.

군집하는 어떤 예시 그룹이 긴밀하게 연관돼 있는지는 말해주지만 실행 가능하고 의미 있는 레이블을 적용하는 것은 사람의 책임이다.

### 응집도

데이터가 얼마나 응집해있나..

### k-means algorithm

n개의 예시를 k개의 클러스터 중 하나에 할당하는데 이때 k는 사전에 결정해야 한다.

처음에 k개의 군집 좌표를 랜덤하게 찍고

군집 좌표와 점(데이터) 끼리 거리 계산을 해서 점을 편입시키고 한 군집으로 편입시키고(Clustering assignment)

군집 좌표의 좌표를 군집된 점들끼리의 평균으로 옮김(move centroid)

.. 반복

## K-means algorithm

Input:

- $K$  (number of clusters) ←
- Training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  ←

$x^{(i)} \in \mathbb{R}^n$  (drop  $x_0 = 1$  convention)

## K-means algorithm

$$\mu_1 \quad \mu_2$$

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster  
Assignment  
step
for  $i = 1$  to  $m$   
 $c^{(i)}$  := index (from 1 to  $K$ ) of cluster centroid  
 closest to  $x^{(i)}$ 
 $\min_{c^{(i)}} \|x^{(i)} - \mu_k\|^2$ 
  
for  $k = 1$  to  $K$   
 $\rightarrow \mu_k$  := average (mean) of points assigned to cluster  $k$ 
 $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$      $\rightarrow c^{(1)}=2, c^{(2)}=2, c^{(3)}=2, c^{(4)}=2$ 
  
}
 $\mu_2 = \frac{1}{4} [x^{(1)} + x^{(2)} + x^{(3)} + x^{(4)}] \in \mathbb{R}^n$

k-means는 이상치데이터에 민감하다.

이상치 데이터로 인해 중심이 크게 틀어질수 있음

만약 클러스터에 할당된 점이 없는 경우

이 경우는 해당 클러스터를 삭제하거나(보통 이렇게 씀),

꼭  $k$ 개의 클래스가 필요한 경우  $\mu_k$ 를 random하게 재배치함으로써 해결한다.

optimization objective, Cost function, distortion function

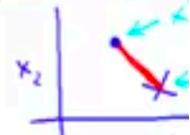
## K-means optimization objective

$\rightarrow c^{(i)}$  = index of cluster ( $1, 2, \dots, K$ ) to which example  $x^{(i)}$  is currently assigned

$\rightarrow \mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )       $K$        $k \in \{1, 2, \dots, K\}$   
 $\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned       $x^{(1)} \rightarrow 5$        $c^{(1)}=5$        $\mu_{c^{(1)}} = \mu_5$

Optimization objective:

$$\rightarrow J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

$$\min_{\substack{c^{(1)}, \dots, c^{(m)}, \\ \mu_1, \dots, \mu_K}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$


$c^{(i)}$ 는  $x^{(i)}$ 에 할당된 클러스터

$\mu_k$ 는  $k$  클러스터의 centroid

$\mu_{c^{(i)}}$ 는  $c^{(i)}$ 에 할당된 클러스터의 centroid

이에 대한 목표함수는 데이터와 centroid의 거리를 최소화하는  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ 를 선택하는 것이다.

이러한 목표함수는 k-means 알고리즘에서 찾아볼수 있는데,

1) k-means 알고리즘은 clustering assignment시에 가장 가까운  $\mu_k$ 를 선택함으로써 목표함수를 최소화 시킨다.

이는 목표함수가 최소화 되는  $c^{(1)} \dots c^{(m)}$ 을 선택하는 것이고

2) 클러스터의 중심점을 average 함으로써 목표함수를 최소화시킨다.

이는 목표함수가 최소화 되는  $\mu_1 \dots \mu_K$ 를 선택하는 것이다.

### K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster assignment step  
Minimize  $J(\dots)$  wrt  $(c^{(1)}, c^{(2)}, \dots, c^{(m)})$  (holding  $\mu_1, \dots, \mu_K$  fixed)

for  $i = 1$  to  $m$   
 $c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

for  $k = 1$  to  $K$   
 $\mu_k :=$  average (mean) of points assigned to clu

}

minimize  $J(\dots)$  wrt  $\mu_1, \dots, \mu_K$

Random initialization

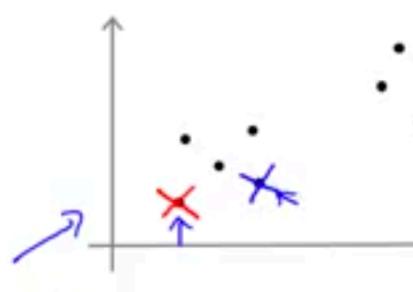
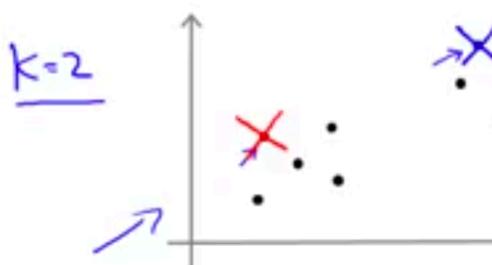
### Random initialization

Should have  $K < m$

Randomly pick  $K$  training examples.

Set  $\mu_1, \dots, \mu_K$  equal to these  $K$  examples.

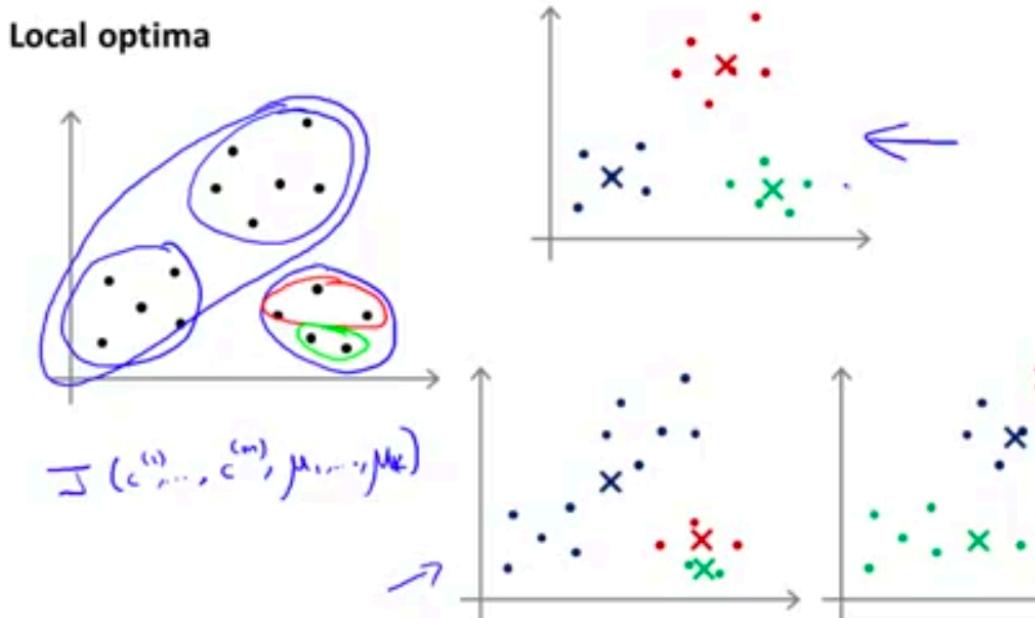
$$\begin{aligned}\mu_1 &= x^{(i)} \\ \mu_2 &= x^{(j)} \\ &\vdots\end{aligned}$$



K-means를 사용할때 초기값 설정은 보통 위와 같이 한다.

즉, 데이터셋에서 random하게 골라서 초기 클러스터의 centroid로 삼는다.

Random initialize and local optima



$k$ -means는 여러개의 해를 가질수 있다.  
이는 클러스터링 하는 초기값에 따라 결정된다.

해결방법

### Random initialization

```
For i = 1 to 100 {
    Randomly initialize K-means.
    Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ .
    Compute cost function (distortion)
     $\rightarrow J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$ 
}
```

Pick clustering that gave lowest cost  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

$k=2-10$

이를 해결하는 방법은 그냥  $k$ -means를 여러번 돌려보는 것이다.  
그리고 가장 작은 값을 가지는 목적함수의 파라미터를 선택한다.

그런데 이는 클러스터가 2~10개일대 잘 작동한다고 하고, 클러스터가 너무 많으면 잘 안된다고 한다.

### K-means의 장단점

비통계적 용어로 설명될 수 있는 간단한 원리를 사용한다.  
매우 유연하며, 간단한 조정으로 거의 모든 단점을 해결하며 적응될 수 있다.

실제 많은 사용 사례에서 충분히 잘 실행되고 있다.

#### 단점

임의의 우연(random chance) 요소를 사용하기 때문에 최적의 클러스터 집합을 찾는것이 보장되지 않는다.

더 최신의 군집화 알고리즘만큼은 정교하지 않다.

데이터에 존재하는 자연스런 클러스터 개수에 대한 합리적인 추측이 필요하다.

비구형(non-spherical) 클러스터 또는 밀도가 크게 변하는 클러스터에는 이상적이지 않다.

#### Process

##### 초기 할당 단계

$k$ -평균 알고리즘은 특정 공간에서 클러스터의 중심으로 제공할  $k$ 개의 점을 선택하면서 시작한다.

보통 이 점들은 훈련 데이터셋에서  $k$ 개의 예시를 임의로 골라서 선택한다.

$k$ -평균 알고리즘은 클러스터 중심의 출발 위치에 매우 민감한데, 임의의 우연이 최종 클러스터 집합에 상당한 영향을 미친다는 것을 의미한다.

그래서 초기 클러스터 중심을 선택하는 것이 중요한데.. 이를 위해 다양한 방법이 사용되고 있다.(랜덤 등등..)

초기 클러스터 중심을 선택한 이후 각 예시는 거리 함수에 따라 가장 가까운 클러스터 중심에 할당된다.

##### 수정 단계

클러스터 수정의 첫 단계는 초기 중심을 새로운 위치로 이동시키는 것으로, 이 새로운 위치를 중심점(centroid)이라고 하며, 해당 클러스터에 현재 할당된 점들의 평균 위치로 계산된다.

##### 종료 단계

수정 단계를 진행하다가, 점들이 재할당이 되지 않을 경우  $k$ -평균 알고리즘은 중단된다.

#### 적합한 클러스터 개수 선택

$k$ -means는 클러스터 수에 매우 민감하다.

클러스터 수를 선택할 때는 섬세한 균형이 필요하다.

$k$ 를 아주 크게 설정하면 클러스터의 동질성이 향상되며, 동시에 데이터에 과적합될 위험을 무릅써야 한다.

이상적으로는 실제 그룹에 대한 선형적(a priori) 지식이 있을 것이며, 이 정보를 클러스터 개수를 선택하는 데 적용할 수 있다.

ex) 아카데미 시상식에서 고려하는 장르 개수와 동일하게  $k$ 를 설정

##### 사전 지식이 없다면?

$k$ 를  $n/2$  의 제곱근과 동일하게 설정하게 제안한다.

이때  $n$ 은 데이터셋의 예시 개수다.

하지만 이 경험 법칙은 대형 데이터셋의 경우 다루기 힘들 정도로 많은 클러스터를 만들어 낼 수 있다.

#### Elbow method

그냥 한번 해보는 방법임

엘보법으로 알려진 이 기법은 다양한  $k$ 값에 대해 클러스터 내의 동질성(homogeneity)과 이질성(heterogeneity)이 어떻게 변하는지 측정한다.

동질성은 클러스터가 추가되면서 증가할 것으로 예상되고, 이질성은 더 많은 클러스터와 함께 계속해서 감소할 것이다.

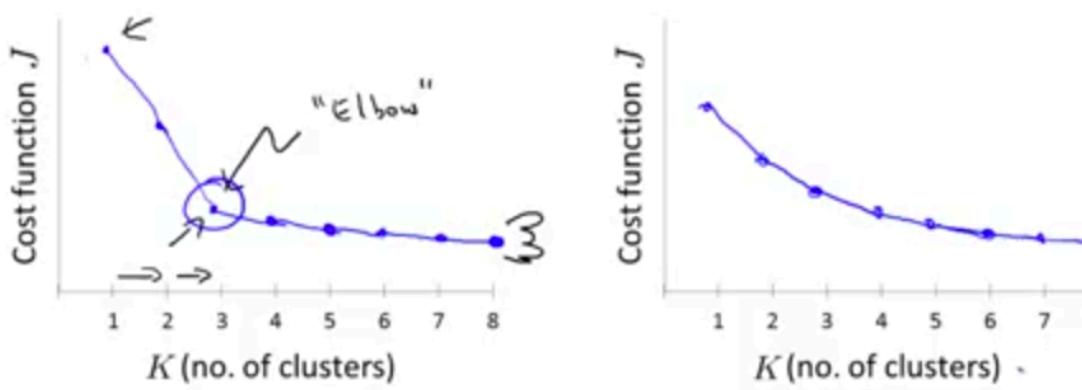
각 예시가 자신의 클러스터에 있게 될 때까지 계속해서 개선되는 것을 볼 수 있기 때문에 목표는 동질성을 최대화하고 이질성을 최소화하는 것이 아니라 특정 포인트를 넘으면 결과가 약화되는  $k$ 를 찾는 것이다.

이  $k$  값을 엘보 포인트(elbow point)라 한다.

그런데 대부분의 군집화 응용에서는 편의에 근거해서  $k$ 값을 선택해도 충분하다. (각  $k$ 값을 반복적으로 테스트하는게 어렵기 때문)

### Choosing the value of K

Elbow method:



위 그림에서 왼쪽 그림은 Elbow가 꽤 잘나타나 있다.

오른쪽 그림은 Elbow가 어딘지 잘 모르겠다.

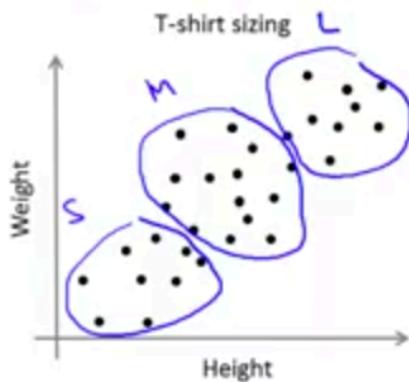
사전지식이 있는 경우

## Choosing the value of K

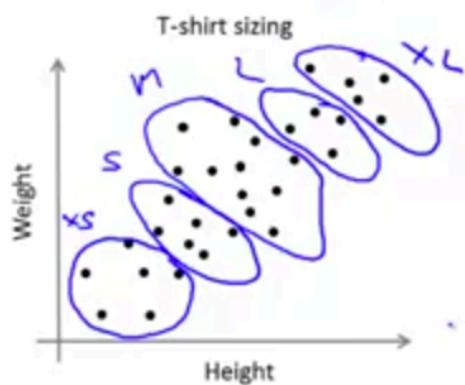
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

$K=3 \quad S, M, L$

E.g.



$K=5 \quad XS, S, M, L, XL$



비즈니스의 관점에서 해석한다.

몇개의 클러스터를 나눠야 비즈니스에 도움이 될까를 생각한다.

ex) 사이즈를 세분화 하는것이 판매에 도움이 될지, 간략화하는것이 판매에 도움이 될지 생각해본다.

## 모델 성능 평가

군집화 결과를 평가하는 것은 다소 주관적일수 있다.

궁극적으로 모델의 성공과 실패는 클러스터가 그들이 의도한 목적에 도움이 되는지에 달려있다.

클러스터 집합의 유용성을 평가하는 가장 기본적인 방법 중 하나는 각 그룹에 속하는 예시 수를 검토해보는 것이다.

그룹이 너무 크거나 너무 작으면 매우 유용하지 않을 가능성이 있다.

## IN R

ex)

```
data <- c(3,4,1,5,7,9,5,4,6,8,4,5,9,8,7,8,6,7,2,1)
row <- c('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
       'J')
col <- c('X', 'Y')

m <- matrix(data,
             nrow=10,
             ncol=2,
             byrow=TRUE,
             dimnames= list(row,col))
km <- kmeans(m, 2)

plot(round(km$center),
     col=km$center,
     pch=22,
     bg=km$center,
     xlim=range(0:10),
```

```

        ylim=range(0:10))
par(new=T)
plot(m, col=km$cluster, xlim=range(0:10),
      ylim=range(0:10))

install.packages('factoextra')
library(factoextra)

fviz_cluster(km, data=data)
  stats library 활용
  library(stats)
    kmeans()
    kmeans( [matrix], [군집할개수(k)] )
  ex)
  m <- matrix(data,
                nrow=10,
                ncol=2,
                byrow=TRUE,
                dimnames= list(row,col))
  km <- kmeans(m, 2)
  center
  중심점을 알수있음
  ex) km$center

  cluster
  각 점이 어떻게 클러스터 되었는지 알수 있음

```

#### IN PYTHON

```

ex)
from sklearn.cluster import KMeans
academy = pd.read_csv('c:/data/academy_utf8.csv')

model = KMeans(n_clusters=4)
model.fit(academy.iloc[:, 2:4])
model.labels_
model.cluster_centers_

import numpy as np

colormatp = np.array(['red', 'blue', 'black',
                     'yellow'])
plt.scatter(academy.iloc[:, 2],
            academy.iloc[:,3],
            c=colormatp[model.labels_],
            s=30)

centers = pd.DataFrame(model.cluster_centers_)
centers.iloc[:,0]
plt.scatter(centers.iloc[:,0],
            centers.iloc[:,1],

```

```

s=50,
marker='D',
c='g')

kMeans()
k-Means 알고리즘을 적용한 모델을 만듦
ex) model = KMeans(n_clusters=4)
    fit()
    모델을 훈련
    model.fit(academy.iloc[:, 2:4])

```

### 상관분석(correlation analysis)

'상관관계' 또는 '상관'은 확률론과 통계학에서 두 변수간에 어떤 선형적 또는 비선형적 관계를 갖고 있는지를 분석하는 방법이다.(분석 기법중 하나)

상관관계분석은 독립변수와 종속변수의 영향관계가 아니라 두 변수 간의 관련성의 정도와 방향을 보여준다.

### 상관관계(correlation)

두 변수간의 관계의 강도

#### 상관계수(correlation coefficient)

두 변수간의 연관된 정도를 나타냄

#### 범위

상관계수의 범위는  $-1 \leq r \leq 1$

피어슨 상관계수(Pearson correlation coefficient,

Pearson's r)

$r = \frac{\text{X와 Y가 함께 변하는 정도}}{\text{X와 Y가 각각 변하는 정도}} = \frac{\text{공분산}(x, y)}{x\text{의 표준편차} * y\text{의 표준편차}}$

$r = \frac{\text{공분산}(x, y)}{x\text{의 표준편차} * y\text{의 표준편차}}$

#### 해석

$r$ 값은 X와 Y가 완전히 동일하면 +1, 전혀 다르면 0, 반대방향으로 완전히 동일하면 -1

일반적으로

$|r| < 0.7$  사이이면, 강한 음적 선형관계,

$|r| < 0.3$  사이이면, 뚜렷한 음적 선형관계,

$|r| < 0.1$  사이이면, 약한 음적 선형관계,

$|r| > 0.1$  사이이면, 거의 무시될 수 있는 선형관계,

$|r| < 0.3$  사이이면, 약한 양적 선형관계,

$|r| < 0.7$  사이이면, 뚜렷한 양적 선형관계,

$|r| > 0.7$  사이이면, 강한 양적 선형관계

로 해석한다.

#### IN PYTHON

corr()

상관계수

ex) corr = df.corr(method = 'pearson')

## 스피어만 상관계수 (Spearman correlation coefficient)

### 겐달 순위 상관계수

#### 우연한 상관관계

현실에서 아무렇게나 고른 두 변수를 조사했을 때 두변수가 전혀 관계없는 경우보다 작게라도 상관관계를 나타내는 경우가 더 흔한다.  
통계학에서는 이를 허위(spurious) 관련성이라고 한다.  
문제는 이런 상관관계를 어떤 인과관계가 있는 것처럼 해석할때 발생한다.

#### 산포도 시각화

ex)

```
boston = pd.read_csv('c:/data/boston.csv')

features = ['CRIM', 'LSTAT', 'RM', 'B']

for i, col in enumerate(features):
    plt.subplot(1, len(features), i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker = 'o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```

## 랜덤 과정(Random Process), 확률 과정(Stochastic Process)

확률론에서 시간의 진행에 대해 확률적인 변화를 가지는 구조

## 회귀분석(regression)

통계학에서 선형 회귀는 종속변수와 한 개 이상의 독립 변수 X와의 선형 상관 관계를 모델링하는 회귀분석 기법이다.

한 개의 설명변수에 기반한 경우에는 단순 선형 회귀, 둘 이상의 설명 변수에 기반한 경우에는 다중 선형 회귀라고 한다.

회귀 방정식은 기울기-절편 형식으로 데이터를 모델링한다.

일반적으로 회귀 분석은 데이터 요소 간에 복잡한 관계를 모델링하고, 처리가 결과에 미치는 영향을 추정하며, 미래의 값을 보간하기 위해 사용된다.

회귀모델을 구축하기 전에 정규성(normality)을 자주 확인해보는 것이 좋다.

선형 회귀는 종속 변수가 정규 분포를 따르도록 엄격하게 요구하진 않지만, 그럴 경우 모델이 좀 더 잘 적합된다.

회귀 모델과 다른 머신러닝 방식의 주요 차이점은 회귀는 전형적으로 사용자가 특징을 선택하고 모델을 명시한다는 것이다.

#### 가정

종속 변수는 연속 범위에서 측정된다고 가정한다.

## 보통 최소 제곱법(Ordinary Least Squares)

회귀분석에서 기울기와 절편을 추정하기 위한 방법

OLS 회귀에서 기울기와 절편은 오차 제곱 합(sum of the squared errors)이 최소화 되게 선택되며, 오차는 y의 예측값과 실제 값사이의 수직거리다.

이 오차를 잔차(residuals)라 한다.

관측치와 회귀선과의 거리인 잔차의 제곱의 합을 최소화 하는 직선을 찾는 방법

$$\sum (x - \bar{x}) * (y - \bar{y})$$

$$\text{기울기} = \frac{\sum (x - \bar{x})^2}{\sum (x - \bar{x})^2}$$

$$\text{절편} = \bar{y} - (\bar{x} * \text{기울기})$$

기울기 B에 대한 설명

독립 변수가 한 단위 증가 했을때 B만큼 y가 변한다.

### 단순회귀분석(simple linear regression)

독립변수가 1개인것

#### 회귀계수

$$a = cov(\text{height}, \text{weight}) / \text{var}(\text{height})$$

#### 절편

$$b = \text{mean}(\text{weight}) - a * \text{mean}(\text{height})$$

#### IN R

`lm(목표변수 ~ 설명변수, data = [dataframe])`

이를 호출하면 회귀계수와 절편이 도출됨

ex) `m = lm(weight~height, data = df)`

#### IN PYTHON

##### scipy의 stats 라이브러리

```
from scipy import stats
stats.linregress(설명변수, 목표변수)
ex)
from scipy import stats
height = [176, 172, 182, 160, 163, 165, 168, 163,
          182, 182]
weight = [72, 70, 70, 43, 48, 54, 51, 52, 73, 88]
slope, intercept, r_value, p_value, stderr =
    stats.linregress(height, weight)
```

### 다중회귀분석(multiple linear regression)

독립변수가 2개이상인것

#### 단점

데이터에 대한 강한 가정을 한다.

모델 형태가 사용자에 의해 미리 지정돼야만 한다.

누락 데이터를 처리하지 않는다.

수치 특징만 처리하므로, 범주 데이터는 추가 처리를 해야 한다.

모델을 이해하려면 약간의 통계 지식이 필요하다.

#### IN R

`lm(목표변수 ~ 설명변수들)`

이를 호출하면 회귀계수와 절편이 도출됨

ex) `ins_model <- lm(charges ~ age + children + bmi + gender + smoker + region, data = insurance)`

IN PYTHON

`sklearn`

`from sklearn import linear_model`

`model 생성`

`LinearRegression()`

ex) `reg = linear_model.LinearRegression()`

`데이터 넣기`

`reg.fit([DATAFRAME형태의 설명변수들], 목표변수)`

ex) `reg.fit(score.iloc[:,2:], score['성적'])`

`절편`

`[LinearRegression model].intercept_`

`기울기`

`[LinearRegression model].reg.coef_`

`시각화`

`ex)`

`model =`

`LinearRegression().fit(boston[['CRIM']], boston['MEDV'])`

`plt.scatter(boston['CRIM'], boston['MEDV'])`

`plt.plot(np.arange(min(boston['CRIM']), max(boston['CRIM'])), model.intercept_ + np.arange(min(boston['CRIM']), max(boston['CRIM']))) * model.coef_[0])`

`분석을 위해`

1. 산점도를 그려보는게 좋다.

2. 모델의 선을 그려본다(최소제곱법을 이용해서 선을 그린다.)

$y = ax + b$  ( $y$ : 종속변수,  $x$ : 독립변수,  $a$ : 회귀계수(기울기),  $b$ : 절편  
(시작점)

IN R

R을 이용해서 선형 회귀 모델을 데이터에 적합시키려면 `lm()` 함수를 사용할 수 있다.

`lm()` 함수는 팩터 타입 변수에 더미코딩 기법을 적용한다.(명목특징을 수치로 취급하기 위해 특징 범주별로 더미변수라고 하는 이진 변수 생성)

더미 변수는 관측이 명시된 범주에 속하면 1로 그렇지 않으면 0으로 설정한다.

회귀모델에 더미 변수를 추가할 때 범주중 하나는 참조 변수(reference category)로서 사용하기 위해 남겨둔다. - 디폴트로 팩터 변수의 첫번째 레벨을 참조로 사용

### 모델 성능 평가

summary() 를 사용하면 모델 성능을 평가할수 있다.

ex) summary(ins\_model)

#### Adjusted R-squared

모델이 데이터를 얼마만큼 설명하는지 표현

ex) Adjusted R-squared: 0.8069 <-> 80%

### p-value

모델의 유의성을 확인하는 방법

알파값이 0.05라면, p-value가 0.05보다 작으면 귀무가설이 기각됨

ex) p-value: 0.0002555 <->

## 인과관계 (causality)

어떤 변수가 어떤 변수에게 어떤 영향을 주는지를 판단한다.

변수들간의 인과관계는 독립변수가 종속변수에 어떠한 영향을 미치는지 파악하는 것이다.

인과관계는 시간과 관련이 있다. (사건 A가 있을때 그다음 사건 B가 일어나야 두 사건 간에 인과관계가 있고 할수 있다.)

외생변수를 통제해야 한다. (다른 요인을 통제하고 인과관계를 분석

### 상관관계와의 관계

상관관계는 인과관계를 포함할수 있다.

인과관계가 성립하기 위해서는 직접적인 관계가 있어야 한다. (특수한 조건을 만족하는 상관관계가 인과관계임)

### 밀이 제시한 인과관계 성립조건

1. 원인이 결과보다 시간적으로 앞서야 한다.
2. 원인과 결과는 관련이 있어야 한다.
3. 결과는 원인이 되는 변수만으로 설명이 돼야 하고 다른 변수에 의한 설명은 제거돼야 한다.

그러나 이런 조건들이 만족됐다 하더라도 그것은 인과관계를 추론하는 데 합리적 근거가 될 수는 있지만 인과관계의 존재가 입증됐다고 할 수는 없다.

### 전후인과의 오류 (post hoc fallacy)

A가 일어난 다음 B가 일어났다고 해서 A가 B의 원인이라고 결론짓는 것은 명백한 오류이며 이를 전후인과의 오류라 한다.

시간적 발생에 따라 인과를 해석하려는 오류를 경계해야한다.

## 로지스틱 회귀 (logistic regression)

영국의 통계학자인 D.R. Cox가 1958년에 제안한 확률 모델로서 독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는데 사용되는 통계 기법이다.

우리가 예측하려는 y값을 A일 확률이라고 하고, y값(A일 확률)이 0.5보다 크면 A로 분류하고 0.5보다 작으면 B로 분류하자고 생각한것이다.

일반적으로 로지스틱 회귀분석은 이항 로지스틱 회귀분석을 의미한다.

Logistic이란 말이 붙은 이유는 이 회귀분석이 독립변수와 종속변수의 관계를 S자형으로 가정하기 때문이다.

직선형의 함수가 아닌 곡선형으로 나타나는 함수중에 지수함수가 있는데, 이를 Logistic 곡선이라고 부르기도 한다.

곡선형을 가정하는 이유는 종속변수가 다른 형태이기 때문이다.

단순회귀분석과 다중회귀분석에서 종속변수는 척도(등간, 비율) 변수 였다.

하지만 현실에서 종속변수를 명목 척도로 측정하는 경우도 흔히 있다.

이 로지스틱 회귀분석을 가장 쉽게 이해할 수 있는 예는 소비자의 성향과 경제력을 독립 변인으로 해서 종속변인을 물건을 구입하지 않음, 구입함이라고 보는 두가지다. –  
구입함: 1, 구입하지않음: 0

이렇게 두 개의 항으로 이루어진 변수를 이분형 변수(binary variable)이라고 한다.

그렇다면 왜 이분형 변수인 경우에 로지스틱 회귀분석을 사용하는가?

그 이유는 이분형 변수인 경우에는 결과값이 정규분포를 따르는게 아니라 이항분포를 따른다고 보는 것이다.

여기서 이항 분포란 마치 동전던지기와 같이 이것 아니면 저것이라는 이항 실험을 통해서 나타나는 분포이다.

### 목적

로지스틱 회귀의 목적은 일반적인 회귀분석의 목표와 동일하게 종속변수와 독립 변수간의 관계를 구체적인 함수로 나타내어 향후 예측 모델에 활용하는 것이다.  
이는 독립 변수의 선형 결합으로 종속 변수를 설명한다는 관점에서는 선형 회귀 분석과 유사하다.

하지만 로지스틱 회귀는 선형 회귀 분석과는 다르게 종속 변수가 범주형 데이터를 대상으로 하며 입력 데이터가 주어졌을 때 해당 데이터의 결과가 특정 분류로 나누기 때문에 일종의 분류(classification) 기법으로도 볼 수 있다.

### 수식

$$h_{\theta}(x) = g(\theta^T x)$$

여기서  $g$ 는 logistic function

### 모델의 해석

$h_{\theta}(x) = x$ 가 입력되었을 때  $y = 1$ 일 가능성(estimated probability)

$h_{\theta}(x) = P(y = 1 | x; \theta)$

$x$ 가 주어졌을 때  $y$ 가 1일 가능성의 측정값

주어진  $x$ 에 대해,  $\theta$ 에 매개된 가능성(probability that  $y = 1$ , given  $x$ , parameterized by  $\theta$ )

$$P(y = 0 | x; \theta) + P(y = 1 | x; \theta) = 1$$

$P(y = 0 | x; \theta) = 1 - P(y = 1 | x; \theta)$  –  $x$ 에 대한 가설의 값  
( $y=1$ )이 주어진다면  $y=0$ 일 가능성을 구할 수 있다.

ex)

악성 종양 데이터 –

$$x = [x_0 \ x_1] = [1 \ tumorSize]$$

$h_{\theta}(x) = 0.7$  이라고 하면

특정  $x$ 를 가진 사례에서  $y$ 가 1일 확률은 0.7

이는 70% 확률로 악성이라는 의미

$$h_{\theta}(x) = P(y = 1 | x; \theta)$$

### (logistic function)

성공확률  $p$ 를 갖는 사건 A를 바탕으로 해석해보면 Logistic function은 성공확률  $p$  그 자체이다.

$$\frac{p}{1-p} = e^{\log Odds(A)},$$

$$p = e^{\log Odds(A)} - e^{\log Odds(A)} p,$$

$$(1 + e^{\log Odds(A)})p = e^{\log Odds(A)},$$

$$\begin{aligned}\therefore p &= \frac{e^{\log Odds(A)}}{1+e^{\log Odds(A)}} \\ &= \frac{1}{1+e^{-\log Odds(A)}}\end{aligned}$$

(**logistic function, sigmoid function**) 유도

우리는 분류시, 새로운 데이터를 분류할 때 사후 확률을 기준으로 삼는다.

$$X : Y_1 \text{ if } P(Y_1|X) > P(Y_2|X)$$

$$X : Y_2 \text{ if } P(Y_1|X) < P(Y_2|X)$$

위와 같이 두 개의 Class가 있을 때 판별 기준인 Posterior를 전개해보자

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X)}$$

$$P(Y_2|X) = \frac{P(X|Y_2)P(Y_2)}{P(X)}$$

그런데 여기서 분모는

$$P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)$$

이 되므로 두 식에서 모두 같다.

즉, 사후 확률은 Likelihood와 Prior의 곱이 결정한다는 것이다.

즉 각 Class 별로 Likelihood와 Prior의 곱을 정의하는 것은 의미가 있다.

여기서  $a_k$ 를 Likelihood와 Prior의 곱으로 정의하자. ( $k$  클래스에 대한 likelihood와 prior의 곱)

$$a_k = \ln(P(X|Y_k)P(Y_k))$$

로그를 붙이는 이유는 Log likelihood를 본다면 알 수 있다.

- Log는 단조 증가 함수로서 극점을 변화시키지 않는다.

- 각 시행이 독립일 경우 곱이 덧셈으로 바뀌어 계산이 용이해진다.

- 확률 분포가 Gaussian 등의 Exponential Family일 경우 계산이 용이해진다.

그러면 Class 1의 사후확률은 다음과 같다.

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)} = \frac{e^{a_1}}{e^{a_1} + e}$$

분모와 분자를 모두 분자로 나누면

$$P(Y_1|X) = \frac{1}{1 + e^{a_2 - a_1}}$$

이 되고,

$$a_2 - a_1 = \ln\left(\frac{P(X|Y_2)P(Y_2)}{P(X|Y_1)P(Y_1)}\right)$$

이고, 분모 분자를 뒤집어 새로 하나를 정의하면 (이것을 Log Odds라 부른다)

$$a = -(a_2 - a_1) = \ln\left(\frac{P(X|Y_1)P(Y_1)}{P(X|Y_2)P(Y_2)}\right)$$

그러면 다시 사후확률로 돌아가서

$$P(Y_1|X) = \frac{1}{1 + e^{-a}}$$

가 된다.

a에 대한 함수로서 이는 Sigmoid Function이다.(정식명칭 Logistic Function)

그런데 a는 위의 정의에 따라

$$\begin{aligned} a &= \log \frac{P(X|Y_1)P(Y_1)}{P(X|Y_2)P(Y_2)} \\ &= \log \frac{P(Y_1|X)}{P(Y_2|X)} \\ &= \log \frac{P(Y_1|X)}{1 - P(Y_1|X)} \\ &= \log Odds(Y_1|X) \end{aligned}$$

즉, 데이터 X를 바탕으로 Y1이 옳은 클래스라고 판단할 log odds이다.(logit)

그런데 이 a에 logit의 역함수인 logistic을 적용하면  
 $P(Y_1|X)$ 만 남는다.

결론적으로

logistic(  $Y_1 | X$ )는 사건  $Y_1 | X$ 의 확률, 즉 내재적으로  $P(Y_1 | X)$  그 자체이다.

이때문에 우리는 확률분포에 대한 가정 없이도 deterministic하게 logistic function을 각 클래스로 판단할 확률값으로 곧바로 사용할 수 있게 된다.

만약 우리가 Likelihood와 Prior의 확률 분포를 구분하지 않고 다음 함수에 Fitting하는 문제로 바꾼다면 이것은 대표적인 Discritive 문제인 Logistic Regression이 된다.

$$P(Y|X) = \frac{1}{1 + e^{-\theta x}}$$

### 선형 회귀의 문제점

<https://m.blog.naver.com/lyshyn/221285013102>

선형 식의  $x$ 값은 무한대 ~ 무한대의 범위를 갖는다.

그러나 우리는 0~1 사이의 확률값만을 얻기 원한다.

그런데 기존의 선형식은 직선이므로 한쪽으로 계속 늘어나거나 줄어들수 있다.

그래서 기존 가설의 결과를 0과 1범위로 압축시키는 방법이 필요하다.

#### 문제를 해결해나가는 과정

입력값의 범위가 0~1이며 출력값의 범위를 -무한대~무한대를 갖는 함수를 만든다.

이에 대한 역함수를 취하면

입력값의 범위가 -무한대 ~ 무한대이고 출력값이 0~1인 함수를 만들수 있다.

#### (odds)

0부터 1까지로 값이 제한된 확률 값의 범위를 0부터 무한대로 확장하기위해 odds를 취한다.

odds는 쉽게 말해 실패 확률에 대한 성공확률의 비율이다.

성공확률을  $p$ 라고 한다면 실패확률은  $1-p$ 가 된다.

발생 확률(또는 성공확률)이  $p$ 인 어떤 사건 A의 Odds는 다음과 같이 나타낼수 있다.

$$Odds(A) = \frac{p}{1-p}$$

$p$ 가 0이면 odds는 0이되고  $p$ 가 1에 가까워지면 odds는 무한대가 된다.

즉, 0~무한대의 범위를 갖는다.

그런데 아직 -무한대의 범위를 갖지 못하기 때문에 만족스럽지 않다.

#### 사용예

odds를 통해 분류를 할수도 있다.

0.5보다 크면 성공확률이 더 크다는 얘기고

0.5보다 작으면 실패확률이 더 작다는 얘기다.

따라서 0.5를 기준으로 성공 or 실패를 결정할수 있다.

#### (log odds)

Odds에 (자연)로그를 취한것이다.

로그 변환을 한것이다.

$$\log Odds(A) = \log \frac{p}{1-p} = \log p - \log(1-p)$$

로그 변환은 통계학에서 자주 사용하는 변환으로,

- 함수의 증감 형태, convex/concave 형태를 유지시키고
- 극점의 위치를 유지시키며
- 곱(또는 나눗셈)으로 표현된 식을 선형조합의 꼴로 풀어쓸수 있도록 해준다.

### logit function

다음과 같은 함수꼴을 말한다.

$$logit(x) = \log \frac{x}{1-x}$$

logistic function은 logit 함수의 역함수이다.

### log

음의 무한대를 범위에 포함시키기 위해 log를 취한다.

log 중에서도 자연상수 e를 밑으로 하는 자연로그를 취한다.

## ln(odds)

odds는 0~무한대까지의 범위를 취하고 이에 대해 로그를 취했으므로 다음과 범위를 생각해볼수있다.

$e^x$  이 0인 x를 찾는다. => -무한대

$e^x$  이 무한대인 x를 찾는다. => 무한대

따라서 출력값이 무한대~무한대를 커버할수 있게 되었다.

결국 odds를 구하고 거기에 자연로그를 취했더니 확률 p에 대해 -무한대 ~ 무한대를 범위로 갖는 함수가 만들어졌다. (이 과정을 logit 변환이라고도 한다)

그러면 이제 다음과 같이 표현할수 있다. (로지스틱 회귀에서 로짓변환의 결과는 x에 대한 선형함수와 동일)

여기서 W와 X의 곱은 선형회귀의 가설을 의미

$$\ln \frac{p}{1-p} = W \cdot X$$

하지만 우리의 최종목적은 이 함수를 통해 확률 p를 찾는것이므로 이 함수를 p에 대한 함수로 만든다.

$$\frac{p}{1-p} = e^{W \cdot X}$$

1. 양변에 역수를 취한다.

$$\frac{1-p}{p} = \frac{1}{p} - 1 = \frac{1}{e^{W \cdot X}}$$

$$\frac{1}{p} = \frac{1}{e^{W \cdot X}} + 1 = \frac{1}{e^{W \cdot X}}.$$

$$= \frac{1+e^{W \cdot X}}{e^{W \cdot X}}$$

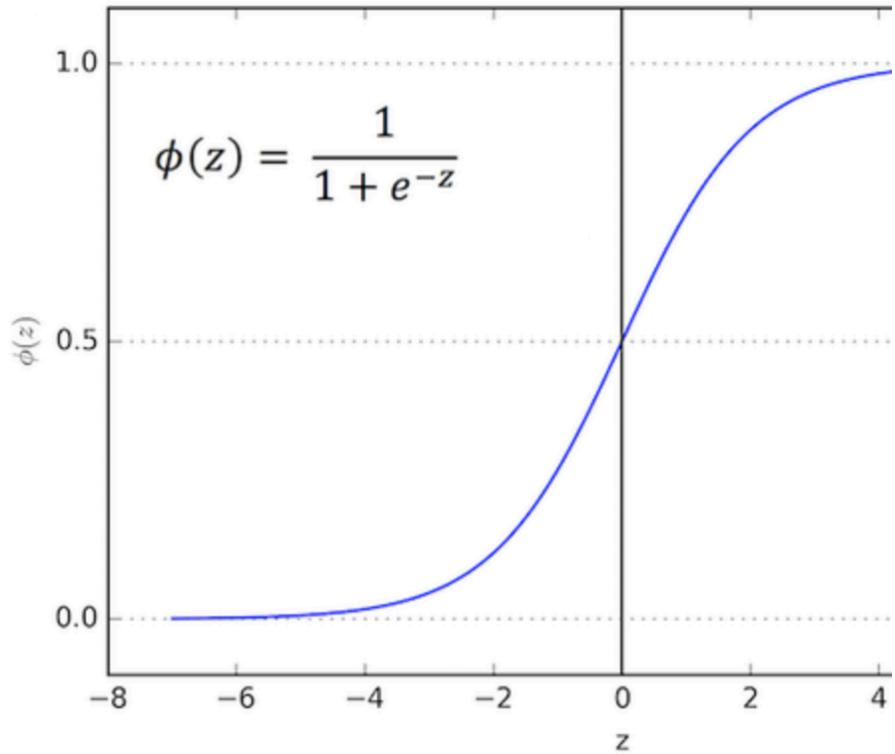
2. 한번 더 역수를 취한다.

$$p = \frac{e^{W \cdot X}}{1 + e^{W \cdot X}}$$

3. p에 관한 식으로 정리한다.

$$p = \frac{e^{W \cdot X}}{1+e^{W \cdot X}} \cdot \frac{\frac{1}{e^{W \cdot X}}}{\frac{1}{e^{W \cdot X}}} = \frac{1}{\frac{1}{e^{W \cdot X}}+1} =$$

이 식을 그래프로 그렸을 때 나오는 곡선이 로지스틱 곡선이다.



### 정리

무한의 값을 갖는 선형 회귀식을 통해 두 개의 값(0 또는 1)만을 갖는 결과를 가져야 한다.

두 개의 결과만을 도출하는 과정이 연속되면 확률이 되며 확률 또한 범위는 0부터 1까지로 제한된다.

0부터 1까지로 값이 제한된 확률 값의 범위를 0부터 무한대로 확장하기 위해 odds를 취한다.

이를 다시 음의 무한대로 확장하기 위해 자연로그를 취한다.

이 등식을 변형하면 로지스틱 함수가 도출된다.

결국! 선형회귀의 값을 넣어서 이에 대한 분류의 probability를 구하는게 로지스틱함수

$$\begin{aligned}
 \Pr(Y_i = 1 \mid \mathbf{X}_i) &= \Pr(Y_i^* > 0 \mid \mathbf{X}_i) \\
 &= \Pr(\boldsymbol{\beta} \cdot \mathbf{X}_i + \varepsilon > 0) \\
 &= \Pr(\varepsilon > -\boldsymbol{\beta} \cdot \mathbf{X}_i) \\
 &= \Pr(\varepsilon < \boldsymbol{\beta} \cdot \mathbf{X}_i) \\
 &= \text{logit}^{-1}(\boldsymbol{\beta} \cdot \mathbf{X}_i) \\
 &= p_i
 \end{aligned}$$

### Sigmoid의 문제점

기존 선형 회귀 그래프의 Cost function은 Convex function 형태로 그려졌다.

그러나 시그모이드 함수는 Non-Convex function 형태로 그려진다.

Non-Convex function은 Local Minimum이 존재할 수 있다. (즉, 컴퓨터가 최적의 cost라고 착각할만한 구간(Local Minimum)이 있다

는 이야기다.

시그모이드 함수의 그래프가 Non-Covex function이 된 이유는 이 함수에 지수함수(exponential function)이 들어갔기 때문이다.  
이를 원래대로 돌리기 위해 손실함수에 Log를 적용한다.

### 손실함수

$$C(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

이러한 손실함수를 사용하는 이유는 다음과 같다.

로지스틱 함수의 결과는 확률로 표현하면?  $-P(Y = 1 | x = x)$  즉,  $x$ 가 주어졌을 때  $Y$ 가 1일 조건부 확률

그러므로  $y$ 가 1일때(정답 레이블이 1일때) 로지스틱함수의 결과가 1에 가까우면 좋은것임

그런데,  $y$ 가 1일때 로지스틱 함수의 결과가 0에 가까우면 많은 패널티를 먹여야 하므로 로그를 취하고 음의 값을 양의 값으로 바꾸기 위해  $-$ 를 계수로 둔다.

반대로  $y$ 가 0일때 로지스틱함수의 결과가 0에 가까우면 좋은것이다.

그런데 1에 가까우면 패널티를 먹어야 하므로  $1 - H(x)$ 에 로그를 취한다.

$y=1$ 일때의 식의 경우  $H(x)$ 값이 0일때 값이 무한대로 발산한다. 그런데, 1일때 값은 0으로 수렴한다.

$y=0$ 일때의 식의 경우  $H(x)$ 값이 1일때 값이 발산한다. 반면 0일때 수렴한다.

즉, Cost 함수의 경우 잘된 예측은 Cost의 값이 작아야하고 잘안된 예측은 값이 커야 한다.

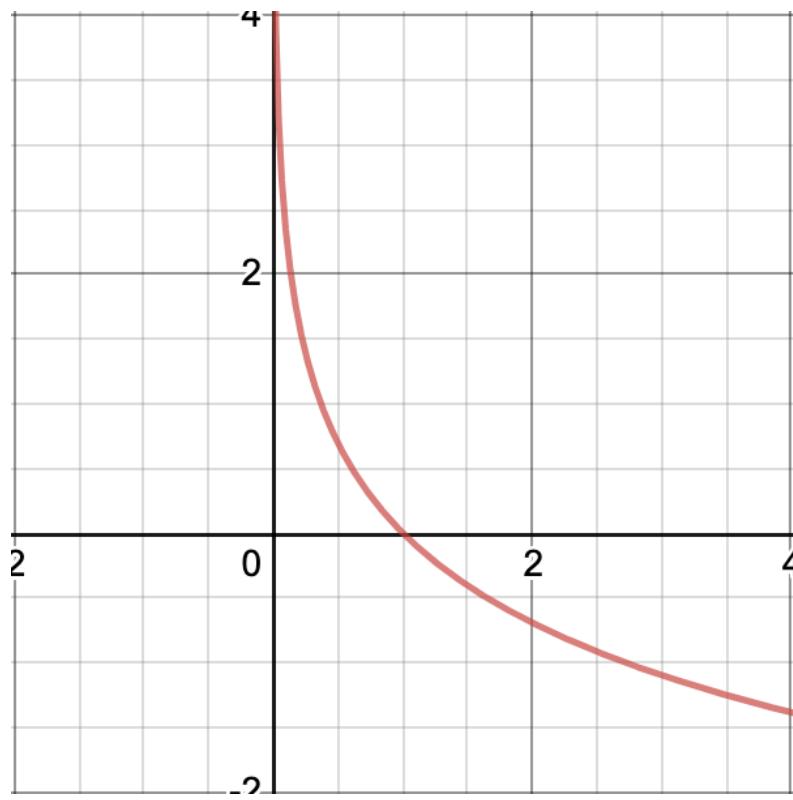
따라서 위와 같은 식을 활용하는 것이다.

### 수식 종합

$$\text{Cost}(H(x), y) = -y * \log(H(x)) - (1 - y) * \log(1 - H(x))$$

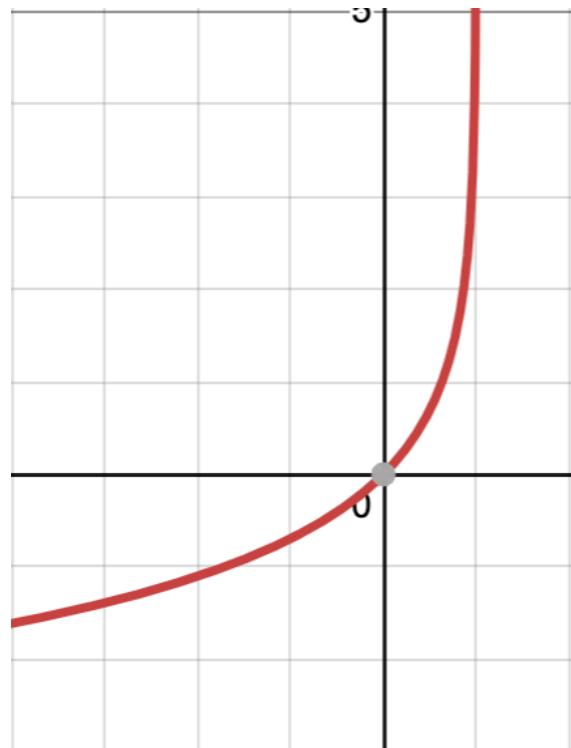
### 로그함수 그래프(참조)

[-ln\(x\) <- 로그함수를 x축 대칭한 그래프](#)



- $\ln(1-x)$  <- 로그함수를 x축 대칭하고 x축으로 1만큼 이동한 그래

프



cost function

$$\begin{aligned}
J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\
&= -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right]
\end{aligned}$$

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

**vectorized implementation**

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} \cdot \left( -y^T \log(h) - (1 - y)^T \log(1 - h) \right)$$

### Gradient Descent

*Repeat {*

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

*}*

*Repeat {*

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

*}*

**vectorized implementation**

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

### 다항 로지스틱 회귀분석

이항로지스틱 회귀분석 말고 종속변수의 형태가 여러 개인 경우 다항로지스틱 회귀 분석을 사용할 수 있다.

**ex)**

대통령 선거를 할 때, 후보자가 여러명이 있을 것이다.

이 여러 명의 후보자에 대한 우리의 선택 여부는 선택 (1), 선택하지 않음 (0)으로 부호화될 수 있다.

이 여러명의 후보자를 동시에 비교할 때, 다항 로지스틱 회귀분석을 적용할 수 있다.

## 사용

흔히 로지스틱 회귀는 종속변수가 이항형 문제를 지칭할 때 사용된다.  
이외에, 두 개 이상의 범주를 가지는 문제가 대상인 경우엔 다항 로지스틱회귀 또는 분화 로지스틱 회귀라고 하고 복수의 범주이면서 순서가 존재하면 서수 로지스틱 회귀(ordinal logistic regression)라고 한다.  
로지스틱 회귀 분석은 의료, 통신, 데이터마이닝과 같은 다양한 분야에서 분류 및 예측을 위한 모델로서 폭넓게 사용되고 있다.

## 기초

로지스틱 회귀는 이항형 또는 다항형이 될 수 있다.  
로지스틱 회귀는 일반적인 선형 모델(generalized linear model)의 특수한 경우로 볼수 있으므로 선형 회귀와 유사하다.  
하지만, 로지스틱 회귀의 모델은 종속 변수와 독립 변수 사이의 관계에 있어서 선형 모델과 차이점을 지니고 있다.  
첫번째 차이점은 이항형인 데이터에 적용하였을 때 종속 변수  $y$ 의 결과가 범위  $[0, 1]$ 로 제한된다는 것이고  
두번째 차이점은 종속변수가 이진적이기 때문에 조건부 확률의 분포가 정규분포 대신 이항 분포를 따른다는 점이다.

따라서 대상이 되는 데이터의 종속변수  $y$ 의 결과는 0과 1, 두 개의 경우만 존재하는 데 반해, 단순 선형 회귀를 적용하면 범위  $[0, 1]$ 를 벗어나는 결과가 나오기 때문에 오히려 예측의 정확도만 떨어뜨리게 된다.

이를 해결하기 위해 로지스틱 회귀는 연속이고 증가함수이며  $[0, 1]$ 에서 값을 갖는 연결 함수  $g(x)$ 를 제안하였다.  
연결 함수의 형태는 다양하게 존재하는데 그 중 대표적인 두 개는 아래와 같다.  
밑의 모형중에 계산의 편리성으로 인하여 로지스틱 모형이 널리 사용된다.

### 로지스틱 모형

$$g(x) = e^x / (1 + e^x)$$

### 검별 모형

$$g(x) = e^{(-e^x)}$$

### 결정 경계(Decision Boundary)

$h_{\theta}(x) = g(\theta'x)$  에서  
 $g$ 는 logitic function 인데, 이때  $g$ 가 0.5 가 됨을 기준으로 하여  $y=1$  혹은  $y=0$  이 나뉜다.

그런데  $g$ 가 0.5가 되는 값은  $\theta'x = 0$ 이 되는  $x$ 값이다.

따라서  $\theta'x$  가  $\geq 0$  이면,  $y=1$ 이되고,

0보다 작으면  $y=0$ 이 된다.

이때  $\theta'x = 0$  인 수식을 decision boundary라고 한다. ( $h_{\theta}(x) = x$ 가 입력되었을 때  $y = 1$ 일 가능성(estimated probability)이 0.5가 되는 line)

결정 경계는 training set 이 아닌 hypothesis의 property이다.

### 다중 분류(Multiclass Classification), one-vs-all, one-vs-rest

다중 분류 문제를 해결하는 방법은 다음과 같다.

한 클래스를 양(+)으로 놓고 나머지 클래스를 음(-)으로 놓은 다음

한 클래스를 분류할수 있는  $h_{\theta^i}(x)$  를 찾는다. - 즉, 다중 분류문제를 이진 분류 문제로 바꾼다.

이를 모든 클래스에 대해 수행한다.

만약 k개의 클래스가 존재한다면  $h_{\theta^i}(x)$  는 k개가 존재하게 된다.

각 가설의 의미는 다음과 같다.

$h_{\theta^i}(x) = P(y = 1 | x; \theta) \quad (i=1, 2, 3, \dots, k)$

즉, 각 클래스 i에 대해 y가 1인 확률을 예측한것이다.

이 가설들중 가장 큰 값을 선택하게 되면, 가장 확률적으로 가능성이 큰 class 를 선택할수 있다.

이를 요약하면 다음과 같다.

$\max h_{\theta^i}(x)$

### Logistic Regression에서의 정규화(regularized)

로지스틱 회귀에서, 많은 feature가 있는 경우 over fitting 하는 경향이 있다.

여기 로지스틱 회귀에서의 cost function이 있다.

$$J(\theta) = \frac{1}{m} \sum [ -y \log(h_{\theta}(x)) - (1-y) \log(1 - h_{\theta}(x)) ] + \frac{\lambda}{2m} \sum (\theta_j)^2 \quad \leftarrow \theta_1 \dots \theta_n$$

여기에 다음 항을 더한다.

이는  $\theta$ 가 커지면 벌금이 부과됨을 의미한다.

### Gradient Descent

Repeat {

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha(1/m) * \sum (h_{\theta}(x^{(i)}) - y^{(i)}) x_{\theta^0}(i) \\ \theta_j &:= \theta_j - \alpha [ (1/m) * \sum (h_{\theta}(x^{(i)}) - y^{(i)}) x_{\theta^j}(i) + (\lambda/m) \theta_j ] \end{aligned}$$

}

$\theta_j$ 를 업데이트 하는식에서  $\alpha$  뒤에 붙는 절은 새로운 (정규화한) cost function의 편미분 항이다.

### IN PYTHON

```
sklearn.linear_model Library
from sklearn.linear_model import LogisticRegression
ex)
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

iris = pd.read_csv('c:/data/iris.csv')

X = iris.iloc[:, :-1]
Y = iris['Name']

log = LogisticRegression()
log.fit(X, Y)
```

```
test1 = [[5.1, 3.5, 1.4, 0.2]]  
log.predict(test1)
```

### 분류 규칙 학습 알고리즘

분류 규칙 학습 알고리즘은 분리 정복(seperate and conquer)으로 알려진 휴리스틱을 활용한다.

이 과정에서 훈련 데이터에서 예시의 부분집합을 커버하는 규칙을 식별하고 이 부분집합을 나머지 데이터와 분리한다.

규칙이 추가되면서 데이터의 부분집합도 추가적으로 분리되고 전체 데이터셋이 다 커버되고 더 이상의 예시가 남아있지 않을 때까지 진행된다.

규칙 학습자는 동질적인 그룹을 발견하기 위해 가용한 특징을 이용한다.

규칙이 데이터의 부분을 커버하는 것 같이 보이기 때문에 분리정복 알고리즘은 커버링(covering algorithms)이라고 하며, 만들어진 규칙은 커버링 규칙이라고 한다.

#### 사용분야

규칙 학습자는 의사결정 트리처럼 미래의 행동을 위한 지식을 만들어내는 응용에 사용될수 있다.

규칙 학습자는 일반적으로 특징이 주로 또는 전체적으로 명목형인 문제에 적용된다.

### 분류 규칙(classification rules)

클래스를 레이블이 없는 예시에 할당하는 논리적인 if-else문 형태로 지식을 표현한다.

분류 규칙은 조건부(antecedent)와 결론부(consequent)에 대해 명시된다. 즉, "이것이 발생한다면 저것이 발생한다"라는 가설을 구성한다.

조건부는 특정 특징 값들의 조합으로 이뤄지지만, 결론부는 규칙의 조건을 만족할 때 배정할 클래스 값을 지정한다.

### 의사결정 트리와의 비교

위에서 아래 방향으로 일련의 결정을 통해 적용되는 트리와 달리 규칙은 사실을 서술한 것처럼 읽을 수 있는 명제다.

### 장점

희소한 사건(rare events)이 특징 갯 사이에 매우 특정한 상호작용에서 발생한다 하더라도 규칙 학습자는 희소 사건을 잘 식별한다.

### oneR 알고리즘

하나의 사실만 가지고 간단하게 분류하는 알고리즘

#### IN R

```
OneR library  
OneR()  
OneR( 목표변수 ~ 설명변수들, data = [dataset] )  
ex)  
model <- OneR(type ~ ., data=mushroom_train)  
result <- predict(model, mushroom_test[, -1])
```

### Riper 알고리즘

복수개의 사실(조건)을 가지고 분류하는 알고리즘

### IN R

```
RWeka library <- 이거 데이터 프레임에 헤더 있으면 에러난다고  
함.. 따라서 헤더 없이 줘야함  
JRip()  
ex)  
model <- JRip(V1~., data=mushroom_train)  
result <- predict(model, mushroom_test[, -1])
```

### IN R

### C50 library

C5.0()에서 rules 파라미터를 TRUE로 주면 규칙기반으로 분류함

### 드릴 다운(drill down)

드릴 다운(Drill down)은 가장 요약된 레벨로부터 가장 상세한 레벨까지 차원의 계층에 따라 분석에 필요한 요약 수준을 바꿀 수 있는 기능이다. 고위관리자 뿐만 아니라 데이터 분석이 필요한 하위직원에게도 유용하다. [1]

### 부트스트랩(bootstrap sample)

여기서 bootstrap 자료란 단순 복원 임의 추출법(random sampling)을 통해 원자료(raw data)로부터 크기가 동일한 여러 개의 표본 자료를 말한다.

### 앙상블(ensembles)

#### IDEA

약한 학습자 여러 개를 결합하면 강한 학습자가 만들어진다.

#### 앙상블 방법에 대한 구분

약한 학습 모델이 어떻게 선택되고 구성되는가?

하나의 최종 예측을 만들기 위해 약한 학습자의 예측이 어떻게 결합되는가?

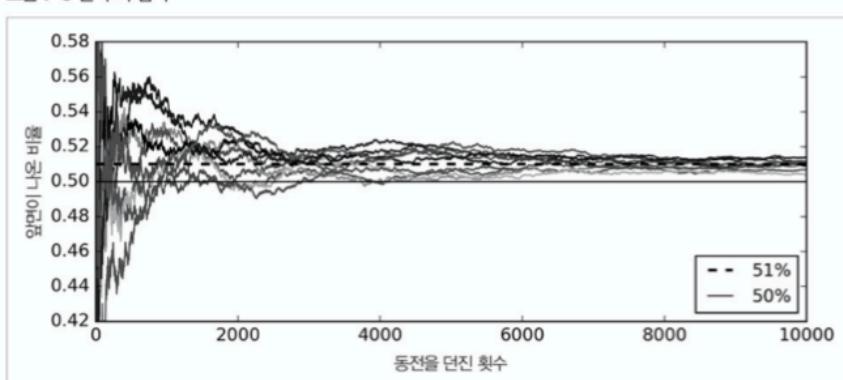
#### 큰 수의 법칙(Law of large numbers)

큰 모집단에서 무작위로 뽑은 표본의 평균이 전체 모집단의 평균과 가까울 가능성이 높다.

ex)

앞면이 51% 뒷면이 49% 확률로 나오는 동전을 가정한 경우

그림 7-3 큰 수의 법칙



여기서 앞면이 다수일 경우를 생각해보면,

1000번을 던진 후 앞면이 다수일 확률은 75%에 가까워진다.

더 많이 던질수록 확률은 증가한다.

(이는 이항분포의 누적분포함수로 구할수 있다)

## 양상불과 큰수의 법칙과의 연관성

51% 정확도를 가진 1000개의 분류기로 양상을 모델을 구축한다고 가정하자  
이때 가장 많은 클래스를 예측으로 잡는다면 75%의 정확도를 기대할 수 있다.  
즉, 모델을 많이 돌리면 정확도가 높아질 수 있다.

### 가정

하지만, 모든 분류기가 완벽하게 독립적이고 오차에 상관관계가 없어야 가능하다.

그래서 양상을 방법은 예측기가 가능한 한 서로 독립적일 때 최고의 성능을 발휘 한다.

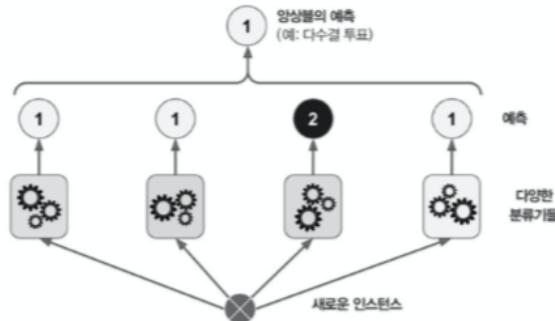
다양한 분류기를 얻는 한 가지 방법은 각기 다른 알고리즘으로 학습시키는 것이다.

이렇게 하면 매우 다른 종류의 오차를 만들 가능성이 높기 때문에 양상을 모델의 정확도를 향상시킨다.

## 투표기반분류기

### 직접 투표(hard voting)

각 분류기의 예측을 모아 다수결 투표로 예측을 정하는 것을 의미한다.



### 간접 투표(soft voting)

개별 분류기의 예측을 평균 내어 확률이 가장 높은 클래스를 선택

## 메타 학습(meta learning)

학습 방법을 학습하는 기법이다.

## 배깅(bagging, bootstrap aggregating)

다양한 분류기를 만드는 한 가지 방법은 각기 다른 훈련 데이터를 사용하는 것이다.  
배깅은 훈련 세트에서 중복을 허용하여 샘플링하는 방식을 의미한다.

bootstrap aggregating의 준말로서 주어진 데이터에 대해서 여러 개의 bootstrap 자료를 생성하고 각 bootstrap 자료를 modeling 한 후 결합하여 최종의 예측 모형을 산출하는 방법이다.

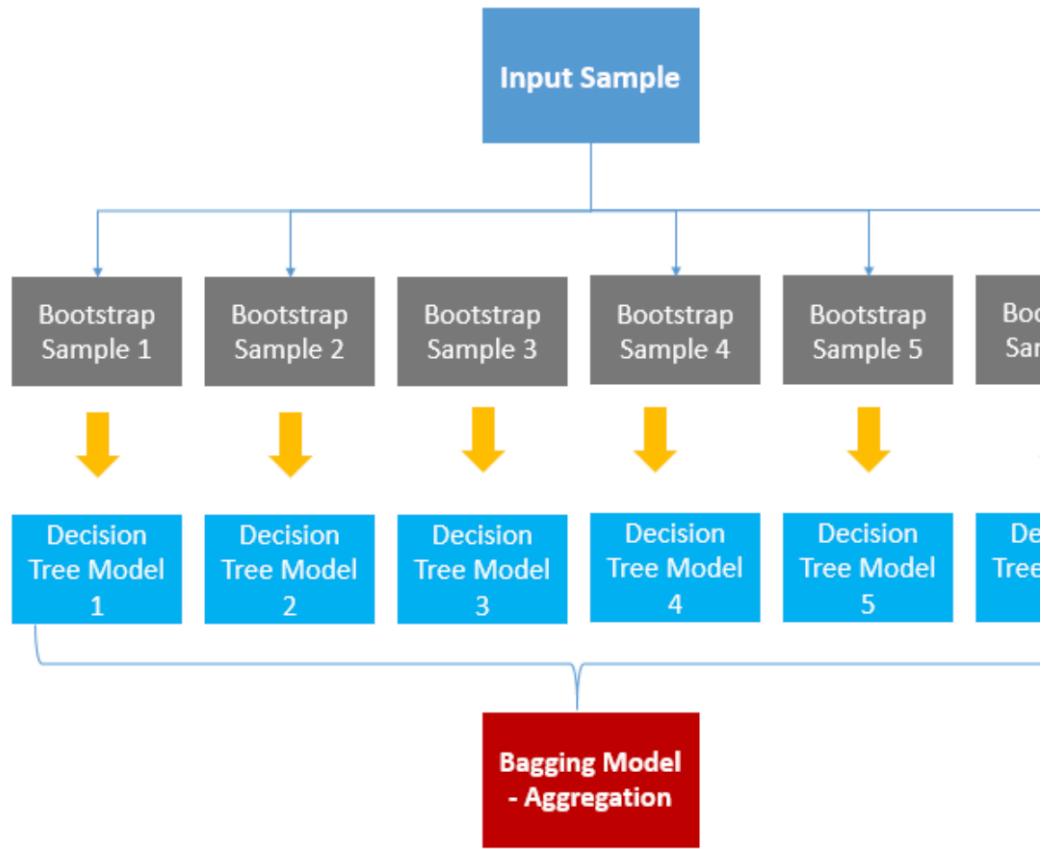
배깅은 예측 모형의 변동성이 큰 경우 예측모형의 변동성을 감소시키기 위해 사용된다.

즉, 원자료로부터 여러 번의 복원 샘플링(sampling)을 통해 예측 모형의 분산을 줄여 줌으로써 예측력을 향상 시키는 방법을 배깅이라고 한다.

따라서 배깅은 일반적으로 과대적합 된 모형, 편의가 작고 분산이 큰 모형에 사용하는 것이 적합합니다.

적은 데이터인 표본으로 decision tree를 만든 다음에 다수결에 의해 class를 리턴 - decision tree와 knn을 결합한 느낌이다.

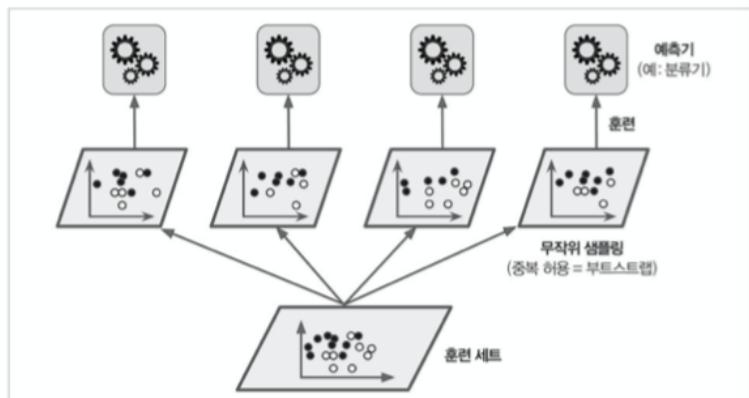
트레이닝 데이터를 반복(복원) 추출하여 표본을 여러개 만든후에 각 표본에 맞는 분류 모델을 표본 수만큼 생성한 후에 각각의 분류모델을 양상화 하는 방법 (투표를 통해서 최고의 모델을 선택)



### 수집함수

수집함수는 전형적으로 분류일때는 통계적 최빈값이고 회귀에 대해서는 평균을 계산 한다.

그림 7-4 페이스팅/배깅의 훈련 세트 샘플링과 훈련



### bootstrapping

통계학에서 중복을 허용한 리샘플링(resampling)을 의미한다.

### 알고리즘

원자료로부터 n번의 랜덤 복원 샘플링을 하고 각 샘플의 모델링을 통해 나온

예측변수들을 결합하여 최종 모형을 생성한다. |

이러한 각 샘플의 예측변수들을 결합하는 방법은 목표변수가 연속형일 때는 평균(average), 범주형일 때는 다중 투표(majority vote)를 사용하는 것이 일반적이다.

### OOB(out of bagging)

여러가지로 표본을 뽑았지만 샘플로 추출되지 않은 데이터  
왜 추출되지 않은 데이터가 존재할까? – 복원 추출방식이기 때문

수학적으로

n개의 샘플에서 하나를 무작위로 추출했을 때 선택되지 않을 확률은  $1 - 1/n$  이다.

그리고 이를 n번 반복했을 때 한번도 선택되지 않을 확률 y는 이 값을 n번 곱하면 된다.

$$y = \left(1 - \frac{1}{n}\right)^n$$

수학적 트릭을 위해 양변에 로그를 취한다.

$$\ln(y) = \ln\left(1 - \frac{1}{n}\right)^n = n \ln\left(1 - \frac{1}{n}\right)$$

그런 다음 새로운 변수  $x = 1/n$ 을 적용한다.

$$= \frac{1}{x} \ln(1 - x) = \frac{\ln(1 - x)}{x}$$

$n$ 이 무한대에 수렴하고  $x$ 가 무한대에 수렴할 때, 로피탈의 정리를 이용하면

$$\ln(y) = \lim_{x \rightarrow 0} \frac{\ln(1 - x)}{x} = \lim_{x \rightarrow 0} \frac{\frac{d}{dx} \ln(1 - x)}{\frac{d}{dx} x} = \lim_{x \rightarrow 0} \frac{-1}{1 - x}$$

이제 로그를 풀고 y에 대해 정리하면 다음과 같다.

$$y = e^{-1} = 0.3678794\dots$$

즉 샘플의 개수  $n$ 이 아주 클 때, 어떤 샘플이 무작위로 n번의 선택에 한번도 포함되지 않을 확률은 36.8%이다.

그리고 샘플 개수가 아주 작을 때 가령  $n=3$ 이면

$$y = \left(1 - \frac{1}{3}\right)^3 = 0.296296\dots$$

그래서 샘플의 개수 차이가 크더라도 대략 30~37% 사이가 된다.

따라서 랜덤 포레스트에서 선택되지 않고 누락될 샘플의 양은 대략 1/3 정도이다.

### OOB 평가

OOB 데이터로 평가 한것

**OOB error(Out-of-bagging error)**

OOB 데이터로 test했을때 발생하는 에러율

## 랜덤포레스트(random forest)

배깅의 한 종류이다.

매 실행시마다 랜덤하게 관측치와 변수를 선택하므로 실행결과가 조금씩 달라지게 한다.

decision tree와 bagging 과 결합한 알고리즘이다.

### IDEA

decision tree의 주요 단점은 훈련 데이터에 과적합되는 경향이 있다는 것이다.

랜덤 포레스트는 이 문제를 회피할수 있는 방법이다.

랜덤 포레스트는 기본적으로 여러 결정 트리의 묶음이다.

각 트리는 비교적 예측을 잘하고 있지만 일부에 과적합되는 경향을 가지고 있음에 기초한다.

서로 다른 방향으로 과대적합된 트리를 많이 만들면 그 결과를 평균값으로써 과대 적합된 양을 줄일수 있다.

이러한 전략을 구현하기 위해서는 결정 트리를 많이 만들어야 한다.

각각의 트리는 타깃 예측을 잘 해야 하고 다른 트리와 구별되어야 한다.

### 학습 원리

1. 주어진 트레이닝 데이터 셋에서 무작위로 중복을 허용해서 n개를 선택한다(배깅)

2. 선택한 n개의 데이터 샘플에서 feature를 중복 허용없이 d개 선택한다.

3. 이를 이용해 의사결정트리를 학습하고 생성한다.

1~3 단계를 k번 반복한다.

1~4 단계를 통해 생성된 k개의 의사결정트리를 이용해 예측하고, 예측된 결과의 평균이나 가장 많이 등장한 예측 결과를 선택하여 최종 예측값으로 결정한다.

### feature 선택

d값은 보통 주어진 트레이닝 데이터의 전체 특성의 개수의 제곱근으로 주어진다.

즉, 트레이닝 데이터의 전체 특성의 개수를 m이라고 하면 d의 값은 다음과 같다.

$$d = m^{(1/2)}$$

### 변수 중요도

랜덤포레스트의 장점은 변수의 상대적 중요도를 측정할수 있다는 것이다.

사이킷런은 어떤 특성을 사용한 노드가 (랜덤 포레스트에 있는 모든 트리에 걸쳐서) 평균적으로 불순도를 얼마나 감소시키는지 확인하여 특성의 중요도를 측정한다.

### 페이스팅(pasting)

다양한 분류기를 만드는 한가지 방법은 각기 다른 훈련 데이터를 사용하는 것이다. 페이스팅은 훈련세트에서 중복을 허용하지 않고 샘플링하는 방식을 의미한다.

### 부스팅(boosting)

일반적으로 분류 문제는 잘못 분류된 개체들에게 관심을 가지고 이들을 더 잘 분류 하는 것이 목적이다.

이러한 아이디어로부터 개발되어진 부스팅(boosting)은 잘못 분류된 개체들에 집중하여 새로운 분류규칙을 만드는 단계를 반복하는 방법이다.

즉, 약한 예측모형들을 결합하여 강한 예측모형을 만드는 것이 바로 부스팅 알고리즘이다.

약한 학습기들의 성능을 올려서 강한 학습기의 성능을 얻는 방법

전체 데이터에서 여러 샘플링 데이터를 추출하여 순차적으로 이전 학습 분류기의 결과를 토대로 다음 학습 데이터의 샘플 가중치를 조정하면서 학습을 한다.

### 적응형 부스팅(adaptive boosting, AdaBoost)

#### IDEA

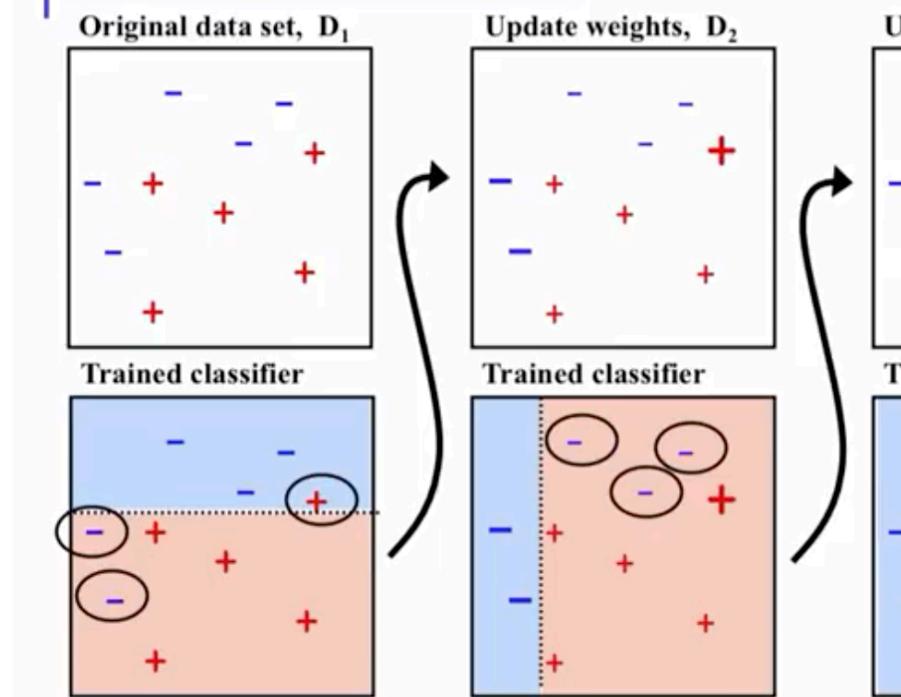
간단한 약분류기(weak classifier)들이 상호보완 하도록 단계적(순차적)으로 학습

약분류기들을 한번에 하나씩 순차적으로 학습시킬 때,

먼저 학습된 분류기가 잘못 분류한 결과 정보를 다음 분류기의 학습 시 사용하여 이전 분류기의 단점을 보완하도록 한다.

즉, 이전 분류기가 오분류한 샘플의 가중치를 adaptive하게 바꾸어가며 잘못 분류되는 데이터에 더 집중하여 잘 학습하고 분류할 수 있게 한다.

## Boosting Example



훈련 샘플의 weight를 업데이트하는게 핵심

처음에는?

각 weight 값은 초기에  $1 / m$  ( $m$ : 훈련 데이터의 수)로 초기화된다.

$$w^{(i)} = 1/m$$

에러율

에러가 일어난다는 건 어떤 데이터에 대해 가중치가 잘못 매겨져있다고 볼수 있다.

따라서 잘못 매겨져있는 가중치를 찾는것을 통해 에러율을 정의한다.

$$r_j = \frac{\sum_{i=1}^m w^{(i)}}{\sum_{i=1}^m \hat{y}_j^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction}$$

$r_j$ 는  $j$ 번째 예측기의 가중치가 적용된 에러율이다.

이 식의 의미는 다음과 같다

$j$ 번째 예측기로 예측한 것들 중 예측이 틀린것의 가중치를 분자로 두고, 전체 가중치는 분모로둔다.

그래서 해당 예측기에서 예측이 틀린 가중치들을 합한 것을 에러율이라고 한다.

이를 더 엄밀하게 정의한 건 다음과 같은 식이다.

$$\text{error}(t) = \sum_{i=1}^n w(t)_i I(h_t(x_i) \neq y_i)$$

where  $n$  is the number of data and  $t$  is the index of the predictor.

where  $h_t(x_i)$  is the predicted value of  $x_i$  with  $t$ th predictor.

where  $I(A)$  is 1 if  $A$  is true, and 0 if  $A$  is false

$h(x)$ :  $h$ 라는 함수는  $x$ 가 threshold를 넘으면 1 넘지 않으면 0으로 만든다.  $= h(x < \text{threshold})$

즉,  $h_t(x_i)$  는  $t$ 번째 예측기에서  $x_i$  데이터에 대해 threshold를 기준으로 예측을 하는 것이다.

$I(x)$ :  $\text{true}$ 가 들어오면 1,  $\text{false}$ 가 들어오면 0으로 만드는 함수이다.

그래서  $I(h_t(x_i) \neq y_i)$  는 예측값과 실제값을 비교해서, 틀린것에 대해 1을 반환하고 맞는건 0을 반환한다.

그러면  $I(x)$ : 에 의해 반환되는 값은 틀린 값들만 1이 되므로

$\sum(w(t)_i * I(x))$  는  $t$ 번째 예측기에서 틀린 값의 weight를 합하는 식이 된다.

weak learner에 대한 평가

에러를 구한후 그 weak learner를 평가한다.  
이를  $a(t)$ 로 나타낸다.

$$a(t) = \frac{1}{2} \log \frac{1-e_t}{e_t}$$

---

예측기가 정확할 수록 에러율이 낮아져 가중치  $a(t)$ 가 높게 된다.

### 업데이트

for  $i = 1, 2, \dots, m$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

1. 위 식을 이용해 각 샘플의 가중치를 업데이트 한다(즉, 잘못분류된 샘플의 가중치가 증가된다)
2. 그리고 모든 샘플의 가중치를 정규화 한다( $w_i$ 의 sum으로 나눔)
3. 업데이트 된 가중치를 사용해서 새 예측기를 훈련시킨다.
4. 3~7 과정 반복
5. 지정된 예측기의 수에 도달하거나 완벽한 예측기가 제작되면 중단한다.

### 예측

Adaboost는 예측을 할때 모든 예측기의 예측을 계산하고 예측기 가중치 ( $a_t$ )를 곱해 예측결과를 만든다.

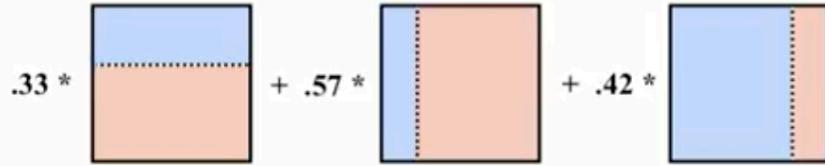
$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$

$\text{sign}$ 은 부호함수임

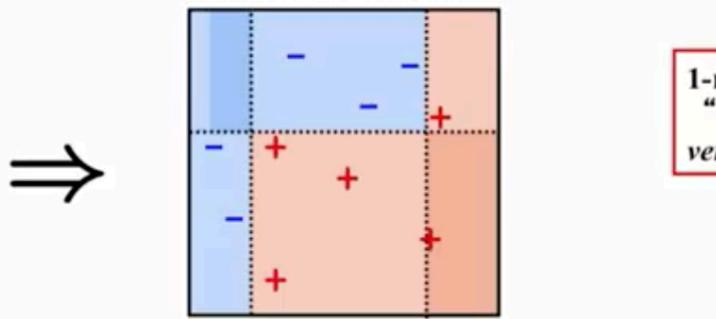
그래서 합한 것들이 0보다 크냐 작냐를 판단해서 Classification을 수행 함

# Boosting Example

Weight each classifier and combine them:



Combined classifier



부스팅을 모든 의사결정 트리에 적용하지 않는 이유

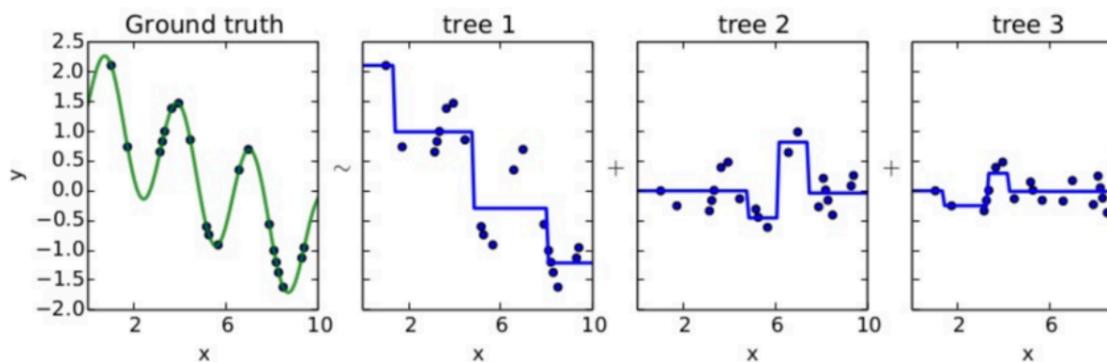
1. 의사결정 트리를 한 번 만드는데 계산 시간이 많이 걸린다면 여러 개의 트리를 만드는 것은 계산적으로 비현실적 일수 있다.
2. 훈련 데이터에 잡음이 많으면 부스팅으로 전혀 개선되지 않을수 있다.

그래디언트 부스팅(Gradient Boosting, GBM(Gradient Boosting Algorithm))

그래디언트 부스팅도 애다부스트 처럼 이전 예측기의 오차를 보정하도록 예측기를 순차적으로 추가하지만,

반복마다 샘플의 가중치를 수정하는 게 아니라 이전 예측기가 만든 잔여오차(residual error)에 새로운 예측기를 학습시킨다.

직관적 이해



GBM을 이해하는 가장 쉬운 방법은 Residual fitting으로 이해하는 것이다.

아주 간단한 모델 A를 통해  $y$ 를 예측하고 남은 잔차를 다시 B라는 모델을 통해 예측하고 A+B모델을 통해  $y$ 를 예측한다면

A보다 나은 B모델을 만들수 있다.

이러한 방법을 계속하면 잔차는 계속해서 줄어들게 되고, training set을 잘 설명하는 예측 모형을 만들 수 있게 된다.

하지만 이러한 방식은 bias를 상당히 줄일 수 있어도, 과적합이 일어날 수도 있다는 단점이 있다.

따라서 GBM을 사용할 때는 sampling, penalizing 등의 regularization 테크닉을 이용하여 더 advanced 된 모델을 이용하는 게 보편적이다.

residual 과 gradient

negative gradient는 residual이 된다.

loss function을 squared error로 설정했을때,

$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

이때  $f(x_i)$ 에 대해 편미분하면 다음과 같은 식이 만들어진다.

$$\frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial \left[ \frac{1}{2}(y_i - f(x_i))^2 \right]}{\partial f(x_i)} = f(x_i) - y_i$$

이에 대해 음의 부호를 취하면  
 $y_i - f(x_i)$  가 된다.(즉, residual이 된다.)

그래서  $f(x_i) - y_i$ 를 negative residual이라 부름

gradient boosting은 다음 모델을 만들 때, negative gradient를 이용해 만들기 때문에 gradient boosting이라 할수 있다.

또한 residual fitting model은 gradient boosting 모델의 한 종류이다.

왜냐하면 다른 loss function을 사용해서 gradient boosting을 할 수 있기 때문이다.

예를 들어, 회귀 문제가 아닌 분류문제라면 loss function으로 squared error 를 이용할 수 없기 때문에 새로운 loss function을 만들어야 하는데, 이 경우에도 negative gradient를 이용해서 새로운 model을 fitting하고 이를 합산해 나가는 식으로 최종 모델을 만들게 된다.

왜 negative gradient를 이용해 새로운 모델을 만드는가?

negative gradient는 pseudo-residual이라고도 불리며, 이것은 어떤 데이터 포인트에서 loss function이 줄어들기 위해  $f(x)$ 가 가려고 하는 방향이다.

이 방향에 새로운 모델을 fitting해서 이것을 이전 모델과 결합하면,  $f(x)$ 는 loss function이 줄어드는 방향으로 업데이트 된다.

### Fitting (시각적 이해)

<https://www.youtube.com/watch?v=3CC4N4z3GJc&t=1s>

<결론>

1. initial tree를 만들고 예측하여 residual 계산
2. residual을 예측하는 새로운 트리를 만들고 이전 트리와 새 트리를 더하여 예측을 수행
3. 트리를 합해 만들어진 모델을 가지고 residual을 계산하여 새로운 tree를 만들고 예측
4. 반복

다음과 같은 data set이 있을때

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

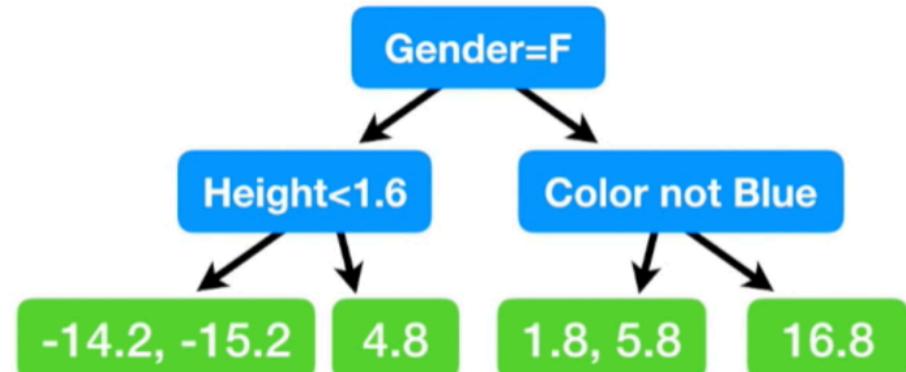
weight를 예측한다고 해보자

Gradient Boost는 single leaf부터 시작하며, 그 single leaf 모델이 예측하는 추정값은 평균이다.

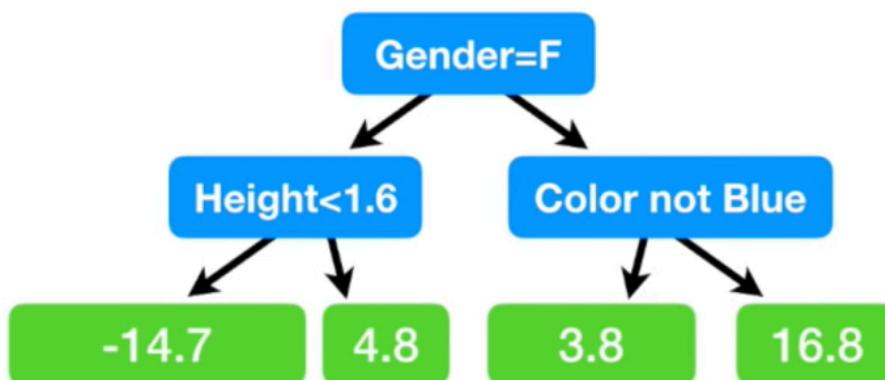
single leaf 모델이 예측하는 값과 weight의 차이를 구하여 residual을 구한다.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

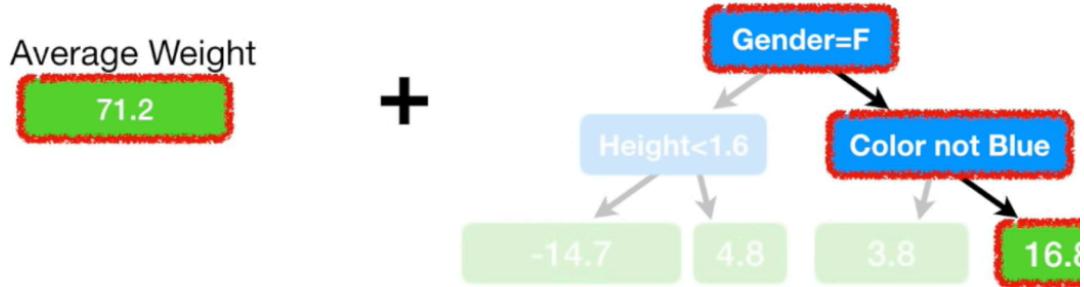
그리고 이 residual을 예측하는 트리를 만들어 본다.



여기서 마지막 leaf 노드에 두개의 residual 값이 같은 경우가 있다.  
이때는 평균한다.

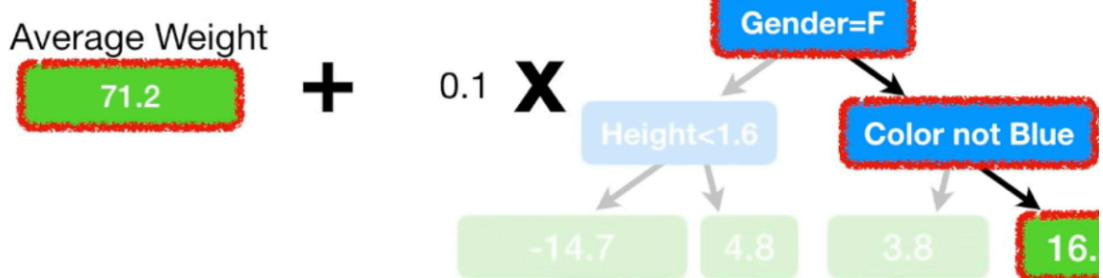


그리고 single leaf 모델과 우리가 구한 트리를 조합하여 몸무게를 예측해본다.



$$\dots \text{so the Predicted Weight} = 71.2 + 16.8 = 88$$

완벽하다. 그러나 이는 overfitting의 위험이 있다.  
scaling 하여 이를 피한다.

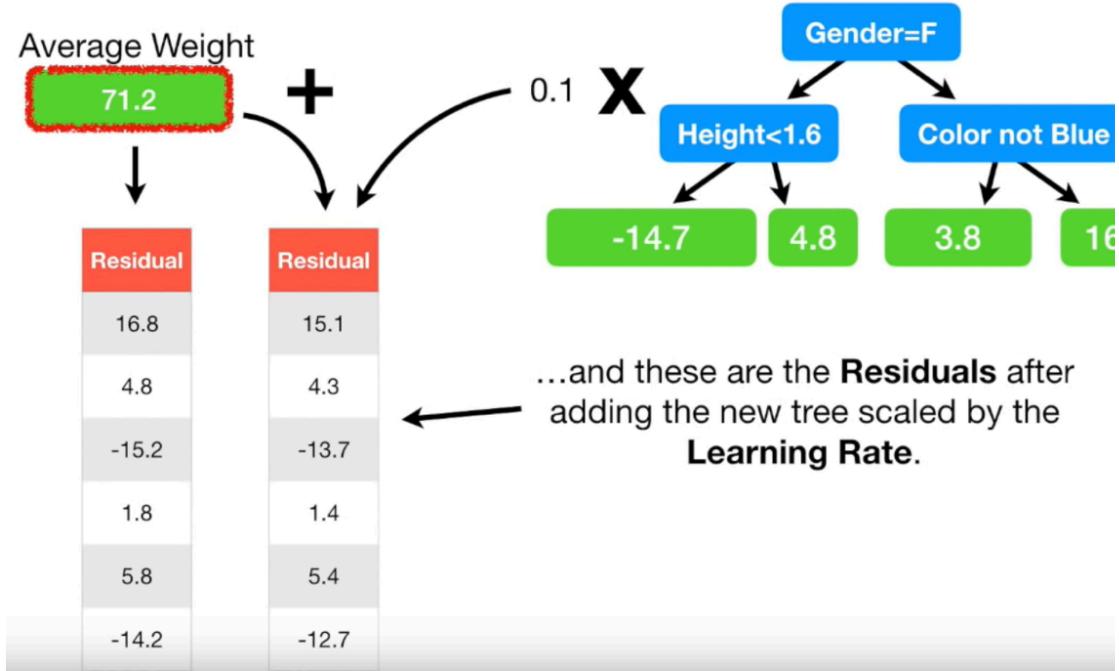


$$\text{Now the Predicted Weight} = 71.2 + (0.1 \times 16.8) = 72.9$$

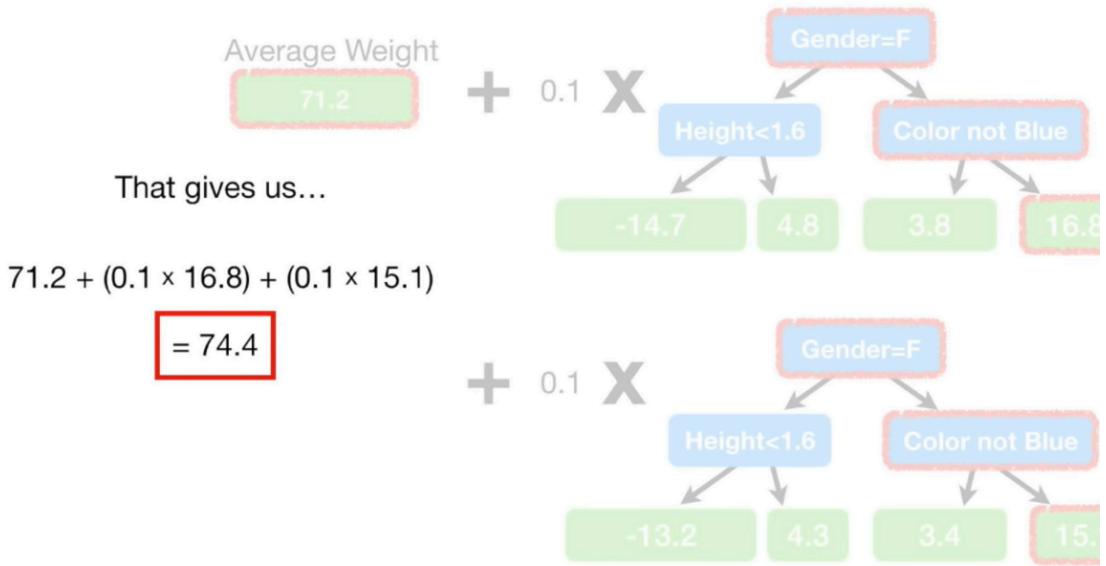
그리고 이 모델에 대한 Residual을 계산하면 다음과 같아진다.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

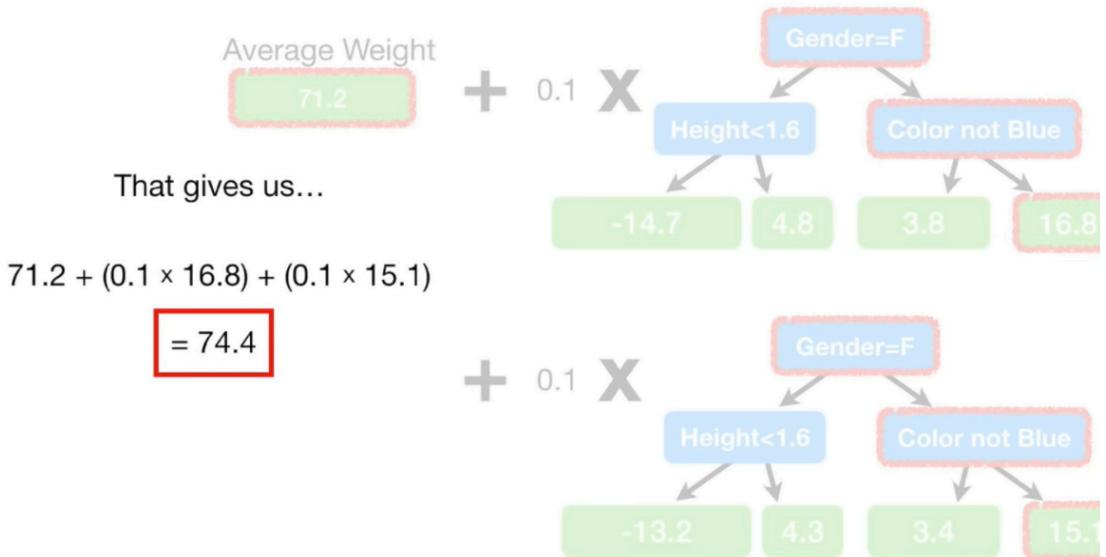
이를 처음 Residual과 비교해보면  
Residual이 떨어진다!



그리고 다음으로 위 모델에서 구한 Residual을 이용하여 트리를 만들고  
모든 트리의 결과를 더하여 새로운 모델을 만든다



이런식으로 트리를 추가해나가면서 모델을 만들고 해당 모델로 예측을 하면 점점 Residual이 떨어진다.



## Regression Details

**Input:** Data  $\{(x_i, y_i)\}_{i=1}^n$ , and a differentiable **Loss Function**  $L(y_i, F(x))$

**Step 1:** Initialize model with a constant value:  $F_0(x) = \operatorname{argmin}_\gamma \sum_{i=1}^n L(y_i, \gamma)$

**Step 2:** for  $m = 1$  to  $M$ :

(A) Compute  $r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$  for  $i = 1, \dots, n$

(B) Fit a regression tree to the  $r_{im}$  values and create terminal regions  $R_{jm}$ , for  $j = 1 \dots J_m$

(C) For  $j = 1 \dots J_m$  compute  $\gamma_{jm} = \operatorname{argmin}_\gamma \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update  $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

**Step 3:** Output  $F_M(x)$

$F(x)$ : Prediction

n: 데이터의 개수

m: 트리 index

M: 총 트리 개수

i: sample index

j: leaf index

J\_m: m 번째 tree의 Leaf의 수

$\gamma$ : 예측값

r: 잔차

R: leaf

$\operatorname{argmin}$ : 어떤 함수가 최소가 되는 파라미터 선택

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

### Input

Input: 데이터셋을 집어 넣고, Loss function을 넣는다.

보통 Loss function은 다음과 같은 식을 넣는다.

$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

이는 다음과 같은 의미임

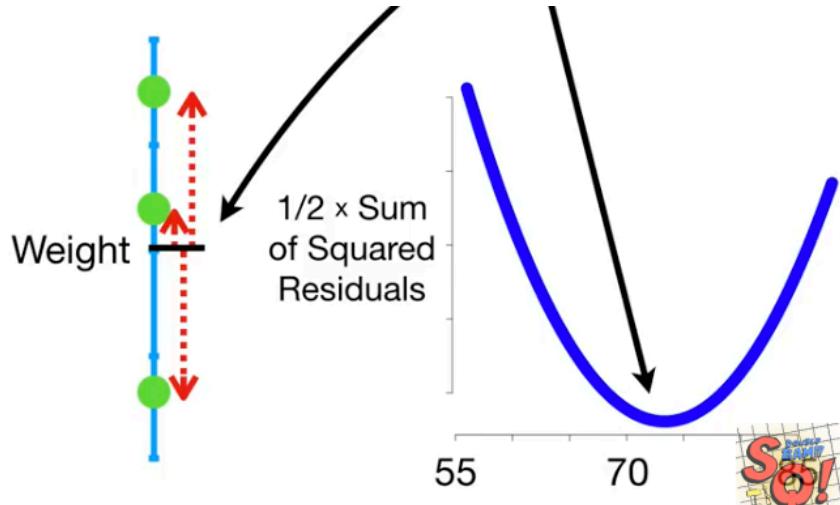
$$\frac{1}{2} (\text{Observed} - \text{Predicted})^2$$

### Step 1

첫번째 모델을 생성한다.

이 첫번째 모델을 생성할 때 Loss function이 최소화가 되는  $\gamma$ 를 찾게 되는데

이  $\gamma$ 가 최소가 되려면 미분하여 0이 되는 지점을 구하면 됨



$$\begin{aligned}
 & \frac{1}{2} (88 - \text{Predicted})^2 + \rightarrow -(88 - \text{Predicted}) + \\
 & \frac{1}{2} (76 - \text{Predicted})^2 + \rightarrow -(76 - \text{Predicted}) + \\
 & \frac{1}{2} (56 - \text{Predicted})^2 \rightarrow -(56 - \text{Predicted}) \\
 & \frac{d}{d \text{ Predicted}}
 \end{aligned}$$

이를 계산하면 평균이 된다.

$$\text{Predicted} = \frac{88 + 76 + 56}{3}$$

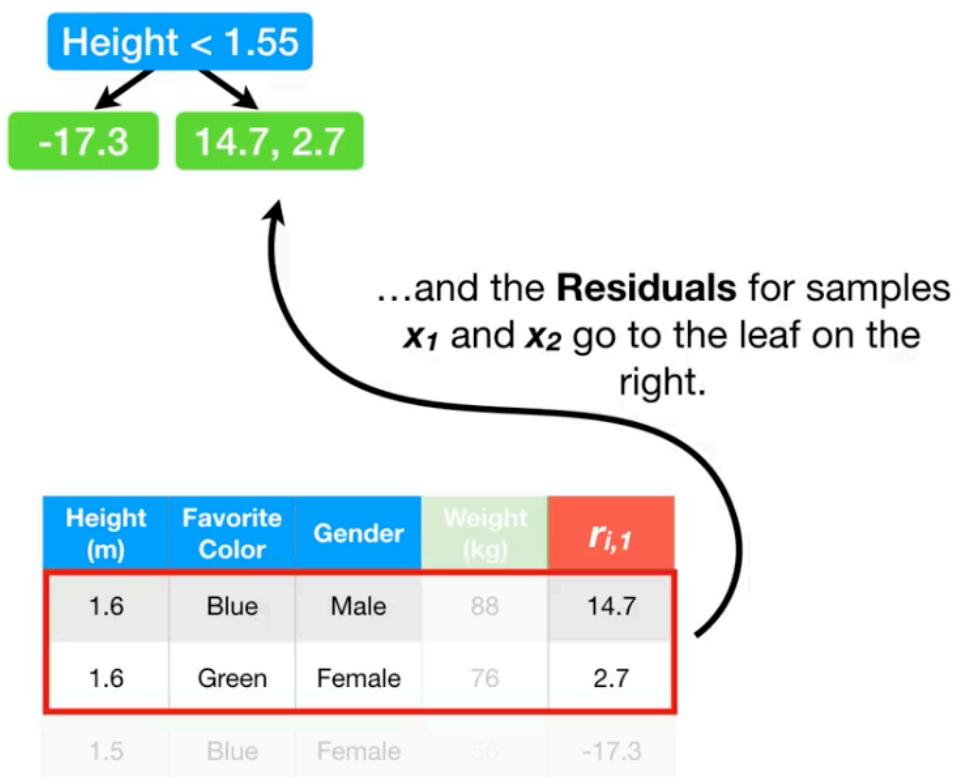
### Step 2

#### 잔차의 계산

잔차는 Loss function에 대해 이전 예측 함수를 가지고 미분한 것에 음의 부호를 취하여 구할 수 있음  
(negative residual)

Height (m)	Favorite Color	Gender	Weight (kg)	$r_{i,1}$
1.6	Blue	Male	88	14.7
1.6	Green	Female	76	2.7
1.5	Blue	Female	56	-17.3

Regression tree 생성  
 residual을 fitting하는 decision tree를 생성한다. (잔차를 prediction 함)  
 $R_{jm}$  은 m번째 트리의 j번째 leaf



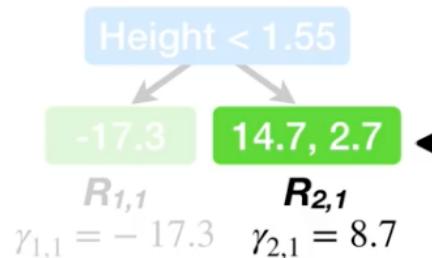
모든 Leaf에 대해 Loss function이 최소가 되는 잔차의 Prediction 선택  
 잔차로 만든 트리의 Leaf에 잔차값이 겹칠 수 있다. 이를 해결해주는 방법이다.

$$\gamma_{jm} = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

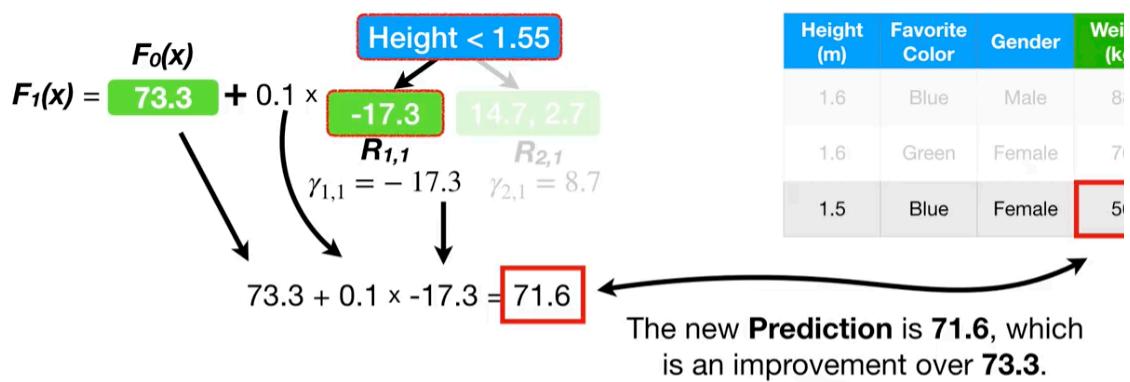
m번째 트리의 j번째 leaf를 위의 loss function이 최소화되는 잔차의 prediction 을 구한다.  
 $x_i$  가  $R_{ij}$ 에 속한다는 말은  $R_{ij}$  Leaf에 속하는 데이터만 사용한다는 의미임

최소화 되는 값을 구하라면 당연히 미분해서 0이 되는 지점을 택하면 됨

$$\frac{d}{d\gamma} \left[ \frac{1}{2}(14.7 - \gamma)^2 + \frac{1}{2}(2.7 - \gamma)^2 \right] \longrightarrow -14.7 + \gamma + -2.7 + \gamma = 0$$



### 예측 업데이트



있게 됨

(D) Update  $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

이전 예측 값에 잔차 예측 값을 더함으로써 더 나은 예측을 만들수

앞에 ν는 overfitting을 막기 위한 학습률

STEP3

If  $M = 2$ , then  $F_2(x)$  is the output from the **Gradient Boost** algorithm.

$$F_2(x) = F_0(x) + 0.1 \times \begin{cases} -17.3 & \text{Height} < 1.55 \\ 14.7, 2.7 & \end{cases} + 0.1 \times \begin{cases} -15.6 & \text{Height} < 1.55 \\ 13.8, 1 & \end{cases}$$

$$\gamma_{1,1} = -17.3 \quad \gamma_{2,1} = 8.7 \quad \gamma_{1,2} = -15.6 \quad \gamma_{2,2} = 1$$

### Pseudo Residual

Loss function을 다음과 같이 취하면

$$j(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$$

이를 미분했을 때 negative residual이 나오지만

앞의 계수를 1/2이 아니라 2와 같은 값을 취할 수도 있는데 이렇게 하면 residual이랑 완전히 같다고 볼 수 없게 된다.

그니까 비슷한 residual이란 의미로 Pseudo Residual이라고 함

### Classification

$\log(\text{odds})$ 를 사용해서 첫 번째 트리를 생성



Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

When we use **Gradient Boost for Classification**, the initial **Prediction** for every individual is the  **$\log(\text{odds})$** .

I like to think of the  **$\log(\text{odd})$**  as the **Logistic Regression** equivalent of the average.

로지스틱 함수에  $\log(\text{odds})$ 를 집어 넣어 Prediction값을 도출한다.

$$\log(4/2) = 0.7$$

Probability of Loving Troll 2 = 0.7

**NOTE:** These two numbers, the **log** and the **Probability** are the same because I'm rounding. If I allowed digits passed the decimal place..

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

$$\log\left(\frac{4}{2}\right) = 0.6931$$

$$\frac{e^{\log(4/2)}}{1 + e^{\log(4/2)}} = 0.6667$$

그리고 잔차를 계산하면 다음과 같음

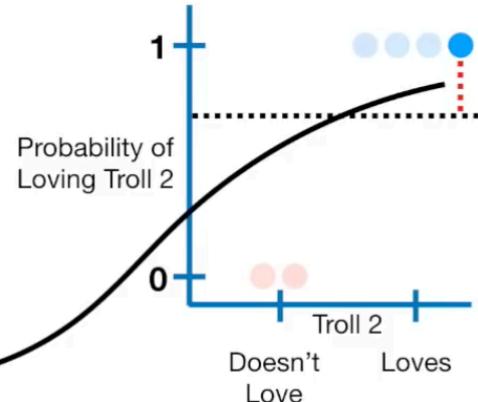
$$\log(4/2) = 0.7$$

Probability of Loving Troll 2 = 0.7

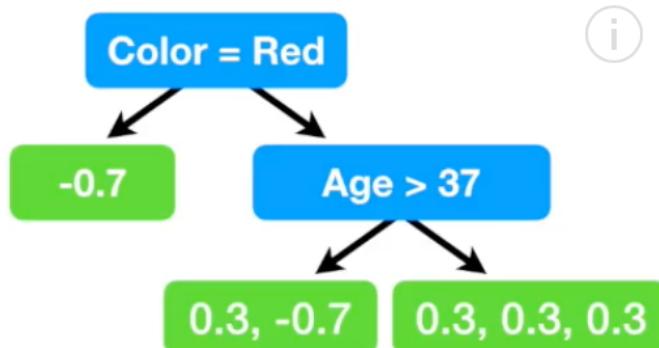
Then we calculate the rest of the **Residuals**...

$$\text{Residual} = (\text{Observed} - \text{Predicted})$$

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3



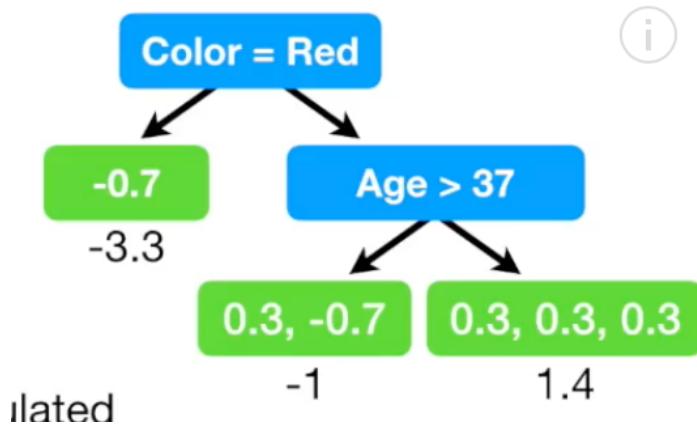
잔차로 트리를 만든다



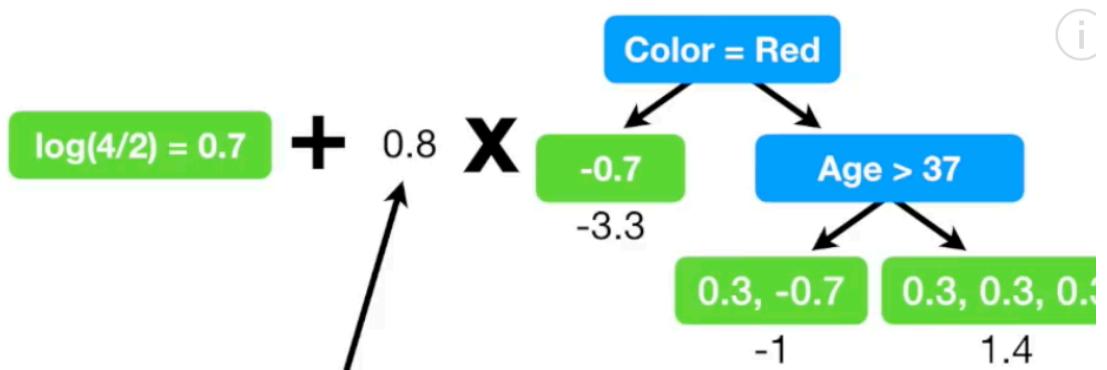
그리고 첫번째 트리값과 합해줘서 새로운 모델을 만들어 줘야하는데,  
 $\log(\text{odds})$  랑 residual이랑 바로 합해줄수 없어서  
 어떤 함수에 residual을 넣고 합해야 한다  
 그 함수의 공식은 다음과 같다.

$$\frac{\sum \text{Residual}_i}{\sum [\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)]}$$

이 공식을 사용하면 다음 값들이 도출되고



이를 scaling해서 합해주면 각각의 데이터마다 새로운  $\log(\text{odds})$ 값이 도출될 것이다.



그런데 우리는 Predication 값을 원하므로 각  $\log(\text{odds})$ 를 Predication으로 변환해주어야 한다.

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.
Yes	12	Blue	Yes	0.9
Yes	87	Green	Yes	0.5
No	44	Blue	No	
Yes	19	Red	No	
No	32	Green	Yes	

**NOTE:** This new predicted probability is worse than before, and this is one reason why we build a lot of trees, and not just one.

$$\text{Probability} = \frac{e^{-0.1}}{1 + e^{-0.1}} = 0.5$$

$$\log(\text{odds}) \text{ Prediction} = 0.7 + (0.8 \times -1) = -0.1$$

이렇게 하면 더 잔차가 줄어든 fitting 된 모델이 나온다.

여기서 한 스텝 더 가면

새 Prediction 값이 나왔으니까 이에 대한 잔차도 구해줄수 있다.

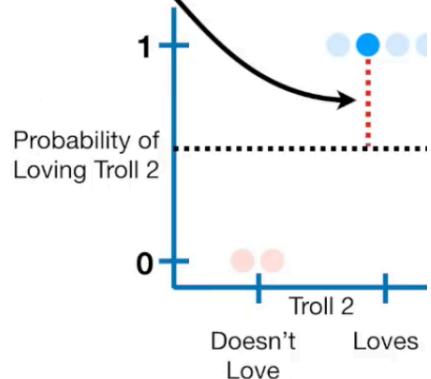
그런데 주의할점은, 이전 모델에서는 Prediction 값이 하나여서 저 점선이 각 데이터마다 일정했는데,

지금 모델에서는 데이터마다 Prediction값이 다르기 때문에 저 점선이 각 데이터마다 달라진다는 것이다.

그래서 각기 다른 Prediction 값과 label 값의 차이를 구해 잔차를 구해준다.

...and the **Residual** is the difference.  $\rightarrow \text{Residual} = (\text{Observed} - \text{Predicted})$

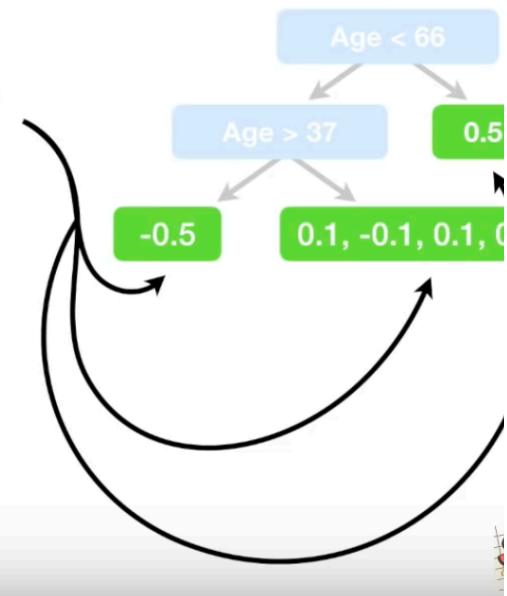
Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	
No	44	Blue	No	0.5	
Yes	19	Red	No	0.1	
No	32	Green	Yes	0.9	
No	14	Blue	Yes	0.9	



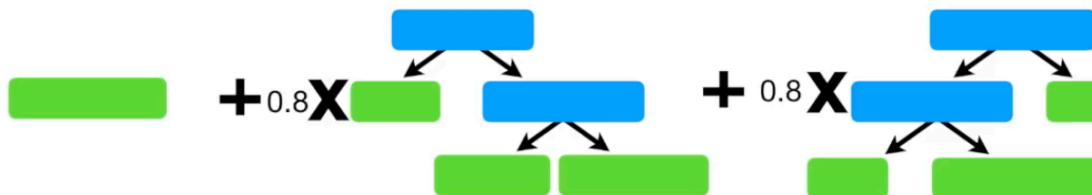
잔차가 구해졌으니 새로운 트리도 만들수 있다

...and then we need to calculate the **Output Values** for each leaf.

Likes popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	0.5
No	44	Blue	No	0.5	-0.5
Yes	19	Red	No	0.1	-0.1
No	32	Green	Yes	0.9	0.1



이런 식으로 계속해서 잔차 트리를 만들어가면서 예측을 계산하면 더 fitting 된 모델을 만들수 있다.



### 카파 계수(kappa coefficient)

카테고리 정보에 대한 2명의 평가자의 일치도를 측정하는 통계적 지표를 의미한다.

이 지표는 0(완전불일치) 또는 1(완전일치) 사이의 값을 가지게 된다.

kappa 계수의 값이 높다는 것의 의미는 신뢰도가 높다는 것이다.

$$k = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

$\Pr(a)$ : 데이터 관찰된 2명의 평가자들의 일치 확률

$\Pr(e)$ : 2명 평가자들이 데이터로 부터 계산된 확률적으로 일치할 확률

ex)

시험을 응시한 학생이 100명이라 할 때 2명의 평가자가 합격, 불합격을 판정했다.

평가자A

합격 불합격

평가자B 합격 40 10

불합격 20 30

평가자 A와 B는 모두 40명에게 합격, 30명에게 불합격을 주었습니다.

$\Pr(a)$ 는 2명의 평가자들의 일치확률이므로 0.7

$\Pr(e)$ 를 계산하기 위해서는 평가자 A와 평가자 B의 각각의 합격, 불합격을 줄 확률

평가자 A는 합격 60, 불합격 40이므로 평가자 A의 합격 확률은 60%, 불합격 확률은 40%

평가자 B는 합격 50, 불합격 50이므로 평가자 A의 합격 확률은 50%, 불합격 확률은 50%

평가자 A와 B 둘 모두가 확률적으로 합격을 줄 확률은  $0.6 * 0.5 = 0.3$

평가자 A와 B 둘 모두가 확률적으로 불합격을 줄 확률은  $0.4 * 0.5 = 0.2$

$pr(e)$ 는 데이터로부터 계산된, 확률적으로 일치학할 확률이므로 둘을 더해야 한다.  
 $pr(e) = 0.3 + 0.2$

### 기술통계(descriptive statistics)

우리가 수집한 데이터를 묘사하고 설명하는 통계기법

### 추측통계(Inferential statistics), 추론통계

모집단의 정보를 알기위해서 표본을 통해 나온 통계량으로 모집단의 모수를 추정하고 추론한다.

#### 표본추출

모집단 -----> 표본

#### 가설

----->  
모수 추정                      통계량  
-----<

#### 추정

모집단의 모수를 알지 못해서 표본의 통계량을 가지고 모집단의 모수를 추정하고 추론한다.

ex)

A후보 지지율이 48%

B후보 지지율이 52%

#### 점추정

어떤 한점을 추정

#### 구간추정

범위 추정

#### 유의수준(significance level), 신뢰수준

신뢰구간 안에 모수가 들어 있을 확률( $1 - \alpha$ )

참값을 구하기 위한 작업을 많이 반복했을 때 참 값이 특정 범위에 있는 비율

또는 "방법의 정확도"를 뜻함(After a sample is taken, the population is either in the interval made or not, there is no chance)

ex)

10,000 명이 치른 통계 시험성적이 있을때,

100명을 랜덤 샘플링해서 샘플로부터 99%의 신뢰수준으로 평균이  $\mu$

$\pm a$  이다 라고 추정을 했을때,

이때 99%를 신뢰수준이라하고  $\mu \pm a$ 를 신뢰구간(confidence

interval)이라고 한다.  
신뢰수준 99%란, 신뢰구간을 구하는 일을 무한히 반복할 때 99%의 경우엔 신뢰구간안에 모집단의 평균이 있다는 뜻이다.

### 가설

모집단의 모수를 가지고 가설을 세우면서 통계량이 맞는지 틀리는지 알수 있다.

### 허용 오차(margin of error), 오차 범위

설문 조사 등의 결과에서 랜덤 샘플링 오차의 양을 나타내는 통계학 용어다.

### 중심극한 정리

동일한 확률분포를 가진 독립변수  $n$ 개의 평균의 분포는  $n$ 이 적당히 크다면 정규분포에 가까워진다.

### 전수조사

모집단의 전체를 조사

### 표본조사

모집단에서 표본을 추출해서 나온 부분집합들을 가지고 모집단의 모수를 추정한다.

### 표본비율

지지율, 불량률, 실업률등 비율을 추정하는 것은 실제 데이터 분석에서 매우 중요하다.

$P(\hat{p}) = \text{대상수} / \text{표본수} = x / n$

### 표본비율의 평균

표본비율의 평균은 모비율이다. =  $p$

### 표본비율의 분산

표본비율의 분산은  $pq / n$

### 모비율 추정

#### 점추정

표본비율 자체가 추정값임

#### 구간추정

$p \pm (\text{오차한계} * \sqrt{pq / n})$

### 퍼셉트론(perceptron)

1957년 Rosenblatt가 퍼셉트론 알고리즘을 고안했다.

사람의 뇌의 동작을 전기 신호 on/off로 흉내낼수 있다는 이론을 증명했다.

인공신경망 모형의 하나이다.

### 기초개념

동물의 신경계를 본따 만들었다.

퍼셉트론은 다수의 신호(Input)를 입력받아서 하나의 신호(Output)을 출력한다.

이는 뉴런이 전기신호를 내보내 정보를 전달하는 것과 비슷해 보인다.

weight와 입력신호를 통해 계산된 값이 임계값( $\theta$ )을 넘었을때 1을 출력한다.

퍼셉트론의 출력 값은 1 또는 0(or -1)이기 때문에 선형 분류(linear classifier) 모형이라고도 볼 수 있다.

보통 실수형의 입력 벡터를 받아 이들의 선형조합을 계산하는 것이며 벡터의 내적

과도 유사하다.

### 기계 학습이 하는 일

여기서 기계학습이 하는 일은 이 weight(입력을 조절하니 매개변수로도 볼 수 있음)의 값을 정하는 작업이라고 할 수 있다.

### 동작 방식

각 노드의 가중치와 입력치를 곱한 것을 모두 합한 값이 활성함수에 의해 판단되는 데,

그 값이 임계치(보통 0)보다 크면 뉴런이 활성화되고 결과값으로 1을 출력한다.  
뉴런이 활성화되지 않으면 결과값으로 -1(or 0)을 출력한다.

$$w_1x_1 + w_2x_2 + \dots + w_nx_n > \theta \Rightarrow 1$$

$$w_1x_1 + w_2x_2 + \dots + w_nx_n \leq \theta \Rightarrow 0$$

### 학습방법

학습 데이터를 퍼셉트론 모형에 입력하며 분류가 잘못됐을 때 weight를 개선해 나간다.

퍼셉트론은 모든 학습 데이터를 정확히 분류시킬 때까지 학습이 진행되기 때문에 학습 데이터가 선형적으로 분류될 수 있을 때 적합한 알고리즘이다.

### 가중치와 편향

퍼셉트론 수식에서 나오는 세타를  $-b$ 로 치환하여 좌변으로 넘기면

$$b + w_1x_1 + w_2x_2 < 0 \Rightarrow 0$$

$$b + w_1x_1 + w_2x_2 \geq 0 \Rightarrow 1$$

위와 같이 되며 여기에서  $b$ 를 편향이라고 할 수 있다.

기계학습 분야에서는 모델이 학습 데이터에 과적합 되는 것을 방지하는 것이 중요하다.

앞에서 설명했듯이 편향은  $\theta$ 로 학습 데이터가 가중치와 계산되어 넘어야 하는 임계점으로 이 값이 높으면 높을 수록 그만큼 분류의 기준이 엄격하다는 것을 의미한다.

그래서 편향이 높을 수록 모델이 간단해지는 경향이 있으며(변수가 적고 더 일반화되는 경우) 오히려 과소적합(underfitting)의 위험이 발생하게 된다.

반대로 편향이 낮을 수록 한계점이 낮아 데이터의 허용범위가 넓어지는 만큼 필요 없는 노이즈가 포함될 가능성도 높다.

이를 편향과 분산의 트레이드오프 관계라고 부른다.

결국 핵심은 학습의 과정에서 불순물(Noise)가 얼마나 포함되고 이것을 잘 걸러내줄 수 있느냐가 관건이다.

### 요약

가중치(weight)는 입력신호가 결과 출력에 주는 영향도를 조절하는 매개변수이고,

편향은 뉴런(또는 노드;  $x$ 를 의미)이 얼마나 쉽게 활성화(1로 출력: activation)되느냐를 조정하는(adjust) 매개변수이다.

### 편향이 필요한 이유

편향이 없다면 무조건 원점을 지나는 직선의 방정식이 나오기 때문에 즉 입력값과 출력값 사이에 편향이 필요하다!

입력값 ( $X$ ) ----- sum() -----> 출력값  
( $Y$ )  
+

## bias

### 벡터 표현

$y = f(x \cdot w + b)$ 로 표현할 수 있다.

### 한계점

퍼셉트론의 한계는 선형으로 분류를 할 수 있지만 XOR와 같이 선형 분류만 가능하며 비선형 분류는 불가능하다는 점이다.

XOR 논리는 exclusive(배타적) 논리연상이다.

$x_1$ 과  $x_2$  어느 한쪽이 1일때만 1을 출력한다.

그런데 XOR 형태의 데이터는 선형으로 분류가 불가능하다.

즉, 직선 하나로 나눈 영역만 표현할수 있어 XOR와 같은 데이터 형태는 분류가 불가능하다는 한계가 있다.

### 다층 퍼셉트론을 통한 한계 극복

단일 퍼셉트론으로는 XOR을 분류할수 없지만, 다층 퍼셉트론을 만들면 이를 극복할 수 있다.

다층(multi-layer) 퍼셉트론이라는 말은 하나의 퍼셉트론에 또 다른 퍼셉트론을 덧붙인다는 의미로 볼 수 있다.

단층 퍼셉트론이 비선형 영역을 분리할 수 없다는 것이 문제이며 다층으로 할 경우 비선형으로 이를 해결할 수 있다.

### step function

퍼셉트론이 사용하는 활성화 함수이다.

임계값이 0을 넘으면 1을 출력하고 그외에는 0을 출력한다.

### 단층 퍼셉트론

일반적으로 단층 퍼셉트론은 step function을 활성화함수로 사용한 모델을 가리킨다.

### 다층 퍼셉트론(Multi-Layer Perceptron, MLP)

층이 여러개이며 sigmoid function을 활성화함수로 사용하는 네트워크를 가리킨다.

#### 층

처음 층을 1층(입력층), 그 다음은 2층 ..., 마지막 층을 출력층이라고 한다.

## 신경망(Neural Network)

신경망은 뉴런을 입력 데이터와 학습 문제의 답에 해당하는 출력 노드에 연결해 만든다.

### 신경세포

뉴런의 입력은 다수이고 출력은 하나이며 여러 신경세포로부터 전달되어 온 신호들을 합산하여 출력한다.

합산된 값이 설정값(threshold) 이상이면 출력신호가 생기고 이하이면 출력신호가 없다.

### 최종

#### perceptron과의 비교

퍼셉트론이 하나의 뉴런 단위로 다루어진다면 각 뉴런이 모여 하나의 뇌가 되는 것과 같은 신경망은 퍼셉트론의 하위요소일 것이다.

단층 퍼셉트론이 하나의 나무와 같이 신경망은 나무가 모여 숲을 이룬 것과 같은 느낌으로 이해하면 된다.

### 층(layer)

신경망은 Input(입력층), Hidden(은닉층), Output(출력층)으로 표현할 수 있다.

은닉층은 양쪽의 입력층과 출력층과는 달리 우리 눈에는 보이지 않기 때문에(내부적으로 돌아가고 있는 Black Box와도 같다.)

Hidden이라고 한다.

### 전방향 신경망(feed-forward neural network)

같은 층의 뉴런들 사이에는 연결이 없고 위층에서 아래층으로의 데이터 전송도 없는 신경망

### 은닉층(hidden layer)

입력층과 출력층에 끼어있는 층

### 뉴런의 개수

대개 신경망은 원래 입력의 압축된 표현들을 배우도록 강제하기 위해 은닉층이 입력층보다 뉴런의 수가 적다.

### 행렬 표현

신경망을 수학적으로 연속된 벡터와 행렬 연산으로 표현할 수 있다.

한 개의 입력 데이터가 4개 항목으로 구성된다면, 이 데이터를 벡터로 표현할 경우 4차원 벡터라고 한다.

이 4차원 벡터 한개를 인스턴스라고 한다.

이 인스턴스를 행렬로 표현하는 방식은 Column vector 방식과 Row vector 방식이 있다.

관계형 데이터베이스는 row vector 방식을 사용하는 대표적인 예이다.

그러나 ML에서는 column 벡터 방식을 더 선호한다.

학습에 이용할 데이터를 컬럼 벡터 방식으로 로딩하고 사용한다. (위 쪽이 index를 표현하고, 왼쪽이 변수를 표현하는 방식)

### W 행렬의 요소 표기법

$l$ : W 행렬이 적용될 레이어 인덱스

$j$ :  $l$ 계층의 노드 위치 인덱스

$i$ :  $l-1$  계층의 노드 위치 인덱스

$n$ : 노드의 수

### W\_ij 표기법

$ij$  표기법을 사용할 경우 W 행렬은 학습 데이터(X)와 같은 컬럼벡터 형태로 행렬이 만들어집니다.

따라서 학습 데이터와 W 행렬을 행렬곱하여 Weighted Sum인  $Z^l$ 을 만들기 위해서는 W행렬을 전치해야 합니다.

=>  $Z = \text{transpose}(W) X + b$

### Shape

$ij$  방식 사용시 shape는 다음과 같이 만들어집니다.

$W^{l-1}$ 의 shape =  $(n_{l-1}, n_l)$

=> (input 뉴런의 개수, output 뉴런의 개수)

### W\_ ji 표기법

ji 표기법을 사용할 경우 W행렬은 열벡터 형태로 행렬이 만들어집니다.

따라서 학습 데이터와 W행렬을 행렬곱하여 Weighted Sum인  $Z^l$  를 만들때, 행렬 변환 없이 바로 행렬곱을 할 수 있습니다.

$$\Rightarrow Z = WX + b$$

### Shape

ji 방식을 사용할 경우에 Shape는 다음과 같습니다.

$W^{l+1}$  의 shape =  $(n_l, n_l - 1)$

$\Rightarrow (\text{output 뉴런의 개수}, \text{input 뉴런의 개수})$

### 연결

모든 뉴런의 출력이 다음 층에 있는 모든 뉴런의 입력과 연결될 필요는 없다.

사실 어떤 뉴런이 다음 층의 어떤 뉴런과 연결될지를 결정하는 것은 경험으로부터 얻어지는 기술이다.

### 파라미터 구성

$k$  번째 층의  $i$  번째 뉴런과  $k+1$  번째 층의  $j$  번째 뉴런을 연결하는 가중치를  $w_{i,j}^{(k)}$  라고 하자.

이 가중치들은 파라미터 벡터  $\theta$ 를 구성한다.

그리고 신경망으로 문제를 해결하는 능력은  $\theta$ 에 들어갈 최적값을 찾는 것에 달려 있다.

### 활성화 함수(Accivation Function)

무언가를 해주는 기능을 가진 것으로 활성화(Accivation) 이라는 말 그대로 입력신호의 총합이 활성화를 일으킬지 정하는 역할을 한다.

신경망에서는 전달받은 데이터를 가중치를 고려해서 합산하고 그 값을 활성화 함수를 적용해 다음층에 전달한다.

### 시그모이드 함수(Sigmoid function)

신경망에서 주로 사용하는 활성화 함수이다.

step function에 비해 곡선 형태로 비선형이다.

특정 경계를 기준으로 출력이 확 바뀌어버리는 계단함수와는 달리 시그모이드 함수는 완만하게 매끄럽게 변화하는데 이 매끄러움이

신경망 학습에서 중요하며 활성화 함수로 시그모이드 함수를 사용하는 이유기도 하다.

신경망에서는 활성화함수를 통해서 각 노드(뉴런)로부터 받은 신호를 변환하고 변환된 신호를 다음 뉴런으로 전달한다.

실제로는 계단 함수와 시그모이드 함수는 사용되지 않는 것으로 보인다.

시그모이드 함수는 값을 실수형으로 가진다.

시그모이드 함수의 매끄러움은 가중치 값을 전달할 때 좀더 부드럽게 양을 조절해서 전달할 수 있다는 점이 계단 함수와 다른 점이다.

$$f(t) = 1 / (1 + e^{-t})$$

### IN PYTHON

`tf.sigmoid()`

### tanh

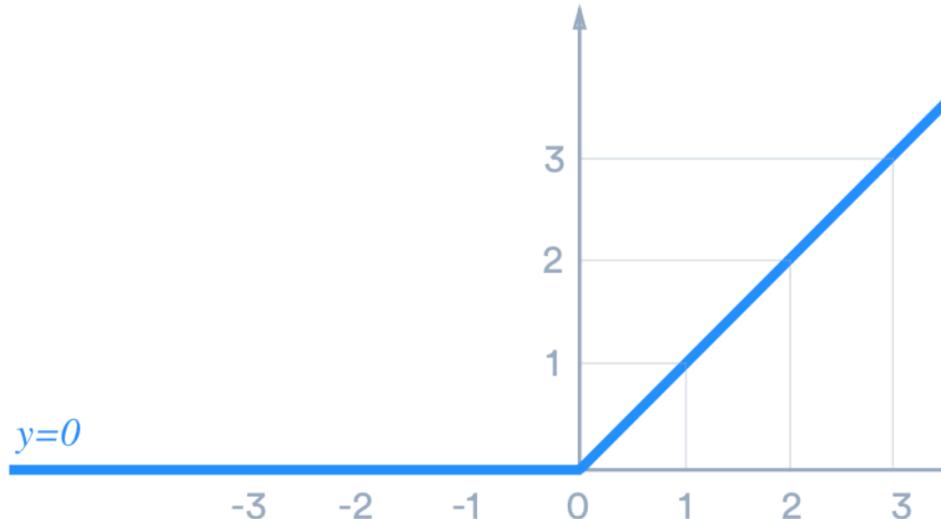
쌍곡 탄젠트 함수는 S자 비선형성을 사용하지만, 결과의 범위가  $-1$ 에서  $1$ 사이이다.

$$f(z) = \tanh(z)$$

**ReLU(restricted linear unit)**

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

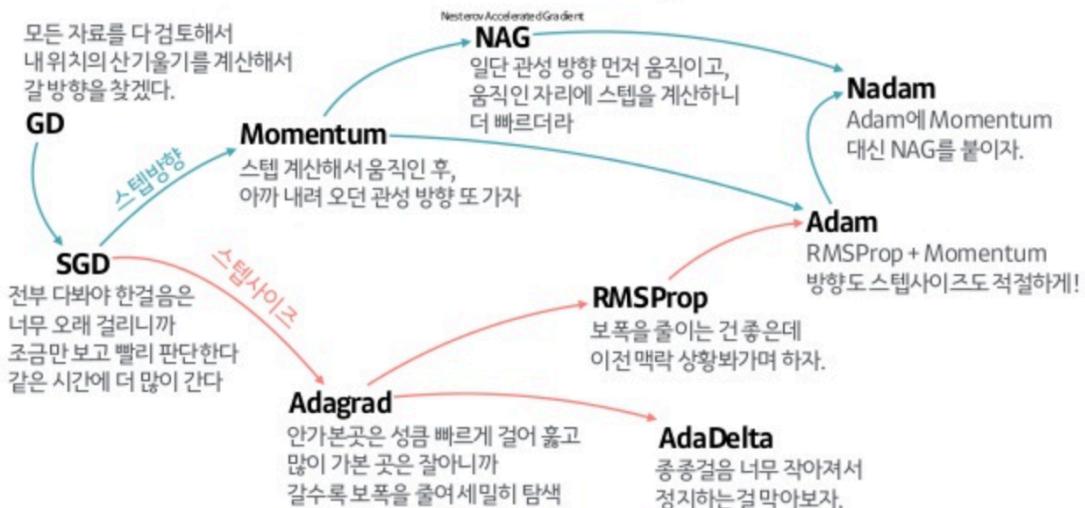
}



### 최적화(Optimization, Optimizer)

최적화 도구는 오차가 최소화될 때까지 파라미터를 반복적으로 조정해 머신러닝 모델의 성능을 최대화하는 것을 목표로 한다.

## 산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



### logit

logistic + probit의 합성어

probit은 확률을 재는 단위라는 뜻이며 odds(두 확률의 비율)의 의미를 그대로 가져온다.

`logit`은 그 값이 0보다 큰지 아닌지로 결정의 기준을 세운다.  
Odds에 Log를 취한 것이다.

### odds ratio

Odds는 Probability의 또 다른 표현법이라고 볼수 있다.

어떤 사건이 일어날 확률을  $p$ 라고 한다면 그 사건에 대한 odds는  $p / (1-p)$ 로 구할 수 있다.

Odds ratio를 이용하면 두 개의 property에 대한 연관성을 확인할 수 있다.

예를 들어 특정 유전자(A)를 갖는 사람의 특정 질병(B)의 Odds ratio를 구하면 특정 유전자(A)와 특정 질병(B)의 연관성을 평가할 수 있다.

### 사용 예

Odds는 도박 배팅이나 혹은 질병의 발병 확률을 표현할 때 많이 사용된다.

예를 들어 경마에서 특정 말이 이길 확률이 75%라고 한다면 Odds는  $0.75 / 0.25$ 가 되며 통산적인 표현으로는 소수 자리 표현보다는 정수 표현을 사용하여 3:1로 표현된다.

4게임 중에 3게임은 이기고 1게임은 진다고 볼 수 있다.

### 뉴런 모델

인공 뉴런은  $n$ 개의 입력( $x_1, x_2, \dots, x_n$ )을 받고, 각각의 입력에 특정 가중치( $w_1, w_2, \dots, w_n$ )을 곱한다.

이러한 가중치 입력은 신경세포의 로짓(Logit)을 생성하기 위해 합쳐진다.

많은 경우 로짓은 상수인 바이어스를 포함한다.

### 벡터 표현

$x = [x_1, x_2, \dots, x_n]$  – 입력

$w = [w_1, w_2, \dots, w_n]$  – 가중치

이 있다면, 뉴런의 출력을 다음과 같이 표현할수 있다.

$b$ 는 바이어스 항이다.

$y = f(x \cdot w + b)$

### 선형 분류기(Linear Classification)

가장 먼저 단순하게 생각해 볼수 있는 분류기는 직선적으로 데이터를 구획해주는 것이다.

즉, 직선(데이터가 3차원인 경우는 평면, 4차원 이상의 데이터에서는 hyperplane)을 그어서 서로 다른 라벨을 지닌 데이터를 분리해주는 것이다.

데이터를 구획해주는 이 직선(또는 평면, hyperplane)의 함수를 우리는 판별함수 (decision function)이라 부른다.

각 관찰에 대한 판별함수값은 판별 점수(decision score), 또는 클래스 점수(class score; 관찰이 해당 클래스=카테고리에 속할 가능성)라고 부르기도 한다.

### 선형 판별 함수

$s = wx + b$  (2차원 데이터에서, 직선 판별함수의 경우)

$s = w_1x_1 + w_2x_2 + b$  (3차원 데이터에서, 평면 판별함수의 경우)

여기서  $s$ 는 관찰 값에 대한 판별 점수,  $x$ 는 관찰의 feature 데이터,  $w$ ,  $b$ 는 각각 판별함수의 '기울기'와 '절편'이다.

이 판별함수를 고차원에 대해 일반화해서 다음과 같이 행렬식으로 간단히 표현할수 있다.

$s = XW + b$

여기서  $s$ 는 관찰값에 대한 판별 점수의 행렬,  $X$ 는 관찰한 features의 행렬,  $w, b$ 는 각각 판별함수의 기울기(?)행렬과 절편(?) 벡터이다.

직선의 경우 기울기와 절편이라 표현하는 것이 직관적이나, hyperplane에서는 이 표현이 직관적이지 않으므로  $w$ 를 가중치 행렬,  $b$ 를 bias 벡터라 부르기로 한다.

위의 판별함수는 데이터의 차원만 다를 뿐, 모든 가중치  $w_i$ 와 feature  $x_i$ 의 선형조합의 꼴로 표현할 수 있으므로 선형 판별함수라 할 수 있다.

**가중치와 편향 이해하기**

가중치  $w_i$ 는 "i번째 feature가 라벨의 예측에 미치는 영향"이라고 해석할 수 있다.

i번째 가중치값  $w_i$ 의 절대값이 클수록 i번째 feature가 예측에 미치는 영향이 크다고 할 수 있고, 따라서 i번째 feature가 라벨의 예측에 중요한 정보라고 판단할 수 있다.

또한 가중치값의 부호에 따라 예측에 미치는 영향이 양의 영향인지, 음의 영향인지 알수 있다.

편향  $b_i$ 에 대한 해석은 가중치에 비해서는 덜 명확하다.

편향  $b_i$ 값은 i번째 feature  $x_i$ 값이 0일때 예측되는 라벨의 값으로 이해할 수 있다.

분류기 식에서 편향을 제거한다면 판별함수는 항상 원점을 지나는 형태일 것이다. 따라서 편향은 기하학적으로는 판별함수가 원점을 지나지 않아도 되도록 해주는 역할을 한다고 이해할 수 있다.

우리가 세운 선형 분류기 식에서,  $X$ 는 관측된 데이터이므로 마음대로 바꿀 수 있는 값이 아니다.

대신 분류기가 데이터  $x_i$ 를 보고 학습함에 따라 새로운 관찰을 더 정확히 분류할 수 있는 방향으로 가중치  $w_i$ 와 편향  $b_i$ 의 값이 변하게 된다.

기하학적으로는 더 정확히 판별 직선(혹은 평면, 또는 hyperplane)을 긋는 방향으로 기울기와 절편 값이 변한다고 이해할수 있다.

### 손실함수(cost function)

선형 분류기가 학습을 하고 있는지, 즉 더 나은 방향으로 선형 분류기의 가중치와 편향 값이 변하고 있는지를 판단하기 위해서는 분류기의 성능을 평가하는 손실 함수를 설정해야 한다.

손실 함수는 분류기의 성능이 낮을수록 손실 함수 값이 커지는 특성을 가져야 한다. 손실 함수를 어떻게 설정하느냐에 따라 선형 분류기의 특성이 달라진다.

### 목적

적절한 회귀선을 찾는다.

적절한 회귀선을 찾는다는것은 회귀 방정식에 존재하는 적절한 파라미터를 찾는것이다. ( $\theta_0, \theta_1, \dots$ )

적절한 파라미터를 찾기위해 목적함수를 둔다.

### 선형 회귀의 손실함수(오차 제곱 평균, Mean Squared Error, MSE)

밑의 방정식은 잔차 제곱의 평균이다.

그런데 2로 한번 더 나눠준 이유는 값을 더 쉽게 보기위해서이다.(값이 더 작아지니까)

$$\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J = \frac{1}{2m} \sum_{i=1}^m \text{cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

$m$ : 자료의 개수

$i$ : 자료의 첨자

$h_{\theta}$ : 가설 함수

$x$ : 입력 변수

$y$ : 출력 변수

#### IN PYTHON

```
from sklearn.metrics import mean_squared_error
mean_squared_error( [예측값들], [종속변수의 값들] )
```

#### 손실함수와 가설함수의 구분

$h_{\theta}(x)$ : 가설함수는  $x$ (입력값)의 함수이고

$J_{\theta}$ : 손실함수는 파라미터( $\theta$ )의 함수이다.

#### 평균 제곱근 오차(Root Mean Square Error, RMSE)

평균 제곱 오차의 값의 단위가 커질수 있으므로 제곱근을 한다.

$RMSE(\theta_1, \theta_2) = \sqrt{MSE(\theta_1, \theta_2)}$

#### 교차 엔트로피 오차(cross entropy error, cee)

$$E = -\sum_k t_k \log y_k$$

여기서  $\log$ 은 밑이  $e$ 인 자연로그,  $t$ 는 정답레이블,  $y$ 는 신경망 출력,  $k$ 는 클래스수  
손실함수는 정답을 맞췄을때 값이 작아야 하며, 정답이 아닌 경우 값이 매우 커야 한다.  
교차 엔트로피는 이를 매우 잘 충족시킨다.

만약 분류문제에서 클래스가 3개이고 이에 대한 예측 확률이 다음과 같다면

$t = [0, 0, 1]$

$y_i = [0.01, 0.01, 0.99]$

1에 대응 하는 (즉 정답레이블에 대응하는) 확률값인  $y_i$ 가 0.99이기 때문에

$\log(y_i)$ 가 매우 작아진다.

그런데  $y_i$ 가 다음과 같은 경우

$y_i = [0.9, 0.09, 0.01]$

$\log(y_i)$ 가 매우 커지기 때문에 비용이 매우 커지게 된다.

즉, 잘못 예측 했기 때문에 패널티를 제대로 받은 것이다.

#### n개의 데이터의 경우

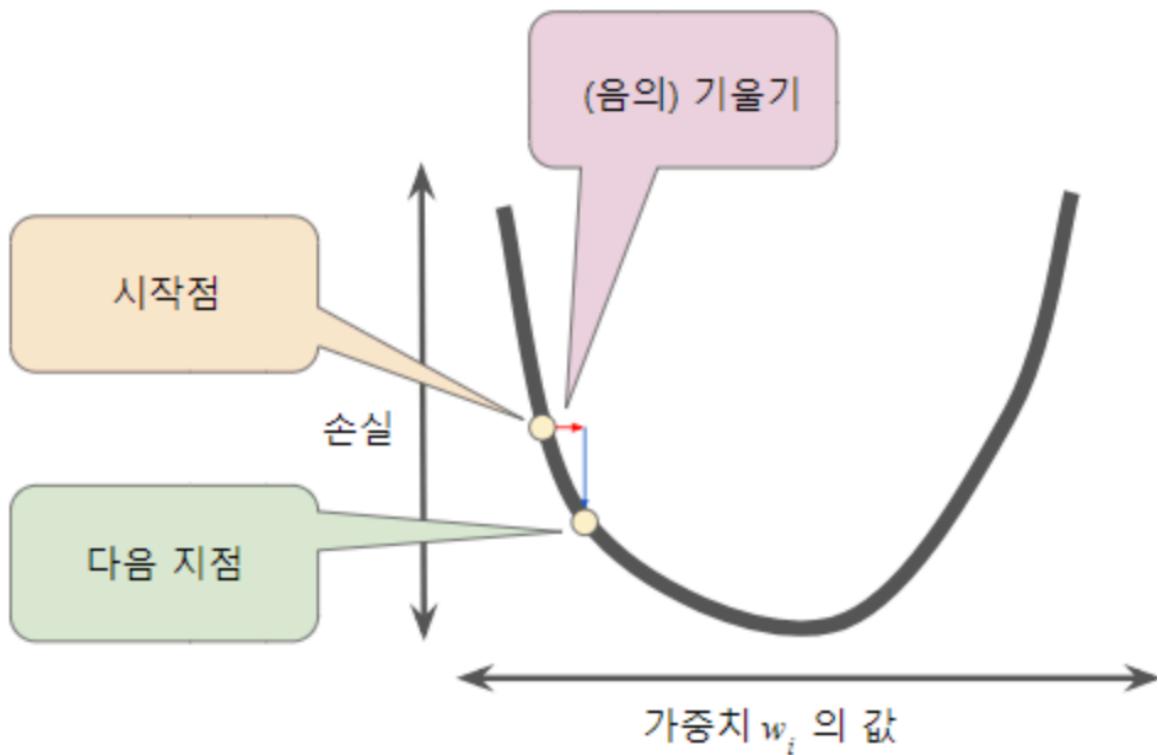
$$E = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$$

#### 경사 하강법(Gradient descent)

가설 함수의 적절한 파라미터( $\theta$ )를 찾기위한 방법이다.

여기서 경사는 각 파라미터에 대한 오차 함수의 편도함수이다. (다른 파라미터를 다 무시

하고 파라미터 하나만 초점을 맞춰 경사를 구하니까 당연히 편도함수)



$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

### 알고리즘

```
- repeat until convergence {
     $\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j}$  ( for j=0 and j=1 )
}
```

여기서  $\alpha$ 는 훈련 비율, 학습률(learning rate)  
 $:=$ 은 할당기호(Assignment)

### Correct Simultaneous update

```
temp0 =  $\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0}$ 
```

```
temp1 =  $\theta_1 := \theta_1 - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1}$ 
```

```
 $\theta_0 := temp0$   
 $\theta_1 := temp1$ 
```

### 주의

경사하강법은 지역최소값에 민감하지만,

선형회귀의 경우, 손실함수가 활모양의 볼록함수이기 때문에 전역최소값을 갖는다.

### Batch Gradient Descent

경사하강법을 이런 용어로도 부르는데, 그 이유는 경사하강법 알고리즘을 사용할때 모든 Example을 사용하기 때문이다. (모두 다 더함)

### Stochastic gradient descent

경사하강법의 한 스텝 업데이트를 진행할때 1개의 트레이닝 데이터만 사용하는 기법

### Mini Batch Gradient Descent

Batch gradient descent와 stochastic gradient descent의 절충적인 기법

mini batch의 수만큼 손실 함수 미분값 평균을 이용해서 파라미터를 한 스텝씩 업데이트 하는 기법이다.

### 학습률 (learning rate)

등고선에 수직으로 이동하는 각 단계에서 새로운 방향을 재계산하기 전에 얼마나 멀리 나갈지를 결정해야 한다.

이 거리는 곡면의 가파른 정도에 의존하는데 왜그럴까?

최소값에 가까울수록 앞으로 나가는 거리는 더 짧아진다.

곡면이 많이 평평해지면 최소값에 가까워졌다는 것을 알 수 있다.

그래서 최소값에 얼마나 가까운지에 대한 지표로써 가파른 정도를 사용할 수 있다. 하지만 오차 곡면(error surface)이 매우 완만하다면 학습하는데 시간이 오래 걸릴 가능성이 있다.

그래서 종종 경사에 학습률을 곱한다.

### 경사

경사는 각 파라미터에 대한 오차 함수의 편도함수(partial derivative)이다.

(경사는 기울기이고, 기울기는 변화율임,

오차 함수는 파라미터에 대한 함수이고, 각 파라미터의 변화율은 경사가 됨

그러므로 각 파라미터의 편도함수는 각 파라미터의 변화율이고, 당연히 각 파라미터의 경사가 될것임)

### Feature Scaling

Gradient descent를 더빠르게 할수 있는 방법이다.

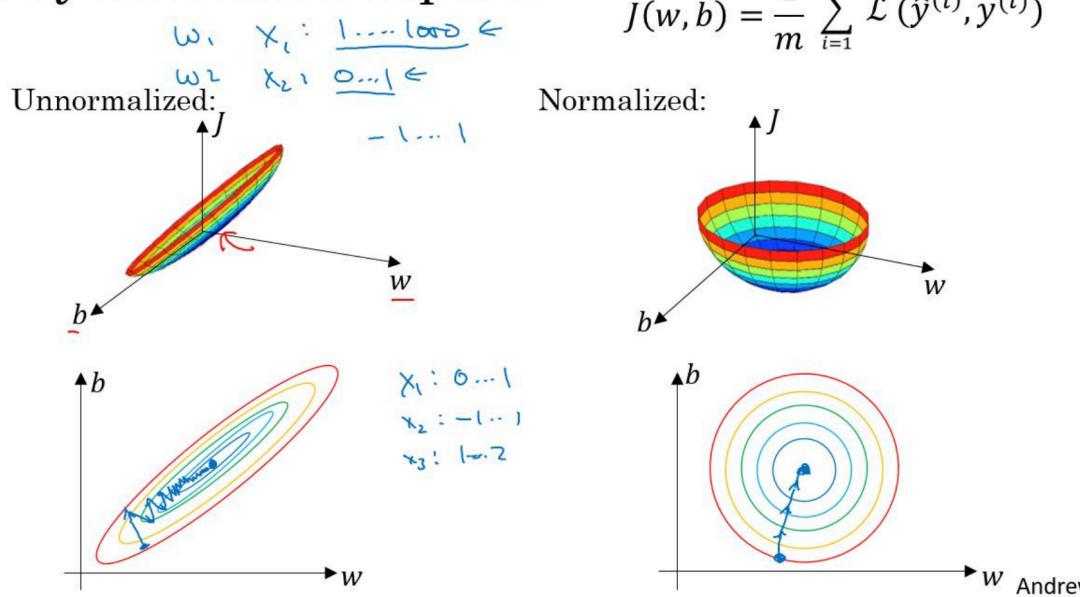
각 Feature의 범위를 제한하여, 손실함수의 등고선이 원에 가깝게 할수 있다.

(각 Feature의 범위가 너무 차이나면, 손실함수의 등고선이 가늘고 긴 타원형이 됨)

여기서 정규화와 표준화 방법을 사용할수 있으며,  $x_0$ 는 적용하지 않는다. (항상 1이므로)

만약 표준편차를 알고있다면 정규화를 사용한다.

## Why normalize inputs?



### 적절한 반복횟수

경사 하강법을 사용할 때 적절한 반복횟수를 알기는 어렵다.

### automatic convergence test

$J(\theta)$ 의 감소량이 어떤 작은 값  $\epsilon$  (ex:  $10^{-3}$ ) 보다 작을 때 convergence 했다고 말함

### $\alpha$ 값 선택

$\alpha$ 가 너무 작으면 slow convergence가 일어남

$\alpha$ 가 너무 크면  $J(\theta)$ 가 매 반복마다 감소하지 않고 심하면 수렴하지 않는다.

앤드류온이  $\alpha$ 를 선택하는 방식은 다음과 같다

$0.003 \rightarrow 0.03 \rightarrow 0.3 \rightarrow 3$  이런식으로  $\alpha$  값을 10배씩 늘려나가면서, 빠르게 수렴하는  $\alpha$ 를 찾는다.

### 선형회귀

선형회귀에서 learning rate가 적절히 작을 때,  $J(\theta)$ 는 매 반복마다 감소한다.

### IN PYTHON

tensorflow의 GradientDescentOptimizer 이용

```
x_data = [1,2,3,4,5,6]
y_data = [2,4,6,8,10,12]
```

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
w = tf.Variable(tf.random_normal([1], seed = 0, name =
    'weight'))
b = tf.Variable(tf.random_normal([1], seed= 0, name =
    'bias'))
```

```
hypothesis = w*x + b
```

```

cost = tf.reduce_mean(tf.square(hypothesis - y))
optimizer =
    tf.train.GradientDescentOptimizer(learning_rate =
        0.001)
train = optimizer.minimize(cost)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(2001):
    cost_v, w_v, b_v, _ = sess.run([cost, w, b, train],
                                    feed_dict = {
                                        x : x_data,
                                        y: y_data
                                    })
    if step % 20 == 0:
        print(step, cost_v, w_v, b_v)

sess.run(hypothesis, feed_dict = { x:7})

```

### 고급 최적화

경사하강법보다 더 좋은 최적화 방법이 있는데 이는 다음과 같다.

- Conjugate gradient
- BFGS
- L-BFGS

### 이점

- $\alpha$ 를 수동으로 pick할 필요가 없다(자동으로 적합한  $\alpha$ 가 선택된다)
- gradient descent보다 빠르다.

### 단점

- 복잡하다

## 하이퍼파라미터 (hyperparameter)

모델의 parameter 값을 측정하기 위해 알고리즘 구현 과정에서 사용 모델에서 외적인 요소이다.

하이퍼파라미터는 주로 알고리즘 사용자에 의해 정해진다.

경험에 의해 정해지기도 한다. 즉, 알고리즘을 여러 번 수행해보면서 최적의 값을 직감적으로 알게된다.

예측 알고리즘 모델링의 문제점을 위해 조절된다.

## 아핀 결합(Affine combination)

유한 개의 벡터들로 선형결합(Linear combination)을 할 때, 모든 계수 (coefficient)들의 합이 1인 경우를 Affine combination이라고 한다. r개의 벡터  $x_1, \dots, x_r$ 에 대해서 Affine combination은 다음과 같이 표현된다.

$$\lambda_1x_1 + \dots + \lambda_rx_r$$

여기서 실수  $\lambda_i$ 는  $\lambda_1 + \dots + \lambda_r = 1$ 을 만족한다.

유클리드 공간(Euclidean space)에서 임의의 두 점을 선택했을 때, 두 점을 지나는 직선 상의 모든 점들은 Affine combination으로 표현할 수 있다.

### Affine layer

인접하는 계층의 모든 뉴런과 결합되어 있는 layer이다. (fully connected)

### 텐서플로우(tensorflow library)

구글이 오픈 소스로 공개한 머신러닝(machine learning) 라이브러리

다차원 행렬계산(tensor), 대규모 숫자계산

빅데이터 처리를 위한 병렬 컴퓨팅 지원을 한다.

C 계열의 언어에 좋다.

#### 이름 해석

Tensor를 흘려보내면서 Flow 데이터를 처리한다.

여기서 Tensor는 다차원 배열을 의미한다.

#### import

```
import tensorflow as tf
```

#### 계산 그래프(computational graph)

계산 그래프란 프로그램의 계산 구조를 정의하는 방향성 그래프(directional graph)를 말한다.

계산 그래프는 노드(node)와 엣지(edge)로 구성된다.

텐서플로우에서 노드는 오퍼레이션(OP) 이라고 부른다.

그리고 실제 값을 전달하는 것이 엣지이다.

엣지는 화살표로 표현된다.

텐서플로우의 계산 그래프는 엣지를 따라 텐서(tensor)라는 형태로 값이 다른 오퍼레이션으로 이동한다고 말한다.

그래서 언어이름이 텐서(tensor) 흐름(flow) 이다.

텐서는 데이터를 표현하는 기본 단위로 다차원(multi-dimensional) 행렬을 의미한다.

#### 국소적 계산

계산 그래프의 특징은 ‘국소적 계산’을 전파함으로써 최종 결과를 얻는다는 점이다.

국소적이란 ‘자신과 직접 관계된 작은 범위’라는 뜻이다.

국소적 계산은 결국 전체에서 어떤 일이 벌어지든 상관없이 자신과 관계된 정보만으로 결과를 출력할 수 있다는 것이다.

국소적 계산은 단순하지만 그 결과를 전달함으로써 전체를 구성하는 복잡한 계산을 해낼수 있다.

#### 계산 그래프로 푸는 이유?

각 노드에서 단순한 계산에 집중하여 문제를 단순화 할수 있다.

계산 그래프는 중간 계산 결과를 모두 보관할수 있다.

#### 버전

```
tf.__version__
```

#### 상수선언

```
tf.constant(value, dtype = None, shape = None, name =  
           'Const', verify_shape = False)  
ex) hello = tf.constant('hello')
```

### value

상수의 값이다.

### dtype

상수의 데이터 형이다.

### shape

행렬의 차원을 정의한다.

### name

상수의 이름을 정의한다.

ex) `tf.constant(120, name='x1')`

### 배열

ex) `x1 = tf.constant([[1,2], [3,4]]) # shape(2,2)`

## 변수선언

```
tf.Variable.__init__(initial_value=None, trainable=True,  
collections=None, validate_shape=True,  
caching_device=None, name=None, variable_def=None,  
dtype=None, expected_shape=None, import_scope=None)  
ex) v = tf.Variable(0, name='v')
```

### 값할당

ex) `assign_op = tf.assign(v, calc_op)`

### 변수에 초기값을 넣는 코드

텐서플로우에서 그래프를 실행하기 전에 초기화를 해줘야 그 값이 변수에 저장이 된다.

세션을 초기화 하는 순간 변수에 값이 지정되는데, 초기화 하는 방법은 변수들을 `global_variables_initializer()`를 이용해서 초기화한 후,

초기화된 결과를 세션에 전달해주는 것이다.

`sess.run(tf.global_variables_initializer())`

## placeholder

형태를 미리 만들어놓고, runtime에서 값을 할당하고 싶을때 사용

`Variable`보다 더 간단한 형태의 기본 구조이다.

`placeholder`은 단순히 나중에 데이터를 할당하기 위한 변수이다.

이는 데이터 없이도, 연산을 생성하고, 계산 그래프를 빌드할수 있게해준다.

우리는 `placeholder`을 통해 `graph`에 데이터를 공급한다.

`tf.placeholder([형식])`

`tf.placeholder(dtype, shape, name)`

ex)

`p1 = tf.placeholder('int32')`

`p2 = tf.placeholder('int32')`

`y = tf.add(p1, p2)`

`sess.run(y, feed_dict = {p1:10, p2:3})`

### dtype

플레이스 홀더에 저장되는 데이터형이다.

### shape

행렬의 차원을 정의한다.

### name

플레이스 홀더의 이름을 정의한다.

### placeholder 배열(무한)

```
tf.placeholder( [형식] )
```

ex)

```
x = tf.placeholder('float32')  
y = tf.placeholder('float32')  
z = tf.multiply(x,y)
```

```
print(sess.run(z, feed_dict = {x: [3,4], y: [5,6]}))
```

### placeholder 배열(shape 지정)

```
tf.placeholder( [형식], shape = () )
```

ex)

```
x = tf.placeholder(tf.int32, shape = (2,3))  
y = tf.placeholder(tf.int32, shape = (3,2))  
z = tf.matmul(x,y)
```

```
sess.run(z, feed_dict = {  
    x: [[1,2,3], [4,5,6]],  
    y: [[1,2], [3,4], [5,6]]  
})
```

### 공간 지정 배열

ex)

```
a = tf.placeholder(tf.int32, [3]) <- 공간 3개짜리 배열을 생성
```

```
sess.run(x2_op, feed_dict= {a:[10,20,30]}) <- 배열 값 할당
```

### feeding

데이터를 넣는것

sess.run( x \* 2, feed\_dict = { x: input\_data } )와 같이 세션을 통해서 그래프를 실행할때,

feed\_dict 변수를 이용해서 플레이스 홀더에 데이터를 피딩한다.

```
ex) result = sess.run(y,feed_dict={x:input_data})
```

### 세션

텐서플로우는 세션을 기준으로 성능을 향상시키기위해 정의와 실행을 분리하였다.

파이썬은 다른 프로그램 언어에 비해 실행속도가 느린 단점이 있다.

이를 극복하기 위해 외부에서 연산을 수행하는 numpy와 같은 라이브러리를 사용 하지만 외부에서 연산한 값이 파이썬으로 전환될 때 오버헤드가 발생하는 문제가 생긴다.

텐서플로우는 이런 오버헤드를 피하기 위해 세션 이전에는 파이썬에서 계산 그래프를 그리고 세션을 통해 계산 그래프를 CPU와 GPU가 처리할 수 있는 다른 언어로 변환하여 연산하는 방법으로 속도문제를 극복했다.

```
tf.Session()
```

```
ex) sess = tf.Session()
    run()
    해당 세션에서 표현식 실행
    ex) sess.run(tf.constant('hello'))

    tf.InteractiveSession()
    대화형 처리
    ex) sess = tf.InteractiveSession()
        conv2d = tf.nn.conv2d(image2, filter,
                             strides=[1,1,1,1], padding='VALID')
        conv2d_img = conv2d.eval()

수학 함수
    tf.add()
    덧셈
    tf.subtract()
    뺄셈
    tf.multiply()
    곱셈
    tf.div()
    나눗셈의 몫, 소수점
    tf.truediv()
    나눗셈의 몫, 소수점?
    tf.mod()
    나눗셈의 나머지
    tf.negative()
    음수를 리턴
    tf.sign()
    부호를 리턴
    tf.reciprocal()
    역수를 리턴
    tf.square()
    제곱을 계산
    tf.round()
    반올림
    tf.sqrt()
    제곱근
    tf.pow()
    거듭제곱
    tf.exp()
    지수값
    tf.log()
    로그값
    tf.maximum()
    최대값
    tf.minimum()
    최소값
    tf.cos()
    코사인 함수
    tf.sin()
```

사인 함수  
**tf.matmul()**  
행렬 곱셈  
ex) `tf.matmul(tf.constant([[1,2,3]]),  
 tf.constant([[2], [2], [2]]))`

**tf.reduce\_mean()**  
평균

함수  
**tf.expand\_dims()**  
tensor의 차원을 확장  
ex)  
`tf.expand_dims(tf.constant([[1,2,3], [4,5,6]]), 0)`

**tf.zeros\_like()**  
특정 행렬의 shape에 맞게 영행렬을 만듦  
ex)  
`tf.zeros_like([[10, 20, 30], [1,2,3]],  
 dtype = tf.int32,  
 name = 'zeros')`

**tf.zeros()**  
shape에 맞게 영행렬을 만듦  
ex)  
`tf.zeros([2,3],  
 dtype = tf.int32,  
 name = 'zeros1')`

**tf.ones()**  
shape에 맞게 1로 채워진 행렬을 만듦  
ex) `ones1 = tf.ones([2,3], dtype=tf.int32,  
 name='ones1')`

**tf.fill()**  
shape에 맞게 특정 값으로 채워진 행렬을 만듦  
ex) `tf.fill([2,3], 7)`

**tf.random\_normal()**  
shape에 맞게 정규분포에서 나온 난수값을 이용해 행렬을 만듦  
ex) `tf.random_normal([3,3])`  
표준 정규분포 난수  
`tf.random_normal([3,3], mean = 0, stddev = 1)`

**tf.shuffle()**  
주어진 값들을 shuffle  
ex) `tf.random_shuffle(tf.constant([[1,2], [3,4],  
 [5,6]]))`

**tf.random\_uniform()**  
어떤 값의 범위 안에서 값을 뽑아내 shape에 맞게 행렬을 구성  
ex) `tf.random_uniform([2,3], minval=1, maxval=3) : 2`  
행, 3열의 행렬을 숫자 1~3 범위내에서 값을 뽑아 구성  
seed  
seed 지정가능  
ex) `tf.random_uniform([1], seed = 1)`

**tf.set\_random\_seed()**

전체 seed 설정

**tf.linspace()**

시퀀스를 표현, 단 실수형으로만 해야 한다.

**tf.linspace(시작점, 끝점, 표현할 개수)**

ex) `tf.linspace(10., 12., 3, name='linspace')`

**tf.range()**

시퀀스를 표현

**tf.range(start = [시작값, 기본: 0], limit = [끝값], delta = [변화값, 기본: 1])** // limit 지정시 limit 전까지의 값 출력

ex) `tf.range(start=1, limit=10, delta=1)`

**tf.one\_hot()**

one-hot tensor를 반환한다.

특정한 데이터를 one-hot encoding 한다.

one-hot 인코딩은 정수형이어야 한다.

ex)

`y = tf.placeholder(tf.int32, shape = [None, 1])`

`y_one_hot = tf.one_hot(y, 3)`

`tf.one_hot(y_data, depth = 3).eval(session=sess)`

선언

```
tf.one_hot(  
    indices,  
    depth,  
    on_value = None,  
    off_value = None,  
    axis = None,  
    dtype = None,  
    name = None  
)
```

인덱스에 해당하는 위치에 on\_value 그렇지 않으면 off\_value를 표현한다.

on\_value 및 off\_value에는 일치하는 데이터 유형이 있어야한다.

dtype이 제공될 경우 dtype에 의해 지정된 것과 동일한 데이터 형식이어야 한다.

on\_value가 제공되지 않으면 기본적으로 1로 설정

off\_value가 제공되지 않는 경우 기본적으로 0으로 설정

입력 indices의 rank가 N이면, 출력 rank는 N+1이 된다. (새로운 축이 끝에서 생성됨)

**depth**

차원의 깊이를 정의하는 스칼라(scalar)

**on\_value**

indices[j] = i 일 때, 출력을 채울 값을 정의하는 스칼라

**off\_value**

indices[j] != i 일 때, 출력을 채울 값을 정의하는 스칼라

**dtype**

출력 tensor의 data type

**axis**

값을 채울 축

**name**

연산에 대한 이름

**tf.softmax()**

소프트 맥스 함수

ex)

`logits = tf.matmul(x, w) + b`

`hypothesis = tf.nn.softmax(logits)`

**tf.argmax()**

배열에서 가장 큰 값을 찾아 인덱스를 리턴

`tf.argmax(a, 0)` : 1차원 배열의 가장 큰 값을 찾아 인덱스 반환

`tf.argmax(a, 1)`: 2차원 배열의 가장 큰 값을 찾아 인덱스를 반환

**tf.nn.conv2d()**

`tf.nn.conv2d(input, filter, strides, padding,`

`use_cudnn_on_gpu=None, data_format=None,`  
`name=None)`

**input**

[batch, in\_height, in\_width, in\_channels] 형식

**filter**

[filter\_height, filter\_width, in\_channels,  
out\_channels] 형식

ex)

3x3x1 필터를 32개 만드는 것을 코드로 표현하면 [3,3,1,32]가 된다.

순서대로 너비(3), 높이(3), 입력채널(1), 출력채널(32)을 뜻한다.

출력 채널을 부르는 용어는 activation map이다.

**strides**

크기 4인 1차원 리스트, [0], [3]은 반드시 1

**padding**

'SAME' 또는 'VALID'

패딩을 추가하는 공식

SAME은 출력 크기를 입력과 같게 유지

**tf.nn.max\_pool()**

`tf.nn.max_pool(value, ksize, strides, padding,`  
`data_format='NHWC', name=None)`

**value**

[batch, height, width, channels] 형식의 입력 데이터.

ReLU를 통과한 출력 결과가 된다.

**ksize**

4개 이상의 크기를 갖는 리스트로 입력 데이터의 각 차원의 윈도우 크기

ex)

ksize가 [1,2,2,1] 이라는 뜻은 2칸씩 이동하면서 출력 결과를 1개 만들어 낸다는 것이다.

다시 말해, 4개의 데이터 중에서 가장 큰 1개를 반환하는 역할을 한다.

**data\_format**

NHWC 또는 NCHW. n-count, height, width, channels의 약자 사용

**padding**

```

    패딩을 추가하는 옵션
    'SAME' , 'VALID'
tf.nn.relu()
tf.nn.relu(features, name = None)
features의 타입과 같은 Tensor를 반환한다.
    features
Tensor이다.
다음과 같은 타입을 따라야 한다.
    float32, float64, int32, uint8, int16, int8,
    int64, bfloat16, uint16, half, uint32, uint64,
    qint8
tf.nn.dropout()
메소드
get_shape()
tensor의 shape를 반환
ex) tf.constant([[2], [2], [2]]).shape

```

### 변환

```

tf.convert_to_tensor()
주어진 value를 Tensor로 변환
이 함수는 여러가지 Python Object를 Tensor Object로 변환한다.
    (numpy array, list, scalar)
tf.convert_to_tensor(
    value,
    dtype=None,
    dtype_hint=None,
    name=None
)
value
Tensor 변환 함수에 등록된 변환되어질 객체
dtype
tensor 반환을 위한 선택 요소. 만약 지정하지 않는다면 value를
    통해 타입을 추론한다.
dtype_hint
선택 요소이고, dtype이 None일때 사용된다.
name
Tensor가 생성될때의 이름(선택요소)

```

### 다변수 선형 회귀(Multivariate Linear Regression)

$x_{-j}^i(i)$

여기서  $j = \text{column index}$ ,  $i = \text{row index}$ ,  $x = \text{입력 features}$

가설 표현

$\theta$ 와  $X$ 가 열벡터로 표현된다고 했을 때..

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

위 식에서  $x_0$ 를 1이라고 놓고 보면,

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

로 표현 할 수 있고,  
이는 행렬을 이용해서

$$h_{\theta}(x) = \theta' X$$

으로 표현할 수 있음 (여기서  $\theta$ 와  $X$ 는  $n+1$  차원 벡터)

비용함수 표현

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^m (h_\theta(x^i) - y^i)^2$$

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^m (\theta' x^{(i)} - y^{(i)})^2$$

파라미터 업데이트 rule

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

비용함수를 미분하면 다음과 같이 됨

$$\theta_j := \theta_j - \alpha \frac{\sum_{i=0}^m (h_\theta(X^{(i)}) - y^{(i)})}{m}$$

### 다항 회귀(Polynomial regression)

feature 선택

문제를 어떻게 이해하느냐에 따라 새로운 feature로 더 나은 모델을 정의 할 수 있다.

ex)

집의 밑변과 높이라는 feature가 있다고 했을때, 집의 가격을 알아본다고 하면,  $h(\theta) = \theta_0 + \theta_1 * \text{밑변} + \theta_2 * \text{높이}$  라고 회귀식을 세우는것보다

$h(\theta) = \theta_0 + \theta_1(\text{밑변} * \text{높이})$  이렇게 회귀식을 세우는게 낫다.

모델 종류

모델은 선형모델만 있는게 아니라, 방정식의 차수가 2차인것도 있고, 제곱근을 사용한 모델도 있고, 3차도 있고 다양하다

모델 표현(차수 제거)

$h(\theta) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3$  이라는 모델이 있다고 했을때

치환을 통해 차수를 제거할수 있다.

$h(\theta) = \theta_0 + \theta_1(x_1) + \theta_2(x_2) + \theta_3(x_3)$   
 그런데 이때,  $x_1 = \text{size}$ ,  $x_2 = \text{size}^2$ ,  $x_3 = \text{size}^3$  이런식으로 되므로, 각  
 feature의 scale이 차이가 나므로,  
 feature scaling을 통해 scale을 조정해주면 좋다.

### 소프트맥스(softmax)

$k$  개의 값이 존재할 때 각각의 값의 편차를 확대시켜 큰 값은 상대적으로 더 크게, 작  
 은 값은 상대적으로 더 적게 만든 다음에 normalization 시키는 함수이다.

$$p_j = \frac{e^{x_j}}{\sum_{k=\text{first}}^{\text{all}} e^{x_k}} = \frac{e^{x_j}}{e^{x_1} + e^{x_2} + \dots + e^{x_k}}$$

분자에 있는 지수함수가 각각의 값의 편차를 확대시키는 역할을 한다.

지수함수 그래프를 생각해보면 입력값이 커질수록 기하급수적으로 출력값이 커짐을 알수  
 있다.

따라서  $K$ 개값 중에서 큰 값이었던 것은 상대적으로 확실히 더 커지게 되고, 작은 값이  
 었던 것은 상대적으로 작아지게 된다.

분모에 모든  $K$ 값의 지수함수값을 더했기 때문에 상대성을 갖게 된다.

softmax를 통해서  $K$ 개의 값들은 상대적인 중요도를 나타내는  $K$ 개의 값으로 재탄생되  
 고 그 값들의 총합은 1이 된다.

이러한 특성이 있기 때문에 딥러닝의 마지막 출력단에서 사용된다.

fully-connected layer의 마지막 층의 출력값에 softmax를 적용하면 그 값들은  
 클래스 확률로 변모한다.

어떤 클래스에 속할 확률을 나타내는 것이다.

softmax 함수를 거친  $K$ 개의 합은 1이 된다.

$$p_1 + p_2 + \dots + p_K = 1$$

### Normalization function

Softmax 함수는 Normalization 함수로써 출력값들의 합을 1로 만들어준다.

### softmax 유도(Bayesian 이용)

$k$  클래스에 대한 Likelihood와 Prior의 곱을 다음과 같이 정의한다.

$$a_k = \ln(P(X|Y_k)P(Y_k))$$

Class 1에 대한 사후 확률은 Bayes로 전개해보면 다음과 같다.

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2) + \dots + P(X|Y_k)P(Y_k)}$$

여기에 위에 정의한것을 대입해보면

$$P(Y_1|X) = \frac{e^{a_1}}{e^{a_1} + e^{a_2} + \dots + e^{a_k}} = \frac{e^{a_1}}{\sum_j e^{a_j}}$$

결국 이것이 Deep Learning에서 많이 보던 softmax function이 된다.

저 Normalization Term(분모) 때문에 한 Class의 확률이 다른 Class에 영향을 미치게 되며, Generative Model(Naive Bayes나 Gaussian Discriminant Analysis 등)에서는 Prior P(Y)에도 영향을 주게 된다.

이것을 다음 함수

$$P(Y|X) = \frac{e^{\theta_1 x}}{\sum_j e^{\theta_j x_j}}$$

Prior 고려 없이 바로  $P(Y|X)$ 를 fitting하면 Discriminant Model인 이것은 Softmax Regression이 된다.

**logit, sigmoid, softmax 관계**

logit과 sigmoid는 역함수 관계이다. (시그모이드를 역함수로 만들어보면 logit식이 나타남)

softmax 함수는 sigmoid의 일반형이다.

유도..

오른쪽 식은 클래스가 2개일 때 odds를 표현한 것이고, 왼쪽식은 클래스가 K개일 때 odds를 표현해놓은 것이다.

이 빨간색 테두리친 식은 서로 같다.

log(Odds)를 t라고 놓으면

Odds =  $e^t$  임

$$\boxed{\frac{P(C_i|\mathbf{x})}{P(C_K|\mathbf{x})}} = \exp(t_i)$$

$$\boxed{\frac{y}{1-y}} = \exp(t)$$

양변을  $i=1$ 부터  $K-1$ 까지 더해준다.

$$\sum_{i=1}^{K-1} \frac{P(C_i|\mathbf{x})}{P(C_K|\mathbf{x})} = \sum_{i=1}^{K-1} \exp(t_i)$$

분모에 있는  $C_K$ 는  $i$ 의 영향을 받지 않으므로 분자의  $P(C_i|\mathbf{x})$ 만 더해진다.

확률의 합은 1이기 때문에, 1부터  $K-1$ 번째 클래스의 확률을 더한 값은 1에서  $K$ 번째 클래스의 확률을 뺀 것과 같다.

$$\frac{1 - P(C_K | \mathbf{x})}{P(C_K | \mathbf{x})} = \sum_{i=1}^{K-1} \exp(t_i)$$

이를  $P(C_K | \mathbf{x})$ 를 기준으로 정리하면 다음과 같다.

$$P(C_K | \mathbf{x}) = \frac{1}{1 + \sum_{i=1}^{K-1} \exp(t_i)}$$

이는 맨처음에 봤던 빨간박스쳐진 식을 다르게 표현한 것이고

$$P(C_i | \mathbf{x}) = \exp(t_i) \cdot P(C_K | \mathbf{x})$$

$P(C_K | \mathbf{x})$ 를 위 식에 대입해준다.

그리고 분모에 있는 1이  $\exp(t_K)$ 로 바뀌었는데 왜냐하면

$$\frac{\exp(t_i)}{1 + \sum_{j=1}^{K-1} \exp(t_j)} = \frac{\exp(t_i)}{\exp(t_K) + \sum_{j=1}^{K-1} \exp(t_j)}$$

아까봤던 식에서 i대신에 K개 들어가면 1이기 때문이다.

$$\boxed{\frac{P(C_i | \mathbf{x})}{P(C_K | \mathbf{x})} = \exp(t_i)} \quad \left( \because 1 = \frac{P(C_K | \mathbf{x})}{P(C_K | \mathbf{x})} = \exp(t_K) \right)$$

마지막으로  $\exp(t_K)$ 를 시그마 기호 안으로 집어 넣으면 softmax 함수가 완성된다.

$$\boxed{\frac{\exp(t_i)}{\sum_{j=1}^K \exp(t_j)} = \text{softmax}(t_i)}$$

### 크로스 엔트로피 (Cross entropy)

정보 (information)

degree of surprise이다. – 놀람의 정도

$h(x) = -\log P(x)$

ex)

브라질이 이기는 경우:  $-\log P(x) = -\log(0.99) = 0.01$

중국이 이기는 경우:  $-\log P(x) = -\log(0.01) = 4.6$

즉, 중국이 이기는 경우가 브라질이 이기는 경우보다 460배 '놀랍다'고 본다.

## 엔트로피(entropy)

정보량(놀람의 정도)의 평균을 의미한다.

ex)

브라질 vs 중국 의 경우 entropy는 다음과 같이 계산

$$0.99 * -\text{np.log}(0.99) + 0.01 * -\text{np.log}(0.01)$$

브라질 vs 아르헨티나 의 entropy는 다음과 같이 계산

$$0.5 * -\text{np.log}(0.5) + 0.5 * -\text{np.log}(0.5)$$

평균적으로 볼때 브라질 vs 중국 보다 브라질 vs 아르헨티나 경기를 보고 더 놀란다는 결론을 얻는다.

$$\text{entropy} = E(-\log(P(x)))$$

KL-divergence(Kullback-Leibler Divergence, KLD)의 줄임말이다.

relative entropy, 즉 상대적인 entropy를 의미한다.

추론확률과 실제확률에 대한 엔트로피의 차를 구한것이다.

$Q(x)$ 는 예측,  $P(x)$ 는 실제 확률이라고 하면,

- KL-divergence = relative entropy =  $E(-\log(Q(x))) - E(-\log(P(x)))$

- 개념상 두 확률 분포 사이의 distance라고 볼수 있다. (하지만 실제 distance가 아니다)

KLD 값은 항상 0보다 크거나 같으며 예측한 확률이 실제와 같아질 수록 0에 가까워 진다.

즉, error(~ 비용, cost, loss)의 측정 기준이 될수있다.

### 해석

쿨백-라이블러 발산은 두 확률분포의 차이를 계산하는 데 사용하는 함수이다.

딥러닝 모델을 만들 때 우리가 가지고 있는 데이터의 분포  $P(x)$ 와 모델이 추정한 데이터의 분포  $Q(x)$  간에 차이를 KLD를 활용해 구할수 있다.

$$D_{KL}(P||Q) = E_{X \sim P} \left[ \log \frac{P(x)}{Q(x)} \right] = E_{X \sim P} \left[ -\log \frac{Q(x)}{P(x)} \right]$$

$P$ 와  $Q$ 가 동일한 확률분포일 경우 KLD는 정의에 따라 그 값이 0이 된다.

하지만 KLD는 비대칭(not symmetric)이므로  $P$ 와  $Q$ 의 위치가 바뀌면 KLD 값도 달라진다.

따라서 KLD는 거리함수로는 사용할수 없다.

## cross entropy

그런데 우리는 신이 아니므로 브라질 vs 아르헨에서 실제로 누가 이길지를 미리 알 수 없다.

바꿔말하면,  $P(x)$ 를 모르기 때문에 KL-divergence를 minimize 하려면,  $E(-\log(Q(x)))$ 를 minimize 해야 한다.

이때  $E(-\log(Q(x)))$ 를 cross entropy라 부른다.

그런데  $E(-\log(Q(x)))$ 를 구할때 각  $Q(x)$ 에 대한 가중치로  $P(x)$ 가 필요한데, 이때  $P(x)$ 는 우리가 가진 데이터의 비율을 사용한다. (empirical

distribution)

예를 들어 위의 측구 경기에서 cross entropy는 다음과 같다.

$$\begin{aligned} E(-\log(Q(x))) &= -\sum P(x) * \log(Q(x)) \\ &= - (P(\text{브라질이 이긴 확률}) * \log(Q(\text{브라질이 이길 확률})) + P(\text{아르헨티나가 이긴 확률}) * \log(Q(\text{아르헨티나가 이길 확률})) ) \end{aligned}$$

만약 브라질이 아르헨티나를 이겼다는 데이터를 가지고 있다면,

$$\text{cross entropy} = - (1 * \log(Q(\text{브라질이 이길 확률})) + 0 *$$

$$\log(Q(\text{아르헨티나가 이길 확률})) ) = - \log(Q(\text{브라질이 이길 확률}))$$

즉, log likelihood에 -1 을 곱해준 값 (negative log likelihood)  
과 동일해진다.

식

$$H(P, Q) = E_{X \sim P} [-\log Q(x)] = - \sum_x P(x) \log Q(x)$$

이산변수에 대한 크로스엔트로피는 위와 같이 정의된다.

로그 바깥에 곱해지는 확률이  $P(x)$ 이고 로그 안에 들어가는 것이  $Q(x)$ 이다.

엔트로피는 엔트로피이되, 두 확률 분포가 교차로 곱해진다는 의미로 cross라는 이름  
이 붙었다.

**cross entropy와 KLD**

KLD를 전개하면 다음과 같다.

$$\begin{aligned} D_{KL}(P||Q) &= - \sum_x P(x) \log \left( \frac{Q(x)}{P(x)} \right) \\ &= - \sum_x P(x) \{\log Q(x) - \log P(x)\} \\ &= - \sum_x \{P(x) \log Q(x) - P(x) \log P(x)\} \\ &= - \sum_x P(x) \log Q(x) + \sum_x P(x) \log P(x) \\ &= H(P, Q) - H(P) \end{aligned}$$

이를 크로스 엔트로피 기준으로 바꾸면 다음과 같다.

$$H(P, Q) = H(P) + D_{KL}(P||Q)$$

$H(P)$ 는 바뀌지 않기 때문에, 크로스 엔트로피를 최소화 한다는 것은 KLD를 최소화 한다는  
것과 같다.

즉, 두 확률분포가 최대한 일치하게 하는 것이다.

### 선형 회귀(linear regression)

Gradient descent 혹은 정규방정식을 사용할수 있음

#### 정규방정식(normal equation)

정규방정식은 통계학에서 선형 회귀상에서 알지못하는 값(parameter)를 예측하기 위한 방법론이

$\theta$ 를 분석적으로 구하는 방법이다. (한번의 식으로 구할수 있음)

정규방정식에 따라  $\theta$ 를 구하면 cost function이 최소화된 값을 구할수 있다.

$\theta$ 가  $n+1$  차원의 벡터라면

비용함수가 있을때,  $J(\theta_0, \dots, \theta_m)$

비용함수를 각  $\theta_j$ 로 편미분 했을때,  $\theta_0$ 이 되는 값을 통해 cost function이 최소화된 값을 구할수 있다.

이를 행렬로 표현해보면

$$\theta = (X'X)^{-1} X'y$$

로 표현할수 있다. (X는 designed matrix, y는 종속변수 벡터)

#### designed Matrix

sample data가 다음과 같이 있을때,

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

designed matrix를 다음과 같이 구성할수 있다.

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

여기에 앞에 1을 붙여서 이렇게 만든다.

$$X = \begin{bmatrix} 1 & x_1^{(1)'} & \dots \\ 1 & x_1^{(2)'} & \dots \\ \dots & \dots & \dots \\ 1 & x_1^{(m)'} & \dots \end{bmatrix}$$

여기서 1은  $x_0$  값 (편향과 곱해질 수)

### 정규방정식과 feature scaling

정규방정식을 사용할 경우 feature scaling이 필요하지 않다.

### 경사하강법과의 비교

$n$  = feature의 개수

$n$ 이 10000을 넘어가면 gradient descent가 낫다.

### Gradient Descent

- 학습률을 선택해야 함
- 많은 반복이 필요

- $n$ 이 클때 잘 돌아간다.

### Normal Equation

- 학습률을 선택하지 않아도
- 반복이 필요 없다
- $(X'X)^{-1}$  를 계산해야한다.
- 저 행렬을 계산해야하므로  $n$ 이 클때 느려진다.  
(행렬의 역행렬을 계산하는데 걸리는 시간은 대략  $O(n^3)$  만큼 증가)

$X'X$ 의 역행렬이 존재하지 않는 경우(singular matrix의 경우)

$X'X$ 의 역행렬은 대부분 존재한다.

그러나 존재하지 않는 경우 다음을 체크한다.

불필요한 feature를 가진 경우(feature가 종속관계에 있는 경우)

ex)

$x_1 = \text{size in feet}$

$x_2 = \text{size in m}^2$

$(x_1 = (3.28)^2 \times 2)$  이므로 종속관계여서 역행렬 존재하지 않음

너무 많은 feature를 가진 경우

너무 적은 데이터를 가지고 여러 파라미터를 표현하기는 어렵다.

ex)

$m \leq n$

몇몇 feature를 제거하거나 regularization을 한다.

### 계산복잡도

정규방정식은  $(n+1) \times (n+1)$  크기가 되는  $X'X$  의 역행렬을 계산한다. ( $n$ 은 특성수)

역행렬을 계산하는 계산 복잡도는 일반적으로  $O(n^{2.4})$ 에서  $O(n^2)$  사이이다.

다시 말해 특성 수가 두배로 늘어나면 계산 시간이 대략  $2^{2.24} = 5.3$ 에서  $2^3 = 8$  배로 증가한다.

### 분류문제

선형 회귀는 분류문제를 해결하기에는 적합하지 않다.

이진 분류의 경우,

데이터가 존재하고 이에 대한 회귀선  $h_{\theta}(x)$ 가 있을때 특정한 임계값(threshold)이 있으면, (예를 들어  $y = 0.5$  가 임계값이면, 이에 대응하는  $x$ 값도 존재할것이다.)

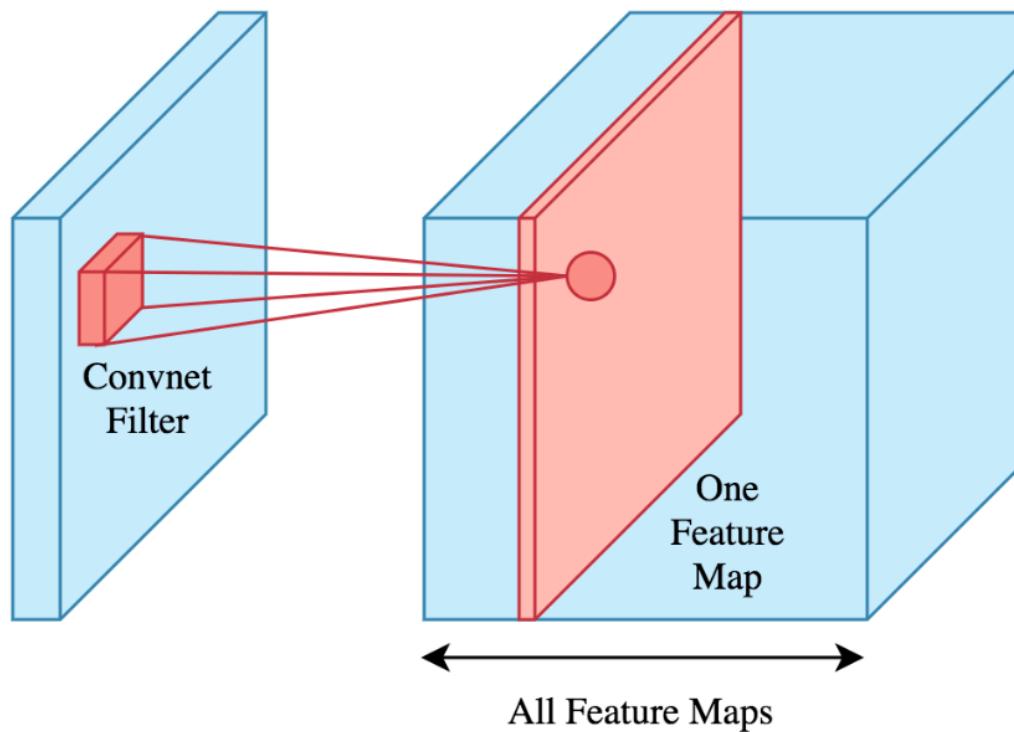
그  $x$ 값을 넘어서는 데이터는 1이고, 넘어서지 않는 데이터는 0이라고 하자.

이때 분류가 잘 되었다고 하더라도,  $x$ 의 scale이 아주 큰값이 들어왔다 치면은, 기울이가 급변하면서 분류가 제대로 되지 않는다.

또한, 선형 회귀는 -무한대~무한대 값을 범위로 갖는데, 이진 분류의 경우 0과 1의 값을 가지므로 1이상의 값을 갖는게 이상해진다.

### (convolution)

#### Standard Convolution



### of parameters

W: input width

H: input height

C: input channel

K: kernel size

M: output channel

이라고 할때,

필터의 크기가 K이고 input channel의 수는 C이므로 하나의 필터가 가지는 크기는  $K^2C$  이다.

하나의 필터에서 나오는 output은 하나의 channel에 해당하니, output을 M개의 채널로 만들어주려면 필터의 개수가 M개 있어야 한다.

따라서 parameter의 수는  $K^2CM$ 개가 된다.

### computational cost

input의 크기가  $H \times W$ 이고 output도  $H \times W$ 라 할때 하나의 필터당 총  $H \times W$  번 연산을 해야 하니 하나의 output channel이 생성된다.

따라서 총 param의 수는  $K^2CM$ 개가 된다. <- All Feature Map의 연산량

-> 패딩과 스트라이드가 결정되어 있을 때 특정 output 크기로 출력을 하려면 필터의 사이즈가 제한될수 밖에 없고, 이를 통해 계산하면

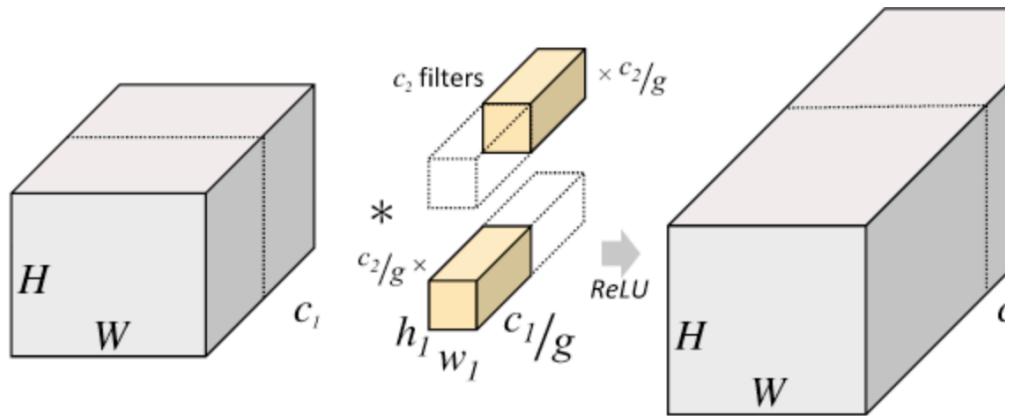
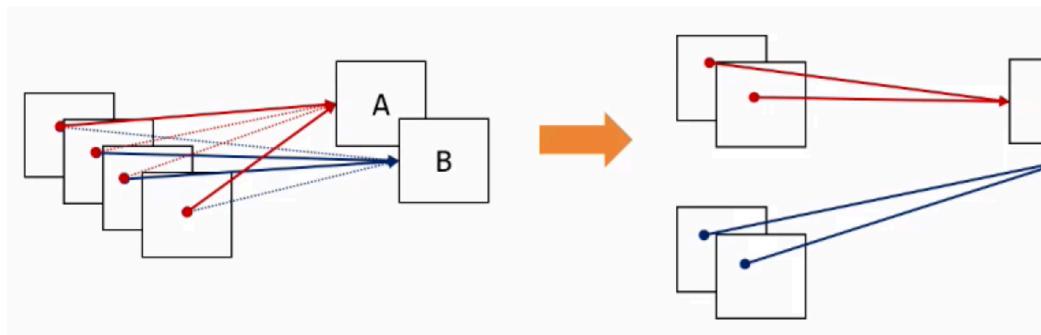
$H \times M$  만큼의 필터 연산이 수행될수밖에 없음

### Grouped Convolution

input의 channel들을 여러개의 group으로 나누어 독립적으로 convolution을 수행하는 방식

그룹지어서 필터를 학습시키겠다.

g: 나눌 group의 수



### 장점

병렬 처리에 유리

기존의 convolution 보다 낮은 param의 수와 연산량을 가진다.

각 그룹에 높은 correlation을 가지는 channel이 학습 될 수 있다.

### of parameters

group convolution은 input channel들을  $g$ 개의 group으로 나누어 연산을 수행한다.

그렇다면 각 그룹당 channel의 수는  $C/g$ 가 된다.

그리고 output channel도 각 그룹에서  $M/g$ 개씩 생성된다고 가정했을 때

그러면 한 그룹당 filter의 param의 수는  $K^2(C/g)M$ 가 된다.

### computational cost

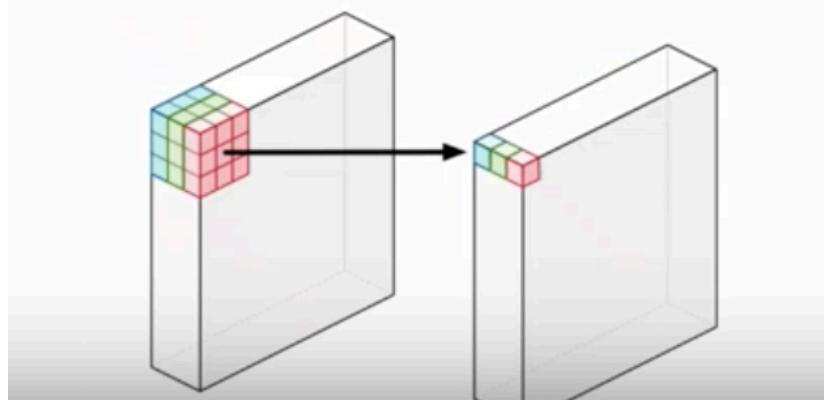
연산량은 standard conv 와 같은 이유로 다음과 같이 된다.

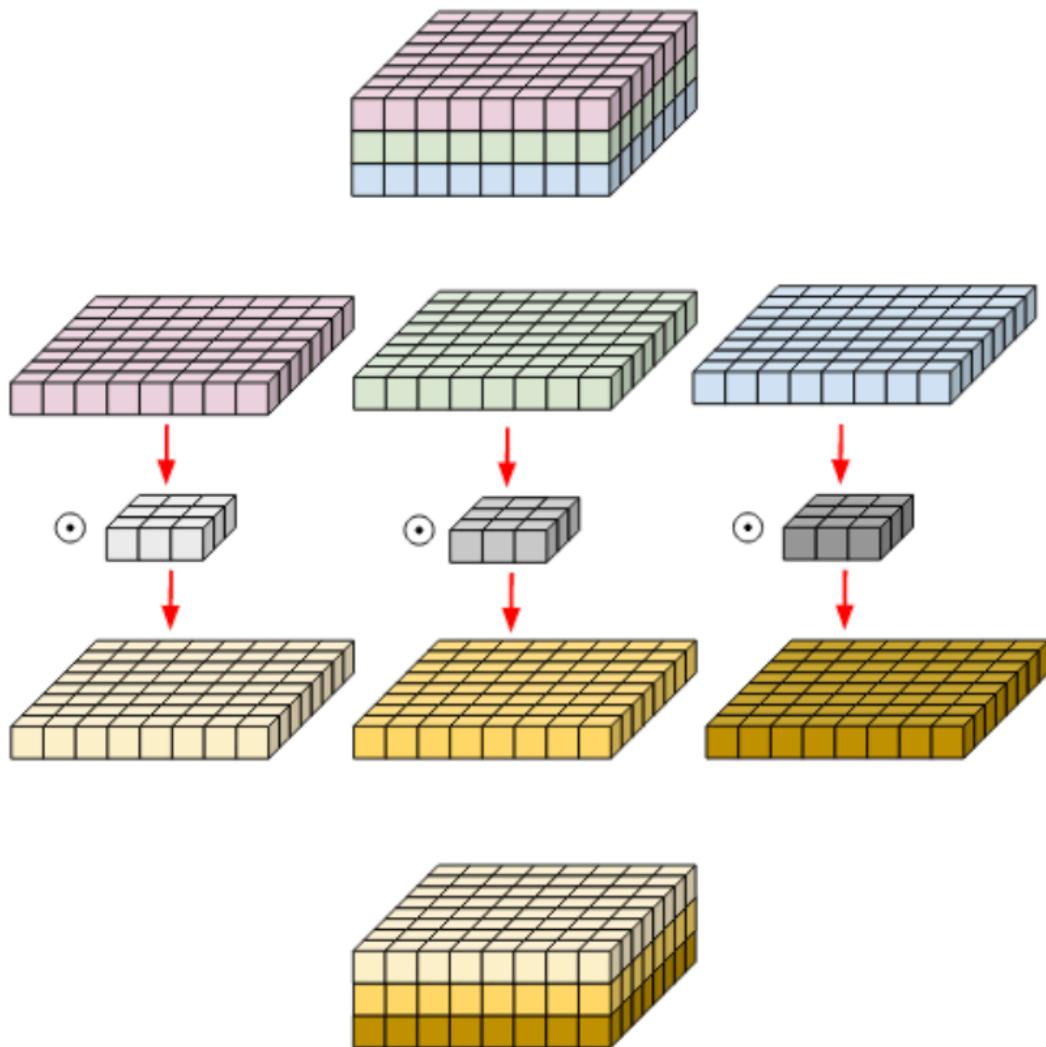
$(K^2CMHW)/g \leftarrow$  전체 Feature Map에서 한 그룹에 대한 연산량

### Depth-wise convolution

# Depth-wise Convolution

Channel-independent Convolution





Depth-wise convolution은 Standard Convolution이 각 채널만의 partial feature를 추출 해내는 것이 불가능하기 때문에 고안해낸 방법이다.

+ 채널마다 따로 학습해주겠다는 방법이다.

Depth-wise conv에서 하려는 것은 각 channel마다의 spatial feature를 추출하는 것이다. 따라서 여기서는 각 channel마다 filter가 존재하게 되고, 이러한 특징때문에 input과 output의 channel이 같게 된다.

결론적으로 위의 Grouped convolution에서  $g$ 를 입력 채널 개수  $C$ 만큼 늘린 경우라고 보면 된다.

#### of parameters

grouped convolution에서 약간 변형만 해주면 된다.( $g$ 를 input channel의 개수  $C$ 만큼 늘린것이기 때문이다)

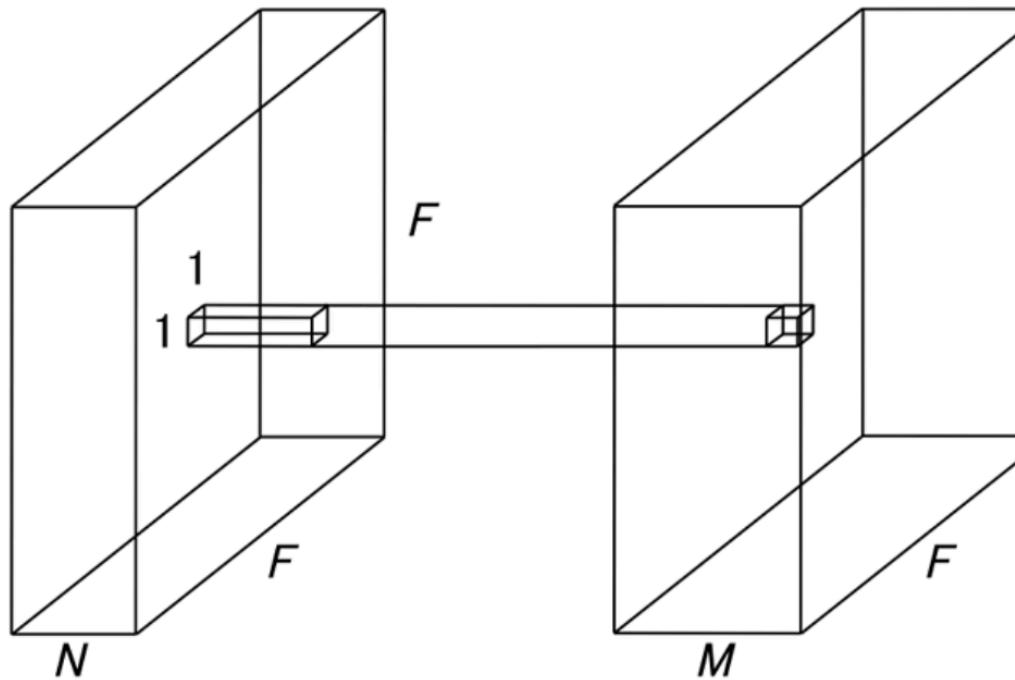
Depth-wise convolution 특성상 input과 output의 channel 개수가 같으므로  $M=C$  가 된다.

따라서 총 param 수는  $K^2C$ 가 된다.

#### computational Cost

grouped에서 치환만해주면 된다.  
 $K^2CHW$ 이다.

**Point-wise Convolution(1x1 convolution)**



Point-wise convolution의 filter의 크기는  $1 \times 1$ 으로 고정되어 있기 때문에  $1 \times 1$  convolution이라고 불린다.

어떻게 보면 depth-wise와는 반대로 spatial feature는 다루지 않고 channel들에 대한 연산만 수행하게 된다.

따라서 output의 크기는 변하지 않고, channel의 수를 조절할 수 있는 역할을 하게 된다. 보통 dimensional reduction을 위해 많이 쓰인다.

이것은 channel의 수를 줄이는 것을 의미하는데, 추후 연산량을 많이 줄여줄 수 있어 중요한 역할을 하게 된다.

#### of parameters

Point-wise convolution의 parameter 수는 standard conv에서  $K=1$ 인 경우와 같다.

따라서 총 param의 수는  $CM$ 이다.

#### computational cost

연산량 또한  $K=1$ 인 경우이기 때문에  $CMHW$ 가 된다.

### Depth-wise separable convolution

depth-wise와 point-wise를 합친 것이다.

#### of parameters

단순히 depth wise conv 후에 point wise conv를 수행하는 방식이므로 두 방식의 param을 더해주면 된다.

따라서 총 param의 수는  $K^2C + CM = C(K^2 + M)$ 이 된다.

#### computational cost

이또한 두가지 작업을 수행하기에 더해주면 된다.

$CHW(K^2 + M)$ 이 된다.

#### standard conv과의 비교

standard conv 연산량:  $K^2CMHW$

Depth-wise separable convolution 연산량:  $CHW(K^2 + M)$

depth wise / standard =  $K^2 + M / K^2M$

$$= 1/m + 1/(k^2)$$

보통 k보다는 m이 훨씬 크기 때문에 M항을 제거하면

연산량에서 약  $k^2$ 배의 차이가 있다. (depth-wise separable convolution의 연산량이  $k^2$ 배만큼 더 적다)

## CNN(Convolution Neural Network)

합성곱 신경망

convolution 층과 pooling 층을 포함하는 신경망

convolution  $\rightarrow$  ReLU  $\rightarrow$  Pooling

### convolution 층

feature map을 만들고 그 feature map을 선명하게 하는 층

#### 합성곱(convolution)?

수학적 의미: 두 함수 중 하나를 반전(reverse), 이동(shift) 시켜가며 나머지 함수와의 곱을 연이어 적분한다.

#### Affine 계층의 문제점

"데이터의 형상이 무너진다."

- 이 형상에는 가까운 픽셀의 값이 비슷하거나, 거리가 먼 픽셀은 관련이 없는 등 소중한 공간적 정보가 담겨있지만

Affine 계층에서는 1차원 데이터로 평탄화 해주어야 하기 때문에 공간적 정보가 날아간다.

#### 합성곱 계층

합성곱 계층은 형상을 유지한다.

이미지도 3차원 데이터로 입력받으며, 마찬가지로 다음 계층에도 3차원 데이터로 전달한다.

#### 특징 맵(feature map)

합성곱 계층의 입출력 데이터를 의미한다.

##### 입력 특징 맵(input feature map)

입력 데이터

##### 출력 특징 맵(output feature map)

출력 데이터

#### 계층 구조

CNN의 계층은 Conv – ReLU – (Pooling) 흐름으로 연결된다.

또, 마지막 출력 계층에서는 Affine–Softmax 조합을 그대로 사용한다.

#### 이미지 필터링

이미지 필터링은 여러 수식을 이용하여 이미지를 이루고 있는 픽셀 행렬을 다른 값으로 바꾸어 이미지를 변형하는 것을 말한다.

#### 합성곱의 연산

합성곱 연산은 이미지 처리에서 말하는 필터연산이다.

이미지 3차원(세로, 가로, 색상) data의 형상을 유지하면서 연산하는 작업

합성곱 연산은 필터의 window를 일정 간격으로 이동해가며 입력 데이터에 적용한다.

입력과 필터에 대응하는 원소끼리 곱한 후 그 총합을 구한다. (이 계산을 단일 곱셈-누산이라 한다.: fused multiply-add, FMA)

그 결과를 출력의 해당 장소에 저장한다.

이 과정을 모든 장소에서 수행하면 합성곱 연산의 출력이 완성된다.

완전 연결 신경망에는 가중치 매개변수와 편향이 존재하는데, CNN에서는 필터의 매개변수가 그동안의 '가중치'에 해당한다.

그리고 CNN에도 편향이 존재한다. (필터와 데이터를 합성곱한후 편향을 더함)

### 3차원 데이터의 합성곱 연산

3차원은 이미지 데이터가 채널 쪽으로 여러개 늘어난다.

채널쪽으로 특징 맵이 여러개 있다면 입력데이터와 필터의 합성곱 연산을 채널마다 수행하고, 그 결과를 더해서 하나의 출력을 얻는다.

3차원 합성곱 연산에서 주의할 점은 입력 데이터의 채널 수와 필터의 채널 수가 같아야 한다는 것이다.

### 블록으로 생각하기

3차원의 합성곱 연산은 데이터와 필터를 직육면체 블록이라고 생각하면 쉽다.

3차원 데이터를 다차원 배열로 나타낼 때는 (channel, height, weight) 순서로 쓴다. (C, H, W)

필터도 같은 순서로 쓴다. (C, FH, FW)

그런데 3차원 합성곱 연산을 하면 출력이 하나가 된다. (채널이 1개)

### 다수의 채널 내보내기

입력데이터에 필터를 FN개 사용하면 출력 맵도 FN개가 생성된다. (필터의 개수를 늘린다)

필터구성 - (FN, OH, OW)

FN개의 맵을 모으면 형상이 (FN, OH, OW)인 블록이 완성된다.

이 완성된 블록을 다음 계층으로 넘기겠다는 것이 CNN의 처리흐름이다.

### 배치처리

신경망 처리에서는 입력 데이터를 한 덩어리로 묶어 배치로 처리했다.

합성곱 연산도 마찬가지로 배치 처리를 지원할수 있다.

그래서 각 계층을 흐르는 데이터의 차원을 하나 늘려 4차원 데이터로 저장한다.

구체적으로 데이터를 (데이터 수, 채널 수, 높이, 너비) 순으로 저장한다. (N, C, H, W)

이렇게 함으로써 신경망에 4차원 데이터가 하나 흐를 때마다 데이터 N개에 대한 합성곱 연산이 이뤄진다.

### 패딩(padding)

합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정값으로 채워 늘리는것을 의미한다.

### 패딩이 필요한 이유?

패딩을 하지 않을 경우 데이터의 공간 크기는 합성곱 계층을 지날때 마다 작아지게 되므로 가장 자리 정보들이 사라지게 되는 문제가 발생하기 때문에 이를 방지하고자 패딩을 사용한다. (공간적 구조를 고정시킬수 있음)

### 스트라이드(stride)

필터를 적용하는 위치의 간격

필터를 적용하는 윈도우가 이동하는 간격

스트라이드를 키우면 출력 크기는 작아진다.

한편, 패딩을 크게하면 출력 크기는 커진다.

### 콘볼루션 연산을 통한 출력 데이터 크기(shape) 계산

(input - filter) / stride + 1

```
# 입력데이터 크기 (H,W), 필터크기 (FH, FW), 패딩 P, 스트라이드 S, 출  
력데이터 크기(OH, OW)  
OH = ((H + 2*P - FH) / S) + 1  
OW = ((W + 2*P - FW) / S) + 1
```

### 보정

4x4 데이터에 4x4 필터를 적용한다고 생각해보자. 그러면 하나의 출력이 나올것이다.

즉, 아무리 사이즈가 큰 필터를 취해도 하나의 출력은 나온다  
따라서 +1을 써준다.

### 나누기

입력데이터를 스트라이드 크기만큼 나누는 이유는 다음과 같다.

x 축에 0 ~ 6 이라는 값이 1이라는 간격으로 나눠져 있다고 해보자.

그럼 6개의 점이 생길것이다.

x 축에 0~6이라는 값이 2이라는 간격으로 나눠져 있다고 해보자.

그럼 3개의 점이 생길것이다.

즉, 어떤 간격으로 나아간다 라는 것은 그 간격 안에 있는 아이들로 한꺼번에 처리하는 것과 같으며(묶어서 처리)

스트라이드는 나아가는 것이기 때문에 그 만큼 나눈다.

### 더하기

크기에  $2 * P$ 를 더하는 이유는 패딩이 하나 추가 되면 크기가 그만큼 늘기 때문이다.

### 빼기

필터의 크기만큼 빼는 이유는 필터의 크기가 늘어난 만큼 출력 되는 데이터가 줄어들기 때문이다.

### 다음과 같이 행렬을 구성해야함

```
# image: 1, 3, 3, 1 (이미지 n개, 행, 열, 필터수)  
# filter: 2, 2, 1, 1 (행, 열, 색수, 필터수)  
# stride : 1, 1, 1, 1 (무시, 행이동수, 열이동수, 무시)  
ex)  
image1 = np.array([  
    [[[1], [2], [3]],  
     [[4], [5], [6]],  
     [[7], [8], [9]]]  
], dtype = np.float32)  
  
filter = np.array(  
    [[[1], [[1]]],  
     [[[1]], [[1]]]])
```

### 색상정보

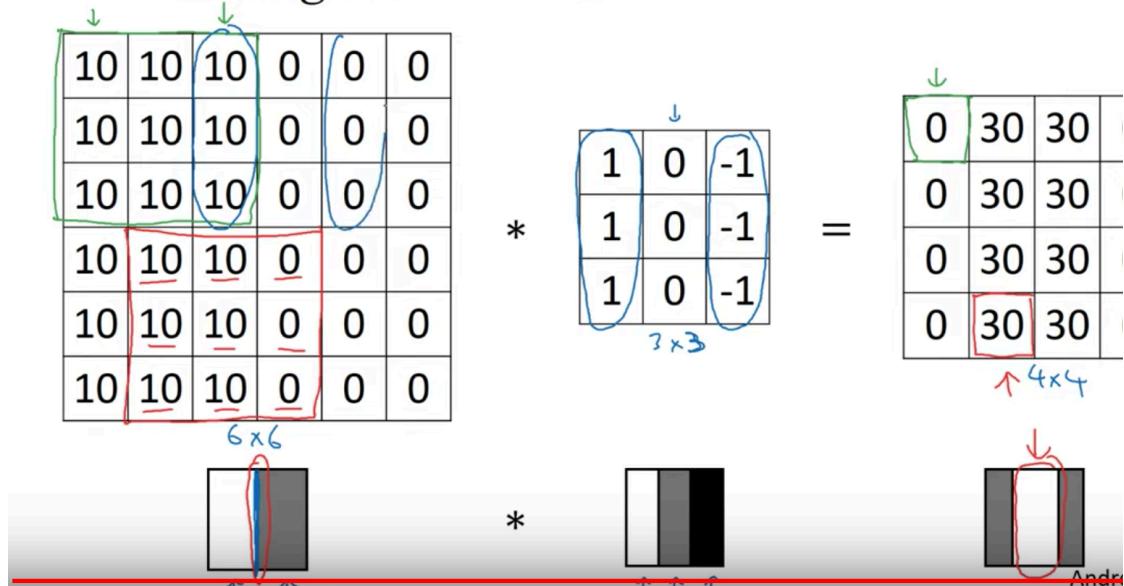
```
dir(plt.cm)
```

### Vertical edge detection

합성곱을 통해 수직경계선을 찾을수 있다.

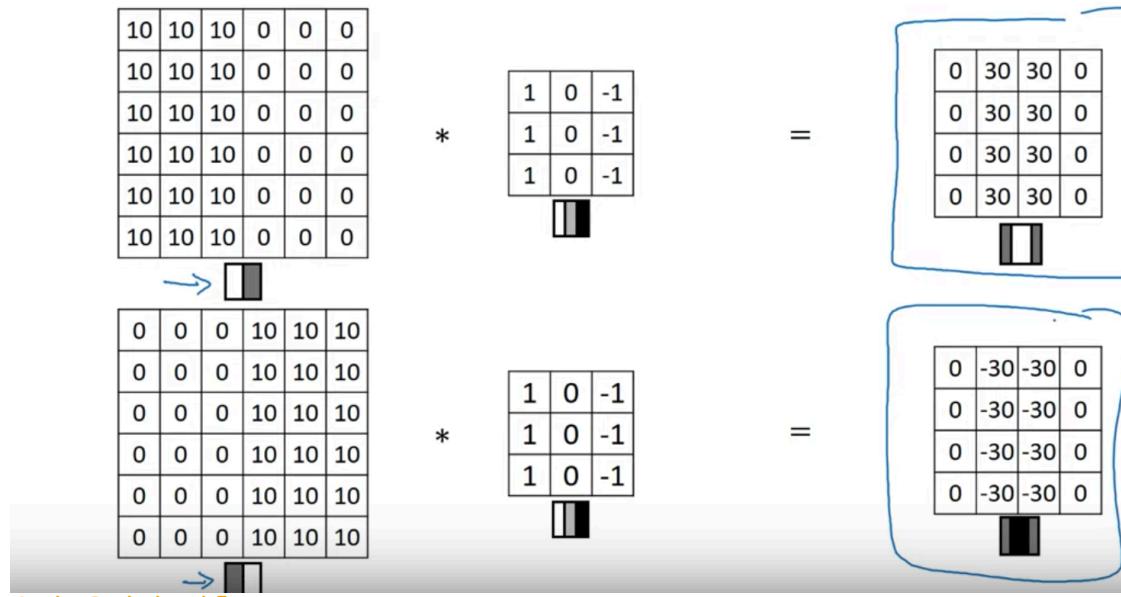
필터의 왼쪽이 상대적으로 밝은 부분이고, 오른쪽이 어두운 부분

## Vertical edge detection



위 이미지는 왼쪽이 밝고 오른쪽이 어두워서 양의 윤곽선을 가짐  
아래 이미지는 왼쪽이 어둡고 오른쪽이 밝아서 양의 윤곽선을 가짐

## Vertical edge detection examples



### 수평 윤곽선 검출

수평 윤곽선은 상대적으로 위가 밝은 부분이고 아래가 어두운 필터를 적용

아래 이미지의 노란색 부분은 상대적으로 절대값이 적은데, 소스 이미지가 커지면 상대적으로 적은 값은 무시될것이므로 수평윤곽선이 검출될것이다.

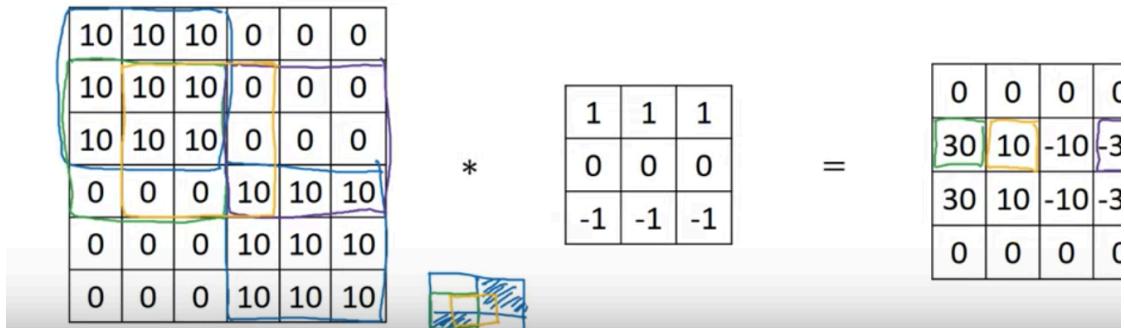
# Vertical and Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

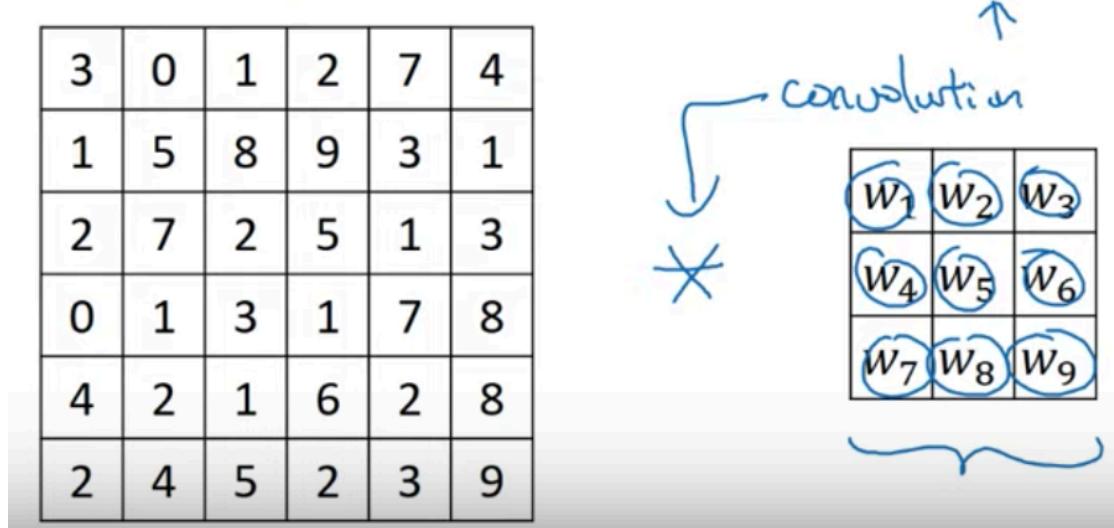
1	1	1
0	0	0
-1	-1	-1

Horizontal



## 필터 요소 결정

필터에 있는 각 요소를 변수로 두고, 역전파를 통해 학습시키면 필터 안에 들어가는 숫자를 일일히 택할 필요는 없다.



## 미니배치(mini batch)

딥러닝에서 한번의 반복을 위해 들어가는 인풋데이터는 보통 batch라고 하여 수십 수백 개의 데이터를 한 그룹으로 사용하게 된다.

### 데이터를 한개 쓰는 경우

#### 장점

iteration 한번 수행하는데 소요되는 시간이 매우 짧다.

cost function의 최적의 값을 찾아가는 과정을 한걸음 한걸음 minimum 을 향해 걸어가는 것으로 생각한다면 매우 빠르게 걸을수 있다.

#### 단점

데이터 전체의 경향을 반영하기 힘들다.  
그래서 업데이트를 꼭 좋은 방향으로만 하지 않는다.  
현재 학습을 진행하는 데이터 한개에 대해서는 cost function의 값이 줄어들더라도 이로 인해 다른 데이터에 대해서는 cost가 증가할수 있기 때문이다.  
하드웨어 입장에서 비효율적이다.  
현재 딥러닝에 GPU를 많이 쓰는 이유는 강력한 병렬연산 능력 때문이다.  
그런데 한번에 데이터 한개만 학습에 사용한다면 그 병렬연산을 안쓰는 것이나 마찬가지이다.

### 전체 데이터를 쓰는 경우

#### 장점

전체 데이터를 반영하여 한걸음 한걸음을 내딛는다.  
즉, 정말로 cost function의 값을 줄이는 양질의 이동을 하게 된다.

#### 단점

데이터셋의 크기가 커질 경우 iteration을 한번 수행하는데 소요되는 시간이 매우 길다.  
최적의 위치를 찾아가기 위해서 최소한으로 수행해야 하는 iteration이 존재하기에 학습시간이 매우 길어진다.  
하드웨어 입장에서 부담스럽다.  
데이터셋이 커질 경우 그 데이터를 메모리에 올려야 될 뿐만 아니라 그 데이터의 전처리한 결과나 레이어를 거친 아웃풋등도 수시로 메모리를 드나든다. 즉, 매우 큰 메모리 용량이 필요하게 된다.

### 과적합(overfitting)

모델이 너무 training data에 맞춰져있다.

ex)

만약 우리가 많은 feature들을 갖는다고 할때, hypothesis가 training set에 아주 잘 맞도록 학습되어지나 일반화에 실패한 상태

#### 잡음에 의한 정의(본질적으로 정의)

심층 신경망 같은 복잡한 모델은 데이터에서 미묘한 패턴을 감지할 수 있지만, 훈련 세트에 잡음이 많거나 데이터셋이 너무 작으면(샘플링 잡음이 발생하므로) 잡음이 섞인 패턴을 감지하게 된다.

당연히 이런 패턴은 새로운 샘플에 일반화되지 못한다.

#### 어떤때에 일어나는가?

훈련 데이터에 있는 잡음의 양에 비해 모델이 너무 복잡할 때 일어난다.

#### 해결방법

잡음을 줄인다.

파라미터가 적은 모델을 선정한다.

훈련 데이터를 많이 모은다.

#### 샘플에 의한 정의(나타나는 결과로 정의)

in-sampling testing에는 성능이 좋으면서, out-of-sample testing에는 성능이 안좋은 것 규제(regularization)

모델을 단순하게 하고 과적합의 위험을 감소시키기 위해 모델에 제약을 가하는 것을 말한다.

ex) 선형회귀에서 기울기를 고정 혹은 특정 범위만을 갖게 하고 나머지 파라미터만 움직일 수 있게함(자유도가 2에서 1로 내려감)

## 모델을 단순화 한다는 것?

학습을 진행하면서 모델이 데이터의 패턴을 파악하는데, 이러한 패턴을 덜 구체적으로 파악 한다는 것임, 이렇게 하면 패턴에 영향을 끼치는 데이터의 잡음도 덜 구체적으로 파악할것이므로 과적합해소에 도움이 되는듯

## IDEA

어떤 고차함수를 데이터에 맞추려고 할때 이렇게하면 커다란 변동성을 갖는다.

(High Variance를 갖는다고 함)

즉, 너무 훈련 데이터에 맞춰져있기 때문에 일반화가 되지 않음

어떻게 과적합인지 알수 있는가?

feature가 너무 많은 경우 => 과적합이 될수 있음

## 과적합 문제의 해결

feature의 개수를 줄인다.

유지할 feature를 수동으로 선택한다. (불필요한건 버린다)

model selection algorithm (알고리즘이 어떤 feature를 선택할지  
자동으로 선택하게 함)

## 정규화(Regularization)

모든 특성을 남기되 각 특성이 갖는 영향을 줄인다. (keep all  
features, but reduce magnitude/values of  
parameters  $\theta_j$ )

### 정규화 방법

minimum  $\theta$

$$\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum \theta_j^2$$

$$J = \frac{\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum \theta_j^2}{2m}$$

### 직관적 이해

cost function이 다음과 같이 있는 경우 -

$$\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J = \frac{\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2}{2m}$$

분자에 계수가 큰 특정한 항을 더해주면

$$10^3 * \theta_1 + 10^3 * \theta_2$$

이 cost function이 최소화되는 경우는  $\theta_1$ 와  $\theta_2$ 가 0에 수  
렴하게 되는 경우이다. (단축)

따라서  $\theta_1$ 과  $\theta_2$ 가 0에 수렴하므로  $x_1$ 과  $x_2$ 를 없는 것처럼 취  
급할수 있다.

어떤 파라미터를 단축할수 있을지 모름

cost function이 다음과 같다면,

$$\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J = \frac{\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum \theta_j^2}{2m}$$

분자에 다음 항을 더해준다.

$\lambda \sum \theta_j^2$  <- 이때 sum은  $i=1 \sim i=n$  까지 ( $n$ = 파라미터의  
개수 즉, feature의 개수)

$\lambda$  = 정규화 매개변수 (regularization parameter)  
하는일

두 가지 목표의 균형을 컨트롤한다.

1. training set에 잘 맞게 한다.

2. 매개변수를 작게 유지한다.

이 두 목표사이의 trade off를 제거한다.

### $\lambda$ 가 매우 큰 경우

$\lambda$ 가 너무 크면 모든 파라미터가 0에 수렴하므로,

hypothesis가  $0\theta$ 가 되게 된다.

따라서, 적당한  $\lambda$ 를 선택해야 한다.

### 정규화 된 cost function 으로 경사 하강법 적용하기

정규화 된 cost function은 다음과 같다.

$$J(\theta) = \frac{\sum (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum \theta_j^2}{2m}$$

이 cost function은  $\theta_1 \dots \theta_n$  까지만 적용한다. ( $n =$  feature의 개수)

$\theta_0$ 는 그냥 bias 이기 때문

위 식을 편미분하면 다음과 같은 식이 나타난다.

```
Repeat {  
     $\theta_0 := \theta_0 - \alpha(1/m) \sum (h_{\theta}(x^{(i)}) -$   
     $y^{(i)}) x_{\theta}^{(i)}$   
     $\theta_j := \theta_j - \alpha[(1/m) \sum (h_{\theta}(x^{(i)}) -$   
     $y^{(i)}) x_{\theta}^{(i)} + \lambda/m \theta_j] \quad (j = 1, \dots,$   
     $n)$   
}
```

그리고 이를 정리하면 ( $\theta_0$ 를 제외하고)

$$\theta_j := \theta_j (1 - \alpha(\lambda/m)) - \alpha(1/m) \sum (h_{\theta}(x^{(i)}) -$$
 $y^{(i)}) x_{\theta}^{(i)}$

#### 특이사항

$1 - \alpha(\lambda/m)$  는 1보다 작다.

$\alpha(\lambda/m)$ 에서  $m$ 으로 나눴기 때문에 이는 아주 작은 값이 됨  
( $m$ : 데이터셋의 크기)

따라서  $1 - \alpha(\lambda/m)$ 은 보통 1보다 조금 작다. ex) 0.99

그래서 식은 이런식으로 고쳐진다.

$$\theta_j * (0.99) - \alpha(1/m) \sum (h_{\theta}(x^{(i)}) -$$
 $y^{(i)}) x_{\theta}^{(i)}$

여기서 두번째항은 우리가 경사하강법에서 자주보던 식이며,  
첫번째항은  $\theta_j$ 를 조금 축소하는 것이 된다.

### 정규방정식과의 관계

정규방정식은 Cost function이 0가 되는 지점을 찾는 방법이다.

그런데, cost function이 정규화되었기 때문에 식이 조금 바뀐다.

$$\theta = ((X'X) + \lambda[ (1, 1) 원소를 제외한 대각원소가 1인 행렬 -$$
 $나머지 0 ])^{-1} X'y$

이때 regularization parameter 뒤에 붙는 행렬의 shape은 다음과 같다. ( $n = \text{feature}$ 의 개수 일때)  
X의 차원은  $(m, n+1)$  이 되므로  $X'X$  의 차원은  $(n+1, n+1)$  이 되기 때문에, (column 수가  $n+1$  인 이유는 편향과 곱해질 값이 붙기 때문)  
정규화 매개변수 뒤에 붙는 행렬의 차원은  $(n+1, n+1)$ 이 되어야함

### Non-invertibility

$m \leq n$  인 상황을 가정할 때,  $X'X$  는 singular matrix가 된다.  
그런데 정규화한 cost function에서의 정규방정식의 경우  
 $(X'X) + \lambda [ (1, 1) \text{ 원소를 제외한 대각원소가 } 1 \text{인 행렬} - \text{나머지 } 0 ]$  는 invertible matrix이다.

### 일반화(generalization)

추상화된 데이터를 이용해 지식과 추론을 생성함으로써 새로운 상황(context)에서 실 행을 하게 만든다.

일반화라는 용어는 이전에 봤던 것과 비슷하지만 같지는 않은 작업에 대해 추상화된 지식을 미래의 실행에 활용할 수 있는 형태로 변환하는 과정을 말한다.

일반화는 발견된 패턴을 미래의 작업에 가장 적합한 패턴들로 제한하는 작업을 맡는다.  
머신러닝 알고리즘은 휴리스틱을 사용한다.

### 편향

결론이 체계적으로 틀리거나 너무 뻔한 방식으로 잘못됐다면 알고리즘은 편향을 가졌다고 말할 수 있다.

### 약간의 편향의 유용성

편향은 실행을 할 때 어떤 정보는 보지 못하게 하는 대신 다른 정보는 활용 할 수 있게 한다.

편향은 학습 작업에 고유한 추상화와 일반화 과정과 관련돼 있는 필요악이다.

무한한 가능성에 맞서 실행을 추진하려면 각 학습자는 특정 방식으로 편향되어야만 한다.

### 과소적합(underfitting)

모델이 너무 단순해서 데이터에 내재된 구조를 학습하지 못함

#### 해결방법

- 파라미터가 더 많은 모델을 선택한다.
- 학습 알고리즘에 더 좋은 특성을 제공한다.(특성 엔지니어링)
- 모델의 제약을 줄인다

### IDEA

직선을 데이터에 맞추는 경우에 알고리즘이 강한 선입견(preconception)을 갖는다.

즉, 강한 편향을 갖는다.(high bias) 라고도 함

### Keras

이미지 변환, 이미지 부풀리기

```
from keras.preprocessing.image import ImageDataGenerator,  
    img_to_array, load_img  
이미지 부풀리기?
```

### **load\_img()**

이미지 로드

ex) `img = load_img('c:/101_ObjectCategories/chair/image_0001.jpg')`

### **img\_to\_array()**

이미지를 array 형식으로 반환

ex) `img_to_array( load_img('c:/101_ObjectCategories/chair/image_0001.jpg') )`

### **ImageDataGenerator()**

#### **rotation\_range**

random rotation에 대한 이미지 회전 범위 (0~180)

#### **width\_shift\_range**

그림을 수평으로 랜덤하게 평행 이동시키는 범위(원본 가로길이에 대한 비율 값)

#### **height\_shift\_range**

그림을 수직으로 랜덤하게 평행 이동시키는 범위(원본 세로 길이에 대한 비율 값)

#### **shear\_range**

y축 방향으로 각도를 증가시켜 이미지를 변형시킨다.

#### **zoom\_range**

확대, 축소 범위

#### **horizontal\_flip**

True로 설정할 경우 50% 확률로 이미지를 수평으로 뒤집는다.

#### **fill\_mode**

이미지를 회전, 이동하거나 축소할때 생기는 공간을 채우는 방식

#### **rescale**

원본 영상은 0~255의 RGB 계수로 구성되는데 이 같은 입력값은 모델을 효과적으로 학습시키기에 너무 높다.

그래서 이를 스케일링하여 0~1범위로 변환시켜 준다.

```
keras.layers.Dense(units, activation=None, use_bias=True,
                    kernel_initializer='glorot_uniform',
                    bias_initializer='zeros', kernel_regularizer=None,
                    bias_regularizer=None, activity_regularizer=None,
                    kernel_constraint=None, bias_constraint=None)
```

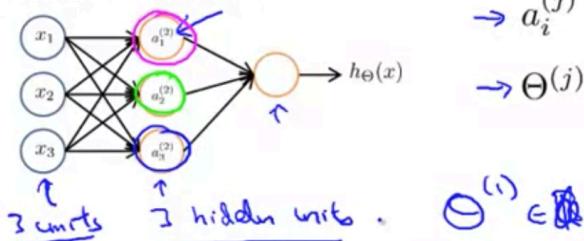
보통의 밀집 연결 신경망 레이어

## **인공 신경망(Neural Network)**

비선형 가설을 학습하는 좋은 방법

인공 신경망은 여러개의 뉴런 그룹들을 연결하여 가설  $h_{\theta}(x)$ 를 정의하는 것이다.

## Neural Network



$\rightarrow a_i^{(j)}$  = "activation" of unit  $i$  in layer  $j$

$\rightarrow \Theta^{(j)}$  = matrix of weights controlling function mapping from layer  $j$  to layer  $j + 1$

$$\Theta^{(j)} \in \mathbb{R}$$

$$\rightarrow a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$\rightarrow a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$\rightarrow a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

### 기원

뇌를 모방하기 위한 알고리즘

우리가 알고 있는 가장 놀라운 학습 기계는 뇌이다.

1980년 ~ 1990년대초에 많이 사용되다가 1990년대 후반 인기가 떨어짐

최근 다시 인기를 얻고 있는데, 신경망이 계산시 매우 비용이 큰데, 컴퓨터 하드웨어의 발전에 따라 계산작업이 빨라졌기 때문임

### 뉴런(Neuro)

뉴런은 계산 단위로 볼수 있다.

즉 input 을 받아 어떤 계산을 수행하고 output을 출력하는 단위이다.(unit)

- 뇌에서의 뉴런은 수상돌기로 입력을 받아 세포핵에서 계산을 수행하고 축삭돌기로 출력을 내보낸다.

### 층(layer)

#### 입력층(input layer)

입력 되는 층이다.

여기서  $x_0 \sim x_n$  까지의 입력을 받는데 여기서  $x_0$ 는 항상 1이다.(bias에 곱해질 값이므로)

bias를 위한 뉴런을 bias unit이라 함  
1번째 층이다.

#### 은닉층(hidden layer)

X도 아니고 Y도 아닌 층이다.

즉, 숨겨진 층이다.

#### 출력층(output layer)

가설을 계산하여 최종값을 출력하는 층이다.

#### 층의 개수

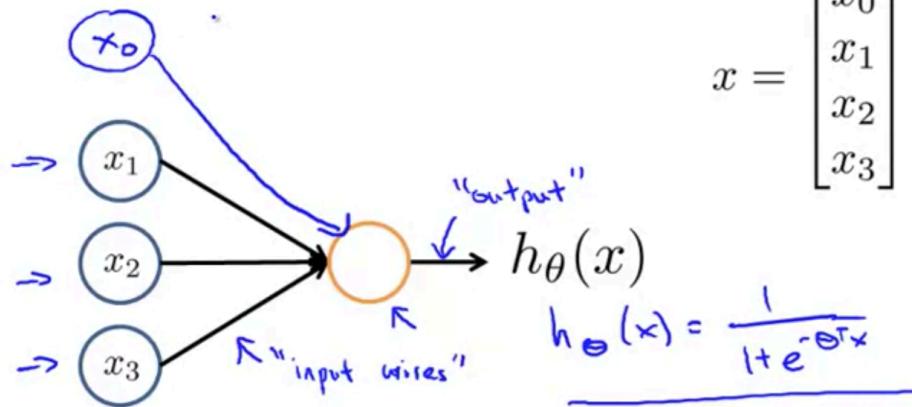
1(입력층) + hidden layer의 수 + 1(출력층)

#### 층 index

1번째 층부터 시작

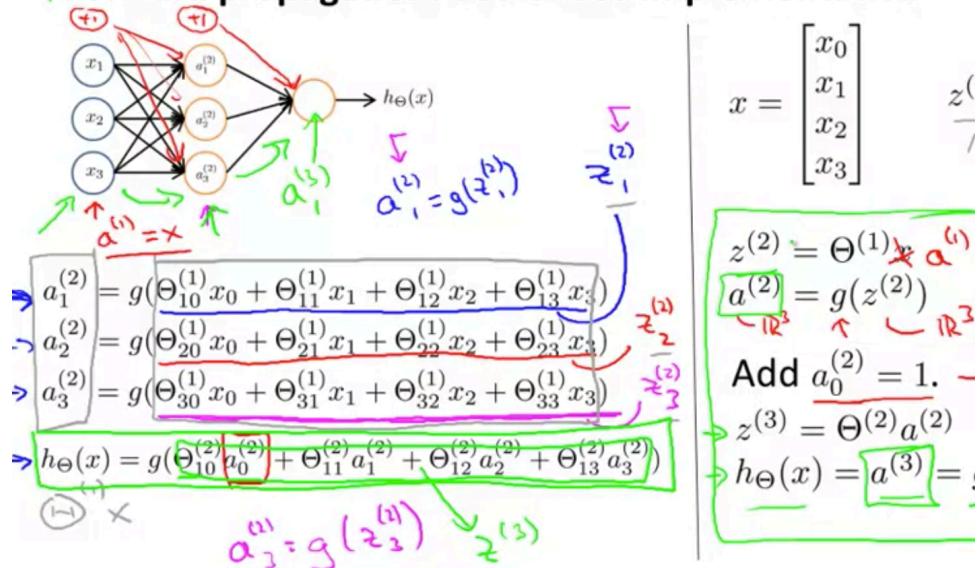
시그모이드 인공 뉴런(Sigmoid artificial neuron), Sigmoid(logistic) activation function  
 출력으로 시그모이드 함수를 적용하는 뉴런  
 $h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$

### Neuron model: Logistic unit



표현 방법(Model Representation)

### Forward propagation: Vectorized implementation



$a_i^{(j)}$

activation(output) of unit i in layer j  
 j층에서의 i번째 출력을 의미

ex)

$$a_1^{(2)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$$a_2^{(2)} = g(\Theta_{20}^{(2)} a_0^{(2)} + \Theta_{21}^{(2)} a_1^{(2)} + \Theta_{22}^{(2)} a_2^{(2)} + \Theta_{23}^{(2)} a_3^{(2)})$$

$$a_2^{(3)} = g(\Theta_{20}^{(3)} a_0^{(3)} + \Theta_{21}^{(3)} a_1^{(3)} + \Theta_{22}^{(3)} a_2^{(3)} + \Theta_{23}^{(3)} a_3^{(3)})$$

$\theta^j$

한 레이어를 다른 레이어로 매팅하는 함수를 조절하는 가중치 Matrix

dimension

$s_j$ : j층의 units

$s_{(j+1)}$ : j+1층의 units 라고 할때,

$\theta^j$  의 dimension은 다음과 같다. ( $s_{(j+1)} * (s_j + 1)$ )

$s^j$

bias 항을 제외한 j번째 layer에서의 unit의 수

$h_\theta(x)$

출력층이 3일 때,

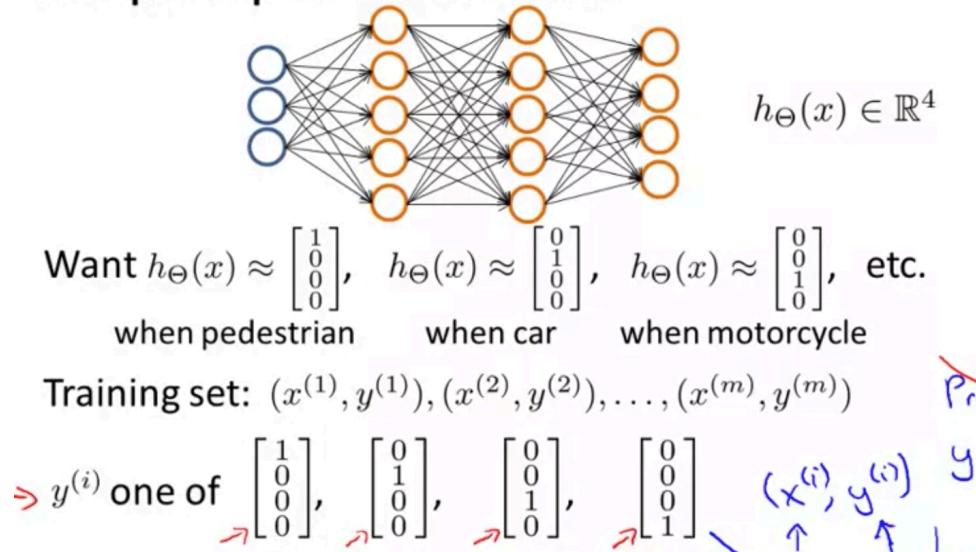
$$h_\theta(x) = a_1 \cdot 1^3 = g(\theta_{10} \cdot a_0^2 + \theta_{11} \cdot a_1^2 + \dots + \theta_{1n} \cdot a_n^2)$$

### Multiple Classification

one-vs-all 를 이용

출력층의 첫 번째 뉴런에서는 보행자임을 판단, 2 번째 뉴런에서는 차임을 판단 ...

### Multiple output units: One-vs-all.



cost function

## Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Neural network:

$$\rightarrow h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$\rightarrow J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

순전파(Forward Propagation)

신경망에서 모든 뉴런의 활성화 값을 계산

역전파(Backward Propagation)

Neural network의 Cost function의 미분을 계산하기 위해 역전파 알고리즘을 사용한다.  
 $\delta$ 의 계산을 출력단부터 시작해서 히든레이어까지 거꾸로 계산해 나가는데서 지어졌다.

역전파 알고리즘은 다층신경망을 학습하는 방법인데, 역전파 알고리즘의 아이디어는 은닉 층 노드들의 오차를 확인하고

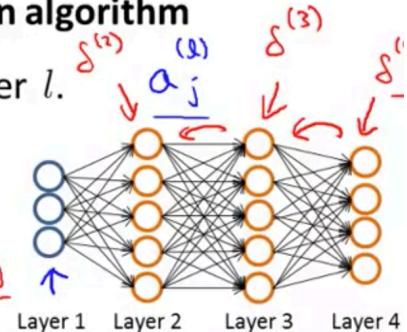
델타 규칙에 따라 이 오차들로 가중치들을 조정하는 것이다.

데이터 셋이 하나일 경우

결국 손실함수의 미분을 구하기 위해서 error를 구하게됨

## Gradient computation: Backpropagation algorithm

Intuition:  $\delta_j^{(l)}$  = "error" of node  $j$  in layer  $l$ .



For each output unit (layer  $L = 4$ )

$$\delta_j^{(4)} = a_j^{(4)} - y_j \quad (h_\Theta(x))_j \quad \delta^{(4)} = a^{(4)} - y$$

$$\rightarrow \delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \cdot g'(z^{(3)})$$

$$\rightarrow \delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} \cdot g'(z^{(2)})$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignore } \lambda; \text{ if } \lambda = 0)$$

$\delta_j^{(l)}$  = layer  $l$ 에서  $j$  노드의 오류

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_i^{(l)} \delta_j^{(l+1)}$  (ignore  $\lambda$ ; if  $\lambda = 0$ )

ex)

만약 출력레이어가 4번째라고 하면

$\delta_j^{(4)} = a_j^{(4)} - y_j$  <- 4번째 Layer의 j번째 노드의 오류 계산 여기서  $a_j^{(4)}$ 는  $(h_\Theta(x))_j$

이를 행렬로 표현하면

$$\delta^{(4)} = a^{(4)} - y$$

$\delta^{(3)} = (\Theta^{(3)})' \delta^{(4)} \cdot g'(z^{(3)})$  여기서  $\cdot$ 는 행렬의 요소곱

$$\delta^{(2)} = (\Theta^{(2)})' \delta^{(3)} \cdot g'(z^{(2)})$$

여기서  $g'(z^{(3)})$ 는 다음과 같다.  $a^{(3)} \cdot (1 - a^{(3)})$

$\delta^{(1)}$ 은 존재하지 않음 (인풋이니까)

### 데이터셋이 여러개인 경우

여기서 i는 특정 데이터를 의미

### Backpropagation algorithm

→ Training set  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set  $\Delta_{ij}^{(l)} = 0$  (for all  $l, i, j$ ). (use to compute  $\frac{\partial}{\partial \Theta^{(l)}} J(\Theta)$ )

For  $i = 1$  to  $m \leftarrow (x^{(i)}, y^{(i)})$ .

Set  $a^{(1)} = x^{(i)}$

→ Perform forward propagation to compute  $a^{(l)}$  for  $l = 2, 3, \dots, L$

→ Using  $y^{(i)}$ , compute  $\delta^{(L)} = a^{(L)} - y^{(i)}$

→ Compute  $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

→  $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

→  $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$  if  $j \neq 0$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

→  $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$  if  $j = 0$

대문자 델타는 축적자로써 사용(accumulator) - 오류를 축적하는 용인듯  
결국 경사하강법을 하기 위해(파라미터를 구해야함), 비용 함수의 특정 파라미터에 대한 미분을 구해야하고 이 미분을 구하기 위해 역전파법을 사용한다.

역전파법은 에러를 축적한다.(모든 데이터의 오류를 축적)

### 직관

신경망의 비용함수은 다음과 같다.

$$J(\Theta) = -\frac{1}{m} \sum_{t=1}^m \sum_{k=1}^K \left[ y_k^{(t)} \log(h_\Theta(x^{(t)}))_k + (1 - y_k^{(t)}) \log(1 - h_\Theta(x^{(t)}))_k \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ij}^{(l)})^2$$

정규화를 제외시키고 다중분류를 제외시켜 표현하면  $J(\Theta)$ 에 들어가는 함수 cost는 다음과 같이 정의할 수 있다.

$$cost(t) = y^{(t)} \log(h_\Theta(x^{(t)})) + (1 - y^{(t)}) \log(1 - h_\Theta(x^{(t)}))$$

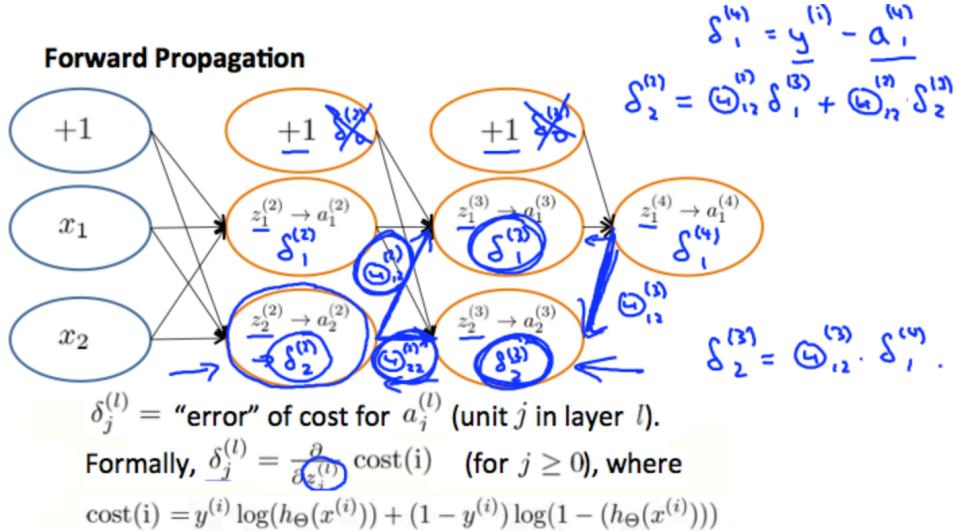
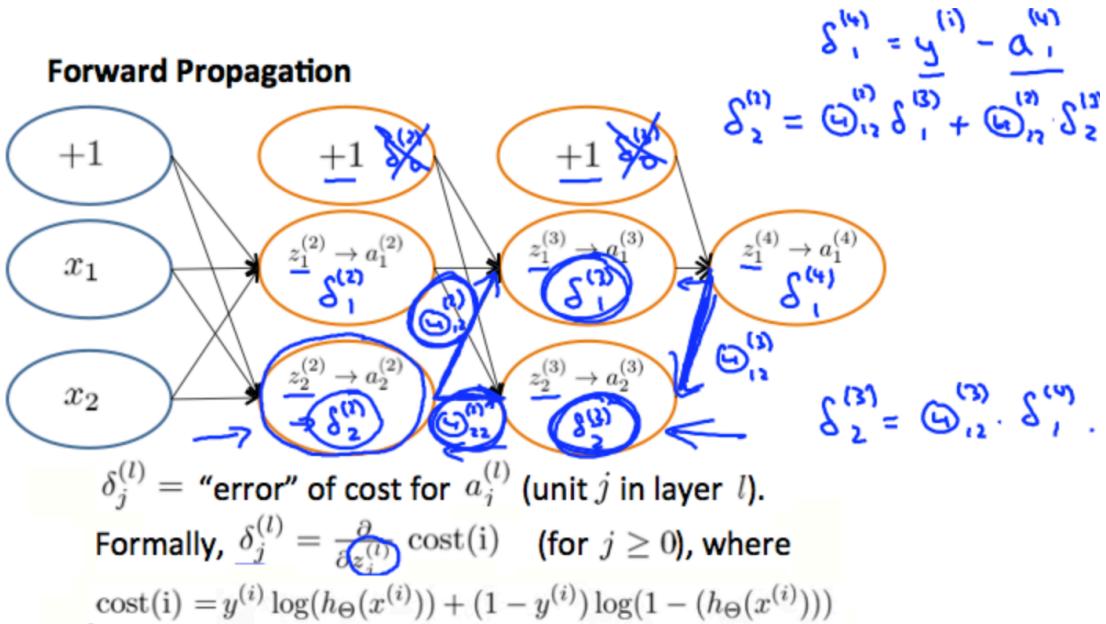
cost는 Network가  $y^{(t)}$ 를 얼마나 잘 예측하는지를 측정한다.

직관적으로  $\delta_j^{(l)}$ 는 레이어 l에서  $a_j^{(l)}$ 에 대한 오류(error)이다. (위 식을 선형 회귀에서 오차제곱과 같다고 보면됨)

형식적으로는 cost의 미분이다.

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(t)$$

위 도함수는 cost의 경사이다. 즉, 경사가 가파를수록 더 부정확하다.



Andrew Ng

In the image above, to calculate  $\delta_2^{(2)}$ , we multiply the weights  $\Theta_{12}^{(2)}$  and  $\Theta_{22}^{(2)}$  by their respective  $\delta$  values found to the right of edge. So we get  $\delta_2^{(2)} = \Theta_{12}^{(2)} * \delta_1^{(3)} + \Theta_{22}^{(2)} * \delta_2^{(3)}$ . To calculate every single possible  $\delta_j^{(l)}$ , we could start from the right of our diagram. We can think of our edges as our  $\Theta_{ij}$ . Going from right to left, to calculate the value of  $\delta_j^{(l)}$ , you can just take the over all sum each weight times the  $\delta$  it is coming from. Hence, another example would be  $\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}$ .

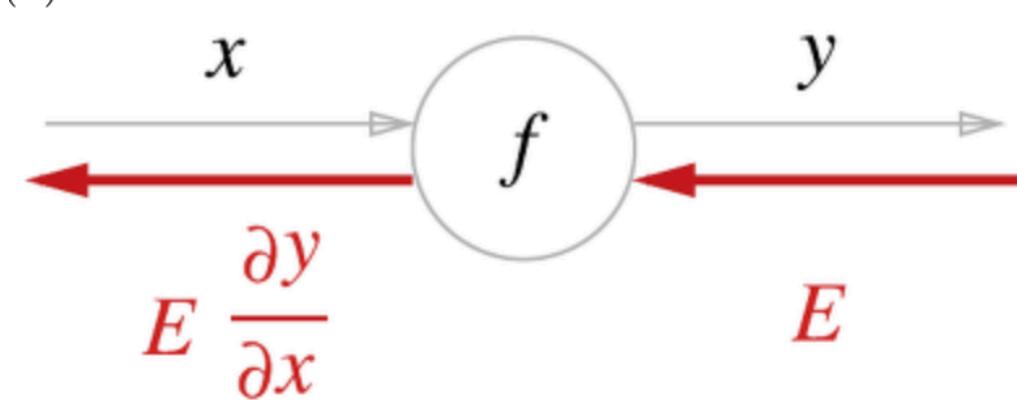
결국에는,  $\delta_j^{(l)}$ 는 특정 뉴런의 출력이 cost 함수에 얼마나 영향을 미치는지 알아보는건데, 이를 쉽게 하려면 역전파법을 사용하면 된다.

즉, 각 뉴런에서 출력 되는 오류에 가중치를 곱한 값을 더해주면 됨(내 뉴런에서 다른 뉴런으로 특정한 계수만큼의 오류가 전해졌다는 뜻이니까 이를 합치면 한 뉴런에서의 오류

가 됨)

### 계산 그래프를 통한 표현

역전파의 계산 절차는 신호  $E$ 에 노드의 국소적 미분을 곱한 후 다음 노드로 전달하는 것이다.



### 역전파와 연쇄법칙

이러한 방식을 따르면 목표로하는 미분 값을 효율적으로 구할 수 있다는 것이 이 전파의 핵심이다.

#### 연쇄법칙

합성함수의 미분은 합성함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다.

$$\begin{aligned} f(g(h(x))) &= y \\ g(h(x)) &= t \\ h(x) &= s \quad \text{라고 때}, \\ \frac{\partial f(g(h(x)))}{\partial x} &= \frac{\partial f(g(h(x)))}{\partial g(h(x))} \cdot \frac{\partial g(h(x))}{\partial h(x)} \cdot \frac{\partial h(x)}{\partial x} \\ \Leftrightarrow \frac{\partial y}{\partial x} &= \frac{\partial y}{\partial t} \cdot \frac{\partial t}{\partial s} \cdot \frac{\partial s}{\partial x} \\ \Rightarrow \text{복잡한 함수가 } \text{과연 } \text{미분은 } "국소적 미분" \text{을 } \text{통해 } \text{도전할 수 있다} \end{aligned}$$

### 덧셈 노드의 역전파

덧셈 노드의 역전파는 1을 곱하기만 할 뿐이므로 입력된 값을 그대로 다음 노드로 보내게 된다.

### 곱셈 노드의 역전파

곱셈 노드의 역전파는 상류의 값에 순전파 때의 입력신호들을 “서로 바꾼 값”을 곱해서 하류로 보낸다.

### ReLU

ReLU 함수는 다음과 같다.

ReLU는 전기 회로의 ‘스위치’에 비유할 수 있다. 순전파 때 전류가 흐르고 있으

면 스위치를 ON으로 하고, 흐르지 않으면 OFF로 한다.

역전파 때는 스위치가 ON이라면 전류가 그대로 흐르고, OFF면 더이상 흐르지 않는다.

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

### 역전파

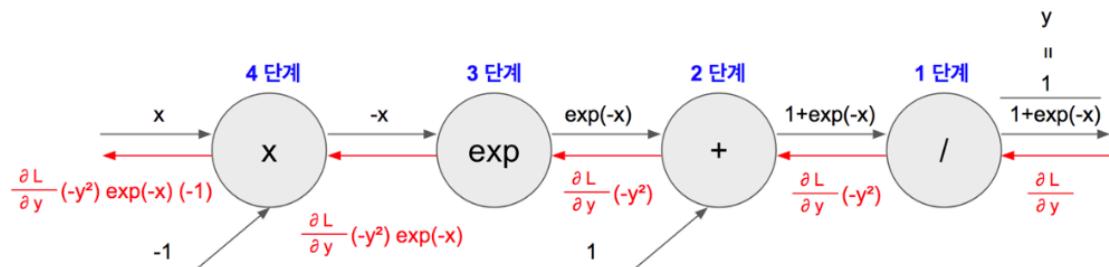
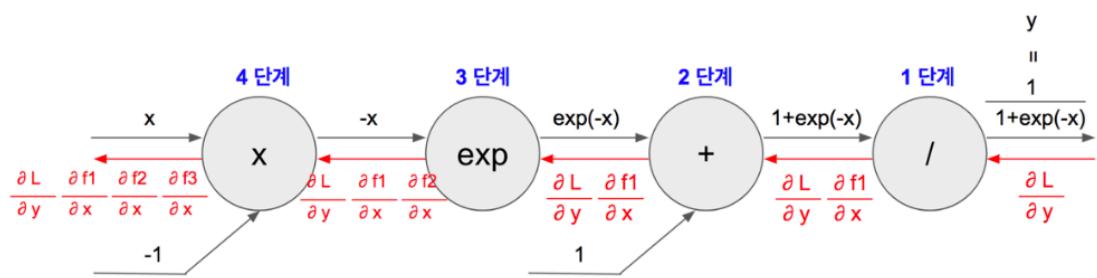
순전파 때의 입력인  $x$ 가 0보다 크면 역전파는 상류의 값을 그대로 하류로 흘린다.

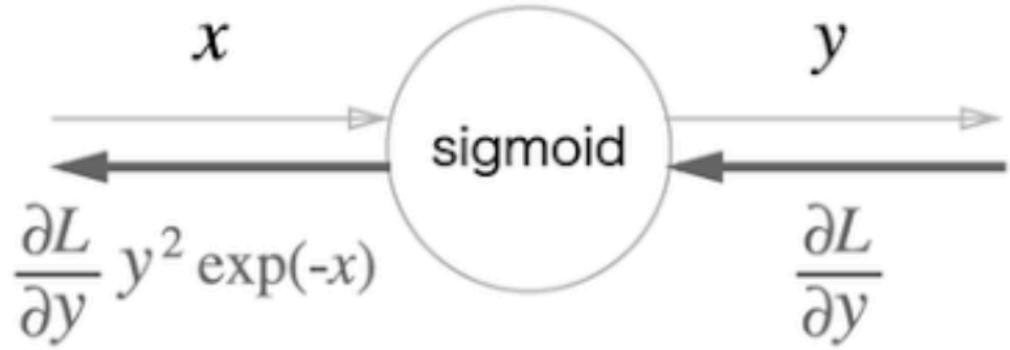
순전파 때  $x$ 가 0이하면 역전파 때는 하류로 신호를 보내지 않는다.

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

### Sigmoid

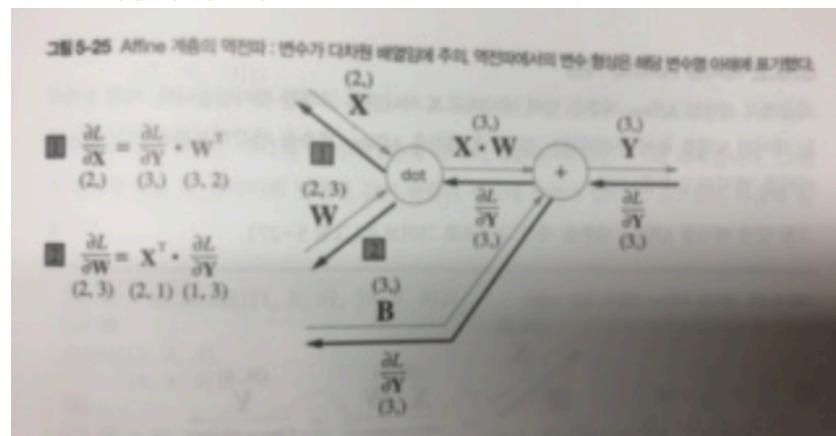
Sigmoid 계층의 역전파는 순전파의 출력( $y$ )만으로 계산할수 있다.



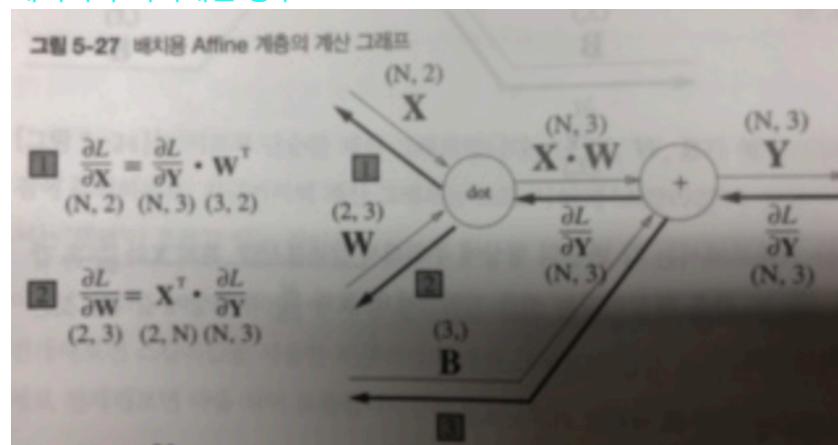


$$\begin{aligned}
 \frac{\partial L}{\partial y} y^2 \exp(-x) &= \frac{\partial L}{\partial y} \frac{1}{[1 + \exp(-x)]^2} \exp(-x) \\
 &= \frac{\partial L}{\partial y} \frac{1}{1 + \exp(-x)} \frac{\exp(-x)}{1 + \exp(-x)} \\
 &= \frac{\partial L}{\partial y} y(1 - y)
 \end{aligned}$$

### Affine 계층의 역전파



### 데이터가 여러개인 경우



unroll to vector

$\theta$ 와  $D$ 는 matrix인데, 이를 layer의 벡터로 표현할 수 있다.

만약 3계층이 존재한다고 했을 때

$$\dim(\theta^1) = 10 \times 11$$

$$\dim(\theta^2) = 10 \times 11$$

$$\dim(\theta^3) = 1 \times 11$$

$$\theta = [\theta^1, \theta^2, \theta^3]$$

$$D = [D^1, D^2, D^3]$$

이렇게 표현한 것을 각각의 요소로 나타내고 싶다면,

$$\theta^1 = \text{reshape}(\theta(1:110), 10, 11)$$

$$\theta^2 = \text{reshape}(\theta(111:220), 10, 11)$$

$$\theta^3 = \text{reshape}(\theta(221:231), 1, 11)$$

### Gradient Checking

역전파법에 버그가 있는지 확인하는 방법이다.

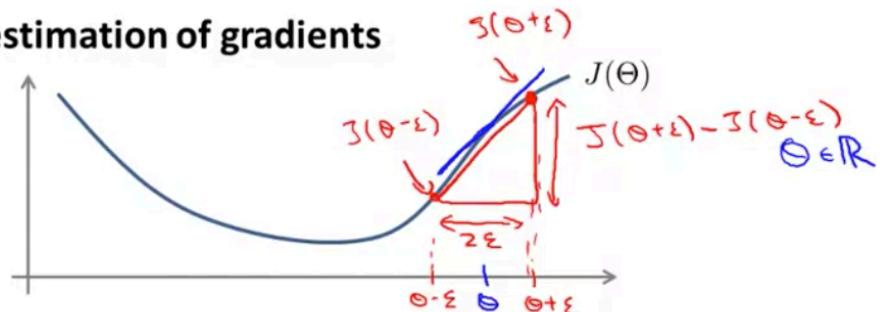
미분을 근사적으로 표현하는 수치적인 미분이다.

$\epsilon$ 은 매우 작은 값으로 둔다(ex:  $10^{-4}$ )

그런데  $\epsilon$ 을 너무 작은 값으로 두면 수치문제를 일으킬 수도 있다.

two-sided 미분이 더 나은 견적을 제공한다.

### Numerical estimation of gradients



$$\frac{\partial}{\partial \theta} J(\theta) \approx$$

$$\frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

$$\epsilon = 10^{-4}$$

~~$$\frac{J(\theta + \epsilon) - J(\theta)}{\epsilon}$$~~

---

$\theta$ 가 unroll vector로 표현되어 있으면 다음과 같이 표현할 수 있다.

## Parameter vector $\theta$

$\rightarrow \theta \in \mathbb{R}^n$  (E.g.  $\theta$  is “unrolled” version of  $\underline{\Theta^{(1)}}, \underline{\Theta^{(2)}}, \underline{\Theta^{(3)}}$ )

$$\rightarrow \theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$$

$$\rightarrow \frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$$

$$\rightarrow \frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon}$$

⋮

$$\rightarrow \frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon}$$

### 구현 노트

역전파로 D를 구한다.

Gradient Checking을 통해 D와 값을 비교한다.(소수점 둘째짜리까지 일치하는지) 올바름이 확인된다면 gradient checking을 끈다.

### turn off gradient checking

역전파법이 수치미분을 통한 근사보다 훨씬 빠르다.(수치미분은 비용이 크다)  
따라서 각 Gradient descent마다 gradient checking을 한다면 엄청 느려질

것이다.

따라서 backward propagation이 잘 동작함을 확인한다면 gradient checking 을 끄자.

(Once you have verified **once** that your backpropagation algorithm is correct, you don't need to compute gradApprox again. The code to compute gradApprox can be very slow.)

### random initialization

최적화 알고리즘 실행시  $\theta$ 의 초기값 설정이 필요하다.

만약 모든  $\theta$ 에 대해 0으로 놓는다고 가정했을 때

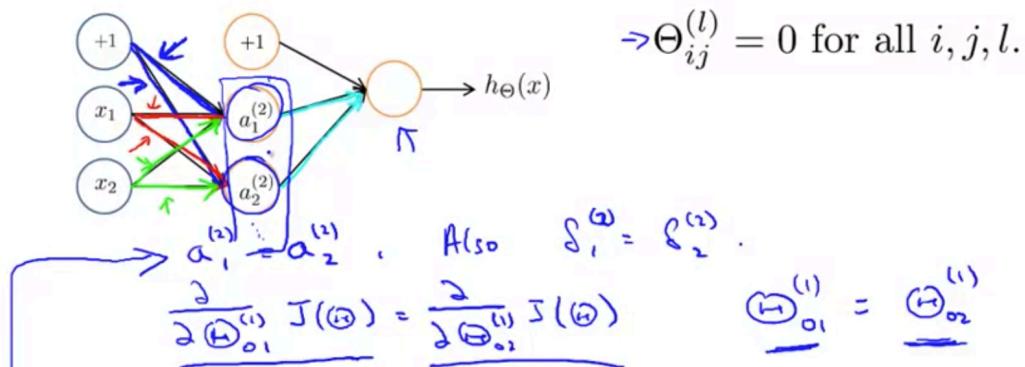
$\theta = 0$  이면

모든 노드의 출력값이 같아지고, 이로 인해 error 또한 같아지고 이로 인해  $J(\theta)$ 의 미분값도 같아진다.

따라서 Gradient descent 할 시, 각 업데이트마다 노드에서 주는  $\theta$ 들이 같아진다.  
(밑의 그림에서 파란색 끼리 같고, 빨간색 끼리 같고, 녹색이 끼리 같고, 청록색끼리 같음)

즉,  $\theta$ 가 Symmetric 하면 각 노드에서 주는 가중치가 같아지는 문제 발생!

## Zero initialization



After each update, parameters corresponding to inputs going into each of two hidden units are identical.

$$a_1^{(2)} = a_2^{(2)}$$

Symmetric breaking

random하게 설정해주면 됨

## Random initialization: Symmetry breaking

$\Rightarrow$  Initialize each  $\Theta_{ij}^{(l)}$  to a random value in  $[-\epsilon, \epsilon]$

(i.e.  $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$ )

E.g.

Random  $10 \times 11$  matrix (butw. and )

$\Rightarrow \Theta_{1} = \boxed{\text{rand}(10, 11) * (2 * \text{INIT\_EPSILON})}$

$- \text{INIT\_EPSILON};$

$[-\epsilon,$

$\Rightarrow \Theta_{2} = \boxed{\text{rand}(1, 11) * (2 * \text{INIT\_EPSILON})}$

$- \text{INIT\_EPSILON};$

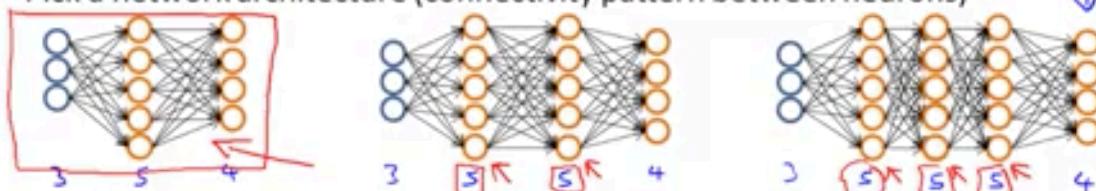
Putting it together

아키텍처 선택하기

아키텍처란? - 뉴런들을 잊고 있는 패턴

## Training a neural network

Pick a network architecture (connectivity pattern between neurons)



→ No. of input units: Dimension of features  $x^{(i)}$

→ No. output units: Number of classes

[Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)]

1. input unit의 수 결정하기(feature의 차원)
2. output unit의 수 결정하기(class의 수)
3. Hidden layer의 개수 결정하기 - 기본 1개, 만약 1개보다 레이어가 커지면 각 레이어의 유닛의 수를 같게 한다.(기본)
  - 일반적으로 hidden layer의 unit이 많을수록 좋지만, 계산이 힘들어진다.
  - 일반적으로 hidden layer의 유닛의 수는 대체로  $x$ 의 차원과 동일하거나 2배, 3배, 4배 정도이다.

계산 구현

## Training a neural network

→ 1. Randomly initialize weights

→ 2. Implement forward propagation to get  $h_{\Theta}(x^{(i)})$  for any  $x^{(i)}$

→ 3. Implement code to compute cost function  $J(\Theta)$

→ 4. Implement backprop to compute partial derivatives  $\frac{\partial}{\partial \Theta_j^{(l)}} J(\Theta)$

→ for  $i = 1:m$  {  $(x^{(1)}, y^{(1)})$   $(x^{(2)}, y^{(2)})$ , ...,  $(x^{(m)}, y^{(m)})$  }

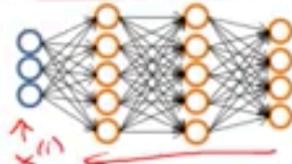
→ Perform forward propagation and backpropagation using example  $(x^{(i)}, y^{(i)})$

(Get activations  $a^{(l)}$  and delta terms  $\delta^{(l)}$  for  $l = 2, \dots, L$ ).

$$\Delta^{(2)} := \Delta^{(2)} + \delta^{(2)} (\alpha^{(2)})^T$$

}

$$\text{compute } \frac{\partial}{\partial \Theta}$$



- 5. Use gradient checking to compare  $\frac{\partial}{\partial \Theta_{ik}^{(l)}} J(\Theta)$  computed using backpropagation vs. using numerical estimate of gradient of  $J(\Theta)$ .

→ Then disable gradient checking code.

- 6. Use gradient descent or advanced optimization method with backpropagation to try to minimize  $J(\Theta)$  as a function of parameters  $\Theta$

$$\frac{\partial}{\partial \Theta_{ik}^{(l)}} J(\Theta) \quad \nwarrow$$

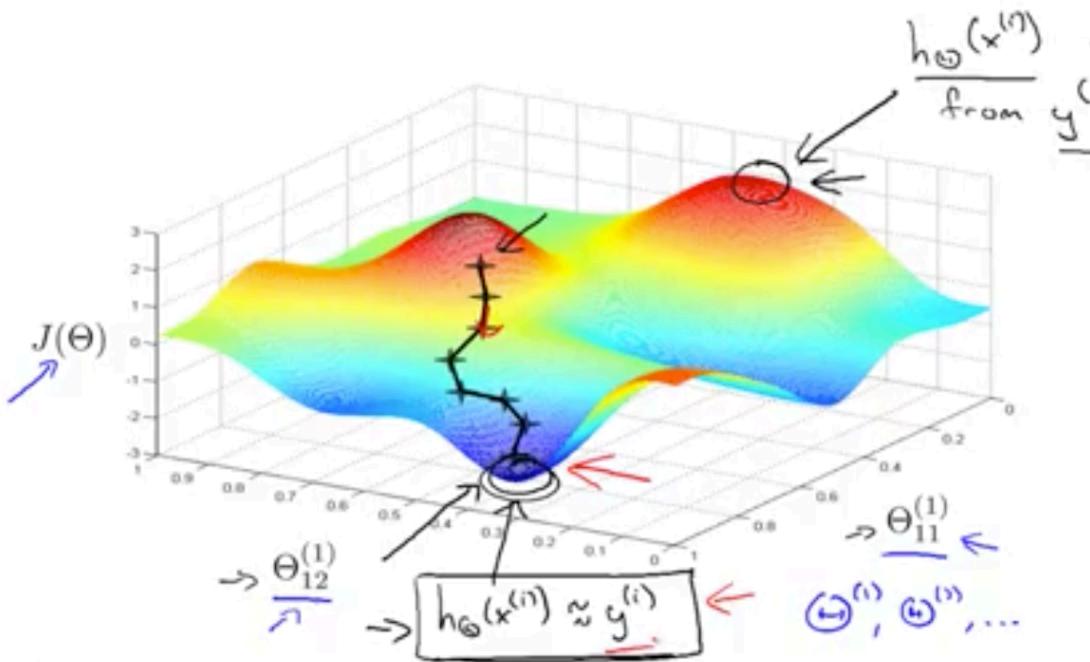
$J(\Theta)$  - non-convex.

1. weight값 랜덤으로 주기(0부근의 작은값)
2. 순전파 구현
3. 비용함수 구현
4. 역전파 구현을 위해 편미분 계산 구현
5. gradient checking을 이용하여 역전파의 값이 제대로 나오는지 확인(제대로 나오는 걸 확인하면 gradient checking을 disable 시킴)
6. 경사하강법이나 기타 최적화 기법을 사용하여 비용함수가 최소화되는 파라미터를 얻는다.

#### Network가 하는일

결국 가설과 실제치가 비슷한 지점을 찾는것

경사하강법은 언덕을 내려가게 해주고, 역전파는 그래디언트의 방향을 계산한다.



#### 컴퓨터 비전(Computer Vision)

기계의 시각에 해당하는 부분을 연구하는 컴퓨터 과학의 최신 연구 분야 중 하나이다.

공학적인 관점에서, 컴퓨터 비전은 인간의 시각이 할 수 있는 몇 가지 일을 수행하는 자율적인 시

스템을 만드는 것을 목표로 한다.

과학적 관점에서는 컴퓨터 비전은 이미지에서 정보를 추출하는 인공 시스템 관련 이론에 관여한다.

### 컴퓨터 비전과 feature

컴퓨터 비전에서 feature의 수는 매우 많다.

예를 들어 pixel1과 pixel2를 가지고 고급승용차인지, 저가 승용차인지 구분하려다면, pixel1과 pixel2가 독립변수가 될것이다.

그런데 pixel의 수는 매우 많다.

50x50의 경우 총 2500개의 pixel을 갖게 된다.

여기다가 RGB 를 적용하면 총 7500개의 pixel을 갖는다. ( $n=7500$ )

### 비선형 가설

여러개의 pixel을 가지고 비선형 가설을 표현하려 할때 어마어마한 feature의 개수가 만들어진다.(특정 차수의 항을 표현할때  $(n^2 / 2)$  만큼의 feature가 만들어지므로)

## 채널(Channel)

정보 신호를 시간 및 공간적으로 전달해 주는 한정된 통로

### 패딩(padding)

n: source의 크기

p: padding 의 크기

f: filter의 크기

### convolution의 문제점

가장자리의 정보를 덜 활용함

### 출력크기 계산

필터만 존재하는 경우:  $(n - f + 1) \times n - f + 1$

패딩이 존재하는 경우:  $(n + 2p - f + 1) \times n + 2p - f + 1$

### 패딩의 종류

valid: no padding

same: input size와 동일하게끔 padding 을 해줌 /  $p = f - 1 / 2$

## 스트라이드(strides)

s: stride의 크기

### 출력크기 계산

$(n + 2p - f) / s + 1 <-$  stride가 있으니까 s로 나눔

만약, 나눈 값이 정수가 아니라면 내림

## Multiple filters

### Convolutions on RGB

(height x width x #channels)를 갖는 input이 있다고 했을때

필터의 채널수는 input의 #channels과 같아야함.

그리고 input에 필터의 채널들을 적용하여 합하면 convolution을 구할수 있다.

### 채널

filter의 채널을 한데 묶어 블록으로 묶어 생각하면 편함

### Multiple filters

입력이 다음과 같고 ( $n \times n \times n_c$ )  $n_c$ 는 입력의 채널수

필터가 다음과 같을 때 ( $f \times f \times f_c$ )  $f_c$ 는 필터의 채널수  
출력크기는 다음과 같다. (스트라이드 x, 패딩 x)  
( $n-f+1 \times n-f+1 \times n_c'$ )  $n_c'$ 은 필터의 개수

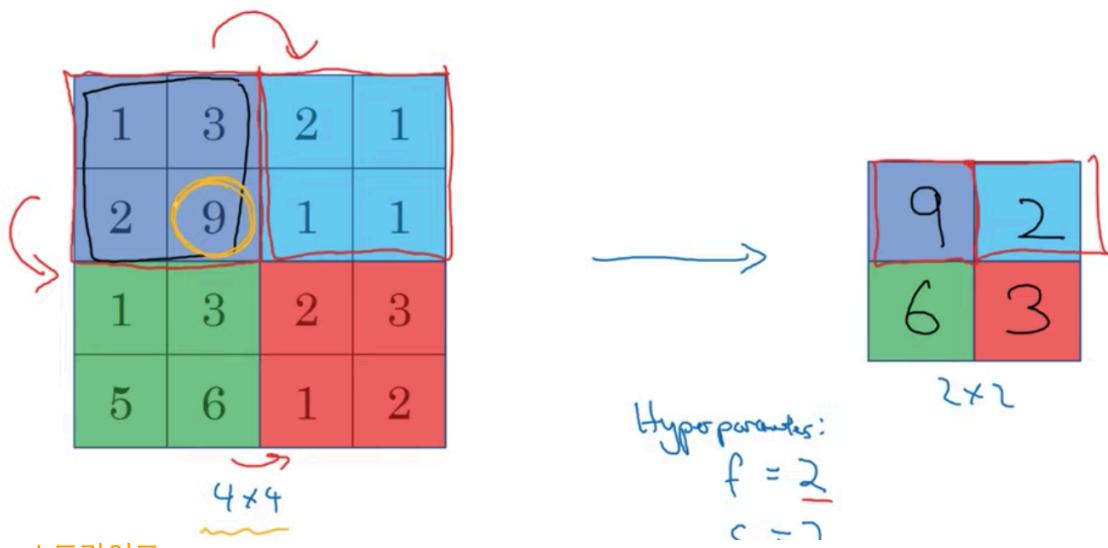
### (Pooling)

주위의 픽셀을 묶어서 하나의 대표 픽셀로 바꾼다.  
즉 이미지의 차원을 축소한다. (세로, 가로 방향의 공간을 줄인다)

합성곱 신경망은 풀링층을 사용하여 표현 크기를 줄임으로써 계산 속도를 높이고 특성을 훨씬 더 잘 검출할 수 있다.

표현의 크기를 줄이는 것은 2차원 매트릭스에서 특징적인 data만 뽑아내기 때문에 가능하다.

## Pooling layer: Max pooling



### 스트라이드

참고로 풀링의 윈도우 크기와 스트라이드는 같은 값으로 설정하는 것이 보통이다.  
( $3 \times 3$  이면 스트라이드도 3)

### 특징

학습할 파라미터가 없다.(즉 고정된 함수이다)

Pooling의 파라미터는  $f$ 와  $s$ 만 존재 ( $f$ 는 필터의 크기,  $s$ 는 스트라이드)

입력 채널의 개수와 출력채널의 개수가 같음(필터의 개수가 하나이기 때문에 당연)

### output

filter size를  $n$ 이라 했을 때,

입력의 폭과 높이가  $n$ 만큼 나뉘지는 효과가 있다.

( $m \times m$ ) 이미지가 input 되었을 때  $\rightarrow$  출력은  $(m/n, m/n)$  - 스트라이드가  $n$ 이라고 가정했을 때

결국 이미지의 크기가  $n^2$ 만큼 줄어든다.

### Max Pooling

가장 큰 수가 특정 특성을 의미할 수 있다는 IDEA

### Average Pooling

평균을 취함

### Residual Network(ResNet)

$a^{(l)}$ 에서  $a^{(l+2)}$ 까지 갈 때, short cut 혹은 skip connection을 연결하는 네트워크

$a^{(l)} \rightarrow [layer] \rightarrow [layer] \rightarrow a^{(l+2)}$ 에서 short cut 연결

즉,

$a^{(l)} \rightarrow \text{Linear} \rightarrow \text{Relu} \rightarrow (a^{(l+1)}) \rightarrow \text{Linear} \rightarrow \text{Relu} \rightarrow (a^{(l+2)})$  가 될때

$a^{(l)}$ 을 마지막 Linear 연산때 합하여 연결해줌

그래서,

$$a^{(l+2)} = g(z^{(l+2)} + a^{(l)})$$

$$z^{(l+2)} = w^{(l+2)}a^{(l+1)} + b^{(l+2)}$$

이때  $a^{(l)}$ 을 잔여블록이라 한다.

잔여블록을 사용하면 훨씬 깊은 신경망을 학습시킬수 있다.

ResNet을 구성하는 방법은 이러한 잔여블록들을 쌓아서 깊은 신경망을 만드는 것이다.

### Plain network에서

층의 개수를 늘릴 수록 훈련 오류는 감소하다가 다시 증가하는데, 이론 상으로는 신경망이 깊어질수록 training error가 감소해야한다.

하지만 실제로는 평형망의 깊이가 매우 깊다면 최적화 알고리즘으로 훈련을 하는것이 더 어려워 질테고, 너무 깊은 신경망을 선택하면 훈련 오류는 더 커진다.

-> 네트워크가 깊어질수록 훈련을 잘 하지 못할수 있음

하지만 ResNet은 이를 해결해서, 레이어가 깊어질수록 training error가 감소한다.

### 가정

$$a^{(l+2)} = g(z^{(l+2)} + a^{(l)})$$
에서

$z^{(l+2)}$  와  $a^{(l)}$ 이 같은 차원을 가진다고 가정한다.

만약  $a^{(l)}$ 과  $z^{(l+2)}$ 의 차원이 같지 않다면 a앞에 가중치를 붙여 같게 만들어준다.

$$\Rightarrow a^{(l+2)} = g(z^{(l+2)} + W_{sa}^{(l)})$$

## 전이 학습(Transfer Learning)

다른 이들이 훈련시켜놓은 구조를 다운 받는다.

그것을 이용해서 새로운 관심있는 작업에 전달

누군가가 오랜 기간 걸쳐 얻은 오픈소스를 사용하면 신경망의 좋은 초기 자료로 사용할수 있다.

전이 학습을 이용해서 공유된 데이터 세트를 통해 얻을 지식을 우리의 문제에 전달

소프트맥스층만 재구성하면됨.. 소프트맥스 층과 관련된 변수만 훈련시킨다.(이전 층의 가중치는 동결)

훈련시키지 않기 때문에 하나의 고정함수가 발생해서 이미지 X를 특정 활성값으로 전달한다.

훈련 속도를 높이는 방법은 hidden 층을 미리 계산하면됨

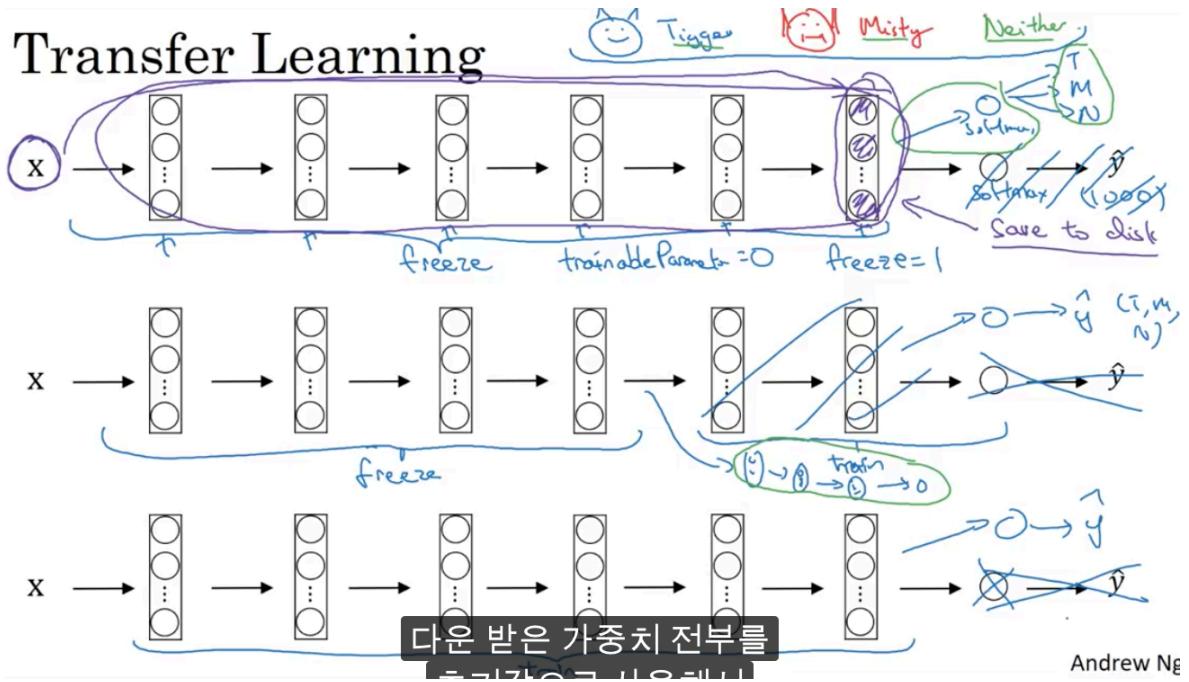
활성값의 특성을 디스크에 미리 저장해놓으면 된다.(save to disk)

모든 훈련 세트의 활성값을 저장하면 소프트맥스만 훈련하면 된다.

혹은 몇개의 층만 동결(freeze)시키고 이후의 층들은 훈련시켜줌. 출력층은 다르기 때문에 바꿔줌 데이터가 많을 수록 동결시키는 층의 개수는 줄어들고, 훈련시킬 층의 개수는 늘어나게 된다.

충분한 데이터가 있다면 하나의 소프트맥스 유닛만 훈련하는게아니라 마지막 몇개의 층을 조합한 작은 신경망을 만들수 있다.

아주 많은 데이터가 있다면 네트워크 전체를 학습시키면됨



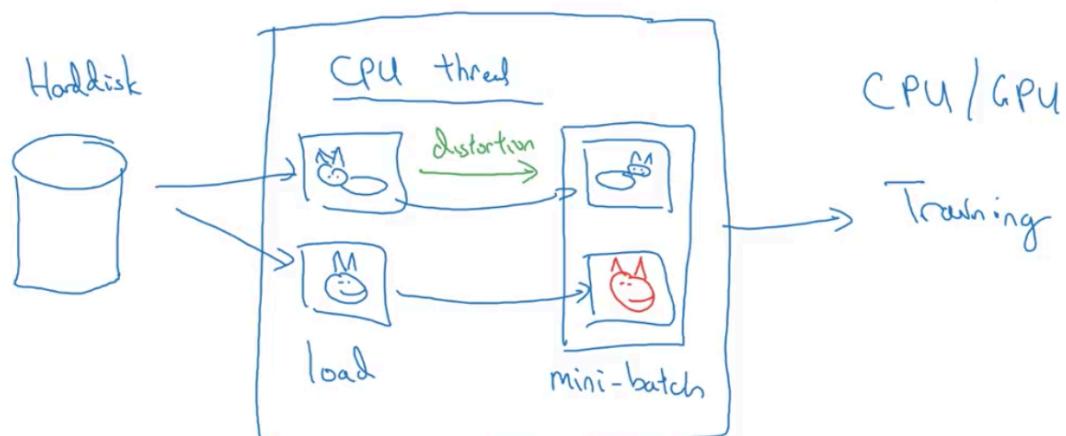
### Data Argument

더 많은 데이터가 거의 모든 컴퓨터 비전 작업에 도움을 준다.

데이터 확대도 몇개의 파라미터를 가짐 (얼마만큼 색 변환을 할것인지, random cropping은 어떻게 할것인지)

학습하는 동안 변형시키기

## Implementing distortions during training



대표적인 것

Mirroring

양옆으로 뒤집는 것

Random Cropping

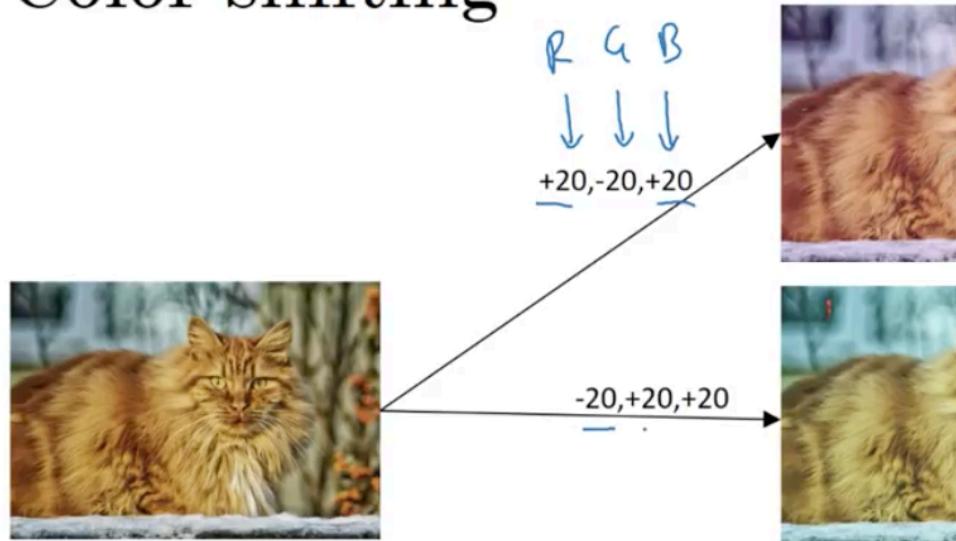
무작위로 잘라내는 것

color shifting

color에 변화를 주는 것

색변형을 통해 학습 알고리즘이 색 변화에 더 잘 반응할 수 있게 해준다.

# Color shifting



Advanced

PCA Color Argumentation

빨강과 파랑에 큰 수를 더하거나 빼고 초록색에는 상대적으로 적게해줌으로써 전체적인 색조를 유지해준다.

자주 사용하지 않는 것

복잡하기 때문에 자주 사용하지 않음

Rotation

Shearing

Local warping

Data vs. hand-engineering

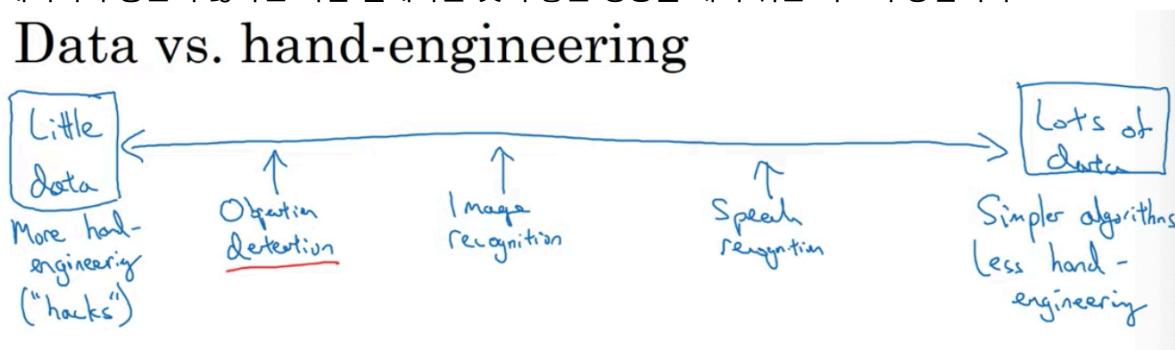
평균적으로 데이터가 많다면 간단한 알고리즘을 사용하고 수작업이 적은 경향이 있다.

특성들을 너무 조심스럽게 디자인 할 필요가 없고, 네트워크가 간단해짐

반면에 데이터가 충분히 많지 않다면 평균적으로 수작업 설계가 많아지게 되고 안 좋게 말하면 핵이 많아진다.

데이터가 충분치 않다면 직접 설계하는 것이 좋은 성능을 내기 위한 최고의 방법이다.

## Data vs. hand-engineering



컴퓨터 비전 분야에서 더 복잡한 네트워크 구조가 개발 된 이유는 데이터의 부재 때문이다.

더 좋은 성능을 내려면 구조설계에 공을 들여야 한다.

직접 설계는 굉장히 까다로운 작업이고 많은 영감을 필요로 한다.

물체인식은 이미지 인식보다 더 작은 데이터셋을 가지고 있어서 물체 인식 분야에서는 훨씬 더 복

잡하다.

데이터가 적을때 큰 도움이 되는 것이 transfer learning이다.

### Two sources of knowledge

#### Labeled data

(x,y) 쌍

Hand engineered features/network architecture/other components

직접 특성을 설계 하는것, 직접 네트워크 구조를 설계하는것, 시스템의 다른 요소를 설계하는 것

레이블데이터가 충분하지 않다면 직접 설계에 의존해야 한다.

#### 벤치마킹이 잘되게 하는 방법

#### Ensembling

어떤 신경망을 사용할지 결정한후 몇개의 신경망을 독립적으로 훈련시킨후 평균을 내는것

- Train several networks independently and average their outputs

가중치가 아닌 Y의 평균값을 내야 한다.

3개 ~ 15개의 네트워크를 사용하는것이 일반적

#### 문제점

여러개의 네트워크를 계속 유지해야 하기 때문에 많은 양의 메모리를 필요로 한다.

#### Multi-crop at test time

Run classifier on multiple versions of test images and average results

이미지를 자르고 각각의 이미지를 분류기에 넣어준다음 평균을 함

#### 문제점

다중 크로핑은 하나의 네트워크만 유지해도 되서 메모리를 많이 차지하지는 않지만 실행시간을 꽤나 늦춘다.

## Object Detection

한 이미지에 여러 개의 물체를 가질수 있음

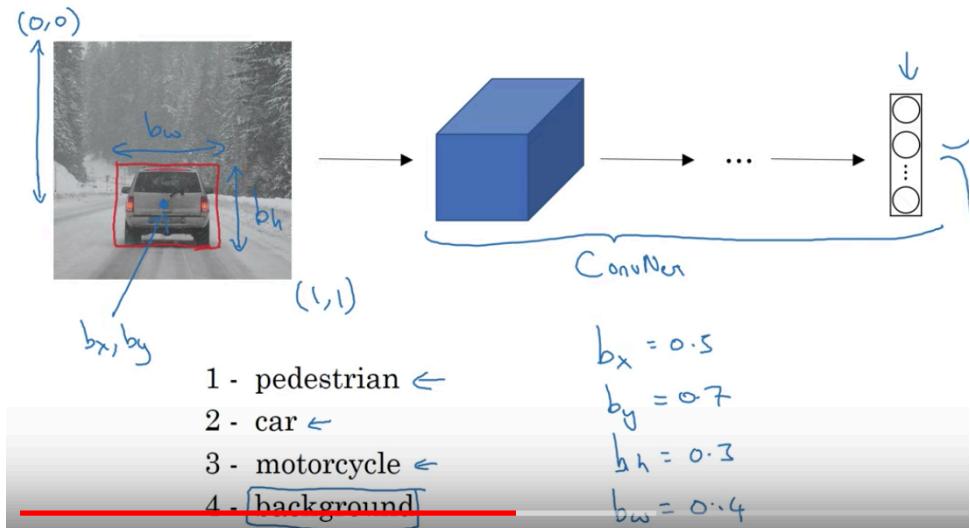
### Localization

감지된 것이 사진의 어느쪽에 위치하는지를 설명하는 것

softmax 출력층에서 분류를 하고

다른 곳에서 box의 좌표를 반환함

# Classification with localization



지도 학습 작업에서 어떻게 목표 레이블  $y$ 를 정의할 것인가?

이미지가 단 하나의 물체만을 가진다고 가정할 경우

$p_c$ : 물체가 있는지 나타내는 확률(감지하려는 물체의 분류에 대한 확률)

$b_w$ : 경계상자를 표현

$b_y$ : 경계상자를 표현

$b_h$ : 경계상자를 표현

$b_w$ : 경계상자를 표현

$c_1$ : 클래스1

$c_2$ : 클래스2

$c_3$ : 클래스3

$c_4$ : 클래스4

## Defining the target label $y$

- 1 - pedestrian
- 2 - car
- 3 - motorcycle
- 4 - background

Need to output  $b_x, b_y, b_h, b_w, \text{class}$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \text{is there any object?}$$



$(x, y)$

그림 왼쪽 - 물체를 가진 경우  $[1 \ bx \ by \ bh \ bw \ 0 \ 1 \ 0]$

그림 오른쪽 - 물체를 가지지 않은 경우  $(0 \ ? \ ? \ ? \ ? \ ? \ ? \ ?)$

### 손실함수

$y$ 가 벡터이기 때문에, 다음과 같이 표현할 수 있음

$y_{10} = 1$ 인 경우는 즉 물체가 존재하는 경우이다.  $\rightarrow$   $y$  벡터의 요소를 고려해 손실함수를 만들

$y_{10} = 0$ 인 경우는 물체가 존재하지 않는 경우이다.

$y_{10} = 0$ 인 경우 다른 요소가 무관항이므로 계산하지 않는다.

$$l(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_{10} = 1 \\ (\hat{y}_{10} - y_{10})^2 & \text{if } y_{10} = 0 \end{cases}$$

### Landmark Detection

일반적으로 신경망은 xy좌표를 가지는데, 이미지에서 주로 특징점이라고 불리는 주요한 지점을 나타낸다.

특징점을 포함하는 레이블 훈련세트를 만들면 신경망이 어디에 특징점들이 있는지 말할 수 있도록 할 수 있다.

이러한 신경망을 훈련하기 위해서 레이블 훈련 세트가 필요하다.

여러 개의 출력 유닛을 추가해서 인식하고자 하는 특징점들의 각 좌표를 출력  
이것을 확실히 하기 위해서 특징점은 다른 이미지에서도 동일해야 한다.

특징점1은 항상 눈꼬리가 되어야 하고

특징점2는 다른쪽 눈꼬리가 되어야 함

### Sliding windows detection

다음의 트레이닝셋 준비 - 자동차가 명확인 트레이닝 데이터

Training set:

X                    y



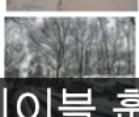
1



1



1



0

이 레이블 훈련 세트를  
합성곱 신경망을 훈련할 수  
있도록 했습니다.

0

수많은 잘려진 이미지를 합성곱 신경망에 통과시키는 것이 각 위치에 대해서 0 또는 1의 값  
으로 분류된다.

윈도우를 계속 옮겨가며 합성곱 신경망에 통과시킴

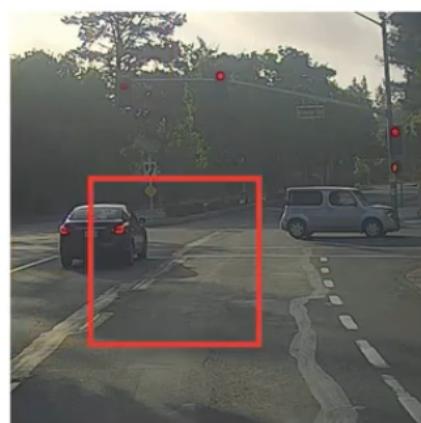
→ ConvNet → 0



그리고 약간 더 큰 윈도우를 사용해 계산 수행



더큰 윈도우로 계산 수행



사각형 상자 윈도우를 이용해 전체 이미지에 대해 슬라이드 시키고 모든 사각형 영역에 대해서 자동차가 포함되는지 분류

슬라이딩 윈도 검출에는 큰 단점이 존재한다.

바로 계산 비용인데, 이미지의 수 많은 영역을 모두 잘라내야 하고 그리고 합성곱 신경망을 통해 이것을 각각 계산해야한다.

매우 큰 슬라이드 간격을 사용한다면, 합성곱 신경망을 통과시켜야하는 윈도우의 수는 줄어들지만 성능을 저하시킬수 있다.

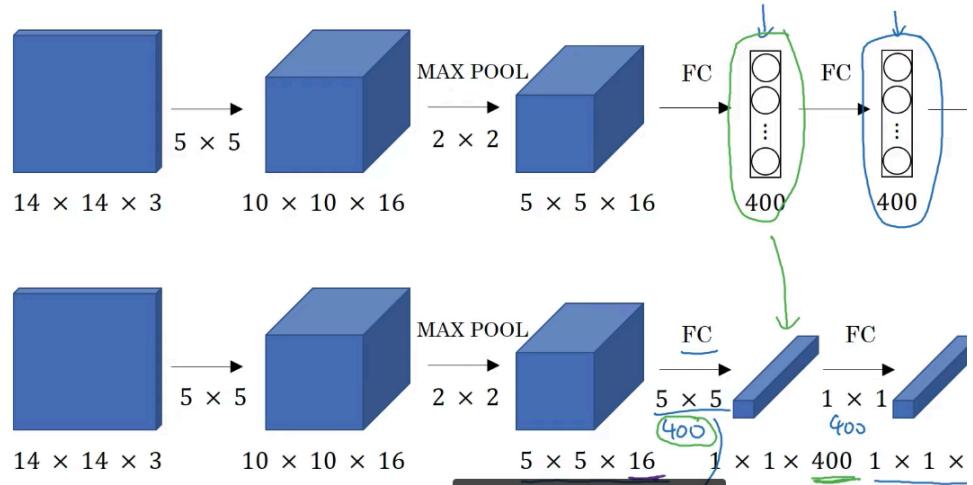
매우 정밀한 입도를 사용하면 작은 영역을 사용하기 때문에 계산비용이 크다.

계산 비용 문제는 해결방법이 있다. 슬라이딩 윈도 물체인식기를 합성곱을 사용해 훨씬 효율적으로 구현할수 있다.

**Turning FC layer into convolutional layers**

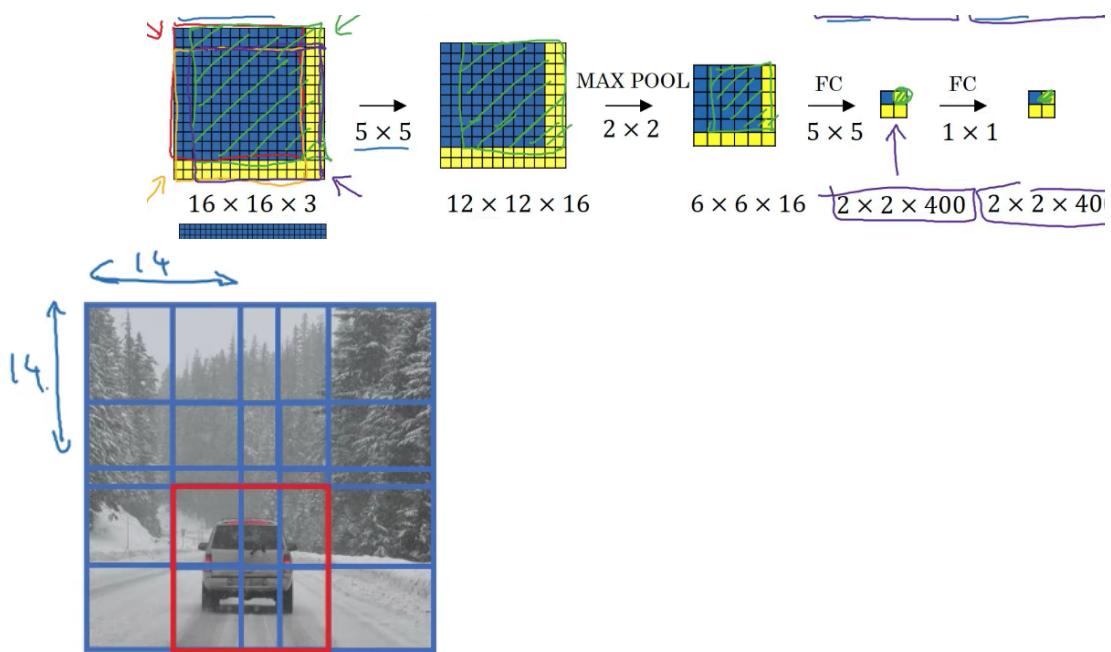
완전 연결 층을 합성곱 층으로 구현하는 과정

## Turning FC layer into convolutional 1



### Convolution implementation of sliding windows

독립적인 정방향 전파를 수행하는 대신 네가지 경우를 한지 계산으로 통합하는 것  
공통적 영역에 대한 계산을 공유하는 것



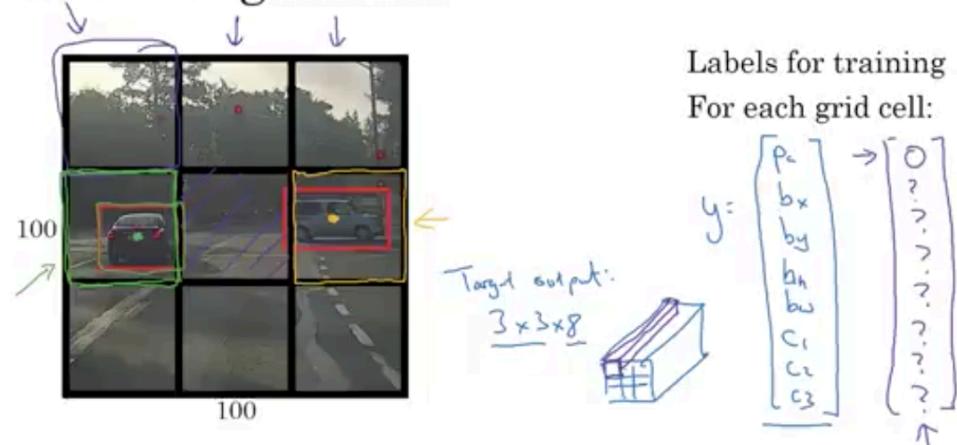
### Neural Networks Bounding Box Predictions

경계가 명확하지 않은 문제 해결

Output accurate bounding boxes

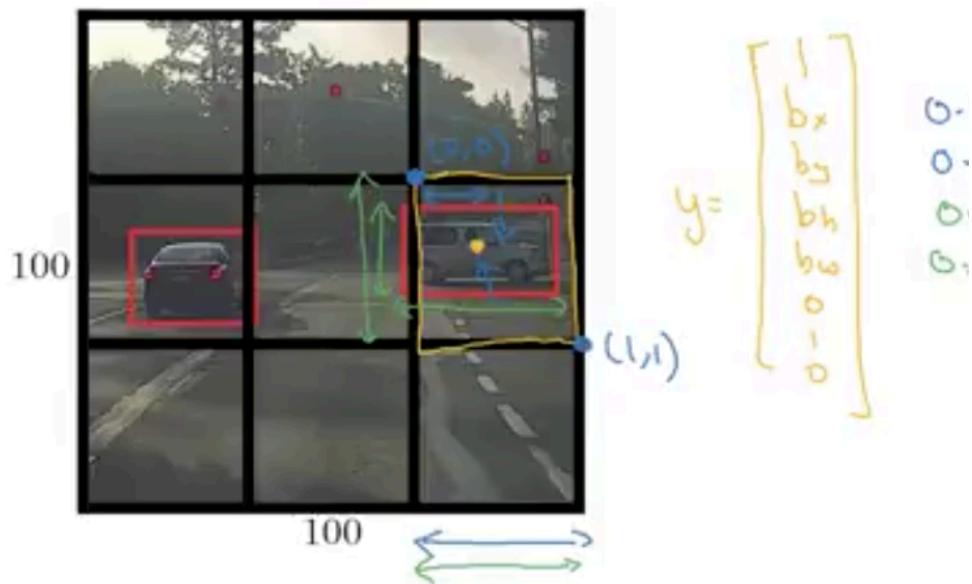
Yolo algorithm

## YOLO algorithm



specify the bounding boxes

## Specify the bounding boxe



합집합 위의 교집합(Intersection over union, IOU)

IOU는 두 경계 상자의 겹침의 측정값이다.

물체 감지 알고리즘이 잘 동작하는지 어떻게 알수있을까?

물체 감지 태스크에서 개체를 지역화해야한다.

합집합 위의 교집합이란 교집합의 크기를 계산해서 이것을 합집합의 크기로 나눈 것이다.

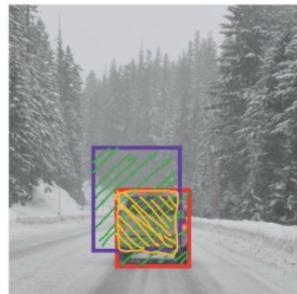
IOU가 0.5보다 크면 맞다고 판단한다.

만약 예측값과 참 값 경계 상자가 완전히 겹친다면 IoU는 1이다.

관례상 0.5는 경계상자가 맞는지 틀렸는지에 대한 임계값으로 사용된다.

IOU가 높을수록 경계상자는 더 명확하다.

# Evaluating object localization



$$\text{Intersection over Union} = \frac{\text{Size of intersection}}{\text{Size of union}}$$

"Correct" if  $\text{IoU} \geq 0.5$  ←

## Non-max suppression

확률의 최대값을 도출하고, 최댓값이 아닌 것들은 억제한다는 의미이다.

각 물체를 한번씩만 감지하게 보장한다.

감지된 것을 정리한다.

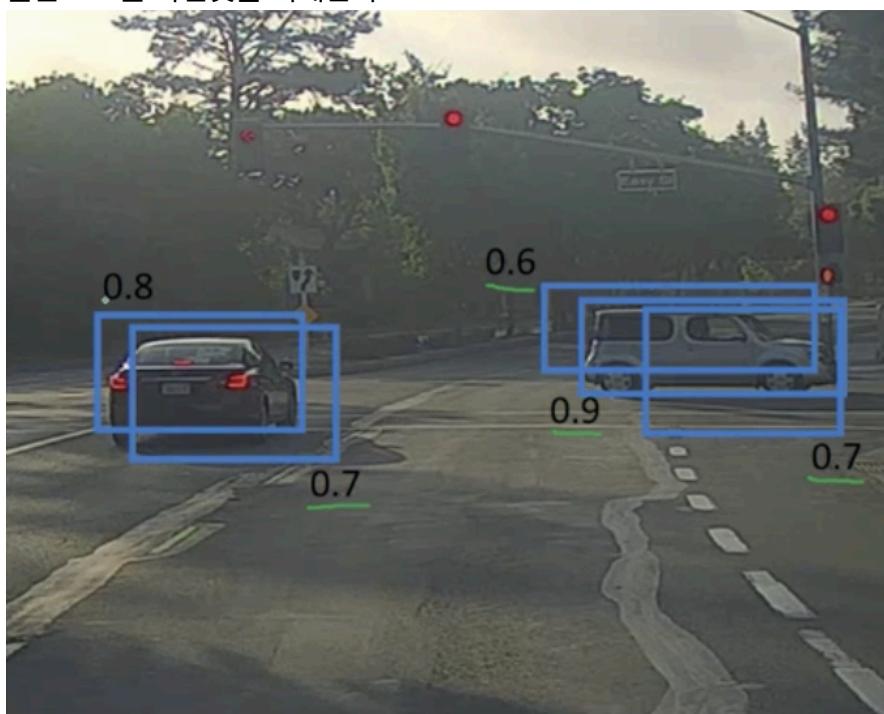
먼저 각 감지의 확률을 살핀다.

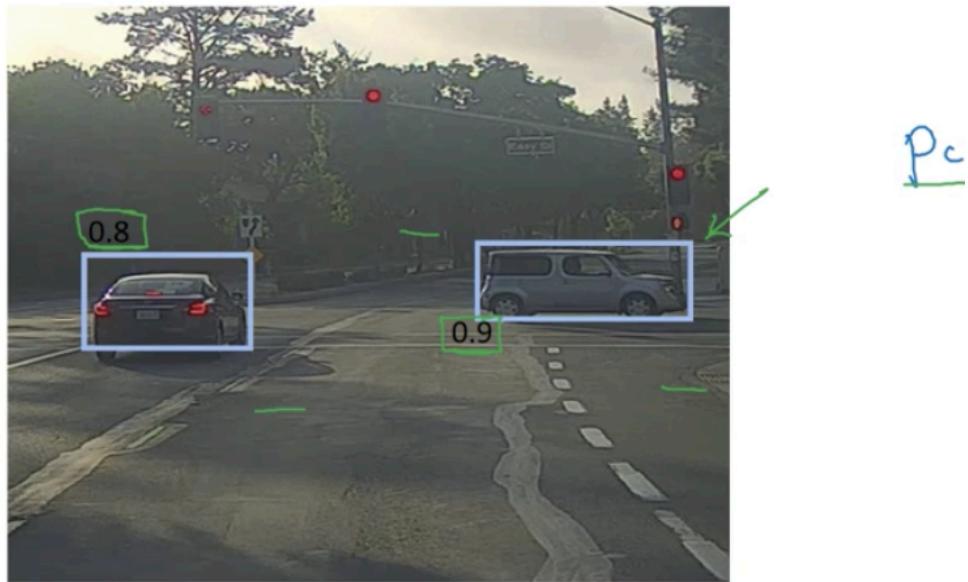
감지확률을  $P_c$ 라 할때, 먼저 가장 큰 값을 고르고 강조 표시를 한다.

Non-max suppression에서 나머지 직사각형 검사해서 그 직사각형과 많이 겹치는 즉 IOU 가 높은 직사각형들을 억제시킨다.

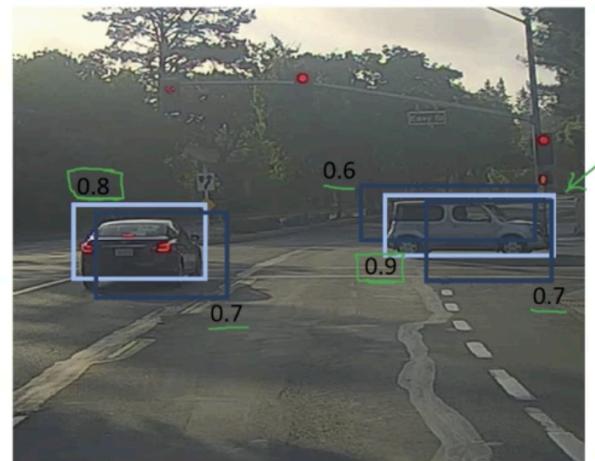
다음으로 나머지 직사각형 중 가장 높은 확률을 찾는다.(강조 표시를 함)

높은 IOU를 가진것을 억제한다.





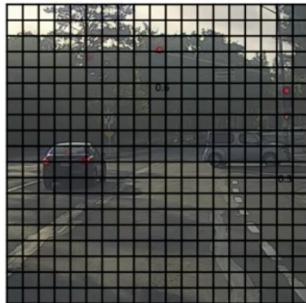
## Non-max suppression example



### algorithm

만약 클래스가 1개가 아니라 많아지면 결과 벡터는 추가적인 요소를 가질것이고 각각의 결과 클래스에 대해  
독립적으로 세번의 Non-max suppression을 해야 한다.

# Non-max suppression algorithm



19 × 19

밝은 색으로 강조했던  
경계 상자와 많이 겹치는 것을

Each output prediction is:

$p$   
 $b$   
 $b$   
 $b$   
 $b$

Discard all boxes with  $p_c \leq 0$ .

While there are any remaining:

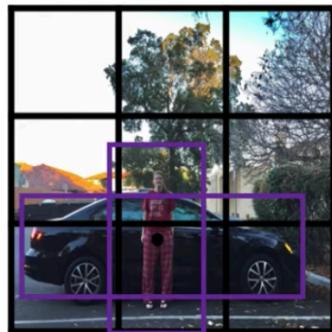
- Pick the box with the highest  $p_c$ . Output that as a prediction.
- Discard any remaining boxes with  $\text{IoU} \geq 0.5$  with the box picked in the previous step.

## Anchor boxes

각각의 격자 셀이 오직 하나의 물체만 감지할 수 있느냐?

만약 격자 셀이 여러 개의 물체를 감지하고 싶다면 앵커 박스를 사용하면된다.

## Overlapping objects:

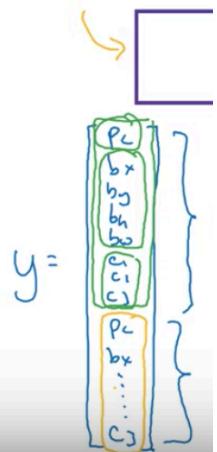


$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Anchor box 1:



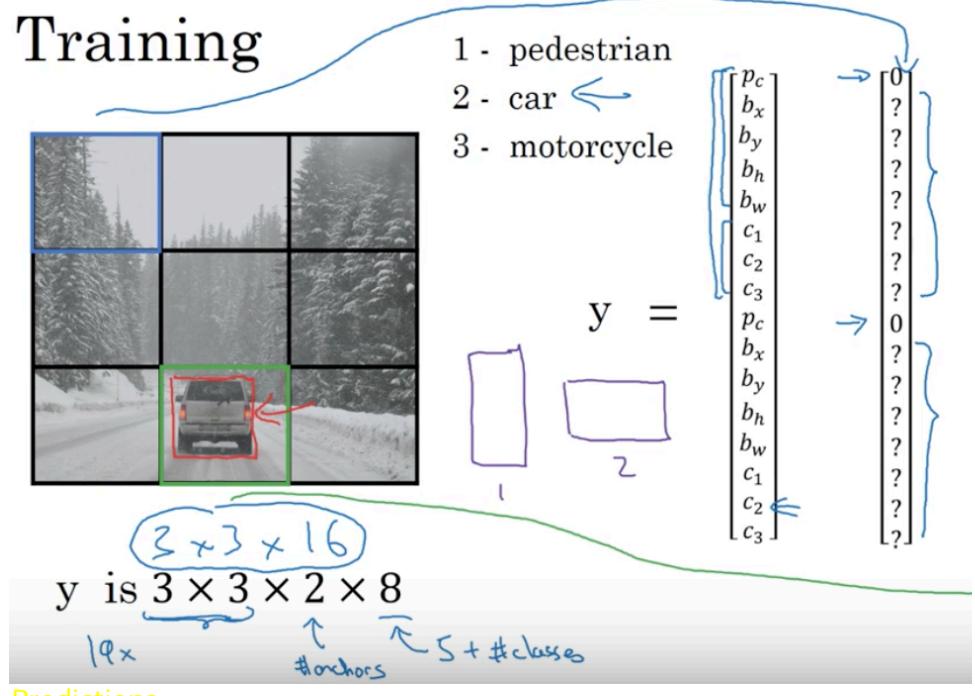
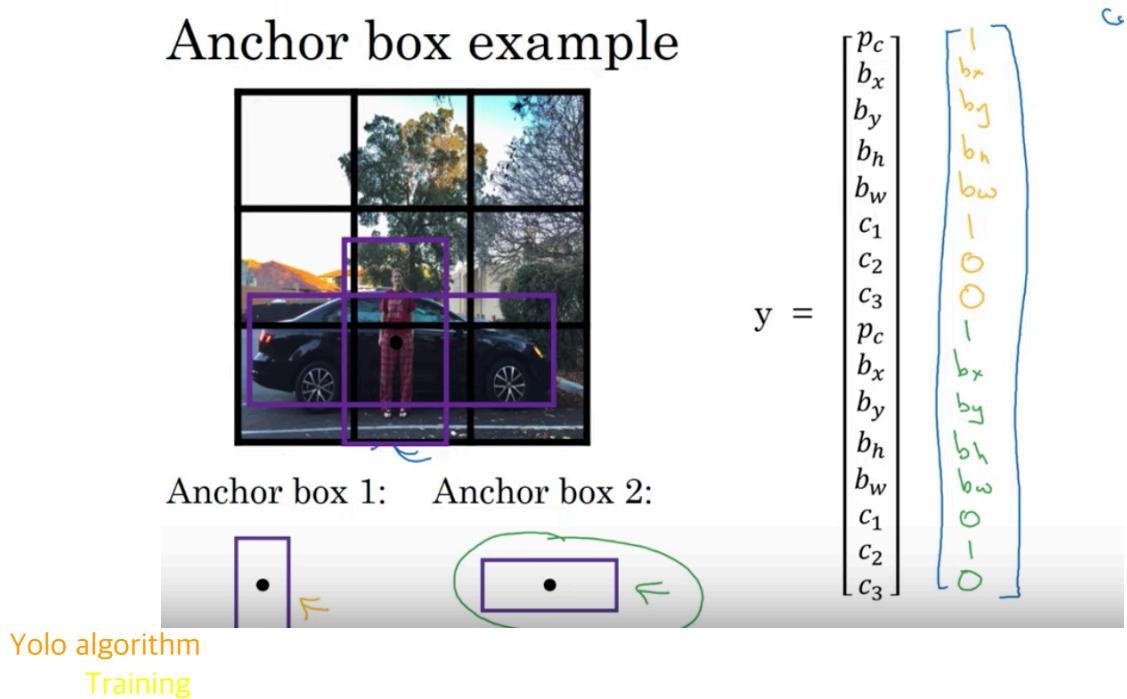
Anchor



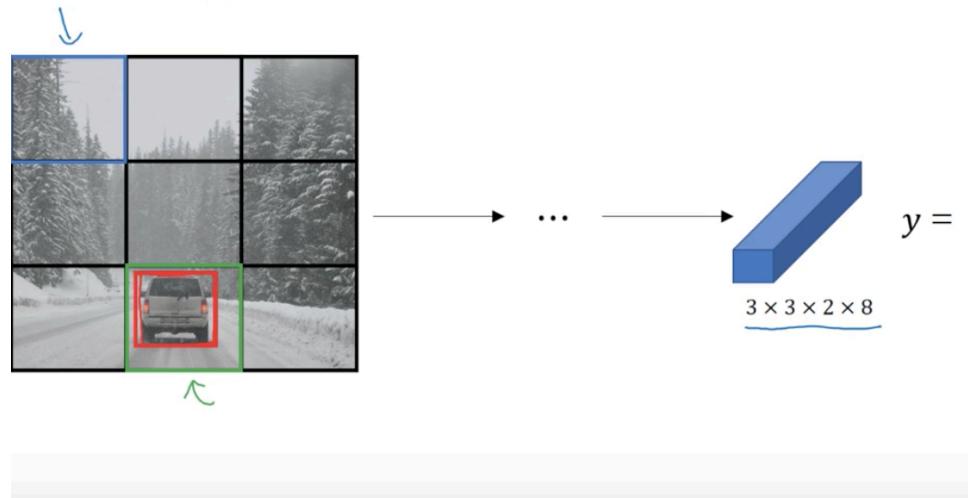
앵커 박스를 사용하기 전에는 훈련 세트 사진에 있는 각 물체들은 물체의 중심점이 있는 격자 셀에 배정되었다.

앵커 박스를 가지고는, 각 물체는 이전과 같이 중심점이 있는 셀에 배정되지만 물체의 모양과 가장 높은 IOU를 가지는 격자 셀과 앵커 박스에 배정된다.

## Anchor box example



# Making predictions



outputting the non-max suppressed outputs

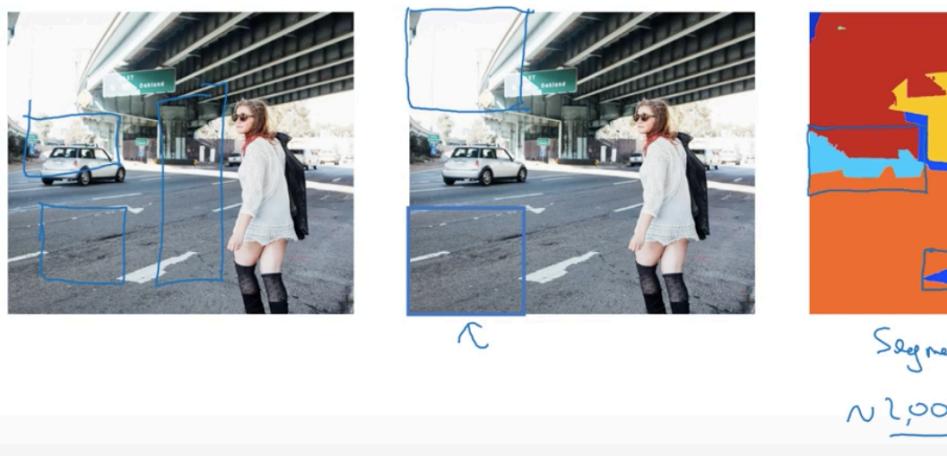


- For each grid cell, get 2 predict boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, ...), use non-max suppression to get final predictions.

Region Proposal

RCNN

특정한 지역만 object detection을 하자는 아이디어



Segment

$\sim 2,000$

⇒ R-CNN: Propose regions. Classify proposed regions in parallel. Output label + bounding box. ↵

Fast R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions. ↵

Faster R-CNN: Use convolutional network to propose regions. ↵

### 필터(Filter)

특정한 특징만 검출하기 위한 방법

## Vertical edge detection

$3 \times 1, 1 \times 1 + 2 \times 1$

3	0	1	2	7	4
-1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$6 \times 6$

"convolution"

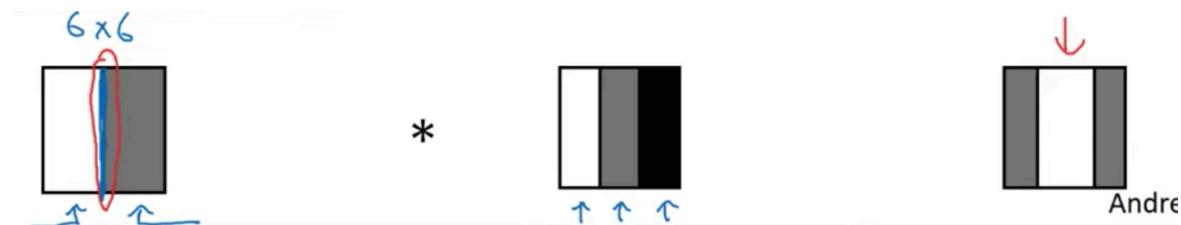
$*$

1	0	-1
1	0	-1
1	0	-1

$3 \times 3$   
filter

=


$4 \times 4$



### convolutional network

#### notation

$I$ : 층

$f^I(l)$ : filter size

$p^I(l)$ : padding

$s^I(l)$ : stride

$m$ : number of data

$n_c^I(l)$ : number of filter

$n_H^I(l)$ :  $l$ 층에서의 input의 높이:  $n_H^{I-1} + 2p^I(l) - f^I(l) / s^I(l) + 1$

$n_W^I(l)$ :  $l$ 층에서의 input의 폭:  $n_W^{I-1} + 2p^I(l) - f^I(l) / s^I(l) + 1$

Input 차원:  $n_H^I(l-1) \times n_W^I(l-1) \times n_c^I(l-1)$

Output 차원:  $n_H(l) \times n_w(l) \times n_c(l)$   
 $a^l(l)$ : l층에서의 필터의 출력(l층의 뉴런들의 출력) 차원:  $n_H^l(l) \times n_W^l(l) \times n_c^l(l)$   
 $A^l(l)$ : 전체 데이터를 고려한 출력 차원:  $m \times n_H^l(l) \times n_W^l(l) \times n_c^l(l)$   
weights의 개수:  $f^l(l) \times f^l(l) \times n_c^{l-1} \times n_c^l(l) <- \text{input 채널의 개수} * \text{필터의 크기} * \text{필터의 개수}$

bias의 개수:  $n_c^l(l)$

### 총 파라미터의 개수

\* 필터의 높이 \* 필터의 폭 \* 필터 채널의 개수 + bias의 개수(1개)

### Type of layer in a convolutional Network

- convolution (Conv)
- Pooling (Pool)
- Fully Connected (FC)

### Why convolution is efficient?

convolution을 사용하지 않고, fully connected 를 사용한다면, 입력 개수 \* 출력 개수 만큼 파라미터의 개수가 늘어날 것이다.

그런데 convolution을 사용한다면 파라미터를 공유하기 때문에 더 적은 파라미터를 사용할 수 있다.(parameter sharing)

또한 convolution을 사용할 때, window에 해당되는 matrix의 요소만 신경쓰면 된다. 이를 희소 연결(sparsity of connection)이라 한다.

희소 연결(sparsity of connection)

### Network in Network, one by one convolution

1x1 필터를 convolution 하는 것을 의미

만약 채널의 개수가 1개보다 커지면 1x1xf\_c 로 적용

유용성

1x1 convolution으로 채널의 수를 줄여, 네트워크 안의 계산을 용이하게 할 수 있다.

만약

28x28x192라는 입력이 있다고 하면 여기에 1x1x192 필터를 32개 적용시키면  
28x28x32의 출력이 나타남

### Inception Network

인셉션 네트워크는 필터의 크기를 결정하지 않고 합성곱 또는 폴링층을 모두 사용하는 것이다.

인셉션 네트워크의 input에서 특정한 크기(높이, 폭)를 입력하면 출력의 채널수를 모두 더한 값은 높이 x 폭이 됨

## YOLO(You only look once)

YOLO는 실시간 개체 검출 시스템을 수행하는 최첨단 기술이다.

타이탄 X에서 30 FPS로 이미지를 처리한다 그리고 COCO에 대해서 57.9%의 mAP 평가편차 (test-dev)를 가진다.

### Comparison to Other Detectors

YOLOv3은 빠르고 정확하다.

0.5 IOU로 측정한 mAP에서 Yolo v3은 Focal Loss와 동등하지만 4배 더 빠릅니다.

### 작동 원리

종래의 검출 시스템은 classifier와 localizer를 사용하여 detection을 수행합니다.

이들은 모델을 여러 location 및 scale로 이미지에 적용합니다.

이미지의 높은 점수 영역을 detection으로 간주합니다.

YOLO는 다른 접근법을 사용합니다.

하나의 neural network를 전체 이미지에 적용합니다.

이 network는 이미지를 region으로 나누고 각 영역의 bounding boxes와 확률을 예측합니다.

이러한 predicted probabilities로 경계 상자에 가중치를 줍니다.

YOLO는 classifier 기반 시스템에 비해 몇 가지 장점이 있습니다.

테스트 시간에 전체 이미지를 확인하여 이미지의 글로벌 컨텍스트로 예측을 합니다.

yolo는 기본적으로 yolo 저자가 만든 darknet이라는 C기반 딥러닝 프레임워크를 사용한다.  
yolo의 config 파일을 읽어서 C기반 cuda로 export해주는 pytorch로 모델을 생성한다.

### 코딩시작

darknet.py: YOLO의 전반적인 구조를 코딩할 파일

util.py: 여러 도움을 줄 함수들

### Configuration file

cfg파일은 네트워크의 layout을 block 단위로 정의해놓은 파일이다.

official cfg file을 사용한다.

이 파일의 각 괄호들은 하나의 네트워크적인 의미를 갖고 있다.

ex)

--> 3개의 convolution layer와 1개의 shortcut을 나타낸다.

[convolutional]

batch\_normalize=1

filters=64

size=3

stride=2

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=32

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch\_normalize=1

filters=64

size=3

stride=1

pad=1

activation=leaky

[shortcut]

from=-3

activation=linear

shortcut

shortcut은 ResNet등에서 나온 skip connection과 유사하다.

form = -3 이라는 뜻은 shortcut layer의 output에 3번째 전의 layer를 더해준다는 뜻이다.

기존의 skip connection과 일치한다.

#### convolutional

필터 수는 몇개인지

batch\_norm을 할것인지

필터수는 몇개인지

stride는 몇인지

padding은 얼마를 주는지

activation function은 무엇을 쓰는지

#### upsample

feature map을 stride 배수에 따라 unsample하는 layer이다.

ex)

[upsample]

stride=2

#### route

router layer의 layers라는 속성은 한개 혹은 두개의 value를 가진다.

layers 속성에 한개 value만 있을때는 그 value의 index에 해당하는 feature map의 value만을 output으로 내보낸다.

layers 속성에 한개의 value만 있을 때는 그 value의 index에 해당하는 feature map의 value만을 output으로 내보낸다.

위의 예시 -4의 경우, 4번째 이전의 layer의 output layer를 내보낸다고 할수있다.

만약 두개의 value가 있을때는, 그 value들의 layer들의 feature map을 이어붙여서(concatenate) 내보낸다.

ex)

[route]

layers = -4

#### [route]

layers = -1, 61 <- depth dimension의 직전 layer(-1)과 61번째 layer를 내보내는 코드이다.

#### YOLO

YOLO layer는 이미지의 feature들을 뽑고 난 후에 실질적인 prediction을 하는 layer이다.

ex)

[yolo]

mask = 0,1,2

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=80

num=9

jitter=.3

ignore\_thresh = .5

truth\_thresh = 1

random=1

#### anchor

bounding box의 width와 height를 예측하여 정답을 맞추는것이 자연스러워 보이지만,  
실제로 돌려보면 unstable gradient를 발생시키는 원인이 된다.  
그래서 대신 anchor라고 하는 pre-defined default bounding box를 사용한다.  
그렇지 않은 경우 log-space transforms라는 걸 사용하기도 한다.  
**mask**  
위에는 총 9개의 anchor이 정의되어 있는데, 그 중 mask에 적혀있는 tag에 해당하는 anchor들만 사용한다.

## 데이터(data)

### 트레이닝 데이터(Training data)

Training 과정에서 사용하는 데이터

### 테스트

모델의 학습 정도를 테스트

테스트 할때, 테스트 시점에 따라 데이터를 두가지 경우로 나눌수 있다.

1. 학습 중간에 테스트 하는 경우
2. 학습이 끝난후 테스트 하는 경우

1번의 경우는 model이 training 데이터에 overfitting 되는지를 체크하는 테스트이며

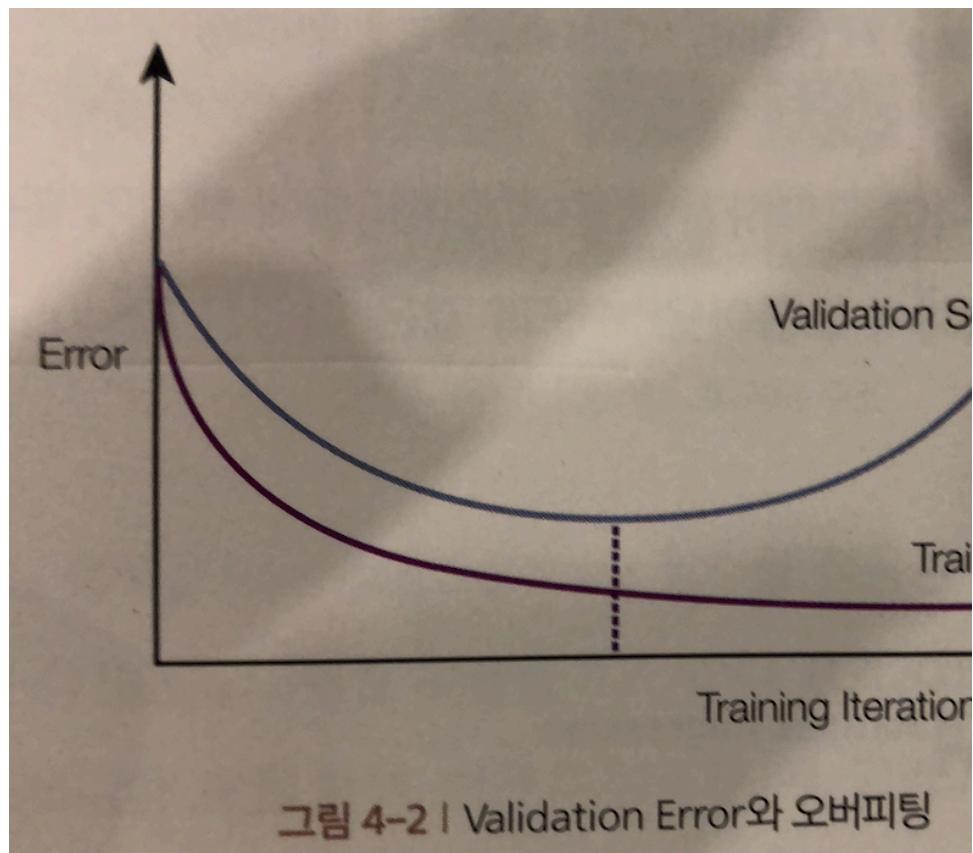
2번의 경우는 model의 추론 정도를 체크하는 테스트이다.

### 테스트 데이터(test data)

학습이 끝난후, 모델의 추론 정도를 테스트하는 데이터

### 검증용 데이터(valid data)

학습에는 사용하지 않지만 중간중간 테스트 하는데 사용하는 데이터(오버피팅에 빠지지 않았는지 체크한다)



처음에는 트레이닝 에러와 검증 에러가 모두 작아지지만 일정 횟수 이상 반복할 경우  
트레이닝 에러는 작아지지만 검증 에러는 커지는 오버피팅에 빠지게 된다.  
따라서 트레이닝 에러는 작아지지만 검증에러는 커지는 지점에서 업데이트를 중지하  
면 최적의 파라미터를 얻을 수 있다.

#### 속성(attribute)

데이터의 데이터 타입을 의미

#### 특성(feature)

데이터의 데이터 타입과 값을 동시에 의미

#### 분할(partition)

테스트 데이터셋과 트레이닝 데이터셋을 나누는것

#### random하게 분할

랜덤하게 분할 할수도 있음

#### 해쉬값 계산을 통한 분할

만약 식별자가 존재한다면 이에 대한 해쉬 계산을 통해 분할할수도 있다.

#### 수치 미분(numerical differentiation)

미분은 한순간의 변화량을 표시한것이다.

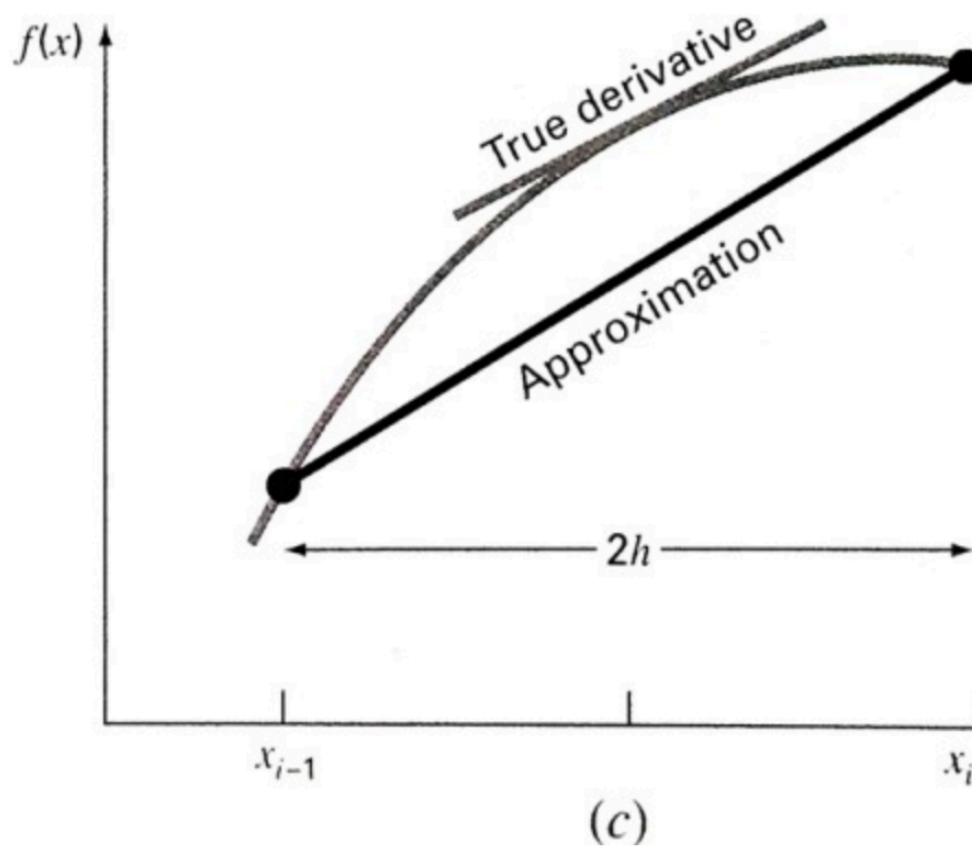
이때,  $x$ 에 대한 변화량을  $h$ 로 표현하면서  $h$ 가 0에 다가가는 극한의 개념을 사용한다. 이  $h$ 를 직접  
지정하는 것이 수치 미분이다.

그런데 이  $h$ 를 무한히 0으로 줍히는 것이 불가능하다.

이에 따라 수치 미분에는 “오차”가 반드시 포함된다.

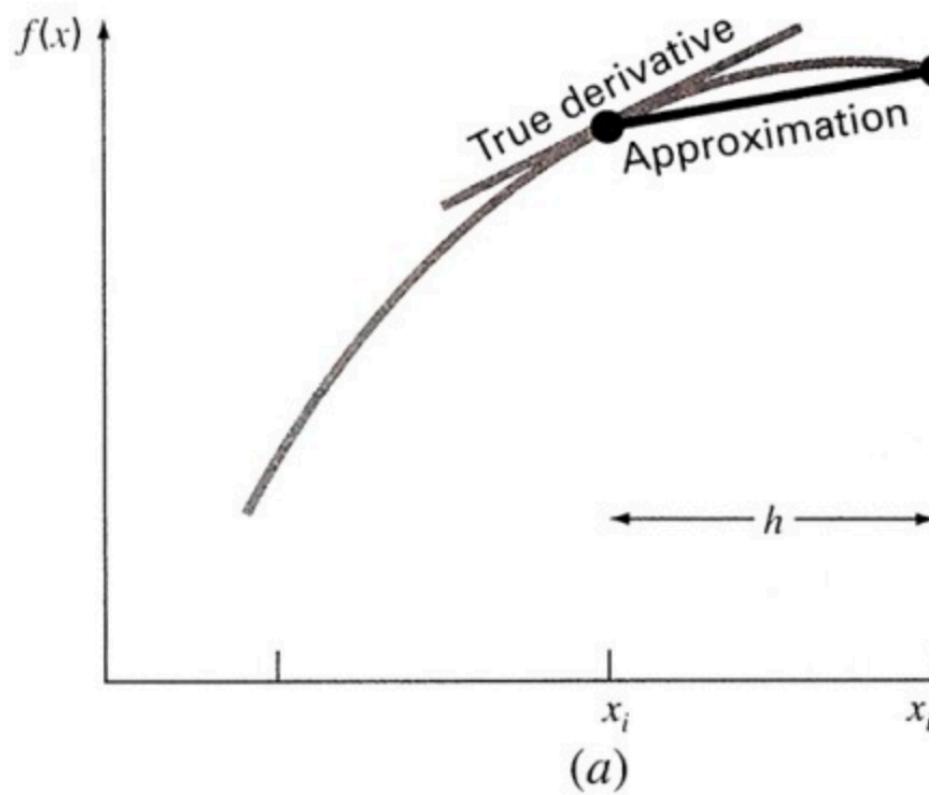
#### 중앙 차분(Centered Divided Difference)

$$f'(x) \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2 \Delta x}$$



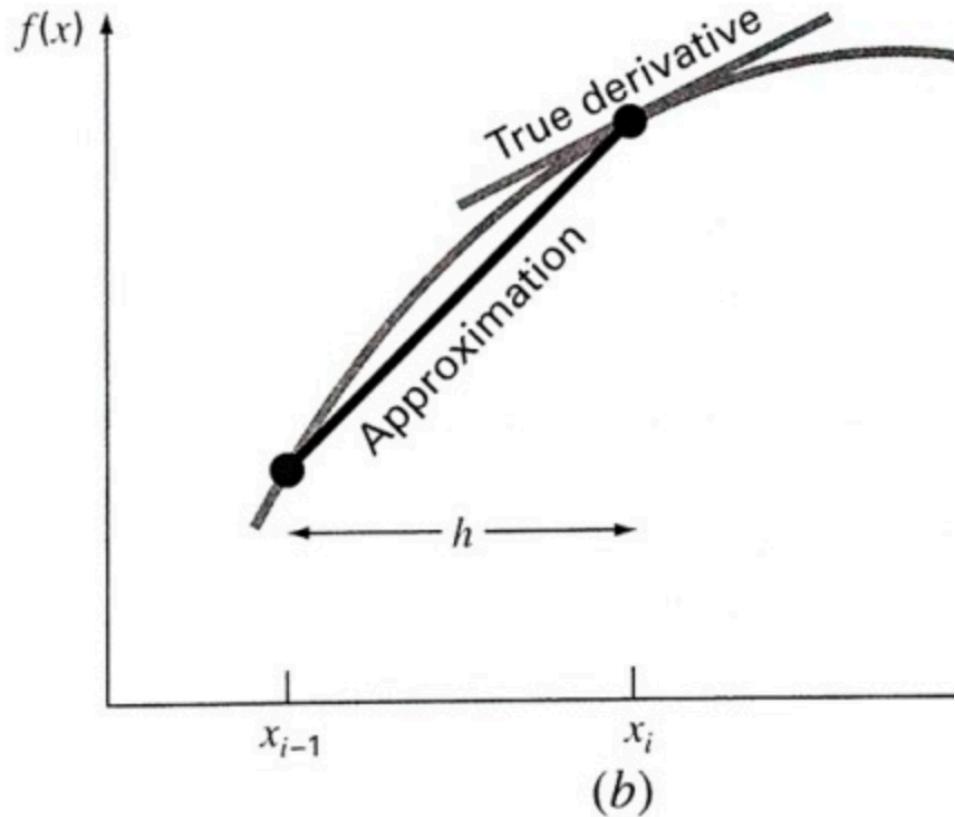
전진 차분(Forward Divided)

$$f'(x) \simeq \frac{f(x+\Delta x) - f(x)}{\Delta x}$$



후진차분(Backward Divided)

$$f'(x) \approx \frac{f(x) - f(x - \Delta x)}{\Delta x}$$



### 모멘텀(momentum)

momentum이라는 말 그대로 “가속”을 주는 것이다.

Gradient Descent를 통해 이동하는 과정에 일종의 “관성”을 주는 것이다.

현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식이다.

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$$\theta = \theta - v_t$$

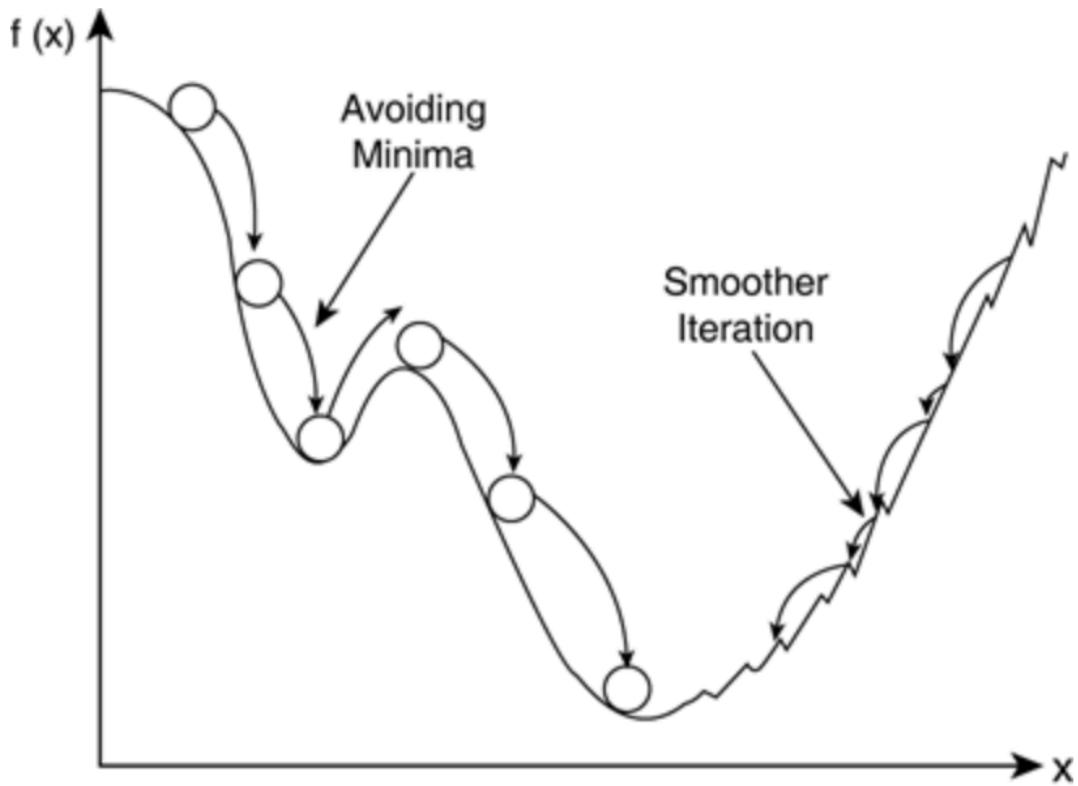
이때,  $\gamma$ 는 얼마나 moementum을 줄 것인지에 대한 momentum term으로서, 보통 0.9정도의 값을 사용한다.

**이점**

local minima를 빠져나올것이라고 기대할수 있다.

모멘텀 방식을 이용하는 경우 local minima를 빠져나오는 효과가 있을 것이라고도 기대할 수 있다.

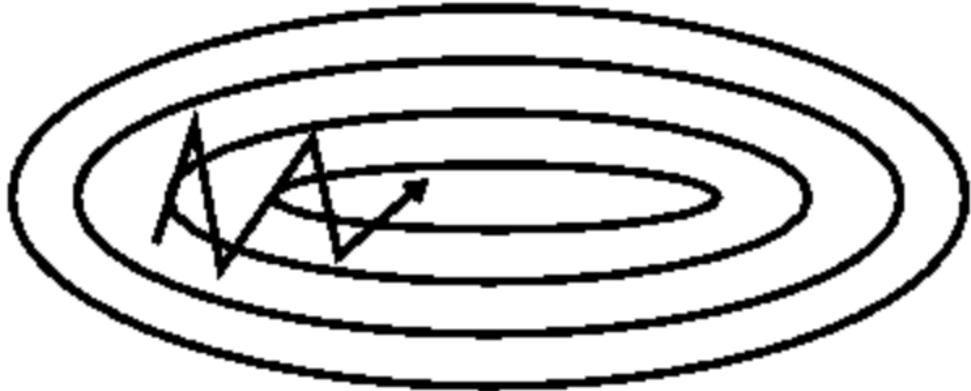
기존의 SGD를 이용할 경우 좌측의 local minima에 빠지면 gradient가 0이 되어 이동 할 수 없지만, momentum 방식의 경우 기존에 이동했던 방향에 관성이 있어 이 local minima를 빠져나오고 더 좋은 minima로 이동할 것을 기대할 수 있게 된다.



그런데 내 예상으로는 그저 “기대” 정도만 할수 있을것 같다는 것이다.  
 관성이 줄어서 0으로 수렴하면 momentum 항의 의미가 없어질 것이기 때문이다.  
 SGD에 비해 더 빨리 이동할수 있다.  
 SGD가 Oscillation 현상을 겪고 있는 상황을 가정하자.



현재 SGD는 중앙의 최적점으로 이동해야하는 상황인데, 한번의 step에서 움직일 수 있는 step size는 한계가 있으므로  
 이러한 oscillation 현상이 일어날 때는 좌우로 계속 진동하면서 이동에 난항을 겪게 된다.



그러나 Momentum 방식을 사용할 경우 다음과 같이 자주 이동하는 방향에 관성이 걸리게 되고,

진동을 하더라도 중앙으로 가는 방향에 힘을 얻기 때문에 SGD에 비해 상대적으로 빠르게 이동할수 있다.

### Adagrad(Adaptive Gradient)

adaptive라는 말 그대로, 학습률을 “조정”하는 방식이다.

변수들을 Update 할때 각각의 변수마다 step size를 다르게 설정해서 이동하는 방식이다.

$$h := h + \frac{\partial L}{\partial W} \otimes \frac{\partial L}{\partial W}$$

$$W := W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

여기서  $\otimes$ 는 원소별 곱셈 기호이다.

$h$ 는 기존 기울기 값을 제곱하여 계속 더해주는 것이다.

그리고 매개변수를 갱신할때 루트값을 씌워 반비례한 크기로 이동을 진행한다.

#### 아이디어

지금까지 많이 변화하지 않은 변수들은 step size를 크게 하고,

지금까지 많이 변화했던 변수들은 step size를 작게 하자

+ 학습률의 값이 너무 작으면 학습시간이 너무 길어지고, 반대로 너무 크면 발산하니까 이를 보정하자!

결국!, 학습속도 감소 + 발산 방지

내 생각에는 이건 좀 안전한 방향으로 움직인다고 해야 할까?

#### 제곱

AdaGrad는 과거의 기울기를 제곱하여 계속 더해간다.

그래서 학습을 진행할수록 갱신 강도가 약해진다.

->  $h$ 에 제곱한 항이 계속 더해지니까  $h$ 는 계속 커질것이다.

그리고 과거의 기울기를 제곱하여 더해가므로 과거의 기울기가 크면 클수록 학습의 step이 감소할 것이다.

-> 너무 빨리 변화하는 것을 방지한다.(너무 빨리 변화하면, 발산할수 있으니까)

### 벡터표현

$$G_t = G_{t-1} + (\nabla_{\theta} J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

Neural Network의 parameter가 k개 라고 할때,

$G_t$ 는 k차원 벡터로서 time step t까지 각 변수가 이동한 gradient의 sum of squares를 저장한다.

$\theta$ 를 업데이트하는 상황에서는 기존 step size  $\eta$ 에  $G_t$ 의 루트값에 반비례한 크기로 이동을 진행하여 지금까지 많이 변화한 변수일수록

적게 이동하고 적게 변화한 변수일 수록 많이 이동하도록 한다.

이때,  $\epsilon$ 은  $10^{-4} \sim 10^{-8}$  정도의 작은 값으로서 0으로 나누는 것을 방지하기 위한 작은 값이다.

여기에서  $G_t$ 를 업데이트 하는 식에서 제곱은 element-wise 제곱을 의미하며  $\theta$ 를 업데이트하는 식에서도 기울기와 계수의 곱은 element-wise한 연산을 의미한다.

### 문제점

그런데 Adagrad에는 학습을 계속 진행하면 step size가 너무 줄어든다는 문제점이 있다.

$G$ 에는 계속 제곱한 값을 넣어주기 때문에  $G$ 의 값들이 계속해서 증가하기 때문인데, 학습이 오래 진행될 경우 step size가 너무 작아져서 결국 거의 움직이지 않게 된다.

이를 보완하여 거친 알고리즘이 RMSProp과 AdaDelta이다.

### RMSProp

제프리 힌튼이 제안한 방법으로서, Adagrad의 단점을 해결하기 위한 방법이다.

Adagrad의 식에서 gradient의 제곱값을 더해가면서 구한  $G_t$  부분을 합이 아니라 지수평균으로 바꾸어서 대체한 방법이다.

이렇게 대체를 할 경우 Adagrad처럼  $G_t$ 가 무한정 커지지는 않으면서 최근 변화량의 변수간 상대적인 크기 차이는 유지할 수 있다.

$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

### AdaDelta(Adaptive Delta)

RMSProp과 유사하게 AdaGrad의 단점을 보완하기 위해 제안된 방법이다.

AdaDelta는 RMSProp과 동일하게  $G$ 를 구할 때 합을 구하는 대신 지수평균을 구한다.

다만, 여기에서는 step size를 단순하게  $\eta$ 로 사용하는 대신 step size의 변화값의 제곱을 가지고 지수평균 값을 사용한다.

$$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$$

$$\Delta_{\theta} = \frac{\sqrt{s + \epsilon}}{\sqrt{G + \epsilon}} \cdot \nabla_{\theta} J(\theta_t)$$

$$\theta = \theta - \Delta_{\theta}$$

$$s = \gamma s + (1 - \gamma)\Delta_{\theta}^2$$

### Adam(Adaptive Moment Estimation)

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수평균을 저장하며, RMSProp과 유사하게 기울기의 제곱값의 지수 평균을 저장한다.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_{\theta} J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

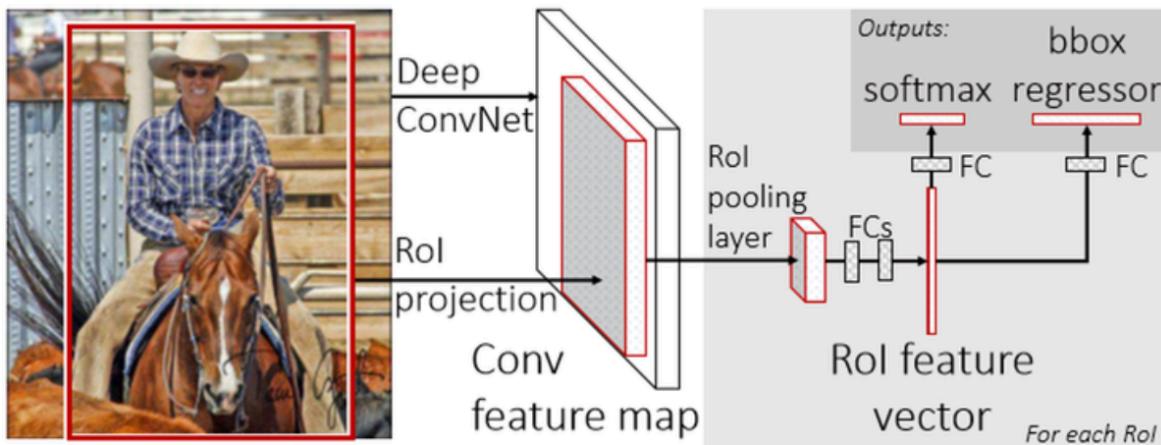
다만, Adam에서는  $m$ 과  $v$ 가 처음에 0으로 초기화되어 있기 때문에 학습의 초반부에서는  $m_t, v_t$ 가 0에 가깝게 bias 되어 있을것이라고 판단하여 이를 unbaised 하게 만들어주는 작업을 거친다.

### 발진(Oscillation)

주기적이고 반복적인 진동

- 정해진 공간에서 같은 운동을 반복하는 주기 운동

### (Detection)



Detection은 Classification과 달리 특정 대상이 있는지 여부만을 가리키는 것이 아니라, 위치 정보를 포함한다.

보통 bounding box라고 부르는 사각형의 영역을 통해 위치 정보까지 포함하게 된다.

Detection은 대상의 여부뿐만 아니라, 위치 정보를 포함해야 하기 때문에 위 그림의 오른쪽처럼, class 여부를 가리는 softmax 부분과

위치정보를 구하는 bbox regressor로 구성이 된다.

### 차원 축소(dimensionality reduction)

차원의 수를 줄이는 방법이다.

관찰 공간 위의 샘플들에 기반으로 잠재 공간을 파악하는 것

#### 차원의 저주(Dimension Curse)

Feature가 많음으로써 생기는 문제이다.

피처가 많으면 연산량이 많아지고, 학습을 위해 더 많은 데이터가 필요해진다.

#### 사용하는 곳

데이터 압축 -> Feature의 중복성 제거

시각화 -> Feature를 줄임으로써 많은 Feature들을 2차원, 3차원의 Feature로 표현함

#### 차원 감소 방식

feature가 유용한지 아닌지 확인하는 과정

##### 특징 선택(feature selection)

데이터의 특성을 가장 잘 나타내는 주요 필드 몇개만을 선택하여 대표 피처로 선택하는 방법이다.

사전 지식을 통해 선택해도 되고, 다른 방식을 통해 선택해도 된다

원래 피처에서 부분 집합만을 선택하는 방법이다.

ex)

[7, 1, 2], [100, 1, 3], [92, 1, 5]가 있을 때 각 첫번째 열과 세번째 열이 변화폭이 크므로 첫번째와 세번째 열만 대표피쳐로 사용한다.

-> [7, 2], [100, 3], [92, 5]

##### 목적

모든 특징의 부분 집합을 선택해서 간결한 특징 집합을 만드는 것이다.

##### 자동 특징 선택(automatic feature selection methods)

특징 중 몇 개를 없애보고 개선된다면 성능을 확인해본다.

##### 특징 추출(feature extraction)

원본 특징들의 조합으로 새로운 특징을 생성하려고 시도한다.

PCA 같은 방법 존재

## 주성분분석(Principal Component Analysis, PCA)

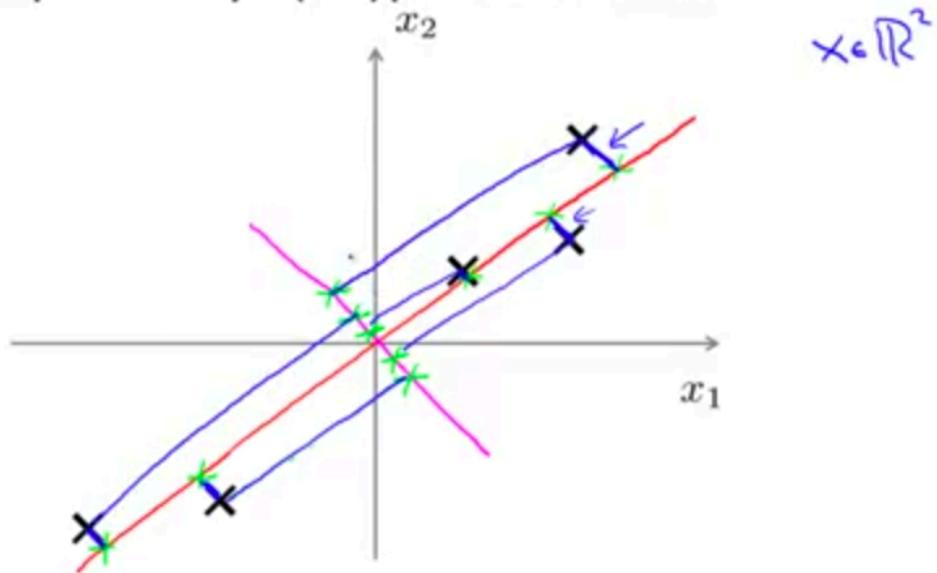
PCA는 분포된 데이터들의 주성분(Principal Component)를 찾아주는 방법이다.

PCA는 먼저 데이터에 가장 가까운 초평면(hyperplane)을 구한 다음, 데이터를 이 초평면에 투영(projection) 시킨다.

아래 그림과 같은 빨간선을 찾아주어야 한다.

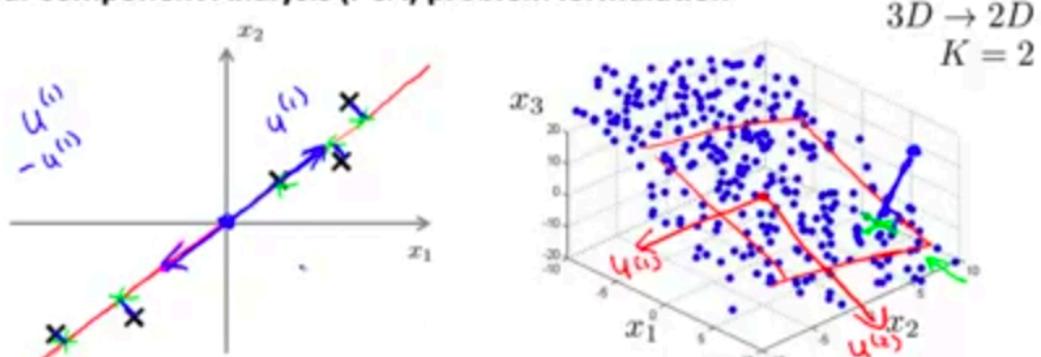
빨간 선을 찾기 위해서는 투영오차(projection error)가 최소화 되는 선을 찾으면 된다.

### Principal Component Analysis (PCA) problem formulation



형식적 정의

### Principal Component Analysis (PCA) problem formulation



Reduce from 2-dimension to 1-dimension: Find a direction (a vector  $u^{(1)} \in \mathbb{R}^n$ ) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find  $k$  vectors  $u^{(1)}, u^{(2)}, \dots, u^{(k)}$  onto which to project the data, so as to minimize the projection error.

2차원에서 1차원으로 감소시키려면 projection 에러를 감소시키는 특정한 벡터를 찾으면 된다.

n차원에서 k차원으로 감소시키려면 projection error를 감소시키는 특정한 벡터 set을 찾으면 된다.

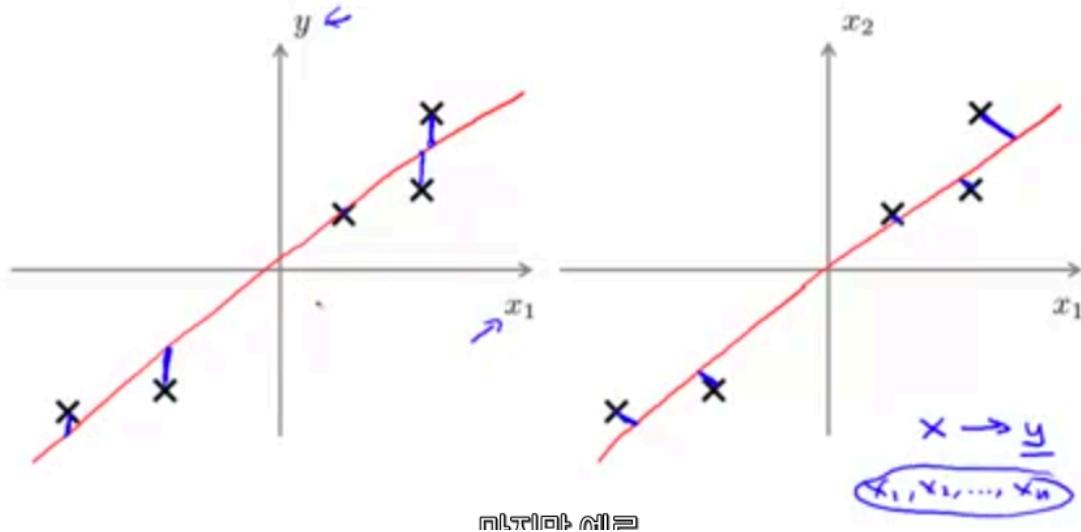
PCA는 Linear regression과 다르다

왼쪽이 Linear regression을 하는 방법이고 오른쪽이 PCA를 하는 방법이다.

Linear regression은 특별한 변수가 있어서(y) 이에 대한 오차를 최소화 하는 방향으로 선을 결정한다.

반면 PCA는 특별한 변수가 없으며 선과 data 상의 최단거리가 최소화되는 방향으로 선을 결정한다.

## PCA is not linear regression



### 목적

d차원의 feature space에 존재하는 N개의 데이터  $X \in R^{d \times N}$

N개의 데이터  $Z \in R^{p \times N}$ 로 변환하는 것이다.

### 초평면의 선택

이때  $X$ 는 행렬로 표현되며,  $X$ 의 열벡터  $X_k = [x_{1k}, x_{2k}, \dots, x_{dk}]'$  는 k번째 데이터를 나타내고,

$x_{ik}$ 는 k번째 데이터의 i번째 요소를 의미한다.

PCA는 원본 데이터  $X$ 를 다음의 두 가지 기준을 만족하는 새로운 feature space로 사용한다.

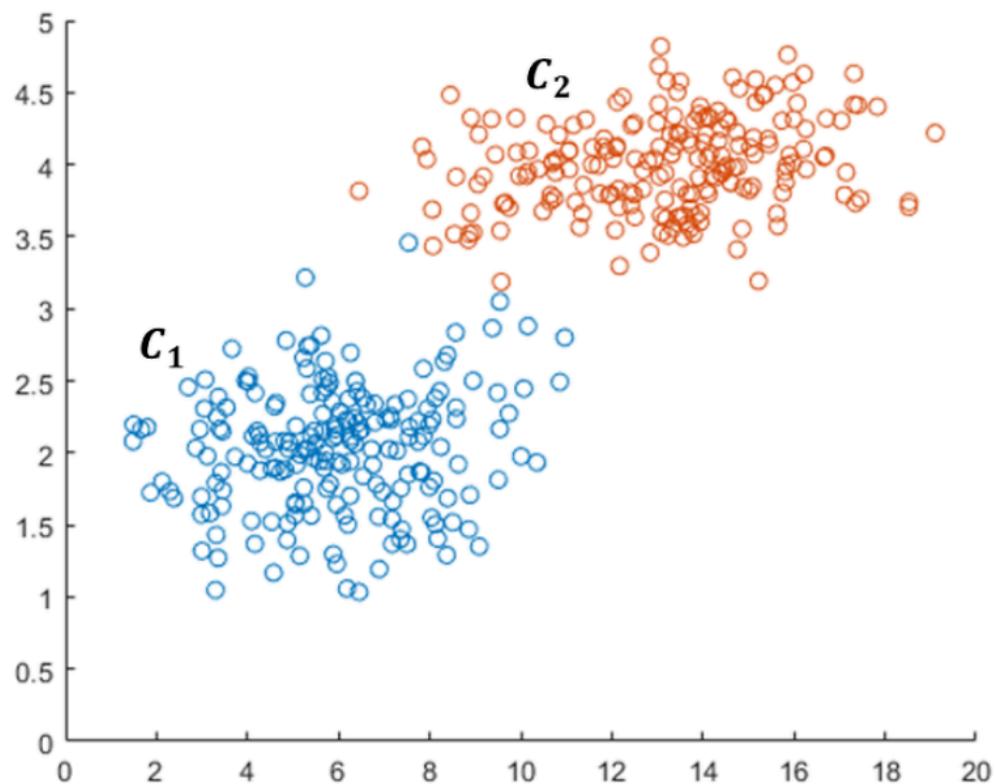
- 새로운 feature space를 구성하는 feature인  $z_1, z_2, \dots, z_p$ 는 서로 선형 독립이다.

- 원본 데이터  $X$ 를  $z_j$ 축으로 사상할 때, 분산이 최대화되도록  $z_j$ 를 결정한다.

이때  $z_1$ 축은 가장 큰 분산을 갖는 축이며,  $z_2$ 는 두번째로 큰 분산을,  $z_j$ 는 j번째로 큰 분산을 나타낸다.

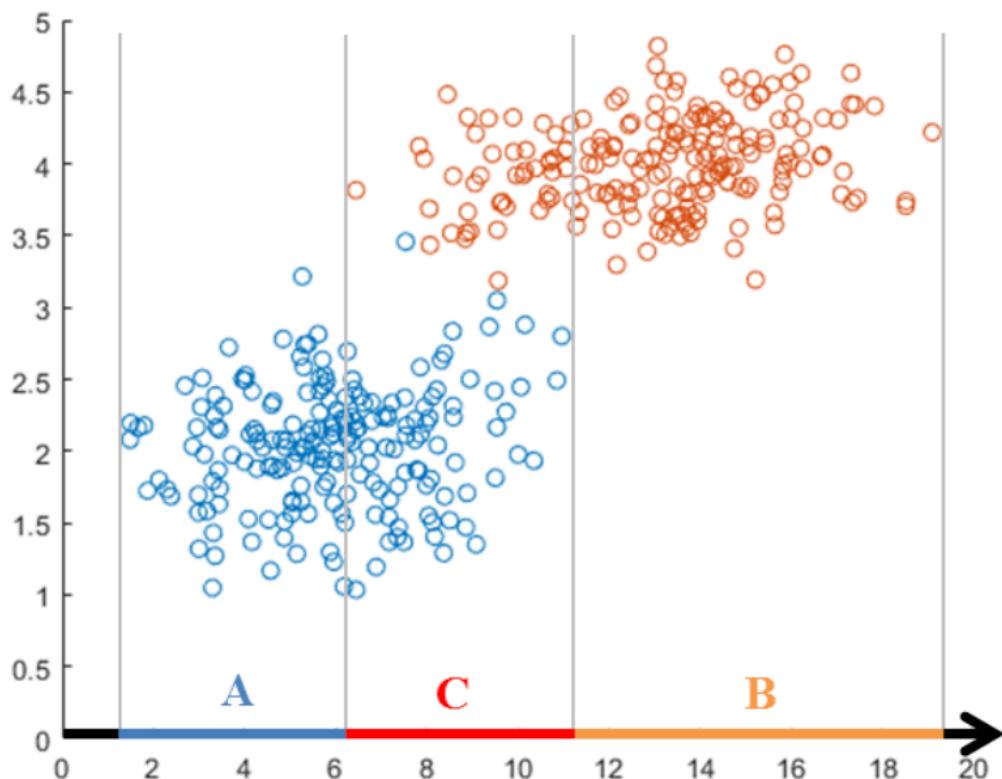
### 분산을 최대화하는 새로운 축을 찾는 이유

다음은 클래스 C1, C2로 구성되며 2차원 공간에 존재하는 데이터 X가 있다.



[그림 1] 예제 데이터의 분포

2차원 공간에 존재하는  $X$ 를 1차원으로 차원 축소를 해야하는 상황에서 데이터를 단순히  $X$ 축으로 사상하는 방법을 생각해볼수 있다.



[그림 2] 예제 데이터에 대한 x-축 사상

A,B에 해당하는 영역은 각각 C1, C2에 속하는 데이터가 x축에 사상되는 영역을 나타내며,

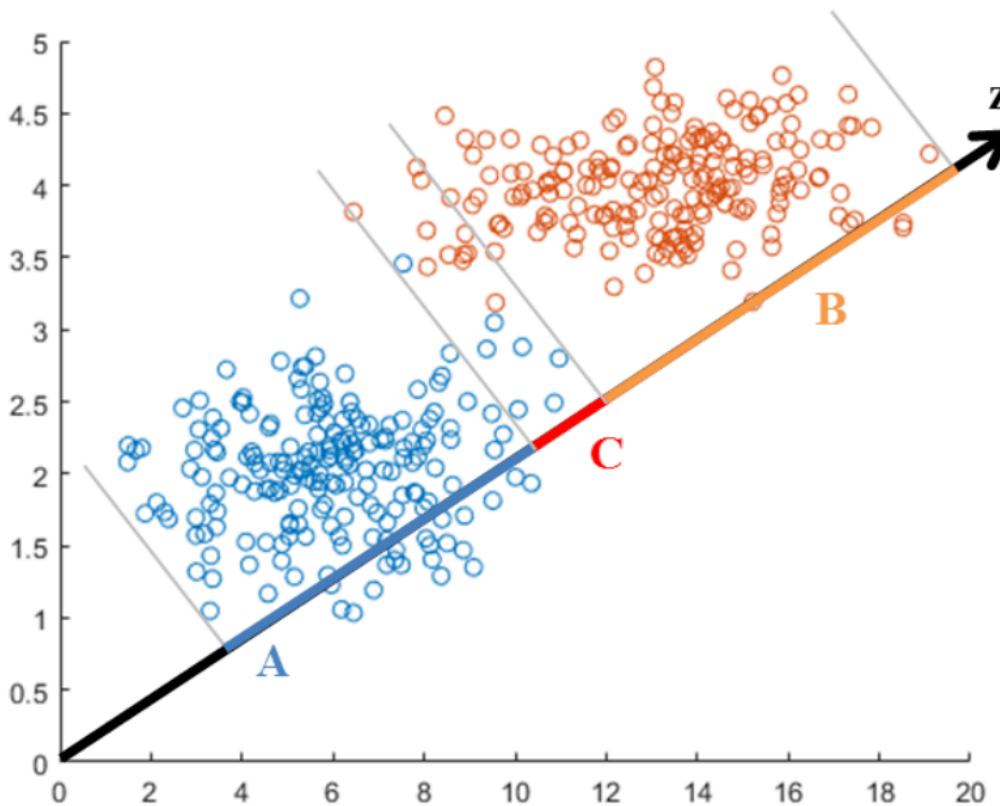
C에 해당하는 영역은 C1, C2의 데이터가 겹쳐지는 영역을 의미한다.

Classification 또는 Clustering의 관점에서 차원 축소 문제를 볼 때 C에 해당하는 영역이 클수록 Classification, Clustering 알고리즘의 정확도는 하락할 것이다.

반면 C에 해당하는 영역이 아예 존재하지 않는다면 100%정확도를 갖는 Classification, Clustering 알고리즘을 만드는 것은 어려운 일이 아닐 것이다.

이 목표는 데이터의 분산을 최대화하도록 차원 축소를 진행함으로써 간접적으로 달성할 수 있다.

이렇나 관점에서 PCA는 z축과 같은 주성분을 갖는다.



[그림 3] 예제 데이터에 대한 주성분, z-축 사상

주성분

그 방향으로 데이터들의 분산이 가장 큰 방향벡터를 의미한다.

### 파인 투닝(Fine Tuning)

기존에 학습되어져 있는 모델을 기반으로 아키텍처를 새로운 목적(나의 이미지 데이터에 맞게)변형하고 이미 학습된 모델 weights로부터 학습을 업데이트 하는 방법을 말한다.

### 이미지 shape의 이해

(이미지 개수, 행수, 열수, 필터수) 이렇게 이루어져 있다.

즉 numpy 배열이 다음과 같이 이루어짐

배열 구성이 이미지 -> 행 -> 열 -> 필터수 이렇게 되어있음

[

[

[1, 2, 3 … 필터의 값들이 들어감]

… 여기엔 각 열에 해당하는 데이터가 들어감

]

… 여기엔 각 행에 해당하는 데이터가 들어감

]

… 여기엔 각 이미지에 대한 데이터가 들어감

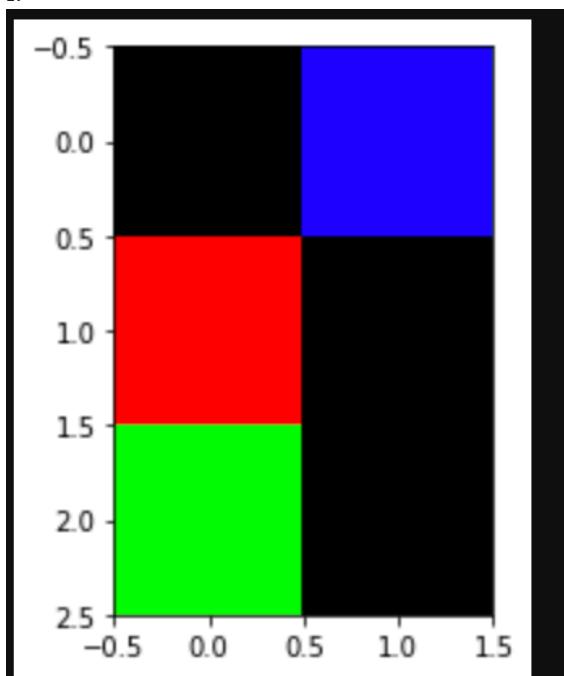
]

ex)

```

import numpy as np
img = np.array([
    [0, 0, 0], # 행이 세개
    [255, 0, 0], # 열이 두개
    [0, 0, 255],
],
[
    [0, 0, 0], # 필터가 세개
    [255, 0, 0],
    [0, 0, 0],
],
[
    [0, 255, 0], # [0, 0, 0],
    [0, 0, 0],
]
])

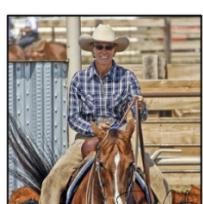
```



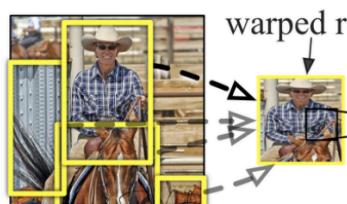
R-CNN(Regions with CNN features, RCNN)

구조

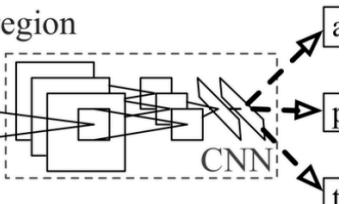
### R-CNN: *Regions with CNN features*



1. Input image



2. Extract region proposals (~2k)



3. Compute CNN features

4

1. 이미지를 집어 넣는다.
2. 2000개의 영역(Bounding Box)을 Selective Search 알고리즘을 통해 추출하여 잘라낸다.(Cropping)

이를 CNN 모델에 넣기 위해 같은 사이즈(227 x 227)로 찌그러뜨린다.

3. Wrapped image를 각각 CNN 모델에 집어넣는다.
4. 각각 Classification을 진행하여 결과를 도출한다.

### 단점

모든 Proposal에 대해 CNN을 거쳐야 하므로 연산량이 매우 많음

### 영역 찾기(Region Proposal)

selective search 방식을 사용한다.

## (Selective Search), ss 알고리즘

### 목표

객체 인식이나 검출을 위한 가능한 후보 영역을 알아낼 수 있는 방법을 제공하는 것을 목표로 한다.

exhaustive search 방식과 segmentation 방식을 결합하여 보다 뛰어난 후보영역을 선정 한다.

SS는 exhaustive search와 같은 무식한 방법으로 후보 영역을 선정하는 대신에, 비록 segmentation 방법이 한계가 있지만,

segmentation이 동원이 가능한 다양한 모든 방법을 활용하여 seed를 설정하고, 그 seed에 대해서 exhaustive한 방식으로 찾는 것을 목표로 하고 있다.

논문에선 이를 segmentation 방법을 가이드로 사용한 data-driven SS라고 부른다.

아래 그림은 그 과정을 보여주는데 입력 영상에 대해 segmentation을 실시하면,

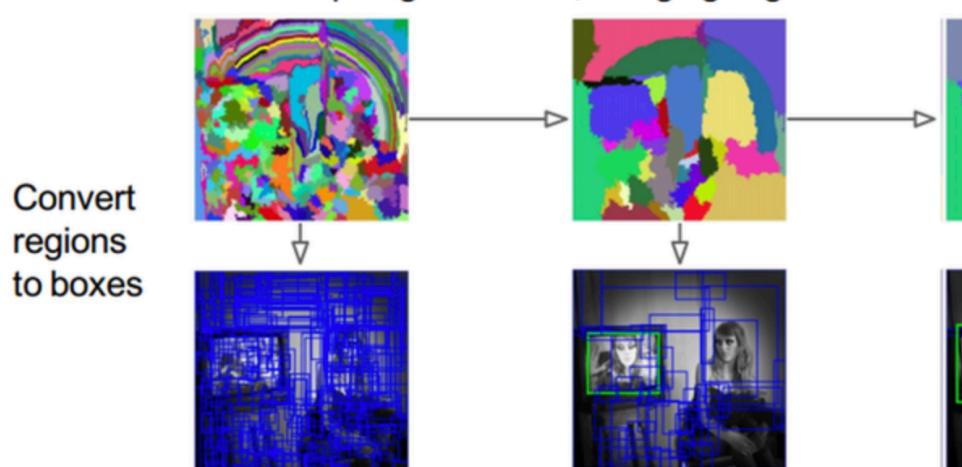
윗줄의 왼쪽에 해당하며, 이를 기반으로 후보 영역을 찾기 위한 seed를 설정한다.

그러면 아래줄의 왼쪽에 해당하는 것처럼 엄청나게 많은 후보가 만들어지게되며, 이것을 적절한 기법을 통하여 통합을 해나가면,

segmentation은 윗줄 오른쪽과 같은 형태로 바뀌게 되며, 결과적으로 그것을 바탕으로 후보영역이 통합되면서 개수가 줄어들게 된다.

이 과정은 이전에 Sunny 엣지 검출기에서 최대가 아닌 자잘한 엣지들을 없앨때 사용했던 non-maximum suppression이나 region growing과 같은 기법들을 떠올리면 도움이 될것이다.

Bottom-up segmentation, merging regions at mult



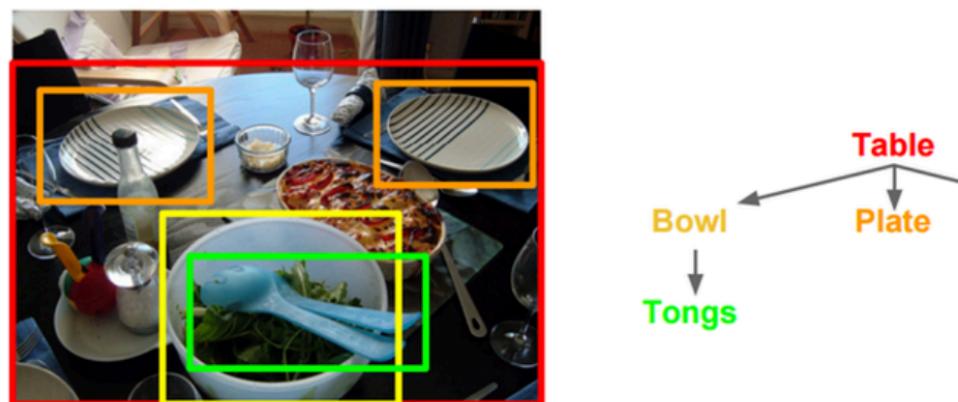
목표 요약

영상은 계층 구조를 가지므로 적절한 알고리즘을 사용하여, 크기에 상관없이 대상을 찾아내야 한다.

컬러, 무늬, 명암 등 다양한 그룹화 기준을 고려한다.  
빨라야 한다.

- 영상에 존재한 객체의 크기가 작은 것부터 큰 것 까지 모두 포함이 되도록 작은 영역에서부터 큰 영역까지 계층구조를 파악할수 있는 bottom-up 그룹화 방법을 사용한다.

### 영상에서의 계층



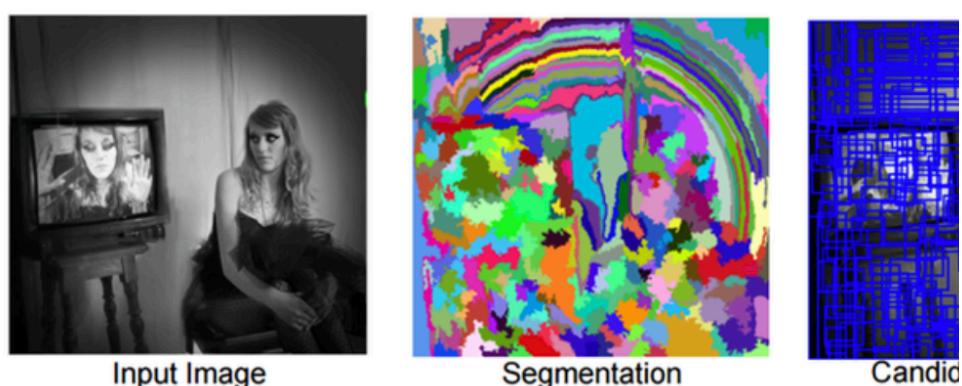
영상은 계층구조를 가진다.

### exhaustive search

후보가 될만한 모든 영역을 샅샅이 조사하는 방법

### 3단계 과정

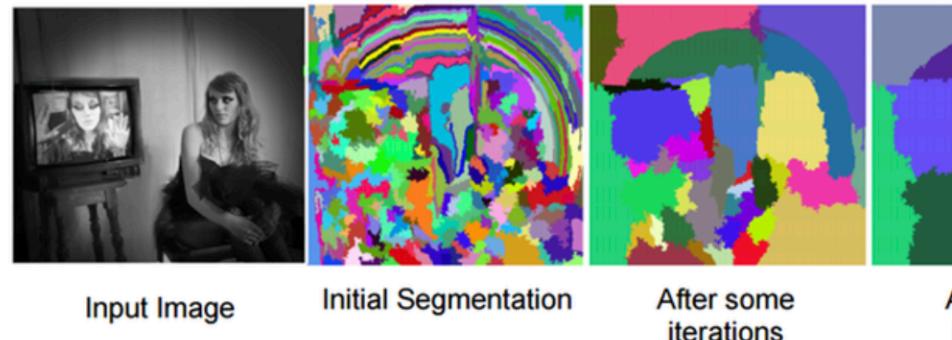
1. 일단 초기 sub-segmentation을 수행한다.



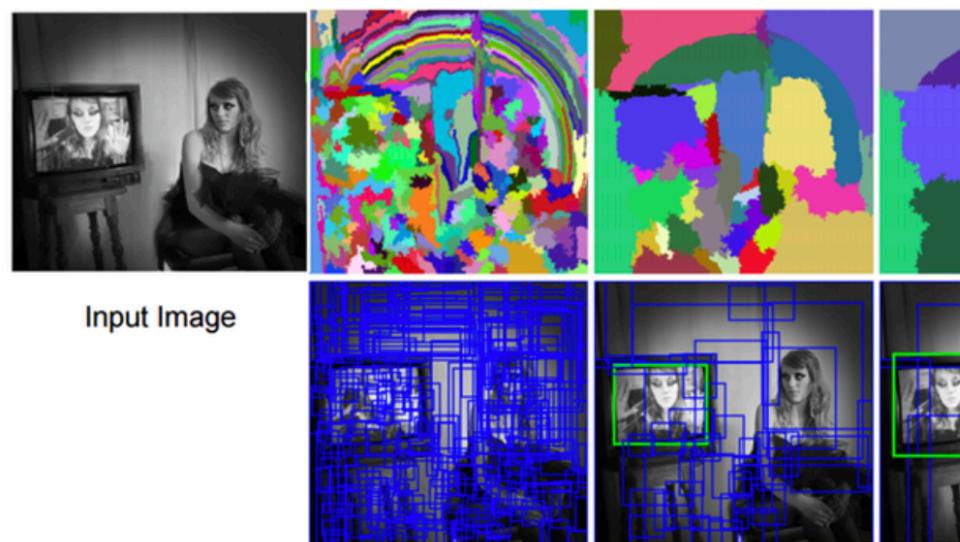
2. 작은 영역을 반복적으로 큰 영역으로 통합한다.

이 때는 Greedy 알고리즘을 사용하며 방법은 다음과 같다.

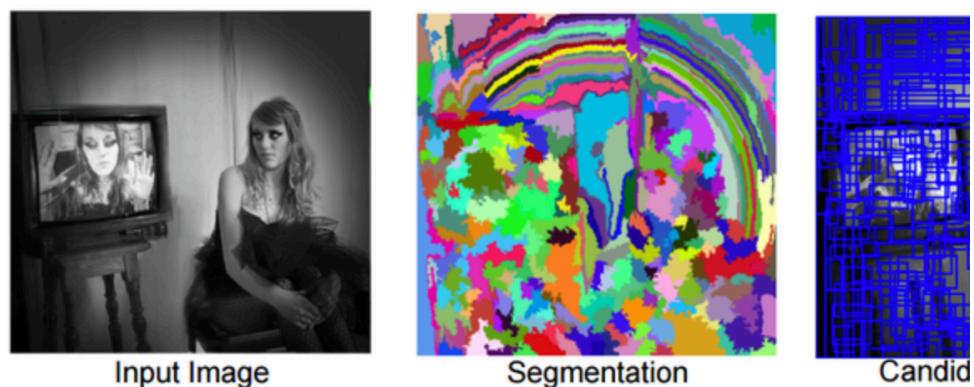
우선 여러 영역으로부터 가장 비슷한 영역을 고르고, 이것들을 좀더 큰 영역으로 통합을 하며, 이 과정을 1개의 영역이 남을때까지 반복한다.



3. 통합된 영역을 바탕으로 후보 영역을 만들어 낸다.



### 유사도 및 통합



영역 병합에는 단순한 greedy 알고리즘을 사용한다.

먼저 Segmentation을 통하여 초기 영역을 설정해준다.

그 후 인접하는 모든 영역들 간의 유사도를 구한다.

영상에 대하여 유사도를 구한후 가장 높은 유사도를 갖는 2개의 영역을 병합시키고, 병합된 영역에 대하여 다시 유사도를 구한다.

새롭게 구해진 영역은 영역 list에 추가한다.

이 과정을 전체 영역이 1개 영역으로 통합될때까지 반복적으로 수행하고, 최종적으로 R-리스트에 들어있는 영역들에 대한 bounding box를 구하면 된다.

---

**Algorithm 1:** Hierarchical Grouping Algorithm

---

**Input:** (colour) image

**Output:** Set of object location hypotheses  $L$

Obtain initial regions  $R = \{r_1, \dots, r_n\}$  using [13]

Initialise similarity set  $S = \emptyset$

**foreach** Neighbouring region pair  $(r_i, r_j)$  **do**

Calculate similarity  $s(r_i, r_j)$

$S = S \cup s(r_i, r_j)$

**while**  $S \neq \emptyset$  **do**

Get highest similarity  $s(r_i, r_j) = \max(S)$

Merge corresponding regions  $r_t = r_i \cup r_j$

Remove similarities regarding  $r_i : S = S \setminus s(r_i, r_*)$

Remove similarities regarding  $r_j : S = S \setminus s(r_*, r_j)$

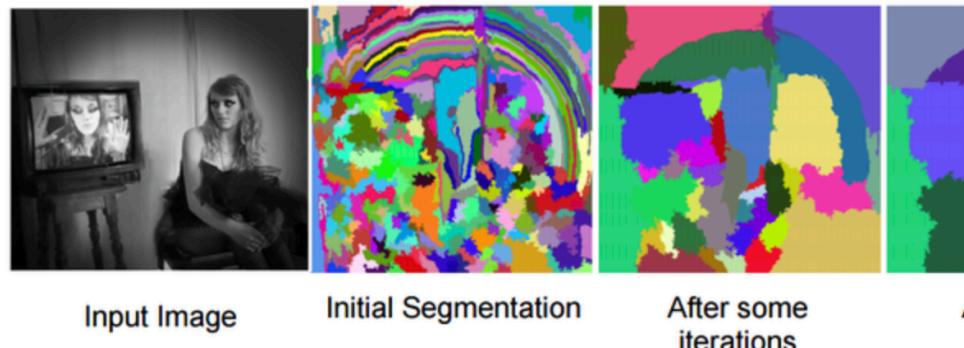
Calculate similarity set  $S_t$  between  $r_t$  and its neighbours

$S = S \cup S_t$

$R = R \cup r_t$

Extract object location boxes  $L$  from all regions in  $R$

---



### 다양화 전략(Diversification Strategy)

후보 영역 추천의 성능 향상을 위해 SS에서는 다음과 같은 다양화 전략을 사용한다.

- 다양한 컬러 공간을 사용
- color, texture, size, fill 등 4가지 척도를 적용하여 유사도를 구하기

### 다양한 컬러 공간

다른 컬러 공간은 서로 다른 항상성(invariance)을 보인다.

그러므로 단순하게 RGB나 흑백 컬러 공간을 사용하는 대신에 8개의 서로 다른 컬러 공간을 사용하게 되면 특정 컬러 공간에서는 검출하지 못했던 후보 영역까지 검출이 가능하게 된다.

가령 RGB 컬러 공간을 사용하는 경우는 그림자나 광원의 세기 변화 등으로 인한 조도 변화에 3개의 채널이 모두 영향을 받게 되지만, HSV컬러 공간을 사용한다면 색상이나 채도는 밝

기의 변화에 거의 영향을 받지 않지만 명도의 영향을 받는다.

아래의 표는 각각의 컬러 공간이 영향을 받는 수준을 평가한 것이다.

이 표에서 -는 영향을 받는 경우이고, +는 영향을 받지 않는 경우이며 +/-는 부분적으로 영향을 받는 것을 나타낸다.

<i>colour channels</i>	R	G	B	I	V	L	a	b	S	r	g	C	H
Light Intensity	-	-	-	-	-	-	+/-	+/-	+	+	+	+	+
Shadows/shading	-	-	-	-	-	-	+/-	+/-	+	+	+	+	+
Highlights	-	-	-	-	-	-	-	-	-	-	-	+/-	+

<i>colour spaces</i>	RGB	I	Lab	rgI	HSV	rgb	C	H
Light Intensity	-	-	+/-	2/3	2/3	+	+	+
Shadows/shading	-	-	+/-	2/3	2/3	+	+	+
Highlights	-	-	-	-	1/3	-	+/-	+

논문에서는 sub-segmentation과 영역을 병합하는 과정에 서로 다른 8개의 컬러 공간을 적용했다고 밝히고 있다.

### 컬러 유사도

히스토그램을 사용한다.

각 컬러 채널에 대해 bin을 25로 하며, 히스토그램을 정규화 시킨다.

각 영역들에 대한 모든 히스토그램을 구한 후 인접한 영역의 히스토그램의 교집합을 구하는 방식으로 유사도를 구하며 식은 아래와 같다.

$$s_{colour}(r_i, r_j) = \sum_{k=1}^n \min(c_i^k, c_j^k).$$

### texture 유사도

object matching에서 뛰어난 성능을 보이는 SIFT(Scale Invariant Feature Transform)와 비슷한 방식을 사용하여 히스토그램을 구한다.

8방향의 가우시안 미분 값을 구하고 그것을 bin을 10으로 하여 히스토그램을 만든다.

SIFT는 128차원의 디스크립터 벡터를 사용하지만, 여기서는 80차원의 디스크립터 벡터를 사용하며, 컬러의 경우는 3개의 채널이 있기 때문에 총 240차원의 벡터가 만들어진다.

컬러 유사도의 히스토그램과 마찬가지로 히스토그램에 대하여 정규화를 수행하며, 영역간의 유사도는 컬러 유사도와 마찬가지 방식을 사용한다.

$$s_{texture}(r_i, r_j) = \sum_{k=1}^n \min(t_i^k, t_j^k).$$

### 크기 유사도

$$s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(im)}$$

### fill 유사도

$$fill(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_j)}{\text{size}(im)}$$

유사도를 사용하는 방식

이렇게 구한 4개의 유사도를 결합시켜서 사용하며 수식은 아래와 같다.

$$\begin{aligned} s(r_i, r_j) &= a_1 s_{colour}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + \\ &\quad a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j), \end{aligned}$$

### SS 방식을 사용한 Object Detection

SS 방식을 발표한 Uijlings 팀이 만든 것이다.

feature detection + SVM이라는 방식을 사용한다.

### Bag of Words(BoW)

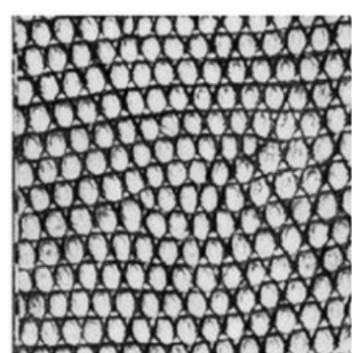
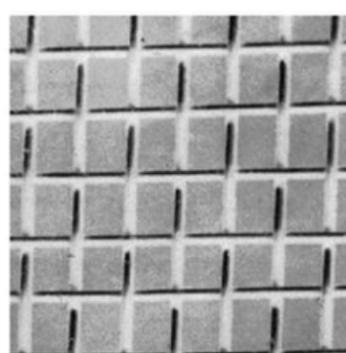
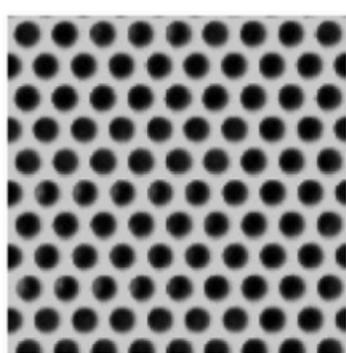
영상을 구성하는 특정한 성분들의 분포를 이용하여 object 여부를 판단한다.

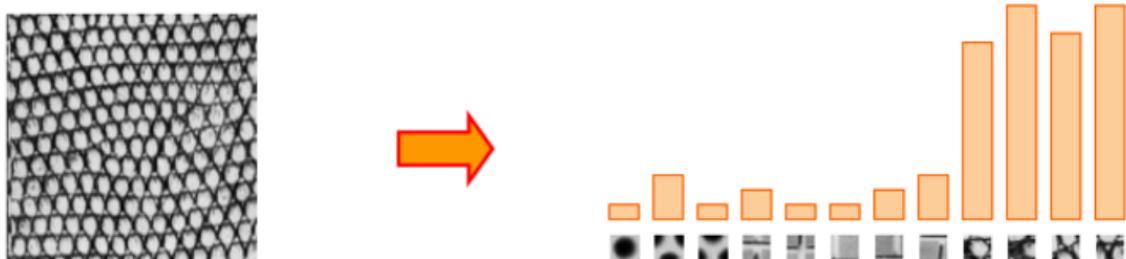
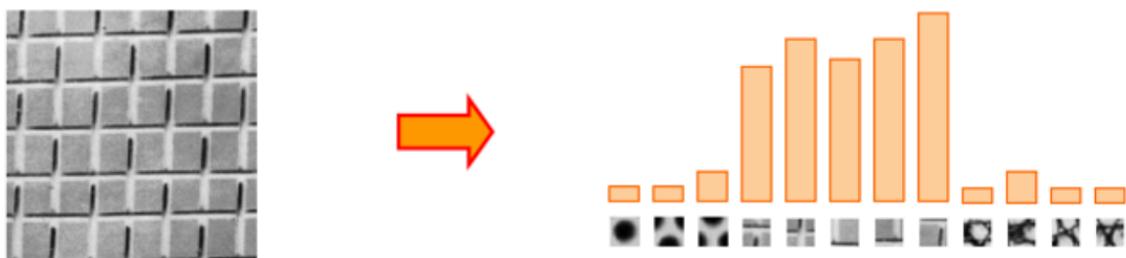
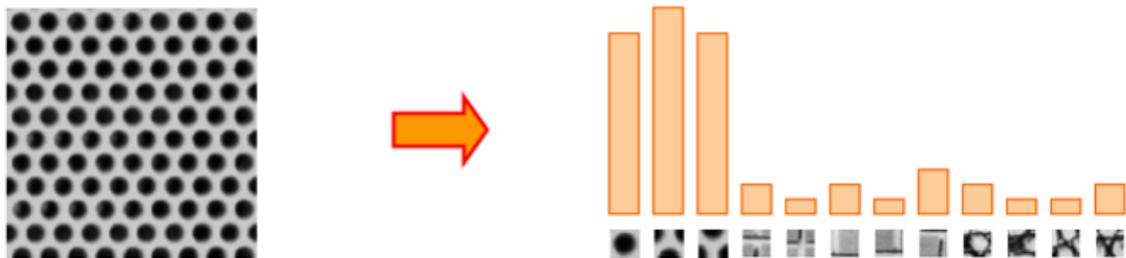
특정 성분들의 조합이 object에 따라 확연히 다른 분포를 보인다면, 크기, 회전, 조명 등에 영향을 받는 object를 검출하기 위해 실제 영상을 직접 비교하는 대신에, 성분들의 분포만을 이용해 해당 object 여부를 판별한다.

여기서 주요 성분은 영상을 구성하는 특징(feature)이라고 볼 수 있으며, 이 특징들의 집합이 바로 코드북(codebook)이 된다.

이 코드북에 속해있는 성분에 대한 분포가 해당 object를 분류나 검출할 수 있게 하는 수단이 된다.

이런 방식을 BoW라 부른다.





### 코드북(codebook)

주요 성분의 집합

#### 코드북을 만드는 방법

SIFT를 많이 사용한다.

BoW를 사용하는 경우 뒷단 분류기로는 SVM이나 Nearest-neighbor 모델을 많이 사용한다.

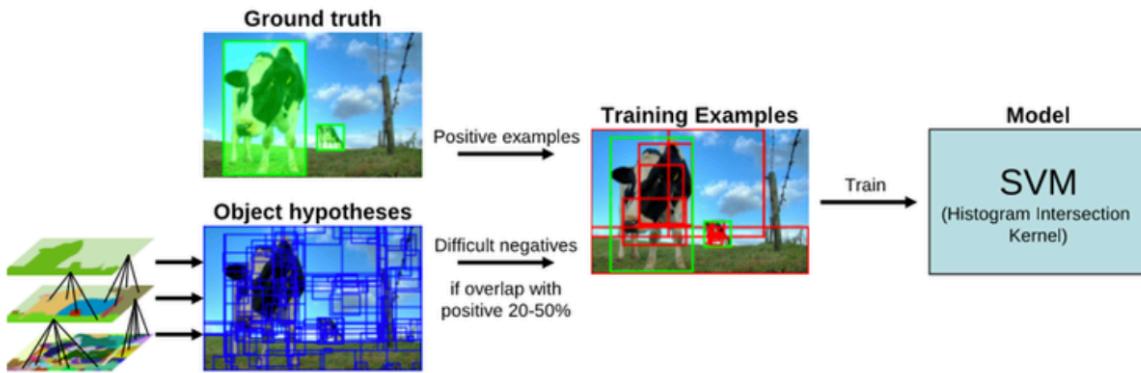
2단계 과정을 거친다.

SVM의 결과가 효과적이려면 true와 false가 확실한 벡터보다는 경계면 근처에 있는 벡터들을 잘 활용해줘야 한다.

### Training

#### 초기학습

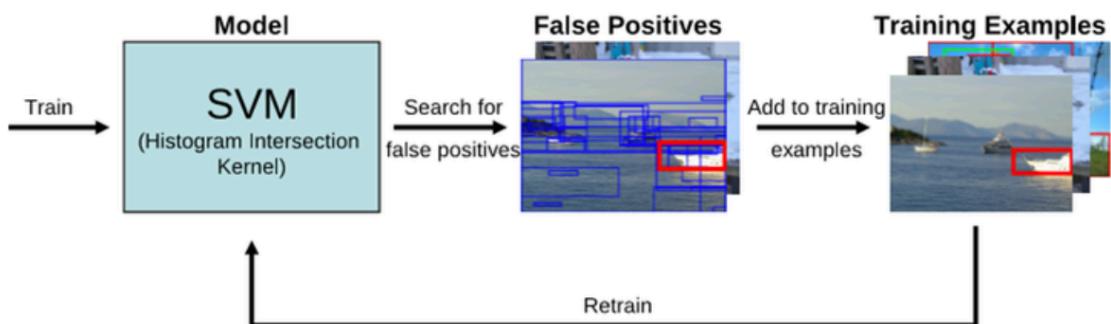
이미 알고 있는 ground truth 데이터로부터 positive example을 구하고, 추천된 후보 영역 중 ground truth 데이터와의 overlap이 20~50% 수준에 있는 경계 근처의 데이터를 negative example로 정하여 SVM을 학습시킨다.



### 학습결과 투닝

초기 학습을 마친 뒤에는 SVM 학습을 투닝시키는 과정을 거치게 되는데, 이 때 필요한 것이 false positive(제대로 검출하지 못한것)이다

False positive를 찾아내고 추가로 negative example을 더하여 SVM을 더 정교하게 투닝하는 과정을 거친다.

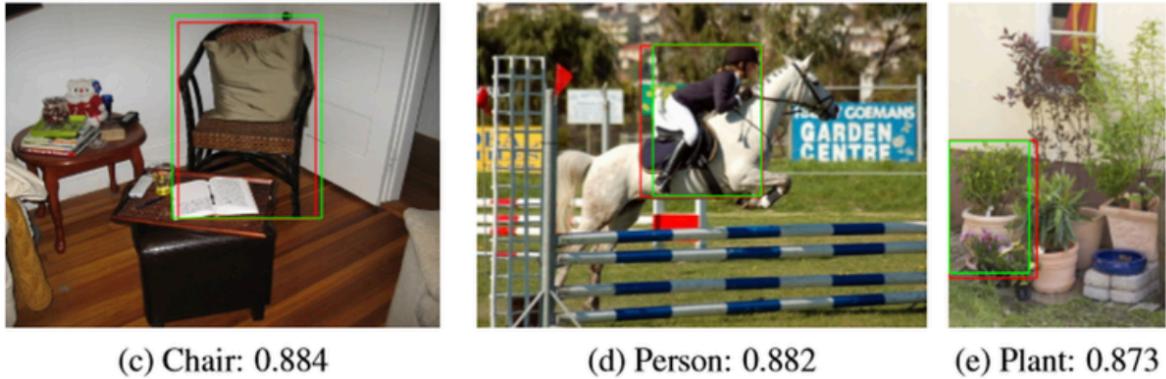


### 실험결과

실험 결과를 확인하는 척도로 사용하는 것이 ABO(Average Best Overlap)인데 이것은 알고 있는 ground data와의 최고의 overlap에 대한 평균을 구한것이다.

$$ABO = \frac{1}{|G^c|} \sum_{g_i^c \in G^c} \max_{l_j \in L} \text{Overlap}(g_i^c, l_j).$$

녹색선은 ground truth이고, 빨간색은 이 알고리즘의 결과이다.  
ABO의 수치가 높을수록 결과가 좋다고 볼수 있다.

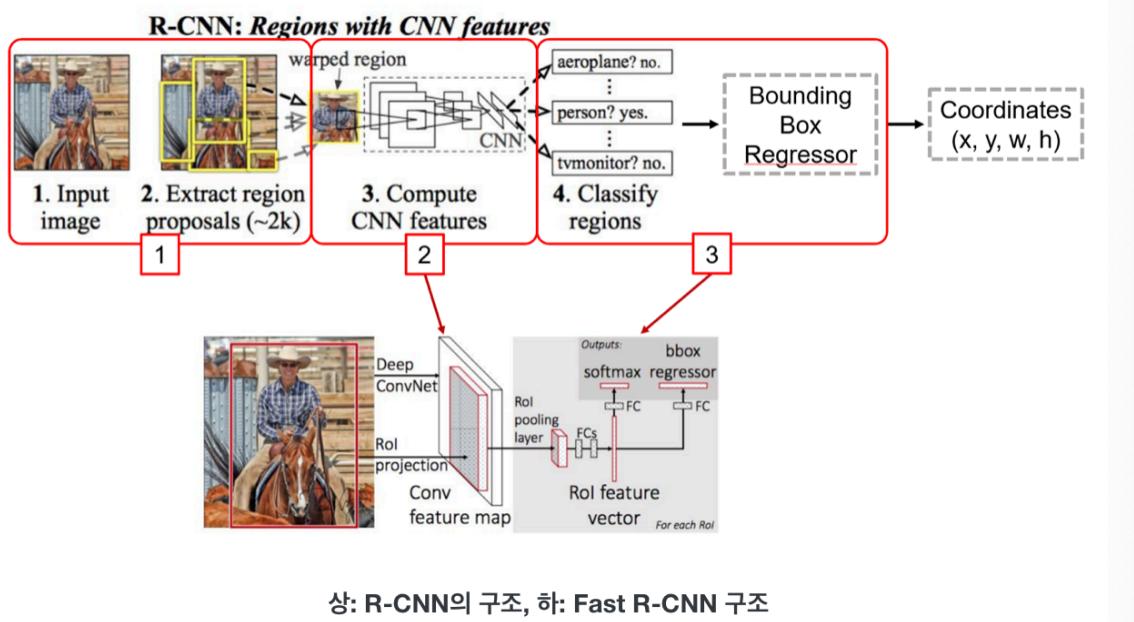


(c) Chair: 0.884

(d) Person: 0.882

(e) Plant: 0.873

### Fast R-CNN



상: R-CNN의 구조, 하: Fast R-CNN 구조

Fast R-CNN은 모든 Proposal이 네트워크를 거쳐야 하는 R-CNN의 병목(bottleneck)구조의 단점을 개선하고자 제안된 방식

가장 큰 차이점은, 각 Proposal들이 CNN을 거치는 것이 아니라 전체 이미지에 대해 CNN을 한번 거친 후 출력 된 특징맵단에서 객체 탐지를 수행

하지만 Fast-RCNN에서 Region Proposal을 CNN Network가 아닌 Selective search 외부 알고리즘으로 수행하여 병목현상 발생

### (Feature Representation)

머신러닝 알고리즘을 작동하기 위해 데이터에 대한 도메인 지식을 활용하여 특징을 만들어내는 과정

### SIFT(Scale Invariant Feature Transform)

속도는 느리지만 local feature를 이용한 패턴 매칭에는 탁월한 성능을 보이는 알고리즈다.  
2004년 Lowe가 발표한 SIFT는 기본적으로 밝기(intensity) 정보의 gradient 방향성에 대한 히스토그램 특성을 구한 후 그것을 128차원 벡터로 표시한다.

### Fully Convolutional Network(FCN)

2015년에 발표한 논문 / 제목은 Fully Convolutional Networks for Semantic Segmentation  
FCN이 주목한 부분은 classification에서 성능을 검증 받은 좋은 네트워크(AlexNet, VGGNet,

GoogLeNet) 등을 이용하는 것이다.

이들 대부분의 classification을 위한 네트워크는 뒷단에 분류를 위한 fully connected layer가 오는데,

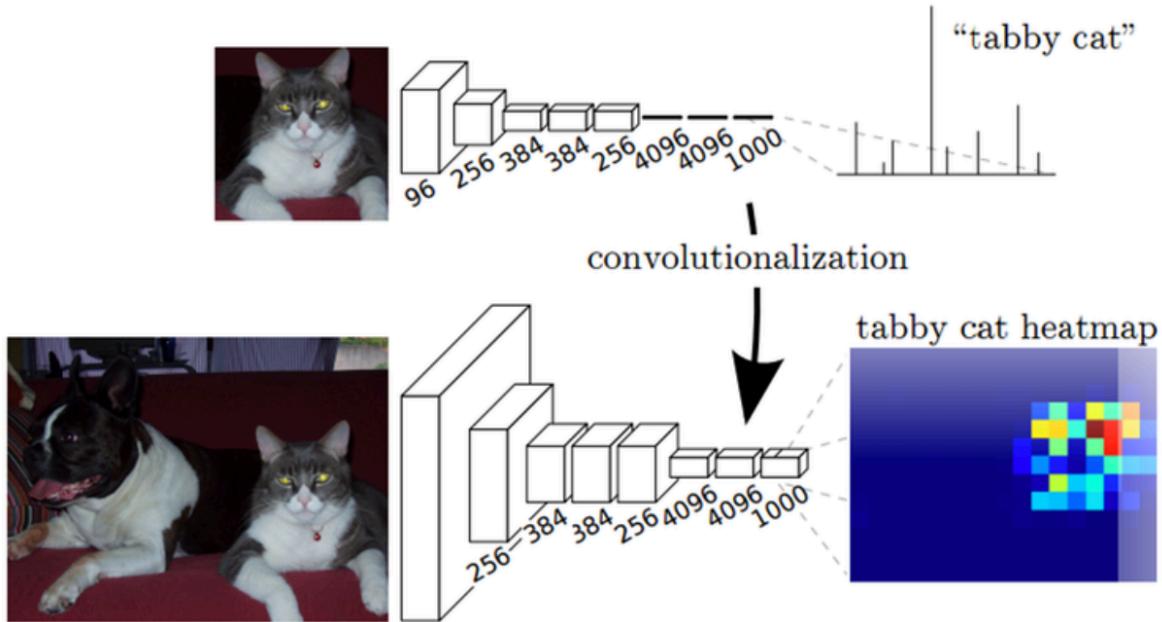
이 fully connected layer가 고정된 크기의 입력만을 받아들이는 문제가 있다.

또 한가지 결정적인 문제는 fully connected layer를 거치고 나면 위치정보가 사라지는 것이다.

segmentation에 사용하려면 위치정보를 알 수 있어야 하는데 불가능하기 때문에 심각한 문제가 된다.

하지만 FCN 개발자들이 주목한 부분은 fully connected layer를 1x1 convolution으로 볼 수 있다 는 점이다.(이를 convolutionalization이라 부름)

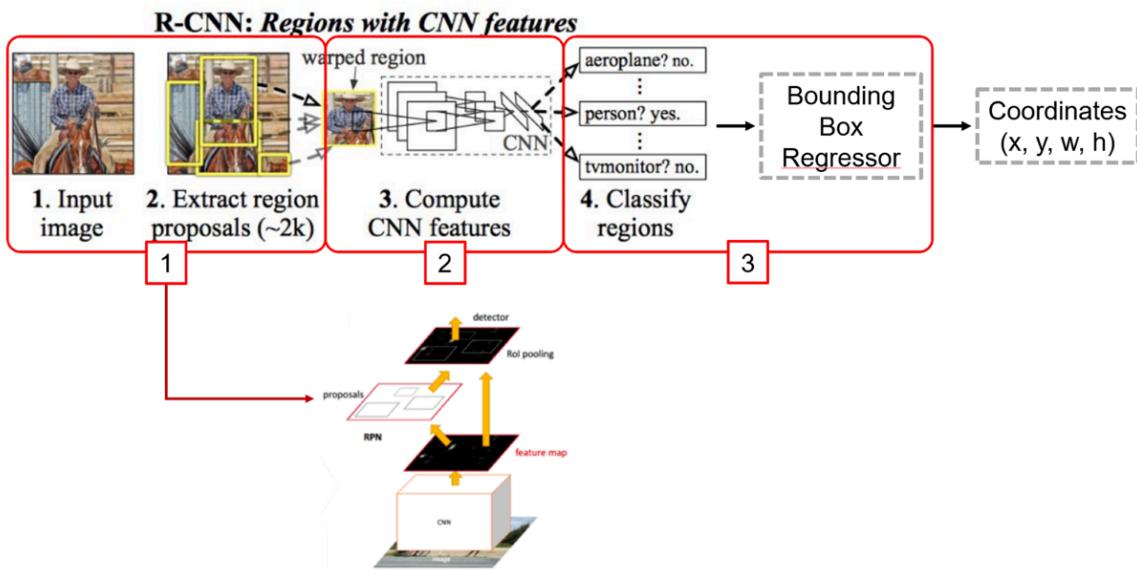
Fully Connected layer를 1x1 convolution으로 간주하면 위치정보가 사라지는 것이 아니라 남게 된다.



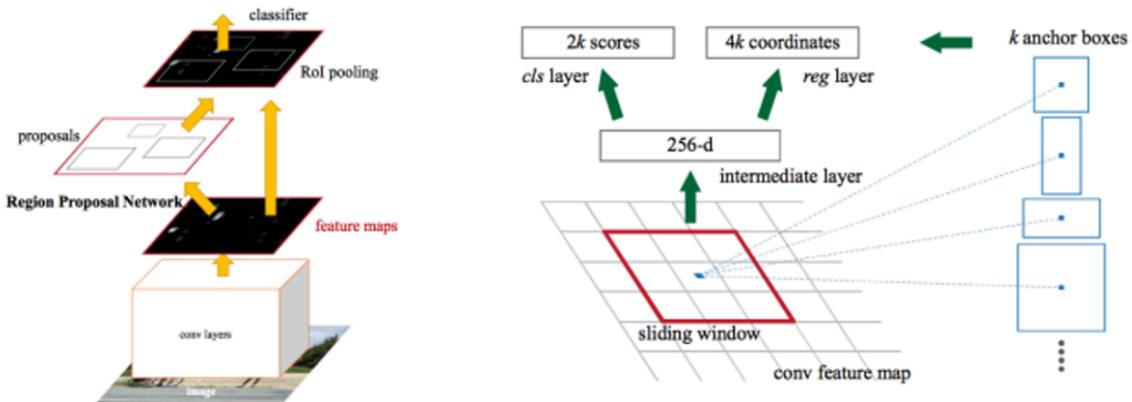
또한 이제는 모든 network가 convolutional network로 구성되기 때문에 더 이상 입력 이미지의 크기 제한을 받지 않게 된다.

또한 patch 단위로 영상을 처리하는 것이 아니라, 전체 영상을 한꺼번에 처리할 수 있어서 겹치는 부분에 대한 연산 절감 효과를 얻을 수 있게 되어 속도가 빨리지게 된다.

(Faster R-CNN)



상: R-CNN 구조, 하: Faster R-CNN 구조



Region Proposal을 RPN이라는 네트워크를 이용하여 수행

Region proposal 단계에서의 bottleneck 현상 제거

- 해당 단계를 기존의 Selective Search가 아닌 CNN(RPN)으로 해결

CNN을 통과한 Feature map에서 슬라이딩 윈도우를 이용해 각 지점(anchor)마다 가능한 바운딩

박스의 좌표와 그 점수를 계산

2:1, 1:1, 1:2의 종횡비(Aspect ratio)로 객체를 탐색

### (Image segmentation)

Segmentation의 목표는 의미적인 면이나 인지적인 관점에서 서로 비슷한 영역으로 영상을 분할 하는 것이다.

영상을 구성하고 있는 주요 성분으로 분해를 하고, 관심인 객체의 위치와 외곽선을 추출해낸다.

### (Gestalt) 시지각 이론

Gestalt는 ‘형태, 모양’을 뜻하는 독일어이다.

Gestalt 시지각 이론에서는 일반적으로 시각을 통해 대뇌로 전달되는 외계의 형태에 대한 정보들은 기억하기 쉬운 상태로 혹은 특징 지워질 수 있는 상태로 정리되는지 설명하고 있으며, 이 이론에 따르면 영상 속에 있는 개체를 집단화 할 수 있는 5개의 법칙이 있다.

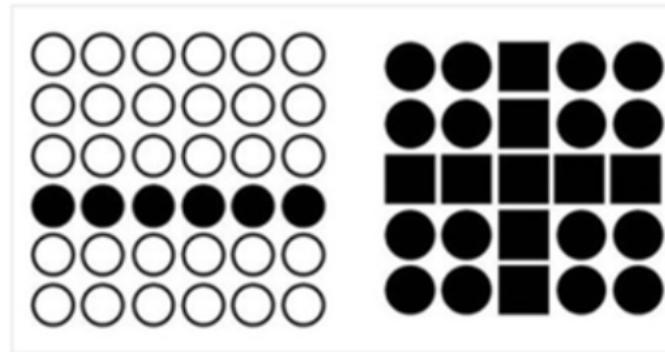
이것을 집단화의 법칙(Law of Grouping)이라고 하고, 1923년 베르트하이머가 체계화 시켰다.

게스탈트 이론은 이런 인지심리학적인 면에 대한 고찰의 결과이며, 이런 특성들을 활용하여 광고나 예술 등에도 활용되며, 영상에 대한 segmentation 연구나 이론 개발 시에도 고려가 되어야 한다.

### 집단화의 법칙

#### 유사성(Similarity)

모양이나 크기, 색상등 유사한 시각 요소들끼리 그룹을 지어 하나의 패턴으로 보려는 경향이 있으며,  
다른 요인이 동일하다면 유사성에 따라 형태는 집단화

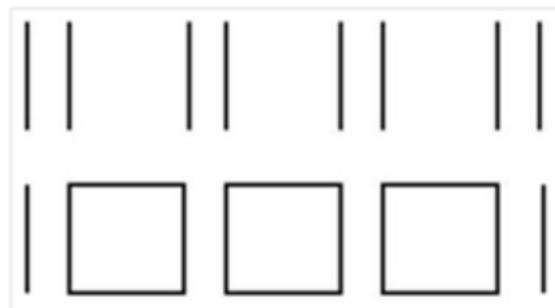


유사성

-> 위 그림에서는 경계를 통해 패턴을 알수있는데 이를 통해 기억을 쉽게 할수 있다.

#### 근접성(Proximity)

시공간적으로 서로 가까이 있는 것들을 함께 집단화해서 보는 경향

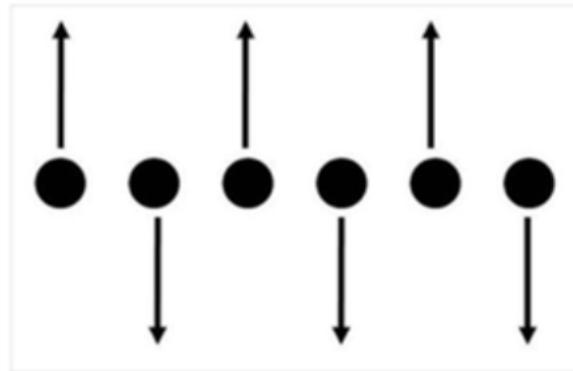


근접성

-> 위 그림에서는 사각형이 근접해 있는데 이를 통해 집단화 해서 볼수 있다.

#### 공통성(Commonality)

같은 방향으로 움직이거나 같은 영역에 있는 것들을 하나의 단위로 인식하며,  
배열이나 성질이 같은것들끼리 집단화 되어 보이는 경향

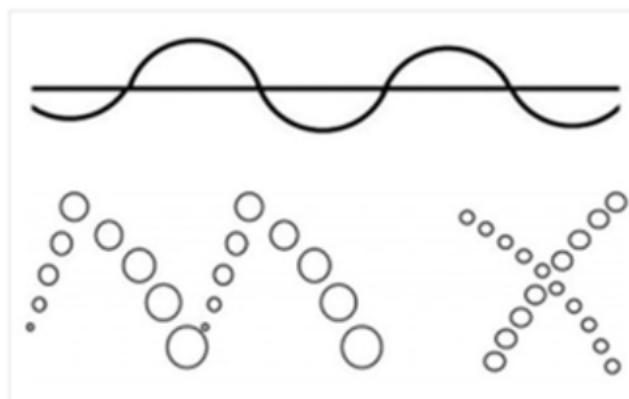


## 공통성

-> 위 그림에서는 “성질”을 운동 방향으로 보면 운동방향이 같은 것들을 그룹화 할수 있다.

### 연속성(Continuity)

요소들이 부드럽게 연결될 수 있도록 직선 혹은 곡선으로 서로 묶여 지각되는 경향

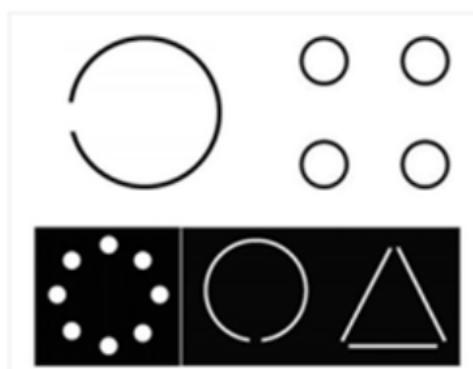


## 연속성

-> 위 그림에서는 부드럽게 연결되는 것을 묶어서 기억할수 있다.

### 통폐합(Closure)

기존의 지식을 바탕으로 완성되지 않은 형태를 완성시켜 지각하는 경향



-> 기존 지식을 바탕으로 원이 아닌 것을 원으로 기억할 수 있다.

### Segmentation 방법

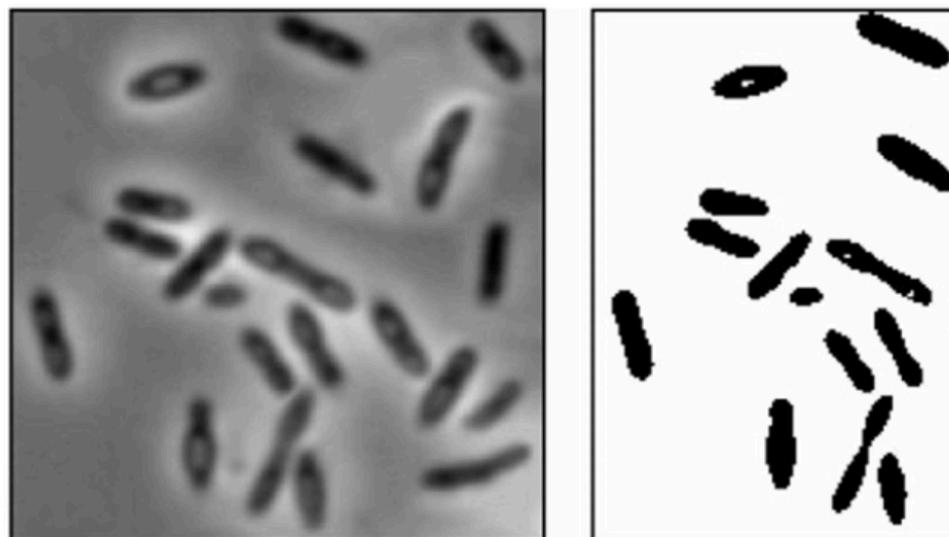
### 픽셀 기반 방법

thresholding에 기반한 방식으로 histogram을 이용해 픽셀들의 분포를 확인 한후 적절한 threshold를 설정하고, 픽셀 단위 연산을 통해 픽셀 별로 나누는 방식이며, 이진화에 많이 사용된다.

예1)

$$p(x, y) = \begin{cases} \text{white} & p(x, y) \geq T \\ \text{black} & p(x, y) < T \end{cases}$$

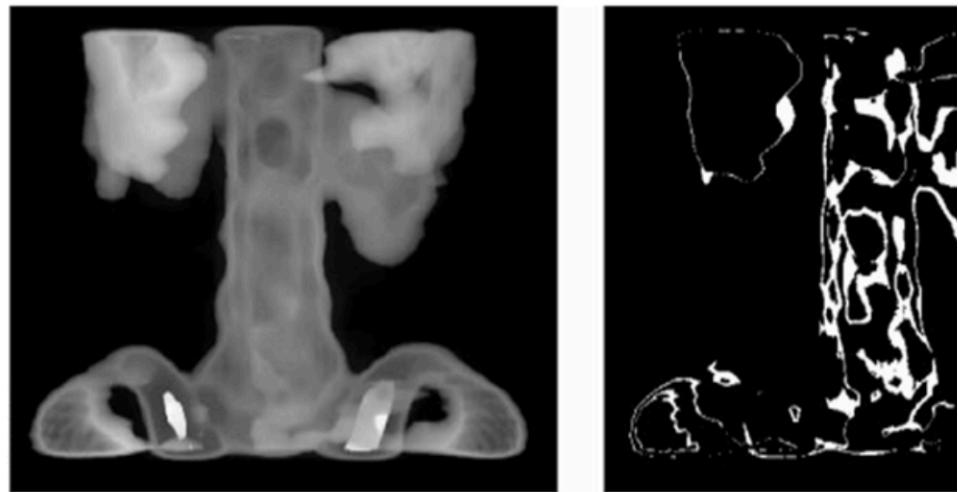
$T$  : Threshold



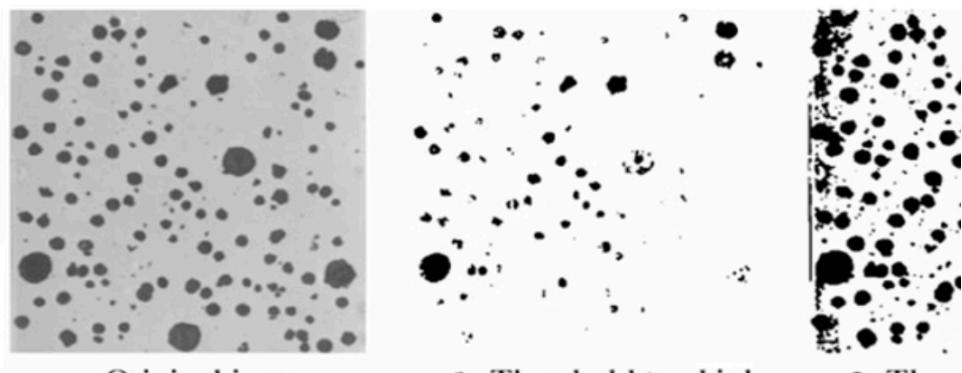
예2)

$$p(x, y) = \begin{cases} \text{white} & T_1 \leq p(x, y) \leq T_2 \\ \text{black} & \text{otherwise} \end{cases}$$

$T_1, T_2$  : Thresholds



적절한 Threshold 설정하기



n: Original image

n1: Threshold too high

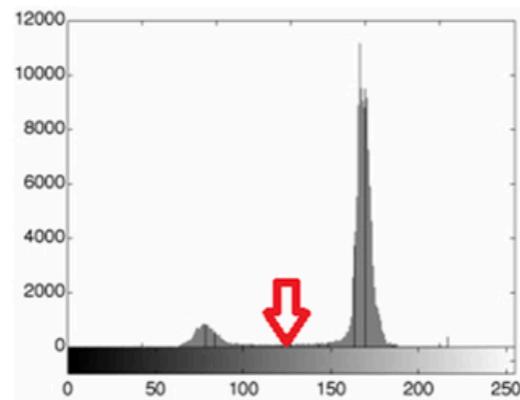
n2: Thres

임계값이 제대로 설정되지 못하면 엉뚱한 결과가 나오므로, 임계값을 적절하게 설정하기 위한 방법이 필요하다.

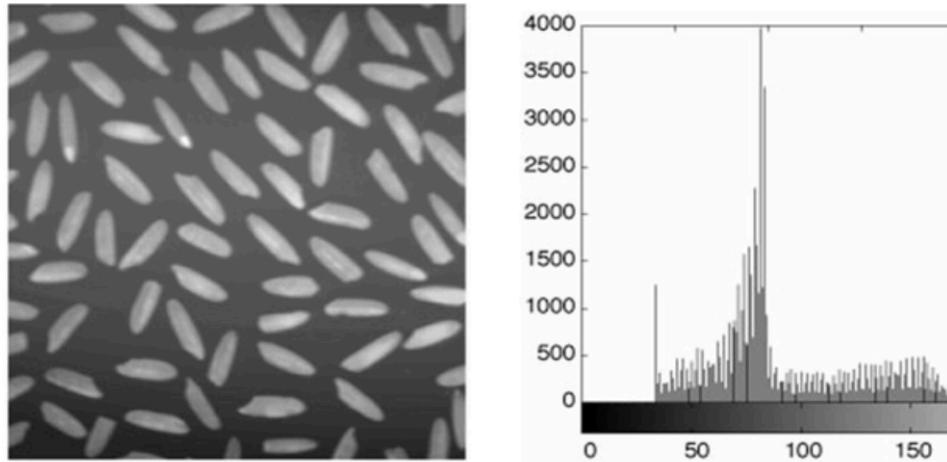
이때 가장 많이 사용되는 방식이 픽셀값의 누적 분포를 알 수 있는 히스토그램을 사용하는 것이다.

아래 그림을 보면 픽셀값 분포를 보여주는 히스토그램이 나오는데 peak가 2개이고 그 간격도 상당히 떨어져 있다.

이럴 경우는 대략적으로 2개의 peak 가운데 값으로 잘라주는 방식으로 쉽게 thresholding이 가능하다.



## Otsu Algorithm



임계값 설정에 대한 여러 알고리즘이 있지만, Otsu 알고리즘이 가장 자연스러운 임계값을 설정할 수 있게 해준다.

위의 경우 Peak의 경계가 명확하지 않기 때문에 어느 값을 기준으로 삼아야 할지 애매하다.

Otsu 알고리즘을 사용하면 알맞은 T값을 구할 수 있다.

임의의 임계값 T를 기준으로 하여 T보다 작은 픽셀의 비율을  $q_1$ 이라 하고, 같거나 큰 픽셀의 비율을  $q_2$ 라고 하자.

또한 T를 기준으로 만들어진 2개의 그룹에 대하여 평균과 표준편차를 구하고 각각  $\mu_1, \mu_2, \sigma_1, \sigma_2$ 라고 하자.

Otsu 알고리즘의 핵심 아이디어는 간단하다.

2개의 그룹에 대하여 그룹내 분산(within class variance)을 최소화하거나 그룹간 분산을 최대로 하는 방향으로 T를 정하면 가장 적절한 임계값을 구할 수 있다.

$$\text{그룹 내 분산: } \sigma_w^2 = q_1 \sigma_1^2 + q_2 \sigma_2^2$$

$$\text{그룹 간 분산: } \sigma_b^2 = q_1 q_2 (\mu_1 - \mu_2)^2$$

그룹 내 분산을 최소화하는 것과 그룹 간 분산을 최대화하는 것은 같은 의미를 갖는다.

하지만 그룹간 분산을 구하는 식이 좀 더 쉽기 때문에 일반적으로는 그룹 간 분산 방식을 더 많이 사용한다.

local thresholding

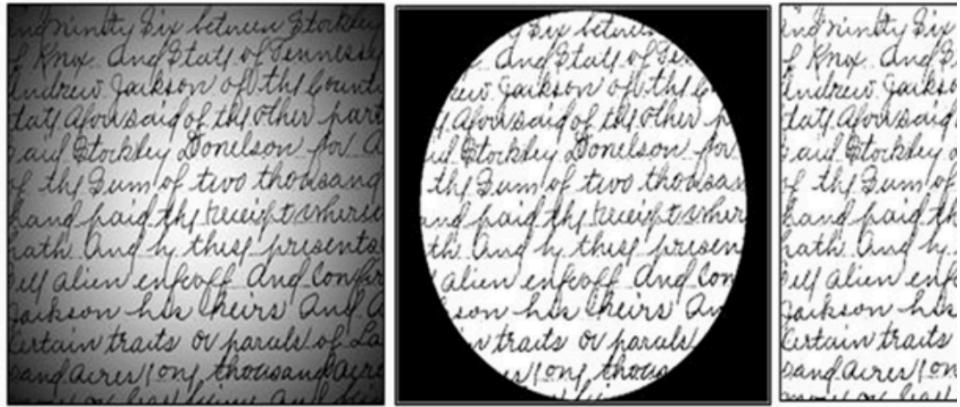


그림 전체에 동일한 임계값을 적용하는 경우에 조명이나 특정 잡음으로 인해 문제가 되는 경우가 있다.

그래서 전체 영역을 여러개로 나누고 각각의 영역에 대하여 적절한 thresholding을 하는 경우는 위와 같이 좋은 결과를 얻을 수 있다.

#### global thresholding

local thresholding의 반대이다. 전체 영역에 대해 동일한 thresholding을 수행한다.

#### Edge 기반 방법

Edge를 추출하는 필터 등을 사용하여 영상으로부터 경계를 추출한다.

엣지(edge)란 영상에 존재한 불연속적인 부분을 말하며, 대상으로는 밝기, 컬러, 표면의 무늬(texture) 같은 것들이 있다.

엣지 검출 방법은 매우 많으며 흔히 공간 필터(spatial filter)를 많이 사용한다.

#### Gradient

공간 필터의 역할은 영상으로부터 gradient를 추출한다.

gradient는 가장 변화의 속도가 빠른 방향을 나타낸다.

즉, 변화의 속도가 빠르니까 그곳은 엣지일 것이다라고 판단하는 것

$$\nabla I = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

x축과 y축에 대해 미분을 해주면 엣지를 찾을 수 있을 것이라고 기대

#### 1차 미분/2차 미분

엣지 검출이란 불연속한 부분을 찾아내는 것이기 때문에 흔히 1차 미분과 2차 미분을 사용한다.

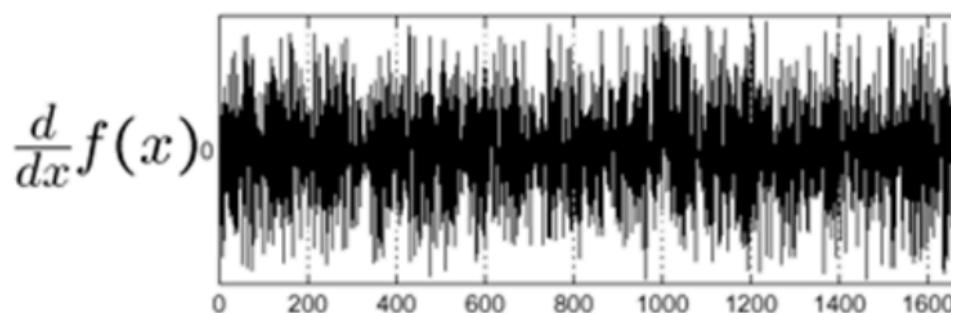
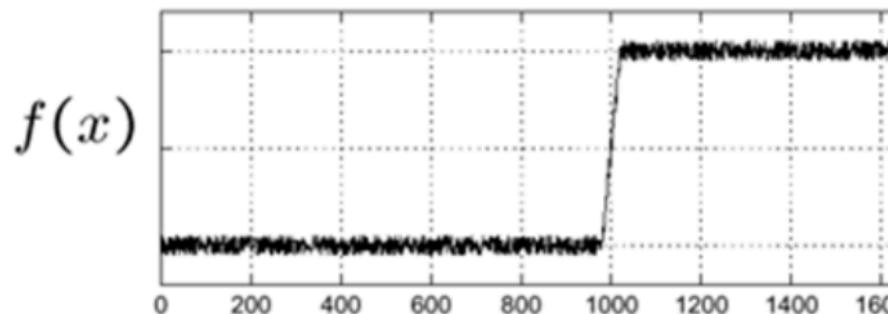
1차 미분을 사용하는 대표적인 필터로는 Prewitt, Robert, Sobel 등이 있으며, 많이 사용하는 Sobel 필터만 살펴보면 아래의 수식과 같으며 x와 y 방향의 엣지를 각각 검출한 후 합치는 과정을 거친다.

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, P_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

2차 미분을 사용하는 대표적인 공간필터는 Laplacian 필터가 있으며 수식은 아래와 같다.

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

잡음



잡음이 많은 경우는 미분을 하더라도 엣지 검출이 어려울수 있다.  
그래서 사전에 smoothing 필터 등을 사용하여 잡음의 영향을 줄이는 것이 필요하다.

#### 영역 기반 방법

Thresholding이나 Edge에 기반한 바식으로는 의미 있는 영역으로 구별하는 것이 쉽지 않으며, 특히 잡음이 있는 환경에서 결과가 좋지 못하다.

하지만 영역 기반 방법은 기본적으로 영역의 동질성(homogeneity)에 기반하고 있기 때문에 다른 방법보다 의미 있는 영역으로 나누는데 적합하지만 동질성을 규정하는 rule을 어떻게 정할것인가가 관건이 된다.

흔히 seed라고 부르는 몇개의 픽셀에서 시작하여 영역을 넓혀가는 region growing 방식이 여기에 해당된다.

이외에도 region merging, region splitting, split and merge, watershed 방식 등도 있다.

#### 엣지기반과의 차이

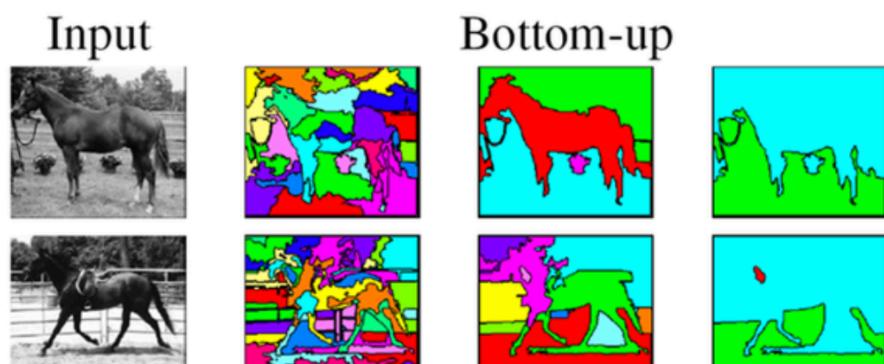
영상에서 차이(difference)가 나는 부분에 집중을 하였다면, 영역 기반 segmentation 방법은 영상에서 비슷한 속성을 갖는 부분(similarity)에 집중하는 방식이며, 엣지 기반이 outside-in 방식이라고 하면 영역 기반 방식은 inside-out 방식으로 볼수 있다.

여기서 비슷한 속성이란 흔히 사용하는 밝기뿐만 아니라, 컬러나 표면의 무늬등도 해당된다.

#### 영역 기반에서 엣지의 정의

객체의 경계를 다른 것들과 구별짓는 그 무엇이라고 본다.

### Image Segmentation 과정



#### Bottom-Up 방식

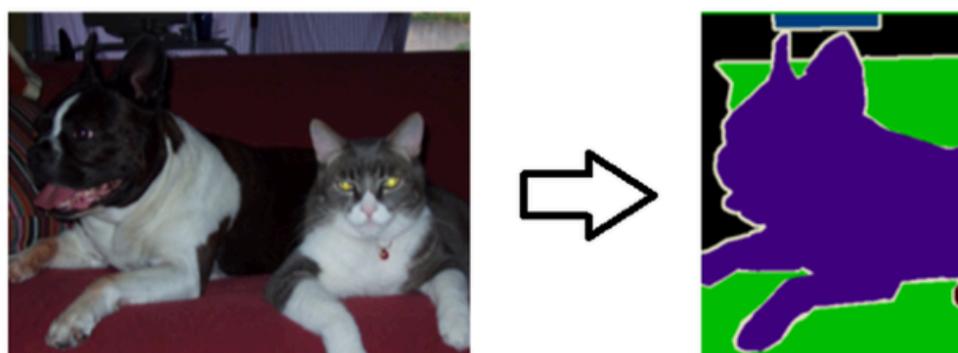
비슷한 특징을 갖는 것들끼리 집단화 하는 것을 말한다.

#### Top-Down 방식

같은 객체에 해당하는 것들끼리 집단화 하는 것을 말한다.

#### 의미적 분할(semantic segmentation)

영상에서 의미 있는 부분으로 구별해내는 기술



픽셀 단위의 예측을 수행하여 의미 있는 단위로 대상을 분리해낸다.

위 그림처럼 개와 고양이를 픽셀 단위로 구별해내는 기술이 바로 semantic segmentation

기술이다.

Semantic segmentation은 영상속에 무엇(what)이 있는지를 확인하는 것(semantic) 뿐만 아니라 어느 위치(where)에 있는지(location)까지 정확하게 파악을 해줘야 한다.

하지만 semantic과 location은 그 성질상 지향하는 바가 다르기 때문에 이것을 조화롭게 해결해야 semantic segmentation의 성능이 올라가게 된다.

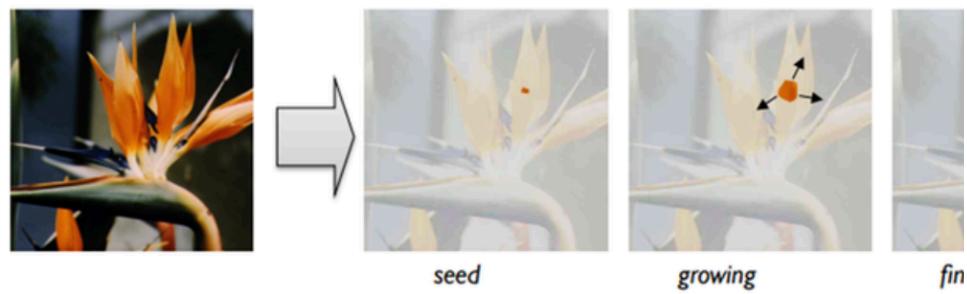
### (region segmentation)

#### (region-growing)

이 방식은 기준 픽셀을 정하고 기준 픽셀과 비슷한 속성을 갖는 픽셀로 영역을 확장하여 더 이상 같은 속성을 갖는 것들이 없으면 확장을 마치는 방식이다.

예)

먼저 임의의 픽셀을 seed로 정한 후 같은 속성(비슷한 컬러)을 갖는 부분으로 확장해나가면 최종 영역을 구별해 낼수 있게 된다.



#### seed 설정

시작 픽셀 설정

다음과 같은 종류가 있다.

- 사전에 사용자가 seed 위치를 지정
- 모든 픽셀을 seed라고 가정
- 무작위로 seed 위치를 지정

#### 영역 확장 방법

다음의 종류가 있다.

#### 원래의 seed 픽셀과 비교

영역 확장 시 원래의 seed 픽셀과 비교하여 일정 범위 이내가 되면 영역을 확장 하는 방법

#### 확장된 위치의 픽셀과 비교

원래의 seed 위치가 아니라 영역이 커지면 비교할 픽셀의 위치가 커지는 방향에 따라 바뀌는 방식

장점은 조금씩 값이 변하는 위치에 있더라도 같은 영역으로 판단이 되나, 한쪽으로만 픽셀값의 변화가 생기게 되면 seed와 멀리있는 픽셀은 값 차이가 많이 나더라도 같은 영역으로 처리될 수 있음

- (심각한 드리프트(drift) 현상)

예제)



#### 영역의 통계적 특성과 비교

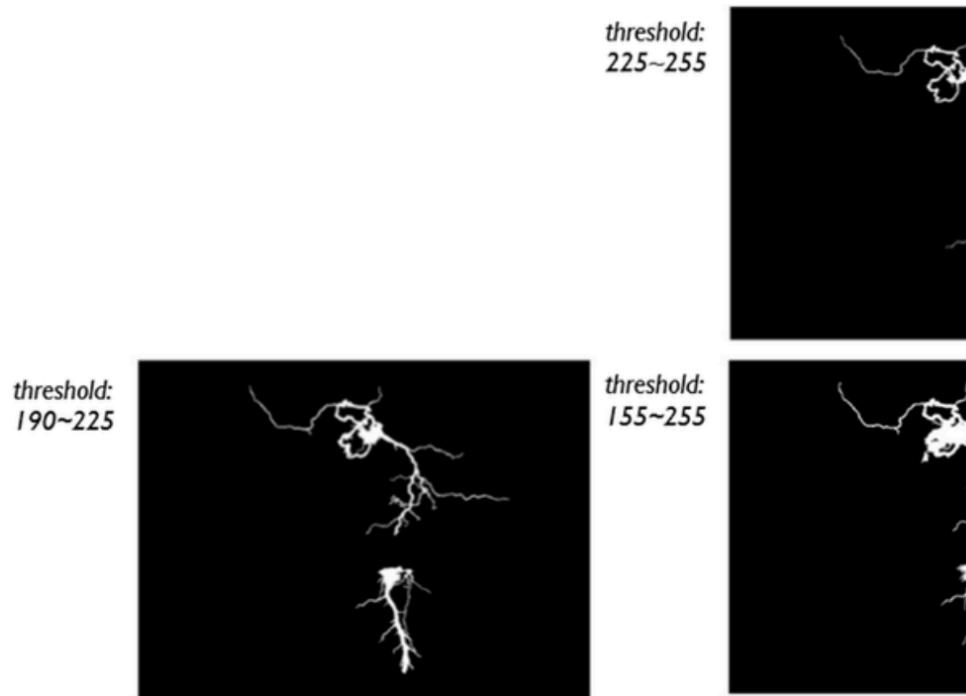
새로운 픽셀이 추가될 때마다 새로운 픽셀까지 고려한 영역의 통계적 특성(예를 들면 평균)과 비교하여

새로운 픽셀을 영역에 추가할 것인지를 결정.  
영역 내에 포함된 다른 픽셀들이 완충작용을 해주기 때문에 약간의 drift는 있  
을수 있지만 안전.  
centroid region growing이라고도 함

예제



가장 밝은 영역 만을 연결하고 싶다면, seed 값을 255로 설정하면된다.  
이렇게 설정된 seed를 바탕으로 임계값을 변경시키면서 region을 확장하면 아래와  
같은 결과를 얻는다.



### 장점

처리 속도가 빠르다  
개념적으로 단순하다  
seed 위치와 영역 확장을 위한 기준 설정을 선택할 수 있다.  
동시에 여러 개의 기준을 설정할 수도 있다.

### 단점

영상의 전체를 보는 것이 아니라 일부만 보는 지역적 방식(local method)이다.  
알고리즘이 잡음에 민감하다.  
seed 픽셀과 비교하는 방식이 아니면 drift현상이 발생할 수 있다.

### (Region Merging)

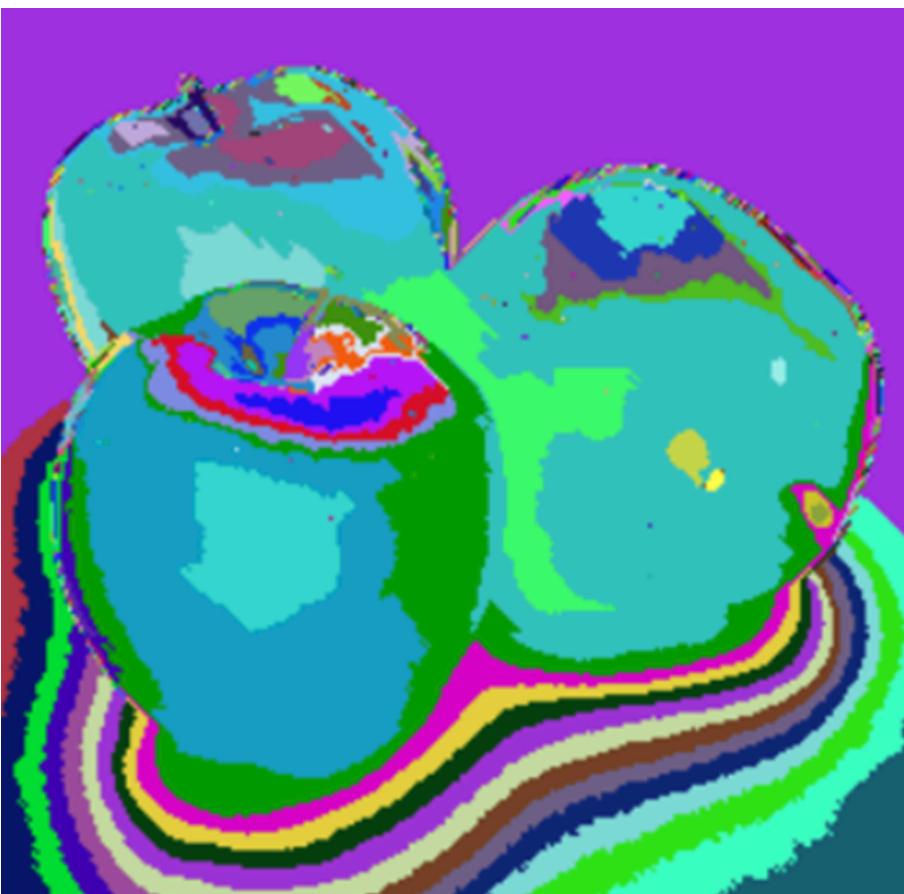
비슷한 속성을 갖는 영역들을 결합시켜 동일한 꼬리표(label)를 달아주는 방식이다.  
region merging은 어떤 경우는 매 픽셀 단위가 될 수 있으며, 일반적으로 심하게 나눈 영역  
(over-segmented region)을 시작점으로 하며, 일반적으로 아래와 같은 과정  
을 거친다.

1. 인접 영역을 정한다.
2. 비슷한 속성인지 판단할 수 있는 통계적인 방법을 적용하여 비교한다.
3. 같은 객체라고 판단되면 합치고, 다시 통계를 갱신한다.
4. 더 이상 합칠것이 없을 때 까지 위 과정을 반복한다.

Region growing은 region merging 방법 중 하나이며 region growing 방법은 1개 혹은 적은 수의 seed를 사용하는 방식으로 픽셀 단위로 판단을 하는 점만 차이가 있다.  
반면에 merging은 영역을 기본 단위로 하며, 물론 가장 작은 영역은 픽셀이기 때문에 픽셀  
을 기본 영역으로 볼수 있음, 그림 전체에 여러 개의 seed 를 사용한다고 볼수  
있다.

### ex)

매 픽셀에 각각의 꼬리표를 할당하고 이것을 기반으로 하여 밝기 값의 차가 10이하이면 동  
일한 영역으로 본다고 가정하고,  
merging 알고리즘을 충실히 적용을 하여 분리를 하면 아래 그림과 같다.



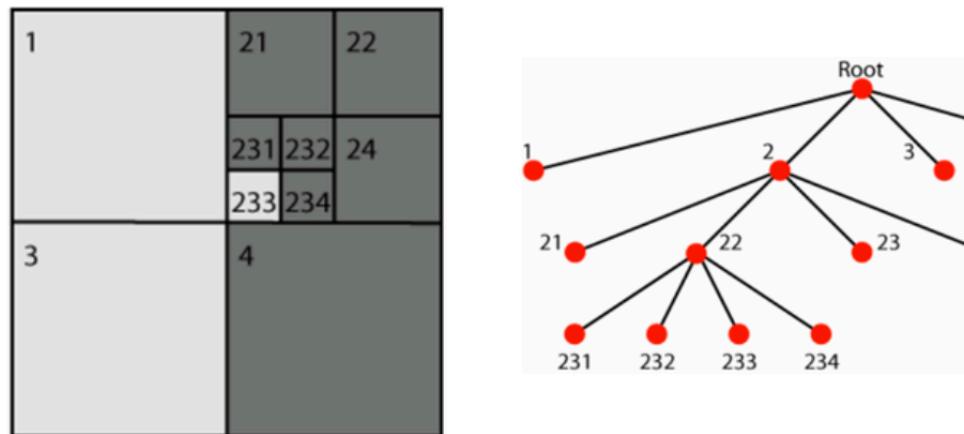
### (Region Splitting)

merging과는 정 반대의 개념이다.

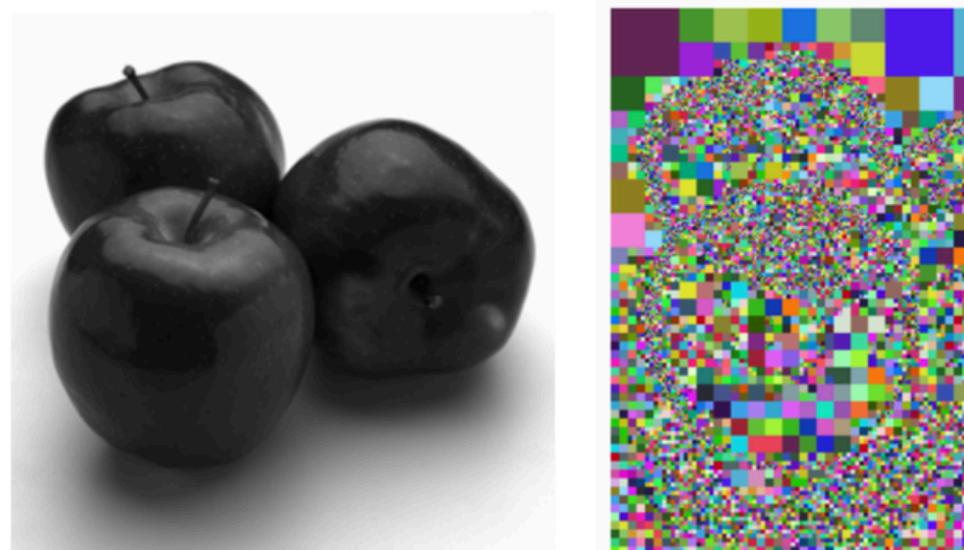
그림 전체와 같은 큰 영역을 속성이 일정 기준을 벗어나면서 쪼개면서 세분화된 영역으로 나누는 방식을 사용한다.

보통은 아래 그림과 같이 4개의 동일한 크기를 갖는 영역으로 나누기 때문에 quad-tree

splitting 방식을 많이 사용한다.  
 큰 영역을 먼저 4개의 영역으로 나누고, 다시 각 영역을 검토하여 추가적으로 나눠야 할것인지를 결정한다.  
 그림에서는 2번 영역이 추가로 나눠야 하기 때문에 21부터 24까지 다시 4개의 영역으로 나눈다.  
 이렇게 region splitting 방식에서는 해당 영역에서 분산이나 최대값과 최소값의 차와 같은 통계 방식의 기준을 설정하고,  
 그 값이 미리 정한 임계 범위를 초과하게 되면 영역을 분할한다.



예)



segmentation이 잘 안되었다고 볼수 있을것 같은데, 이는 splitting 방식의 원리 때문에 같은 속성을 갖는 부분도 다른 꼬리표가 붙게 되기 때문이다.

### (Split & Merge 방법)

splitting 방식만으로는 원하는 결과를 얻기가 어렵기 때문에 동일한 영역을 다시 합쳐주는 과정을 거쳐야 한다.

이것이 바로 split & Merge이다.

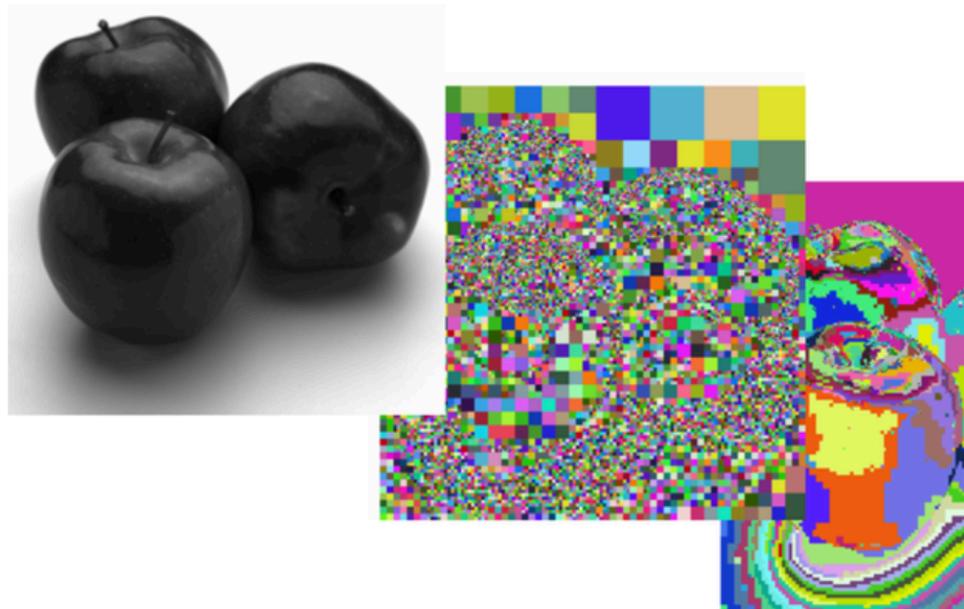


Quad-tree splitting 방식을 사용하면 실제로는 모양이 위 그림과 같더라도 여러 개의 영역으로 분리될 수 밖에 없기 때문에 over-segmentation이 일어난다.

이것을 같은 특성을 갖는 부분끼리 다시 묶어주면, {1, 3, 233}과 {4, 21, 22, 24, 231, 232, 234} 두개의 영역으로 된다.

이것을 split & Merge라고 한다.

(적용 결과 예)



#### region merging과의 차이점

split & merge가 좀 더 속도가 빠르다는 점이다.

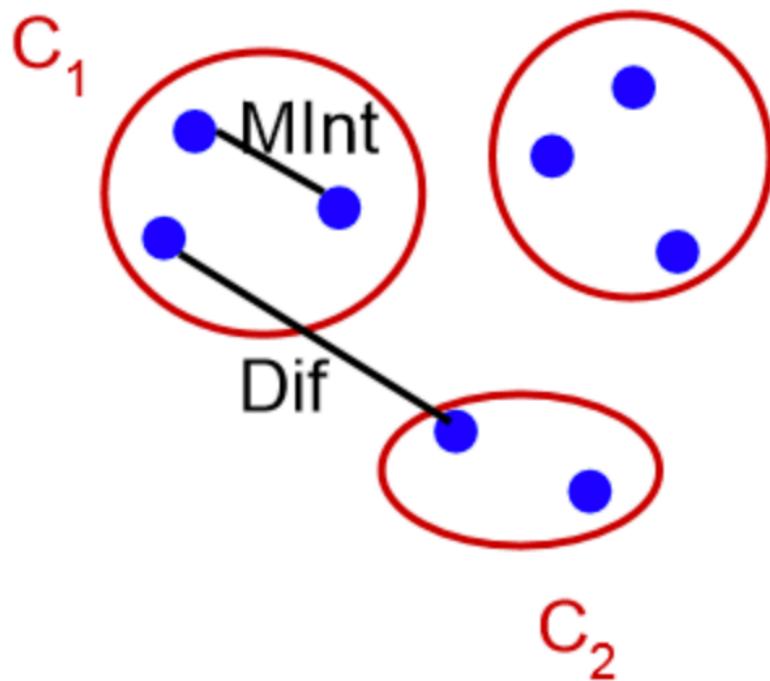
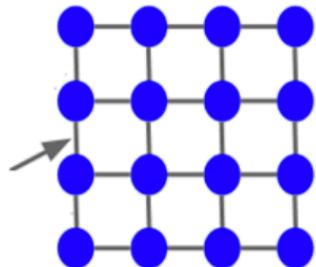
단순 merge가 막고 품는 것과 같다면, quad-tree splitting을 적용하면 비슷한 영역은 통으로 처리되기 때문에 좀더 속도가 빠르다.

#### Felzenszwalb and Huttenlocher Segmentation

<https://laonple.blog.me/220925179894>

코렐대의 Felzenszwalb는 2004년에  
인지적인 관점에서 의미 있는 부분을 모아서 그룹화와 연산량 관점에서 효율성 증대 라는 2개의  
목표를 기반으로 segmentation 기법을 개발하였다.  
논문에서는 사람이 인지하는 방식으로의 segmentation을 위해 graph 방식을 사용하였다.  
이 방식에서는 픽셀들 간의 위치에 기반하여 가중치( $w$ )를 정하기 때문에 grid graph 가중치 방식  
이라고 부르며, 가중치는 아래와 같은 수식으로 결정이 되고  
graph는 상하좌우 연결된 픽셀에 대하여 만든다.  
 $E(\text{edge})$ 는 픽셀과 픽셀의 관계를 나타내며 가중치  $w(v_i, v_j)$ 로 표현이 되는데, 위 식에서 알 수 있  
듯이 가중치는 픽셀간의 유사도가 떨어질수록 큰 값을 갖게 되며, 결과적으로  
 $w$ 값이 커지게 되면 영역의 분리가 일어나게 된다.

$$w(v_i, v_j) = |I(p_i) - I(p_j)|$$



위 그림과 같이  $C_1$ 과  $C_2$ 가 있는 경우에, 영역을 분리할것인지 혹은 통합할것인지를 판단하는 아래  
와 같은 수식을 이용한다.

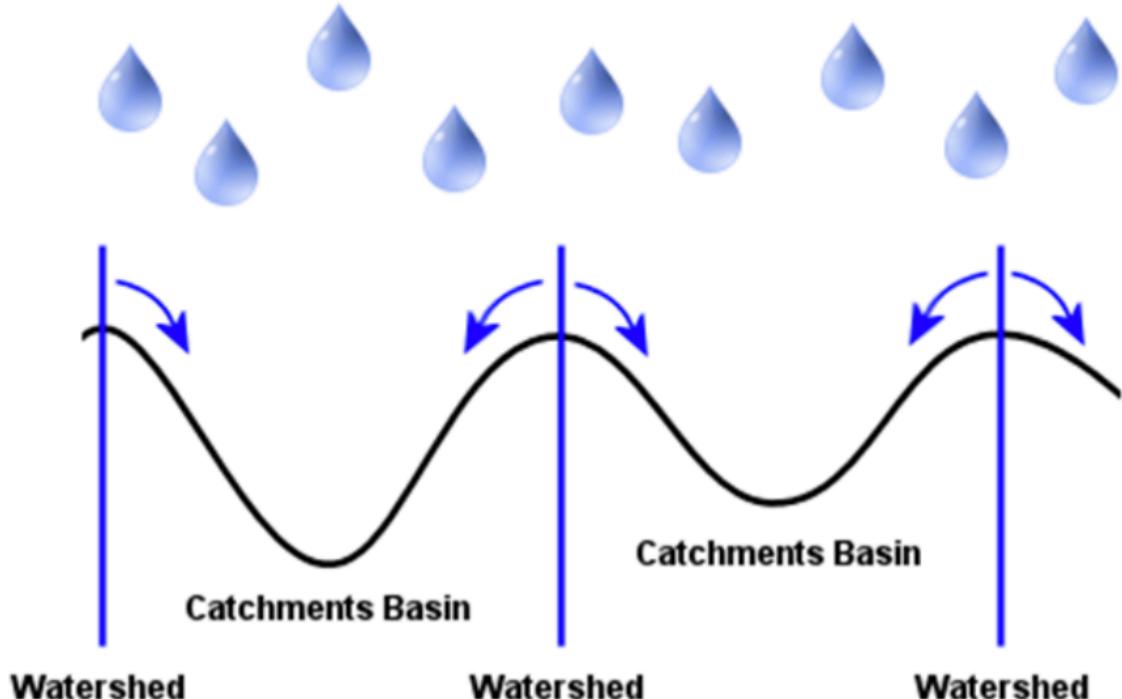
$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

위식에서  $Dif(C_1, C_2)$ 는 두개의 그룹을 연결하는 변의 최소 가중치를 나타내고  
 $MInt(C_1, C_2)$ 는  $C_1$ 과  $C_2$  그룹에서 최대 가중치 중 작은 것을 선택한 것이다.  
즉, 그룹간의 차가 그룹 내의 차보다 큰 경우는 별개의 그룹으로 그대로 있고, 그렇지 않은 경우에  
는 병합을 하는 방식이다.

논문에서는 인접한 픽셀끼리 공간적 위치 관계를 따지는 방법뿐만 아니라, feature space에서의 인접도를 고려한 방식도 제안하였다.

이 방식은 nearest neighbor graph 가중치 방식이라고 부르며, 적정한 연산 시간을 유지하기 위해 feature space에서 가장 가까운 10개의 픽셀에 한하여 graph를 형성한다.

### (Watershed Segmentation)



Watershed는 산등성이거나 능선처럼 비가 내리면 양쪽으로 흘러 내리는 경계에 해당이 되며, Catchment Basin은 물이 흘러 모이는 집수구역에 해당이 된다. Watershed는 기본적으로 영역을 구분해주는 역할을 하기 때문에 Watershed만 구하면 영상을 Segmentation할 수 있게 된다.

#### Watershed를 구하는 방법

##### Flooding 알고리즘

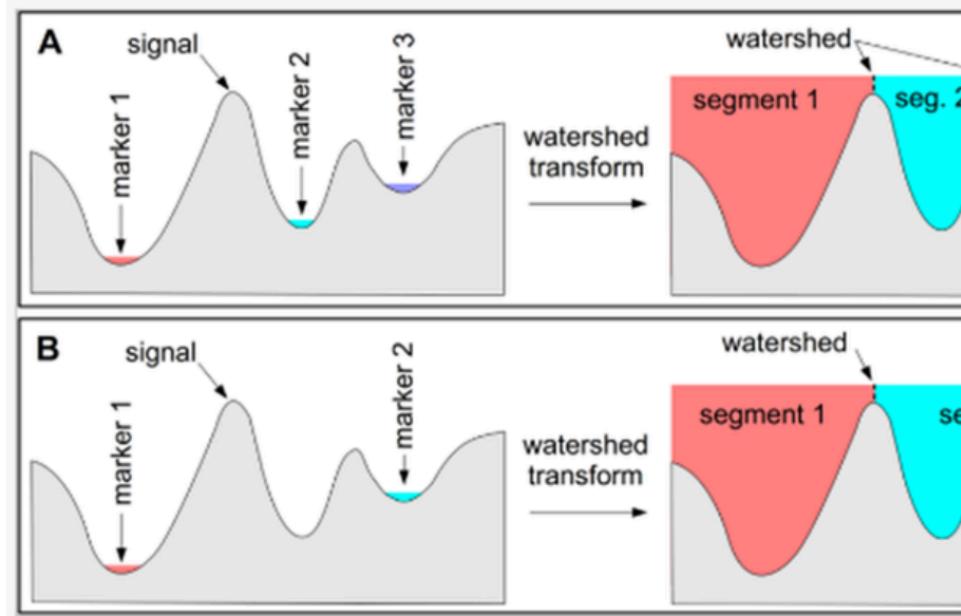
이 알고리즘은 기본적으로 각각의 Catchment Basin의 최소값(local minima)에 구멍이 있고, 그 구멍에서 물이 차오르기 시작한다고 가정하는데서 출발한다.

물이 일정 수위에 오르게 되면, 서로 다른 2개의 Catchment Basin이 합쳐지는 상황이 발생할 수 있는데, 이때는 물이 합쳐지는 것을 막기 위해 댐(dam)을 설치한다.

이 댐이 바로 영역을 구별하는 영역의 경계(boundary) 역할을 하게 되며 이것이 Flooding 알고리즘의 기본 개념이다.

아래 그림 A는 3개의 basin에 각각 마커를 할당한 경우로, 각 basin에서 물이 점점 차오르면 어느 순간에 2개의 basin이 합쳐지는 상황이 발생하게 되는데, 그러면 그 자리에 댐을 건설한다.

B는 애초에 2개의 마커만을 할당하였기 때문에 최종적으로는 영역이 2개로 나뉘지게 된다.



### Marker-Controlled Watershed Segmentation

자연 영상에 watershed 알고리즘을 적용하면 대부분 over-segmentation 문제가 발생한다.

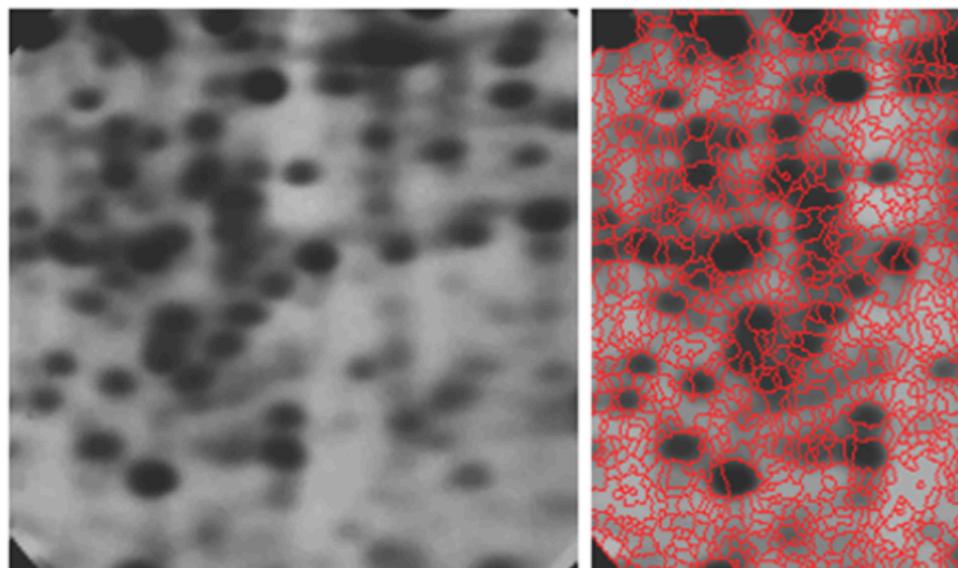
(영상에 존재하는 detail이나 잡음이 local minimum을 만들기 때문)

이 문제를 피하기 위한 방법으로 마커(marker)를 사용하여 segmentation 될 영역을 지정하는 방식이다.

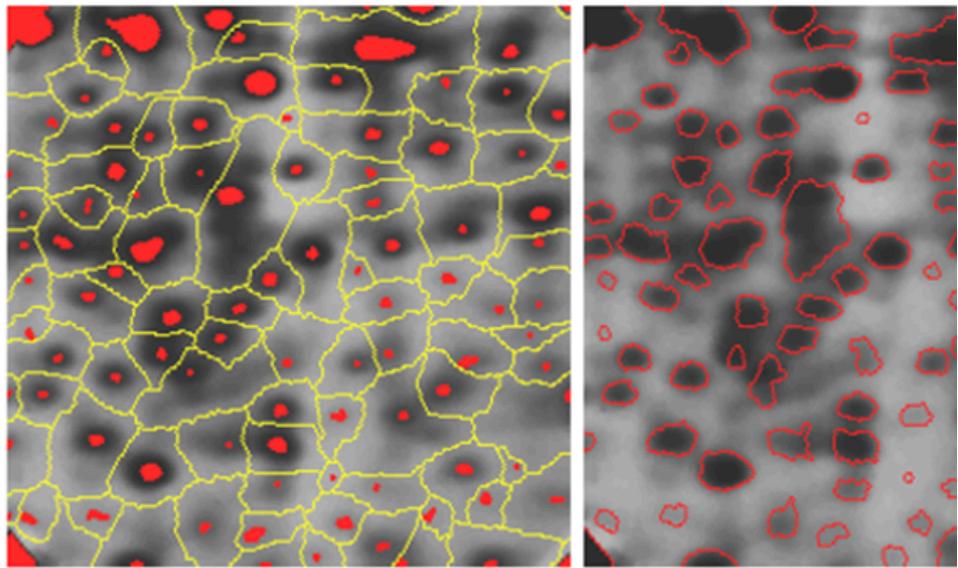
마커의 지정은 수동이나 자동으로 가능하며, 보통은 최종적으로 segmentation 되는 영역의 marker의 수와 동일하다.

마커를 지정할 때는 보통 blob 연산을 많이 사용하며, 모포로지 연산을 같이 사용하는 것이 일반적인 추세이다.

아래 그림에서 왼쪽은 원래 영상이며, 오른쪽은 watershed 알고리즘을 적용하여 얻은 over-segmented image이다.



이 문제 해결을 위해 blob 이미지에 marker를 할당하면 아래 그림의 왼쪽이 되며, marker가 할당된 영역에 대해서만 watershed 알고리즘을 적용한 결과는 다음과 같다.



### 예지 강조(Edge Enhancement)

#### Roberts Operator

Roberts Operator는 대각선으로 인접한 화소 끼리의 차를 취하여 1차 미분에 근사한 값을 구하는 방식이다.

즉, 인접한 픽셀의 차분을 통해 미분의 근사를 하는건데, 이때 변화량을 1만큼 주기 때문에 차분만의 식으로 미분식이 이루어짐

$$G[f[x, y]] = |f[x, y] - f[x+1, y+1]| + |f[x+1, y] - f[x, y+1]|$$

$$G[f[x, y]] = |G_x| + |G_y| \text{ 와 같이 됩니다.}$$

즉,

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

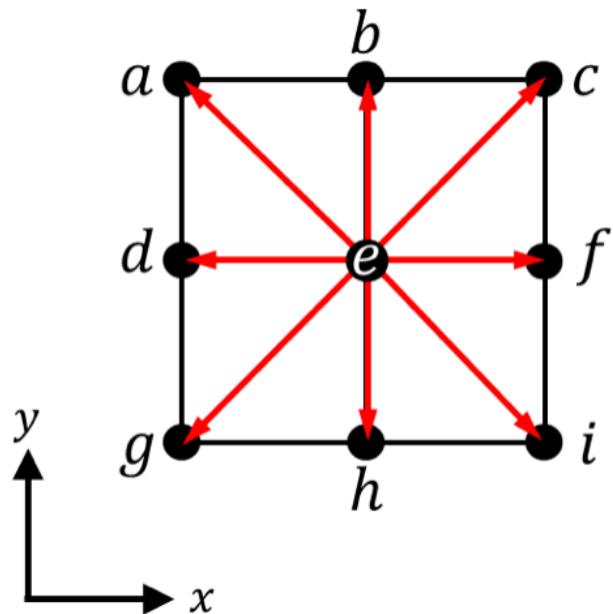
마스크와 convolution 연산을 통해 각 대각선의 pixel 차를 구해 edge를 구하는 방식이 Roberts detection algorithm이다.

#### Sobel

##### Deriving the sobel kernels

커널에 대응하는 픽셀이 다음과 같을때 편미분은 다음과 같이 정의 된다.  
중앙 픽셀과 인접 픽셀간의 차이 / distance

$a$	$b$	$c$
$d$	$e$	$f$
$g$	$h$	$i$



e와 b의 directional derivate는 다음과 같이 정의된다.  $(e - b) / 1$

f와 e의 directional derivate는 다음과 같이 정의된다.  $(f - e) / 1$

즉 x축과 y축에 대해서는 거리가 1이기 때문에 difference만 남음

#### 방향도함수 표현

예시를 들면 i가 x의 positive 방향으로의 유닛 벡터

j가 y의 positive 방향으로의 유닛 벡터라고 할때

e를 뒤에다 쓰는걸로 표현하면,

f와 e는 다음과 같이 표현된다.  $\rightarrow (f - e)i$

d와 e는 다음과 같이 표현된다.  $\rightarrow -(d - e)i$

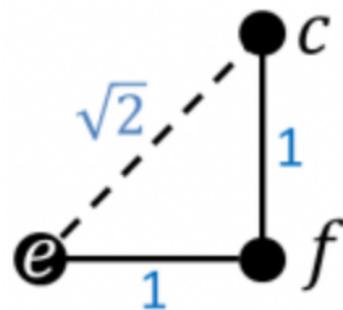
e와 b는 다음과 같이 표현된다.  $\rightarrow (b - e)j$

h와 e는 다음과 같이 표현된다.  $\rightarrow -(h - e)j$

이는.. 특정 방향으로 특정한 걸음(차분)만큼 간다는 표현임

#### 대각선 방향으로의 미분은 어떠한가?

e와 c의 미분



루트 2로 나눠 주는 이유는 미분을 구할때 거리를 나눠줘야 하기 때문

$$\frac{1}{\sqrt{2}}[(c - e) \cos 45^\circ \hat{i} + (c - e) \sin 45^\circ \hat{j}]$$

$\cos 45 = \sin 45 = 1 / \text{root}(2)$  이므로 다음과 같이 표현됨

$$\frac{1}{2}(c - e)(\hat{i} + \hat{j})$$

종합

모든 방향에 대해 미분에 대한 합

다음과 같이 표현됨

$$[(f - d) + \frac{1}{2}(-a + c - g + i)]\hat{i} \\ + [(b - h) + \frac{1}{2}(a + c - g - i)]\hat{j}$$

i와 j는 G\_x, G\_y로 표현한다면

다음과 같이 커널을 만들수 있음

$$G_x = \frac{1}{2} \begin{array}{|c|c|c|} \hline & -1 & 0 & 1 \\ \hline -1 & & & \\ \hline -2 & 0 & 2 & \\ \hline -1 & 0 & 1 & \\ \hline \end{array}$$

$$G_y = \frac{1}{2} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

sobel의 논문 원본에 따르면, gradient를 계산하는데 8개의 픽셀을 사용했기 때문에(neighbor의 수)

커널에 1/8을 곱해야 한다고 한다.

결국 앞에 있는 계수는  $1/2 * 1/8 = 1/16$

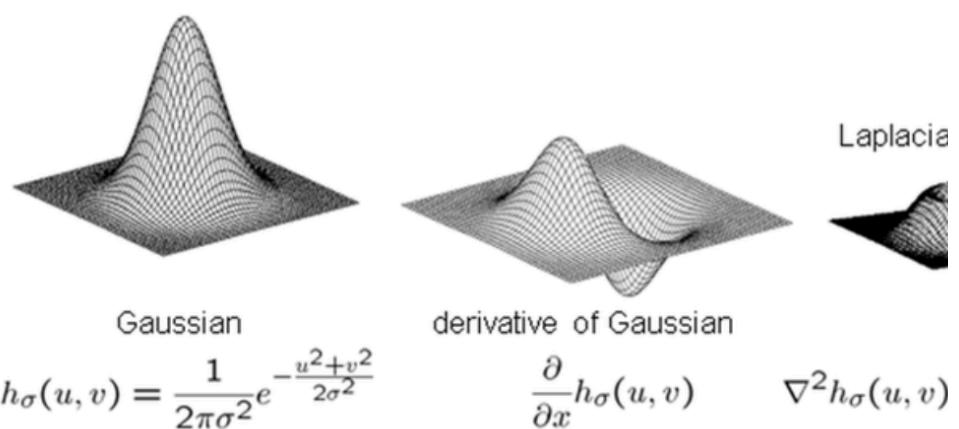
그러나 보통 이 계수를 생략한다.

종합적인 gradient는 다음과 같이 계산한다

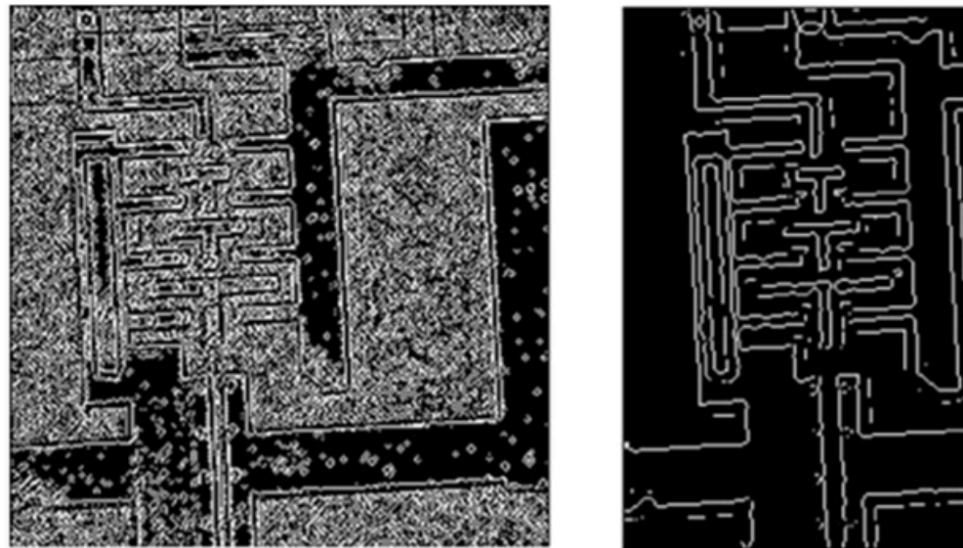
$G_x$ 와  $G_y$ 의 합에 대한 magnitude를 구함

$$G = \sqrt{G_x^2 + G_y^2}$$

### LoG 필터(Laplacian of Gaussian, LoG)



가우시안 필터를 이용하여 잡음을 제거하고 여기에 2차 미분 필터인 라플라시안 함수를 적용하면 아래 그림처럼 큰 엣지를 비교적 쉽게 찾아낼수 있다.



#### post-processing

공간필터를 통해 검출된 엣지는 곧바로 사용하기에는 부적절하기 때문에 후처리를 필요로 한다.

#### (Thresholding)

일정 크기 이하의 엣지들을 제거하여 큰 엣지 위주로 정리

#### (Thinning)

non-maximum suppression을 적용하여 굵은 엣지를 얇게 처리

#### Edge thining

검출한 엣지를 1픽셀 크기로 얇게 만드는 것을 말한다.

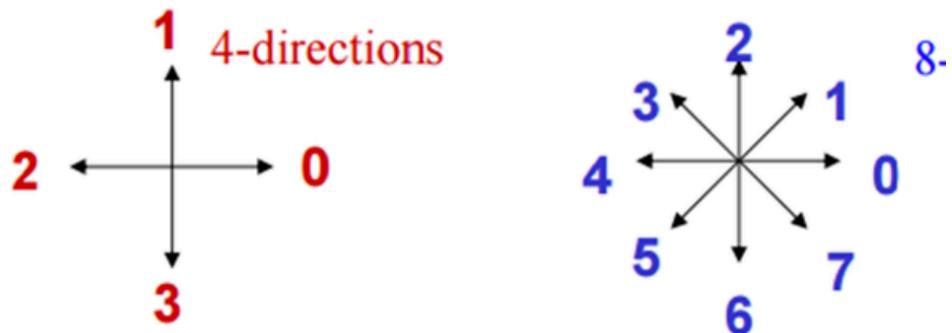
Thining을 하는 이유는 얇고 선명한(shape) 엣지가 객체 등의 응용에서 훨씬 유용하기 때문이며,

허프 변환(Hough transform)을 이용하여 직선이나 타원등을 검출할 때도 얇은 경우에 결과가 더 좋게 나오기 때문이며

때로 엣지가 객체의 경계에 위치하고 있는 경우는 객체의 perimeter를 구할 때 복잡한 공식을 사용하지 않고도 쉽게 할 수 있기 때문이다.

#### 방향

대체로 4방향 혹은 8방향을 사용한다.

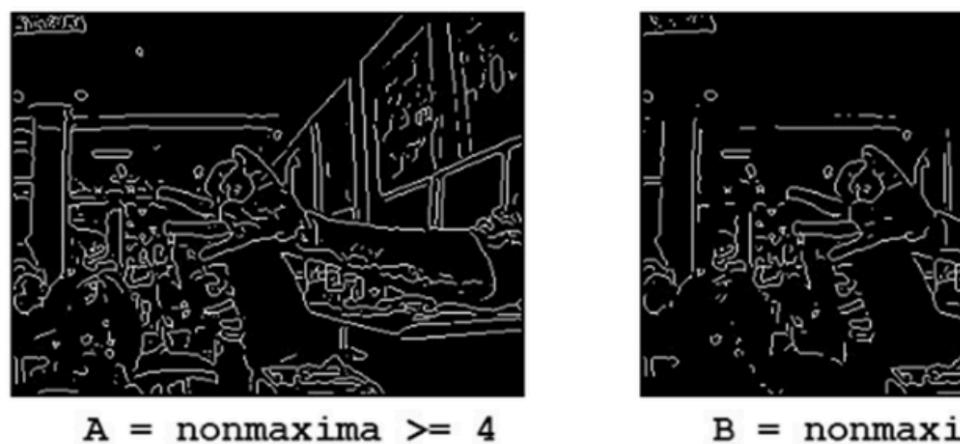


방향을 정했다면, 중심 픽셀의 엣지값과 주변 픽셀의 엣지값을 비교하여 중심픽셀이 가장 큰 값인 경우는 남겨두고 그렇지 않은 경우는 제거한다.

그래서 이것을 non-maximum suppression 이라고도 부른다.  
 ex)  
 4방향을 설정하고 3x3 윈도우를 적용 (맨왼쪽위부터 sliding window를  
 적용)

#### (Hysteresis thresholding)

1개의 threshold를 이용하는 대신에, high와 low 2개의 threshold를 사용하며, high threshold 옆지에 연결된  
 low threshold 옆지는 제거하지 않고 살려두는 것을 말한다.



#### Canny Edge Detector

1986년에 J.Canny가 발표한 엣지 검출기이다.  
 성능이 뛰어나기 때문에 가장 많이 사용되는 엣지 검출 알고리즘중 하나이다.

### 최적 엣지 찾기

- 잡음 제거 또는 평활화(Noise smoothing)
  - 엣지 개선(Edge enhancement)
  - 엣지 위치 파악(Edge localization)
- 위 세가지 특성이 좋아야 한다.

### 상세 과정 - Smoothing, Gradient 구하기

1. 잡음의 영향을 최소화하기 위해 가우시안 필터를 적용하여 영상을 부드럽게 만든다.

2. gradient를 구하기 위해 x 및 y방향으로 미분을 한다.

원래는 가우시안 필터링을 하고 미분을 하는 것이 맞지만, 가우시안 함수의 convolution 특성상 미분한뒤 convolution해도 상관없다.

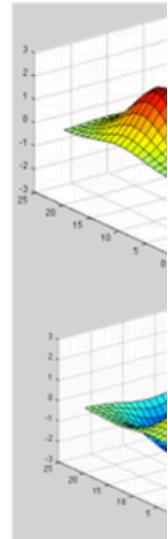
$$S = \nabla(g * I) = (\nabla g) * I =$$

$$= \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix} \quad \nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix}$$

가우시안 함수를 미분한 수식은 아래와 같으며 여기서  $\sigma$ 는 연산을 적용할 커널의 크기를 결정한다.

$$h_x(x, y) = \frac{\partial h(x, y)}{\partial x} = \frac{-x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$h_y(x, y) = \frac{\partial h(x, y)}{\partial y} = \frac{-y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

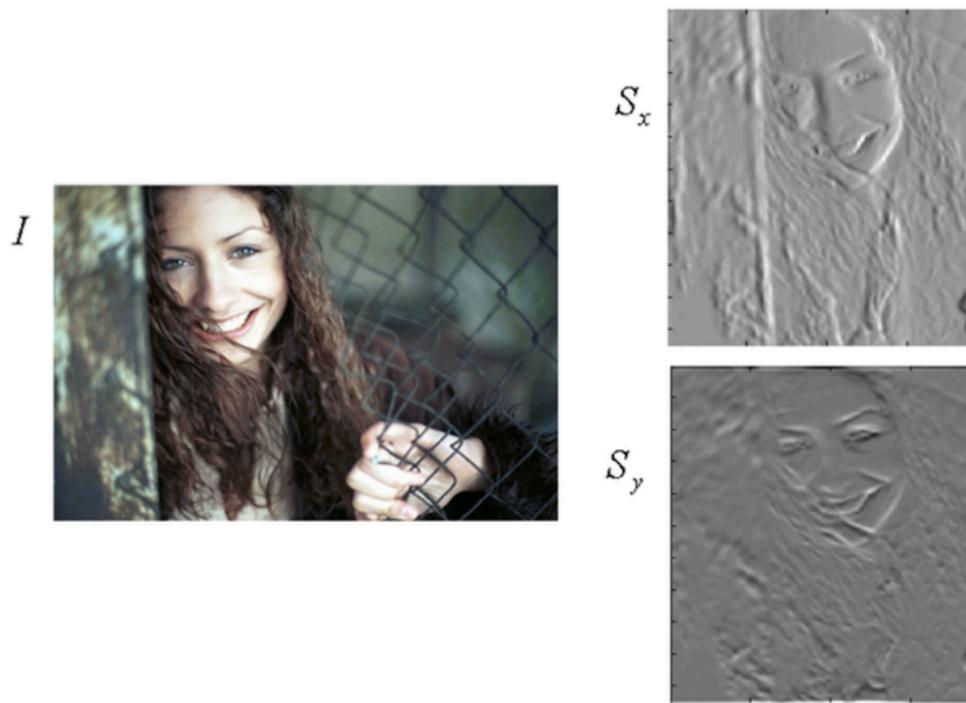


Scale

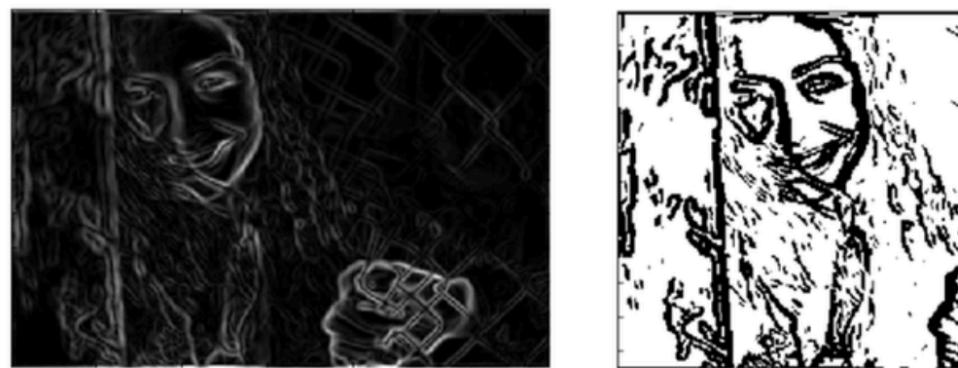
이 식에서 scale( $\sigma$ )은 smoothing이 되는 정도를 결정하는 변수이며 이 값이 커지면 커질수록 아래와 같은 성질을 갖는다.

- 잡음 엣지를 더 많이 제거할수 있게 됨
- 엣지를 더 부드럽게 하면서 두껍게 만드는 경향이 있음
- 영상에 있는 섬세한 detail들을 제거

이런 점들을 고려하여 smoothing된 영상에 대하여 x, y 방향의 미분을 구하면 아래 그림과 같이 된다.



이 결과를 이용해 gradient의 크기와 방향을 구하고 이렇게 구해진 엣지를 thresholding을 통해 임계값 이상만을 보여주면 아래와 같아진다.



$$|\nabla S| = \sqrt{S_x^2 + S_y^2}$$

$$|\nabla S| \geq \text{Threshold}$$

#### 상세 과정 - Thinning (Non-maximum suppression)

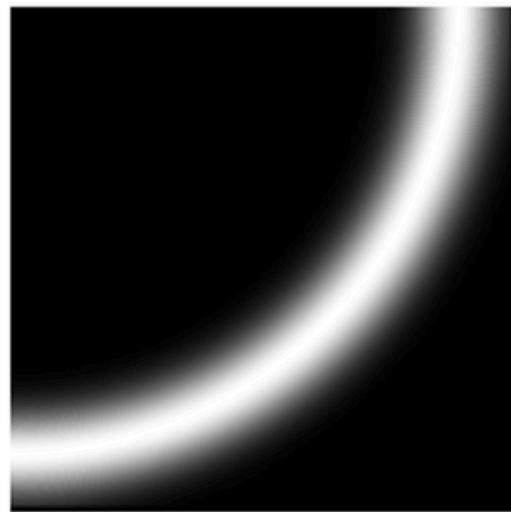
첫번째 과정을 통해 gradient의 방향을 구했기 때문에, gradient의 변화의 방향을 이 미 알고 있으므로

변화하는 방향으로 스캔을 하면서 최대값이 되는 부분을 찾는다.

(즉 엣지가 나올만한 방향을 찾았기 때문에 그 방향으로만 비교하면 됨 - 4방향, 8방향 필요없이)

왼쪽과 같이 gradient magnitude 영상이 얻어졌고, gradient의 방향이 오른쪽의 직선과 같은 방향이라면,

직선 방향으로 스캔을 하면서 가장 큰 값만 남기고 픽셀값을 모두 0으로 만든다. 그러면 1픽셀 단위의 얇은 선이 만들어지게 된다.



thinning -> thresholding 적용

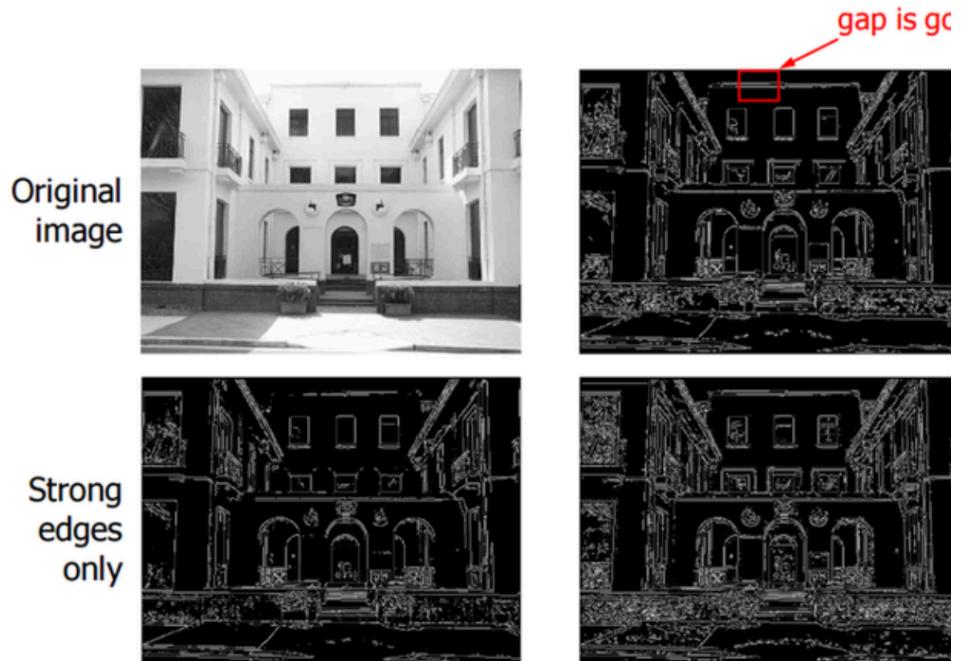


$$|\nabla S| = \sqrt{S_x^2 + S_y^2}$$

$M \geq Threshold = 25$



상세 과정 - Hysteresis  
히스테리시스를 적용한다.



### SUSAN(smalles Univalue Segment Assimilating Nucleus, smallest usan)

low-level 영상 처리에 대한 새로운 접근법 제시

엣지 검출기뿐만 아니라, 잡음 제거에도 괜찮은 성능을 보인다.

#### 새로운 접근법

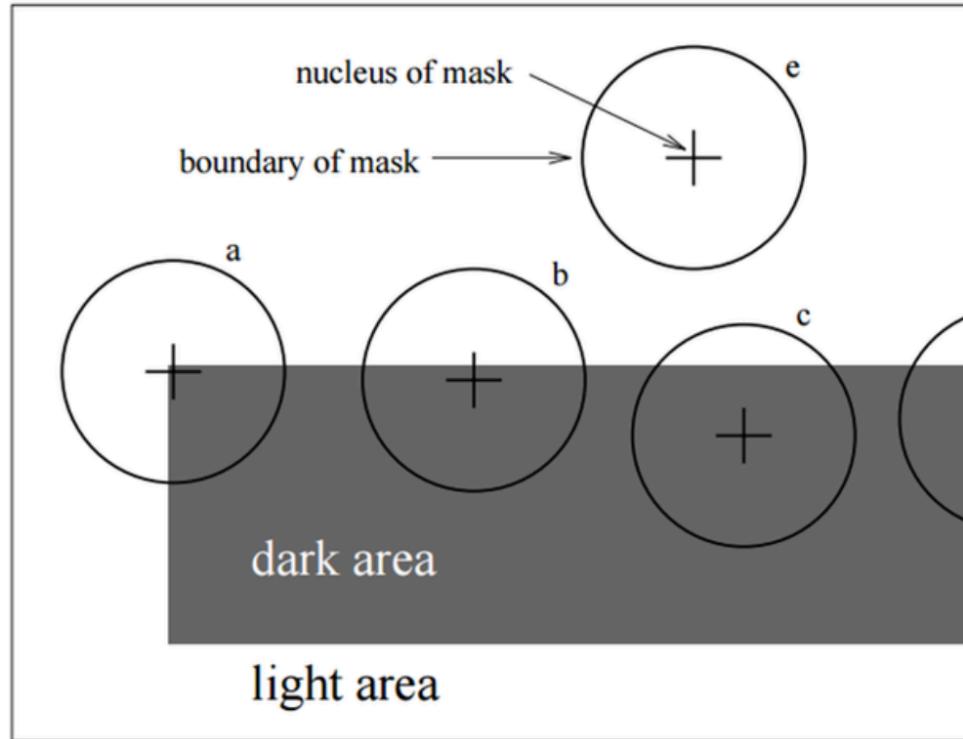
대부분의 엣지 검출 알고리즘은 미분을 사용하지만 SUSAN에서는 전혀 그렇지 않다.

대부분의 영상 처리에 관련된 알고리즘들이  $3 \times 3$  혹은  $5 \times 5$ 와 같은 정방형 윈도우 혹은 마스크를 사용하지만

SUSAN에서는 원형 또는 근접한 원형 마스크를 사용하는 점도 다르다.

#### USAN(Univalue Segment Assimilating Nucleus)

+ 부호의 위치에 있는 픽셀이 마스크의 중심이 된다.



위 그림은 5가지 종류의 마스크 형태를 보여준다.

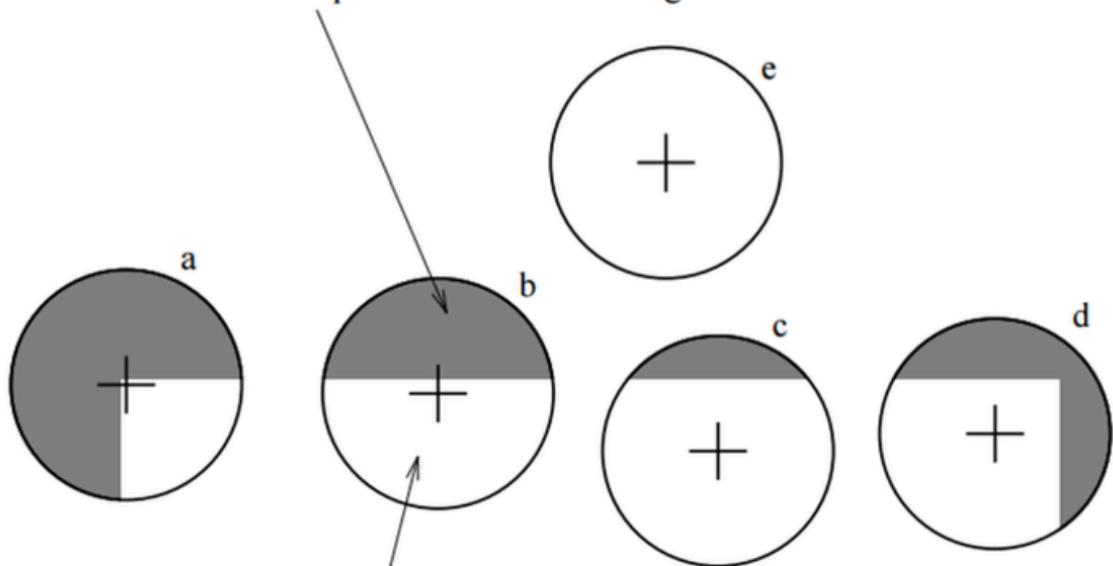
e는 Nucleus와 주변 픽셀이 모두 같은 경우로 edge나 corner가 아니지만, 나머지 “a ~ d”는 edge나 corner가 마스크에 포함된 경우이다.

USAN은 전체 마스크에서 Nucleus와 같은 (혹은 거의 비슷한) 값을 갖는 면적을 뜻한다.

이 면적을 살피면 평탄한 지역에 있는지, 엣지나 코너에 영역에 있는지 파악이 가능하다.

다음은 USAN 영역에 해당하는 부분을 나타낸 그림으로 Nucleus와 같은 부분은 흰색으로, 다른 부분은 검은색으로 표시를 한것이다.

section of mask where pixels have different brightness to nucleus



section of mask where pixels have same brightness as nucleus

엣지나 코너 검출에는 다음과 같은 기본식이 적용된다.

$$c(\vec{r}, \vec{r}_0) = \begin{cases} 1 & \text{if } |I(\vec{r}) - I(\vec{r}_0)| \leq t \\ 0 & \text{if } |I(\vec{r}) - I(\vec{r}_0)| > t, \end{cases}$$

$r$ 은 마스크 영역에 있는 다른 픽셀 위치를 나타내고  $r_0$ 는 Nucleus를 나타낸다.

$t$ 는 유사도를 나타내는 threshold 값이다.

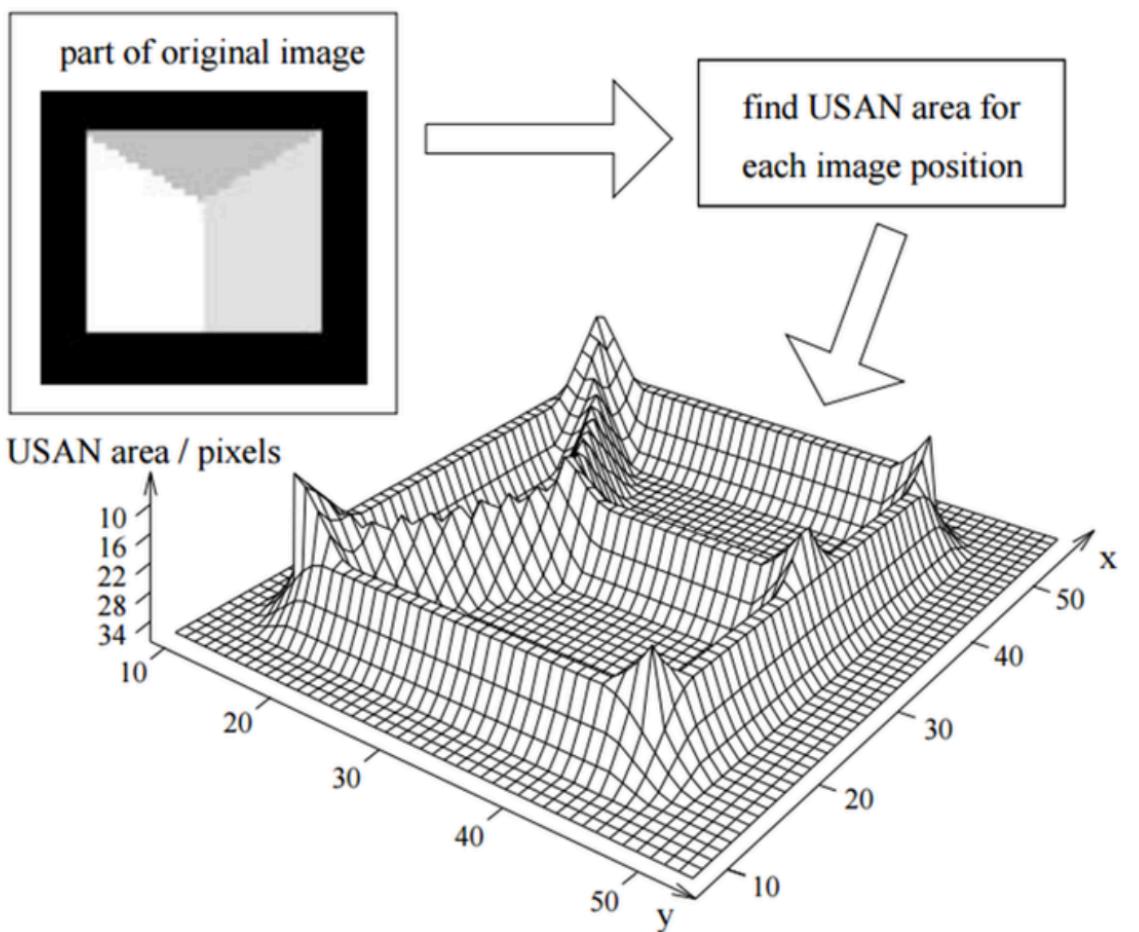
그래서, 중심에 있는 픽셀의 밝기와 임계값( $t$ ) 이내로 비슷한 경우에는 1이 되고, 그 이상의 차이를 보이는 경우는 0이되는 비선형 필터이다.

$c$ 는 결과적으로 전체 마스크 내에서 중심과 임계값 범위에 있는 픽셀의 개수를 의미한다.

USAN 엣지 검출기에서 사용되는 마스크는 반지름이 3.4를 많이 사용하며 이 경우 마스크에 있는 픽셀의 개수는 37개이다.

최소 마스크의 크기는 3x3이다.

(예제)



균일한 영역은 USAN 값이 크고(중심값이랑 비슷한게 많다)

엣지나 코너로 갈수록 USAN 값이 작아지는 특성이 있으며, 시인성(visibility)을 높이기 위해 USAN 값이 작은것을 위로 표시하면,

코너의 USAN이 가장 적고 엣지 부분 역시 값이 낮기 때문에 위 그림에서는 산등성이(ridge) 처럼 보인다.

Nucleus

마스크 중심에 있는 픽셀값

엣지나 코너의 위치 파악

USAN을 통해 대략적으로 엣지나 코너를 알 수 있는데, 엣지나 코너의 위치는 어떻게 파악할까?

정답은 thinning을 이용하는 것이다.

가장 낮은 값을 갖는 위치에 엣지가 있다고 보는 것이다.(USAN을 사용하니까 낮은 값이어야 함)

그래서 smallest USAN이라는 뜻에서 SUSAN이 된 것이다.

#### 방향에 대한 파악

Thining을 하려면 엣지의 방향을 따라가면서 체크해줘야 하는데, USAN은 방향을 구하는 부분이 없다.

이것에 대한 해답은 USAN의 무게 중심을 구하는 것이다.

무게 중심의 위치와 Nucleus의 위치를 비교하면 엣지의 방향을 파악할 수 있다.

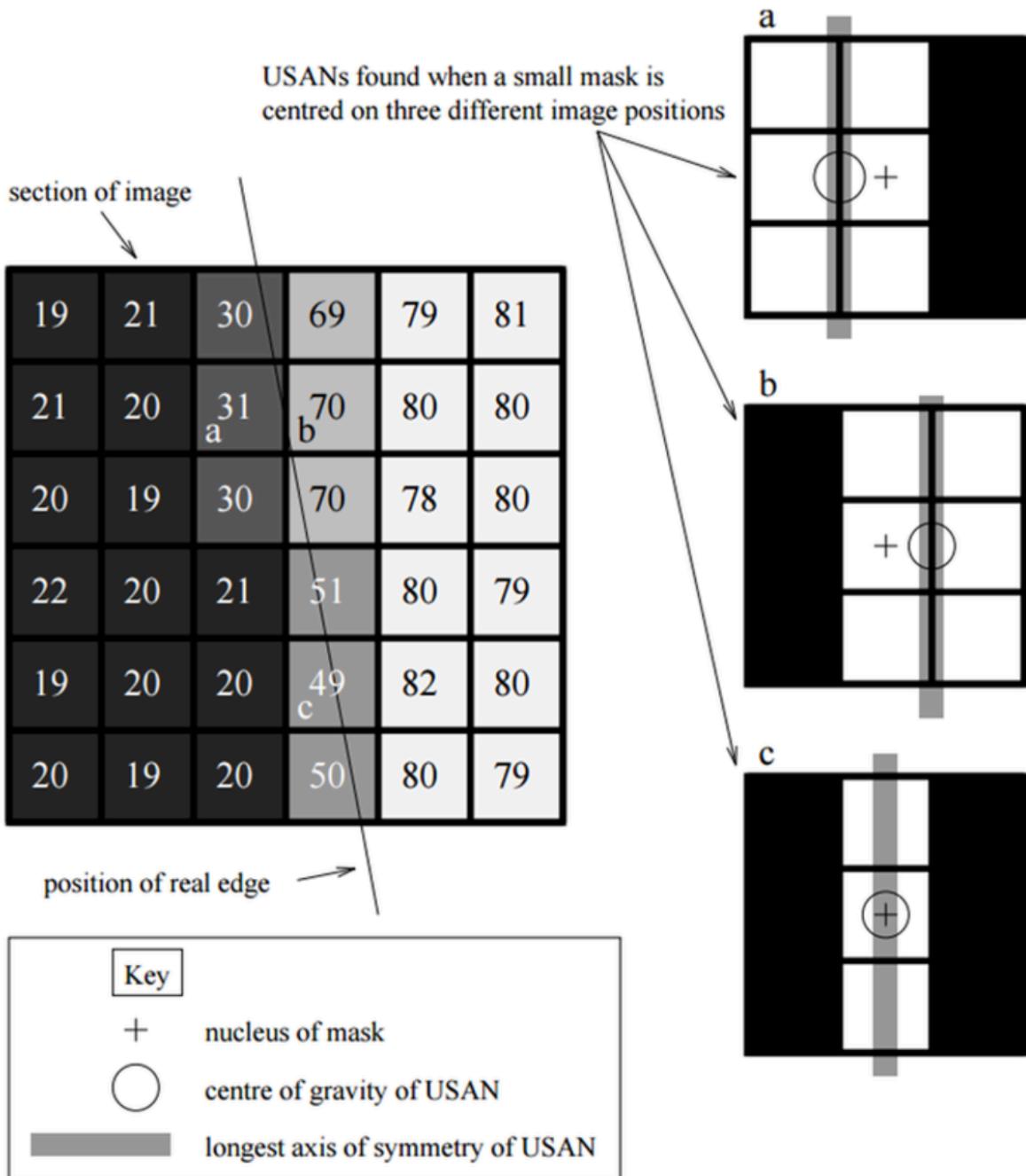
다음 그림은 가장 작은 마스크  $3 \times 3$ 를 이용한 것인데,

a는 Nucleus의 위치가 무게 중심보다 오른쪽에 있다. 이런 경우에는 엣지가 오른쪽에 있으며,

b는 그 반대의 경우이다.

c는 무게 중심의 위치가 Nucleus의 위치가 같은데, 이런 경우는 얇은 엣지가 중심에 있는 경우이다.

이렇게 엣지의 방향을 파악한 뒤에 엣지의 방향 쪽으로 스캔을 하면서 가장 낮은 USAN 값을 찾을 수 있다.



### 장점

미분 연산을 하지 않기 때문에 속도가 빠르다.

### 이미지(Image)

#### depth

이미지의 bit depth, color depth는 각 채널이 취할 수 있는 값의 범위를 결정한다.  
 8비트 이미지에서, 각 채널은  $2^8$ 의 값을 갖는다(0~255)  
 0은 가장 낮은 강도를 뜻하며 255는 가장 큰 강도를 뜻한다.

#### gray scale

grayscale 8-bit 이미지는 하나의 채널만을 갖는다.  
 0은 검은색, 255는 하얀색을 나타냄

#### 밝기

밝을 수록 큰 값을 가짐

### 커널(kernel)

픽셀들로 이루어진 값들(surrounding pixels)에 대해 값을 변환하는 mask 혹은 filter

내 생각으로는 이웃값들에 의해 중심값을 변형시키는 것 같다.

surrounding pixels은 중앙 픽셀의 이웃 픽셀을 의미한다.

커널은 central pixel에 대해 새로운 값을 결정하기 위해 neighborhood pixel 각각과 central pixel을 이용한다.

커널은 이웃에 대해 대응되는 각각의 weight로 표현된다.

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

이웃 픽셀들은 다음과 같이 표현됨

$a$	$b$	$c$
$d$	$e$	$f$
$g$	$h$	$i$

kernel을 사용한 convolution은 다음과 같이 표현된다.

$a$	$b$	$c$
$d$	$e$	$f$
$g$	$h$	$i$

\*

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

### Sharpening Algorithm

샤프닝이란 경계선을 뚜렷하게 만들어서 날카로운 느낌을 나게 해주는 알고리즘을 말한다.

원본 영상과 2차 미분한 영상을 빼주면 윤곽선이 강화되는 영상을 획득할 수 있다.



원본영상

예지영상(중앙값=5)

예지영상(중앙값=1)

### 이미지 프로세싱(Image processing, image filtering)

이미지를 변환하는 것을 의미한다.

#### 가우시안 필터(Gaussian filter)

이해

이미지가 다음과 같이 있다고 했을 때, Mean Filter를 사용한다면 다음과 같이 나타낼 수 있다.

그런데 만약 실제 값이 8.5보다 3에 가까웠다면 이 필터 때문에 노이즈가 오히려 심해져서 이미지가 왜곡되어 버리는 게 아닐까?라는 의문이 든다.

$$\begin{array}{|c|c|c|} \hline 4 & 5 & 27 \\ \hline 5 & 3 & 0 \\ \hline 5 & 5 & 23 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} = \begin{array}{|c|} \hline \end{array}$$

$$(4 \times 1/9) + (5 \times 1/9) + \dots + (23 \times 1/9) = 8.5$$

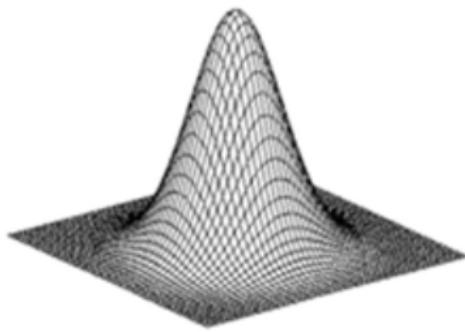
밑의 Mask에서는 가운데 화소에 가중치를 높게 주고, 가운데에서 멀어질수록 가중치를 낮게 주었다.

Mask 내의 모든 가중치를 더한 값은 1이어야 원본 이미지와 동일한 밝기를 유지할 수 있다. 그러나 Mask의 사이즈가 커질수록 내부 연산이 증가하여 속도가 느려지고, 지나치게 Mask를 크게 하면 이미지가 흐려지는 단점이 있을 수 있다.

$$\begin{array}{|c|c|c|} \hline 4 & 5 & 27 \\ \hline 5 & 3 & 0 \\ \hline 5 & 5 & 23 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1/16 & 1/8 & 1/16 \\ \hline 1/8 & 1/4 & 1/8 \\ \hline 1/16 & 1/8 & 1/16 \\ \hline \end{array} = \begin{array}{|c|} \hline \end{array}$$

$$(4 \times 1/16) + (5 \times 1/8) + \dots + (23 \times 1/16) = 6.7$$

결국 가우시안 함수를 이용하여 값을 커널에 매핑시킨 게 가우시안 필터임(중앙에 있는 값을 가장 높게 함)



Gaussian

$$h_{\sigma}(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

### 잡음(Noise)

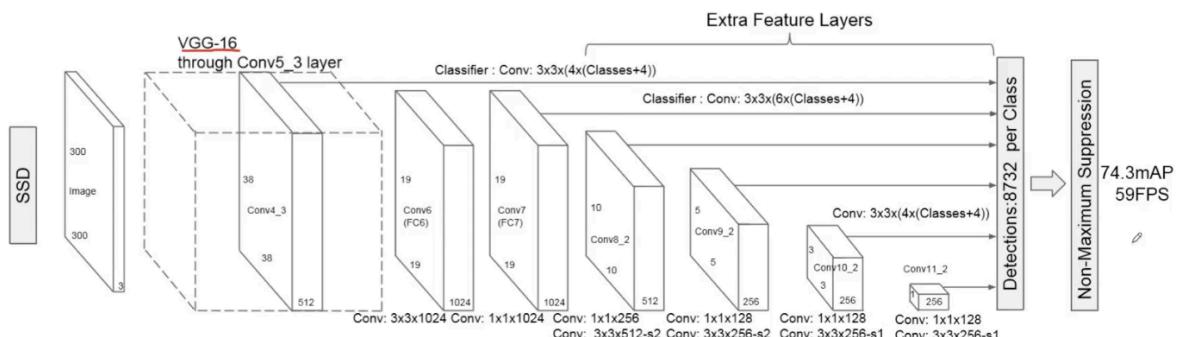
#### Impulse noise

이미지 위에 소금과 후추를 뿌린 것처럼 보인다고 해서 Salt and pepper noise라고도 한다.

#### Gaussian noise

Impulse noise가 갑자기 튀는 잡음이라면 Gaussian noise는 자연적으로 랜덤하게 발생하는 잡음이다.

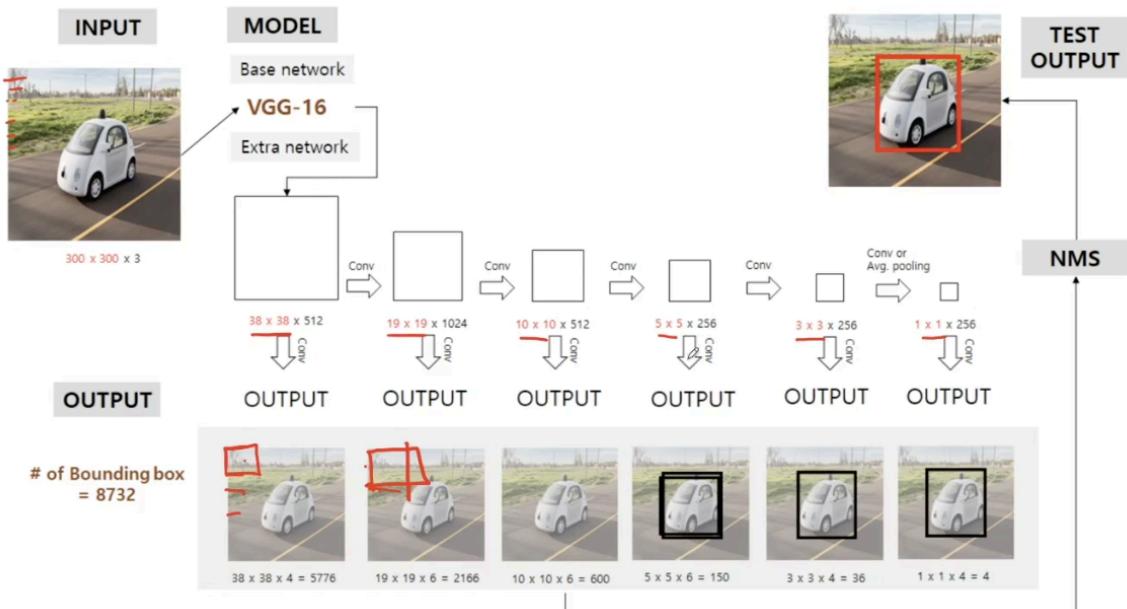
### SSD(Single Shot Detecting, SSD)



하나의 통합된 Framework로 구성됨

각각의 Feature map 들이 detection하는 output layer에 연결 됨

SSD는 중간에 나오는 Feature map들을 output에 다 전달을 해줘서 각 Feature map들을 모두 고려해 디텍션을 수행하게 됨



여러 사이즈로 Detecting 함

### (ground truth)

특정 질문에 관한 진리에 대한 지식과 관련된 개념적 용어이다.  
이는 이상적인 예상 결과이다.

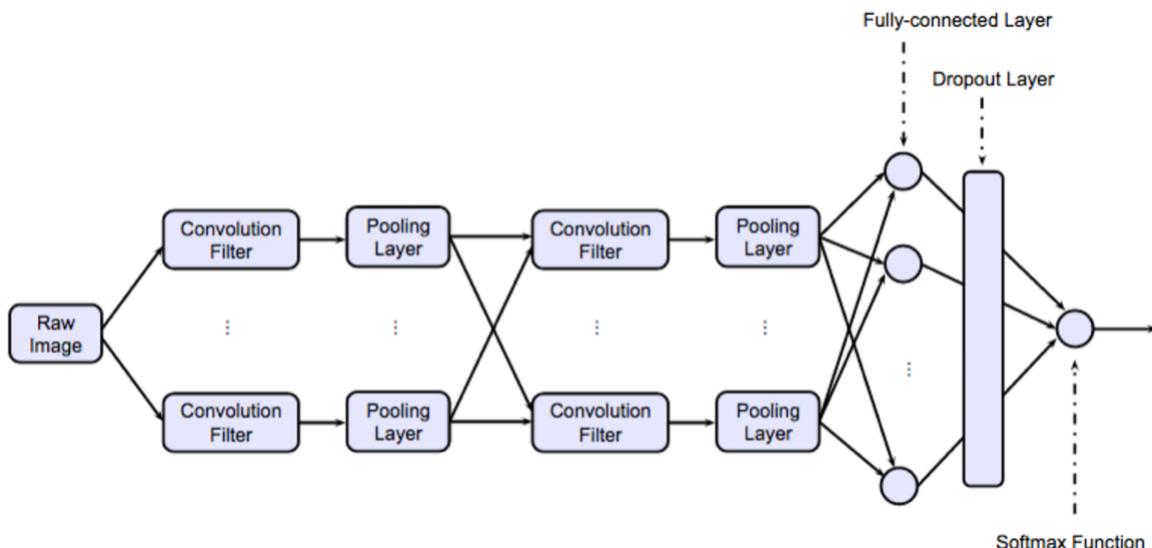
### (ground truthing)

테스트에 대한 적절한(객관적인) 데이터를 수집하는 프로세스를 나타낸다.

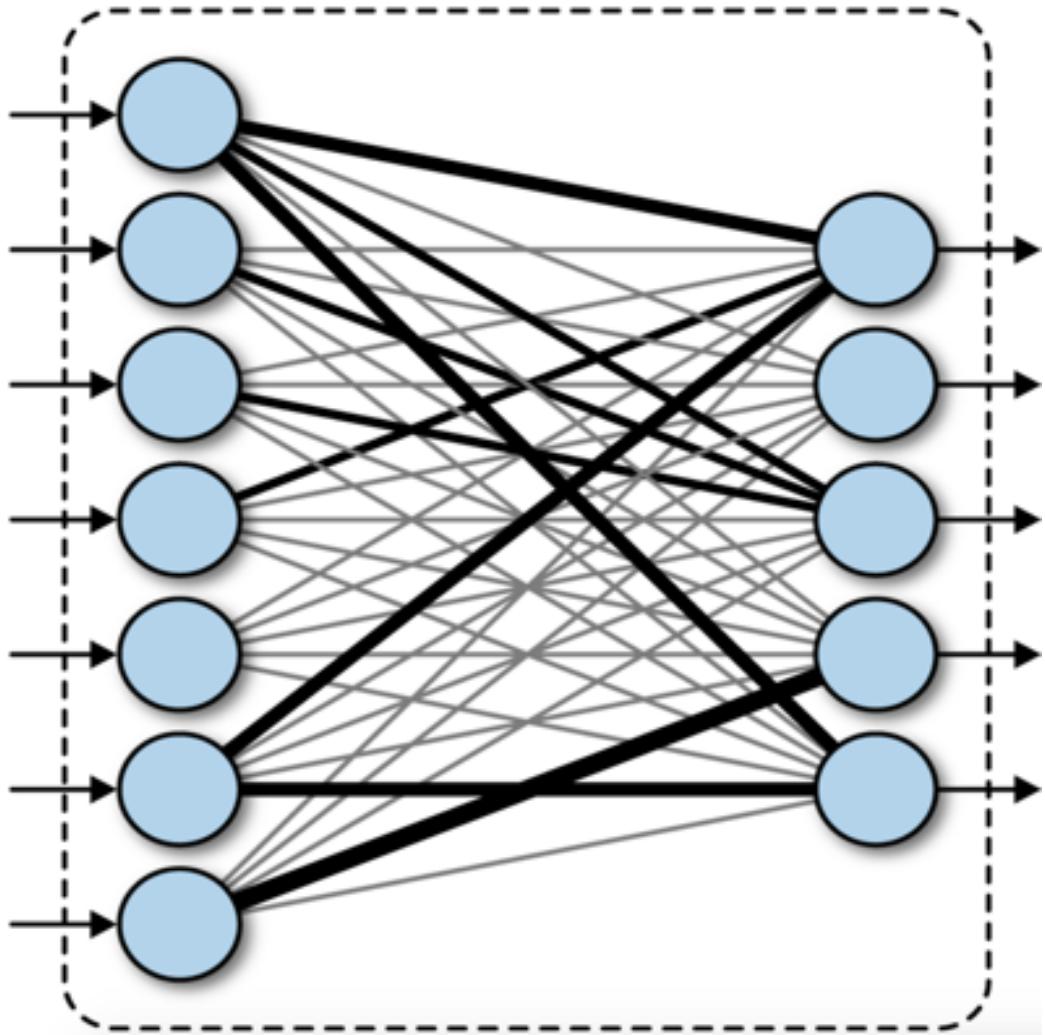
### (Fully Connected Layer)

Convolutional layer에서 특징이 추출되었으면 이 추출된 특징 값을 기존의 뉴럴 네트워크(ANN)에 넣어서 분류를 한다.

그래서 CNN의 최종 네트워크 모양은 다음과 같이 된다.



fully connected layer는  $R^m$  차원을  $R^n$  차원으로 매핑하는 function이다.



$x \in \mathbb{R}^m$  이라고 하고 이를 fully connected layer의 input이라고 해보자.

$y_i \in \mathbb{R}$  를 fully connected layer의  $i$ 번째 output이라고 해보자.

그러면  $y_i$ 는 다음과 같이 표현된다.

$$y_i = \sigma(w_1 x_1 + \dots + w_m x_m)$$

출력 전체를 표현하면 다음과 같이 된다.

$$y = \begin{pmatrix} \sigma(w_{1,1} x_1 + \dots + w_{1,m} x_m) \\ \vdots \\ \sigma(w_{n,1} x_1 + \dots + w_{n,m} x_m) \end{pmatrix}$$

$$y = \sigma(wx)$$

이때  $w$ 는  $n \times m$  차원이 됨

convolutional으로 표현

<https://www.quora.com/Is-a-fully-connected-neural-network-conceptually-similar-to-a-1x1-convolutional-neural-network>

fully connected layer는 필터가 input size의 크기와 같은 convolutional layer로 볼 수 있다.

이때 convLayer의 모든 유닛은 각 input element에 연결된다.

1x1 convolutional 과의 관계

만약, ConvNet의 마지막 Layer인 MLP를 바꾸기 원한다고 하자.

MLP의 input이  $N \times 8 \times 8$  feature map이라고 했을 때 MLP가 3개의 layer를 갖는다  
고 해보자.(A, B, C)

이 각각의 Layer들은 다음과 같이 표현할 수 있다.

$(N \times 8 \times 8) \rightarrow [\text{ConvLayer } 8 \times 8, A \text{ output maps}] \rightarrow$

$(A \times 1 \times 1) \rightarrow [\text{ConvLayer } 1 \times 1, B \text{ output maps}] \rightarrow$

$(B \times 1 \times 1) \rightarrow [\text{ConvLayer } 1 \times 1, C \text{ output maps}] \rightarrow$

$(C \times 1 \times 1)$

먼저 input size와 같은 필터를 적용하여 fully connected layer를 convolutional로  
표현한다.

그 이후는  $1 \times 1$  output이 계속 뜨니까  $1 \times 1$  convolution으로 표현해주면 된다.

### (fully connected network)

fully connected layer의 연쇄로 이루어진 네트워크이다.

### (training set)

학습하는데 사용하는 샘플

### (training instance)

각 훈련 데이터를 의미함

### 파이프라인(pipeline)

데이터 처리 컴포넌트(component)들이 연속되어 있는 것을 파이프라인이라고 한다.

보통 컴포넌트들은 비동기적으로 동작한다.

각 컴포넌트는 많은 데이터를 추출해 처리하고 그 결과를 다른 데이터 저장소로 보낸다.

즉, 각 컴포넌트는 완전히 독립적이다.

컴포넌트 사이의 인터페이스는 데이터 저장소 뿐이다.

### 편향(bias)

#### 표집 편향, 샘플링 편향(Sampling bias)

표본 추출이 잘못되어 대표성을 띠지 못하는 데이터

ex) 대통령 선거 여론조사

#### 데이터 스누핑 편향(data snooping bias)

데이터를 본 후에 기계학습 알고리즘을 결정하는 것으로, 사실 데이터를 보기 전에 기계학습 알고리즘을 선정해야 하지만

현실적으로 협업에서 작업하는 사람들이 흔히 범하는 실수다.

동일한 데이터에 대해 갖가지 기계학습 알고리즘을 적용해서 가장 좋은 성능이 나오는 알고리즘을 선정하는 것

문제는 데이터가 바뀌면 어떨까? 아마 기대했던 성능이 나오지 않을 가능성이 크다.

### 표집(Sampling)

#### 계층적 샘플링(stratified sampling)

전체 모수를 계층이라는 동질의 그룹으로 나누고, 테스트 세트가 전체 모수를 잘 대표하도록  
각 계층에서 올바른 수의 샘플 추출

#### 무작위 샘플링(random sampling)

random하게 샘플링을 하는 방법

데이터셋이 충분히 크다면 일반적으로 괜찮지만, 그렇지 않다면 샘플링 편향이 생길 가능성

이 크다.

## 싸이킷런(Scikit-learn)

### 일관성

모든 객체가 일관되고 단순한 인터페이스를 공유한다.

#### 추정기(estimator)

데이터셋을 기반으로 일련의 모델 파라미터들을 추정하는 객체를 추정기라고 한다.

추정 자체는 fit() 메서드에 의해 수행되고 하나의 매개변수로 하나의 데이터셋만 전달 한다.

(지도 학습 알고리즘에서는 매개변수가 두개로, 두 번째 데이터셋은 레이블을 담고 있다.)

추정 과정에서 필요한 다른 매개변수들은 모두 하이퍼파라미터로 간주되고 인스턴스 변수로 저장된다.

#### 변환기(transformer)

데이터셋을 변환하는 추정기를 변환기라고 한다.

변환은 데이터셋을 매개변수로 전달받은 transform() 메서드가 수행한다.

그리고 변환된 데이터셋을 반환한다.

모든 변환기는 fit()과 transform()을 연달아 호출하는 것과 동일한 fit\_transform() 메서드도 가지고 있다.

#### 예측기(predictor)

일부 추정기는 주어진 데이터셋에 대해 예측을 할수 있다.

예측기의 predict() 메서드는 새로운 데이터셋을 받아 이에 상응하는 예측값을 반환 한다.

또한 테스트 세트를 사용해 예측의 품질을 측정하는 score() 메서드를 가진다.

### 분할

#### train\_test\_split()

ex)

```
from sklearn.model_selection import train_test_split  
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

#### StratifiedShuffleSplit()

계층적 샘플링(stratified sampling)

ex)

```
from sklearn.model_selection import StratifiedShuffleSplit  
split = StratifiedShuffleSplit(n_splits=1, test_size = 0.2, random_state = 42)  
for train_index, test_index in split.split(housing, housing["income_cat"]):  
    strat_train_set = housing.loc[train_index]  
    strat_test_set = housing.loc[test_index]
```

### 원-핫인코딩

#### OneHotEncoder()

원핫인코딩을 수행한다.

출력은 희소행렬이다.

```
from sklearn.preprocessing import OneHotEncoder  
encoder = OneHotEncoder()  
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
```

### Pipeline

연속된 변환을 순서대로 처리해줄수 있도록 도와주는 클래스

파이프라인의 fit 메소드를 호출하면 모든 변환기의 fit\_transform() 메서드를 순서대로 호

출하면서 한 단계의 출력을 다음 단계의 입력으로 전달한다.  
마지막 단계에서는 fit() 메서드만 호출한다.

## 회귀

### LinearRegression()

선형 회귀 모델

```
from sklearn.linear_model import LinearRegression  
lin_reg = LinearRegression()  
lin_reg.fit(X, y)  
    intercept_  
    편향  
    coef_  
    가중치
```

### DecisionTreeClassifier()

ex)

```
iris = load_iris()  
X = iris.data[:, 2:]  
y = iris.target
```

```
tree_clf = DecisionTreeClassifier(max_depth = 2)  
tree_clf.fit(X, y)
```

### VotingClassifier()

ex)

```
log_clf = LogisticRegression(solver = 'liblinear')  
rnd_clf = RandomForestClassifier(n_estimators=10)  
svm_clf = SVC(gamma='auto')
```

```
voting_clf = VotingClassifier(estimators=[('lr',log_clf),  
                                         ('rf',rnd_clf),  
                                         ('svc',svm_clf)],  
                                         voting='hard')
```

```
voting_clf.fit(X_train, y_train)
```

### BaggingClassifier()

n\_estimator: 양상들에 사용할 분류기의 수

max\_sample: 무작위로 뽑을 샘플의 수(0~1 사이의 수로 지정하면 비율이 되어, 훈련 세트에 곱한 것만큼 샘플링)

bootstrapping: True(중복허용, 배깅), False(중복허용 X, 페이스팅)

n\_jobs: 사용할 CPU 수 (-1로 지정하면 모든 코어 사용)

## 분류

### SGDClassifier()

확률적 경사하강법을 사용하는 Linear Classifier

feature가 floating value인 dense or sparse array data에 대해 작동한다.

기본적으로 linear support vector machine에 적합된다.

ex)

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(max_iter = 5, random_state=42)
sgd_clf.fit(x_train, y_train_5)
LinearSVC()
결정경계를 선형으로 하여 분류하는 SVM 분류기
```

검증

```
cross_val_score()
```

K-Fold 교차검증

ex)

```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, x_train, y_train, cv=3, scoring = 'accuracy')
```

### numpy library

과학 계산을 위한 라이브러리로 다차원배열을 처리하는데 필요한 기능을 제공한다.

#### NaN

NaN 값

ex) np.NaN

#### is.nan()

인수로 오는 값이 nan인지 확인해줌

ex) np.isnan( np.nan )

#### 타입관련

##### 데이터 타입 생성

```
np.dtype(obj)
```

obj: object to be converted to a data type

ex)

```
f = np.dtype([('f1', np.int16), ('f2', np.uint64)])
```

```
f['f1']
```

```
f['f2']
```

##### 요소 타입 관련

#### astype()

요소의 타입을 변경한다.

```
Z = np.arange(10, dtype=np.float32)
```

```
Z = Z.astype(np.int32, copy=False)
```

```
print(Z)
```

#### ignore numpy warning

```
# Suicide mode one
```

```
defaults = np.seterr(all="ignore")
```

```
# Back to sanity
```

```
_ = np.seterr(**defaults)
```

#### 로딩

#### loadtxt()

데이터를 불러올수 있음

```
ex) death = np.loadtxt(fname = 'c:/data/death.csv', delimiter = ',', dtype =
np.int)
```

#### fname

파일 경로

#### dtype

요소의 타입

#### delimiter

읽어들일때 구분자

### genfromtxt()

스트링을 numpy 배열로 변환한다.

ex)

```
s = StringIO("1, 2, 3, 4, 5\n")
```

```
    6, , , 7, 8\n
```

```
    , , 9,10,11\n")
```

```
Z = np.genfromtxt(s, delimiter=",", dtype=np.int)
```

### 순회

#### ndenumerate()

다차원 index iterator

배열 좌표와 값 쌍을 생성하는 반복자를 반환한다.

```
np.ndenumerate(arr)
```

### Check

#### np.any(a, axis=None, out=None, keepdims=<no value>)

주어진 축을 따라 배열 요소가 True로 평가되는지 테스트 한다.

a: array\_like, input array

axis: logical OR reduction이 수행되는 축

### 집합연산

#### union1d()

두 데이터 집합을 합침

```
ex) np.union1d(x, y)
```

#### intersect1d()

두 데이터 집합의 교집합

```
ex) np.intersect1d(np.arange(4).reshape(2,2),
```

```
np.array([0,1,2,6]).reshape(2,2) )
```

#### setdiff1d()

두 데이터 집합의 차집합

```
ex) np.setdiff1d(x, y)
```

### 메모리관련

#### np.iinfo()

integer type에 대한 기계의 limit 값

```
ex) np.iinfo(np.int16)
```

#### np.finfo()

부동 소수점 유형의 machine의 limit 값

```
ex) np.finfo(np.float32)
```

### 통계관련

#### percentile()

```
np.percentile( dataset, percentage )
```

dataset을 가지고 percentage가 되는 값을 추출

```
ex) np.percentile([1,2,3,4,5,6], [0, 25, 50, 75, 100])
```

#### median()

중앙값

#### cov()

공분산

ex)

```
x = [184, 170, 180]
```

```
y = [85, 70, 82]
```

```
np.cov(x,y)[0][1]
```

`mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)`

지정된 축을 따라 산술 평균을 계산한다.

배열요소의 평균을 반환한다. 평균은 기본적으로 평평한 배열에, 그렇지 않으면 지정된 축에 적용된다.

keepdims: 차원을 유지할것인지 여부 True이면 축소된 축은 결과적으로 사이즈가 1인 dimension으로 남고 False이면 1차원 배열 반환

ex)

`x = [184, 170, 180]`

`np.mean(x)`

`corrcoef()`

`x = [184, 170, 180]`

`y = [85, 70, 82]`

ex) `np.corrcoef(x,y)[0][1]`

날짜

`np.datetime64()`

`np.datetime64(date)`

return: return date / yyyy-mm-dd format

오늘: `np.datetime('today', 'D')`

시간연산도 가능

ex) `np.datetime('today', 'D') - np.datetime(1, 'D')`

기간

ex)

`Z = np.arange('2016-07', '2016-08', dtype='datetime64[D]')`

`print(Z)`

데이터 샘플링

`np.random.choice(a, size=None, replace=True, p=None)`

a: 1차원 데이터 배열 혹은 int, 만약 int로 주어지면 `np.arange(a)`를 통해 배열을 생성한다.

size: 정수, 샘플 숫자

replace: 불리언, True이면 한번 선택한 데이터를 다시 선택 가능

p: 각 항목과 관련된 확률, 1차원 배열, 만약 각 항목과 관련된 확률이 제시되지 않은 경우 균일분포를 가정한다.

array 타입

행렬 표현

`array()`

배열을 만들

ex) `z2 = np.array([[1, 2, 3], [4, 5, 6]])`

indexing and slicing

[행]

[행, 열]

ex)

`z3[0] <- 0번째 행`

`z3[:, 1] <- 1번째 열의 모든 행 뽑아내기`

`z3[0:2, 0] <- 0~1번째 행, 0번째 컬럼`

`np.unravel_index()`

flat index를 coordinate array 좌표 index로 변환

`print(np.unravel_index(99, (6,7,8)))` - 6,7,8 shape array에 대해 100

번째 element를 뽑음

`indices`

평탄화 된 버전의 index

`shape`

coordinate 형태

`order`

인덱싱의 row-major 혹은 column-major 선택

### 3차원 배열

3차원 배열은 예를 들어 다음과 같이 만들수 있다.

`x = np.zeros((2,3,4))`

그런데 이는 변수로 표현했을때 x,y,z 형식이 아니다.

넘파이 배열은 list와 같은 형식으로 만들어지기 때문이다.

따라서 이는 2개의 리스트를 만들고 그 각각의 요소는 그 안에 3개의 리스트를 가지고 그 각각의 요소는 4개의 요소를 가지는 식이다.

그러므로 z,y,x 형식이 되는 것이다.

따라서 이에 접근할때는 z,y,x 로 접근 해야 한다.

### 요소 관련

`dtype`

행렬 요소의 type을 알수 있음

ex) `a.dtype`

`size`

배열 요소의 개수

`np.argmin()`

최솟값이 들어있는 색인 반환

`np.argmin(x, axis = [0: 세로축, 1: 가로축])`

ex) `np.argmin(x)`

`np.argmax()`

최대값이 들어있는 색인 반환

`np.argmax(a, axis=None, out=None)`

`a: input array`

`axis: 기본적으로 인덱스는 평평한 배열에 있고 그렇지 않으면 지정된 축`

을 따라 배열된다.

즉, 0이면 세로축에서 1이면 가로축에서 최대값을 찾음

ex) `np.argmax(x)`

`unique()`

중복을 제거한 값을 array로 나타냄

ex) `np.unique(['a', 'a', 'b', 'c', 'b', 'c'])`

`return_counts`

`True: 중복된 값의 개수를 반환`

`False: 중복된 값의 개수를 반환하지 않음`

ex) `index, cn = np.unique(lst, return_counts = False)`

`axis`

이 옵션을 설정안할시, 전체 데이터를 대상으로 중복 제거

`0: 세로축으로 중복 제거`

`1: 가로축으로 중복 제거`

### 용량 관련

`itemsize`

원소 하나가 차지하는 바이트

ex) m.itemsize  
**nbytes**  
배열 전체가 차지하는 바이트  
ex) m nbytes  
요소 뽑아내기  
**boolean** 값으로 요소 뽑아내기  
ex)  
z3[[[False, True, False], [True, False, True], [False, True,  
False]]]  
z3[z3%2 == 0]

**차원 관련**

**shape**  
행렬의 차원을 알 수 있음  
ex) a.shape  
**ndim**  
차원의 개수  
ex) m.ndim  
**차원변경**  
**reshape()**  
행렬의 차원을 바꿈  
행 또는 열의 개수에 -1 값을 넣으면 무한을 의미  
ex)  
z.reshape((4,5))  
np.arange(9).reshape(-1,1)  
**order**  
우선순위 설정  
즉, 어떤 축을 기준으로 먼저 기준 데이터를 순회하면서 행렬  
에 어떤 기준으로 데이터를 세팅할까를 설정하는 것  
C: 행우선(기본값) - 기준 데이터에 대해 행을 먼저 순회하면서,  
서, 출력데이터에 대해서도 행을 먼저 채움  
F: 열우선 - 기준 데이터에 대해 열을 먼저 순회하면서, 출력  
데이터에 대해서도 열을 먼저 채움  
ex) x.reshape((5,2), order = 'F')  
**flatten()**  
행렬의 차원을 축소함 -> 1차원  
C: 행을 우선하여 순회  
F: 열을 우선하여 순회  
ex) x.flatten('F')  
**np.atleast\_2d()**  
input 들을 2차원 이상의 배열로 본다.  
ex)  
x = boston['RM'] <- series 형식  
x\_1 = np.atleast\_2d(x)  
결과: 1행, n열

**반복**

**repeat()**  
요소의 값을 반복.. 반환값은 벡터(행벡터)  
ex)  
x = np.arange(3)

`x.repeat([2, 3, 4])`  
`axis`  
축 설정  
axis를 설정하지 않으면 행벡터가 나옴  
0: 세로축으로 데이터가 붙음  
1: 가로축으로 데이터가 붙음

`tile()`  
요소의 값을 반복  
ex)  
`np.tile([[0,1],[1,0]],(4,4))`  
`A`  
input array  
`reps`  
각 축에 대해 A의 반복 횟수

### 정렬

`argsort()`  
정렬했을때 index를 반환함(오름차순)  
ex)  
`x = np.array([50, 30, 40, 10, 20])`  
`x.argsort()`  
`x[x.argsort()][::-1]`  
`np.flip()`  
주어진 축에서 배열에 있는 원소의 순서를 반전시킨다.  
ex)  
`a = np.arange(1, 10)`  
`np.flip(a, 0)`  
`np.sort()`  
`np.sort(a, axis=1, kind=None, order=None)`  
a: array  
axis: 정렬할 축이다. None이면 정렬하기 전에 배열이 평평해진다. 기본값은 -1이며 마지막 축을 따라 정렬된다.

kind: 정렬알고리즘, 기본값은 quicksort이고 stable과 mergesort가 있다.

order: a가 정의 된 필드가 있는 배열인 경우 이 인수는 첫번째, 두번째 등을 비교할 필드를 지정한다.

### 변경

`np.pad()`  
`np.pad(array, pad_width, mode, **kwargs)`  
ex) `np.pad(np.full((3,3), 0), ((1,1),(1,1)), 'constant', constant_values=(1))`

`array`  
패딩할 array

`pad_width`  
각 축의 경계에 대해 패딩될 value 개수

ex) (`left, right`): 왼쪽편에 2개 오른쪽 편에 3개  
ex) (`(top, bottom), (left, right)`): 위, 아래, 왼쪽, 오른쪽

`mode`  
`constant`

상수로 패딩한다.

#### np.put()

np.put(a, ind, v, mode='raise')

배열의 지정된 요소를 주어진 값으로 바꾼다.

indexing은 평탄화 된 target array에서 작동한다.

put은 다음과 같이 작동한다.

a.flat[ind] = v

#### Immutable

array의 flags의 writeable 속성을 False로 바꾼다.

### 행렬 생성

#### arange()

범위 값을 array 형식으로 나타냄

np.arange(시작값, 종료값, 증분) : 시작값 <= 값 < 종료값

ex)

np.arange(4)

np.arange(1,4)

#### dtype

'f': float 형식

ex) np.arange(5, dtype = 'f')

#### zeros()

영행렬을 만듦

np.zeros(shape, dtype=float, order='C')

shape: int or tuple of ints

dtype: data-type, optional, Default is numpy.float64,

order: {'C', 'F'}, optional, default: 'C'

ex) np.zeros((4, 4))

#### full()

특정 데이터로 채워진 행렬을 만듦

ex) np.full((4, 4), 2)

#### eye()

항등행렬을 만듦

ex) np.eye(3) <- 3x3 항등 행렬

#### np.random.rand()

random value로 채워진 행렬을 만든다.

ex) np.random.rand(3,3,3)

#### np.random.uniform()

np.random.uniform(low=0.0, high=1.0, size=None)

균일한 분포에서 표본을 추출한다.

샘플은 반 개방 간격(낮은, 높음)에 걸쳐 균일하게 분포된다. - 낮음 포함, 높음

### 제외

size: output shape, None 이면 single value

ex) np.random.uniform(1,2,(2,2))

#### np.random.random()

반 개방 간격 [0.0, 1.0)에서 임의의 부동 소수점을 반환한다.

결과는 명시된 구간에 대한 “continuous uniform”한 분포에서 비롯된다.

np.random.rand(size = None)

#### np.random.randint(최소, 최대)

입력 파라미터인 최소부터 최대까지 중 임의의 정수를 리턴한다.

### `np.diag()`

`np.diag(v, k=0)`

v: v가 2차원이면 v의 k번째 diagonal의 원소들을 copy한다. v가 2차원이면 k 번째 diagnoal에 v의 원소들을 채워 리턴한다.

k: k>0 를 하면 main diagonal 위에, k<0를 하면 main diagnoal 아래  
ex)

`x = np.arange(9).reshape((3,3))`

`np.diag(x, k=1)`

### `np.diag(np.diag(x))`

### `np.fromiter()`

`np.fromiter(utterable, type, count = -1)`

iterable object로부터 1차원 배열을 생성한다.

iterable: array에 대해 데이터를 제공하는 iterable 객체

dtype: return 될 array의 datatype

count: 아이템의 개수, -1이면 모든 데이터를 읽는것을 뜻한다.

### `np.linspace()`

지정된 간격 동안 균일한 간격의 숫자를 반환한다.

`np.linspace(start, stop, num = 50, endpoint, = True, retstep = False,  
type = None, axis = 0)`

start: starting value

stop: end value

num: 생성할 샘플의 개수

endpoint: 만약 True면, stop은 마지막 표본이다.

retstep: 만약 True면, step도 반환함

dtype: array 타입

axis: 샘플을 저장할 결과의 축

### `np.meshgrid()`

`np.meshgrid(*x1, **kwargs)`

좌표 벡터에서 좌표 행렬을 반환한다. 1차원 좌표 배열이 주어지면 N-D 그리드  
에서 N-D 스칼라/벡터 필드의 벡터화 된 평가를 위해

N-D 좌표 배열을 만든다.

x1, x2, …, xn: array\_like, 그리드의 좌표를 표현할 1차원 배열

### 비교

### `np.allclose()`

두 배열이 tolerance 내에서 요소별로 동일한 경우 True를 반환한다.

`np.allclose(a, b, rtol = 1e-05, atol = 1e-08, equal_nan = False)`

a, b: array\_like

rtol: relative tolerance 값

atol: absolute tolerance 값

equal\_nan: NaN을 비교할지 여부. True인 경우 a에 있는 NaN은 출력배열에  
서 b에 있는 NaN과 같은것으로 간주된다.

다음 방정식이 True라면 allclose는 True를 반환한다.

`absolute(a - b) <= (atol + rtol * absolute(b))`

결국 두 값의 차가 어떤 범위 안에 존재한다면 True를 반환하는 것임

### `np.array_equal()`

두 배열이 같은 shape이고, 같은 elements라면 True를 반환

```
np.array_equal(a1, a2)
```

### 병합

```
np.append()
```

행렬을 합친다.

```
numpy.append(arr, values, axis=None)
```

arr: array\_like

values: arr에 카피하여 append 시킬 값

axis 가 0 이면 flatten되고, axis가 지정되면 차원이 유지됨

```
concatenate()
```

행렬을 합침

0: 세로축을 기준으로 합침

1: 가로축을 기준으로 합침

ex) np.concatenate([x, y], axis = 0)

```
np.vstack()
```

행렬을 vertical 방향으로 합친다(stack)라는 의미

ex) np.vstack((x, y))

```
np.hstack()
```

행렬을 horizontal 방향으로 합친다(stack)라는 의미

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

```
np.r_
```

두 배열을 왼쪽에서 오른쪽으로 붙인다.(hstack과 같은 기능)

```
np.r_[a, b]
```

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

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

두 배열을 위에서 아래로 붙인다.

```
np.r_[[a], [b]]
```

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

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

```
np.c_
```

두개의 1차원 배열을 컬럼으로 세로로 붙여서 2차원 배열 만들기

```
np.c_[a,b]
```

```

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

```

```

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

```

```

np.c_[[[1,2,3], [1,2,3]],[[4,5,6], [4,5,6]]]

```

```

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

```

### 연산

`+, -, *, /, //`

행렬끼리 연산 가능(index가 같은 것에 대하여.)

<=> add(), subtract(), multiply(), divide()

ex) `x+y`

`np.add(x,y)`

`np.subtract(x,y)`

`np.multiply(x,y)`

`np.divide(x,y)`

`out`

출력 변수 지정

ex) `np.add(A,B,out=B)`

`np.subtract.outer()`

`np.subtract.outer(alb)`

두 벡터에 대해 subtract 연산

`z_ij = a_i - b_j` 인데 j부터 순회

outer하게 연산함

ex)

`a = (10,4)`

`b = (2,3)`

`print(np.subtract.outer(a,b))`

`np.floor()`

floor value 반환

`np.negative()`

부호를 붙임

`<<, >>`

`n << [array]`: n \* 2의 array 요소승

`n >> [array]`: n / 2 의 array 요소 승

ex)

`Z = np.arange(5)`

`2<<Z`

`np.prod(x)`

곱

`np.prod(x, axis = 0: 세로축, 1: 가로축)` <- axis가 생략되면 전체 데이터에

## 대해 곱을 구함

`np.around()`  
`np.around(a, decimals=0, out=None)`

a: input data

decimals: 반올림할 소수 자릿수, 만약에 negative value가 온다면, 소수점 왼쪽의 위치가 지정된다.

ex)

```
x = np.random.rand(3,3)
print(x, '\n')
print(np.around(x, 0))
np.clip(a, a_min, a_max)
```

array에 있는 데이터를 a\_min 보다 작은 값을 자르고 a\_max보다 큰 값을 자른다. (자르면, a\_min보다 작은 값은 a\_min으로 a\_max보다 큰 값은 a\_max로 값이 옮겨짐)

## 통계

`np.sum()`

ex) `np.sum(dataset, axis = [0: 세로축, 1: 가로축])`

`np.mean()`

ex) `np.mean(x, axis = [0: 세로축, 1: 가로축])`

`np.var()`

분산

ex) `np.var(x, axis = [0: 세로축, 1: 가로축])`

`np.std()`

표준편차

ex) `np.var(x, axis = [0: 세로축, 1: 가로축])`

`np.max()`

최대값

ex) `np.max(x, axis = [0: 세로축, 1: 가로축])`

`np.min()`

최솟값

ex) `np.min(x, axis = [0: 세로축, 1: 가로축])`

`np.maximum()`

두 데이터 집합의 index를 기준으로.. 최댓값을 출력

ex)

```
x = np.arange(0, 20, 2)
```

```
y = np.arange(0, 30, 3)
```

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

`np.minimum()`

두 데이터 집합의 index를 기준으로.. 최솟값을 출력

ex)

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

## 누적

`np.cumsum()`

누적합

`np.cumsum(x, axis = 0: 세로축, 1: 가로축) <- axis가 생략되면 전체 데이터에 대해 누적합을 구함(열부터 더해짐)`

`np.cumprod()`

누적곱

`np.cumprod(x, axis = 0: 세로축, 1: 가로축) <- axis가 생략되면 전체 데이터에 대해 누적곱을 구함(열부터 더해짐)`

터에 대해 누적합을 구함

### 필터

#### np.take()

index sequence를 기준으로 data를 가져옴

축을 따라 array에서 요소를 가져옴

ex) np.take( data array , index sequence, axis = 0: 세로, 1:가로)

#### np.where()

np.where( condition, [True일때 수행되는 expression], [False일때 수행되는 expression] )

condition: array\_like, bool

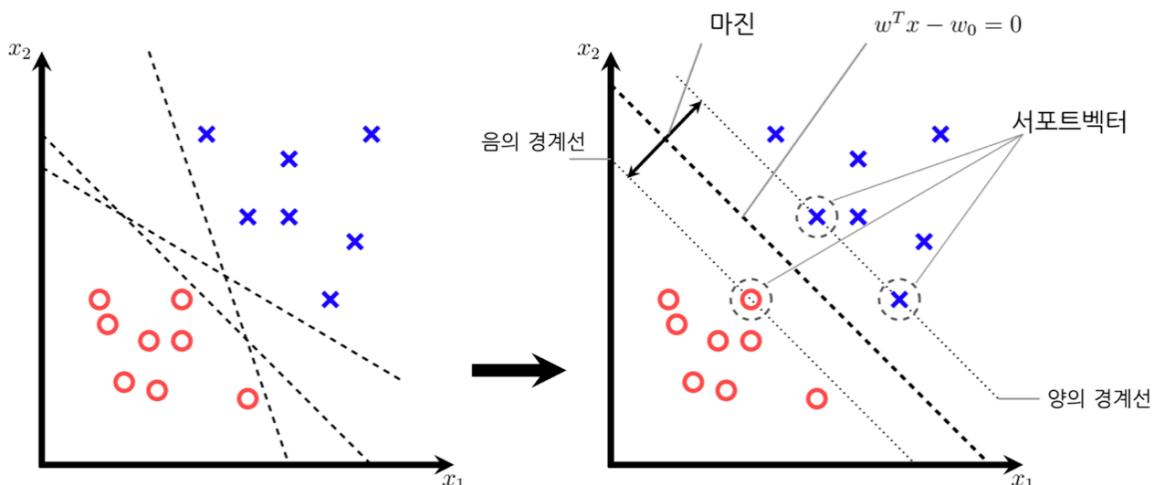
만약 condition만 들어가 있으면, index가 반환됨

ex) teens['age'] = np.where(teens['age'].isnull(), teens['agemean'], teens['age'])

## SVM(Support Vector Machine)

퍼셉트론은 가장 단순하고 빠른 판별 함수 기반 분류 모형이지만 판별 경계선(decision hyperplane)이 유니크하게 존재하지 않는다는 단점이 있다.

SVM은 퍼셉트론 기반의 모형에 가장 안정적인 판별 경계선을 찾기 위한 제한 조건을 추가한 모형이라고 볼 수 있다.



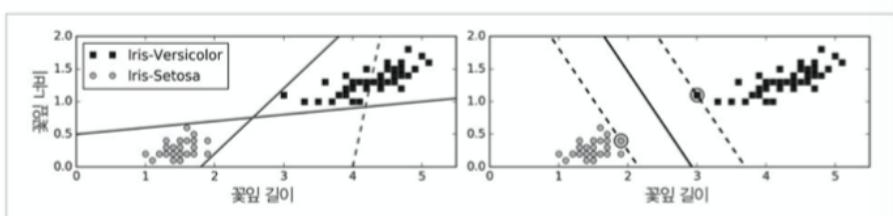
결정 경계가 샘플에 너무 가까우면 새로운 샘플에 대해서 아마 잘 작동하지 못할 것이다.

(Overfitting)

SVM 분류기를 클래스 사이에 가장 폭이 넓은 도로를 찾는 것으로 생각할 수도 있다.

그래서 Large margin classification 이라고도 한다.

그림 5-1 리지 마진 분류



### 서보트 벡터(support vector)

판별함수 모형에서  $y$ 는  $+1, -1$  두개의 값을 가진다.

판별함수 모형에서 직선인 판별 함수  $f(x)$ 는 다음과 같이 나타낼수 있다.

$$f(x) = w^T x - w_0$$

x 데이터 중에서 y값이 +1인 데이터를 x+, y값이 -1인 데이터를 x-라 하자.

판별함수의 정의에 따라 y값이 +1인 그 데이터  $x_+$ 에 대한 판별함수의 값은 양수가 된다.

반대로 y값이 -1인 데이터  $x_-$ 에 대한 판별함수의 값은 음수가 된다.

$$f(x_+) = w^T x_+ - w_0 > 0$$

$$f(x_-) = w^T x_- - w_0 < 0$$

y 값이 +1인 데이터 중에서 판별함수의 값이 가장 작은 데이터와 y값이 -1인 데이터 중에서 판별함수의 값이 가장 큰 데이터가 존재할 것이다.

이를 x+, x-라 하자.

이 데이터들은 각각의 클래스에 속한 데이터 중에서 가장 경계선에 가까이 붙어있는 최전방 (most front)의 데이터들이다.

이러한 데이터를 서포트(support) 혹은 서포트 벡터(support vector)라고 한다.

물론 이 서포트에 대해서도 부호 조건은 만족되어야 한다.

$$f(x^+) = w^T x^+ - w_0 > 0$$

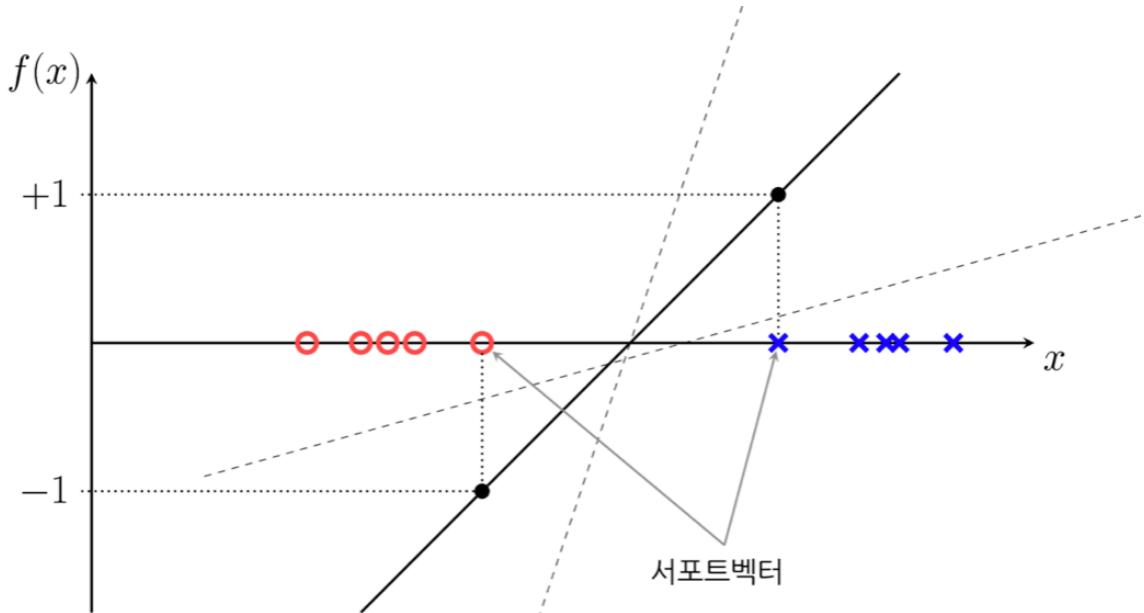
$$f(x^-) = w^T x^- - w_0 < 0$$

서포트에 대한 판별함수 값은 부호 조건만 지키면 어떤 값이 되어도 괜찮다.

따라서 다음과 같은 조건을 만족하도록 판별 함수를 구한다.

$$f(x^+) = w^T x^+ - w_0 = +1$$

$$f(x^-) = w^T x^- - w_0 = -1$$



이렇게 되면 모든  $x_+$ ,  $x_-$  데이터에 대해 판별함수의 값이 절대값이 1보다 커지므로 다음 부등식이 성립한다

$$w^T x_+ - w_0 \geq 1$$

$$w^T x_- - w_0 \leq -1$$

판별 경계선과 점  $x_+$ ,  $x_-$  사이의 거리는 다음과 같이 계산할 수 있다.

$$\frac{w^T x^+ - w_0}{\|w\|} = \frac{1}{\|w\|}$$

$$-\frac{w^T x^- - w_0}{\|w\|} = \frac{1}{\|w\|}$$

이 거리의 합을 마진(margin)이라고 하며 마진값이 클수록 더 경계선이 안정적이라고 볼 수 있다.

그런데 이 합은 위의 정의에 의해 다음과 같이 표현될 수 있다.

$$\frac{w^T x^+ - w_0}{\|w\|} - \frac{w^T x^- - w_0}{\|w\|} = \frac{2}{\|w\|}$$

마진 값이 최대가 되는 경우는  $\|w\|$  즉,  $\|w\|^2$ 이 최소가 되는 경우와 같다.  
즉 다음과 같은 목적함수를 최소화하면 된다.

$$L = \frac{1}{2} \|w\|^2 = \frac{1}{2} w^T w$$

또한 모든 표본 데이터에 대해 분류는 제대로 되어야 하므로 모든 데이터  $x_i, y_i$  ( $i = 1, \dots, N$ )에 대해 다음 조건을 만족해야 한다.

$$y_i \cdot f(x_i) = y_i \cdot (w^T x_i - w_o) \geq 1 \quad (i = 1, \dots, N)$$

$$y_i \cdot (w^T x_i - w_o) - 1 \geq 0 \quad (i = 1, \dots, N)$$

위에서 스케일링을 사용하여 모든 데이터에 대해  $f(x_i)$ 가 1보다 크거나 -1보다 작게 만들었다는 점을 이용한다.

$$y_i \cdot f(x_i) = y_i \cdot (w^T x_i - w_o) \geq 1 \quad (i = 1, \dots, N)$$

$$y_i \cdot (w^T x_i - w_o) - 1 \geq 0 \quad (i = 1, \dots, N)$$

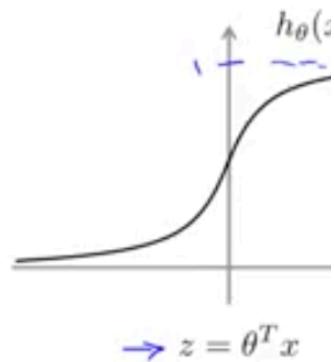
이런 최적화 문제를 풀면 판별함수를 얻을 수 있다.

**Cost function**

SVM은 로지스틱 회귀에서부터 유도될 수 있다.

## Alternative view of logistic regression

$$\rightarrow h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$



If  $y = 1$ , we want  $h_{\theta}(x) \approx 1$ ,  $\theta^T x \gg 0$   
 If  $y = 0$ , we want  $h_{\theta}(x) \approx 0$ ,  $\theta^T x \ll 0$

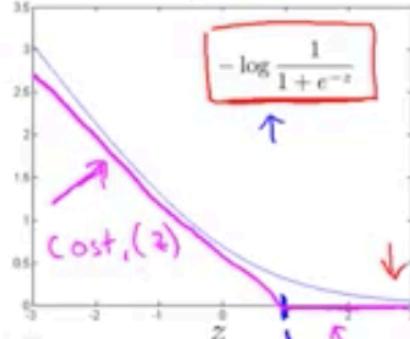
$\theta^T x$ 가 0보다 크면 1을 출력하게 하고 싶고  
 $\theta^T x$ 가 0작으면 0을 출력하게 하고 싶다고 할 때

## Alternative view of logistic regression

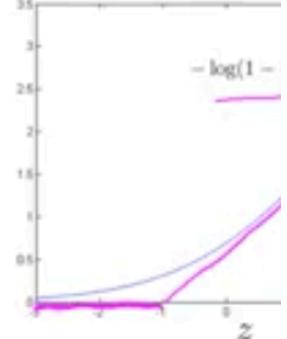
Cost of example:  $-(y \log h_\theta(x) + (1-y) \log(1 - h_\theta(x)))$

$$= -y \log \frac{1}{1 + e^{-\theta^T x}} - (1-y) \log(1 - \frac{1}{1 + e^{-\theta^T x}})$$

If  $y = 1$  (want  $\theta^T x \gg 0$ ):



If  $y = 0$  (want  $\theta^T x \ll 0$ ):



로지스틱 회귀의 cost function은 위와 같다.

그런데, SVM의 cost function은  $y = 1$ 을 위한 왼쪽 항을 1을 기점으로 평탄하게 만드는 비슷한 cost 함수를 사용하고

$y=0$ 을 위한 오른쪽 항은 -1을 기점으로 평탄하게 만드는 비슷한 cost 함수를 사용한다.

이를 cost\_1(z), cost\_0(z) 라 부른다.

1을 위한 cost function  $\rightarrow$  cost\_1(z)

0을 위한 cost function  $\rightarrow$  cost\_0(z)

## Support vector machine

Logistic regression:

$$\min_{\theta} \frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \underbrace{\left( -\log h_\theta(x^{(i)}) \right)}_{\text{cost}_1(\theta^T x^{(i)})} + (1-y^{(i)}) \underbrace{\left( -\log(1 - h_\theta(x^{(i)})) \right)}_{\text{cost}_0(\theta^T x^{(i)})} \right]$$

Support vector machine:

$$\min_{\theta} C \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \lambda \frac{1}{2} \|\theta\|^2$$

$$\min_u u^2 + 1 \rightarrow u=5$$

$$\min_u \|u\|^2 + 10 \rightarrow u=5$$

$$A + \lambda B \leftarrow C \quad C = \frac{1}{2}$$

$$\min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \|\theta\|^2$$

그리고 이를 대치시킨후,  $1/m$ 을 제거한다. ( $1/m$ 은 그냥 상수이므로  $\theta$ 를 최소화시키는데 아무런 영향을 미치지 못함)

그리고 정규화항 앞에 있는 항전체를 A라 놓고 뒤에 있는 정규화항을 B라고 놓았을때 logistic regression의 cost function은  $A + \lambda B$ 라 놓을수 있는데

여기서 SVM은 정규화에 따른 트레이드 오프를 적절하게 계산하여  $\lambda$ 를 C로 대치한다.

그리고 C를 앞의 A에 곱해준다  $CA + B$  ( $C = 1/\lambda$  가 될수도 있음)  
 hypothesis

### SVM hypothesis

$$\rightarrow \min_{\theta} C \sum_{i=1}^m [y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)})]$$

Hypothesis:

$$h_{\theta}(x) \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

SVM의 가설은 다음과 같다

$\theta^T x$ 가 0 보다 크거나 작으면 1을 출력하게 하고, 그외의 경우 0을 출력한다.

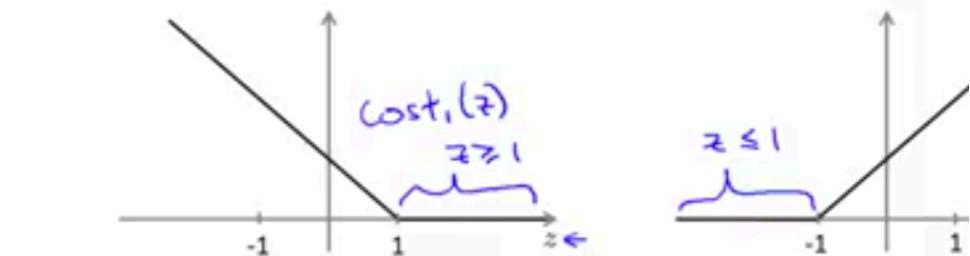
Large Margin Classifier

Support vector machine을 Large margin classifier 라고도 한다.

왜 이렇게 부르는지 이유를 살펴보자

### Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^m [y^{(i)} \underline{\text{cost}_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underline{\text{cost}_0(\theta^T x^{(i)})}]$$



- If  $y = 1$ , we want  $\underline{\theta^T x \geq 1}$  (not just  $\geq 0$ )
- If  $y = 0$ , we want  $\underline{\theta^T x \leq -1}$  (not just  $< 0$ )

Support vector machine은 1과 -1을 기준으로 1보다 더 크면 패널티가 0 이되고( $y$ 가 1일 때), -1 보다 작으면 패널티가 0이된다( $y$ 가 0일때)

그런데 이게 좀 이상하다.  $\theta^T x$ 가 0이 됨을 기준으로 패널티를 줄수도 있지 않은가?

왜냐하면  $\theta^T x$ 가 0이상이면 1로 분류할것이니까 말이다.

그 이유는  $\theta^T x$ 에 1, -1을 기준으로 패널티를 부과하면 좀더 안전하기 때문이다(safety margin)

그리고 이러한 조건(라벨이 1일때  $\theta^T x$ 가 1보다크거나 같음, 0일때  $\theta^T x$ 가 -1보다 작거나 같음)을 만족하도록 하게하면

결국 앞에 있는 항이 0이 되어버린다.

## SVM Decision Boundary

$$\min_{\theta} C \sum_{i=1}^m [y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)})] + \frac{1}{2} \|\theta\|^2$$

Whenever  $y^{(i)} = 1$ :

$$\theta^T x^{(i)} \geq 1$$

$$\min_{\theta} C \sum_{i=1}^m \epsilon_i + \frac{1}{2} \|\theta\|^2$$

$$\text{s.t. } \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

Whenever  $y^{(i)} = 0$ :

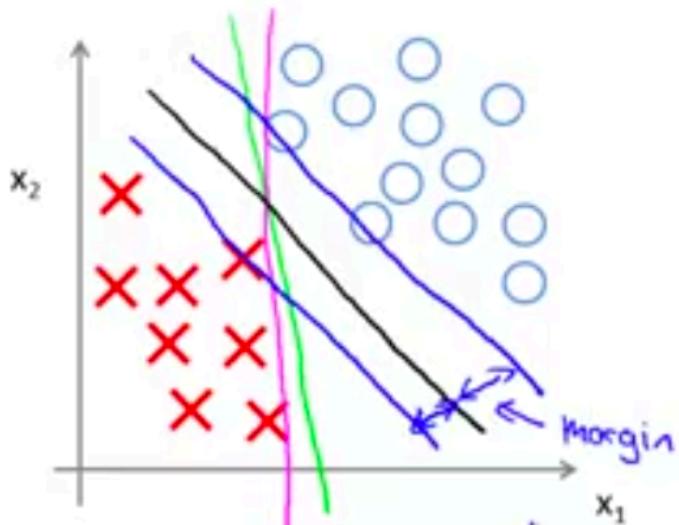
$$\theta^T x^{(i)} \leq -1$$

만약에  $C$ 가 아주 큰값이면 이를 최소화하기 위해 앞에 있는 항을 최소화해야하는 큰 동기가 될것이다.

이때 앞에 있는 항을 최소화해주려면 라벨이 1일때  $\theta^T x$ 가 1보다 커야할것이고 라벨이 0일때  $\theta^T x$ 가 -1보다 작아야할것이다.

그래서 결국 SVM은 이런 값들을 최대한 찾아서 찾아서  $\theta$ 를 최적화 시킨다.

## SVM Decision Boundary: Linearly separable case



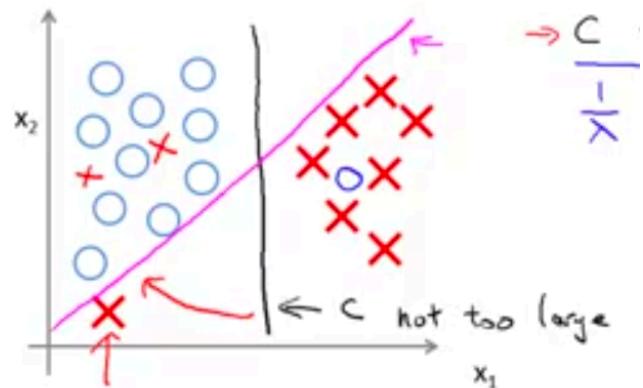
Large margin classifier

위 그림에서, 초록색과 분홍색은 샘플에 너무 가깝기 때문에 일반화가 잘 되지 못할것이다. SVM은 Margin을 두는 decision boundary를 찾기 때문에 파란색과 같이 샘플에서 떨어진 선을 찾을수 있고,

이를 통해 더 일반화가 잘되는 모델을 만들수 있다.

C가 너무 크다면?

## Large margin classifier in presence of outliers



C가 너무 크면 이상치에 민감해진다.

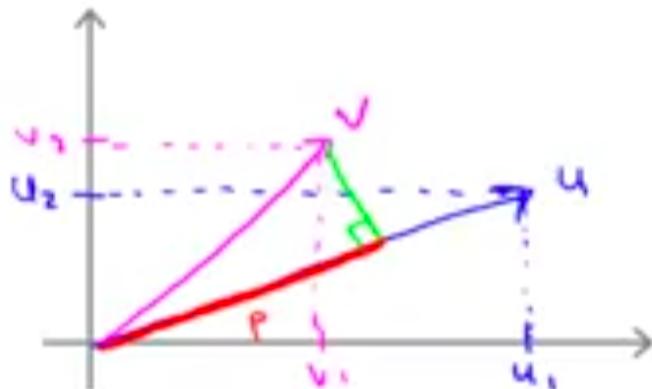
왜냐하면, cost function의 앞에 있는 항을 최소화 시키기 위해 샘플에 최대한 최적화 될것이기 때문이다.

만약 C가 작으면 이러한 이상치에 덜 민감해져서 더 일반화가 좋은 모델을 얻을수도 있다.

그래서 이러한 trade-off를 생각해서 모델을 만들어야 한다.

### 수학적 해석

두 벡터의 내적은 수학적으로 인접변과 기준벡터의 노름의 곱이다.



라벨이 1일때  $\theta^T x \geq 1$ 보다 크거나 같아야하고, 라벨이 0일대  $\theta^T x \leq 0$ 보다 작거나 같다 는 조건을 주었을때 코스트함수는 다음과 같다.

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

여기서  $\theta^2$ 은 노름으로 표현할수 있다.

(단순화를 위해  $\theta$ 를 2개로 제한하고,  $\theta_0$ 는 0이라고 둠)

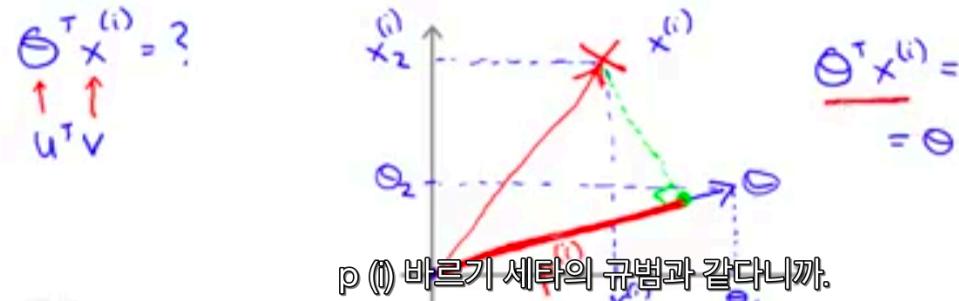
## SVM Decision Boundary

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} (\theta_1^2 + \theta_2^2) = \frac{1}{2} (\underbrace{\sqrt{\theta_1^2 + \theta_2^2}}_{= \|\theta\|})^2 = 1$$

s.t.  $\theta^T x^{(i)} \geq 1$  if  $y^{(i)} = 1$   
 $\theta^T x^{(i)} \leq -1$  if  $y^{(i)} = 0$

Simplification:  $\theta_0 = 0$ ,  $n=2$

$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}, \theta_0 = 0$$



다시 조건을 보면,

여기서  $\theta^T x$ 는 내적으로 해석되어, 인접변( $p$ )과  $\theta$ 의 곱으로 표현할 수 있다.

$$= p^{(i)} * \|\theta\|$$

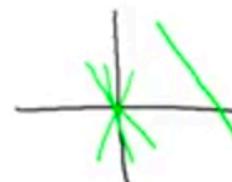
$\theta^T x$ 가 1보다 크거나 -1보다 작게 한다는 건  $p^{(i)} * \|\theta\|$  가 1보다 크거나 -1보다 작게 한다는 것을 뜻한다.

이는 어떤 뜻일까?

## SVM Decision Boundary

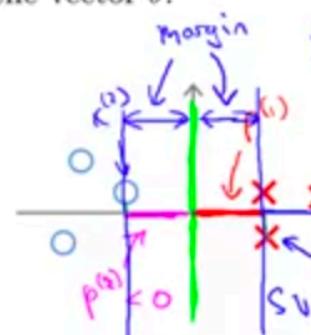
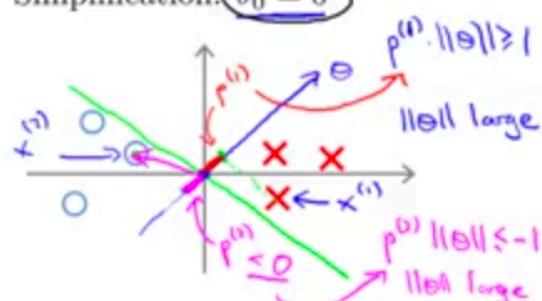
$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} \|\theta\|^2 \leftarrow$$

s.t.  $p^{(i)} \cdot \|\theta\| \geq 1$  if  $y^{(i)} = 1$   
 $p^{(i)} \cdot \|\theta\| \leq -1$  if  $y^{(i)} = 0$



where  $p^{(i)}$  is the projection of  $x^{(i)}$  onto the vector  $\theta$ .

$$\text{Simplification: } \theta_0 = 0$$



위 그림의 왼쪽 그림을 보자

decision boundary가 초록색 영역이라 할 때,

x를 라벨이 1인 경우, o를 라벨이 0인 경우라 하면

첫 번째 x에 대한  $p^{(1)}$ 은 빨간색 영역이고 다음과 같다.

$p^{(1)} * \|\theta\| \geq 1$ 을 만족하게 하려면  $p^{(1)}$ 이 양수이면서 작은 값이므로  $\|\theta\|$ 가 큰 값이 되어야 한다.

그리고 첫 번째 o에 대한  $p^{(2)}$ 는 분홍색 영역이고,

$p^2 * ||\theta|| \leq -1$ 을 만족하게 하려면,  $p^2$ 가 음수이면서 작은 값이므로  $||\theta||$ 가 큰 값이 되어야 한다.

위 그림의 오른쪽을 보면

$p^1$ 과  $p^2$ 가 넓어졌다.

이는  $p^1 * ||\theta|| \geq 1$ 을 만족하게 하는  $||\theta||$ 를 상대적으로 작은값으로 설정할수 있게 하고

$p^2 * ||\theta|| \leq -1$ 을 만족하게 하는  $||\theta||$ 를 상대적으로 작은값을 설정할수 있게 한다. 그런데

cost function의 목표는  $||\theta||$ 를 작게하는 것이므로,

이러한 조건(라벨이 1일때  $\theta^T x$ 가 1보다 크거나 같아야하고, 라벨이 0일때  $\theta^T x$ 가 0보다 작거나 같다는 조건)은

오른쪽의 decision boundary를 설정할수 있게 한다.

다른 말로하면, cost function은  $||\theta||$ 를 작게하는 목적함수이므로  $||\theta||$ 는 작게 설정된다.

그런데  $p^i$ 와  $||\theta||$ 의 내적이 절대값 1보다 커져야 하므로,  $p^i$ 가 커져야 한다.

### 하드 마진 분류(hard margin classification)

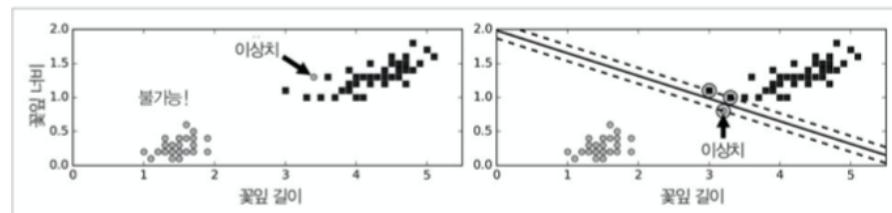
모든 샘플이 도로 바깥쪽에 올바르게 분류된것을 말한다.

#### 문제점

1. 데이터가 선형적으로 구분될수 있어야 제대로 작동한다.

2. 이상치에 민감하다.

그림 5-3 이상치에 민감한 하드 마진

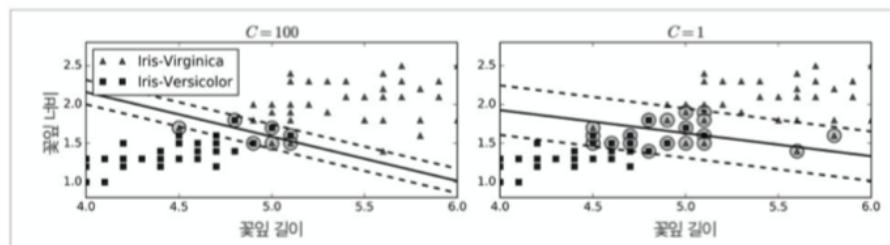


### 소프트 마진 분류(soft margin classification)

도로의 폭을 가능한 넓게 유지하는 것과 마진 오류(margin violation) 사이에 적절한 균형을 잡는 분류

이를 소프트 마진 분류라 한다.

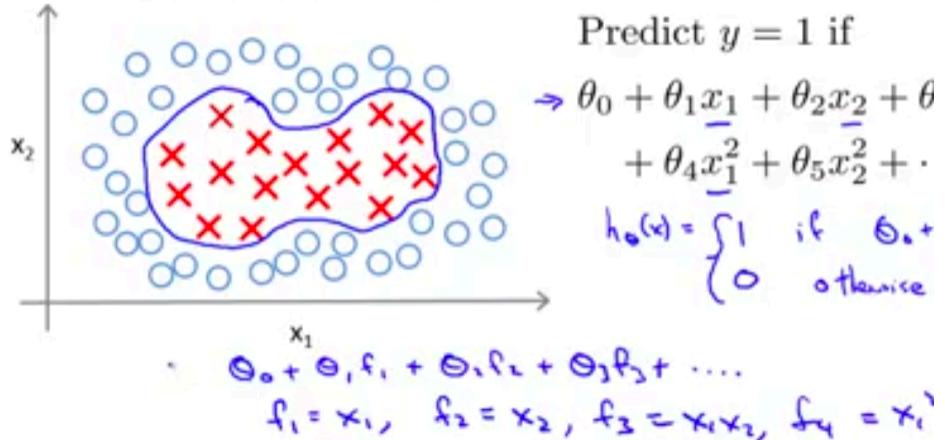
그림 5-4 좁은 마진과 넓은 마진



### Kernel

복잡한 비선형 결정경계를 만드는 방법

## Non-linear Decision Boundary



Is there a different / better choice of the features  $f_1, f_2, f_3$ ,

복잡한 경계를 만드는 방법은 당연히 복잡한 다항식을 사용하는 것이다.

그런데 복잡한 다항식을 만들 때, 더 좋은 feature를 선택할 수 있을까?

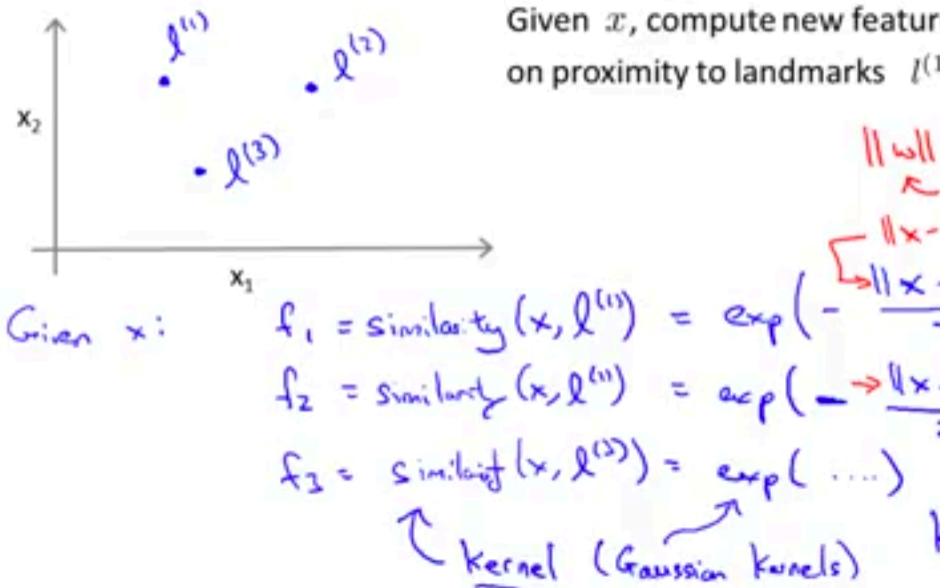
만약에 더 좋은 feature를 선택할 수 있다면,  $x$  대신  $f(featrue)$ 를 사용하여 고차 다항식을 표현할 수 있을 것이다..

이를 위한 한 가지 방법을 소개한다.

$x$ 가 주어졌을 때, landmark에 얼마나 근접하느냐를 기준으로 새로운 feature를 계산하는 것이다.

밑에서 proximiy를 계산하는 similarity function을 kernel이라고 부른다.(가우시안 커널을 사용하기도 함)

## Kernel



커널을 통한 새로운 feature를 만들어내는 것을 이해하기 위해 예를 들어보자

만약 데이터  $x$ 가 랜드마크  $l^{(1)}$ 에 가깝다면  $f_1$  feature는 1에 가까워 질 것이다.(분자가 0에 가까워 지므로 지수함수가 1이 됨)

만약 데이터  $x$ 가 랜드마크  $l^{(1)}$ 에서 멀어진다면  $f_1$  feature는 0에 가까워 질 것이다.(분자가

마이너스 부호로 아주 커지면서 지수함수가 0이 됨)

### Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_j}{2\sigma^2}\right)$$

If  $x \approx l^{(1)}$  :

$$f_1 \approx \exp\left(-\frac{0}{2\sigma^2}\right) \approx 1$$

$\begin{matrix} l^{(1)} \\ l^{(2)} \\ l^{(3)} \end{matrix}$

If  $x$  if far from  $l^{(1)}$  :

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

만약  $l^{(1)}$ 이  $= [3, 5]$ 라면  $x$ 는  $[3, 5]$ 에서 가장 큰값 1을 가질테고, 여기에서 멀어지는 값들은 점점 1에서 멀어질 것이다.

그리고 분모의 시그마 값이 작은 값이 된다면 데이터  $x$ 와  $f_1$ 이 이루는 분포는 점점 기울기가 커지고

시그마의 값이 커진다면 데이터  $x$ 와  $f_1$ 이 이루는 분포는 점점 기울기가 작아지게 된다. (여기서 시그마는 가우시안 커널의 파라미터)

왜냐하면.. 당연히 지수함수의 지수에 있는 분모가 작아지면 지수가 커지는데, 음의 부호가 붙어서 커지므로 지수함수의 값이 작아진다.

따라서, 값들이 전체적으로 땅바닥에 붙게된다.(기울기가 가파라짐)

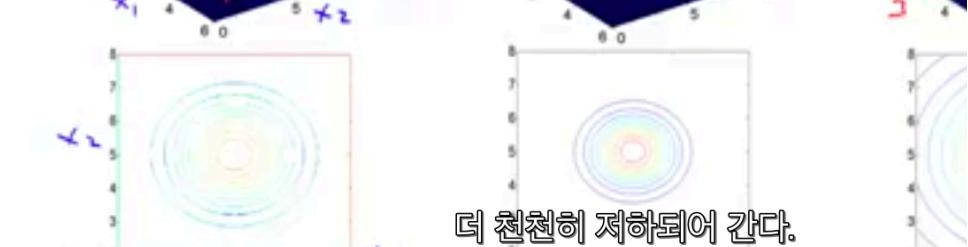
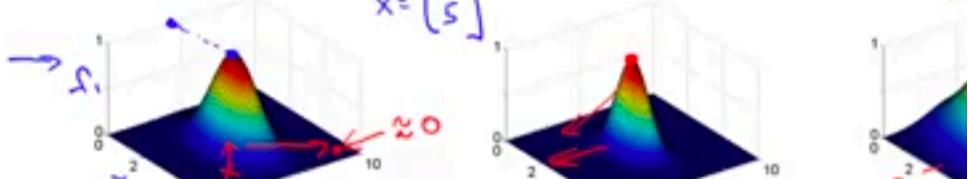
반면, 분모가 커지면 지수가 작아지고 음의 부호가 붙어서 작아지므로 지수함수의 값이 커지게 된다.

따라서, 값들이 전체적으로 위로 붙게된다.(기울기가 평탄해짐)

### Example:

$$\rightarrow l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$\rightarrow \sigma^2 = 1, \quad x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad \sigma^2 = 0.5$$



(The peak gradually decreases.)

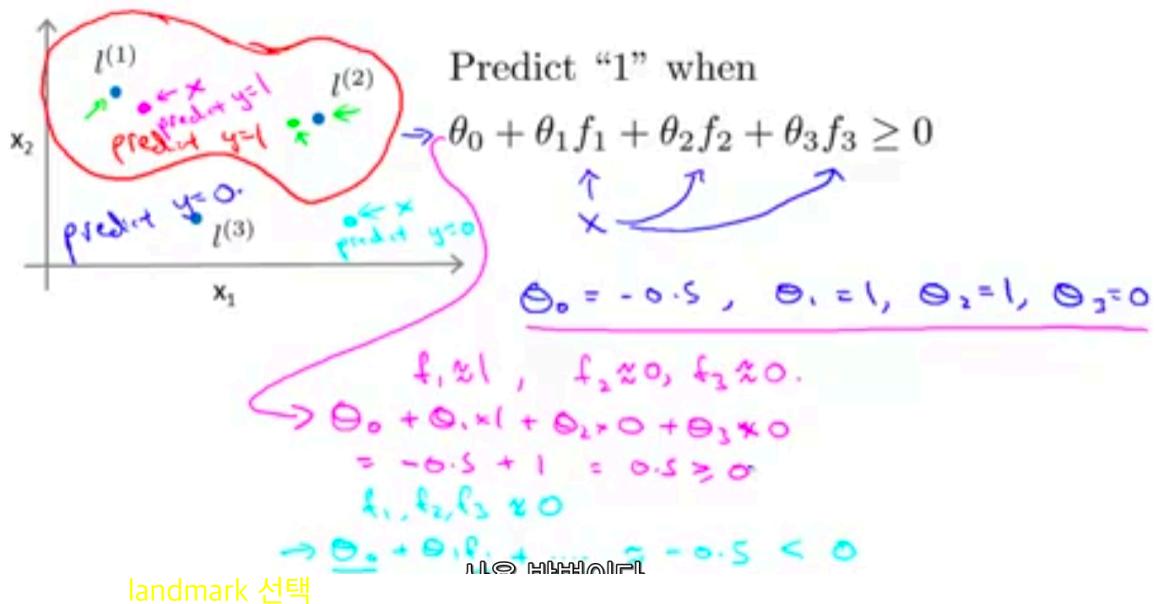
그렇다면, 이러한 랜드마크가 도대체 어떤 역할을 하는가?

1을 예측한다고 해보자.

만약 파라미터가 밑의 그림과 같이 설정되어 있다고 한다면,  
랜드마크  $l^{(1)}, l^{(2)}$ 에 가까운 데이터들은 1로 분류될 것이다.

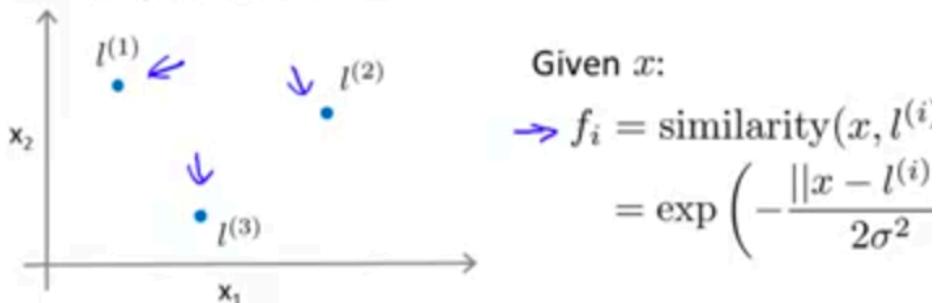
그리고 그 외는 0이 된다.

그래서 결국엔 랜드마크  $l^{(1)}$ 과  $l^{(2)}$  주변에 비선형 결정경계가 생겨난다.



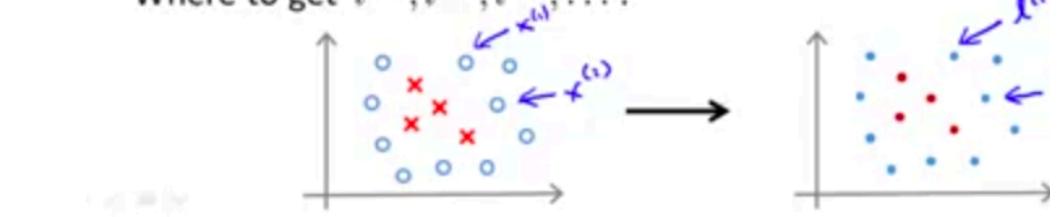
training 데이터 각각과 같은 landmark를 설정

### Choosing the landmarks



Predict  $y = 1$  if  $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get  $l^{(1)}, l^{(2)}, l^{(3)}, \dots$ ?



### SVM with Kernels

- Given  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$
- choose  $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$ .

Given example  $\underline{x}$ :

$$\begin{aligned} \rightarrow f_1 &= \text{similarity}(x, l^{(1)}) \\ \rightarrow f_2 &= \text{similarity}(x, l^{(2)}) \\ \dots \end{aligned}$$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$$

For training example  $(x^{(i)}, y^{(i)})$ :

$$\begin{aligned} \underline{x}^{(i)} \rightarrow f_1^{(i)} &= \text{sim}(x^{(i)}, l^{(1)}) \\ f_2^{(i)} &= \text{sim}(x^{(i)}, l^{(2)}) \\ \vdots & \\ f_m^{(i)} &= \text{sim}(x^{(i)}, l^{(m)}) = \exp(-\frac{\|x^{(i)} - l^{(m)}\|^2}{2\sigma^2}) = 1 \end{aligned}$$

그러면 비용함수 계산은 다음과 같게 됨

$\theta^T x^{(i)}$  대신  $\theta^T f^{(i)}$  를 쓰게 됨

뒤에 있는 항은

$\theta^T \theta$  대신에  $\theta^T M \theta$  으로 표현되는데 이는 거리와 비슷한 표현임

여기서  $M$ 은 데이터 개수가 늘어날 수록 불어남(이 행렬 계산 때문에 느려짐)

### SVM with Kernels

Hypothesis: Given  $\underline{x}$ , compute features  $f \in \mathbb{R}^{m+1}$  ⊕  
 $\rightarrow$  Predict "y=1" if  $\theta^T f \geq 0$        $\theta_0 f_0 + \theta_1 f_1 + \dots + \theta_m f_m$

Training:

$$\min_{\theta} C \sum_{i=1}^m y^{(i)} \underbrace{\text{cost}_1(\theta^T f^{(i)})}_{\theta^T f^{(i)} < 0} + (1 - y^{(i)}) \underbrace{\text{cost}_0(\theta^T f^{(i)})}_{\theta^T f^{(i)} \geq 0} +$$

$$\rightarrow \left[ - \sum_j \theta_j^2 \right] = \theta^T \theta \quad \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix} \quad (\text{ignoring } M = 10,$$

가우시안 커널을 사용시 주의점

Kernel (similarity) functions:

```

function f = kernel(x1, x2)
    f = exp(-||x1 - x2||^2 / (2 * sigma^2))
    return f

```

→ Note: Do perform feature scaling before using the Gauss

$$\boxed{\|x - l\|^2} \rightarrow v = x - l$$

$$\|v\|^2 = v_1^2 + v_2^2 + \dots + v_n^2$$

$$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \dots + (x_n - l_n)^2$$

$\underbrace{\qquad\qquad\qquad}_{1000 \text{ feet}^2}$  1-5 beds

가우시안 커널을 사용할 때, 데이터의 feature의 scale이 차이가 있다면 문제가 있을 수 있다.

그 이유는 landmark와의 거리를 계산할때, 차분을 구하고 이를 제곱하게 되는데, 이 때 큰 scale은 더욱더 커지기 때문에

이런 scale의 feature만 영향을 크게 줄수 있다.

따라서 이런 경우에는 feature scaling을 하는 것이 바람직하다.

여러가지 커널

Similarly,  $\mathbb{P}(\text{H}_1 \mid \text{D}) = \frac{\mathbb{P}(\text{D} \mid \text{H}_1) \mathbb{P}(\text{H}_1)}{\mathbb{P}(\text{D} \mid \text{H}_1) \mathbb{P}(\text{H}_1) + \mathbb{P}(\text{D} \mid \text{H}_0) \mathbb{P}(\text{H}_0)}$

### Other choices of kernel

Note: Not all similarity functions  $\text{similarity}(x, t)$  make valid k-  
→ (Need to satisfy technical condition called "Mercer's Theorem"  
sure SVM packages' optimizations run correctly, and do not diverge)

Many off-the-shelf kernels available:

- Polynomial kernel:  $k(x, l) = \begin{cases} (x^T l)^2 & \text{if } k=2 \\ (x^T l)^3 & \text{if } k=3 \\ (x^T l + 1)^2 & \text{if } k=4 \\ (x^T l)^4 & \text{if } k=5 \end{cases}$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...  
 $\text{sim}(x, y)$

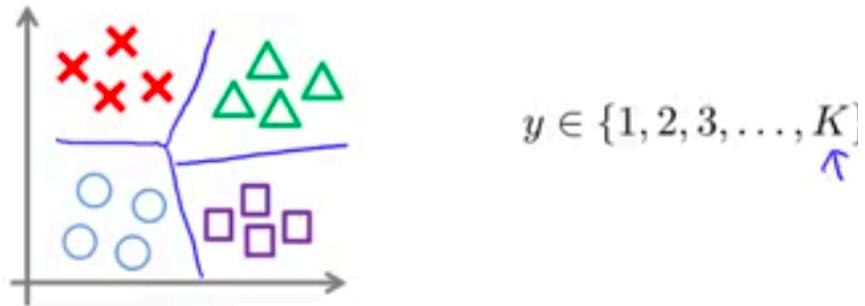
polynomial kernel, string kernel, chi-square kernel, histogram intersection kernel 등이 있는데

여기서 polynomial kernel은 두개의 변수를 받는다.  
constant 값과 degree 값이다.

$$(x^T \theta + \text{constant})^{\text{degree}}$$

Multi-Class Classification

### Multi-class classification



Many SVM packages already have built-in multi-class classification functionality.

→ Otherwise, use one-vs.-all method. (Train  $K$  SVMs, one to  $y = i$  from the rest, for  $i = 1, 2, \dots, K$ ), get  $\theta^{(1)}, \theta^{(2)}, \dots$ . Pick class  $i$  with largest  $(\theta^{(i)})^T x$

$\uparrow$   $y=1$   $\downarrow$   $y=2$

SVM도 Logistic classification과 마찬가지로 multi class classification 시에 one vs all을 사용한다.

그래서  $i$  클래스에 해당하는  $\theta^{(i)}$  를 찾고

$(\theta^{(i)})^T x$  가 큰 값이 되는  $i$ 를 찾아서 classification 한다.

이때 비용함수는 당연히 클래스마다 달라질 것이다. - (랜드마크가 달라질것이고 각각의 데 이터의 label이 달라질 거니까 당연)

### SVM 파라미터

C에 의해 편향 분산 트레이드 오프가 결정됨

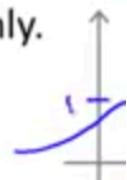
만약 가우시안 커널을 사용한다면 시그마에 의해서도 편향 분산 트레이드 오프가 결정됨

### SVM parameters:

$C = \frac{1}{\lambda}$ .  $\rightarrow$  Large C: Lower bias, high variance. (sm)  $\rightarrow$  Small C: Higher bias, low variance. (large)

$\sigma^2$  Large  $\sigma^2$ : Features  $f_i$  vary more smoothly.  
 $\rightarrow$  Higher bias, lower variance.  

$$\exp\left(-\frac{\|x - \mathbf{x}^{(i)}\|^2}{2\sigma^2}\right)$$



Small  $\sigma^2$ : Features  $f_i$  vary less smoothly.  
Lower bias, higher variance.



### Logistic regression 과의 비교

#### Logistic regression vs. SVMs

$n$  = number of features ( $x \in \mathbb{R}^{n+1}$ ),  $m$  = number of training samples

- $\rightarrow$  If  $n$  is large (relative to  $m$ ): (e.g.  $n \geq m$ ,  $n = 10,000$ ,  $m = 10$ )  
 $\rightarrow$  Use logistic regression, or SVM without a kernel ("linear kernel")
- $\rightarrow$  If  $n$  is small,  $m$  is intermediate: ( $n = 1-1000$ ,  $m = 10-1000$ )
  - $\rightarrow$  Use SVM with Gaussian kernel
- If  $n$  is small,  $m$  is large: ( $n = 1-1000$ ,  $m = 50,000+$ )
  - $\rightarrow$  Create/add more features, then use logistic regression or without a kernel
- $\rightarrow$  Neural network likely to work well for most of these settings, slower to train.

여기서는 하나의 가이드 라인을 보여준다.

feature의 개수가 데이터 개수보다 클 경우

로지스틱 회귀나 kernel 없는 SVM을 사용하는 것이 좋다.

왜냐하면 feature가 크고 데이터의 개수가 작으면 복잡한 다항식을 적용할 때, overfitting의 위험이 있기 때문이다.

feature의 개수가 작고 데이터의 개수가 중간쯤이라면

가우시안 커널을 이용한 SVM을 쓰는게 좋다.

feature의 개수가 작고, 데이터의 개수가 크다면

logistic regression 혹은 커널없는 SVM이 좋다.

이때 가우시안 커널을 사용하면 아주 느려질 수 있기 때문이다.

Neural Network 사용

인공 신경망은 모든 문제에 잘 작동하지만 train 하는 것이 느릴 수도 있다.

그리고 SVM은 global minimum을 반드시 찾기 때문에 local minimum 문제를 고민

할 필요가 없다.

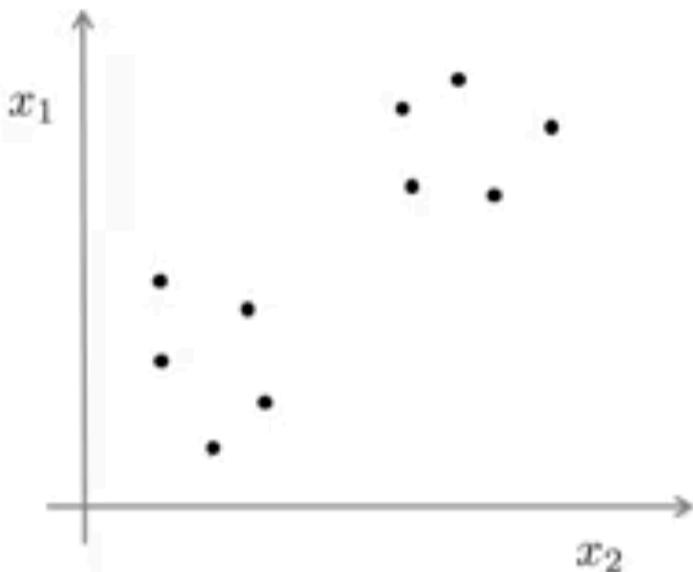
### no kernel SVM

kernel이 없는 SVM은 linear kernel과 동의어이며  
결국  $\theta^T x$ 을 계산하겠다는 의미이다.

### 비지도학습(unsupervised learning)

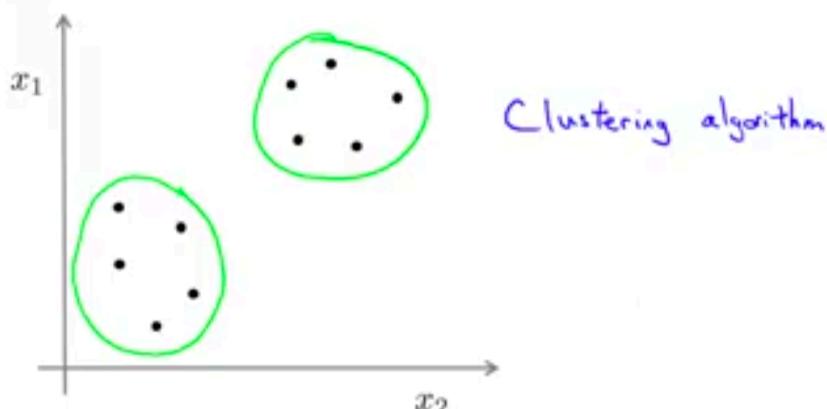
라벨이 없는 데이터에 대해 패턴을 찾아냄

## Unsupervised learning



Training set:  $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$  ←

### Unsupervised learning



Training set:  $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$  ←

