

Reinforcement

Learning

강화학습 Reinforcement Learning [RL]

어떤 환경을 탐색하는 에이전트가 현재의 상태를 인식하여 어떤 행동을 취한다

그러면 그 에이전트는 환경으로부터 보상을 얻게 된다. (보상은 양수/음수 가능)

강화학습의 알고리즘은 그 에이전트가 앞으로 누적될 보상을 최대화하는 일련의 행동으로 정의하는 정책을 찾는 방법이다.

<학습>

- 환경과의 상호작용을 통한 학습 → 이를 Computational하게 접근 ⇒ ML

강화학습은 ML의 한 학습 방법이다.

여러가지 행동해야 할 술은 모르지만 환경과 상호작용하여 배워가는 것

<ML>

1. Supervised L : 자료학습, 정답을 알 수 있음, 바로바로 대응해
2. Unsupervised L : 비지도 학습, 정답이 없는 분류와 같은 문제 끈다.
3. Reinforcement L : 정답보통. 한 행동에 대한 보상으로 학습. (MDP)

ex) 컴퓨터, 로봇 수행

<RL 특징>

1. Trial and Error : 해보면서 조정하는 것
2. Delayed Reward

: 강화학습은 시간이라는 개념이 포함되어 있다.

RL은 시간의 순서가 있는 문제를 풀기 때문에 지금 한 행동으로 인한 환경의 반응이 늦어질 수 있다.

Lecture #1 Introduction

Characteristics of AI

- supervisor 가 종종하지 않고 reward 를 통해 학습 \rightarrow 성능보다도 optimal 한 결과가 나온다!
- delayed feedback (=reward) \rightarrow 문제에 대해 훈련
- Time really matters, sequential
- Agent's action affect the subsequent data it receives

용어 정리

Punaward (Pkt)

- scalar feedback signal : 웃자 하는 힘들한 일을 하거나 행동에 적용
- t: time static
- 주어진 Punaward 를 maximize 하는 것이 목표
- sequential decision making : select actions to maximize total future reward

Agent

: 우리가 학습시킬 대상, machine 이라 부른다.

Environment

: Agent 를 제외한 모든 것
Observation 과 reward 를 Agent에게 준다.

History Ht

: sequence of observations, actions, rewards

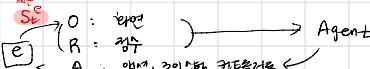
State St

: 다음 행동에 대해 쓰이는 정보

(1) Environment state St^e

- O, R, P 등 다양한 쓰는 모든 정보 agent에게 보여주지 않음
- environment's private representation
- 행동 결과가 환경에 반영으로 가는 경로
- agent 외의 외부세계로 출입하는 경로까지 보면서 더 혼란스러울 수 있다.

Ex. Alarm example



(2) Agent state St^a

- 이는 action 을 학습해 몇몇은 정보를, 내가 학습하는 모든 것
- ex) 주식을 내가 하려고 하면서, 어떤 기록이 불필요하고 어떤 정보도 불필요해 이 모든 정보들이 나의 state = agent state

(3) Information State (Markov state)

이전 state 가 Markov 한지의 지침

Markov 하다 : 어떤 결정을 할 때 바로 이전 state 를 놓쳤다

$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, \dots, S_t]$$

이전 state 를 바로 St(현재) 를 결정하는 경우 전 모든 상태로 St^a 를 생각하는 이 같은 때 Markov 하다고 한다

ex. 헬리콥터 조종.

현재의 위치, 각도, 배터리 수치 등이 주어진 상태

이제 미래 상태가 필요할까?

결정의 내용은 당장 그 전 상태만 필요하다.

즉, environment state 는 Markov state 라고 할 수 있다.

또한 history 도 Markov 하다.

But Agent state 는 Markov 하지 않다.

Ex) Agent : 나 속도를 모르는 상태

차를 몰다가 급발진을 했다. 그 바로 다음에 이런 상태만으로 현재의 속도를 경험하는 건 이상한Markov 하지 않다

(4) Fully Observable Environment

: agent 가 environment state 를 directly 볼 수 있는 것

$$O_t = S_t^a = S_t^e$$

- 즉서적으로 이를 MDP (Markov decision process) 라고 한다

(5) Partial observability

: agent 가 environment state 를 관찰하지 못하는 것

$$O_t \neq S_t^e \neq S_t^a$$

- POMDP (partially observable Markov decision process)

Major Components of an RL agent

*代理人에게 이해하기 어렵거나
파악이 어렵거나 알 수 있는 것

(1) Policy π

: agent 의 행동

- state 를 넣어주면 action 을 뽑아준다.

- 종류

- 1) Deterministic policy : s 를 넣어주면 a 해가 결정된다
- 2) Stochastic policy : s 를 넣어주면 여러 a 를 확률로 준다.

(2) Value function V

: 미래 보상을 예측

$$V_t(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

인정이 S, R, 이 S에 따른 행동 policy 를 학습하는 것

개별의 행동에 대해서만 학습하는 것 (정책)
(policy 가 있으면 V 가 학습될 수 있다.) (environment 정책이 V에 영향을 미친다)
(정책을 기반으로 학습된다)

(3) Model P/R

: 행동이 미래에 될 예측

[Reward 예측 $P_{S|S}$: S에서 a 를 취한 S'로 갔다

State 예측 $P_{S'|S}$: S에서 a 를 취한 S'로 갔다

- model 은 원본처럼 만들 수 있다.

Categorizing RL agents (분류법)

Value Based : No Policy, Value Function

Policy Based : Policy, No Value Function

Actor Critic : Policy, Value Function

Model free : Policy and Value Function, No model

Model based : Policy and Value Function, Model

Problem

Reinforcement Learning

environment- 을 보는
policy- 을 개선하는 나가는 것
environment- 을 만나면 어떤 행동을 하는 것
마지막에 최종적 policy- 을 얻어내는 것

Exploration

좋은 것은 찾는 것, 탐색하는 것
잘못된 행동을 할 수 있으면 그래서 찾을 수 있지
Exploitation 찾을 수 있는 행동을 max 시키는 것

Prediction policy 가 결정했을 때 어떤
 \therefore value function 찾기

Control best policy 를 찾는 것

01 강의

Lecture #2 Markov Decision Process

1. Markov Processes

Introduction to MDPs

환경(env)을 표현하는 것. env가 모든 행동 가능할 상황

The current state completely characterises the process

Markov Property

The future is independent of the past given the present

state만 알면 history를 바라보 된다. state는 향후 행위를 가진다

State Transition Matrix

(* action이 있는 상태)

Markov state s and successor state s', the state transition probability

$$P_{ss'} = P[S_{t+1} = s' | S_t = s]$$

State transition matrix P (all states s to all successor states s')

$$P = \text{from } \begin{bmatrix} & \text{to} \\ P_{11} & \dots & P_{1n} \\ \vdots & & \vdots \\ P_{n1} & \dots & P_{nn} \end{bmatrix} \quad : \text{현 state에서 다음 state로}
어디를 갈지에 대한 확률$$

Markov Process

$\langle S, P \rangle$

action이 있고, state가 있으면 흐름이라는 특징

S: finite set of states

P: state transition probability matrix

2. Markov reward process

$\langle S, P, R, \gamma \rangle$

= Markov process + reward + γ

S: finite set of states

P: state transition probability matrix

R: reward function, $R_s = E[R_{t+1} | S_t = s]$

γ : discount factor, $\gamma \in [0, 1]$

* action이 있고, 향후 보상을 (P) 올 때 다음 state로 넘어가는 process

Return G_t

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

: total discounted reward from time step t

r close to 0: 향후 보상 없이 종료

r close to 1: 향후 모든 보상을 갖는 것.

Why Discount?

사실 가장 중요한 대답은 '수학적으로 편리해서, 편리이 좋은'

Value Function $V(s)$ (In MRP)

: return의 기대값, long-term value of state s

$$V(s) = E[G_t | S_t = s]$$

→ state s에 있을 때 return의 기대값.

* episode after G_t 가 아닌 G_{t+1}

Bellman Equation for MDPs

: value function의 정의

$$V(s) = E[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

why? SUM value는 다음 state에서의 MRP의 long-term value(적어도 state)가 있다.

$$V(s) = R_s + \gamma \sum_{s'} P_{ss'} V(s')$$

적어도 state가 한 번 더 있고, 다음 state에서의 value는 같은 것이다.

$$V = R + \gamma P_V$$

$$(1 - \gamma P)V = R$$

$$V = (1 - \gamma P)^{-1} R \leftarrow 이걸 한번 V를 바로 알 수 있다.$$

3. Markov Decision Process

$\langle S, A, P, R, \gamma \rangle$

= Markov Reward Process + Action

S: finite set of states

A: finite set of actions

P: state transition probability matrix,

$$P_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$$

R: reward function,

$$R_s^a = E[R_{t+1} | S_t = s, A_t = a]$$

γ : discount factor $\gamma \in [0, 1]$

Policies

$$\pi(a|s) = P[A_t = a | S_t = s]$$

* action은 선택한 대의 정책

agent의 policy가 고정되면, state의 가치가 고정된다.

where,

$$P_{s,a,s'} = \sum_{a \in A} \pi(a|s) P_{ss'}$$

$$R_s^a = \sum_{a \in A} \pi(a|s) R_s^a$$

마지막
마지막
마지막
마지막

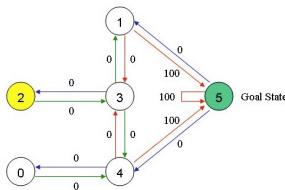


* Q Learning

$Q(state, action)$

$$= R(state, action) + \text{Gamma} * \text{Max}[Q(next state, all actions)]$$

1. 매개변수 Gamma 를 설정하고 matrix P에 현장변수를 입력(기울기)
2. matrix Q 를 모두 0으로 초기화
3. 다음 조작을 따른다



```

For each episode:
    임의의 state를 선택한다.
    Do While ( 목표치에 도달하지 않았다면 )
        현재 State에서 가능한 action 중 하나를 선택한다.
        선택한 action을 사용하여, next state로 향한다.
        모든 가능한 action을 기반으로 next state의 Q value 최댓값을 구한다.
        Compute: Q(state, action) = R(state, action) +
                                    Gamma * Max[Q(next state, all actions)]
        next state를 현재 state로 삼는다.
    End Do
End For

```

위 작업을 반복하다보면 learning 과정이 흐려져다가 어느 순간 수렴.

이 수렴값이 goal state 가 된다

gamma 값은 0~1 사이 값이 되어야 한다. gamma 값이 0에 가까우면 수렴값을 좀 더 빨리 찾게 되는 반면 그에 가까우면 좀 더 큰 weight 값을 가져 값의 의미가 명확해진다

ex. 현재 state = 1, action (1 → 3) 라고 생각했을 때

$$Q(1, 3) = R(1, 3) + 0.8 * \text{Max}[Q(3, 1), Q(3, 4), Q(3, 5)]$$

자정한 값 ↑

$$= 100 + 0.8 * 0 = 100$$

이전 속으로 matrix Q 를 계속 해으면 흐려지기 된다.

Book : 텐서플로로 구현하는 딥러닝과 강화학습

#1. 딥러닝 시작하기

딥러닝 : 특정 유형의 학습 머신러닝을 기반으로 하는 머신러닝 분야
상대적으로 깊은 수준에서 이런 수준의 결과를 입력받아
학습하고 학습화하는 작업을 수행하고, 이 작업을 여러 수준에
걸쳐 반복하는 작업을 수행하는 학습 모델

구조

- 주제 단계 stages 나 계층 layers 이 입력으로 수신한 정보를
이용해 변화.

인공 신경망 ANN

- 상물학적 뉴런

✓ 구성

• 세포체 or soma

• 회자 이상의 수송하기 dendrites : 다른 뉴런의 신호를 수신

• 죽식돌기 axon : 동일한 뉴런이 만든 신호를 연결된 다른 뉴런으로 전달

✓ 뉴런 활동

신호(활성상태)를 송신, 휴식, 다른 뉴런(비활성상태)으로 부터
신호를 수신하는 모래 볼漏

- 인공 신경망 세포

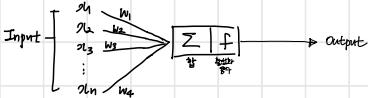
✓ 구성

• 2개 이상의 입력 연결 : 신경 세포 (neuron)로부터 신호를 수집

각 연결은 전달된 신호에 가중치를 부여

• 2개 이상의 출력 연결 : 다른 뉴런에 대한 신호를 전달

• 활성화 함수 : 입력 연결에서 획득한 신호에 대해 가중치를 적용하고,
활성화 함수를 적용한 후 양자화에 근거해 출력 신호
의 값을 최종 결정



✓ 신경망 학습

네트워크, 그리고 활성화 함수의 가중치를 초기화하는 것을 학습.
비행학교 교육을 만드는 과정에서 중요한 학습은 활성화 함수입니다.

- 인공 신경망 학습

• 신경망의 학습 과정은 가중치를 최적화하는 과정을 반복 수렴 과정으로 자주 학습이다.

가중치 수렴 과정은 신경망의 출력이 원하는 동향에 맞아가는 정도를 재설정하는 반복 과정인 것.

✓ 학습과 알고리즘

1. 신경망을 선형 기반으로 초기화

2. 모든 표수 사용 :

• 손정한 경로 : 원하는 출력과 실제 출력 간의 차이 계산

• 손정한 경로 : 모든 계층에 대해 손정한 경로를 학습으로 진행

3. 손비는 입력으로 네트워크에 제출 유통화를 흘리기 (그리 향상)

4. 표수를 흘리는 방법, 즉 → 손정한 경로를 네트워크에 반복

5. 손비의 확장화

- 경우 최적화 Gradient Descent, GD

1. 모든 계층에서 어떤 일부 손정한 경로를 추적해 선택

2. 각 계층에서 어떤 표수를 수정하는 경로 (G 계산)

3. 표수를 흘리는 방법, 즉 → 손정한 경로를 네트워크에 반복

4. 손비에 0이 가중치를 따라서 1. 3번의 반복

- 신경망 구조

노드 연결 맵, 흐름하는 계층들에 해당하는 입력과 출력 사이의 노드수를,
계층별 뉴런의 수만 보면 알 수 있다

✓ 다음 표로 정리

다음 표로는 계층마다 같은 계층의 인용 규모를 수립할 수 있다

• 각 계층은 다른 계층의 모든 뉴런과 연결된다.

• 흐름한 계층의 부른 사이즈는 1회차, 2회차

• 양방향 계층은 계층에 속한 뉴런 사이사이 연결이 없다

• 양방향 계층은 계층 사이에 연결이 아니다. 다른다

• 규모는 계층은 같은 계층과 그 계층과 같은 규모이다.

• 규모는 계층은 같은 계층과 그 계층과 같은 규모이다.

• 규모는 계층은 같은 계층과 그 계층과 같은 규모이다.

✓ DNN은 Deep Neural Networks

• Deep learning의 핵심은 인공 신경망

• 뉴런은 계층마다 서로 다른 계층에 대해 가능. 즉 내부 계층은

2주를 위한 강화학습

Lecture #2 OpenAI Gym

OpenAI Gym : env 를 제공해주는 Framework

Lecture #3 Q-Learning

State
action \rightarrow Q \rightarrow reward
input
output

여기 Q 중에 어느 Q로 갈 것인가? \Rightarrow Policy

MATH.

$\max_a Q(s_1, a)$: Q 가 가질 수 있는 최대값

$\pi_t^*(s) = \arg \max_a Q(s_1, a)$: Q 가 최대값이 되어야 하는 변수 a

optimal 하다는 의미.

Finding, Learning Q

Assume Q in s' exists!

\Rightarrow My condition

- I am in s
- when I do action a , I'll go to s'
- when I do action a , I'll get reward r
- Q in s' , $Q(s', a')$ exists!

$Q(s, a)$ 을 알고 싶다.!!

$$Q(s, a) = r + \max Q(s', a')$$

$$R_t = r_t + r_{t+1} + r_{t+2} + \dots + r_n$$

$$R_{t+1} = r_{t+1} + r_{t+2} + \dots + r_n$$

$$R_t = r_t + R(t+1)$$

$$R^*(t) = r_t + \max_{\text{optimal agent}} R(t+1)$$

Q-Learning Algorithm

For each s, a initialize table entry $Q(s, a) \leftarrow 0$

이제 맵을 보면

Observe current state s

Do forever : 환경 횟수 (episodes) 를 정하고 그 안에 Do forever.

Select an action a and execute it

선택 선택 \rightarrow 행동 실행

Receive immediate reward r

환경의 반응

Observe the new state s'

Update the table entry for $Q(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \max_a \hat{Q}(s', a')$$

$s \leftarrow s'$ \nwarrow 한 에피소드의 All reward 를 저장
하나에

\rightarrow 통 에피소드로 나누면 성능을 알 수 있음

모두를 위한 강화학습

Lecture #4

Exploit VS Exploration

현재 있는 값을 이용

미래를 찾을 모험학

* Lecture #3 Q-learning은 dummy.

E-greedy

e라는 값을 정하고, 랜덤한 값을 하나 뽑아

e보다 작으면 새로운 길로 (exploration),

e보다 크면 가장 좋은 길로 (exploit) 간다

decaying E-greedy

학습 초기에 랜덤한 길을 가는 것을 허용해

있으나 학습이 많이 된 뒤에 랜덤한 길을

가장 적은 비율로. e 값을 점점 낮게 해줌

add random noise

$$\text{argmax}(Q(s,a) + \text{random_values})$$

↑

랜덤한 노이즈를 더해준다.

→ noise가 행동을 주제인 타이밍은 값이기 때문에
앞의 방법처럼 원천 변경하지는 않다

Discounted reward

미래의 R값을 Discount 한다.

$$\hat{Q}(s, a) \leftarrow r + \gamma \cdot \max_{a'} \hat{Q}(s', a')$$

$$\begin{aligned} \hat{r}_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{n-t} r_n \\ &= r_t + \gamma (r_{t+1} + \gamma (r_{t+2} + \dots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

⇒ 더 가까운 미래에 reward를 크게 받는쪽으로
가져온다!

Convergence

\hat{Q} : Q값을 고수한 값 (유체적)

이 값은 Q를 수정한다

But ① In deterministic worlds
(정확한 세상)

* ② In finite states

$$\pi^*(s) = \underset{a}{\text{argmax}} Q(s, a)$$

(포함)

Lecture #5 Nondeterministic Worlds

- 내가 Right를 한다고 하서 원하는 결과가 나오지 않는 경우

→ Nondeterministic = stochastic

Solution

· Listen to Q(s') (just a little bit)

· Update Q(s) little bit (learning rate)

Learning Incrementally

Learning rate, $\alpha = 0.1$ (ex)

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha [r + \max_a Q(s', a)]$$

⑪

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \max_a Q(s', a')] - Q(s, a)$$

Convergence

ok. 많이 학습하면 \hat{Q} 가 Q에 가까워진다

모두를 위한 강화학습

Lecture #6 Q-Network

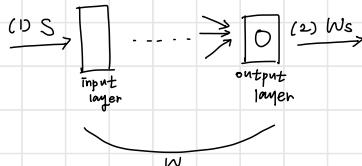
실제 문제는 너무 커서 써보면 Q-table 사용 불가

- ① input : state & action
- ② input : state. ~~사용~~ (Q-function Approximation)
→ 모든 가능한 액션에 대한 값을 찾다

Q-Network training (linear regression)

$$H(x) = Wx \quad \text{output} \rightarrow a^*$$

$$\text{Cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2 \quad \text{→ 관찰 행동 목표값}$$



$$W_s \sim \hat{Q}(s, a | \theta) \sim Q^*(s, a)$$

↑
function
using θ
 $= w(\text{weight})$
 $= \text{network}$

Algorithm

random weight

```

for episode 1, M do
    $S_1$ 을 받아오고 $p_1 = \phi(S_1) \leftarrow$ 전처리($p$는 $S$다)
    $S_1 = \{p_1\}$

    for $t=1, T$ do
        다음 action을 고른다 ($a_t$)
        - 방향 $e \in E$-greedy or $\max Q^*(\phi(S_t), a; \theta)$
        Execute $a_t$ and observe $r_t$ and Image $d_{tt}$
        Set $S_{t+1} = S_t . a_t . d_{tt}$
        and preprocess $\phi_{t+1} = \phi(S_{t+1})$
```

학습부분

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal}(p) \\ r_j + \gamma \max_a Q(\phi_{t+1}, a'; \theta) \end{cases}$$

Perform a gradient descent step on

$$(y_j - Q(\phi_j, a_j; \theta))^2$$

* 왜 Stochastic의식을 쓰지 않느냐?

네이버지도에서는 결정적 학습에서 의사식을 써도 문제된다

Convergence

- 오류다. diverges (불안정하다).
- 이 문제를 해결한 것이 DQN

Lecture #7 DQN

Big issues about 'diverges'

1. Correlations between samples

: training set에 있는 데이터 간의 상관 관계.

→ 문제와 유사한 이후 Linear Regression 시 아래 다를 때 (mean이 만들 이유 있음)

2. Non-stationary targets

$$Q = \text{Target}(q) = r_t + \gamma \max_a Q(\phi_{t+1}, a'; \theta)$$

$$\hat{Q} = \text{prediction}(\hat{q}) = \hat{Q}(\phi_t, a_t; \theta)$$

\$Q\$ 가 실제 가치함수로 \$\hat{Q}\$ 값을 업데이트 시키는데, target 역시 자동으로 바뀜.

같은 네트워크를 사용, 따라서 변경되는 것 (\$\theta\$를 업데이트)

DQN's three solutions

1. Go deep

깊이...

2. experience replay

우리가 돌아온 상태를 버퍼에 저장

정정한 시간이 지난 후 버퍼로부터 정정한 샘플을 가져와 학습을 시킨다

효과: 엔딩까지 가져오기 때문에 problem 2. Correlations between samples 를 해결할 수 있음. '랜덤'이 결국 정확한 결과와 같은 분포를 가져올 때

3. separate target network

\$\theta\$ 와 \$\bar{\theta}\$로 두 개의 네트워크를 사용

\$W_s\$ \$y\$ (target) \$\theta\$ 를 계속 업데이트.



정정한 시간이 지난 후 \$\bar{\theta}\$ 업데이트

모두를 위한 딥러닝 시즌1

Lecture #01 기본적인 ML의 용어와 개념 설명

ML (Machine Learning)

- explicit programming
→ 개별자마다 어떤 상황에 대해 행동을 모두 정해주는 것
→ 일부분 분야에 사용 어렵 → ML!

Supervised / Unsupervised learning

- Supervised : labeled examples (training set)
정해진 퍼터널로 학습하는 것
ex) 고양이 사진을 여러개 주고 학습시킨다
- Unsupervised : un-labeled data
ex) Google news grouping : 뉴스는 랜덤으로 반복수용
- 강의에서는 supervised 를 주로 다룬다.

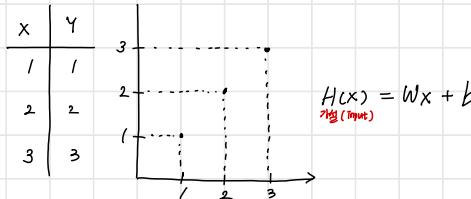
Supervised learning

Ex : Image, Email spam filter, Predicting exam score

Type of supervised learning

- ($y = \text{learning data} \rightarrow$)
- Regression : 점수와 같이 넓은 범위 예측
 - binary classification : pass/non-pass
 - multi-label classification : 여러가지의 라벨을 놓거나.

Lecture #02 Linear Regression



Linear regression : 직선과 같은 형태의 모델이 나올 것이다

Which hypothesis is better? 점과의 거리 측정
 $= \text{Cost function}$
 $(H(x) - y)^2$: 가설값 - 실제값.
: 차이가 클수록 큰 값.

$$\text{Cost} = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$$\text{Cost}(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

Cost 가 최소화 하는 값을 기준으로 학습내용!

→ 이 때 쓰는 알고리즘: Gradient descent algorithm

Gradient descent algorithm

- Minimize cost function.
 - Ex) cost(W, b) 가 주어지면, 그 값이 최소화 하는 W, b 를 찾는다
 - 경사도를 따라 내려가는 것
 - Start는 아무 값에서 가능
 - W 값을 조금씩 나누면서 경사도를 계산.
 - 경사도 구하기 : 미분
- 간단하게 계산하고 싶으면 (6번)
- $$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$
- $$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2 \quad \text{계산 쉽게 하려고 } \frac{1}{2} \text{ 적용함}$$
- $$W := W - \alpha \frac{2}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)}) x^{(i)}$$
- 미분
- 정확한 경사 식
- $$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)}) x^{(i)}$$
- * 미분 계산 사이트 : Derivative Calculator

Convex function

- 경사가 방향 험태로 나와서 항상 위 Algorithm이 시각이 어디까지나 같은 형상을 찾을 수 있다

