머신러닝 인터뷰 준비

김태훈

Contents

[컴공]

 # f(x) = O(g(x))

iff there exists positive constant c, and k such that f(x) <= cg(x) for all x>= k. The value of c and k must be fixed for the function f and must not depend on x.
It means f(x) is less than some constant multiple of g(x) and a method used to find
asymptotic(점근선**) upper bound.**

f(x) = O(g(x)) if there are positive c and k that f(x) <= c * g(x) where x >= k. c and k should be fixed for function f and should not be depend on x

 # Linked List
Advantage:
- **Dynamic** data structure (can grow and prune)
- Insert and deletion is easy

Disadvantage:
- **Use more memory than array because of pointers**
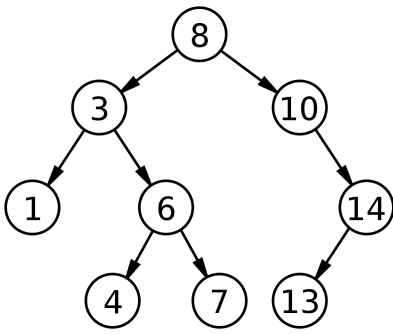- read in order from the beginning

 # Hash table, dealing with collision
Advantage:
- search O(1) (**worst O(n)**)
- Insert and deletion is easy

Disadvantage:
- collision
- worst addition O(n)
- **When key dict size is small**, space overhead of the next pointer is significant

 # Binary Search Tree

It is a tree data structure where left and right child nodes are bigger or lesser than parent node.
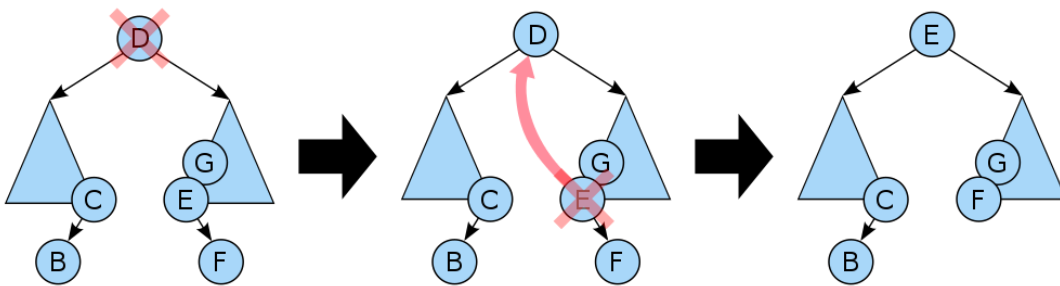
<u>sorting</u>: build tree for arr[:i] from i=1 to n

advantage: insert, search average O(log n), worst O(n)

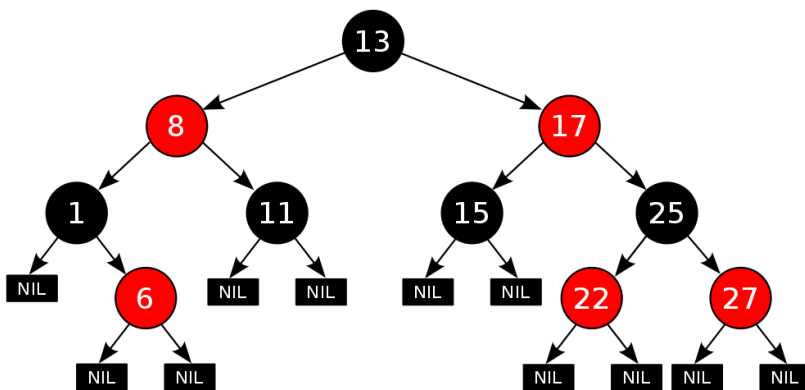**disadvantage**: worst insert (sort) O(n^2) if inputs are sorted as 1,2,3,4,5,.. or 5,4,3,2,1

implementation: array (index means position)

insertion: recursion (from top to bottom)

deletion: move one of leftmost or rightmost of child tree

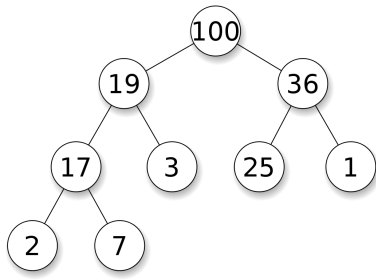# **Unbalanced tree** : **Red-Black tree** = BST + **color change + rotation**

Several constraints enforces "root <-> farthest leaf is no more than twice as long as root <-> nearest leaf "

<u>Heap</u> (**worst sort: O(log n) : because <u>insertion is always to the end</u>**)

It is a tree data structure where **parent node always have bigger or lesser value** than all child nodes.

sorting: **build** max heap and **pop** (**One of the best sorting methods**, **no quadratic worst-case scenarios**)

advantage: insert, remove average O(1), worst O(log n)

disadvantage: search O(n)

implementation: array (index means position)

add: **add the last,** swap if larger than parent (**upward**)

remove: move the last to the position, swap if smaller (down)

 # Heap vs Stack (in RAM memory)

a **special region of memory** that **stores temporary variables.**
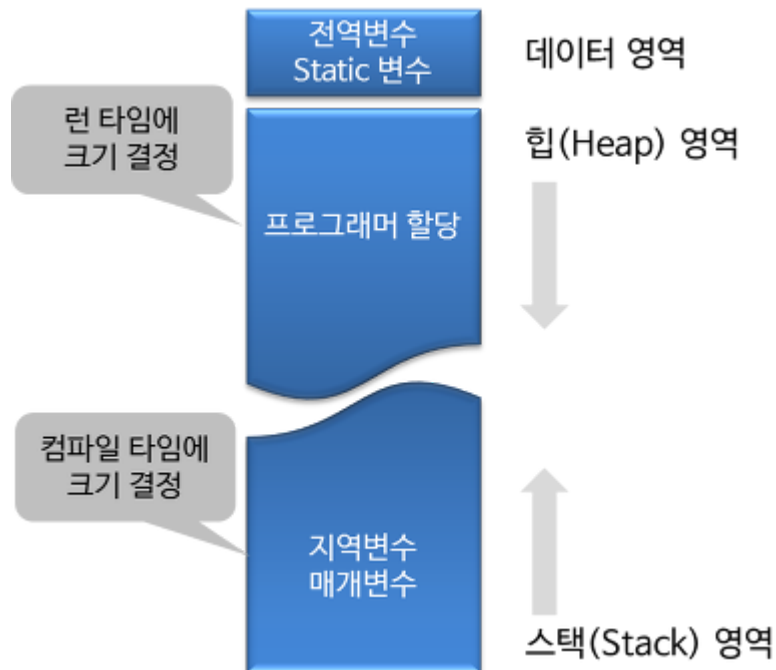
Data: **static, global** variable. Stay until the program ends.

Heap: **dynamic** memory allocation. Stay until free or terminal. **Size is decided during the runtime**.

void main() {

```
    int i = 10;
    int arr[i];
}
```

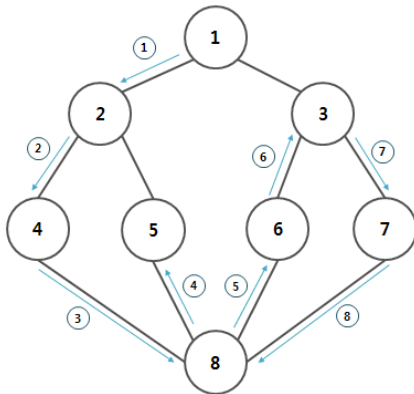**Stack**: **static** memory allocation. Only live in a scope of function. **Size is decided during the compiler time.**

Depth first search (stack, 왔던 곳 체크하기)

Depth First Search is an algorithm for **traversing** or search tree data structure.

implementation: **stack** and **mark visited.** Can get path from root to target by saving meta info.

BFS (queue, 왔던 곳 체크하기)

Breadth First Search is another algorithm for **traversing** or search tree data structure.

implementation: **queue** and **mark visited**. Can get path from root to target by saving meta info.

Memory leak

Type of a resource leak where a program **incorrectly manage memory allocation**

Memory no longer needed is not released. When object is **stored but cannot be accessed.**

Compiler

a software **transform code of one language into another,** mostly to high-level to low-level (machine code).

advantage: check syntax error. optimized to be **faster**.

disadvantage: compilation time.

Interpreter

a software directly execute instruction written in program language.

advantage: no compilation time. **partial execution**. find error before complete a program.

productivity. **disadvantage**: program is **not verified**. **runtime error**. slow execution.

JIT (Just In Time compilation)

A type of compilation. Involve compilation during execution rather than before execution.

Only required code will be converted into machine code.

ex. JVM (Java Virtual Machine)

GPL = General Public License
GNU = GNU's Not Unix!

OOP

1. **Polymorphism** : a single interface (function, class) support different types

   string = number.StringValue();

   string = date.StringValue();

2. **Encapsulation (public, private)** : **bundling of data with the methods** that operate on that data

   Prevent mistake by **restricting direct access**

3. **Inheritance :** when an object or class is based on another object

Refactoring

- Renaing
- **Encapsulate fields: getter and setter**
- **Extract class**
- Extract common codes
- Introduce **assertion**

Lambda function (= Anonymous function)

a function definition that **is not bound to an identifier**

Garbage collection

A way of automatic memory management.

Collector attempts to reclaim **gar   bage**, or **memory occupied by objects that are no longer in use**

Functional programming <->   **Procedural** programming

Treats **computation** as the **evaluation** of functions and **avoids changing-state and mutable data**

- Always returns **the same output for a given input**
- **Order** of evaluation is usually **undefined**
- Must be **stateless**. i.e. **No side effects**
- **Good** fit **for parallel** execution
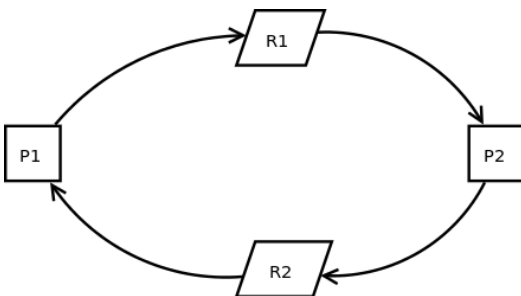- Increased **readability** and **maintainability**

Process : An instance of a computer program that is being executed.

Thread : the smallest sequence of programmed instructions which **share code and the values of variables**

Lock

A way that **limits on access to a resource** where there are many threads of execution

Deadlock

Each thread is waiting for the other thread to relinquish a lock, they both remain waiting forever.

Can prevent by **breaking the symmetry of the locks.**

Semaphore

A variable that is used for controlling access, by multiple processes, to a common resource.

Useful tool in the **prevention of race conditions**

Race condition

When two or more threads can access shared data and they try to change it at the same time.

Execution are **random**. **Root** 권한의 **setuid**가 걸려있는 프로그램으로 악용 가능
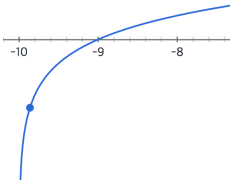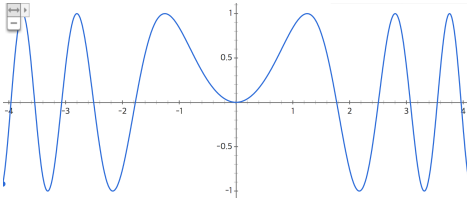
Context Switch

The **process** of **storing and restoring the state** (more specifically, the execution context) of a process or thread so that execution can be resumed from the same point at a later time.
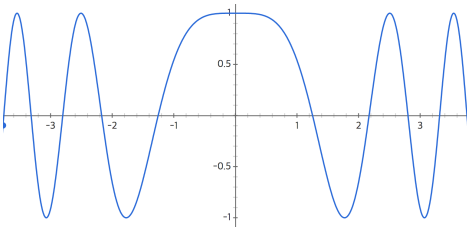
[수학]

Draw **log(x+10)**

Draw **sin(x^2)**

Draw **cos(x^2)**

Derivative of **ln x = 1/x**

Derivative **sin(x^2)** = 2x cos(x^2)
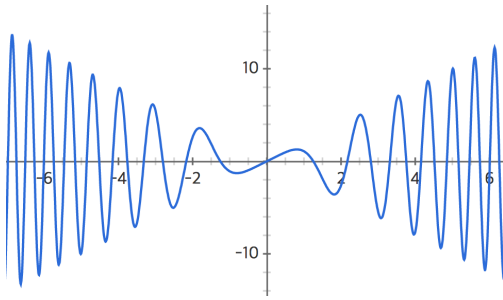
(sinx)^2 + (cosx)^2 = 1
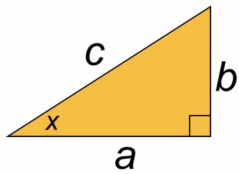
$$(\sin x)^2 + (\cos x)^2 = 1$$

Integration by parts

$$u(x)v(x) = \int u'(x)v(x)\,dx + \int u(x)v'(x)\,dx$$

$$\int u(x)v'(x)\,dx = u(x)v(x) - \int u'(x)v(x)\,dx$$

Integral log(x) (integration by parts)

$$\int \log(x)\,dx = \int \frac{dx}{dx}\log(x)\,dx = x\log(x) - \int x\frac{d}{dx}\log(x)\,dx$$

$$= x\log(x) - \int x \times \frac{1}{x}\,dx = x\log(x) - x + c$$

integral 1/x = ln |x| + C

integral 1/(1-x) = - ln |1 - x| + C

integral x sin x (integration by parts) = -x cos x + sin x + C
integral x cos x (integration by parts) = x sin x + cos x + C

sin x 미분 : cos x   cos x 미분 : - sin x
cos x 적분 : sin x   sin x 적분 : - cos x

(ε, δ)-definition of limit (epsilon–delta)

carpedm20

$$\lim_{x \to c} f(x) = L \iff (\forall \varepsilon > 0, \, \exists \, \delta > 0, \, \forall x \in D, \, 0 < |x - c| < \delta \implies |f(x) - L| < \varepsilon)$$

[[엡실론이 양 끝]]

Derivative (f: **R->R**) dx/dy = **derivative of x "with respect to" y**

the sensitivity to change of the function value

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

Partial derivative (f: **R^M->R**)

of a function of several variables f(x,y,...,z) **is its derivative with respect to one of those variables**

$$\frac{\partial}{\partial a_i} f(\mathbf{a}) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_{i-1}, a_i + h, a_{i+1}, \ldots, a_n) - f(a_1, \ldots, a_i, \ldots, a_n)}{h}$$

Gradient (f: **R^M->R**)

a multi-variable **generalization** of the derivative.

$$\nabla f(a) = \left(\frac{\partial f}{\partial x_1}(a), \ldots, \frac{\partial f}{\partial x_n}(a) \right).$$

Jacobbian (f: **R^M->R^N**)

Generalizes the gradient of a scalar-valued function of multiple variables

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_N} \end{pmatrix}$$

Hessian (f: R^M->R)

Second-order derivatives of a scalar-valued function

$$\nabla f = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} \\[1ex] \dfrac{\partial f}{\partial x_2} \\[1ex] \vdots \\[1ex] \dfrac{\partial f}{\partial x_n} \end{pmatrix} \quad \text{and} \quad \nabla^2 f = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\[2ex] \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\[2ex] \vdots & & \ddots & \vdots \\[2ex] \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Inner product x • y = <x, y>

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^{\mathrm{T}} \mathbf{b}$$

$$= \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$= a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$= \sum_{i=1}^{n} a_i b_i,$$

Outer product (벡터곱)

$$\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^{\mathrm{T}}$$

$$= \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix}$$

$$= \begin{pmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n \end{pmatrix}.$$

Orthogonal vectors $x \perp y$

1. inner product x • y = <x, y> = 0

Orthogonal matrix

1. $AA^T=E$ => $A^{-1}=A^T$

   (E: identity matrix)

Orthonormal

1. inner product x • y = <x, y>=0
2. ||x|| = 1, ||y|| = 1 (L2 norm)

Linearly independence

A set of vectors is said to be linearly **independent**, if one of the vectors in the set **can't** be defined as **a linear combination of the others**

Determinant (det(A) = 0 <-> A^-1 not exists)

A useful value that can be computed from the elements of a **square matrix.**

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= aei + bfg + cdh - ceg - bdi - afh.$$

$$\begin{matrix} a_{11} & a_{12} & a_{13} & \vdots & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & \vdots & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & \vdots & a_{31} & a_{32} \end{matrix}$$
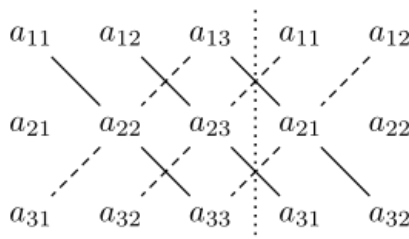
Properties of det

1. **Identity :** det(I) = 1
2. **Transpose**: det(A^T) = det(A)
3. **Inverse**: det(A^-1) = 1 / det(A)
4. **Multiplication**: det(AB) = det(A)det(B)

Eigenvector of a linear transformation

$A\mathbf{v} = \lambda\mathbf{v}.$ (v: eigenvector, lambda: eigenvalue)

a **non-zero** vector that **only changes by an scale (eigenvalue)**

(선형변환 A에 의한 변환 결과가 자기 자신의 상수배가 되는 0이 아닌 벡터)

How to calculate **Eigenvector?**

$$\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$$

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = \mathbf{0}$$

Eigenvectors u should not be zero-vector => **there should be no inverse matrix**

Therefore, $det(\mathbf{A} - \lambda\mathbf{I}) = 0$ **(characteristic equation)**

Eigendecomposition (Only **diagonalizable square matrix** can be factorized)

$$A\begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} \lambda_1 v_1 & \lambda_2 v_2 & \cdots & \lambda_n v_n \end{bmatrix}$$

$$= \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{bmatrix}$$

=> $A = P\Lambda P^{-1}$

조건 **: "A" should have n linearly independent eigenvectors**

왜 중요한가**? Easy to calculate det(A)**, A^2, A^-1, ...

$$det(A) = det(P\Lambda P^{-1})$$
$$= det(P)det(\Lambda)det(P)^{-1}$$
$$= det(\Lambda)$$
$$= \lambda_1\lambda_2\cdots\lambda_n$$

$<= det(inv(A)) = 1/det(A)$

Singular Value Decomposition (SVD)

generalization of eigendecomposition to m x n matrix

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

Matrix inversion

A is invertible if there exists B such that $\mathbf{A}\mathbf{B} = \mathbf{B}\mathbf{A} = \mathbf{I}_n$

B is uniquely determined by **A** and is called the **inverse** of **A**, denoted by $\mathbf{A}^{-1}$.

Pseudoinverse A^+

a generalization of the inverse matrix to m x n matrix

$$A = U\Sigma V^T$$

$$A^+ = V\Sigma^+ U^T$$

$$A = U\begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_s \\ & & & 0 \end{pmatrix}V^T \longrightarrow A^+ = V\begin{pmatrix} 1/\sigma_1 & & \\ & \ddots & \\ & & 1/\sigma_s & 0 \end{pmatrix}U^T$$

Use to compute a **'best fit' (least squares |Ax-b|=0) solution** to a system of linear equations

(대부분의 경우) 역행렬이 존재하는 않을 때 **(m x n matrix)** 해를 구하는 방법

$AX=B$ => det(A) != 0, $X=A^{-1}B$ det(A) = 0, $X=A^+B$

 # Norm of one thing

a **function** that assigns a **strictly positive** length or size to each vector:

- **Scalar** : p(av) = |a| p(v)
- **Sum** : p(u + v) ≤ p(u) + p(v)
- **Positive** : p(v) ≥ 0
- **Zero-vector** : If p(v) = 0 then v = 0

 # L1 norm

$$|\mathbf{x}|_1 = \sum_{r=1}^{n}|x_r|.$$

 # L2 norm (Euclidean norm)

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n}|x_k|^2},$$

A **_norm_** measures the size of a **_single_** thing.
A **_metric_** measures distances between **_pairs_** of things.

Metric of two thing (distance function)

A **function** that defines a distance **between each pair of elements of a set.**

- **Symmetry** : d(x, y) = d(y, x)
- **Sum** : d(x, z) ≤ p(x, y) + p(y, z)
- **Positive** : p(x, y) ≥ 0
- **Zero-equality** : If p(x, y) = 0 then **x = y**

Discrete metric

if x = y, **d(x,y) = 0.** Otherwise, **d(x,y) = 1.**

Euclidean distance

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Newton Method (뉴턴 메소드)

finding **repeatedly** better **approximations** to the **roots** (or where f(x) = 0) of a real-valued function.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$ until **|x_n+1 - x_n|** (change) is small

- Should be continuous and differentiable
- Can **only find one** among multiple answers (**dependent on initial x_0**)
- When gradient is zero

Root of a function
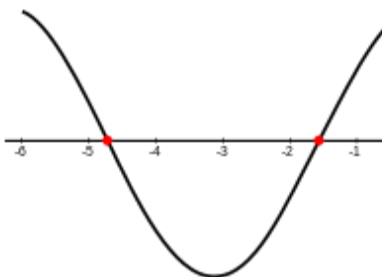
Cartesian coordinate system (x, y)
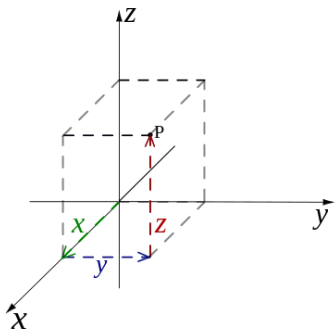
Polar coordinate system (r, φ)
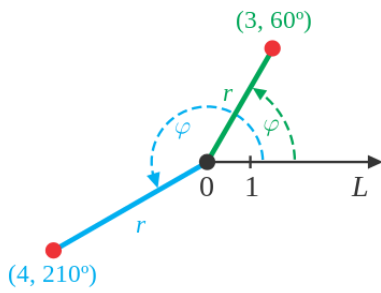
Cartesian **->** Polar

$$x = f(r,\theta) = r\cos\theta$$

$$y = g(r,\theta) = r\sin\theta$$

Find n prime numbers => 에라토스테네스의 체**(sieve of Eratosthenes)**

```
Input: an integer n > 1.

Let A be an array of Boolean values, indexed by integers 2 to n,
initially all set to true.

for i = 2, 3, 4, ..., not exceeding √n:
  if A[i] is true:
      for j = i², i²+i, i²+2i, i²+3i, ..., not exceeding n:
        A[j] := false.

Output: all i such that A[i] is true.
```

Metric (distance function)

Defines a distance between each pair of elements of a set

ex) Euclidean, Discrete, **Levenshtein** distance (**but no KL-divergence b/c not symmetric**)

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

NP (Nondeterministic Polynomial Time)
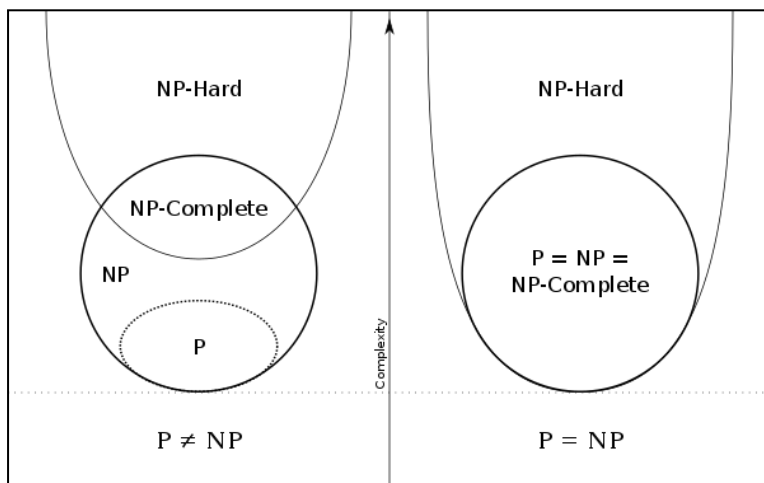
Sudoku is in NP (quickly checkable) but does not seem to be in P (quickly solvable)

NP is the set of **all decision problems** where the 'yes'-answers can be **verified** in polynomial time O(n^k) by a **deterministic** Turing machine, or **solvable** by a **non-deterministic** Turing machine

(polynomial time안에 그 solution이 맞는 solution인지 아닌지 구분할 수 있는지)

P (Nondeterministic Polynomial Time)

P is the set of **all decision problems** which can be solved in polynomial time by a **deterministic** Turing machine. Since they can be solved in polynomial time, they can also be verified in polynomial time. **P is a subset of NP.**

NP-hard (at least as hard **as the hardest problems in NP**)

H is NP-hard when for every problem L in NP, **there is a polynomial-time reduction from L (easy) to H (hard)**, that is **given a solution for L** we can **verify it is a solution for H in polynomial time.** **Solve any NP-hard problem in polynomial time** would solve all NP problem

NP-hard but not NP-complete :

given a program and its input, will it run forever? : **undecidable** because of infinite run

NP-complete

Both in NP and NP-hard. Any NP problem can be reduced into NP-complete.

Turing machine

a **mathematical model** of **computation** with tape
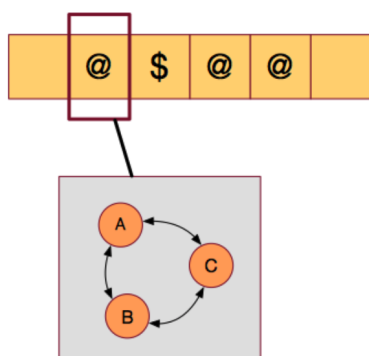
Current State	Read Symbol	Write Symbol	Next Position	Next State
A	@	@	->	A
A	$	$	->	B
B	@	@	<-	C
C	$	@	->	C
C	@	$	->	B

1. **Tape (memory)**
2. **Head**
3. **State register** : A or B or C
4. **Table of instruction** : action table

Turing completeness

A system (like programming language) is said to be **Turing complete if it can simulate any Turing machine.**

It could be used to **solve any computation problem.**

Finite-state machine

A mathematical model of computation without memory (tape)

- **State** : description of the status of a system
- **Transition** : a set of actions to be executed

[통계]

Probability

A likelihood of an event of random variable to be occurred. Sum of p for all possible disjoint events are 1.

Random variable

 a _function_ that maps outcomes to numerical quantities

a variable whose values are **numerical outcomes of a random phenomenon**. (ex. Coin front/back)

PDF (Probability Density Function)

A relative likelihood that the value of the random variable would equal that sample

(the **absolute likelihood** of continuous random variable on any particular value is **0. 0.0231을 뽑을 확률은 0)**

$$\Pr[a \le X \le b] = \int_a^b f_X(x)\, dx.$$

PDF condition :

1. f(x) > 0
2. $\int_{-\infty}^{\infty} f(x)dx = 1$

How PDF > 1 ?

Uniform distribution defined in 0 < x < 1/2

Variance

Expectation of the **squared deviation of a random variable from its mean**

$$\mathrm{Var}(X) = \mathrm{E}\big[(X - \mu)^2\big].$$

(**how far the values are spread out from mean**)

Covariance

$$Cov[X, Y] = E[(X - E[X])(Y - E[Y])]$$

Bernoulli distribution

$$\begin{cases} q = (1-p) & \text{for } k = 0 \\ p & \text{for } k = 1 \end{cases}$$

special case of the Binomial distribution where a single experiment/trial is conducted (n=1)

Binomial : n for # of trial, p

the discrete probability distribution of the # of successes in a sequence of n independent experiments

$$\boxed{Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}}$$

where $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$ (combination **w/o considering order**)

Multinomial: n for # of trials, p_1, p_2, .., p_k (sum p_j=1)

$$f(x_1, \ldots, x_k; n, p_1, \ldots, p_k) = \Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k)$$

$$= \begin{cases} \dfrac{n!}{x_1! \cdots x_k!} p_1^{x_1} \times \cdots \times p_k^{x_k}, & \text{when } \sum_{i=1}^{k} x_i = n \\ \\ 0 & \text{otherwise,} \end{cases}$$

Gaussian: mean, variance

PDF :

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Multivariate Normal distribution:

generalization of the one-dimension to higher dimension

$$f_{\mathbf{X}}(x_1, \ldots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}}$$

Moment

A quantitative measure of the shape of a set of points.

1. Mean : $\mu \equiv \mu_1 \equiv \mathrm{E}[X]$.

2. Variance : $\sigma \equiv \left(\mathrm{E}\left[(x - \mu)^2 \right] \right)^{\frac{1}{2}}.$

3. **Skewness** : $\mathrm{E}\left[\left(\dfrac{X - \mu}{\sigma} \right)^3 \right]$ **(a measure of the asymmetry)**

4. **Kurtosis**

i.i.d (Independent and Identically Distributed)

1. Independent : **not related to previous result**
2. Identically Distributed : **probability distribution is identical over time**

To simplify the underlying mathematics of many statistical methods (not Markov chain P(x_t|x_t-1))

Bayesian probability <-> Frequentist

Bayesian interpretation of probability is a degree-of-belief interpretation.

Take into account of **prior distribution** (can say there was life on Mars a billion years ago is **1/2**)

- **Advantage**:

  1. Prior : easy to **apply domain knowledge**

  2. **Uncertainty**: can predict **uncertainty**. Can detect **anomaly**

- **Disadvantage** :

  1. Choice of **prior**

  2. Computationally intensive : if it is required to **sample lots of variables**

Frequentist probability

limiting value of the number of successes in a sequence of trials

$$p = \lim_{n \to \infty} \frac{k}{n}$$

(p of life on Mars a billion years ago is **can't be assigned**)

Mean, Median and Mode

mode

median

50% | 50%
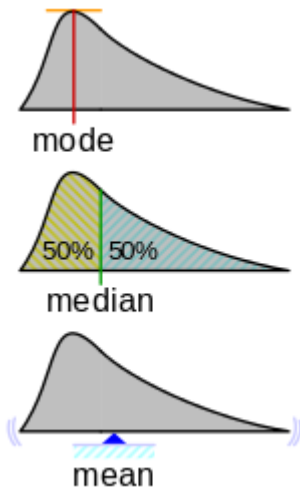
mean

1. **Mean : Expected value** in probability distribution. $\sum x P(x)$, $\int_{-\infty}^{\infty} x f(x)\, dx$

2. **Median :** the **value separating the higher half.**

$$\mathrm{P}(X \le m) = \mathrm{P}(X \ge m) = \int_{-\infty}^{m} f(x)\, dx = \frac{1}{2}.$$

3. **Mode : Most frequent value** in a data set

Joint probability distribution

Join probability P(X=x, Y=y,...) is probability that **each of X, Y, ... falls in any particular values.**

Conditional probability distribution

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

P(Y|X) is probability of Y **when X is known to be a particular value**

Independence

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

Two events are called independent if and only if **P(A∩B) = P(A)P(B)**

Marginal distribution

Marginal distribution of **subset of a collection of random variables** is the probability

distribution of the variables contained in the subset

carpedm20

$$\Pr(X = x) = \sum_y \Pr(X = x, Y = y) = \sum_y \Pr(X = x \mid Y = y) \Pr(Y = y),$$

$$p_X(x) = \int_y p_{X,Y}(x, y)\, dy = \int_y p_{X|Y}(x \mid y)\, p_Y(y)\, dy,$$

Bayes rule

- **Likelihood**: how probable is the **[evidence]** given the hypothesis
- **Prior**: how probable was **[hypothesis] before observing evidence**
- **Posterior**: how probable is **[hypothesis] given the observed evidence**
- **Priori (**Marginal**)**: how probable is the new **[evidence] under all hypothesis**

[머신러닝]

 # Maximum Likelihood Estimation

$$\hat{\theta} \in \{\arg\max_{\theta \in \Theta} \ell(\theta\,;x)\},$$

finding the parameter that **maximize the likelihood** of making the observations given the parameters

 # ((all) Batch) Gradient descent

Repeat until convergence {

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

}   w := w - lr * dL/dw

compute on ALL training set

 # Stochastic gradient descent

compute on a SAMPLE of training set. "stochastic approximation" of the **"true" cost gradient.**

- **Faster matrix operations** (computation)
- Parallelization
- Convergence is **slower than second-order gradient methods (Newton's method)**
  - But benefit of **computational efficient** is greater
  - Can **converge faste**r if **learning rate is adjusted**

 # Regression

Predict **continuous valued** output

(linear regression, **k-nearest neighbors**, nonlinear regression, polynomial regression)

 # Linear regression

$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i,$$

Pros**:** easy to compute

Cons: Sensitive to **Outliers, limited** to **Linear Relationships,** Data should be **Independent,**

Update: gradient descent with **least square (+ L2 regularization)**

$$\mathcal{L}(\beta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda \sum_j \beta_j^2$$

: least square error + l2-regularizer

Classification

Predict a **category (probability for each)** of new data

(logistic regression, decision tree, **k-nearest neighbors**, boosting ...)

Logistic regression (a **generalized** linear model)

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Pros**:** easy to compute

Cons**:** scalability, Data should be **Independent**

Update: gradient descent with (**sigmoid**) cross entropy loss.

$$l(z) = -\log\left(\prod_i \mathbb{P}(y_i|z_i)\right) = -\sum_i \log\left(\mathbb{P}(y_i|z_i)\right) = \sum_i -y_i z_i + \log(1 + e^{z_i})$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if } y = 1$$
$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if } y = 0$$

(maximize log likelihood)

Sigmoid cross entropy loss:

loss = z * -log(sigmoid(x)) + (1 - z) * -log(1 - sigmoid(x))

Ensemble : **use multiple** learning algorithms **to obtain better performance**

- **Bootstrap** aggregating (**bagging**) : ensemble **vote** with equal weight
- **Boosting :** a set of **weak** learners -> **strong** learner (**reduce bias**)
- **Stacking** : **train** **additional** merger model (**theoretically represent any of the ensemble techniques**)

Supervised

labeled training data

Unsupervised

Unlabeled training data

Semi-supervised

Only part of the training data is labeled

Why semi-supervised is important

Overcoming the problem of lack of data by **adding cheap and abundant unlabeled data**

Clustering

task of **grouping** where **objects in the same group (called a cluster)** are more similar

(k-means, hierarchical clustering)

k-means - **Centroid-based** clustering **(NP-hard)**

Pros**:** Simple, No training-time. Always **converge**

Cons**: can't distinguish all distribution (**평행한 두 데이터**), NP-hard (local minimum), wrong k**

Update**:**

1. **Random initialize centroids**
2. **Repeat {Assignment, Update}**

Finding k : **Elbow method**

- **Sum of the squared distance** between each **member of the cluster and its centroid**
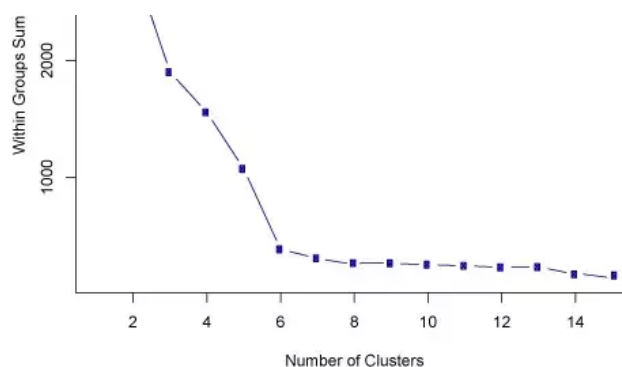
- **k=6** at which the **SSE decreases abruptly**

Hierarchical clustering
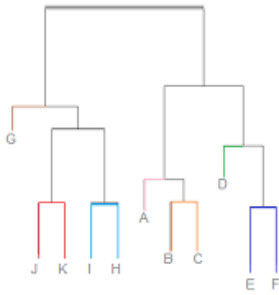
1. Agglomerative (**bottom up**) **: starts** with **N cluster**. **Merge** two successively.
2. Divisive (**top down**) : **starts** with **one cluster**. **Splits** successively.

Overfitting
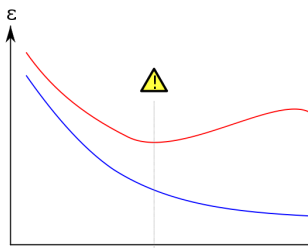
best model : where the **validation error is global minimum**.

test error increases while **training error decreases**

Cross validation

Validation method to **generalizability on an test set.**

Independent round training **prevent to be optimistically biased.**

- **One round** : split a data train/validate/test and train and validate
- **Repeat multiple rounds** with **different partitions** and average the validation results.

k-fold cross validation
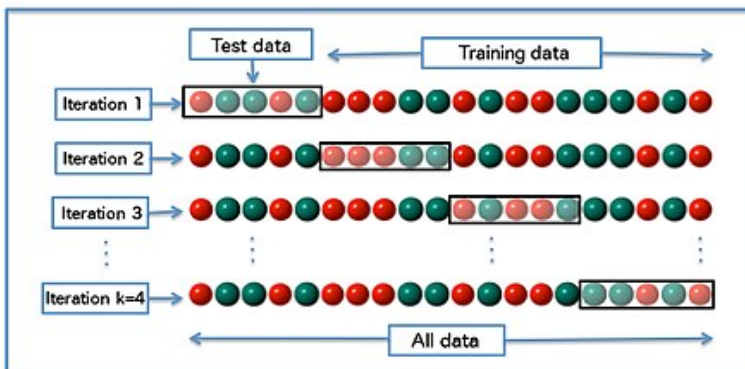
Partitioned into **_k_ equal sized subsamples**. Repeated k **times** (the **folds**) and the k results are averaged

Pros: **when test set is too small,** performance estimate is less sensitive to the partitioning of the data

Reinforcement learning

RL is modeled as a Markov Decision Process.

<reward, state, action, policy, state transition>

Markov Chain (Markov process)

$$P(X_k | X_{k-1}, X_{k-2}, \ldots, X_1, X_0) = P(X_k | X_{k-1})$$
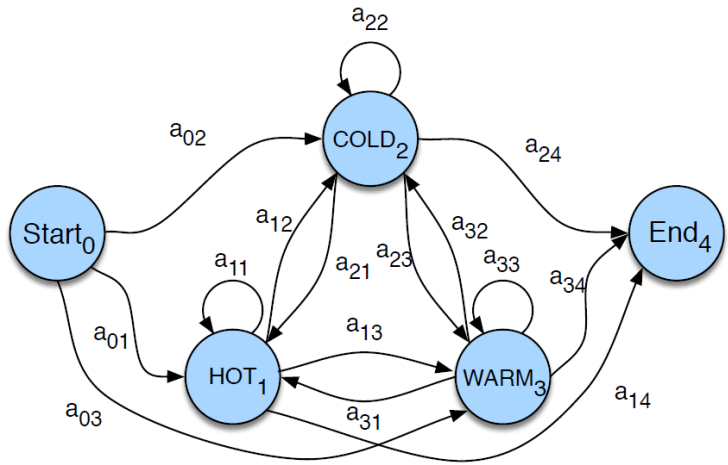
a stochastic model describing a sequence of possible events

=> 이전 **state**에만 **conditioning**하는 **sequential**한 **stochastic** 모델

Hidden Markov model
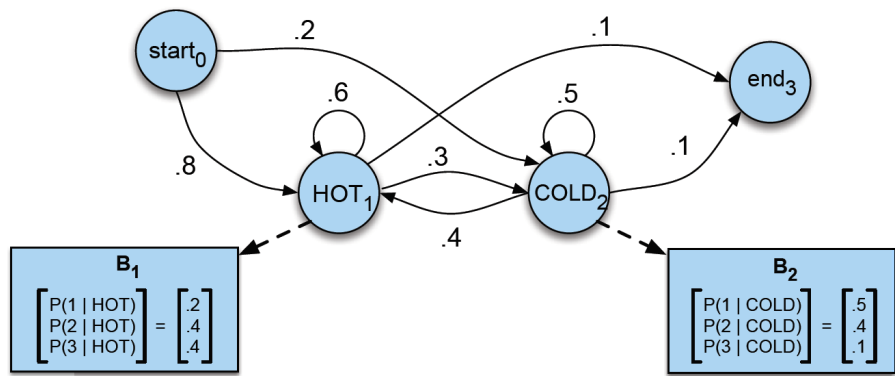
a **statistical Markov model** in which the system being modeled is **assumed to be a Markov process with unobserved (i.e. hidden) states.**

####################

Monte Carlo Method <-> Deterministic Algorithm

####################

```
Deterministic                    Non Deterministic

      ↓                                  ↓
      ↓                              ↙↙ ↓ ↘↘
  f(n) ↓                      accept—•  ↙ ↘ •
      ...                              ↓ ↘
                                        •— reject
      ↓
      ↓ ╱accept or              ↙
      •   reject            •— accept
```
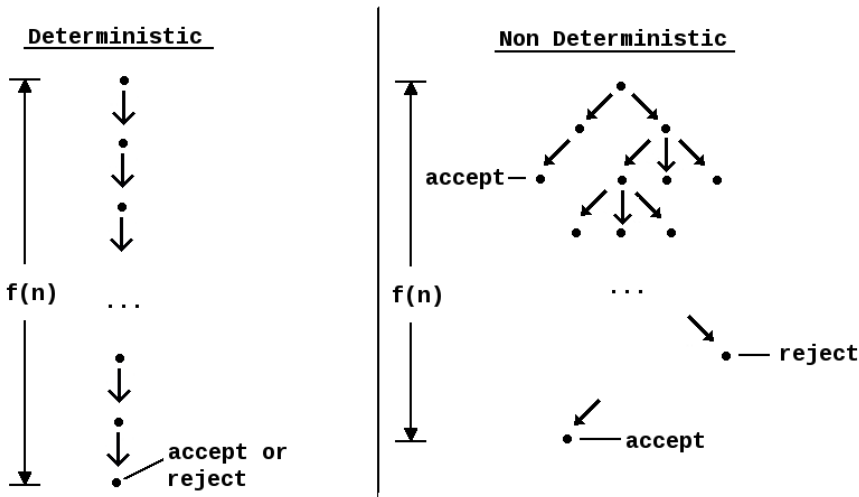
Use <u>repeated random sampling </u>to obtain numerical results (overall distribution)

################################

 # MCMC (Markov <u>chain</u> Monte <u>Carlo</u>)

################################

Posterior mean
$$\mathbb{E}[X] = \int_{\mathcal{X}} X\mathbb{P}(X|Y)dX$$
계산이 어렵기 때문에 **i.i.d.** 가정을 하고

$$\int_{\mathcal{X}} X\mathbb{P}(X|Y)dX \sim \sum_{i=1}^{N} \mathbb{P}(X_i|Y)$$
샘플링 **(approximation), Sampling algorithms** based on

constructing a Markov chain.

1. Gibbs Sampling
2. Metropolis–Hastings algorithm

1. Gibbs Sampling (\in MCMC)

$$p\left(x_j^{(i+1)}|x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \ldots, x_n^{(i)}\right)$$

A MCMC algorithm for obtaining a sequence of observations which are **approximated from a specified multivariate probability distribution**, when direct sampling is difficult.

쉽게 이해하자면 처음 하나의 초기 샘플 **X_0**을 랜덤하기 정한다음, 그 샘플에서 차원 **1**개씩 순차적으로 정하는 방법이다. n번째의 값을 정할 경우, 그 이외의 차원들은 고정된 값으로 본다. **MCMC는 N**차원의 점에서 바로 **N**차원의 점으로 모든 차원을 한번에 이동하면서 샘플링을 한 것인 반면, 깁스 샘플링은 한개의 차원을 제외한 나머지는 고정을 시킨다음**,** 한 차원 씩 샘플링을 해서 총 **N**번의 이동을 하고 난 다음 진짜 새로운 데이터를 샘플링하는 것이다**.**

장점 : **break the curse of dimensionality**

단점 : **doesn't allow the variables to evolve jointly**

2. Metropolis–Hastings algorithm (\in MCMC)

???

 # Markov Decision Process <S, A, R, T, discount>
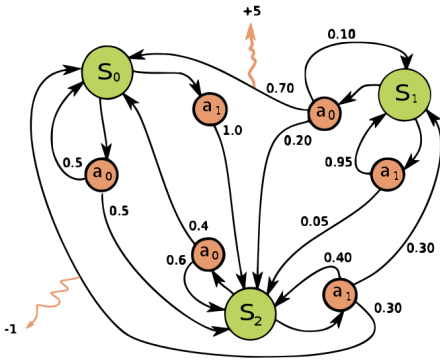
Framework for **modeling decision making** with Markov process.

Algorithm :

    1. Dynamic programming

        $\pi(s)$ and $V(s)$ is **updated alternatively**

$$\pi(s) := \arg\max_a \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V(s') \right) \right\}$$

$$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s') \right)$$

(policy가 있기 때문에 **max**가 없음**)**

    2. Value iteration

        $\pi(s)$ is calculated within $V(s)$

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V_i(s') \right) \right\}$$

(policy가 없기 때문에 **max)**

        If **assignment** became **equal**, it's **Bellman equation**

$$V^*(s) = \max_a \{ R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \}.$$

(R에 randomness만 뺀것)

    3. Policy iteration (Policy evaluation + Policy improvement)
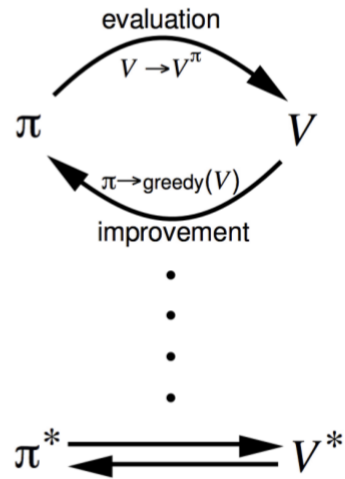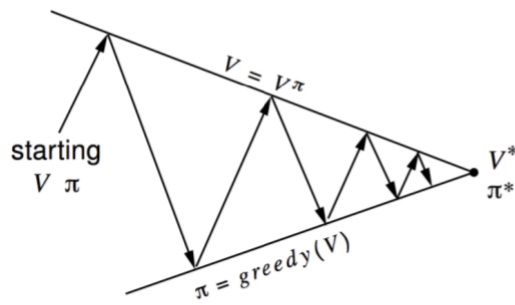
Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

1) Policy evaluation:

$$\forall s \in S : V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s')[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')]$$

2) Policy Improvement:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

Reinforcement Learning

Policy:

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

Return:

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

Value:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Q-value:

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Monte Carlo in RL

Monte-Carlo policy evaluation uses *empirical* *mean* return instead of *expected* return

Importance Sampling

Estimate the expectation of a different distribution

$$\mathbb{E}_{X \sim P}[f(X)] = \sum P(X) f(X)$$

$$= \sum Q(X) \frac{P(X)}{Q(X)} f(X)$$

$$= \mathbb{E}_{X \sim Q} \left[\frac{P(X)}{Q(X)} f(X) \right]$$

Objective function in RL

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$

$$= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \mathcal{R}_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) \mathcal{R}_{s,a}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) r]$$

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) r]$$

$$\nabla_\theta E_x[f(x)] = \nabla_\theta \sum_x p(x)f(x) \qquad \text{definition of expectation}$$

$$= \sum_x \nabla_\theta p(x)f(x) \qquad \text{swap sum and gradient}$$

$$= \sum_x p(x)\frac{\nabla_\theta p(x)}{p(x)}f(x) \qquad \text{both multiply and divide by } p(x)$$

$$= \sum_x p(x)\nabla_\theta \log p(x)f(x) \qquad \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z}\nabla_\theta z$$

$$= E_x[f(x)\nabla_\theta \log p(x)] \qquad \text{definition of expectation}$$
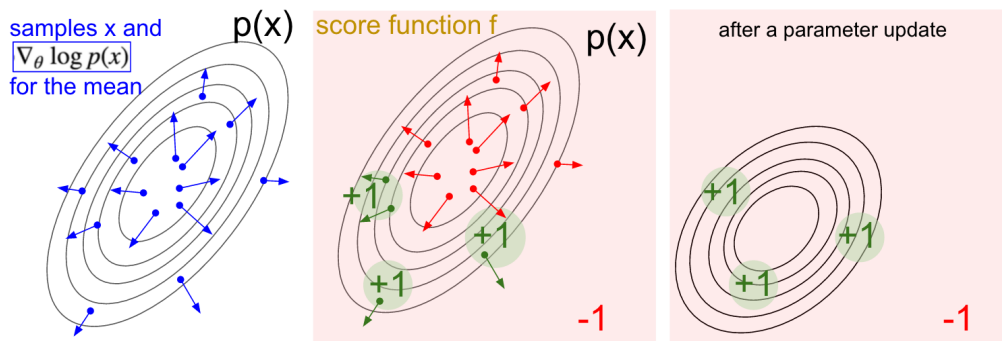
Policy Gradient

- **Cons : using Q is low variance but biased**

Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^\pi(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR},$ or $\frac{1}{1-\gamma}J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) \; Q^{\pi_\theta}(s, a)\right]$$

REINFORCE (Monte-Carlo Policy Gradient)

- **Return = unbiased (but high variance) estimation of Q(a, s)**

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return θ
end function

To deal with high variance of REINFORCE

1. Baseline (or Advantageous function)

Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) B(s)] = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) B(s)$$

$$= \sum_{s \in \mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(s, a)$$

$$= 0$$

- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s, a)$

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

Sum pi = 1

2. Actor critic

carpedm20

Q-learning (Value iteration update, off policy)
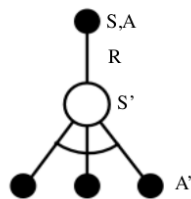
$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma \max_{a'} Q(S', a') - Q(S, A)\right)$$

+ epsilon greedy

a simple **value iteration update**, using **the weighted average** of the old value and the new information

$$\pi(S_{t+1}) = \underset{a'}{\mathrm{argmax}} \ Q(S_{t+1}, a')$$

epsilon greedy :

SARSA (on-policy)
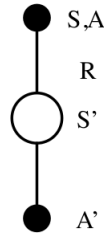
$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma Q(S', A') - Q(S, A)\right)$$

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'; A \leftarrow A';$
    until S is terminal

Figure 6.9: Sarsa: An on-policy TD control algorithm.

On-Policy Control With Sarsa
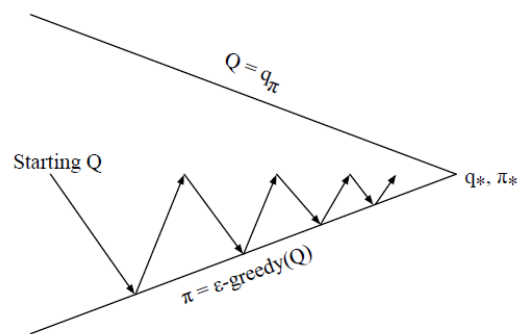
Every time-step:
Policy evaluation Sarsa, $Q \approx q_\pi$
Policy improvement ϵ-greedy policy improvement

Parametric

a **finite number** of **parameters** which **does not depend on data**

(linear regression, logistic regression)

Nonparametric learning

Models become more **complex** with an **increasing amount of data**.

(K-nearest neighbor, Decision Trees, Histogram)

Sequential data

1. Markov (decision) process : P **depends** only on the **state attained in "one-step" before event**

$$P(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \ldots, X_0 = x_0) = P(X_n = x_n | X_{n-1} = x_{n-1})$$

2. RNN

Neural Network

computing systems inspired by the **biological** neural networks composed of **neurons**

SVM loss

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

=>

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

score of the correct class to be **higher** than all other scores **by at least a margin of delta.**

(delta and regularizer coefficient works as same so only need one of them)

Softmax loss

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

1. Information theory : minimizing **cross-entropy** between **softmaxed probability** and true distribution

2. **Maximum Likelihood Estimation: minimizing the negative log likelihood** of the correct class

   f -= np.max(f)   ← **Stablize softmax**
   p = np.exp(f) / np.sum(np.exp(f))

Sigmoid loss

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$

Minimizing **cross-entropy** between **sigmoided probability** and true distribution

RNN

h_t = tanh(W [x_t; h_t-1] + b)

o = softmax(W h_t + b)

LSTM

f, i, c', o = f(W [x_t; h_t-1] + b).   f, i, o : sigmoid, c' : tanh

c_t = f * c_t-1 + i * c'

h_t = o * tanh(c_t)

1. 지금 **x**과 **h**를 바탕으로 할 것 예측
2. 이전 **c**를 지우고 씀 **=> c 끝**
3. 다음 **x**와의 계산에 중요한 것만 남김 **=> h끝**

Vanishing & Exploding gradient

- 0<= 0 d sigmoid x <= **1/4**
- 0<= 0 d tanh x <= 1

Gradient contributions from "far away" steps become zero by chain rule

1. Proper **initialization**
2. **ReLU** instead of tanh or sigmoid
3. LSTM : if f gate is 1 : no vanishing gradient => **long term dependencies** can be learned

Backpropagation

1. Forward pass
2. Compute error
3. Backward pass
4. Repeat

```
    initialize network weights (often small random values)
    do
       forEach training example named ex
           prediction = neural-net-output(network, ex)   // forward pass
           actual = teacher-output(ex)
           compute error (prediction - actual) at the output units
           compute Δw_h for all weights from hidden layer to output layer   // backward pass
           compute Δw_i for all weights from input layer to hidden layer    // backward pass
 continued
           update network weights // input layer not modified by error estimate
    until all examples classified correctly or another stopping criterion satisfied
    return the network
```

Backpropagation Through Time

A weight is updated with sum of gradient for each time step (because **weights are shared**)

Black box model vs White box model

Advantage:

- **Performance**

- **End-to-End optimization**

Disadvantage:

- **Interpretability**

    - Can this network tell a Husky from a Poodle?

    - Which objects are easy to classify for the algorithm, which are difficult?

    - Which part of a dog is the most important for being able to classify it correctly? The tail or the foot?

    - If I photoshop a cats head on a dog, what happens, and **why**?

Curse of dimensionality

With classical **non-parametric** learning algorithms (e.g. **nearest-neighbor, SVM**, etc.), the learner will **need to see at least one example for each** of these many configurations.

# of data is exponentially increasing so solution in **low dim can't be applied to higher dim**

VAE vs GAN

VAE: **maximum likelihood**

- **assign high p** to any point that **occurs frequently (also blurry images).**

- **where the latent is important**

GAN: GAN loss {D_real + D_fake} + {G_fake} is **highly dependent on how D is optimal**

- **avoid assigning high p** to points that the **discriminator recognizes as fake** (such as **blurry images**)
- GAN convergence does not guarantee the performance convergence
- Mode collapsing
    1. Solving **MiniMax** problem:
       But train D, G alternatively makes NN **undistinguishable between minimax and maximin**
       In maximin, G could just generate **single image that can fake the D (non optimal)**

$$G^* = \min_G \max_D V(G, D).$$

<->

$$G^* = \max_D \min_G V(G, D).$$

    2. No quantitative loss:
       Fake or not is subjective based on D (unlike autoencoder loss in VAE)
       Even when G does not cover all modes, it's **still fine** with GAN loss

Blurry image : VAE high p, GAN low p

 # Softmax = exp(x_i) / sum_j exp(x_j) 가 불안전한 이유
0 or infinite division. Can avoid with - max(x_i) to all terms.

 # REINFORCE = E[R d log(p)]를 안정되게 하려면

$$\mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) r \right]$$

- Reducing Variance Using a Baseline : E[(R - mu) d log(p)]. Gradient 가 같기 때문에 update 가 똑같음.
- The advantage function has lower variance since the baseline **compensates for the variance introduced by being in different states.**

 # Policy-Based RL
장점 : learn **stochastic** policies, **continuous** action spaces
단점 : Evaluating a policy is typically inefficient and **high variance**
(may receive very different rewards for similar or even identical behavior)

[Problem solving]

 # Anagram O(2n) = O(n)

```python
def anagram(a, b):
    if len(a) != len(b):
        return False

    counter = {}
    for c in a:
        c = c.lower()
        if c not in counter:
            counter[c] = 1
        else:
            counter[c] += 1
    for c in b:
        c = c.lower()

        if c not in counter:
            return False

        counter[c] -= 1
        if counter[c] < 0:
            return False
    return True


print(anagram('Eleven plus two', 'Twelve plus onn'))
```

 # Max contiguous sum

```python
def max_cont_sum(array):
    max_sum = -9999
    max_sum_sofar = array[0]

    for num in array[1:]:
```

```python
        max_sum_sofar = max(num, max_sum_sofar+num)
        max_sum = max(max_sum, max_sum_sofar)
    return max_sum


print(max_cont_sum([1,2,3,-100,1,2,1]))
```

Kth Largest Element in Array

```python
def find_k(array, k):
    for k_i in range(k):
        for idx in range(k_i+1, len(array)):
            if array[idx] >= array[k_i]:
                array[idx], array[k_i] = array[k_i], array[idx]
        print(array)
    return array[k]


print(find_k([0,-1,5,4,2,1,3], 3))
```

Powerset (permutation)

```python
# f([3, 1, 5])
# = { [3] + x in f([1, 5]) }
#   + { x in f([1, 5]) }
def powerset(x):
    answers = []
    if len(x) <= 1:
        return [x, []]
    else:
        for item in powerset(x[1:]):
            answers.append([x[0]] + item) # [1] + ([2], [])
            answers.append(item) # [2], []
    return answers
```

reverse string and omit multiple space

```python
def reverse(string):
    split_string = []
    tmp = ""
    for char in string:
        if char == " ":
```

```python
        if tmp != "":
            split_string.append(tmp)
            tmp = ""
        else:
            tmp += char
    split_string.append(tmp)
    return " ".join(reversed(split_string))


def reverse(string):
    return " ".join([word[::-1] for word in string[::-1].split()])


def reverse(string):
    return " ".join(reversed(string.split()))


# Return pairs where sum is k
def array_pair_sum(array, k): # O(n^2)
    answers = []
    array.sort() # O(n logn)
    for idx, num in enumerate(array):
        if k - num in array[idx+1:]: # O(n^2)
            answers.append([num, k-num])
    return answers


def array_pair_sum(array, k): # O(nlogn)
    answers = []
    array.sort() # O(n logn)
    left, right = 0, len(array) - 1
    while left < right:
        tmp = array[left] + array[right]
        if tmp == k:
            answers.append([array[left], array[right]])
        elif tmp < k:
            left += 1
        else:
            right += 1
    return answers
```

```python
def array_pair_sum(array, k): # O(n)
    answers = []
    table = {}

    for idx, num in enumerate(array):
        if k - num in table:
            answers.append([num, k-num])
        else:
            table[num] = True
    return answers
```

check combined two string
```python
def check(a, b, merge):
    if len(a) + len(b) != len(merge):
        return False

    if not a or not b or not merge:
        if a + b == merge:
            return True
        else:
            return False

    if a[0] != merge[0] and b[0] != merge[0]:
        return False
    elif a[0] == merge[0] and check(a[1:], b, merge[1:]):
        return True
    elif b[0] == merge[0] and check(a, b[1:], merge[1:]):
        return True

    return False


print(check("abc", "def", "dabcef"))
```

Check binary tree
```python
def isBT(tree, min_val=-999 max_val=999):
    if tree is None:
        return True
```

```python
        if not min_val <= tree.val <= max_val:
            return False

        return isBT(tree.left, min_val, tree.val) and \
            isBT(tree.right, tree.val, max_Val)


# Convert array in-place using constant extra space.
def get_index(idx, N):
    return (idx % 3) * N + idx // 3


def convert_array(array):
    N = len(array) // 3

    for idx in range(len(array)):
        swap_idx = get_index(idx, N)

        while swap_idx < idx:
            swap_idx = get_index(swap_idx, N)

        array[idx], array[swap_idx] = array[swap_idx], array[idx]
    return array


array = list("1234abcdzxyw")
print(convert_array(array))



#############


                                        # Quicksort
def partition(array, start, end):
    if start >= end:
        return start
    else:
        pivot = start
        for idx in range(start+1, end+1):
            if array[idx] <= array[start]:
                pivot += 1
                array[idx], array[pivot] = array[pivot], array[idx]
```

```python
        array[start], array[pivot] = array[pivot], array[start]
        print(array[start:pivot], array[pivot], array[pivot+1:end+1])
        return pivot


def quicksort(array):
    def _quicksort(array, start, end):
        if start >= end:
            return
        else:
            pivot = partition(array, start, end)
            _quicksort(array, start, pivot-1)
            _quicksort(array, pivot+1, end)
    return _quicksort(array, 0, len(array) - 1)
```

Sort

```python
def mergesort(array):
    less = []
    equal = []
    greater = []

    if len(array) > 1:
        pivot = array[0]
        for x in array:
            if x < pivot:
                less.append(x)
            elif x == pivot:
                equal.append(x)
            else:
                greater.append(x)

        return mergesort(less) + equal + mergesort(greater)
    else:
        return array
```

Binary search

```python
import math

def binary_search(array, find):
        start = 0
        end = len(array) - 1

        while True:
                idx = int(math.floor((start + end)/2.0))
                if  find == array[idx]:
                        return idx
```

```python
                    elif find < array[idx]:
                            end = idx  - 1
                    elif find > array[idx]:
                            start = idx + 1
                    if start >= end:
                            if find == array[start]:
                                    return start
                            return False


def fibonacci(num):
        if num == 0:
                return 0
        elif num == 1:
                return 1
        else:
                return fibonaci(num-1) + fibonaci(num - 2)


def fib(n, cache={}):
    if n == 1:
        cache[1] = 1
        return 1
    elif n == 2:
        cache[2] = 1
        return 1
    else:
        if n not in cache:
            cache[n] = fib(n-1, cache) + fib(n-2, cache)
        return cache[n]



import numpy as np

class NQueeun(object):
    def __init__(self, size):
        self.size = size
        self.rows = []

    def place(self, start_row=0):
        if len(self.rows) == self.size:
            print(self.rows)
            return self.rows
        else:
            for row in range(start_row, self.size):
                if self.is_safe(row, len(self.rows)):
                    self.rows.append(row)
                    return self.place()
            else:
                last_row = self.rows.pop()
```

```python
            return self.place(last_row + 1)

    def is_safe(self, row, col):
        for thread_col, thread_row in enumerate(self.rows):
            if row - thread_row + col - thread_col == 0:
                return False
            elif row - thread_row == col - thread_col:
                return False
            elif row == thread_row or col == thread_col:
                return False
        return True

    def print(self):
        board = np.array([[' '] * n] * n)
        for q in self.rows:
            board[self.rows.index(q), q] = 'Q'
        print(board)

n=9
queen = NQueeun(n)
queen.place(5)
queen.print()




import heapq
import numpy as np

G = {1: {2:10, 3:12}, 2:{3:1, 4:5}, 3:{4:2}, 4:{}}

def dijkstra(G, start):
        d = {}
        prev = {}

        for v in G.keys():
                d[v] = np.inf
                prev[v] = None

        d[start] = 0
        s = []
        q = []
        for v in G.keys():
                heapq.heappush(q, [np.inf, v])

        while len(q) != 0:
                u = heapq.heappop(q)
                s.append(u)
```

```python
            for v in G[u[1]].keys():
                if d[v] > d[u[1]] + G[u[1]][v]:
                    d[v] = d[u[1]] + G[u[1]][v]
                    prev[v] = u[1]

    print d

dijkstra(G, 1)

import unittest

class DijkstraTest(unittest.TestCase):
    def test_dijkstra(self):
        self.assertEqual(1,1)

if __name__ == '__main__':
    unittest.main()
```

[퀴즈]

[컴공]

f(x) = O(g(x))

링크드리스트 장단점

해시테이블 장단점 콜리젼나면 어떻게 해결할래

바이너리 서치 트리 정의 장단점, 구현은 어떻게 하냐

Heap vs Stack

depth first search : **stack +** 왔던곳 체크

메모리릭 : **is not released, not accessible**

compiler, interpreter, jit 프로그래밍의 정의

compiler와 interpreter의 장단점

Encapsulation (public, private) : **isolating implementation details, prevent mistake**

==============> Refactoring

Lambda function

Garbage collection

Functional programming : stateless

GPL

[수학]

dx/dy = **derivative of x "with respect to" y**

Integration by parts

$$\int u\,dv = uv - \int v\,du.$$

log(x+10) 그려라

sin(x^2) 그려라

sin(x^2) 미분해라

log(x) 적분

1/x 적분

==============> limit의 정의

1. Derivative : **sensitivity to change of value**

2. **Partial derivative**

3. Gradient

4. **Hessian**

5. **Jacobian**

Find n prime number

metric의 정의와 예시 (Euclidean 말하고 그거 식 씀)

np-complete

Turing machine : **tape (memory), header,** state, transition. **mathematical model of computation**

==============> Turing completeness : A system that can simulate any Turing machine (can solve any computation problem)

Finite state machine : state, transition

[통계]

binomial, multinomial, gaussian

==============> random variable이 뭔지 : $\textbf{function X: }\Omega\rightarrow\mathbb{R}$

- **pdf** : **A relative likelihood that the value of the random variable**
- pdf의 조건, pdf가 1 이상을 가질 수 있는 이유 (ex. **uniform dist**)

Variance

Covariance :
$$Cov[X, Y] = E[(X - E[X])(Y - E[Y])]$$

IID : **previous results are not related**, **distribution is identical over time**

==============> Bayesian statistics :

interpretation of p is **degree-of-belief interpretation**. **Takes into account of the prior distribution**

- Joint probability : **p when all variable falls into specific value**
- Conditional probability : **p of Y when X is known**

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

- Independence

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

- **Marginal** : In **distribution with collections of variable**, **p of the variables contained in the subset**

1st, 2nd, 3rd, 4th moment : **measure of of the shape** of a set of points.

==============> Mean **=>** variance **=> skewness => kurtosis**

==============> Bayes rule

- **Likelihood**: how probable is the **[evidence]** given the hypothesis
- **Prior**: how probable was **[hypothesis] before observing evidence**
- **Posterior**: how probable is hypothesis given the observed evidence
- **Priori**: how probable is the new **[evidence] under all hypothesis**

Determinant: identity, transpose, inverse, multiplication

Eigenvector: In a **linear transformation, non-zero** vector that **only changes by an scale**

Eigendecomposition: should have n linearly independent eigenvectors

Singular Value Decomposition

==============> Matrix inversion

Pseudoinverse : generalization of inverse matrix. Used in finding **least squares |Ax-b|=0 solution**

$$A = U\Sigma V^T$$

$$A^+ = V\Sigma^+ U^T$$

$$A = U \begin{pmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_s & \\ & & & 0 \end{pmatrix} V^T \longrightarrow A^+ = V \begin{pmatrix} 1/\sigma_1 & & & \\ & \ddots & & \\ & & 1/\sigma_s & 0 \end{pmatrix} U^T$$

$AX = B$ => det(A) != 0, $X = A^{-1}B$ det(A) = 0, $X = A^+B$

linearly independence : a linear combination of the others

Orthogonal vector & matrix

===============> Norm : **length** or **size** : **scalar**, sum, positive, **zero-vector**

===============> Metric : **distance** : **symmetry**, sum, positive, **zero-equality**

Newton Method : approximation to find root of a function. x := x - f/f'

[머신러닝]

===============> Maximum Likelihood Estimation

L2 loss, l2 regularizer, l2 norm

$$S = \sum_{i=0}^{n}(y_i - h(x_i))^2 \vdash \boxed{\lambda \sum_{i=1}^{k} w_i^2} \quad |\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2} \, ,$$

nn 정의

gradient descent 의 정의와 식

stochastic gradient descent 정의 그리고 장점

regression의 정의와 예시 알고리즘(linear regression 말함)

linear regression의 정의와 장점 단점, 업데이트 방법, optimal을 찾는법 (least square)

classification의 정의와 예시 (logistic regression 말함)

logistic regression의 정의와 장단점

supervised, unsupervised, semi-supervised 정의

semi-supervised가 중요한 이유

clustering 정의

clustering 예시 (k means 말함)

kmeans가 뭔지 어떻게 업데이트하는지 설명, k는 어떻게 설정하는지

kmeans의 장점과 단점(단점은 구분할 수 있는 데이터 분포가 정해져있음, 클래스 2개가
일직선으로 떨어져 있으면 kmeans로 구분 못함)

cross validation이 뭔지 (k-fold cross validation이 말하고 그걸 쓰는 이유 말함 데이터가 limit 하니까)

reinforcement learning의 정의 (reward, state, action, policy, state transition)

rl 업데이트 방법 (q-function)

q-learning이 off-policy인지 on-policy인지

non bayesian과 bayesian의 차이

non parametric learnin의 정의와 예시 (원래는 binomial이라고 했는데 이거 틀림 왜냐면 p가
필요하기 때문에. 정답은 histogram). sequential data를 표현하는 모델의 예시 (hidden markov,
bayes, rnn)

rnn의 정의와 식

rnn의 장점과 단점 (vanishing gradient) 해결방법 lstm.

rnn과 hidden markov의 장단점 비교 (rnn은 파라미터 쉐어해서 표현력이 떨어지지만 end-to-end
모델이라 gradeint descent해서 optimal 찾긴 찾음)

neural network 업데이트 방법 (backpropagation)

rnn의 업데이트 방법 (time delayed backpropagation)

big one black box 모델과 neural net같은 모델 (이 두 모델을 정의하는 term이 있는데 기억이
안남)을 비교했을때 nn같은 module화된 모델의 장단점 (장점은 모델 expresivity 가 훨 좋음, 단점은
각각의 모듈과 그 모듈들의 condition을 배워야 해서 오래 걸림)

curse of dimensionality

bayesian setting을 왜 잘 안쓰는지 (distribution을 정하는게 tricky하고 각각을 계산하는게 오래걸림)

prior (데이터) 샘플링 방법 : **Gibbs Sampling (\in MCMC)**

히든 마콥 모델이 뭔지 설명

VAE vs GAN

[<u>알고리즘</u>]

Permutation

N queens

Kth Largest Element in Array

2018.01.31 update

How computer represent floating number : 0.2341234123e-3

How to deal with unbalanced tree

Explain dynamic programming and give me an example

Explain quicksort. Can we do better than n log n. **Radix sort**

Integral of log x

Difference between process and thread

How to find A-1

Hessian, Jacobian and when these are used in practice

Jacobian is composed of column vector of gradient

What is positive definite

Why newton method has such form (x := x - f / f')

2018.02.05 update

What is central limit theorem

The law of big number

Who can you derive $mu(x) = E[x^2] - E[X]^2$

$$\begin{aligned}
\text{Var}(X) &= \text{E}\big[(X - \text{E}(X))^2\big] \\
&= \text{E}\big[X^2 - 2X\,\text{E}(X) + [\text{E}(X)]^2\big] \\
&= \text{E}(X^2) - \text{E}[2X\,\text{E}(X)] + [\text{E}(X)]^2 \\
&= \text{E}(X^2) - 2\,\text{E}(X)\,\text{E}(X) + [\text{E}(X)]^2 \\
&= \text{E}(X^2) - 2[\text{E}(X)]^2 + [\text{E}(X)]^2 \\
&= \text{E}(X^2) - [\text{E}(X)]^2
\end{aligned}$$

How to sample from arbitrary continuous random variable with uniform distribution: A general method is the inverse transform sampling method, which uses the cumulative distribution function (CDF) of the target random variable

Recommended Lectures

- [CS231n: Convolutional Neural Networks for Visual Recognition](#)
- [CS294-158 Deep Unsupervised Learning](#)
- [CS224n: Natural Language Processing with Deep Learning](#)
- [CS294-112 Deep Reinforcement Learning](#)