Interview Bible

A new round just began.

Outline

- Sorting
- Data structure + popular questions
- Machine Learning
- SoftDev interview questions

Sorting Algorithms

Let’s do it.

Quick Sort

Quicksort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays.
The steps are:

Pick an element, called a *pivot*, from the array(usually the last element).

Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the *partition* operation.

Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Average performance	$O(n \log n)$
Worst-case performance	$O(n^2)$
Best-case performance	$O(n \log n)$ (simple partition) or $O(n)$ (three-way partition and equal keys)

Insertion Sort

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.

For each iteration, insertion sort removes one element from the input data(usually from the first element), then finds the location it belongs within the sorted list(like bubble sort), and inserts it there.

It repeats until no input elements remain.

Worst-case performance	$O(n^2)$ comparisons, swaps
Best-case performance	$O(n)$ comparisons, $O(1)$ swaps
Average performance	$O(n^2)$ comparisons, swaps

Bubble Sort

Bubble sort repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed.

Worst-case performance	$O(n^2)$
Best-case performance	$O(n)$
Average performance	$O(n^2)$

Heap Sort (code detail not clear)

The heapsort algorithm can be divided into two parts.

In the first step, a heap is built out of the data. The heap is often placed in an array with the layout of a complete binary tree. The complete binary tree maps the binary tree structure into the array indices; each array index represents a node; the index of the node's parent, left child branch, or right child branch are simple expressions. For a zero-based array, the root node is stored at index 0; if i is the index of the current node, then

iParent(i) = floor((i-1) / 2) where floor functions map a real number to the smallest leading integer.
iLeftChild(i) = 2*i + 1
iRightChild(i) = 2*i + 2

In the second step, a sorted array is created by repeatedly removing the largest element from the heap (the root of the heap), and inserting it into the array. The heap is updated after each removal to maintain the heap. Once all objects have been removed from the heap, the result is a sorted array.

Worst-case performance	$O(n \log n)$
Best-case performance	$\Omega(n), O(n \log n)$ [1]
Average performance	$O(n \log n)$

Binary Tree Sort

Adding one item to a binary search tree is on average an O(log n) process (in big O notation)
so adding n items is an O(n log n) process.

Worst-case performance	$O(n^2)$ (unbalanced) $O(n \log n)$ (balanced)
Best-case performance	$O(n \log n)$[citation needed]
Average performance	$O(n \log n)$

Useful link : http://wuchong.me/blog/2014/02/09/algorithm-sort-summary/

Data Structure & Popular Questions

Let’s do it, again.

Array - Questions

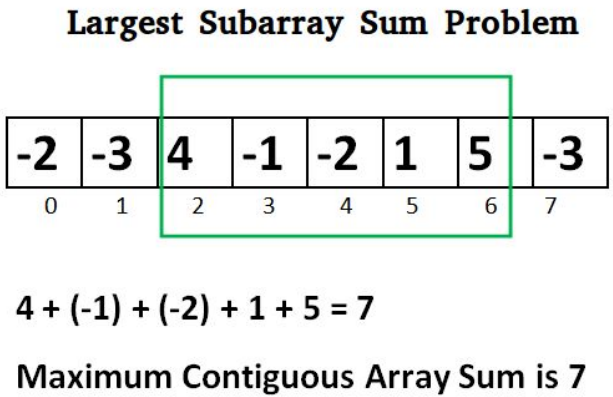
A quicksort...O(nlogn)

Rotation

Find Majority Element

hard

XOR ^ 按位异或/Find the Missing Number



Quick Overview

Linear Data Structure

Binary Tree, BST, Heep, Hash

Graph

Search (AI Class)

Linear Data Structure

Array

Linked List

Stack

Queue

Array

Accessing Time: O(1)

Search Time: O(n)

Insertion Time: O(n) [The worst case occurs when insertion happens at the Beginning of an array and requires shifting all of the elements]

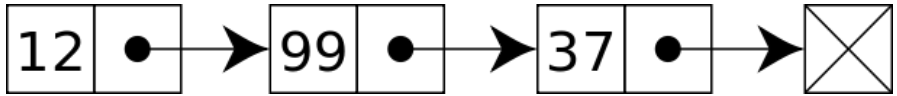
Deletion Time: O(n) [The worst case occurs when deletion happens at the Beginning of an array and requires shifting all of the elements]

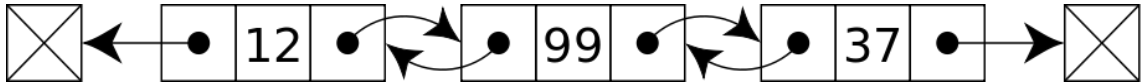
Linked List

Singly Linked List: 1->2->3->4->NULL

Doubly Linked List: NULL<-1<->2<->3->NULL

Circular Linked List: 1->2->3->1

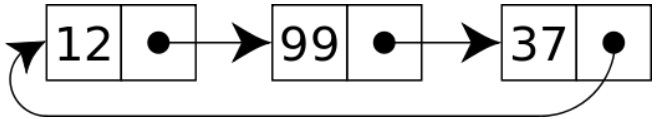




Accessing: O(n)

Search: O(n)

Insertion of an Element : O(1) (without ordering)

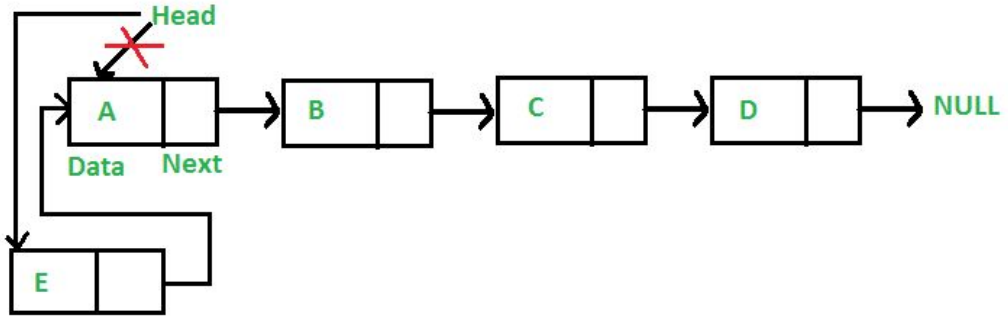


Deletion of an Element : O(1)

Linked List - Questions

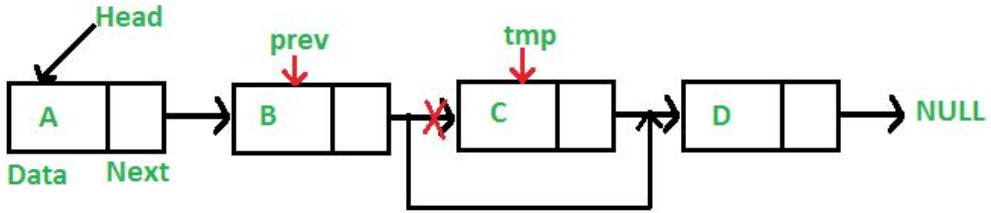
Insertion

1. Allocate new node
2. Put in data
3. NewNode.next = head
4. head = NewNode



Delete

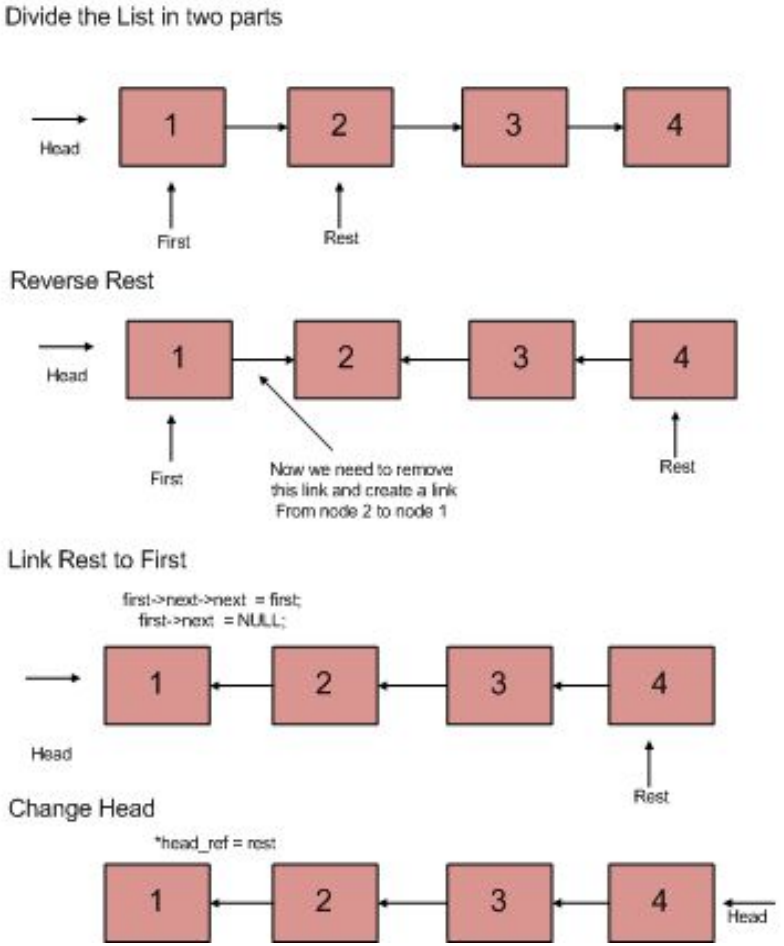
1. previous <-post
2. free memory



Linked List - Questions

Reverse

[Detect and Remove Loop]

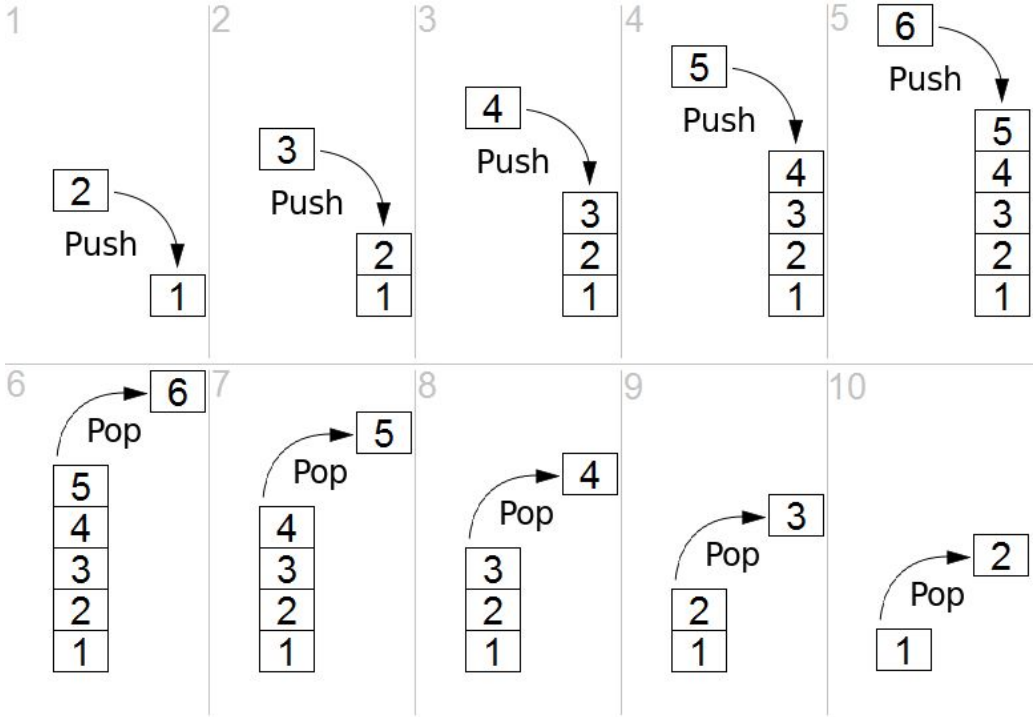


Stack

LIFO

Insertion : O(1)
Deletion : O(1)
Access Time : O(n) [Worst Case]
Insertion and Deletion are allowed

on one end.



Stack - Questions

Infix -> Postfix

Postfix(Reverse Polish Notation): e.p, 3 5 + Infix e.p, 3+5

- (1+2)X3-4 -> 12+3X4-

Rules: (Need a stack for the operators)

Num: Output

Op: a. '(', push;
b. ')', pop until ‘(’, delete ‘(‘.
c. others ch: if precedence of ch is greater than the top, push; otherwise pop until meets a greater than or equals to ch, or a ‘(‘.

Pop all.

Stack - Questions

Postfix Caculation

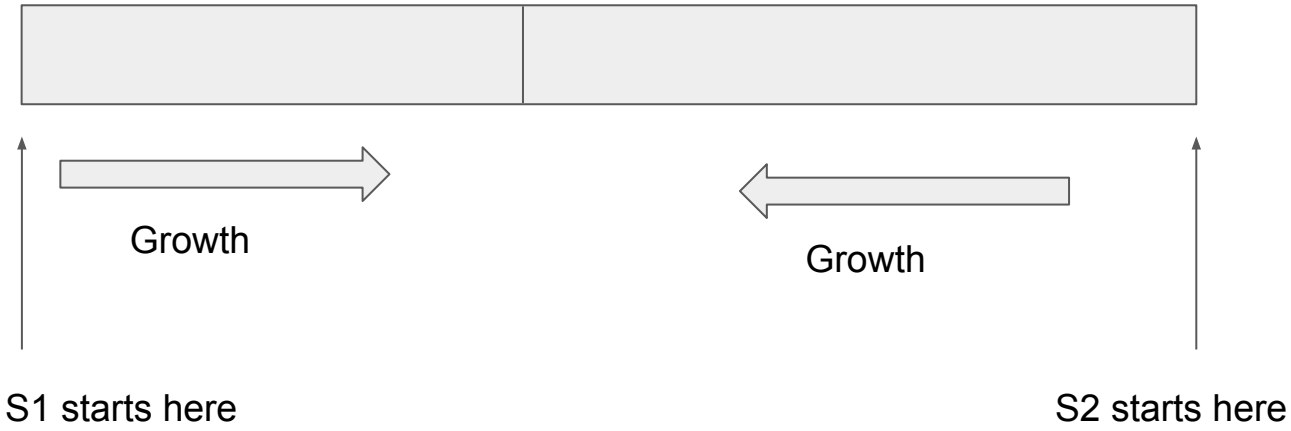
5 + ((1 + 2) * 4) − 3 -> 5 1 2 + 4 * + 3 − -> 14

伪代码 [编辑]

- while有输入符号
  - 读入下一个符号X
  - IF X是一个操作数
    - 入栈
  - ELSE IF X是一个操作符
    - 有一个先验的表格给出该操作符需要n个参数
    - IF堆栈中少于n个操作数
      - （错误） 用户没有输入足够的操作数
    - Else，n个操作数出栈
    - 计算操作符。
    - 将计算所得的值入栈
- IF栈内只有一个值
  - 这个值就是整个计算式的结果
- ELSE多于一个值
  - （错误） 用户输入了多余的操作数

Stack - Questions

Implement two stacks in an array

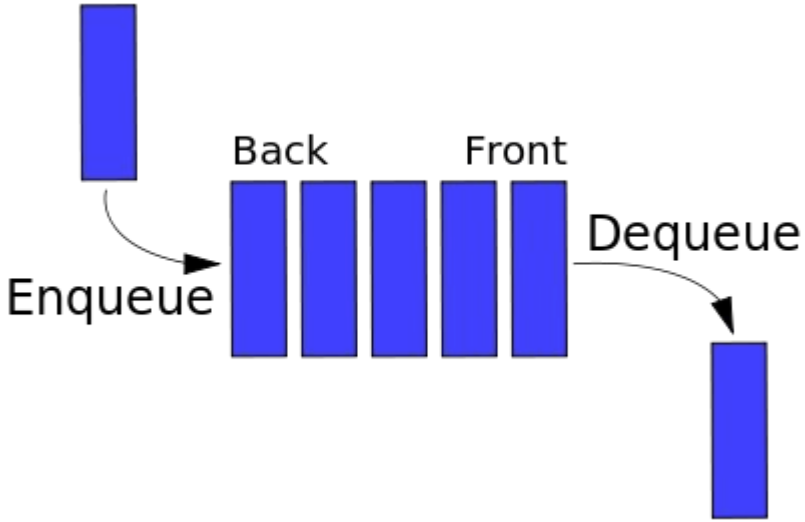


Queue

FIFO

Enqueue,dequeue, front, rear

Insertion : O(1)
Deletion : O(1)
Access Time : O(n) [Worst Case]



Examples include CPU scheduling, Disk Scheduling.

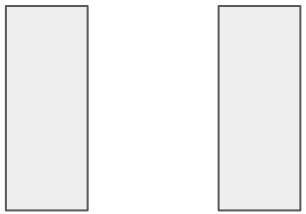
Implementation in an array.

Queue - Questions

1. Priority Queue

   insert(item, priority): Inserts an item with given priority….O(1)
   getHighestPriority(): Returns the highest priority item. (Searching...O(n))
   deleteHighestPriority(): Removes the highest priority item.….O(1)

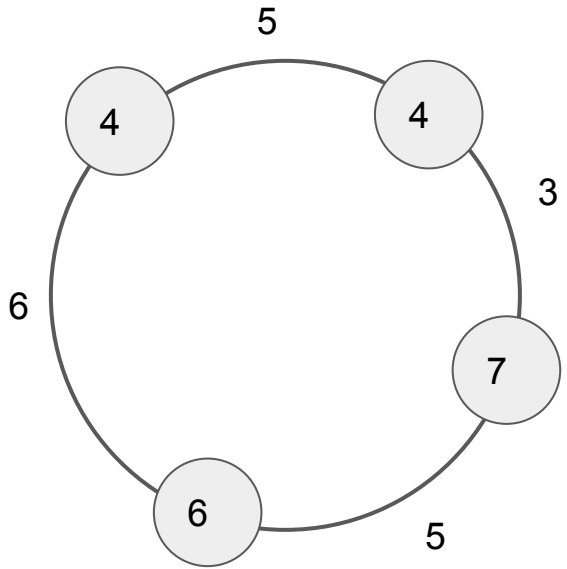
2. Implement Queue using Stacks



2 Stacks

Queue - Questions

3. Find the first circular tour that visits all petrol pumps



Simple: O(n^2)

Better: O(n)
We first enqueue first petrol pump to the queue, we keep enqueueing petrol pumps till we either complete the tour, or current amount of petrol becomes negative. If the amount becomes negative, then we keep dequeueing petrol pumps till the current amount becomes positive or queue becomes empty.

Queue - Questions

dequeue: double-ended queue

http://www.geeksforgeeks.org/maximum-of-all-subarrays-of-size-k/ ? ? ? ?

10,5,4,3,6,2

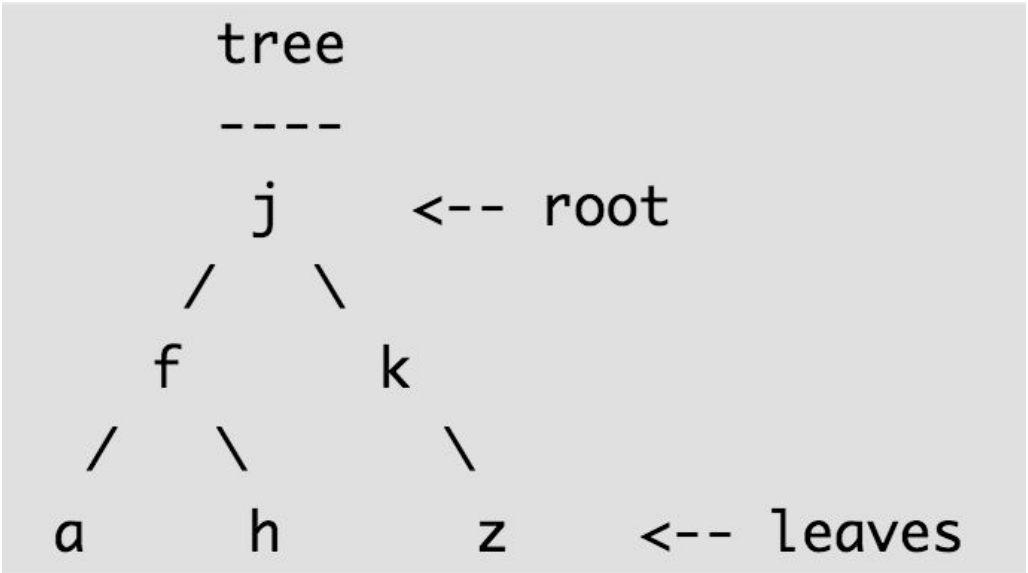
0,1,2,3,4,5

Dequeue: index ; k = 3

front ……rear Output: 10,5,6,6

Binary Tree

Structure in Python



Binary Tree - Properties

1) The maximum number of nodes at level ‘L’ of a binary tree is 2^(L-1).

2) Maximum number of nodes in a binary tree of height ‘h’ is 2^h – 1.

3) In a Binary Tree with N nodes, minimum possible height or minimum number of levels is ⌈ Log2(N+1) ⌉ ...ceil

4) A Binary Tree with L leaves has at least ⌈ Log2L ⌉ + 1 levels

5) In Binary tree, number of leaf nodes is always one more than nodes with two children.

Types

Full Binary (if every node has 0 or 2 children):

L = I + 1

Where L = Number of leaf nodes, I = Number of internal nodes

Complete Binary Tree:只有最下面的两层结点度能够小于2，并且最下面一层的结点都集中在该层最左边的若干位置的二叉树

Perfect Binary Tree: all internal nodes have two children and all leaves are at same level.

Balanced Binary Tree: 它是一棵空树或它的左右两个子树的高度差的绝对值不超过1

Binary Tree - Questions

[Enumeration of Binary Trees]

For example, let T(n) be count for n nodes.
T(0) = 1 [There is only 1 empty tree]
T(1) = 1
T(2) = 2

```
For n  = 1, there is only one tree
   o

For n  = 2, there are two trees
   o       o
  /         \
 o           o
```

T(3) = T(0)*T(2) + T(1)*T(1) + T(2)*T(0) = 1*2 + 1*1 + 2*1 = 5

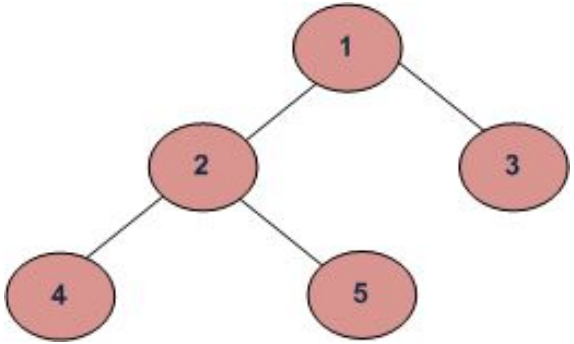
T(4) = T(0)*T(3) + T(1)*T(2) + T(2)*T(1) + T(3)*T(0)
    = 1*5 + 1*2 + 2*1 + 5*1
    = 14

Binary Tree - Questions

Tree Traversals(DFS,BFS)

Depth First Traversals:(Inorder, Preorder and Postorder)

(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

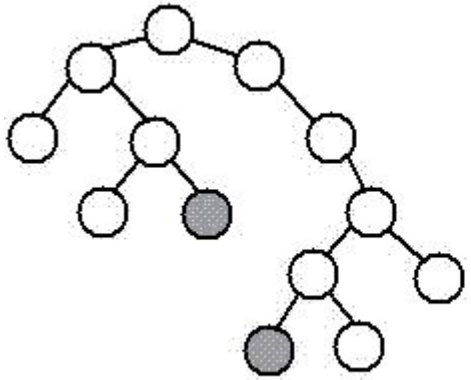


Breadth First Traversal

1 2 3 4 5

Binary Tree - Questions

Diameter of a Binary Tree



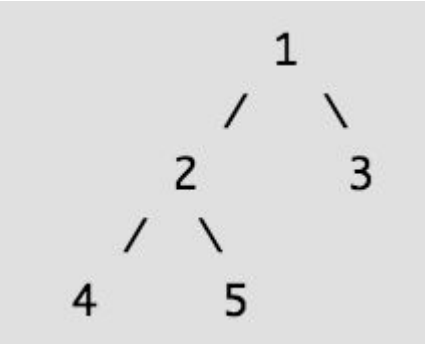
diameter, 9 nodes, through root

```
def height(node):

    # Base Case : Tree is empty
    if node is None:
        return 0 ;

    # If tree is not empty then height = 1 + max of left
    # height and right heights
    return 1 + max(height(node.left) ,height(node.right))
```

Binary Tree - Questions

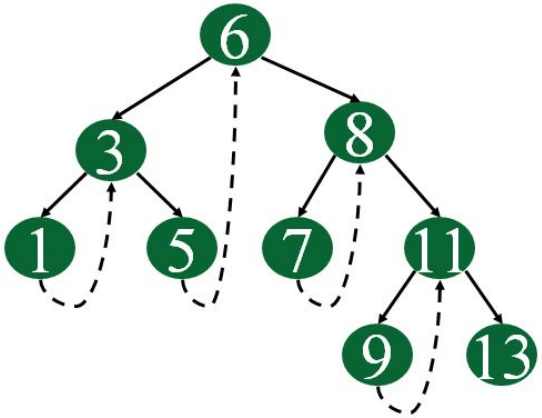


Inorder Tree Traversal without Recursion: Use Stack!

1) Create an empty stack S.
2) Initialize current node as root:**current = root**
3) Push the current node to S and set **current = current->left** until current is NULL
4) If current is NULL and stack is not empty then
    a) Pop the top item from stack.
    b) Print the popped item, set **current = popped_item->right**
    c) Go to step 3.
5) If current is NULL and stack is empty then we are done.

Binary Tree - Questions

Threaded Binary Tree: Fast Inorder Traversal



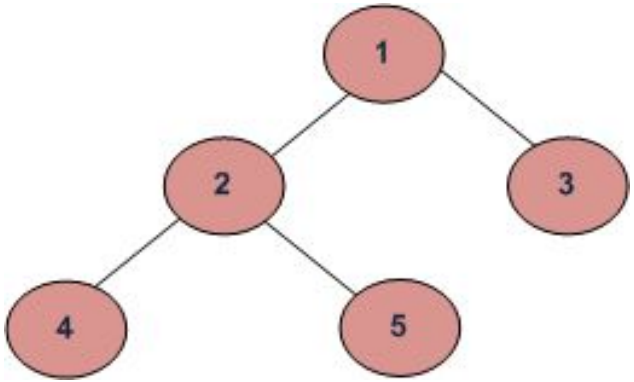
Single Threaded: Where a NULL right pointers is made to point to the **inorder successor** (if successor exists)
Double Threaded: Where both left and right NULL pointers are made to point to **inorder predecessor** and **inorder successor** respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
The threads are also useful for fast accessing ancestors of a node.

Binary Tree - Questions

Find the Maximum Depth or Height of a Tree: 3 <--

maxDepth()

1. If tree is empty then return 0
2. Else
    (a) Get the max depth of left subtree recursively i.e.,
        call maxDepth(tree->left-subtree)
    (a) Get the max depth of right subtree recursively i.e.,
        call maxDepth(tree->right-subtree)
    (c) Get the max of max depths of left and right
        subtrees and add 1 to it for the current node.
        max_depth = max(max dept of left subtree,
                        max depth of right subtree)
                        + 1
    (d) Return max_depth



Binary Tree - Questions

Two traversal sequences:construct the binary tree.

Inorder + any

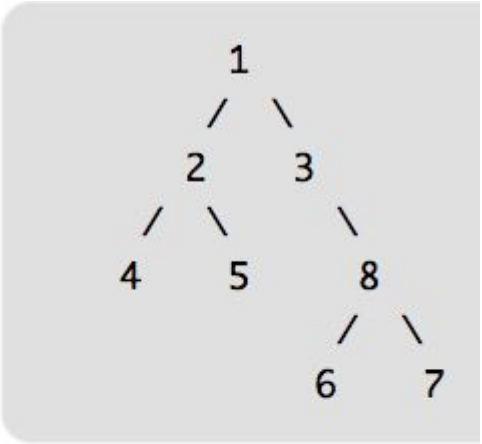
Go here. Code review together!

Binary Tree - Questions

Maximum width of a binary tree

Use level order traversal with **Queue**:

We store all the child nodes at the current level in the queue and then count the total number of nodes after the level order traversal for a particular level is completed.



For the above tree,

width of level 1 is 1,

width of level 2 is 2,

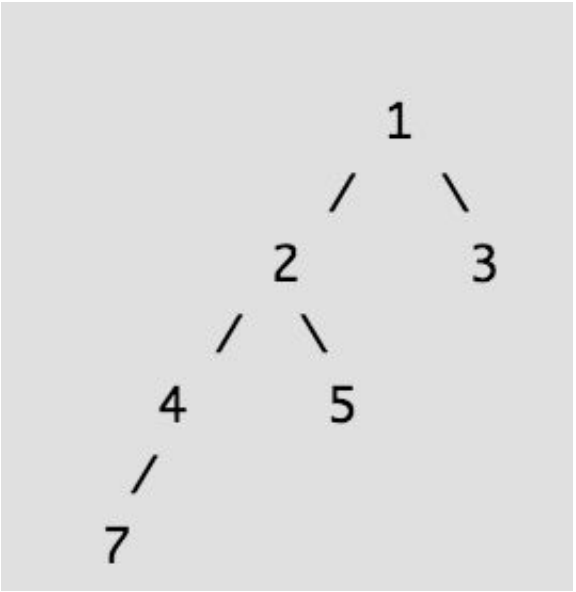
width of level 3 is 3

width of level 4 is 2.

Binary Tree - Questions

Print Ancestors of a given node.

Go here. Code review together!



Given: 7
Print: 4,2,1

Binary Tree - Questions

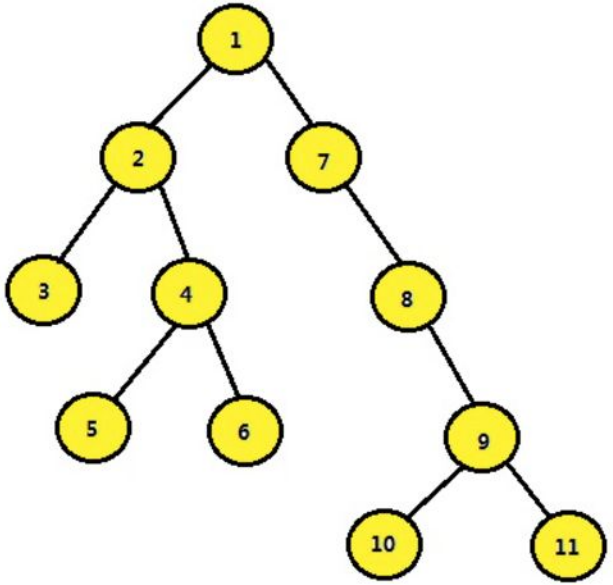
LCA: lowest common ancestor

Print ancestors as lists

Node 3: [2,1]

Node 5: [4,2,1]

Find Node 2.



Given: tree, 3,5
Print: 2

Binary Search Tree

In Python

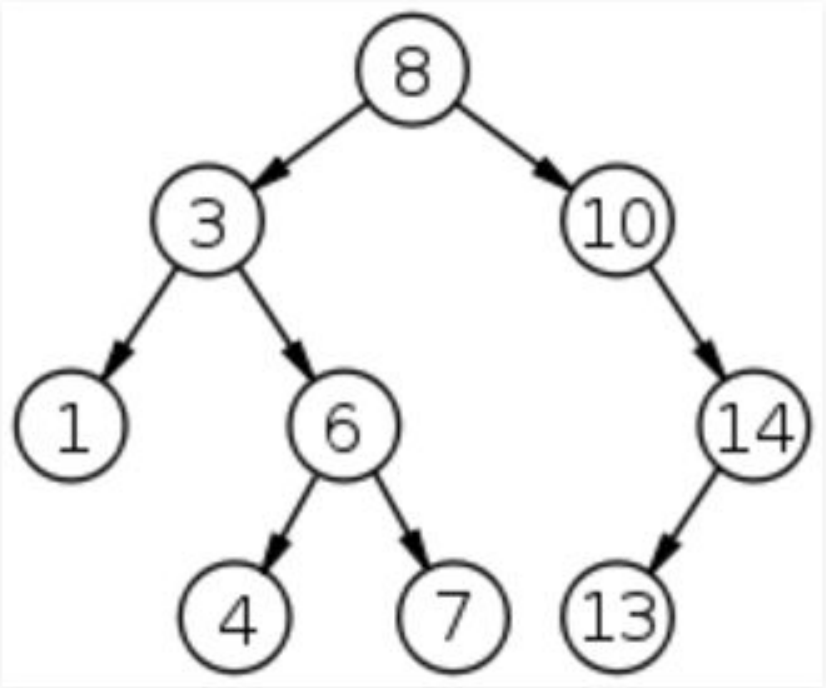
```
# A utility function to search a given key in BST
def search(root,key):

    # Base Cases: root is null or key is present at root
    if root is None or root.val == key:
        return root

    # Key is greater than root's key
    if root.val < key:
        return search(root.right,key)

    # Key is smaller than root's key
    return search(root.left,key)

# This code is contributed by Bhavya Jain
```



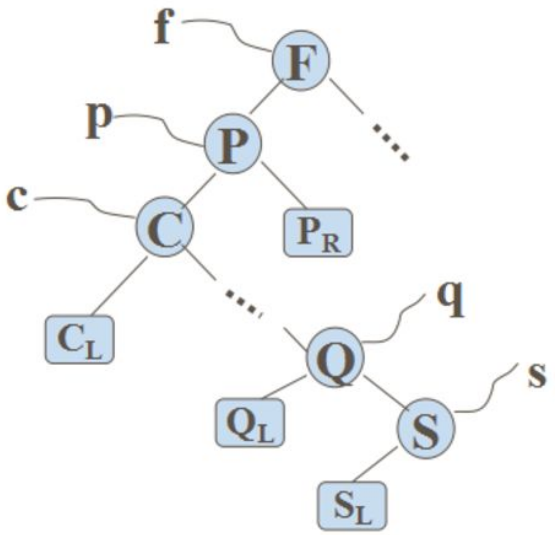
Binary Search Tree

Insertion of a key: Inorder traversal of BST always produces sorted output.

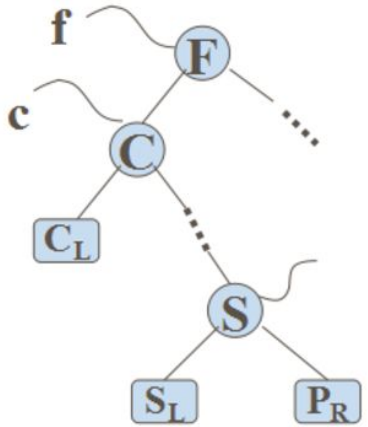
A new key is always **inserted at leaf.**

```
def insert(root,node):
    if root is None:
        root = node
    else:
        if root.val < node.val:
            if root.right is None:
                root.right = node
            else:
                insert(root.right, node)
        else:
            if root.left is None:
                root.left = node
            else:
                insert(root.left, node)
```

方法一：首先找到p结点在中序序列中的直接前驱s结点，如图 (b) 所示，然后将p的左子树改为f的左子树，而将p的右子树改为s的右子树：f->lchild=p->lchild；s->rchild= p->rchild；free(p)；结果如图 (c) 所示。



(b) S为P的直接前驱



(c) 将P的左子树改为F的左子树，
将P的右子树改为S的右子树。

方法二：首先找到p结点在中序序列中的直接前驱s结点，如图 (b) 所示，然后用s结点的值，替代p结点的值，再将s结点删除，原s结点的左子树改为s的双亲结点q的右子树：p->data=s->data；q->rchild= s->lchild；free(s)；结果如图 (d) 所示。

Binary Search Tree - Questions

Find the node with minimum value: lefmost child

Inorder predecessor and successor for a given key in BST

Check if a binary tree is a BST: inorder traversal

LCA in a BST: Traversal from the root. If node.data is in [n1,n2], then node is an common ancestor. If node.data is out of range: greater then search left tree; otherwise go to right tree. [Recursive]

n1=4 node 2

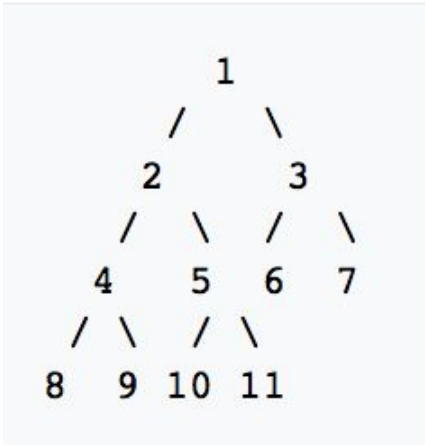
n2=8

Heap

- 任意节点小于（或大于）它的所有后裔，最小元（或最大元）在堆的根上（**堆序性**）。
- 堆总是一棵完全树。即除了最底层，其他层的节点都被元素填满，且最底层尽可能地从左到右填入。

根节点在数组中的位置是1，第*n*个位置的子节点分别在2*n*和 2*n*+1

如果存储数组的下标基于0，那么下标为i的节点的子节点是2i + 1与2i + 2；其父节点的下标是⌊(i − 1)／2⌋。



6 5 3 1 8 7 2 4

位置: 1 2 3 4 5 6 7 8 9 10 11
左图: 1 2 3 4 5 6 7 8 9 10 11

Hashing

Direct Access Table: Insert, Delete, Search...O(1)

Hash Function: h(x) = x **mod** 3 (Collision) x = 8 ...2 x = 4...1

Collision Handling:

1) Chaining: each one is a linked list
2) Open Addressing

Hashing

Complexity :

n= number of keys stored in table

m= number of slots in table

α = Average keys per slot / load factor = n / m

Expected time to insert/search/delete :

O(1 +α)

Collision Handling:

1) Chaining: each one is a linked list

   Simple; Never fills up;

   Good for uncertainty.

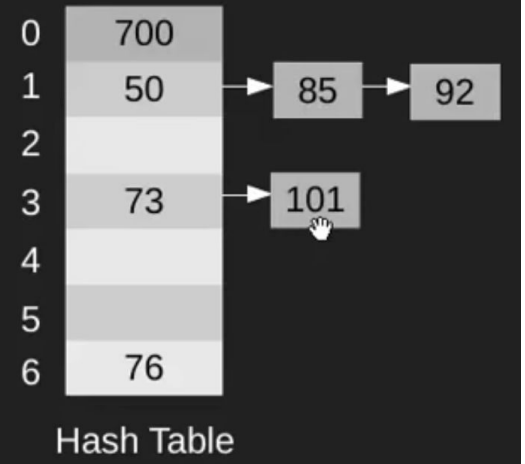
   Long:...O(n)

   Extra space for links.

Hash Function : h(x) = x mod 7

Keys : 50, 700, 76, 85, 92, 73, 101.

101 mod 7 = 3



Hashing

Collision Handling:

2) Open Addressing

Linear Probing: If slot hash(x) % S is full, then we try (hash(x) + 1) % S, +2...

Quadratic Probing: (hash(x) + 1*1) % S, (hash(x) + 2*2) % S, etc..

Double Hashing:

c) Double Hashing We use another hash function hash2(x) and look for i*hash2(x) slot in i'th rotation.

```
let hash(x) be the slot index computed using hash function.
If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S
If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S
If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S
.................................................
```

Hashing - Problems

Find if a subset

Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}
Output: arr2[] is a subset of arr1[]

Hash method: O(m+n)

1) Create a Hash Table for all the elements of arr1[].
2) Traverse arr2[] and search for each element of arr2[] in the Hash Table. If element is not found then return 0.
3) If all elements are found then return 1.

Hashing - Problems

Union & Intersecion of Two linked lists

```
Input:
   List1: 10->15->4->20
   lsit2:  8->4->2->10
Output:
   Intersection List: 4->10
   Union List: 2->8->20->4->15->10
```

Union (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse both lists one by one, for each element being visited, look the element in hash table. If the element is not present, then insert the element to result list. If the element is present, then ignore it.

Intersection (list1, list2)
Initialize the result list as NULL and create an empty hash table. Traverse list1. For each element being visited in list1, insert the element in hash table. Traverse list2, for each element being visited in list2, look the element in hash table. If the element is present, then insert the element to result list. If the element is not present, then ignore it.

Hashing - Problems

Given an array A[] and a number x, check for pair in A[] with sum as x.

Example:
Let Array be {1, 4, 45, 6, 10, -8} and sum to find be 16 -----> Return True

Method 1: Sort & search
Sort the array
A = {-8, 1, 4, 6, 10, 45} ...O(nlogn) Search: O(n)

<key,value>

Method 2: Hash Map

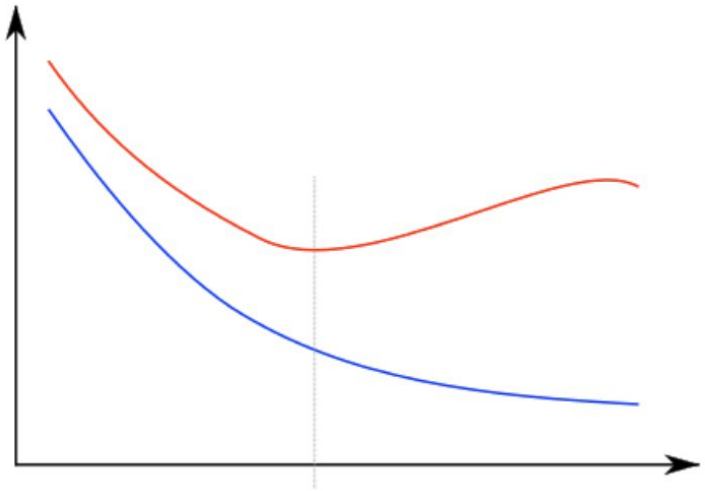
1) Initialize Binary Hash Map M[] = {0, 0, ...}
2) Do following for each element A[i] in A[]
  (a) If M[x - A[i]] is set then print the pair (A[i], x - A[i])
  (b) Set M[A[i]]

ML Algorithms

Let’s do it, again & again.

Regularization!

- Regularization will prevent overfitting when we have a lot of features (or later a very powerful/deep model)
  - x-axis: more powerful model or more training iterations
  - Blue: training error, red: test error



Bayes Classifiers (1): Bayes' Theorem

Bayes' Theorem
事件A在事件B（发生）的条件下的概率，与事件B在事件A（发生）的条件下的概率是不一样的；然而，这两者是有确定的关系的，贝叶斯定理就是这种关系的陈述。

Two events: A & B

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

P(A|B)是已知B发生后A的条件概率，也由于得自B的取值而被称作A的后验概率。
P(B|A)是已知A发生后B的条件概率，也由于得自A的取值而被称作B的后验概率。
P(A)是A的先验概率（或边缘概率）。之所以称为"先验"是因为它不考虑任何B方面的因素。
P(B)是B的先验概率或边缘概率。

Bayes Classifiers (1): Model

Features X, Class Y

- Joint distribution $p(y|x)p(x) = p(x,y) = p(x|y)p(y)$
- Bayes Rule: $\Rightarrow \quad p(y|x) = p(x|y)p(y)/p(x)$

$$= \frac{p(x|y)p(y)}{\sum_c p(x|y=c)p(y=c)}$$

Features	# bad	# good
X=0	42	15
X=1	338	287
X=2	3	5

⇒

p(x | y=0)	p(x | y=1)
42 / 383	15 / 307
338 / 383	287 / 307
3 / 383	5 / 307

⇒

p(y=0|x)	p(y=1|x)
.7368	.2632
.5408	.4592
.3750	.6250

p(y)	383/690	307/690

Bayesian Method

Assumptions:

- Bayes' Theorem
- Features are conditionally independent

Overview:

- Learn Joint probability distribution p(x,y)
- Learn p(x)
- Calculate p(y|x)

Naïve Bayesian: Notations

Dataset: $T=\{(x_1,y_1),,(x_2,y_2),..,(x_N,y_N)\}$

Priori probability: K classes $P(Y=c_k),k=1,2...K$

Conditional probability: $P(X=x|Y=c_k),x\in R^d$

Then learn Joint: P(X,Y)

Calculate Posterior: P(Y|X)

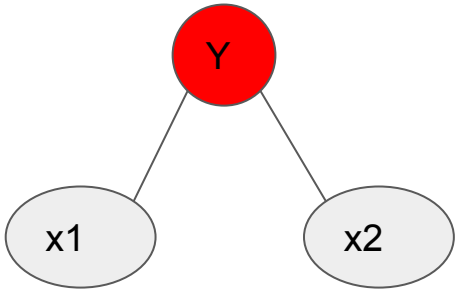
Naïve Bayesian: Attribute Conditional independent assumption

Each input x is a vector with n elements. $x^{(1)}, x^{(2)}, x^{(3)} \ldots$

$$P(X = x \mid Y = c_k) = P(X^{(1)} = x^{(1)}, \cdots, X^{(n)} = x^{(n)} \mid Y = c_k)$$

$$= \prod_{j=1}^{n} P(X^{(j)} = x^{(j)} \mid Y = c_k)$$

Given the class y, features are conditionally independent.



Naïve Bayesian: Calculate

$$P(Y=c_k \mid X=x)=\frac{P(X=x \mid Y=c_k)P(Y=c_k)}{\sum_k P(X=x \mid Y=c_k)P(Y=c_k)}$$

贝叶斯定理通常可以再写成下面的形式：

$$P(B) = P(A, B) + P(A^C, B) = P(B|A)P(A) + P(B|A^C)P(A^C),$$

其中A^C是A的补集（即非A）。故上式亦可写成：

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B|A)P(A) + P(B|A^C)P(A^C)},,$$

在更一般化的情况，假设$\{A_j\}$是事件集合里的部分集合，对于任意的A_i，贝叶斯定理可用下式表示：

$$P(A_i|B) = \frac{P(B|A_i)\,P(A_i)}{\sum_j P(B|A_j)\,P(A_j)},$$

$$P(Y=c_k \mid X=x)=\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)}, \quad k=1,2,\cdots,K$$

Naïve Bayesian: Definition

Want the class that has the max of the probabilities:

$$y = f(x) = \arg\max_{c_k} \frac{P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} \mid Y = c_k)}{\sum_k P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} \mid Y = c_k)}$$

For everyone, the denominators are the same, so we can simplify it:

$$y = \arg\max_{c_k} P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} \mid Y = c_k)$$

Naïve Bayesian: Parameter Estimation

Maximum Likelihood Estimation: 极大似然估计

Goal: estimate priori P(Y=c) for each class

$$P(Y=c_k)=\frac{\sum_{i=1}^{N} I(y_i=c_k)}{N},\quad k=1,2,\cdots,K$$

estimate P(X=x | Y=c) for each class

$$P(X^{(j)}=a_{jl}\mid Y=c_k)=\frac{\sum_{i=1}^{N} I(x_i^{(j)}=a_{jl},y_i=c_k)}{\sum_{i=1}^{N} I(y_i=c_k)}$$

$$\frac{\text{Count(x,y)}}{\text{Count(y)}} = P(x|y)$$

Naïve Bayesian: Cost

Choose **0-1 cost function:**

Right answer = 0

Wrong answer = 1

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases}$$

-- L gives "how big" it is making the mistake...we want to minimize it!

$$= \arg\min_{y \in \mathcal{Y}} \sum_{k=1}^{K} P(y \neq c_k \mid X = x)$$

Add up those we go wrong

$$= \arg\min_{y \in \mathcal{Y}} (1 - P(y = c_k \mid X = x))$$

1- Right cases

$$= \arg\max_{y \in \mathcal{Y}} P(y = c_k \mid X = x)$$

Minimize the errors ⇔ Maximize the right

根据期望风险最小化准则，得到了后验概率最大化准则

Naïve Bayesian: Then Calculate!

Calculate a prob with an input x:

$$P(Y=c_k)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}\mid Y=c_k),\quad k=1,2,\cdots,K$$

Decide the class of the input x:

$$y=\arg\max_{c_k}P(Y=c_k)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}\mid Y=c_k)$$

A demo from the book.

Naïve Bayesian: Bayesian Estimation

When do MLE, could be 0 counts. Then you times it...all become 0.

Add one more thing, lambda >= 0. If lambda = 0, it is MLE; If lambda = 1, it is Laplace smoothing.

$$P_\lambda(X^{(j)} = a_{jl} \mid Y = c_k) = \frac{\sum_{i=1}^{N} I(x_i^{(j)} = a_{jl}, y_i = c_k) + \lambda}{\sum_{i=1}^{N} I(y_i = c_k) + S_j \lambda}$$

Laplace smoothing, it is a probalistic distribution.

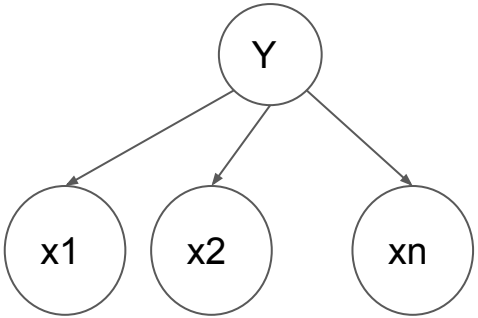
Priori Prob's Bayesian Estimation:

$$P_\lambda(Y = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k) + \lambda}{N + K\lambda}$$

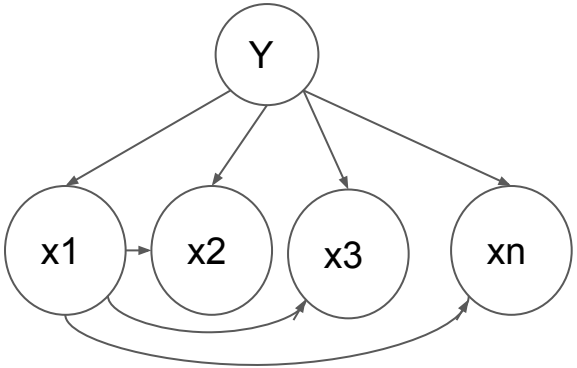
Bayesian: More Models

Naïve Bayesian: attribute Conditional independent assumption

Semi-Naive Bayesain: One-Dependent Estimator (每个属性在类别之外最多依赖于一个其他属性)

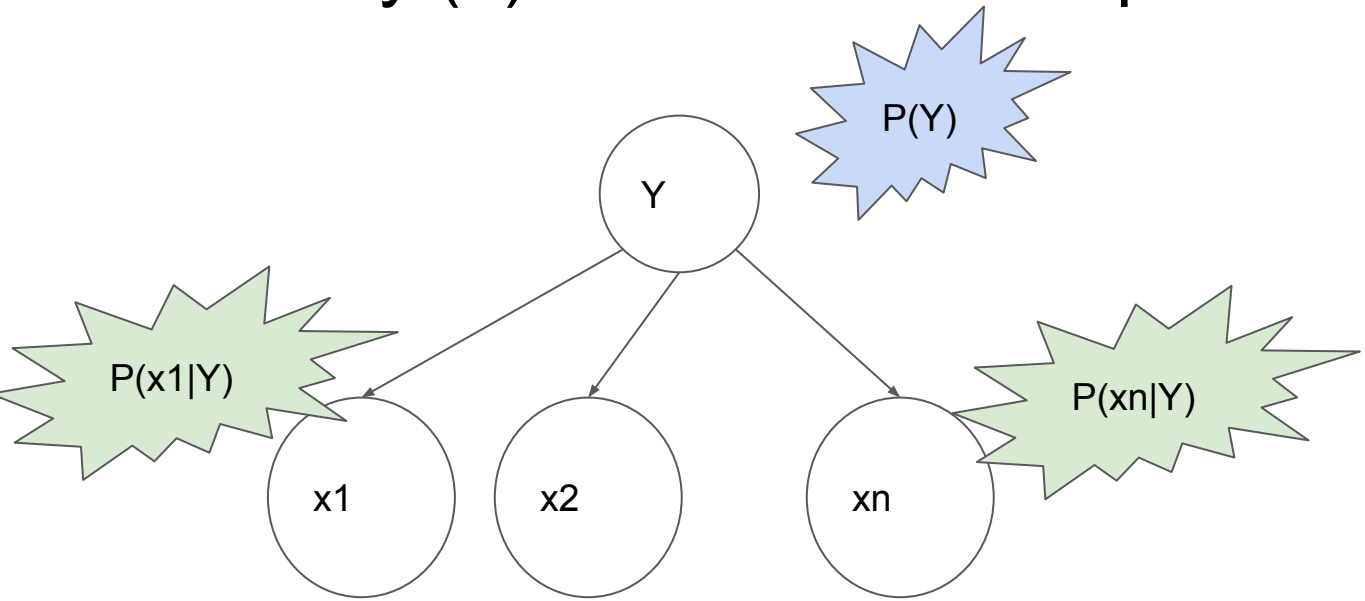


Naive Bayesian



Super-Parent One-Dependent Estimator(SPODE)

Takeaway (1): Probabilistic Graphical Models



Naive Bayesian

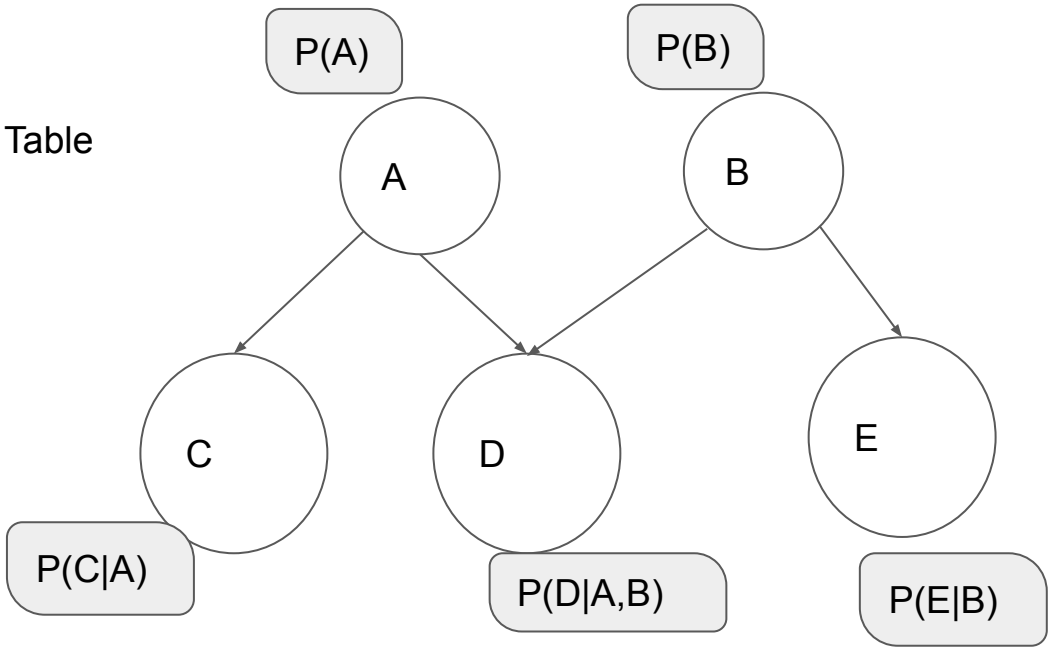
联合概率：P(y)P(x1|y)P(x2|y)..P(xn|y)

Joint Prob! What we want!

Find out More: https://www.coursera.org/learn/probabilistic-graphical-models（Prof.Daphne Koller）

Takeaway (2): Bayesian Network

Named also Belief Network
DAG: Directed Acyclic Graph
CPT: Conditional Probability Table



Given P(A),
C and D, independent
(条件独立性)
C⊥D|A

联合概率: P(A)P(B)P(C|A)P(D|A,B)P(E|B)
Joint Prob! What we want!

Top 10 Algorithms in data mining (2007)

C4.5

K-Means

SVM

The Apriori Algorithm

EM:Expectation–Maximization

PageRank

AdaBoost

KNN

Naive Bayes

CART

Wu http://www.cs.uvm.edu/~icdm/algorithms/10Algorithms-08.pdf

Clustering: Unsupervised Learning

Similarty Methods: Eculidean Distance, etc.

- In-group: high similarity
- Out-group: high distance

Methods

- Prototype-based Clustering 原型聚类 : K-means, LVQ, MoG
- Density-based Clustering 密度聚类: DBSCAN
- Hierarchical Clustering 层次聚类: AGNES,DIANA

Prototype-based Clustering: k-means

Dataset D, Clusters C

$$D=\{x_1,x_2,..,x_m\}$$
$$C=\{C_1,C_2,..,C_k\}$$

Error(each one in each cluster)

$$E=\sum_{i=1}^{k}\sum_{x\in C_i}||x-\mu_i||^2$$

$$\mu_i=\frac{1}{|C_i|}\sum_{x\in C_i}\boldsymbol{x}$$

K-means: algorithm

(Input: whole dataset points x1,...xm)

O(#iter * #clusters * #instances * #dimensions)

Initialization: Randomly place centroids c1..ck

Repeat until convergence (stop when no points changing):

- for each data point xi:

  argmin D (xi,cj) for all cj

  find nearest **centroid**, set the point to the centroid cluster

- for each cluster:

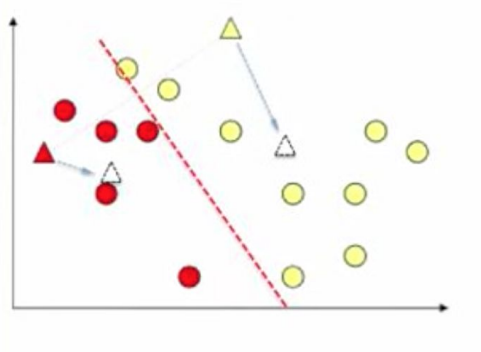
  calculate new centroid (means of all new points)

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

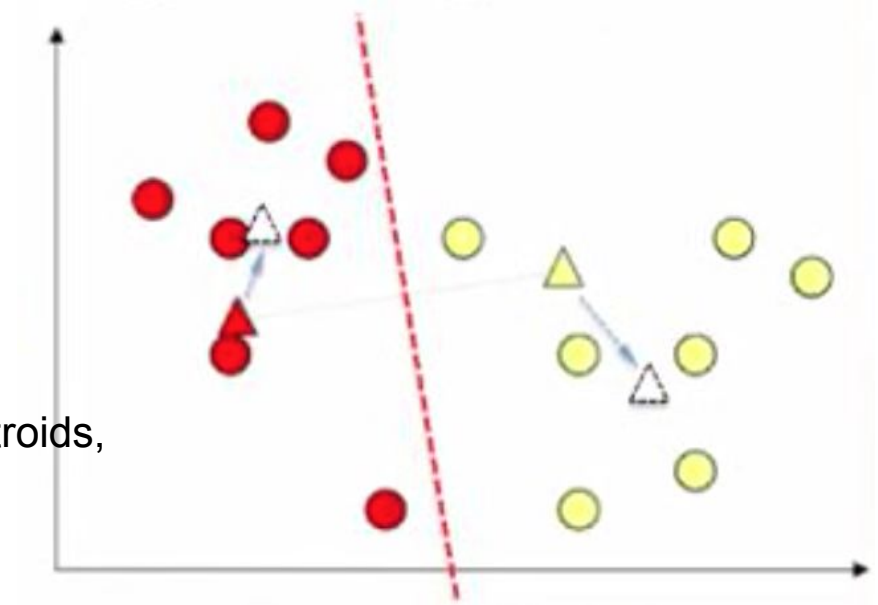
https://www.youtube.com/watch?v=_aWzGGNrcic

K-means: a demo

Two clusters, squares are the centroids.



Calculate new centroids,
finish one iteration.



K-means: an sklearn demo

Initialization matters, like the Neural Networks.

K-means

- Strength
  - *Relatively efficient*: *O*(*tkn*), where *n*, k, and t are the number of objects, number of clusters, and number iterations respectively. Normally, *k*, *t* << *n*.
  - Often terminates at a *local optimum*
  - The *global optimum* may be found using techniques such as: *deterministic annealing* and *genetic algorithms*
- Weakness
  - Applicable only when *mean* is defined, then what about categorical data?
  - Need to specify *k,* the *number* of clusters, in advance
  - Unable to handle noisy data and *outliers*
  - Not suitable to discover clusters with *non-convex shapes*

From UCD, Prof. Tahar.

k-modes:

-cat,frequency

k-prototype:

- num+cat

Clustering: Other Methods(1)

Prototype-based Clustering

- LVQ: Find out a group of vectors to describe the clusters, with **labels**
- MoG: describe the model by a probabilistic model.

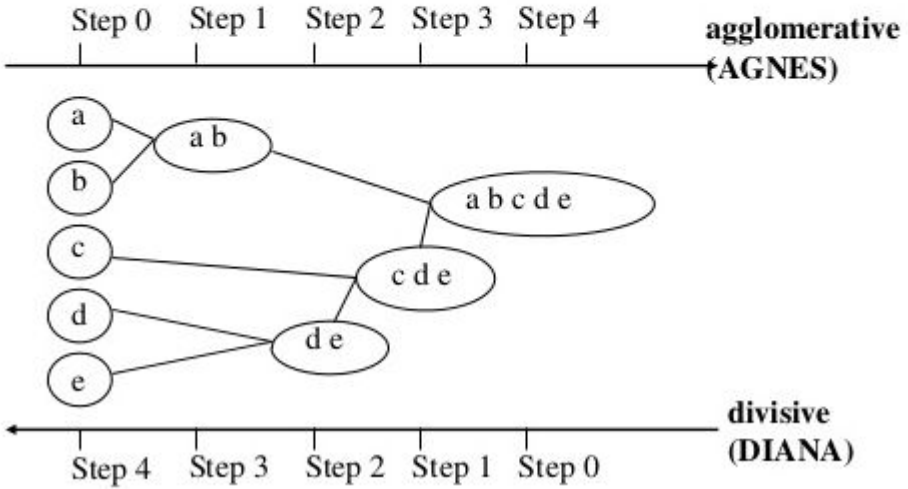
Density-based Clustering 密度聚类

- DBSCAN

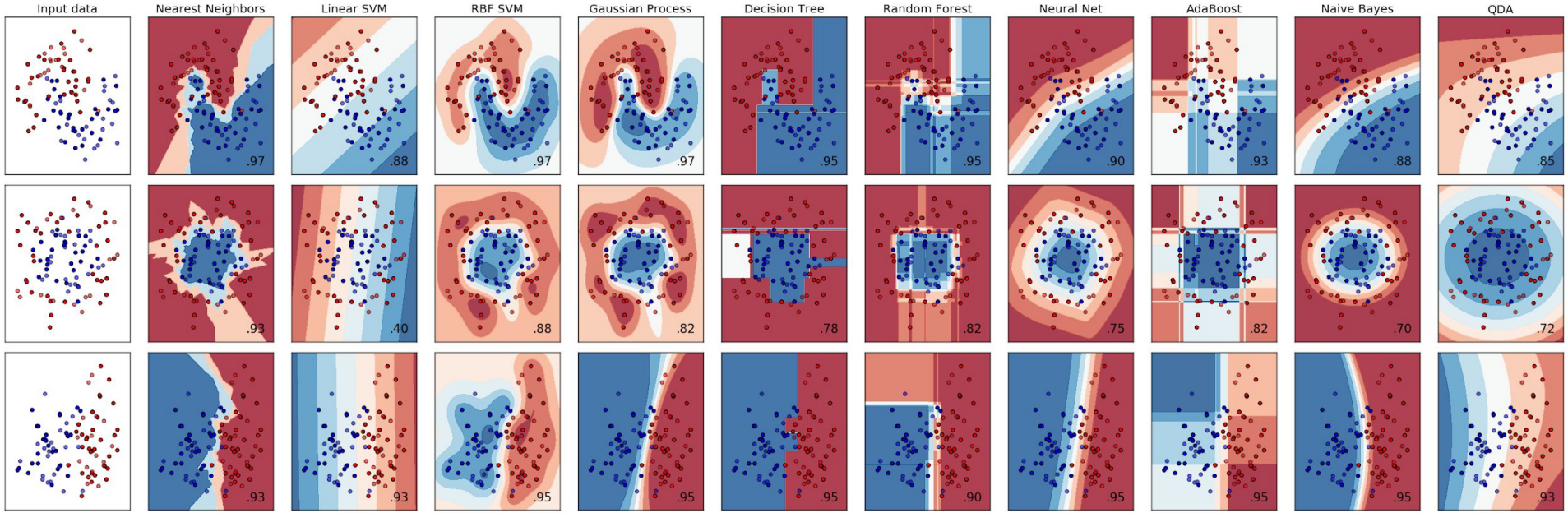
Clustering: Other Methods(2)

Hierarchical Clustering 层次聚类: AGNES,DIANA

- Use distance matrix as clustering criteria. This method does not require the number of clusters ***k*** as an input, but needs a termination condition



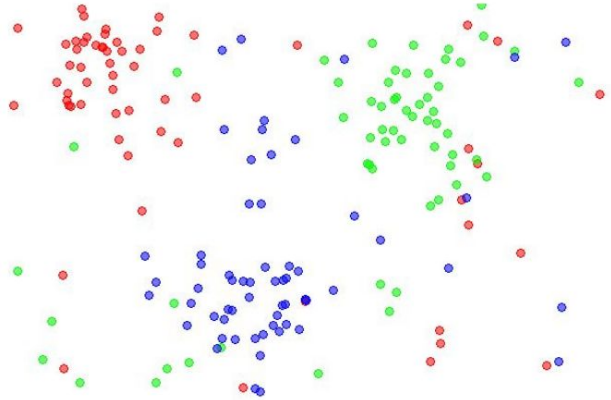
Takeaway(1): recognize algorithms!



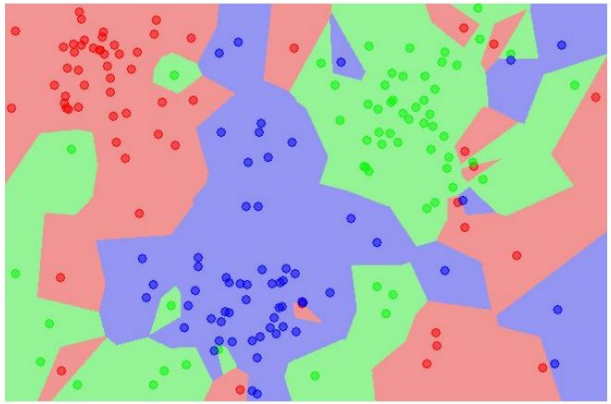
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

KNN

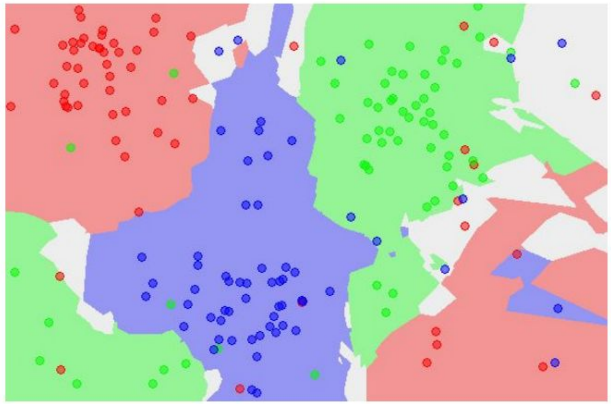
the data



NN classifier



5-NN classifier



http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Takeaway(2): Dataset ML Strategy

1%	称为 Overfitting	1%	Both Bias/Variance
2%		6%	
6%		10%	

Human level error 1%

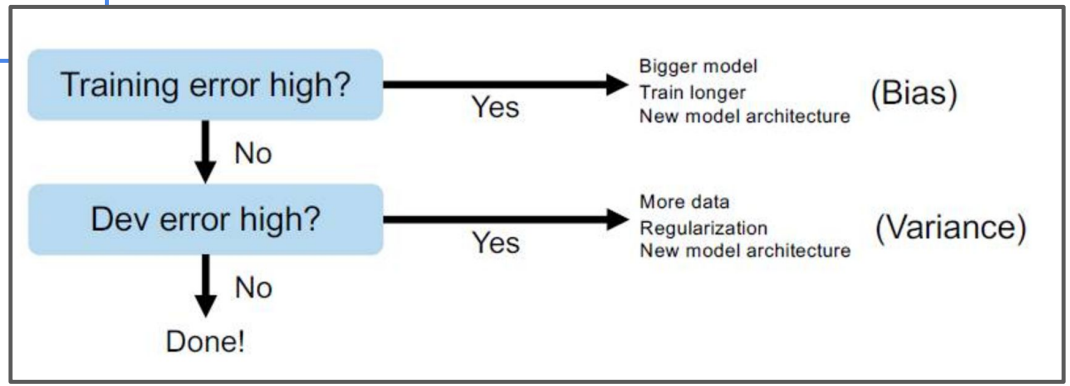
Training set error 5%

Dev set error 8%

"Avoidable bias"

"Variance"

Bias-variance Trade-off =
Avoidable Bias + Variance



From Andrew Ng, NIPS, 2016

More Data; Regularization; New Model

Back Propagation: Main Purpose

The main purpose of back propagation is to find the partial derivatives $\partial C/\partial w_i$
w means all parameters in the network including weights and biases
C is the cost function of the network:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

where n is the sample number and a(x) is the output of the network

It is useful to first point out the naive forward propagation algorithm implied by the chain rule. Then we can find out the advantages of back propagation by simply comparing 2 algorithms.

Naive Forward Propagation

Naive forward propagation algorithm using chain rule to compute the partial derivatives on every node of the network in a forward manner.

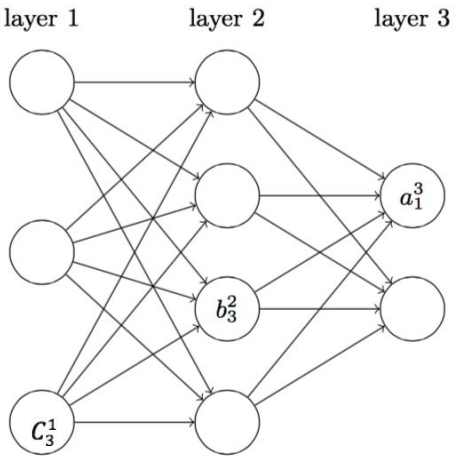
Which means the partial derivatives on layer 3 can only be computed after we have done the same work on layer 1 and layer 2.

How it works:

Compute $\partial u_i / \partial u_j$ for every pair of nodes where u_i is at a higher level than u_j.

In order to compute $\partial a_1^3 / \partial c_3^1$, we need to compute $\partial b_i^2 / \partial c_3^1$ for every input of a_1^3, Thus the total work in the algorithm is $O(VE)$

where v is the total node number of the network and e is the total edge number of the network



Back Propagation -Define Error

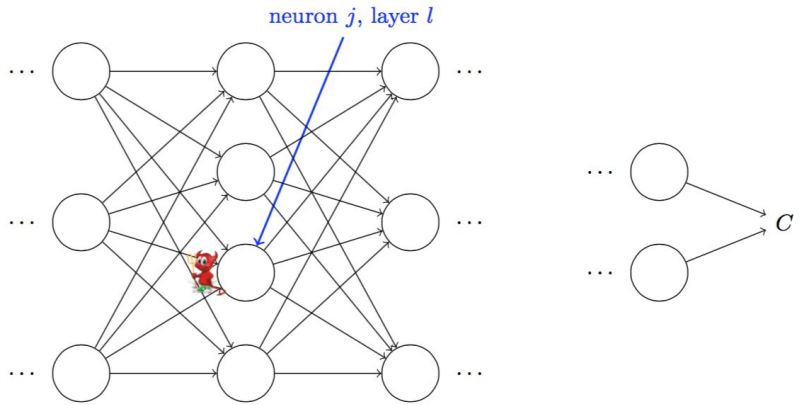
In order to calculate the partial derivative $\partial C / \partial w_i$, let us first define the error on a single node.

δ_j^l stands for the error on the j th node in l th layer.

Back propagation is trying to calculate the errors on the nodes then use it to calculate the partial derivative $\partial C / \partial w_i$

Back Propagation -Calculate Error

If we try to change the value of z_j^l (z is the input of the node), it will affect the result of the next layer and finally affect the output.



Assume $\frac{\partial C}{\partial z_j^l}$ is close to 0, then change the value of z will not help us to minimize the cost C, in this case we can say that node is close to optimum.

Naturally we can define error as: $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$

Back Propagation -Calculate Error

Using chain rule, we can deduce $\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$

By replacing the parital derivative with vector: $\delta^L = \nabla_a C \odot \sigma'(z^L)$

∇aC is a vector with elements equal to ∂C/∂aLj

⊙ is hadamard operater $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1*3 \\ 2*4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$

Back Propagation -Back Propagation

From layer l+1 to layer l, trying to use to δ^{l+1} to represent δ^{l}

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \quad (1)$$

where $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$ (2)

By combine (1) and (2):

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Then we can calculate the errors on all layers. .

Back Propagation -From error to parameters

After calculating the error, we finally need one more step:

Using error to calculate the derivatives on parameters(weights and biases)

Given the equation: $$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}$$

For bias b: $$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

For weight w: $$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Convolutional Neural Network -Convolution Layer

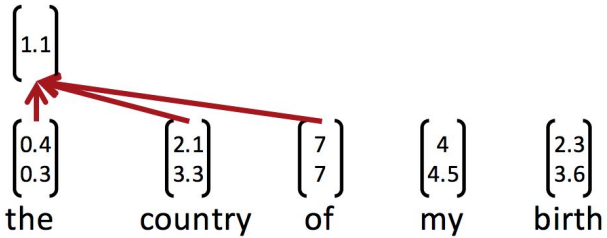
Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$

Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$ (vectors concatenated)

Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (goes over window of h words)

Could be 2 (as before) higher, e.g. 3:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



Convolutional Neural Network -Pooling Layer

The main idea of “Max Pooling Layer” is to capture the most important activation (maximum overtime)

From feature map $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$

Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$

This operation shrinks the feature number(from n-h+1 to 1), how can we get more features?

Applying multiple filters with different window sizes and different weights.

Convolutional Neural Network -Multi-channel

Start with 2 copies of the word vectors, then pre-train one of them(word2vec or Glove), this action change the value of vectors of it. Keep the other one static.

Apply the same filter on them, then sum the Ci of them before max pooling

Softmax

柔性最大值的想法其实就是为神经网络定义一种新式的输出层。开始时和 S 型层一样的，首先计算带权输入[4] $z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$。不过，这里我们不会使用 S 型函数来获得输出。而是，会在这一层上应用一种叫做**柔性最大值函数**在 z_j^L 上。根据这个函数，第 j 个神经元的激活值 a_j^L 就是

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \tag{78}$$

Convolutional Neural Network -Dropout

Create a masking vector r with random variables 0 or 1.
Using hardmard operation to delete some of the features to prevent co-adaption(overfitting?)

At training time, gradients are backpropagated only through those elements of z vector for which $r_i = 1$

At test time, there is no dropout, so feature vectors z are larger.

Hence, we scale final vector by Bernoulli probability p

$$\hat{W}^{(S)} = pW^{(S)}$$

Kim (2014) reports 2 – 4% improved accuracy and ability to use very large networks without overfitting

Word Vectors-Taxonomy

Use a big “graph” to define relationships between words, it has tree structure to declare the relationship between words.

Famous example(WordNet)

Disadvantages:
Difficult to maintain(when new word comes in).
Require human labour.
Hard to compute word similarity.

Word Vectors-One Hot Representation

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a "one-hot" representation. Its problem:

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Word Vectors-Distributional similarity based representations

Main Idea: Represent the word by the mean of its neighbors.

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Use co-ocurrence matrix X to represent.

Full document: As a result, this action will only reveal the general topic of the document.

Windows: Around each word, captures both syntactic (POS) and semantic information

Word Vectors-Window Based Coocurrence-matrix

Problems:
Space consuming.
Difficult to update.

Solution:
Reduce dimension.(Singular Value Decomposition)

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Window size = 1

Word Vectors-Word2vec

Compare to the previous approach, word2vec predicts neighbor words instead of capturing cooccurrence counts.
Faster and can easily incorporate a new sentence/ document or add a word to the vocabulary.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-m \le j \le m, j \neq 0} \log p(w_{t+j}|w_t)$$

Word Vectors-Word2vec

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log p(w_{t+j}|w_t)$$

Where m is the window size. theta means all the parameters that we want to optimize.

For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W}\exp\left(u_w^T v_c\right)}$$

Word Vectors-Word2vec

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

Problems:

With large vocabularies this objective function is not scalable and would train too slow.

Solution:

Negative sampling.

Word Vectors-Word2vec

These representations are *very good* at encoding dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$

  - Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Word Vectors-Skip Gram Model

PSet 1: The skip-gram model and negative sampling

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function: $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$$

Word Vectors-Continuous Bag of Words

Differ from word2vec, this model trying to predict the center word from the surrounding words.

The result will be slightly different from skip-gram model, by average them we can get a better result.

Word Vectors-Comparison

Count based vs direct prediction

LSA, HAL (Lund & Burgess),
COALS (Rohde et al),
Hellinger-PCA (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- NNLM, HLBL, RNN, Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Word Vectors-Glove

Collect the co-occurrence statistics information from the whole corpus instead of going over one window at a time.

Then optimize the vectors using the following equation.

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

Where Pij is the coocurrence counts from the matrix.

Fast training(do not have to go over every window), Scalable to huge corpora, Good performance with small corpus.

Word Vectors-Glove

Performe well at capturing the similarity.

Glove results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Word Vectors-Evaluation Good

How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing one subsystem with another improves accuracy → Winning!

Apache Cassandra

Apache Cassandra is a massively scalable open source non-relational database.

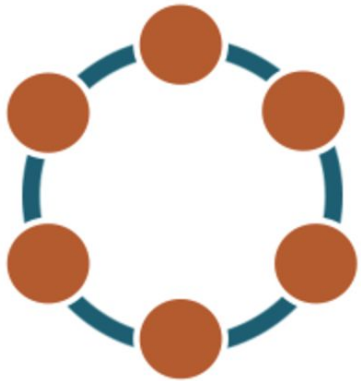
Top features:

Ring architecture differ from master-slaves structure, every nodes are equally important,
all nodes communicating with each other via a distributed, scalable protocol called "gossip."

Capable of handling large amounts of data and and thousands of concurrent users.

Support muti-data-center and cloud deployment.

Using CQL which is similar to SQL



HDFS

Comparing to RDBMS:

HDFS can process simi-structure data(XML, CSV, JSON).

HDFS write speed is faster.

Key features:

Data in HDFS is stored in the form of **blocks** and it operates on the Master Slave Architecture.

The minimum amount of data that can be read or written is generally referred to as a “block” in HDFS. The default size of a block in HDFS1 is 64MB.

MapReduce

MapReduce distributes the workload into various tasks that can run in a parallel way.

The map job breaks down the data sets into key-value pairs or tuples. After that, the reduce job then takes the output of the map job and combines the data tuples to into smaller set of tuples.

3 Core function in Reducer:

1)setup () – This method of the reducer is used for configuring various parameters like the input data size, distributed cache, heap size, etc.

2)reduce () it is heart of the reducer which is called once per key with the associated reduce task.

3)cleanup () - This method is called only once at the end of reduce task for clearing all the temporary files.

MapReduce-partitioning, shuffle and sort

Shuffle Phase-Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.

Sort Phase- Hadoop MapReduce automatically sorts the set of intermediate keys on a single node before they are given as input to the reducer.

Partitioning Phase-The process that determines which intermediate keys and value will be received by each reducer instance is referred to as partitioning. The destination partition is same for any key irrespective of the mapper instance that generated it.

Recurrent Neural Network-Forward prop

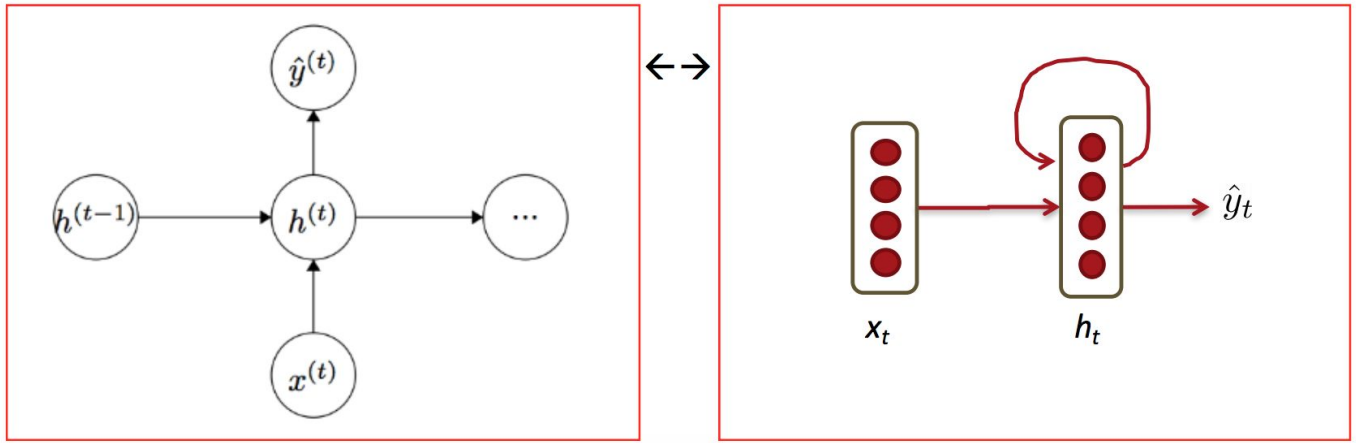
Given list of word **vectors:** $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$



Recurrent Neural Network-Forward prop

Main idea: we use the same set of W weights at all time steps!

Everything else is the same:

$$
\begin{aligned}
h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\
\hat{y}_t &= \text{softmax}\left(W^{(S)} h_t\right) \\
\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) &= \hat{y}_{t,j}
\end{aligned}
$$

$h_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer at time step 0

$x_{[t]}$ is the column vector of L at index [t] at time step t

Recurrent Neural Network-Loss Function

The cross-entropy in one single time step:

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Overall cross-entrophy cost:

$$J = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Where V is the vocabulary and T means the text.

Exp:

yt = [0.3,0.6,0.1] the, a, movie

y' = [0.001,0.009,0.9]
y* = [0.001,0.299,0.7]
y''= [0.001,0.9,0.009]

J1= - log 0.9
J2= - log 0.009]
J* = - log 0.7

J2 > J1
J* > J1

Recurrent Neural Network-Back Propagation

- Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\ \hat{y}_t &= W^{(S)} f(h_t) \end{aligned}$$

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Recurrent Neural Network-

Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

eg:

$$\frac{\partial h9}{\partial h6} = \frac{\partial h7}{\partial h6} * \frac{\partial h8}{\partial h7} * \frac{\partial h9}{\partial h8}$$

Remember: $h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$

More chain rule, remember:

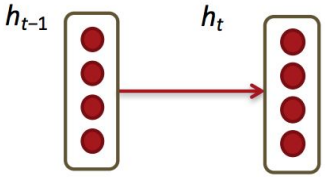
$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Recurrent Neural Network-

- From previous slide: $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$



- Remember: $h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$
- To compute Jacobian, derive each element of matrix: $\frac{\partial h_{j,m}}{\partial h_{j-1,n}}$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \mathrm{diag}[f'(h_{j-1})]$$

- Where: $\mathrm{diag}(z) = \begin{pmatrix} z_1 & & & \\ & z_2 & & 0 \\ & & \ddots & \\ & 0 & & z_{n-1} \\ & & & & z_n \end{pmatrix}$

Check at home that you understand the diag matrix formulation

Recurrent Neural Network-

Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

Where we defined $\bar{}$'s as upper bounds of the norms

The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

Recurrent Neural Network-

The solution first introduced by Mikolov is to clip gradients to a maximum value.

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Makes a big difference in RNNs.

SoftDev Interviews

Let’s recall.

Java: basic 1

OOP vs OBP: Object based programming languages (JavaScript) follow all the features of OOPs except Inheritance.

Constructor: can’t be final, not inherited.

Static: can be var, method, block, nested class

Inheritance: a mechanism in which one object acquires all the properties and behaviour of another object of another class. It represents IS-A relationship. Multiple is not supported.

Overloading: number of arguments, data type

```
void sum(int a,int b){System.out.println(a+b);}

void sum(double a,double b){System.out.println(a+b);}
```

Java: basic 2

Overriding:If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding: same name, same parameter.

Final: var: constant; method:can't be overriden; class:can't be inherited.

Abstract Class: A class that is declared with abstract keyword. Can have abstract/non-abstract methods. Can never be instantiated.

Abstract Method: declared abstract, does not have implementation.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Test it Now

```
running safely..
```

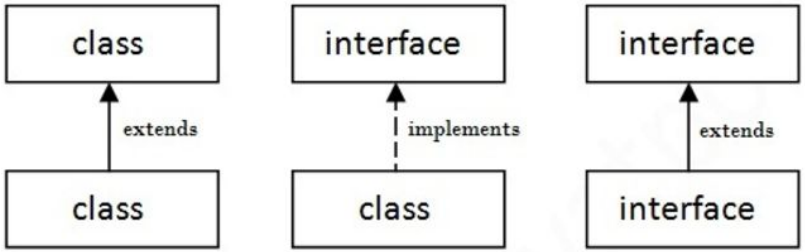
Java: basic 3

Interface: It has **static constants** and **abstract methods** only, cannot be instantiated just like abstract class.

Single inheritance….multiple..?

Abstract class vs Interface: here.

Var in interfaces: are implicitly public, cannot define private and proptected.



Java: basic 4

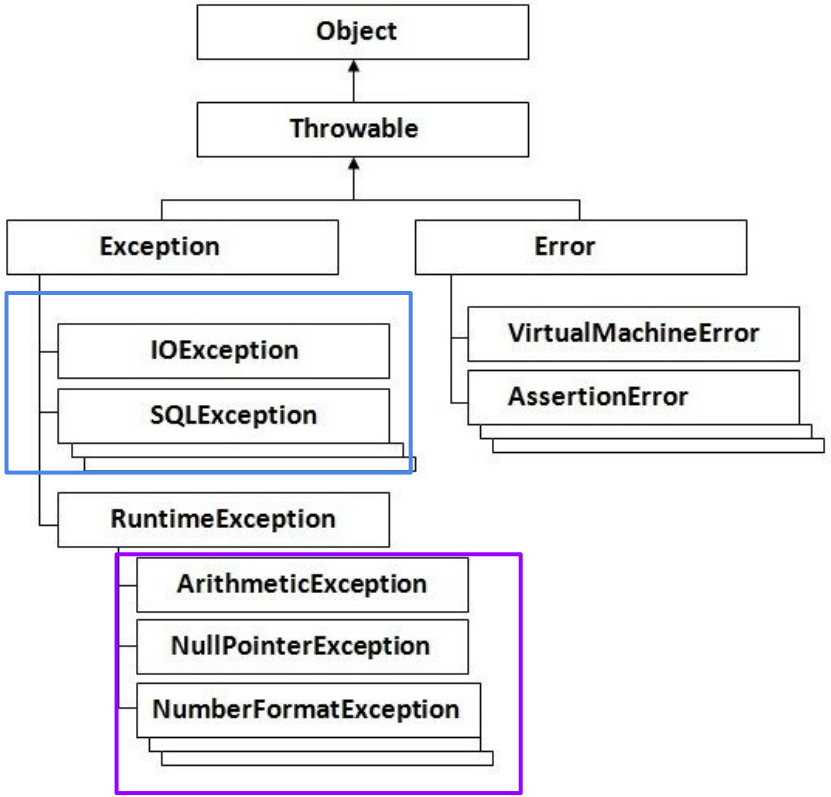
Type	Size in Bytes
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 byte
boolean	not precisely defined*

Java: exceptions 1

Checked excp: extends Throwable class except RuntimeException and Error. Checked at compile-time.

Unchecked excp: extendeds the RuntimeE.. Checked at run time.

Error.



Java: exceptions 2

Note: The finally block will not be executed if program exits(either by calling **System.exit()** or by causing a fatal error that causes the process to abort).

zero/many try,catch

Only one finally

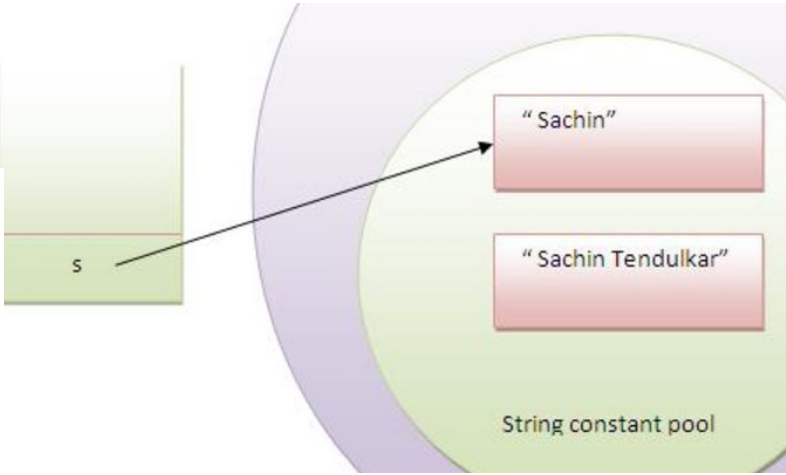
[Throw vs throws](#).

```
public static void main(String args[]){
try{
 int data=25/0;
 System.out.println(data);
}
catch(ArithmeticException e){System.out.println(e);}
finally{System.out.println("finally block is always execut
System.out.println("rest of the code...");
}
```

Java: string handling 1

Immutable: stores in string constant pool/ heap.

```
String s1="Welcome";
String s2="Welcome";//will not create new instance
```



Create via new..

```
String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

Java: string handling 2

String, stringbuffer and stringbuilder:

	String	*StringBuffer*	*StringBuilder*
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable)	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Performance	Fast	Very slow	Fast

Java: garbage collection 1

Garbage means unreferenced objects.

Garbage Collection: process of reclaiming the runtime unused memory automatically. …..it is a way to destroy the unused objects.

How can an obj be unreferenced? Here.

- a = null; //points to null
- assigning to another, the previous obj is done
- by annonymous object ??

The Garbage collector of JVM collects only those objects that are created by **new** keyword. So if you have created any object without new, you can use **finalize** method to perform cleanup processing (destroying remaining objects).

Java: garbage collection 2

```
public class TestGarbage1{
 public void finalize(){System.out.println("obje
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

```
object is garbage collected
object is garbage collected
```

gc(): invoke garbage collector. **Daemon thread**.This thread calls the **finalize()** method before object is garbage collected.

Note: Neither finalization nor garbage collection is guaranteed.

Java: garbage collection 3

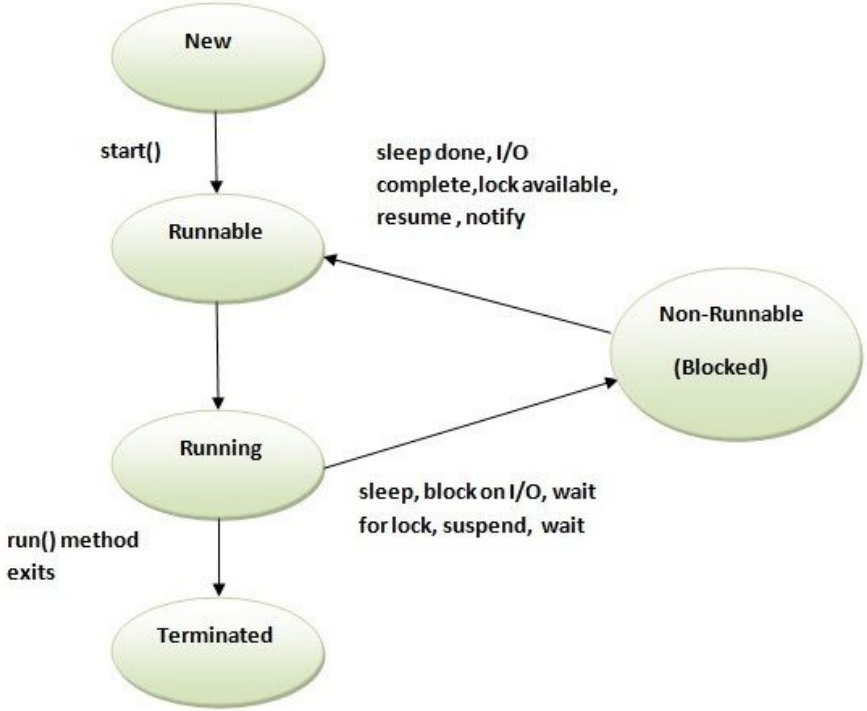
final: final is a keyword, final can be variable, method or class.You, can't change the value of final variable, can't override final method, can't inherit final class.

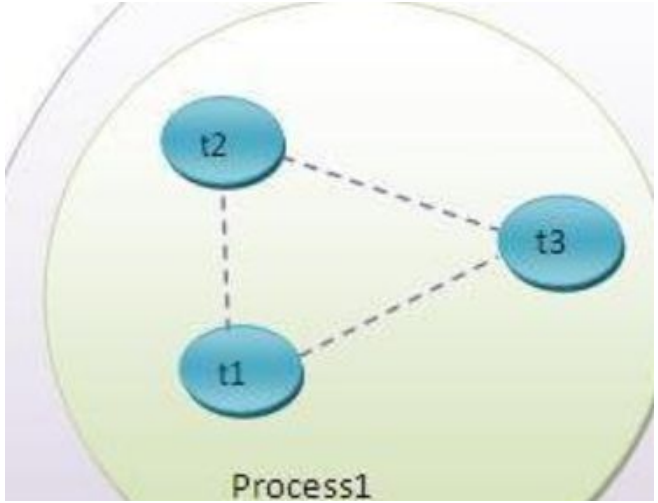
finally: finally block is used in exception handling. finally block is always executed.

finalize():finalize() method is used in garbage collection.finalize() method is invoked just before the object is garbage collected.The finalize() method can be used to perform any cleanup processing.

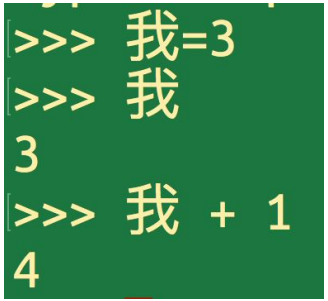
Java: thread

Thread: share the same address space, cost of communication is low.





Python: basic



Features: OOP, dynamic data types, auto-gc.

Standard data types: numbers, string,list, tuple, dictionary

lists vs tuples: lists...brackets [ ], can be updated; tuples...parentheses ( ), cannot be updated, “read-only” lists.

Python2 vs 3: here!

print不再是语句，而是函数，比如原来是 print 'abc' 现在是 print('abc')

整除: p3 int->float

字符串统一用Unicode，并且增加了专门储存字节数据的类

Python basic

compiler interpreter(VM)

py file -> bytecode -> binary code

pyc file is the byte code generate by python compiler. This process is triggerd by import statement.

Because it saves time when we need to frequently use a module, we don’t want to compile that module every time we import it.

Top-k Problem

Definition: Each record/query 1-255 bytes, 1billion (unique ~ 300 millions), memory 1G；find top-k frenquent records.

Step1: Stat

1) External sorting: External merge sort ….O(NlogN)
2) HashTable: traverse once, build the hashtable <Query, Count>....O(N)

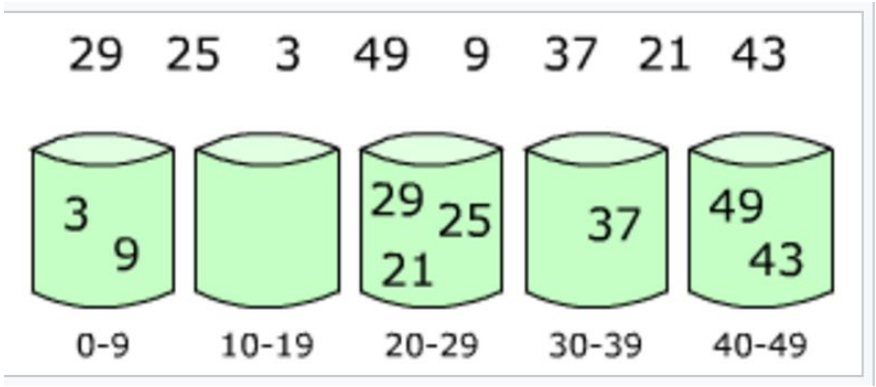
Step2: Find tops….Unique n

1) Sort …. O(nlogn)
2) Partial sort: keep an array of size k….O(n*k)
3) Heap sort: keep a heap of size k ….nlogk

Best: O(N)+O(nlogk)

Bucket Sort

1 Set up an array of initially empty "buckets".
2 **Scatter**: Go over the original array, putting each object in its bucket.
3 Sort each non-empty bucket.
4 **Gather**: Visit the buckets in order and put all elements back into the original array.



1) Stable
2) Fast..even than quicksort in practice
3) Space consuming

Worst-case performance	$O(n^2)$
Best-case performance	$\Omega(n+k)$
Average performance	$\Theta(n+k)$
Worst-case space complexity	$O(n \cdot k)$

Large scale problems (small memory)

1 billion int numbes, find out median.

- Bucket sort: 1) bucket sort: stats; 2) find out the median in the exact bucket.

1 billion int numbers, find out uniques.

- **Bitmap**: binary numbers, assign each number 2bits then:
  - 00: none
  - 01: once
  - 10: more than once
  - 11: --

Merge Sort

Motivation: how to merge two sorted arrays?

6 5 3 1 8 7 2 4

starts from the same side, two pointers

Divide & conquer

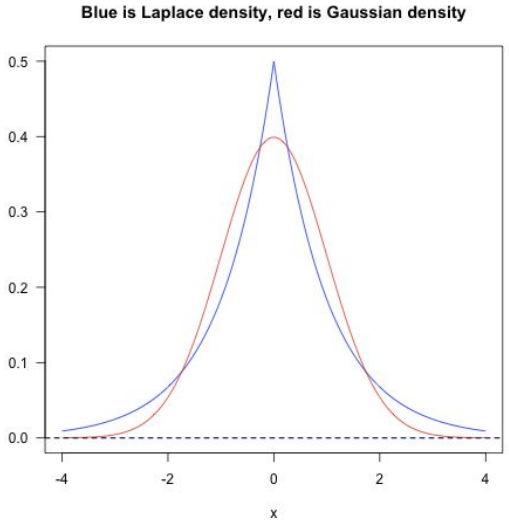
排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
希尔排序	$O(n \log n)$~ $O(n^2)$	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	O(1)	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$~ $O(n)$	不稳定

Regularization

As **regularizers**: (正则因子)

Standard L1 or L2 regularization on weights (w)

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} |w_i|^2$$



- Occam's razor.
- L2 is more sensitive to outliers, models leart are not sparse as uisng L1

Discriminative & Generative Model

Decision Function Y=f(X) or conditional probability P(Y|X)

Generative Model: learn P(X,Y), calculate posterior prob dist P(Y|X):

$$P(Y \mid X) = \frac{P(X,Y)}{P(X)}$$

HMM, Bayesian Classifier

Discriminative Model: learn P(Y|X) directly??

DT, NN,SVM,Boosting,CRF

Metrics - Classification

Accuracy

Precision & Recall (generally binary classification)

$$P = \frac{TP}{TP + FP}$$

TP(正->正)　FP(负->正)

FN(正->负)　TN(负->负)

$$R = \frac{TP}{TP + FN}$$

F1 score

P & R are high, F1 is high.

$$\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R}$$

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

Metrics - Regression

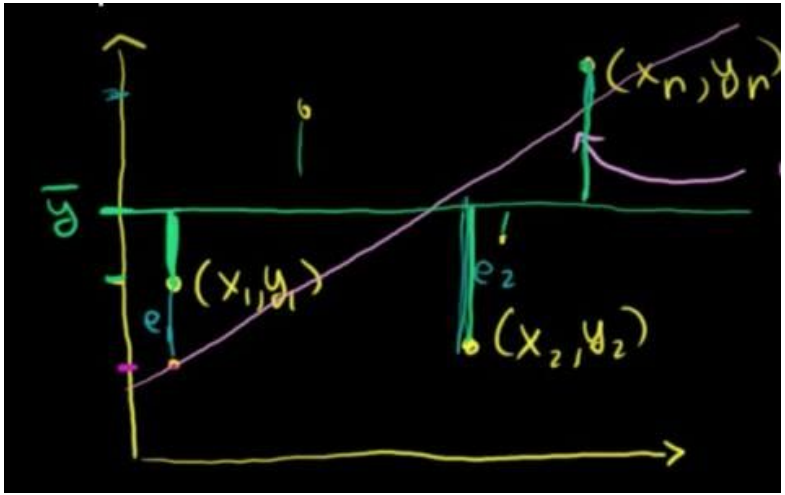
Mean Absolute Errors & M. Square E.

$$MAE(y, \hat{y}) = \frac{1}{n_samples} \sum |y_i - \hat{y}_i|$$
$$MSE(y, \hat{y}) = \frac{1}{n_samples} \sum |y_i - \hat{y}_i|^2$$

R-squared error (coefficient of determination):

$$R^2(y, \hat{y}) = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y}_i)^2}$$

A higher value means a good model.



Ensemble Learning

Strong learning method: learns model with good performance;

Week learning method: learns model slightly better than random.

Ensemble learning methods:

- Iterative method: Boosting
- Parallel method: Bagging

Boosting: algorithm

Boosting: gather/ensemble week learners to become a strong one.

Through majority voting on classification; average on regression.

Algo:

1) Feed N data to train a week learner/ model: h1.
2) Feed N data to train another one: h2; N pieces of data including: h1 errors and new data never trained before.
3) Repeat to train hn, N includes previous errors and new data.
4) Get final model: h_final=Majority Vote(h1,h2,..,hn)

Problems: ensemble methods.

AdaBoost: adaptive boosting

Key: focus more on the mistakes. Why not overfitting?

Bagging: Bootstrap AGGregatING

Bootstrap sampling: take k/n with replacement at each time. Samples are overlapped. Appro. 36.8% would not be selected. If we sample m times:

$$\lim_{m\to\infty}(1-\frac{1}{m})^m=\frac{1}{e}\approx 0.368$$

Training: in parallelization.

Ensemble: voting on classification; average on regression.

Random Forest: a variant of Bagging.

Random Forest: random attribute selection

Bagging: select a number of data to train each DT.

Random Attribute Selection

Each tree, each node, select k attributes, then find out the optimum.

Emperical k:

$$k = \log_2 d$$

where d is the total number of attributes.

GloVe: Overview

Word-word Co-occurance Matrix, X : V * V (V is the vocab size)

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Xij: the number of times word j occurs in the context of word i.

Xi = $\sum_k X_{ik}$: the number of times any word appears in the context of word i.

Pi j = P(j|i) =Xi j/Xi : probability that word j appear in the context of word i.

P(solid | ice) = word “solid” appear in the context of word “ice”.

GloVe: Global Vectors for Word Representation

Ratios of co-occur: define a F

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Linearity:

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} .$$

Vector:

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}}$$

Symmetric ???

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} ,$$

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i} .$$

The solution to Eqn. (4) is $F = \exp$, or,

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i) .$$

GloVe

Add bias: log(Xi) independent from k,add as bi

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

Square error ???

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

GloVe: f(x) as weights for words

f(0)=0

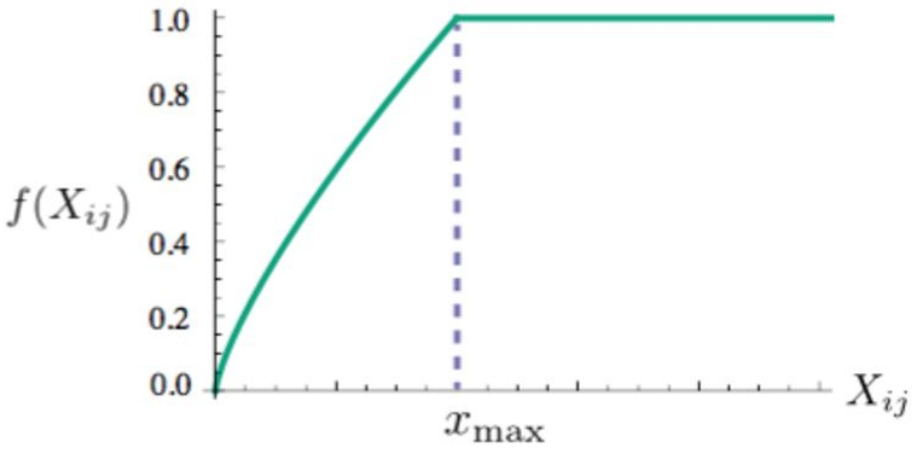
monotonically increasing

large x, small f(x)

Usually,

xmax=100,*α*=3/4

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} .$$

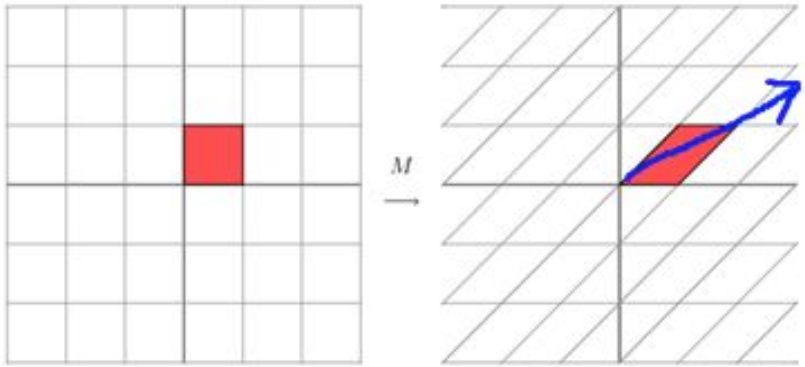


SVD: Singular Value Decomposition

Starts from Matrix Multiplication: Y = A*B

Check codes and plots here.

Want to find the direction & extend:



SVD: Singular Value Decomposition

Eigen-Value & Eigen-Vector:

$$Av = \lambda v$$

Have more than one pairs.

Import numpy to calculate. =>

But only for square matrix!

```
from numpy import linalg as LA
```

```
t = np.diag((1, 2, 3))
print (t)
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

```
w,v = LA.eig(t)
print ("E-value{}\nE-vec{}".format(w,v))
```

```
E-value[ 1.  2.  3.]
E-vec[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

SVD: Singular Value Decomposition

If A has n e-values:

$$A * x_1 = \lambda_1 * x_1$$
$$A * x_2 = \lambda_2 * x_2$$
$$\dots$$
$$A * x_n = \lambda_n * x_n$$

Then:

Q

Q

Sigma

$$A * \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix} * \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

So, *AQ=QSigma*

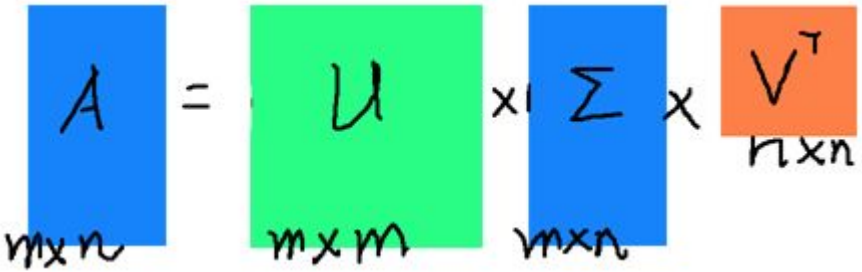
$$A = Q \Sigma Q^{-1}$$

SVD: Singular Value Decomposition

For the non-square matrix? A: m * n.

Similar idea:

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



But how to get e-values and e-vectors?

SVD: Singular Value Decomposition

Find a square-matrix!

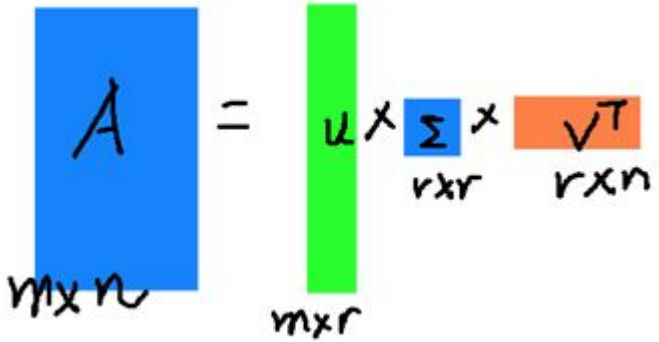
$$(A^T A)x = \lambda x$$

Calculate… then:

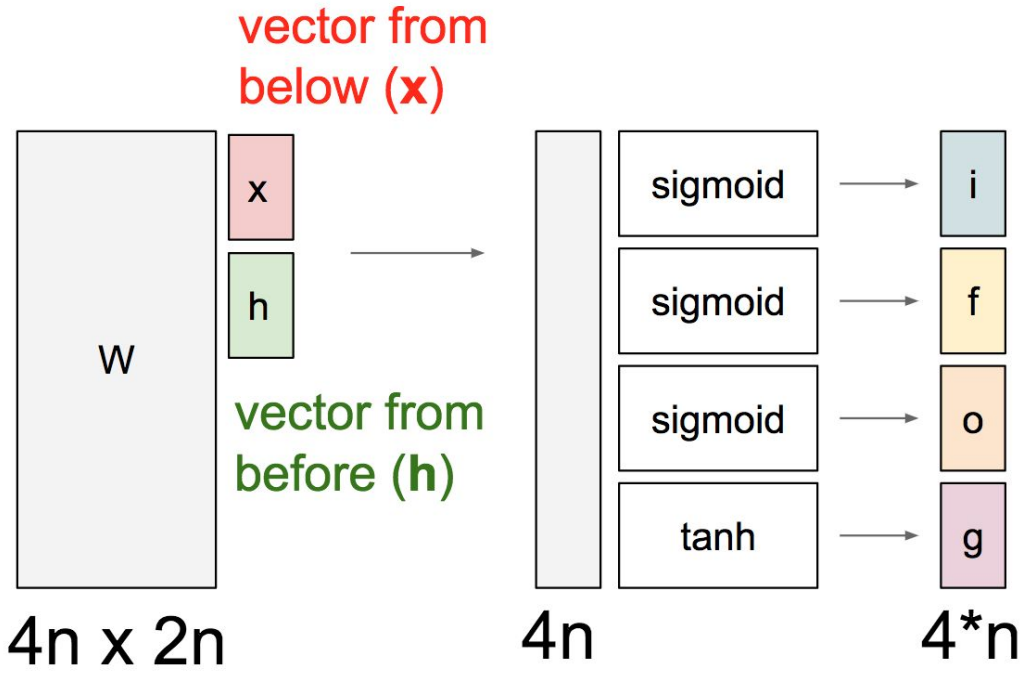
$$A_{m\times n} \approx U_{m\times r}\Sigma_{r\times r}V^T{}_{r\times n}$$

let r << # of e-value to represent A

O(n^3)



硫酸铜镁(lstm)



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

Points

Decorations:

- Math equations
- Style: latex?
-

Contents:

- Supervised: The Perceptron algorithm
- Loss Functions: P34
-

Links

ML优缺点（中文）:http://www.kuqin.com/shuoit/20140927/342357.html

Large-Scale ML: http://www.cs.nthu.edu.tw/~shwu/courses/ml/

Sorting: http://wuchong.me/blog/2014/02/09/algorithm-sort-summary/

NLP: https://medium.com/@frank_chung/notes-for-deep-learning-on-nlp-94ddfcb45723#.yc1zcduud

Reinforcement: https://github.com/0bserver07/Study-Reinforcement-Learning

LDA(中文):http://emma.memect.com/t/9756da9a47744de993d8df13a26e04e38286c9bc1c5a0d2b259c4564c6613298/LDA (Gamma, Dirichlet Dist, MCMC, Gibbs Sampling, Topic Model, PLSA, LDA)