CSE 5717

BIG DATA ANALYTICS

EXAM IV REVIEW

TOPICS:

* RANDOMIZED ALGORITHMS
* PARALLEL ALGORITHMS
* OUT-OF-CORE COMPUTING
  - SINGLE & MULTIPLE DISKS
* STRING ALGORITHMS
  - SUFFIX TREES, SUFFIX ARRAYS
* RULES MINING
* POLYNOMIAL ARITHMETIC
* ML: LINEAR REGRESSION, PERCEPTRONS, NEURAL NETWORKS

RANDOMIZED ALGORITHMS

MONTE CARLO

LAS VEGAS

EX. INPUT: AN ARRAY $A[1:n]$

ONE element has $\frac{n}{2}$ copies.

The others are distinct

Output: Repeated element.

A Las Vegas Alg

Repeat

Basic Step

Flip an n-sided coin to get i;

" to get j;

if $i \neq j$ and $A[i] = A[j]$

then output $A[i]$ and quit;

Forever

$\tilde{O}(f(n)) \Rightarrow$ Run time is $\leq c\alpha f(n)$ with a prob. of $\geq (1-n^{-\alpha})$, for all $n \geq n_0$, where c and n_0 are constant.

$\Rightarrow$ RUN TIME of the above alg. is $\tilde{O}(\log n)$.

By low prob. we mean
a prob. of $\le n^{-\alpha}$, α being
the prob. parameter.

PARALLEL ALGORITHMS:

P = # of Processors.

S = Best known Seq. Run time to solve a problem Π.

Let T be the ||l RUN TIME.

FACT: $T \geq \frac{S}{P}$.

PARALLEL MODELS

FIXED CONNECTION M/C

SHARED MEMORY

Fixed Connection M/C $\rightarrow$ A Directed Graph $G(V,E)$.

SHARED MEMORY MODEL

↓

PRAM

↓

EREW

↓

CREW

↓

CRCW

- COMMON
- ARBITRARY
- PRIORITY

(1) PROBLEM:

INPUT: $b_1, b_2, \ldots, b_n$

OUTPUT: $b_1 \wedge b_2 \wedge \ldots \wedge b_n$.

FACT: We can solve this in $O(1)$ time using n COMMON CRCW PRAM PROCESSORS

COROLLARY: We can Find the MIN or Max of n ARBITRARY Real #'s in $O(1)$ time using n^2 Common CRCW PRAM PROC.

PREFIX COMP.

INPUT: $x = K_1, K_2, \dots, K_n \in \Sigma$.

$\oplus$ IS ASSOCIATIVE, UNIT TIME, & BINARY.

OUTPUT: $K_1, K_2 \oplus K_2, K_1 \oplus K_2 \oplus K_3, \dots,$
$K_1 \oplus K_2 \oplus \dots \oplus K_n$.

INPUT: $X = k_1, k_2, \ldots, k_n;\ K$

Outputs Rank $(K, X) \triangleq 1 + |\{q \in X : q < K\}|$

A

CREW

$A[i] = 1$ IF $k_i < K$.

$= 0$ Otherwise

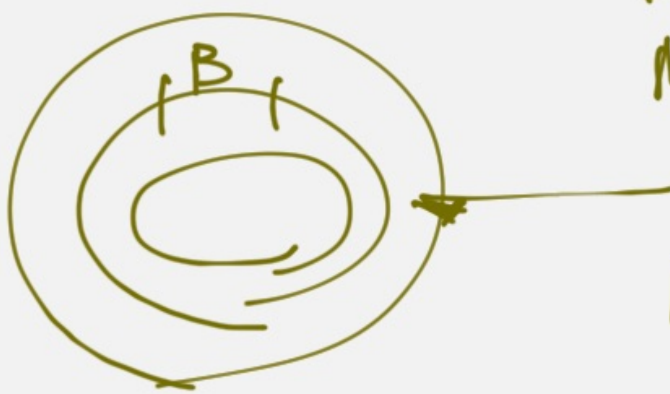
Do a prefix sums on $A[1:n]$.

SLOW-DOWN LEMMA:

IF A PR ALG. RUNS IN TIME T USING P PROCESSORS, the same alg. can be simulated on a P' processor M/C in $O\left(\frac{PT}{P'}\right)$ TIME, FOR ANY $P' \le P$.

OUT_OF_CORE ALGORITHMS:

SORTING ON A SINGLE DISK.



B → BLOCK SIZE

M → CORE MEMORY SIZE.

INPUT SIZE = n.

Sorting needs $\Omega\left(\frac{n}{B}\frac{\log(n/M)}{\log(M/B)}\right)$ I/O operations.

We can sort in $O\left(\frac{n}{B}\frac{\log(n/M)}{\log(M/B)}\right)$ I/O operations.

IDEA:

1. FORM RUNS of length M EACH. → ONE PASS.

2. USE $\left(\frac{M}{B}\right)$-way MERGE to MERGE the $\frac{n}{M}$ RUNS.

height

$$= \frac{\log(N/M)}{\log(M/B)}$$

RANDOMIZED Selection: $X = k_1, k_2, \ldots, k_n$; i

* Pick a Random Sample S.
* identify two elements l_1, l_2 s.t. ① The ith smallest element of $X \in [l_1, l_2]$; and ② $|\{l_1 \le q \le l_2, q \in X\}|$ is "small".
* IDentify $Y = \{q \in X : l_1 \le q \le l_2\}$.
* Perform an approp. selection in Y.

Lemma: Let X be a seq. of n elements.

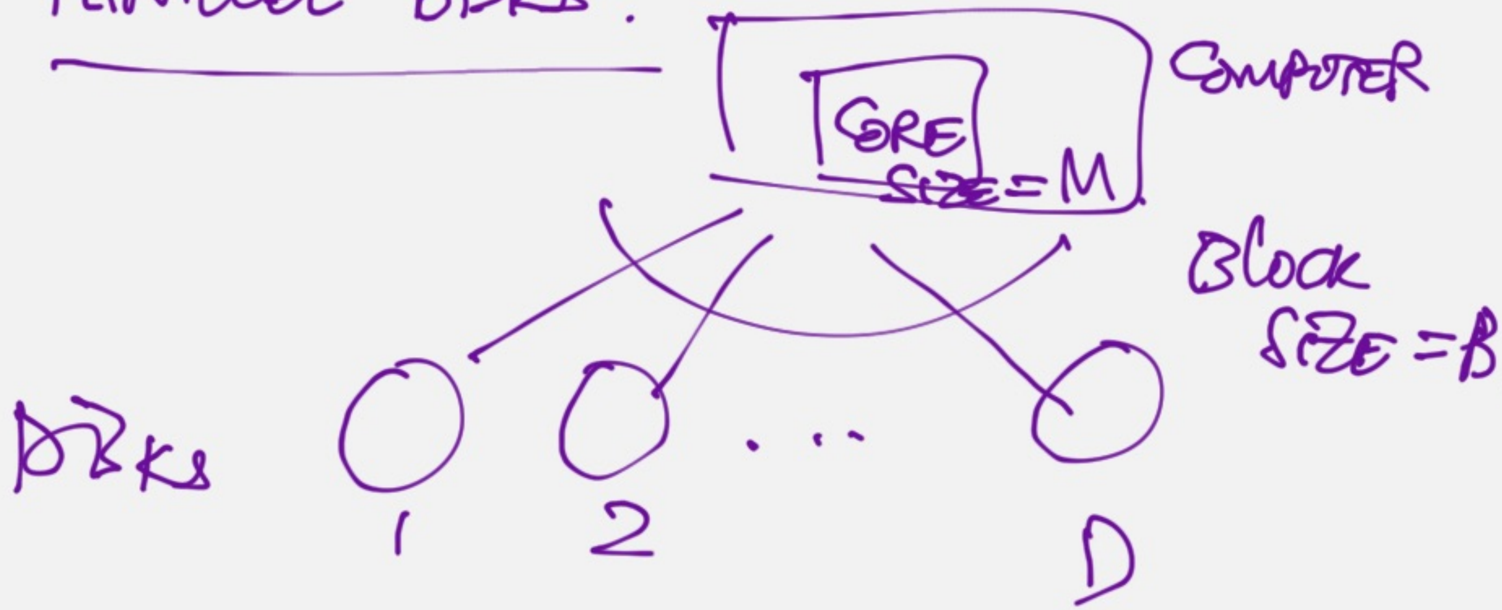
Pick a sample S with $s = |S|$.

Let q be an element of S s.t.

Rank $(q, S) = j$. Let r_j be the rank of q in X.

$$\text{Prob}\left[\left|r_j - j\frac{n}{s}\right| > \sqrt{3\alpha}\,\frac{n}{\sqrt{s}}\sqrt{\log n}\right] < n^{-\alpha}.$$

PARALLEL DISKS:

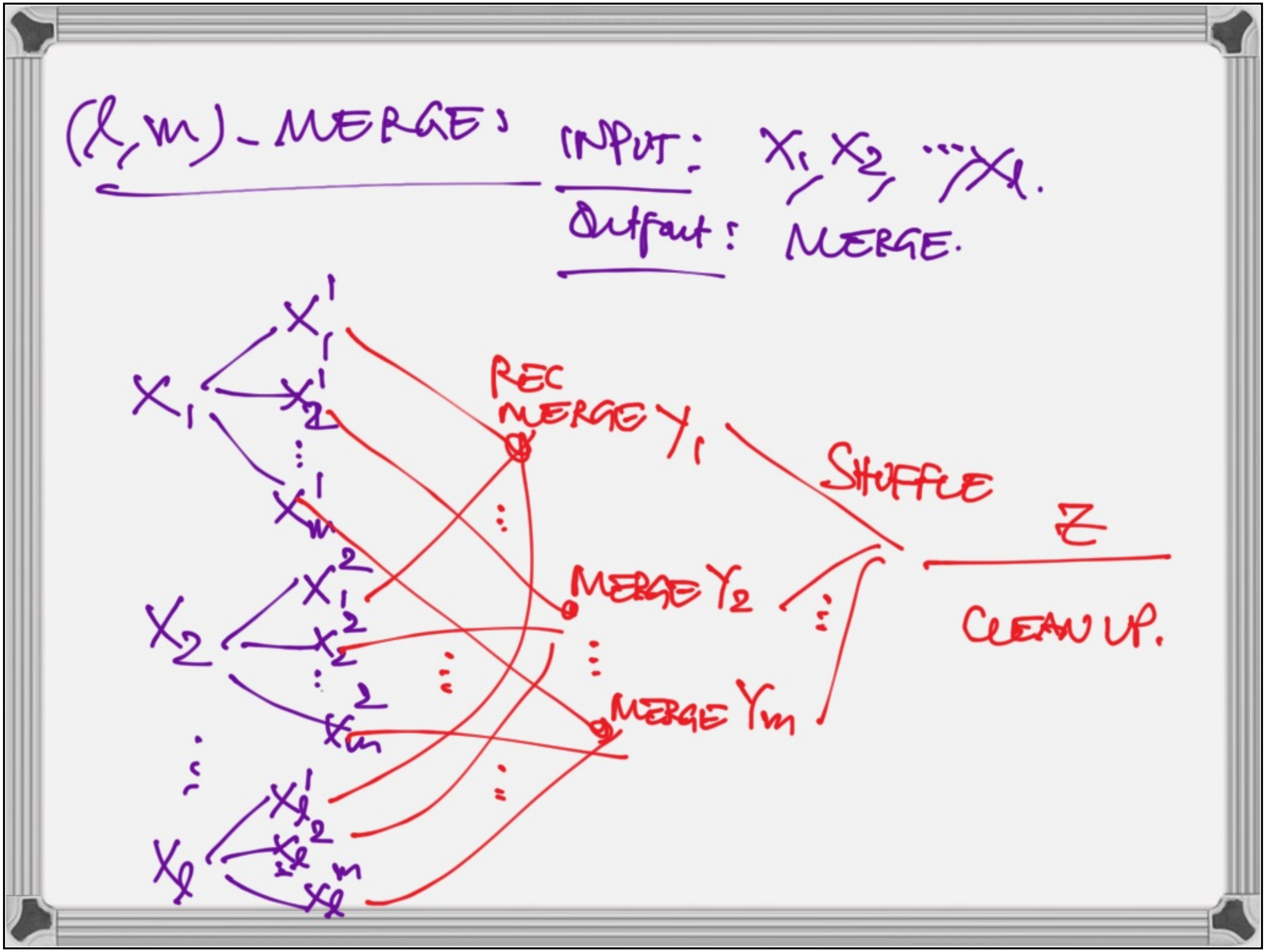


IN ONE ||el I/O WE CAN BRING ONE BLOCK FROM EACH OF THE DISKS.

(ℓ, m)-MERGE SORT.

$X = k_1, k_2, \ldots, k_n$.

* Partition X into ℓ sequences $X_1, X_2, \ldots, X_\ell$.
* Sort each X_i recursively
* MERGE them using the (ℓ, m)-MERGE alg.


(l, m)-MERGE:
INPUT: X1, X2, ..., Xl.
Output: MERGE.
REC MERGE Y1
MERGE Y2
MERGE Ym
SHUFFLE
Z
CLEAN UP.

CLEAN UP:

WE CAN SORT n elements in

$$\ell\left(\frac{n}{BD}\cdot\frac{\log(n/M)}{\log\min\{\sqrt{m}, M/B\}}+1\right)^2$$

I/O operations.

STRING ALGORITHMS:

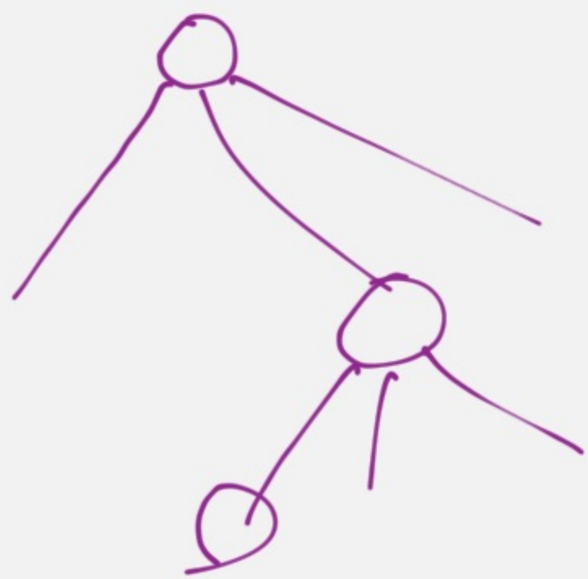
SUFFIX TREES:

STRING MATCHING:

INPUT: Text T; PATTERN P.

Output: all the occurrences of P in T.

If $|T|=m$, $|P|=n$, we can solve this in $O(m+n+k)$ time where k is the # of matches.



INPUT: S_1, S_2

Output: Longest Common Substring

Algorithm: (1) Construct a GST Q on S_1 & S_2

Time = $O(|S_1| + |S_2|)$.

(2) Traverse Q & label a node with 1 if the subtree rooted @ this node has a suffix from S_1;

(3) Traverse Q & label a node with 2 if the subtree rooted @ this node has a suffix from S_2;

(4) Traverse Q & identify the node u that has labels 1 and 2 & whose string depth is the largest.

(5) Output the path label of u;

RUN TIME = $O(|S_1|+|S_2|)$.

LEMMA: We can sort n integers in the range $[1, n^c]$ in $O(n)$ time for any constant c

We showed that a suffix array can be constructed in $O(n)$ time, n = length of the input string.

FACT: String matching can be done in $O(n + \log m)$ time using a suffix array.

ASSOCIATION RULES MINING.

I = set of possible items.

an itemset $\subseteq I$; A transaction $\subseteq I$.

INPUT: A DB of transactions.

Output: Rules of the kind: $A \rightarrow B$

where $A \neq \phi$, $B \neq \phi$, $A \subseteq I$, $B \subseteq I$

$A \cap B = \phi$.

$\sigma(X)$ = # of transactions in which X occurs.

Support for $A \rightarrow B$ is $\frac{\sigma(A \cup B)}{n}$.

Confidence for $A \rightarrow B$ is $\frac{\sigma(A \cup B)}{\sigma(A)}$.

Output should be all the Rules $A \rightarrow B$ s.t. Support$(A \rightarrow B) \geq$ minSupport & Confidence$(A \rightarrow B) \geq$ minConfidence.

An itemset X is frequent

if $\sigma(X) \geq n.$ min support.

IDEA: First generate all the frequent itemsets.

From each freq. itemset, generate relevant rules.

To identify frequent k-itemsets:

Assume that each transaction is a bit array of size d, $d = |I|$.

There are $\binom{d}{k}$ possible k-itemsets.

Run time = $O\left(\binom{d}{k} n k\right)$

$n = |DB|$.

APRIORI PRINCIPLE:

* If X is Frequent, then every subset of X is also FREQ.
* If X is not Freq, then no superset of X can be FREQUENT.

APRIORI ALG.

Let F_k be the set of Frequent k-itemsets.

Let C_k be the set of Candidate Freq. k itemsets.

Compute F_1; $k=1$

Repeat

Generate C_{k+1} FROM F_k;

Compute Support for members of C_{k+1} & Generate F_{k+1};

$k = k+1$

Until $F_k = \phi$.

let $Q \in C_{k+1}$

PRUNING: If q is a k-subset of Q that is not in F_k then PRUNE Q FROM C_{k+1};

RANDOMIZED FREQ. itemsets MINING:

Pick a Random Sample S;

Identify FREQ. itemsets in S using a smaller support;

Polynomial ARITHMETICS

INPUT: degree n - polynomials $f(x)$ & $g(x)$.

Output: The product $f(x)\, g(x)$.

We need the coefficients.

We converted $f(x)$ & $g(x)$ into values FORM; Computed the product in the values FORM; Convert the product into Coefficients FORM.

Choose the points to be N^{th} Roots of unity $N \geq (2n+1)$.

$\omega^0, \omega^1, \omega^2, \ldots, \omega^{2n+1}$

where $\omega = e^{2\pi i/N} = \cos\left(\frac{2\pi}{N}\right) + i \sin\left(\frac{2\pi}{N}\right)$

PRIMITIVE N^{th} Root of unity.

We can evaluate a polynomial at

the N^{th} roots of unity
in $O(N \log N)$ time using
the DFT alg.

Interpolation can also be done
in $O(N \log N)$ time.

$\Rightarrow$ We can multiply two degree-n
polynomials in $O(n \log n)$ time.

MACHINE LEARNING:

$$f: \mathbb{R}^m \to \mathbb{R}^n.$$

Input: Examples: $E_1, E_2, \ldots, E_q$

E_i: $(\vec{x}_i, \vec{y}_i)$ s.t. $f(\vec{x}_i) = \vec{y}_i$

$\vec{x}_i \in \mathbb{R}^m$; $\vec{y}_i \in \mathbb{R}^n$.

Output: A guess for f.

GRADIENT DESCENT:

$f: \mathbb{R} \rightarrow \mathbb{R}$

AN iterative alg.:

Start with $x = x_0$; $i = 0$;

Repeat

$x_{i+1} = x_i - \epsilon \ \mathrm{sign}(f'(x_i))$; $i = i+1$;

until $f(x_i) = 0$

LINEAR REGRESSION:

$$f: \mathbb{R}^n \to R.$$

$$f(x_1, x_2, \dots, x_n) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n.$$

INPUT:

$$(x_1^1, x_2^1, \dots, x_n^1, y_1)$$

$$(x_1^2, x_2^2, \dots, x_n^2, y_2)$$

$$\vdots$$

$$(x_1^m, x_2^m, \dots, x_n^m, y_m).$$

$$X = \begin{bmatrix} x_1^1, & x_2^1, & \cdots, & x_n^1 \\ & \vdots & & \\ x_1^m, & x_2^m, & \cdots, & x_n^m \end{bmatrix} \quad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

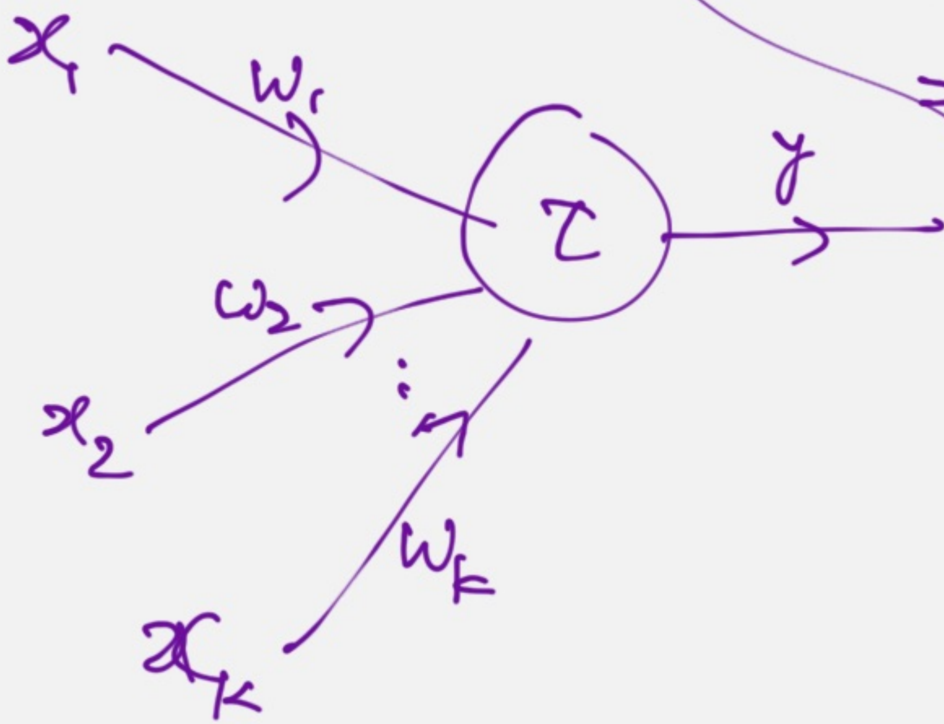
$$\vec{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T$$

$$\vec{w} = (X^T X)^{-1} X^T \vec{y}$$

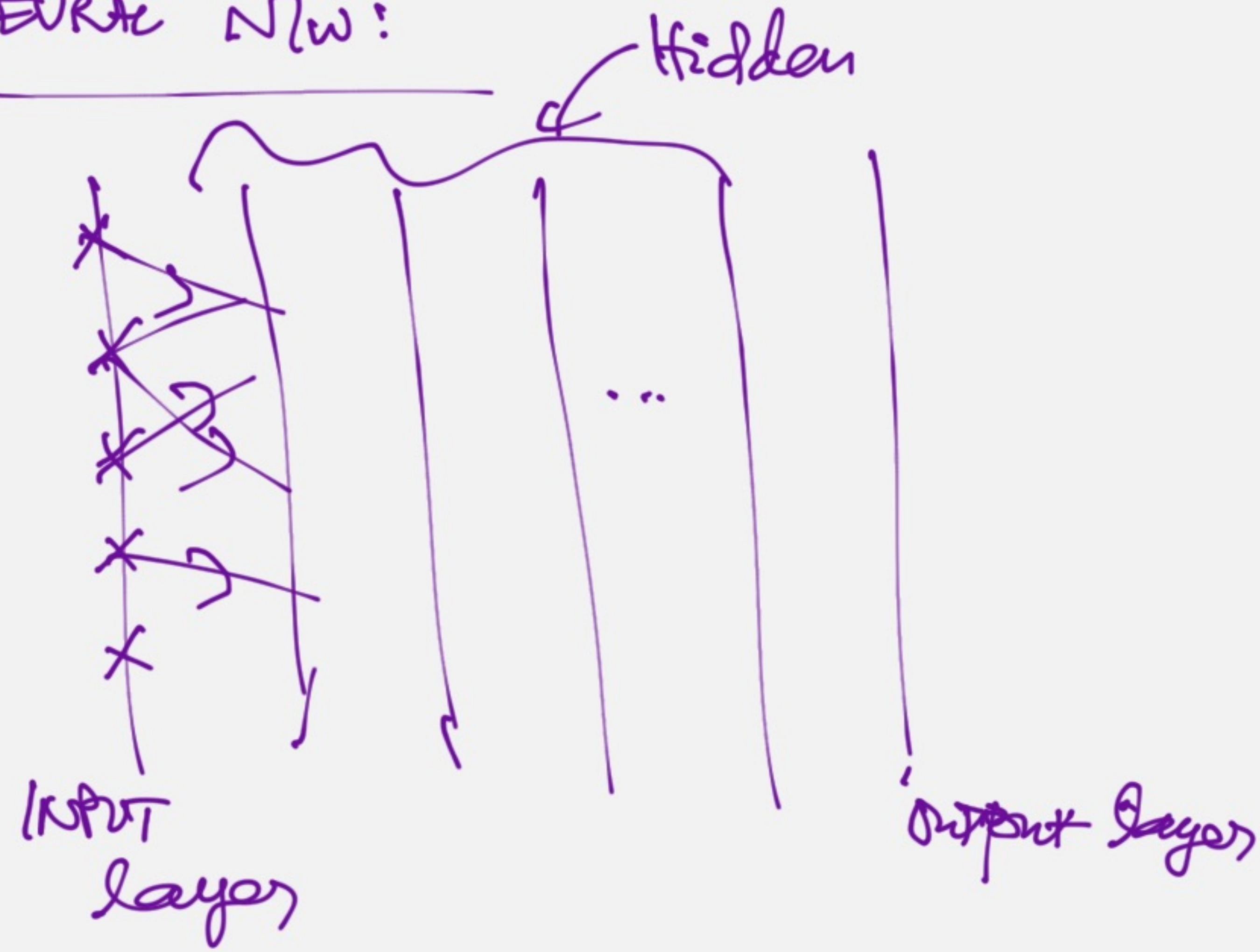
PERCEPTRONS:

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{k} w_i x_i \geq \tau$$

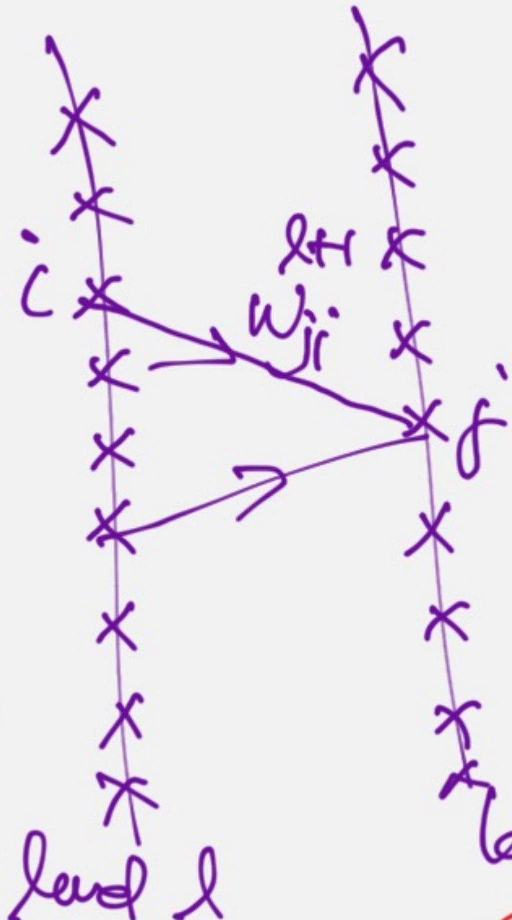
$$= 0 \quad \text{otherwise}$$



NEURAL NTW:



a_i^l → output of node i in level l



weighted input at node j of level $(l+1)$

$$z_j^{l+1} = \sum_{i=1}^{n_l} w_{ji}^{l+1} a_i^l + b_j^{l+1}$$

$$a_j^{l+1} = \sigma\left(z_j^{l+1}\right).$$

σ → Activation function

FACT. We can do Forward & back propagations in $O(n^2 L)$ time, where L = # of layers, and n = # of neurons in each layer. We can also do this in $O(|V| + |E|)$ time where $G(V,E)$ represents the Neural Network.