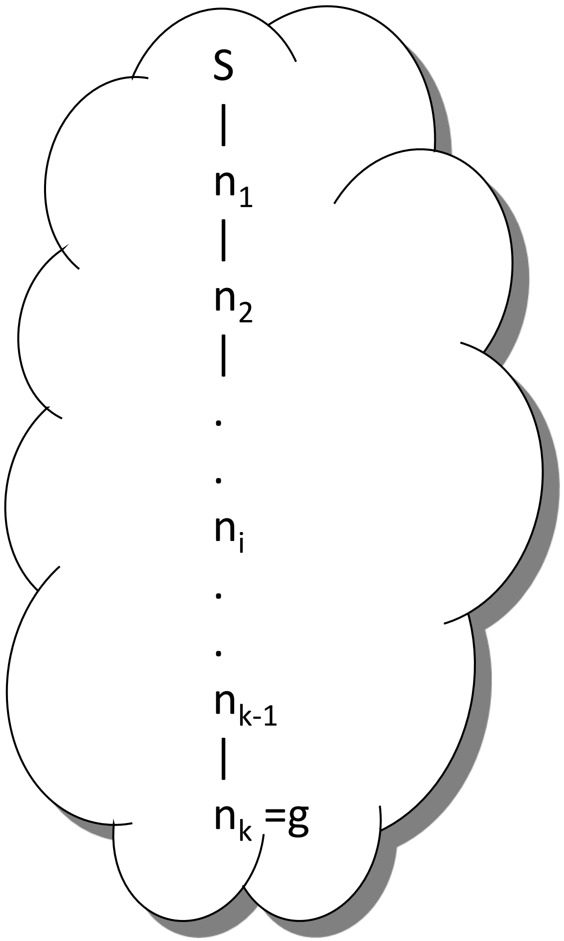
CS217: Artificial Intelligence and Machine Learning (associated lab: CS240)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Week3 of 20jan25, Monotone Restriction, Perceptron

Main points covered: week2 of 13jan25

Key point about A* search



<u>Statement</u>:

Let S $-n_1-n_2-n_3 \ldots n_i \ldots -n_{k-1}-n_k (=G)$ be an optimal path.

At any time during the search:

1. There is a node n_i from the optimal path in the OL
2. For n_i all its ancestors $S, n_1, n_2, \ldots, n_{i-1}$ are in CL
3. $g(n_i) = g^*(n_i)$

Admissibility of A*

- 1. A* algorithm halts
- *2.* A* algorithm finds optimal path
- 3. If $f(n) < f^*(S)$ then node n has to be expanded before termination
- 4. If A* does not expand a node n before termination then $f(n) >= f^*(S)$

Better heuristic performs better

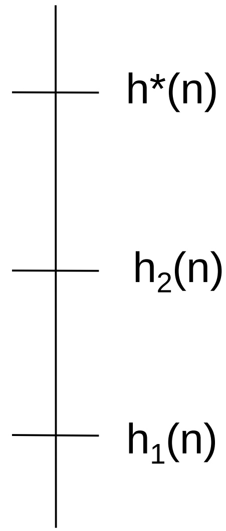
A version A_2* of A* that has a "better" heuristic than another version A_1* of A* performs at least "as well as" A_1*

<u>Meaning of "better"</u>

$h_2(n) > h_1(n)$ for all n

<u>Meaning of "as well as"</u>

A_1* expands at least all the nodes of A_2*



For all nodes n, except the goal node

Foundational Ideas

- Church Turing Hypothesis
- Physical Symbol System Hypothesis
- Uncomputabiliity
- NP-completeness and NP-hardness
- AI is multidisciplinary
- Difference between brain computation and Turing Machine

End of main points

Monotonicity

Steps of GGS
(*principles of AI, Nilsson,)*

- 1. Create a search graph *G*, consisting solely of the start node *S*; put *S* on a list called *OPEN.*
- *2.* Create a list called *CLOSED* that is initially empty.
- 3. Loop: if *OPEN* is empty, exit with failure.
- 4. Select the first node on *OPEN*, remove from *OPEN* and put on *CLOSED*, call this node *n*.
- 5. if *n* is the goal node, exit with the solution obtained by tracing a path along the pointers from *n* to *s* in *G*. (ointers are established in step 7).
- 6. Expand node *n*, generating the set *M* of its successors that are not ancestors of *n*. Install these memes of *M* as successors of *n* in *G*.

GGS steps (contd.)

- 7. Establish a pointer to *n* from those members of *M* that were not already in *G* (*i.e.*, not already on either *OPEN* or *CLOSED*). Add these members of *M* to *OPEN*. For each member of *M* that was already on *OPEN* or *CLOSED*, decide whether or not to redirect its pointer to *n*. For each member of M already on *CLOSED*, decide for each of its descendents in *G* whether or not to redirect its pointer.
- 8. Reorder the list *OPEN* using some strategy.
- 9. Go *LOOP.*

Illustration for CL parent pointer redirection recursively

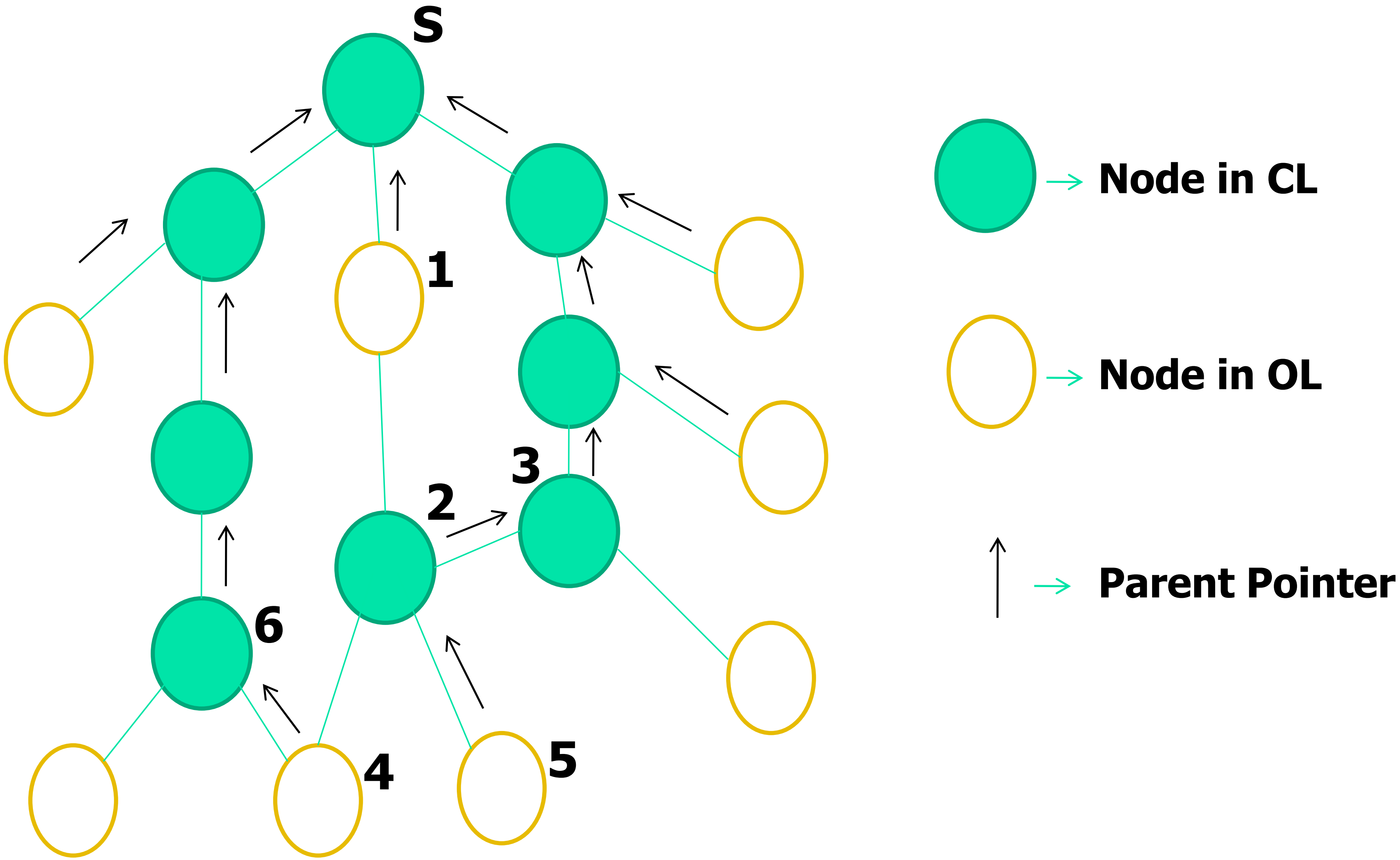
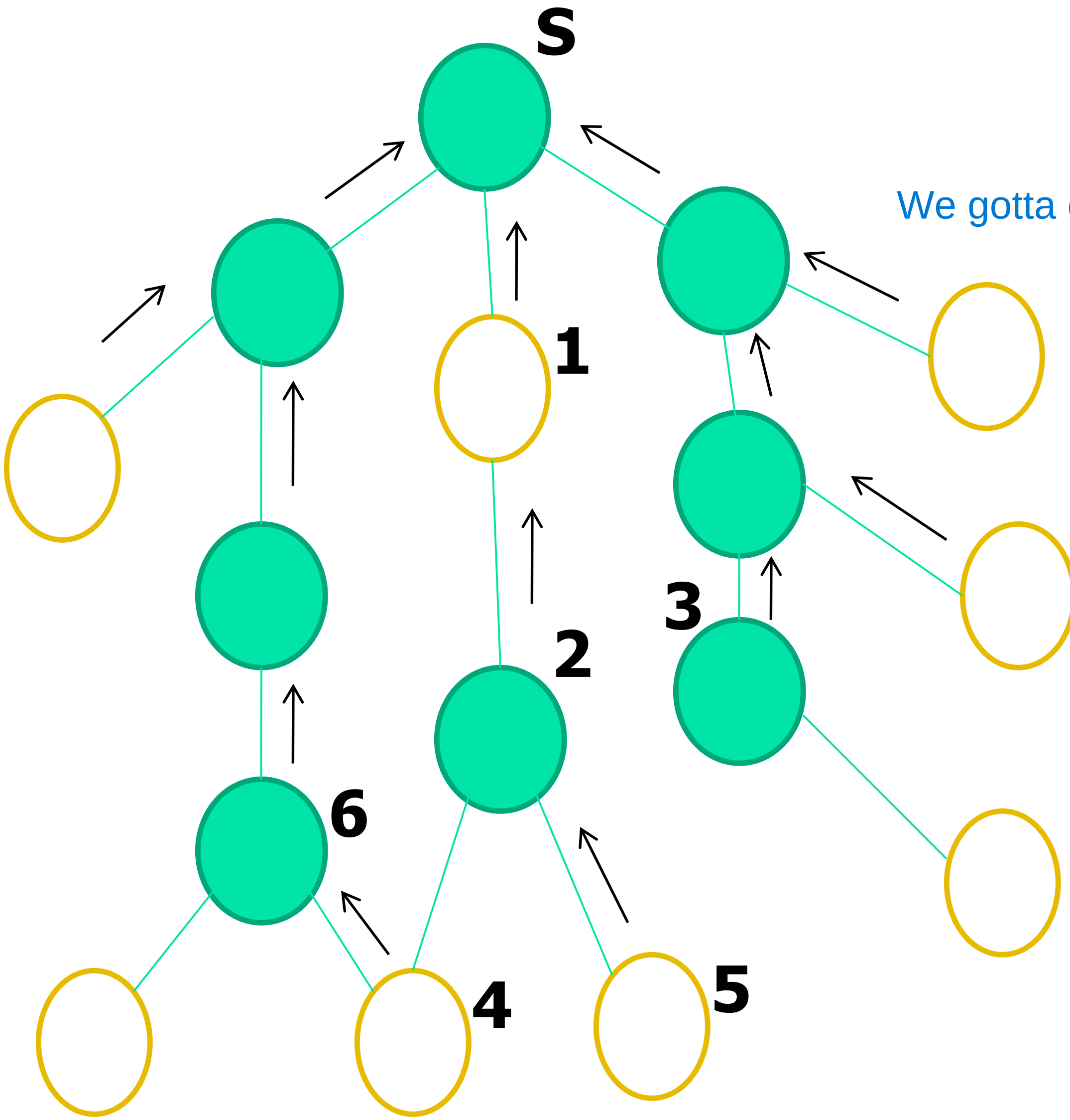


Illustration for CL parent pointer redirection recursively



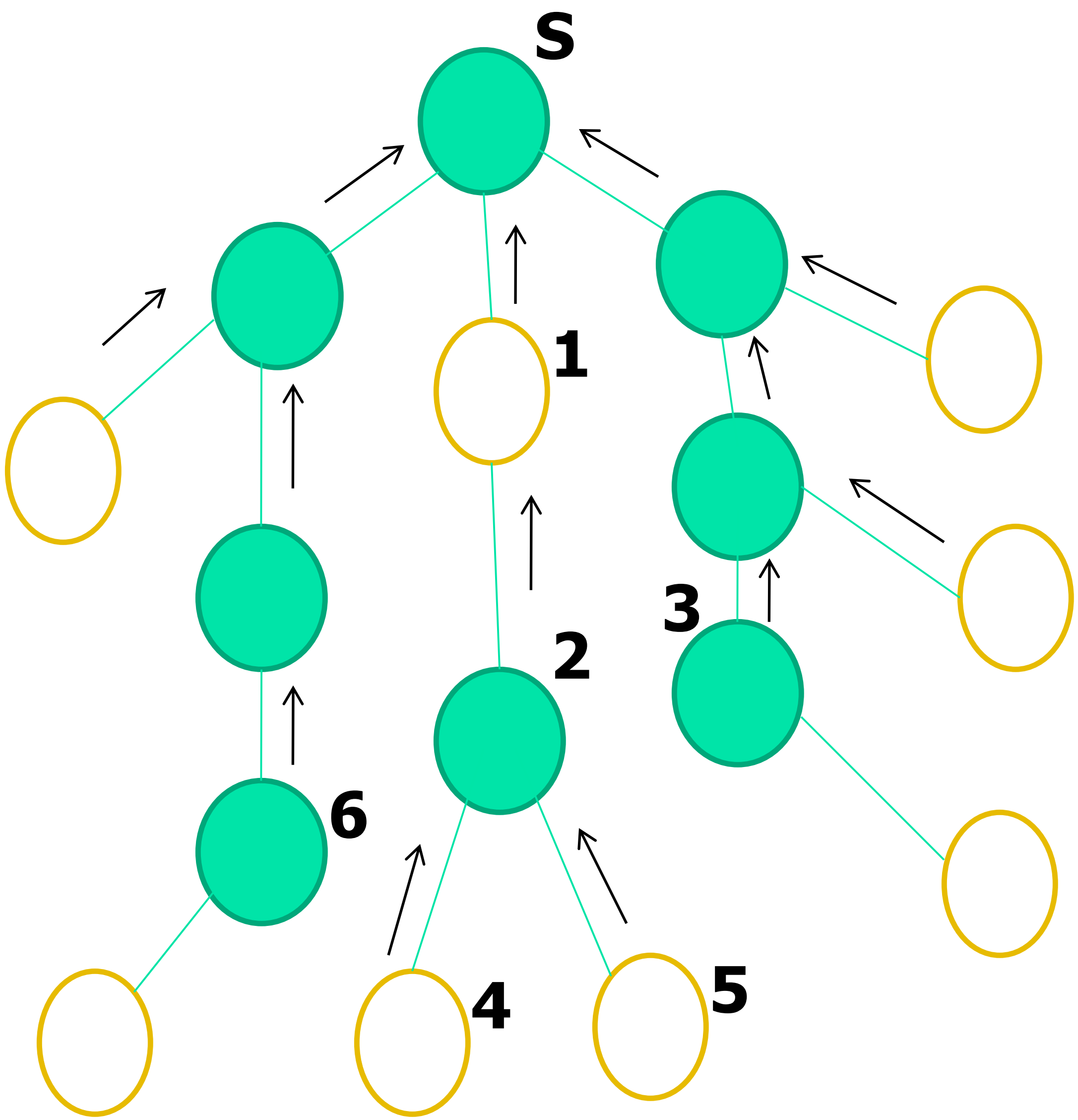
We gotta change it to make it a less cost path.

Stage 1 :

Parent Pointer change from

2 - > 3 (Cost = 4)
to
2 - > 1 (Cost = 2)

Illustration for CL parent pointer redirection recursively

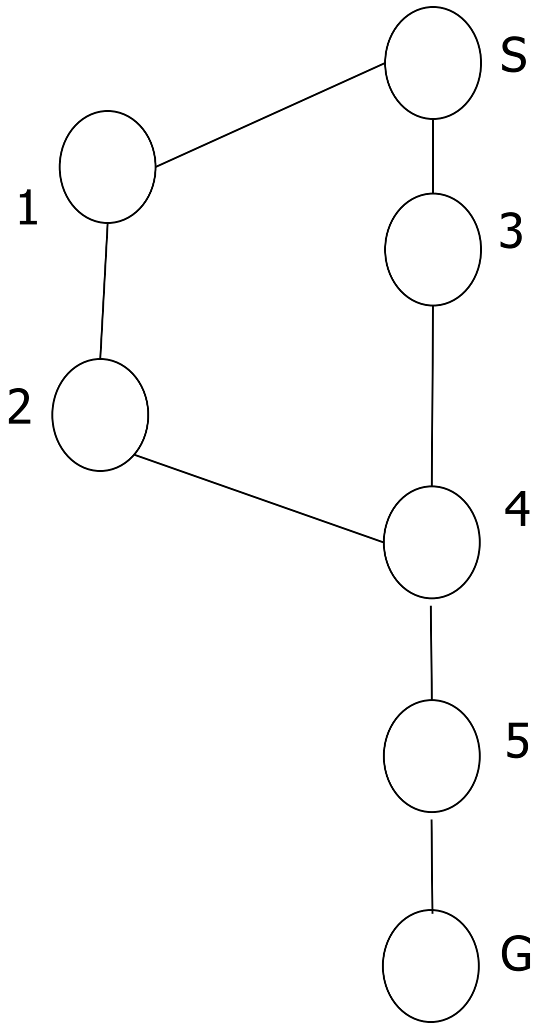


Stage 2 :

Parent Pointer change from

4 - > 6 (Cost = 4)
to
4 - > 2 (Cost = 3)

Another graph



Each arc cost
1 unit
Doubt
h=0 for all nodes,
Except 3, which is
3, i.e., h(3)=3

3 violates MR
h(3)=3
h(4)=0

Sequence of expansions:
S-1-2-4-…

Definition of monotonicity

- A heuristic $h(p)$ is said to satisfy the monotone restriction, if for all 'p', $h(p)<=h(p_c)+cost(p, p_c)$, where '$p_c$' is the child of '$p$'.

Theorem

- If monotone restriction (also called triangular inequality) is satisfied, then for nodes in the closed list, redirection of parent pointer is not necessary. In other words, if any node *'n'* is chosen for expansion from the open list, then $g(n)=g^*(n)$, where $g(n)$ is the cost of the path from the start node *'s'* to *'n'* at that point of the search when *'n'* is chosen, and $g^*(n)$ is the cost of the optimal path from *'s'* to *'n'*

Establishing a basis or foundation or learning something.

Grounding the Monotone Restriction

7	3	
1	2	4
8	5	6

n

1	2	3
4	5	6
7	8	

G

7	3	4
1	2	
8	5	6

n’

h(n) -: number of displaced tiles

Is h(n) monotone ?
h(n) = 8
h(n’) = 8
C(n,n’) = 1

Hence monotone

Monotonicity of # of Displaced Tile Heuristic

- $h(n) <= h(n') + c(n, n')$
- Any move changes h(n) by at most 1

if it increases then we're done and if decreases then there's only decreases by one which get compensated by cost it take.

- $c = 1$
- Hence, $h(parent) <= h(child) + 1$
- If the empty cell is also included in the cost, then *h* need not be monotone (try!)

Didn't get the point. How'd we will count empty cell into heuristic.

Monotonicity of Manhattan Distance Heuristic (1/2)

- *Manhattan distance= X-dist+Y-dist* from the target position
- Refer to the diagram in the first slide:
- $h_{mn}(n) = 1 + 1 + 1 + 2 + 1 + 1 + 2 + 1 = 10$
- $h_{mn}(n') = 1 + 1 + 1 + 3 + 1 + 1 + 2 + 1 = 11$
- $Cost = 1$
- Again, $h(n) < = h(n') + c(n, n')$

Monotonicity of Manhattan Distance Heuristic (2/2)

- Any move can either increase the h value or decrease it by **at most 1.**
- Cost again is 1.
- Hence, this heuristic also satisfies Monotone Restriction
- If empty cell is also included in the cost then manhattan distance does not satisfy monotone restriction (try!)
- Apply this heuristic for Missionaries and Cannibals problem

Relationship between Monotonicity and Admissibility

- Observation:

  Monotone Restriction $\rightarrow$ Admissibility but not vice-versa

- Statement: *If* $h(n_i) <= h(n_j) + c(n_i, n_j)$ *for all* i, j

  then $h(n_i) < = h^*(n_i)$ *for all* i

Proof of Monotonicity→admissibility

Let us consider the following as the optimal path starting with a node $n = n_1 - n_2 - n_3 \ldots n_i - \ldots n_m = g_l$

Observe that

$$h^*(n) = c(n_1, n_2) + c(n_2, n_3) + \ldots + c(n_{m-1}, g_l)$$

Since the path given above is the optimal path from n to g_l

Now,

$h(n_1) <= h(n_2) + c(n_1, n_2)$ ------ Eq 1

$h(n_2) <= h(n_3) + c(n_2, n_3)$ ------ Eq 2

: : : : : :

$h(n_{m-1}) = h(g_i) + c(n_{m-1}, g_i)$------ Eq (m-1)

Adding Eq 1 to Eq (m-1) we get

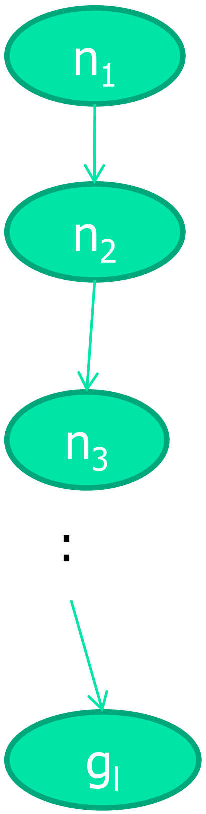
$$h(n) <= h(g_l) + h^*(n) = h^*(n)$$

if h <= h* then the heuristic is admissible.

Hence proved that MR → (h <= h*)

Proof (continued…)

Counter example for vice-versa



$h^*(n_1) = 3$ $\qquad$ $h(n_1) = 2.5$

$h^*(n_2) = 2$ $\qquad$ $h(n_2) = 1.2$

$h^*(n_3) = 1$ $\qquad$ $h(n_3) = 0.5$

: $\quad$: $\qquad$: $\quad$:

$h^*(g_l) = 0$ $\qquad$ $h(g_l) = 0$

MR is not satisfied.

$h < h^*$ everywhere but MR is not satisfied

Proof of MR leading to optimal path for every expanded node (1/2)

Let $S\text{-}N_1\text{-}N_2\text{-}N_3\text{-}N_4 \ldots N_m \ldots N_k$ be an optimal path from S to N_k (all of which might or might not have been explored). Let N_m be the **last** node on this path which is on the open list, i.e., *all* the ancestors from S up to N_{m-1} are in the closed list.

For every node N_p on the optimal path,
for any node in Optimal path g(n) = g*(n).

$g^*(N_p)+h(N_p) <= g^*(N_p)+C(N_p,N_{p+1})+h(N_{p+1})$, by monotone restriction
$g^*(N_p)+h(N_p) <= g^*(N_{p+1})+h(N_{p+1})$ on the optimal path
$g^*(N_m)+h(N_m) <= g^*(N_k)+h(N_k)$ by transitivity

Since all ancestors of N_m in the optimal path are in the closed list,

$g(N_m) = g^*(N_m)$.
$=> f(N_m) = g(N_m) + h(N_m) = g^*(N_m) + h(N_m) <= g^*(N_k) + h(N_k)$

Proof of MR leading to optimal path for every expanded node (2/2)

Now if N_k is chosen in preference to N_m,

$$f(N_k) <= f(N_m)$$

$$g(N_k)+ h(N_k) <= g(N_m)+ h(N_m)$$

$$= g^*(N_m)+ h(N_m)$$

$$<= g^*((N_k)+ h(N_k)$$

$$g(N_k)<=g^*(N_k)$$

But $g(N_k)>=g^*(N_k)$, by definition

Hence $g(N_k)=g^*(N_k)$

This means that if N_k is chosen for expansion, the optimal path to this from S has already been found.

Any node nk chosen then Optimal path from S to it has already been explored if MR.

The key point here is that if MR is satisfied, nodes in an optimal path have to get expanded in the order of their distance from the start node.

TRY proving by induction on the length of optimal path Hard one!

Monotonicity of $f(.)$ values

Statement:

One can easily check that.

f values of nodes expanded by A* increase monotonically, if h is monotone.

Proof:

Expanded in the monotonically increasing order.

Suppose n_i and n_j are expanded with temporal sequentiality, *i.e.*, n_j is expanded after n_i

Proof (1/3)...

- n_i expanded before n_j
  - n_i and n_j co-existing
    - n_j's parent pointer changes to n_i and expanded
    - n_j expanded after n_i
  - n_j comes to open list as a result of expanding n_i and is expanded immediately

Proof (2/3)...

Optimal path exploration is done in the increasing order of F values.

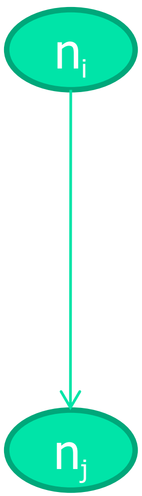
- All the previous cases are forms of the following two cases (think!)
- CASE 1:

  n_j was on open list when n_i was expanded

  Hence, $f(n_i) <= f(n_j)$ by property of A*
- CASE 2:

  n_j comes to open list due to expansion of n_i

Proof (3/3)…



Case 2:

$f(n_i) = g(n_i) + h(n_i)$ (Defn of f)

$f(n_j) = g(n_j) + h(n_j)$ (Defn of f)

$f(n_i) = g(n_i) + h(n_i) = g^*(n_i) + h(n_i)$ ---*Eq 1*

(since n_i is picked for expansion n_i is on optimal path)

With the similar argument for n_j we can write the following:

$f(n_j) = g(n_j) + h(n_j) = g^*(n_j) + h(n_j)$ ---*Eq 2*

Not sure about g*(nj).

Also,

$h(n_i) <= h(n_j) + c(n_i, n_j)$ ---*Eq 3* (Parent- child relation)

$g^*(n_j) = g^*(n_i) + c(n_i, n_j)$ ---Eq 4 (both nodes on optimal path)

From *Eq 1, 2, 3 and 4*

$f(n_i) <= f(n_j)$

Hence proved.

Better way to understand monotonicity of *f()*

- Let $f(n_1), f(n_2), f(n_3), f(n_4)\ldots f(n_{k-1}), f(n_k)$ be the f values of k expanded nodes.
- The relationship between two consecutive expansions $f(n_i)$ and $f(n_{i+1})$ nodes always remains the same, *i.e.,* $f(n_i) <= f(n_{i+1})$
- This is because
  - $f(n_i)= g(n_i) +h(n_i)$ and
  - $g(n_i)=g^*(n_i)$ since n_i is an expanded node (proved theorem) and this value cannot change
  - $h(n_i)$ value also cannot change Hence nothing after n_{i+1}'s expansion can change the above relationship.

Monotonicity of f()

$f(n_1), f(n_2), f(n_3), \ldots\ldots, f(n_i), f(n_{i+1}), \ldots, f(n_k)$
Sequence of expansion of $n_1, n_2, n_3 \ldots n_i \ldots n_k$

f values increase monotonically
$f(n) = g(n) + h(n)$

Consider two successive expansions - > n_i, n_{i+1}

Case 1:
n_i & n_{i+1} Co-existing in OL
n_i precedes n_{i+1}

By definition of A* First expanded node has lower f() value.
$f(n_i) <= f(n_{i+1})$

Monotonicity of f()

Case 2:

ni+1 came to OL because of expanding ni and ni+1 is expanded

$$
\begin{aligned}
f(ni) &= g(ni) + h(ni) \\
&<= g(ni) + c(ni, hi+1) + h(ni+1) \\
&= g(ni) + h(ni+1) \\
&= f(ni+1)
\end{aligned}
$$

Case 3:

ni+1 becomes child of ni after expanding ni and ni+1 is expanded. Same as case 2.

A list of AI Search Algorithms

- A*
  - AO*
  - IDA* (Iterative Deepening)
- Minimax Search on Game Trees
- Viterbi Search on Probabilistic FSA
- Hill Climbing
- Simulated Annealing
- Gradient Descent
- Stack Based Search
- Genetic Algorithms
- Memetic Algorithms

Foundational Points

Symbolic AI

Connectionist AI is contrasted with Symbolic AI

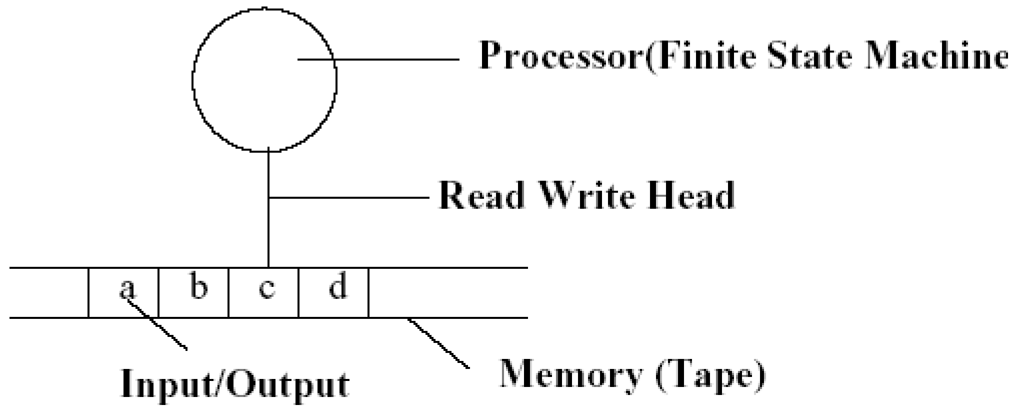
Symbolic AI - Physical Symbol System Hypothesis

> **Every intelligent system can be constructed by storing and processing symbols and nothing more is necessary.**

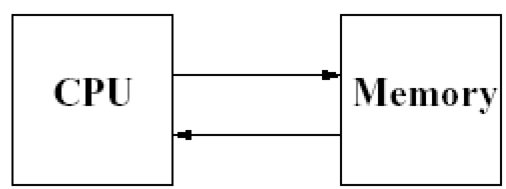
Symbolic AI has a bearing on models of computation such as

- **Turing Machine**
- **Von Neumann Machine**
- **Lambda calculus**

Turing Machine & Von Neumann Machine



Turing machine



VonNeumann Machine

Challenges to Symbolic AI

Motivation for challenging Symbolic AI

A large number of computations and information process tasks that living beings are comfortable with, are not performed well by computers!

The Differences

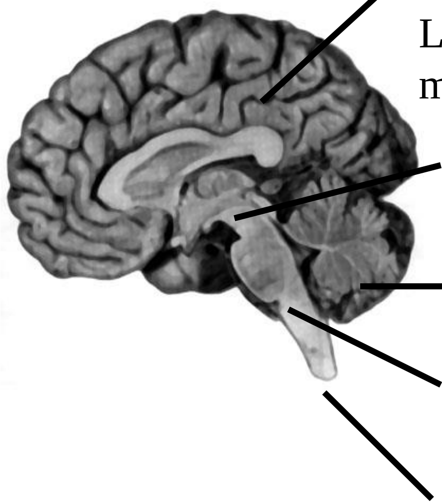
Brain computation in living beings	TM computation in computers
Pattern Recognition	Numerical Processing
Learning oriented	Programming oriented
Distributed & parallel processing	Centralized & serial processing
Content addressable	Location addressable

The human brain

Seat of consciousness and cognition

Perhaps the most complex information processing machine in nature

Beginner's Brain Map



Forebrain (Cerebral Cortex): Language, maths, sensation, movement, cognition, emotion

Midbrain: Information Routing; involuntary controls

Cerebellum: Motor Control

Hindbrain: Control of breathing, heartbeat, blood circulation

Spinal cord: Reflexes, information highways between body & brain

Brain : a computational machine?

Information processing: brains vs computers

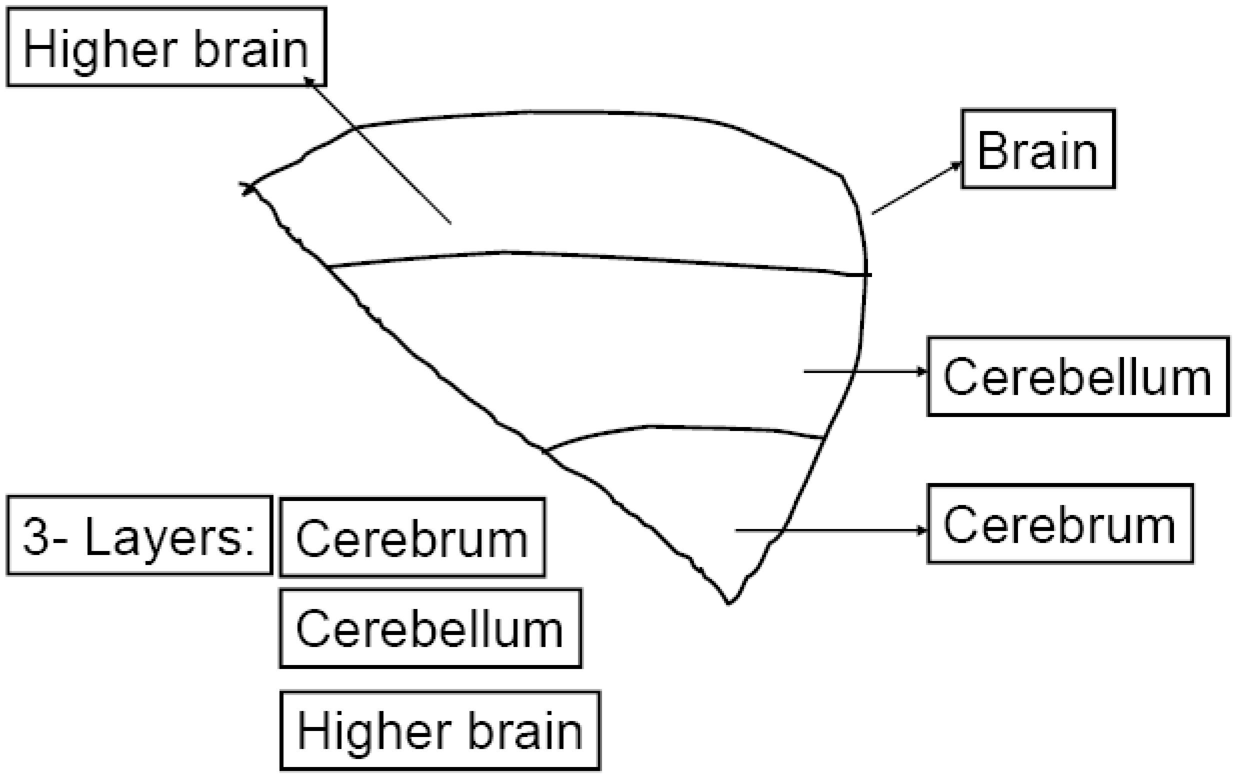
- brains better at perception / cognition
- slower at numerical calculations
- parallel and distributed Processing
- associative memory

Brain : a computational machine? *(contd.)*

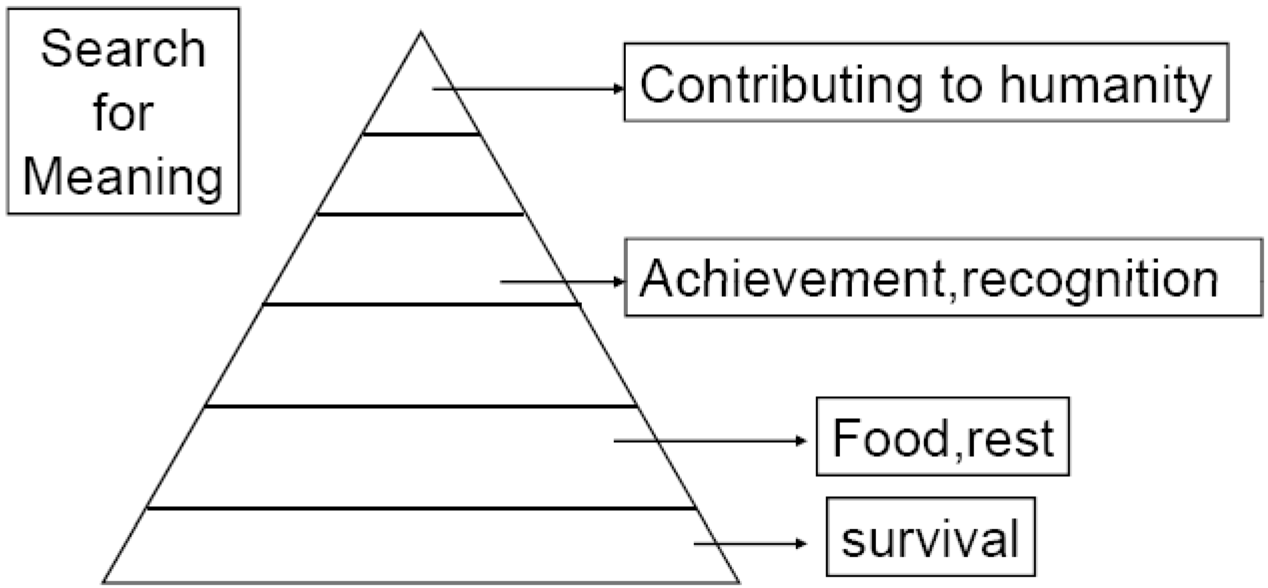
- Evolutionarily, brain has developed algorithms most suitable for survival
- Algorithms unknown: the search is on
- Brain astonishing in the amount of information it processes
  - Typical computers: 10^9 operations/sec
  - Housefly brain: 10^{11} operations/sec

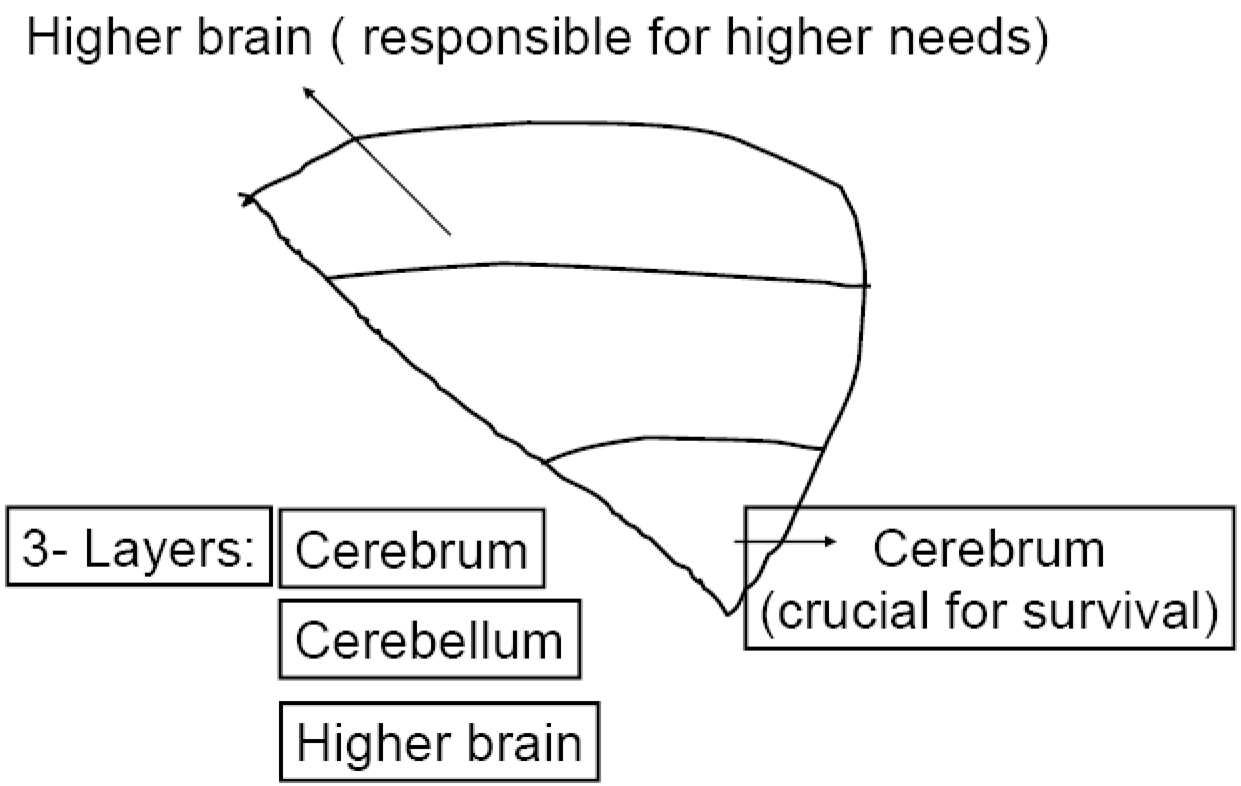
Brain facts & figures

- Basic building block of nervous system: nerve cell (neuron)
- ~ 10^{12} neurons in brain
- ~ 10^{15} connections between them
- Connections made at “synapses”
- The speed: events on millisecond scale in neurons, nanosecond scale in silicon chips

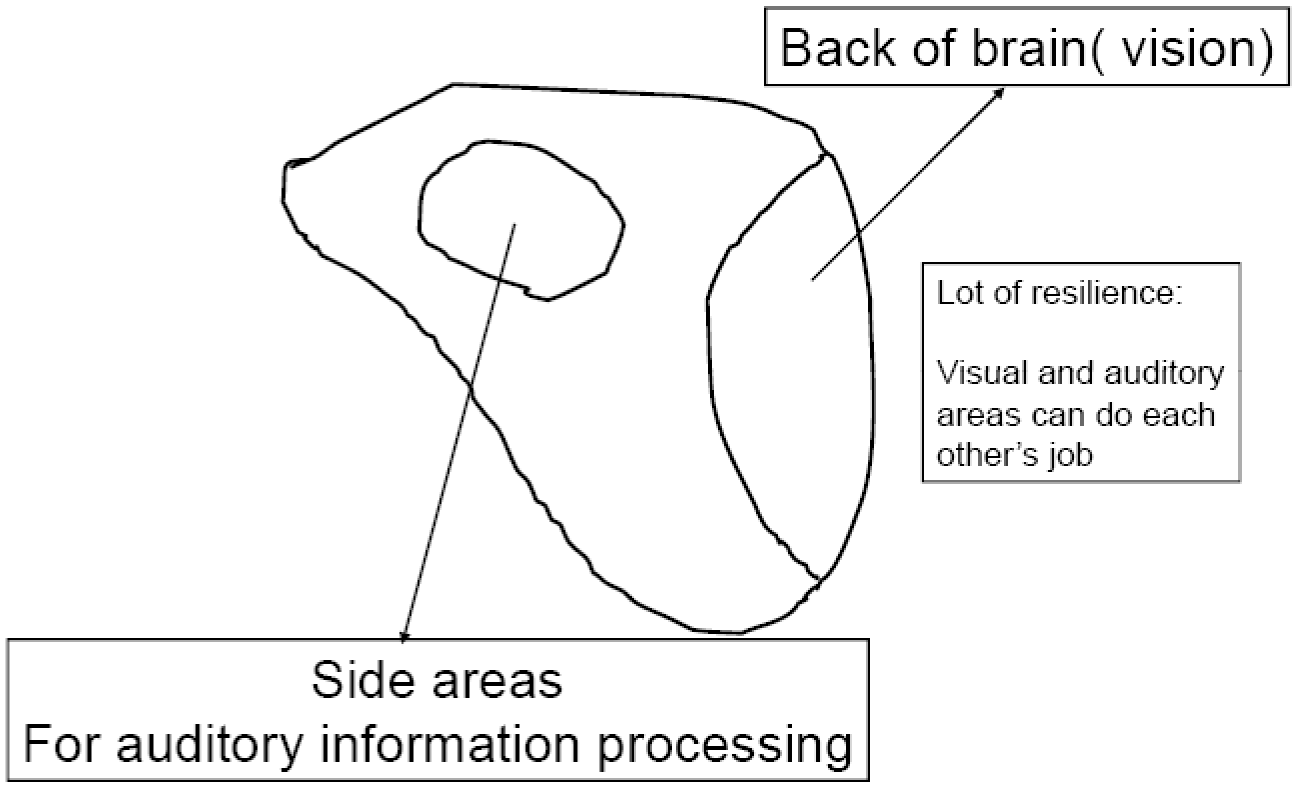

Higher brain
Brain
Cerebellum
Cerebrum
3- Layers:
Cerebrum
Cerebellum
Higher brain

Maslow's hierarchy




Higher brain (responsible for higher needs)
3- Layers:
Cerebrum
Cerebellum
Higher brain
Cerebrum
(crucial for survival)

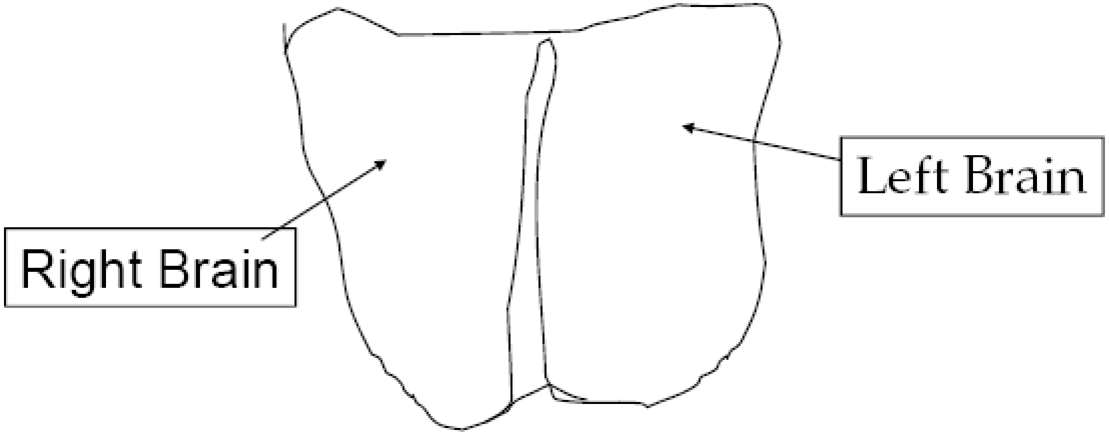
Mapping of Brain



Left Brain and Right Brain

Dichotomy

A difference between two opposite things a division into two opposite groups.



Left Brain – Logic, Reasoning, Verbal ability

Right Brain – Emotion, Creativity

Words – left Brain

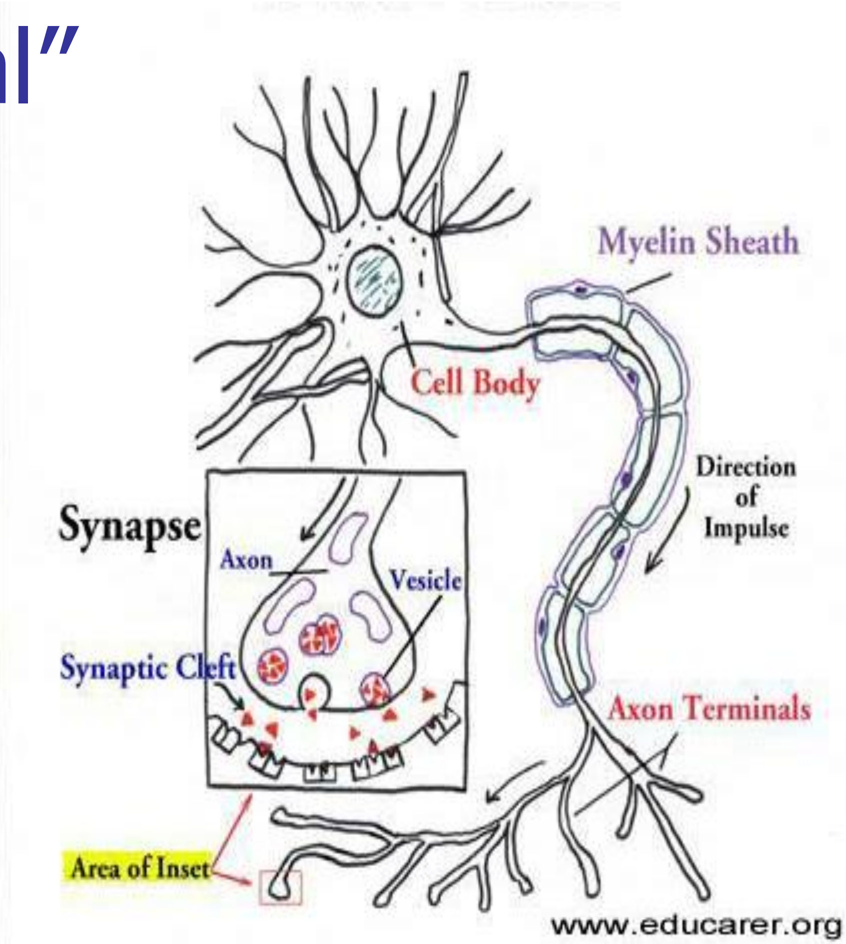
Music

Tune – Right Brain

Maps in the brain. Limbs are mapped to brain

Neuron - “classical”

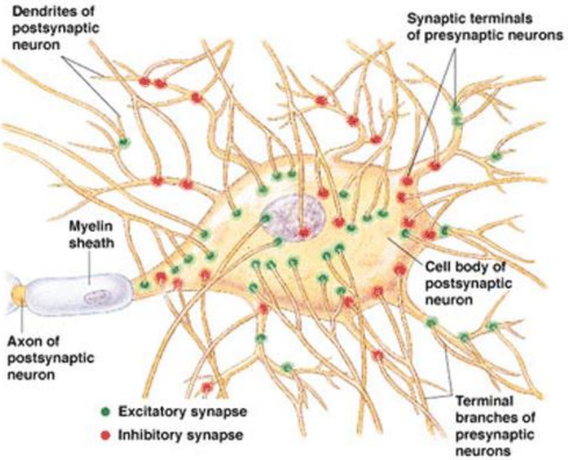
- Dendrites
  - Receiving stations of neurons
  - Don't generate action potentials
- Cell body
  - Site at which information received is integrated
- Axon
  - Generate and relay action potential
  - Terminal
    - Relays information to next neuron in the pathway



http://www.educarer.com/images/brain-nerve-axon.jpg

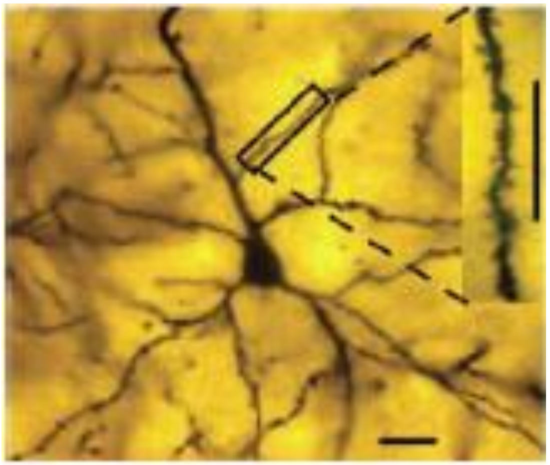
Computation in Biological Neuron

- Incoming signals from synapses are summed up at the soma
- Σ, the biological "inner product"
- On crossing a threshold, the cell "fires" generating an action potential in the axon hillock region

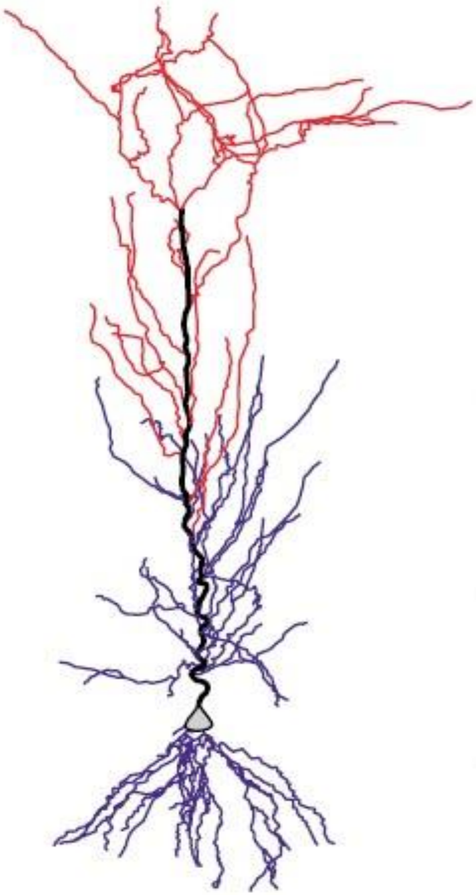


Synaptic inputs: Artist's conception

The biological neuron



Pyramidal neuron, from the amygdala (Rupshi *et al.* 2005)

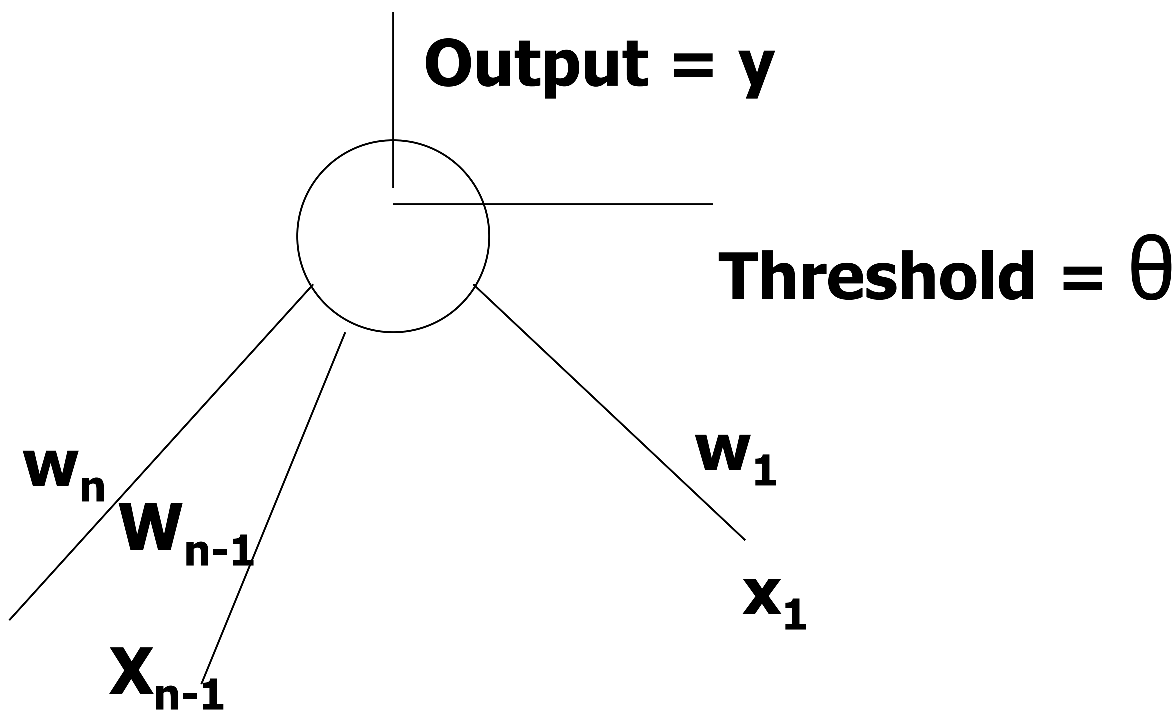


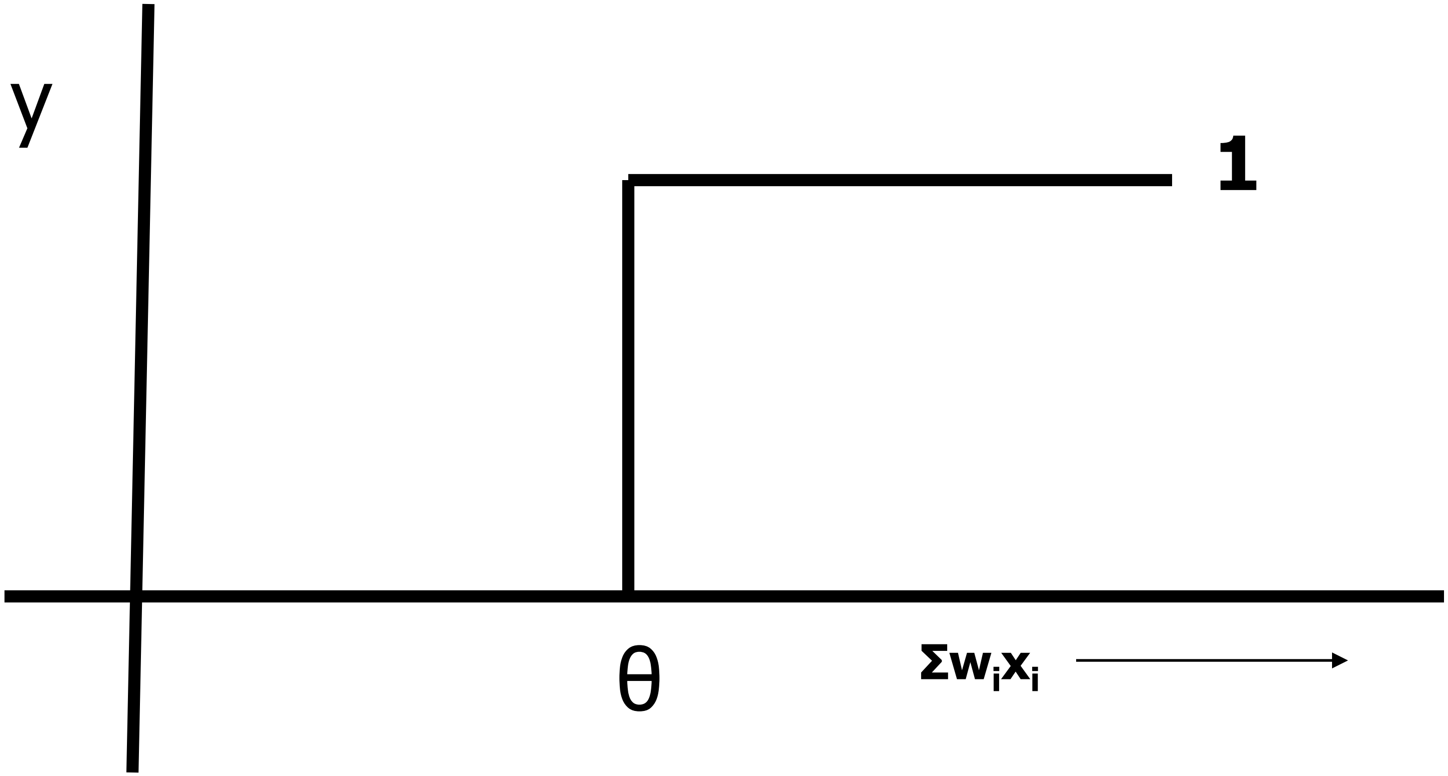
A CA1 pyramidal neuron (Mel *et al*. 2004)

Perceptron

The Perceptron Model

A perceptron is a computing element with input lines having associated weights and the cell having a threshold value. The perceptron model is motivated by the biological neuron.





Step function / Threshold function

y = 1 for $\Sigma w_i x_i >= \theta$

=0 otherwise

Features of Perceptron

- Input output behavior is discontinuous and the derivative does not exist at $\Sigma w_i x_i = \theta$

- $\Sigma w_i x_i - \theta$ is the net input denoted as net

- Referred to as a linear threshold element - linearity because of **x** appearing with power **1**

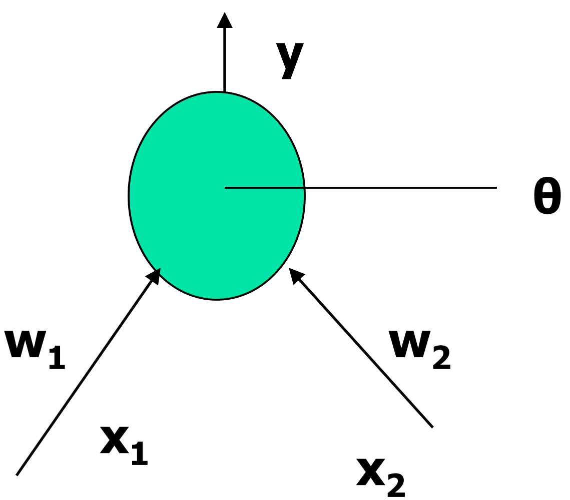
- **y= f(net)**: Relation between y and net is non-linear because function is step function.

Computation of Boolean functions

AND of 2 inputs

X1	**x2**	**y**
0	0	0
0	1	0
1	0	0
1	1	1

The parameter values (weights & thresholds) need to be found.



Computing parameter values

$w1 * 0 + w2 * 0 \;\; <= \theta \Rightarrow \theta >= \;\; 0$; since y=0

$w1 * 0 + w2 \;\; * 1 \;\; <= \theta \Rightarrow w2 \;\; <= \theta$; since y=0

$w1 * 1 + w2 * 0 \;\; <= \theta \Rightarrow \;\; w1 \;\; <= \theta$; since y=0

$w1 * 1 + w2 \;\; *1 > \theta \Rightarrow w1 + w2 > \theta$; since y=1

$w1 = w2 = \;\; = 0.5$

satisfy these inequalities and find parameters to be used for computing AND function.

Other Boolean functions

- **OR can be computed using values of w1 = w2 = 1 and = 0.5**

- **XOR function gives rise to the following inequalities:**

$w1 * 0 + w2 * 0 <= \theta \Rightarrow \theta >= 0$

$w1 * 0 + w2 * 1 > \theta \Rightarrow w2 > \theta$

$w1 * 1 + w2 * 0 > \theta \Rightarrow w1 > \theta$

$w1 * 1 + w2 * 1 <= \theta \Rightarrow w1 + w2 <= \theta$

No set of parameter values satisfy these inequalities.

Threshold functions

n	# Boolean functions (2^2^n)	#Threshold Functions (2^{n^2})
1	4	4
2	16	14
3	256	128
4	64K	1008

- Functions computable by perceptrons - threshold functions
- #TF becomes negligibly small for larger values of #BF.
- For n=2, all functions except XOR and XNOR are computable.

Perceptrons and their computing power

Threshold functions

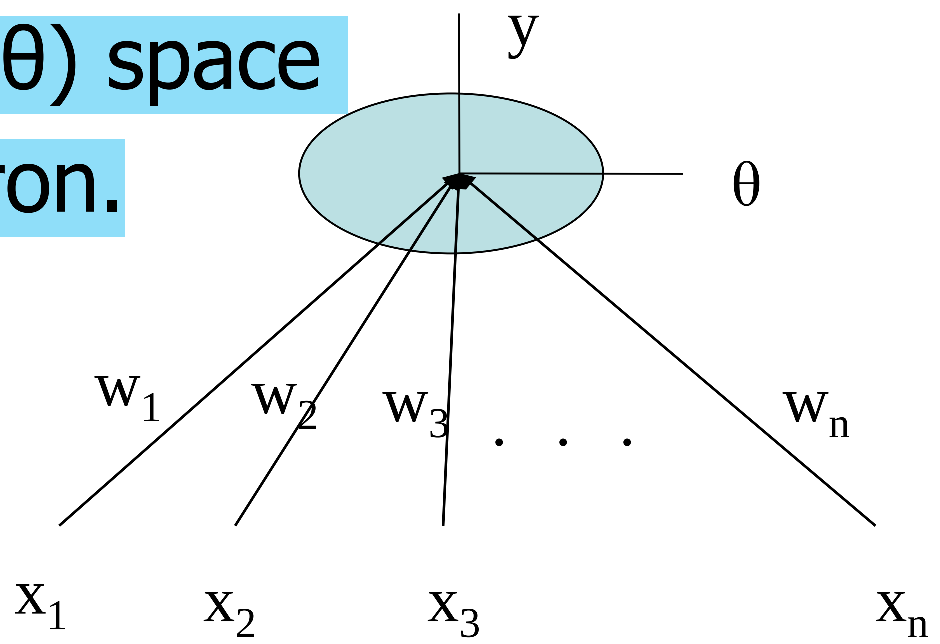
n	# Boolean functions (2^2^n)	#Threshold Functions (2^{n^2})
1	4	4
2	16	14
3	256	128
4	64K	1008

- Functions computable by perceptrons - threshold functions
- #TF becomes negligibly small for larger values of #BF.
- For n=2, all functions except XOR and XNOR are computable.

Link

Concept of Hyper-planes

- $\sum w_i x_i = \theta$ defines a linear surface in the (W,θ) space, where $W=<w_1,w_2,w_3,\ldots,w_n>$ is an n-dimensional vector.
- A point in this (W,θ) space defines a perceptron.



Perceptron Property

- Two perceptrons may have different parameters but same function

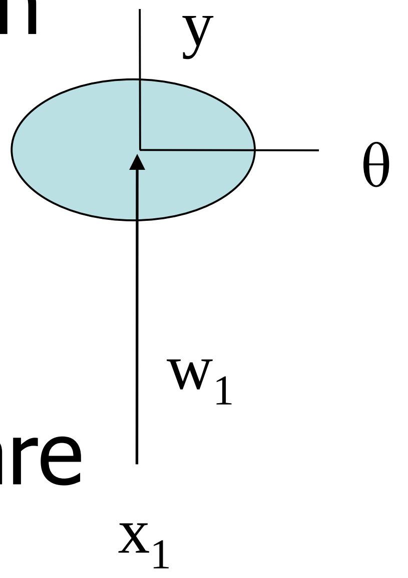
- Example of the simplest perceptron

  w.x>ɵ gives y=1

  w.x≤ɵ gives y=0

  Depending on different values of

  w and θ, four different functions are possible



Simple perceptron contd.

x	F_1	F_2	F_3	F_4
0	0	0	1	1
1	0	1	0	1

<u>True-Function</u>

$\theta<0$

$w>\theta$

<u>0-function</u>

$0<=\theta$

$=>\theta\geq 0$

$w\leq\theta$

<u>Identity Function</u>

$\theta\geq 0$

$w> \theta$

<u>Complement Function</u>

$0> \theta$; $=>$ $\theta<0$

$w\leq \theta$

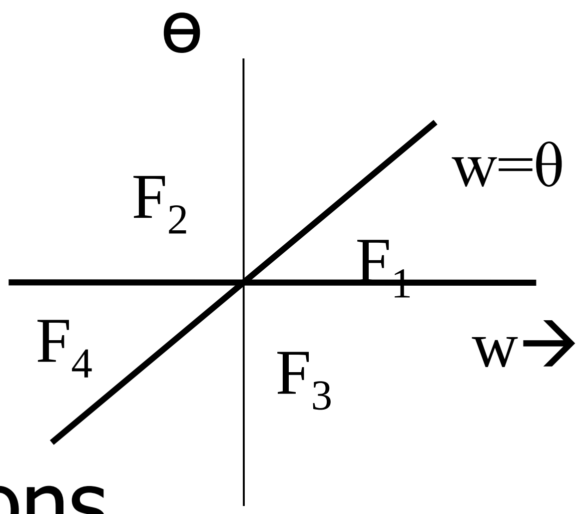
Counting the number of functions for the simplest perceptron

- For the simplest perceptron, the equation is $w.x=\theta$.

Substituting $x=0$ and $x=1$,

we get $\theta=0$ and $w=\theta$.

These two lines intersect to form four regions, which correspond to the four functions.



Fundamental Observation

- The number of TFs computable by a perceptron is equal to the number of regions produced by 2^n hyper-planes,obtained by plugging in the values $<x_1,x_2,x_3,\ldots,x_n>$ in the equation

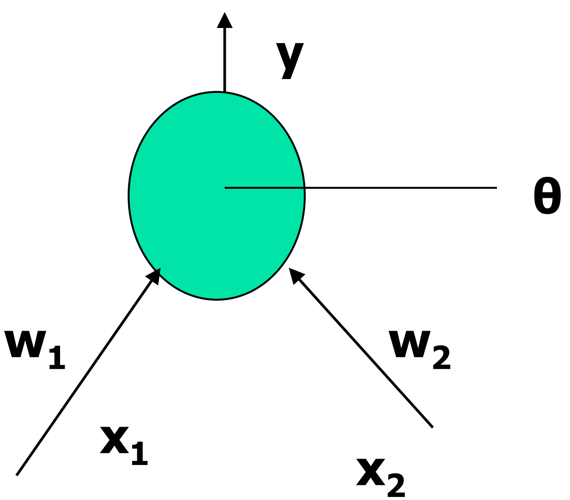
$$\sum_{i=1}^{n} w_i x_i = \theta$$

Try to put the values of Xi's and see the number of regions produced which equals the number of TFs computable by a perceptron.

AND of 2 inputs

X1	**x2**	**y**
0	0	0
0	1	0
1	0	0
1	1	1

The parameter values (weights & thresholds) need to be found.



Constraints on w1, w2 and θ

$w1 * 0 + w2 * 0 <= \theta \Rightarrow \theta >= 0$; since $y=0$

$w1 * 0 + w2 * 1 <= \theta \Rightarrow w2 <= \theta$; since $y=0$

$w1 * 1 + w2 * 0 <= \theta \Rightarrow w1 <= \theta$; since $y=0$

$w1 * 1 + w2 * 1 > \theta \Rightarrow w1 + w2 > \theta$; since $y=1$

$w1 = w2 = \ = 0.5$

These inequalities are satisfied by ONE particular region

The geometrical observation

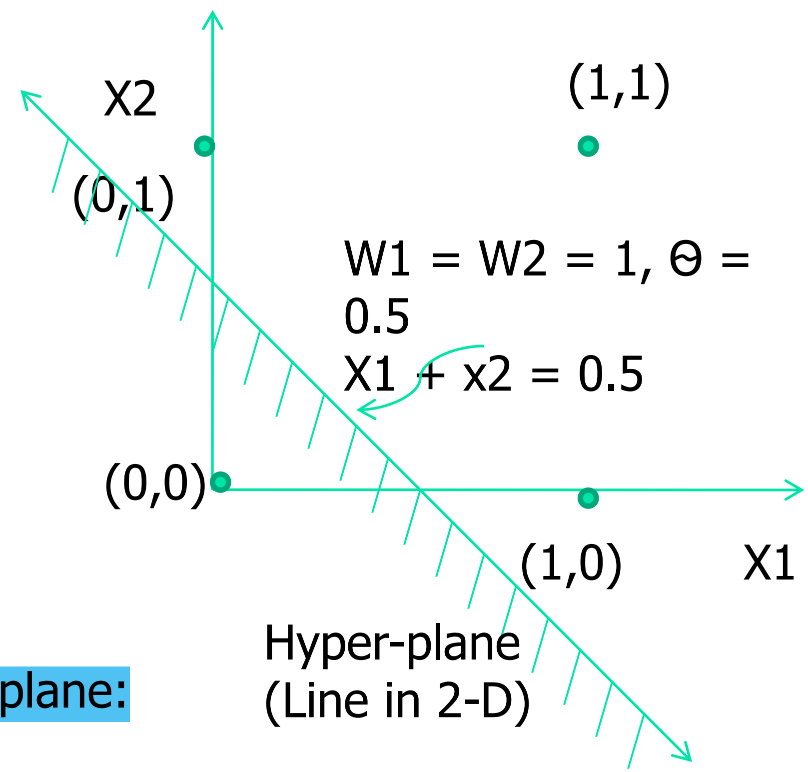
- **Problem:** *m* linear surfaces called hyper-planes (each hyper-plane is of *(d-1)*-dim) in d-dim, then what is the max. no. of regions produced by their intersection?

  i.e., $R_{m,d} = ?$

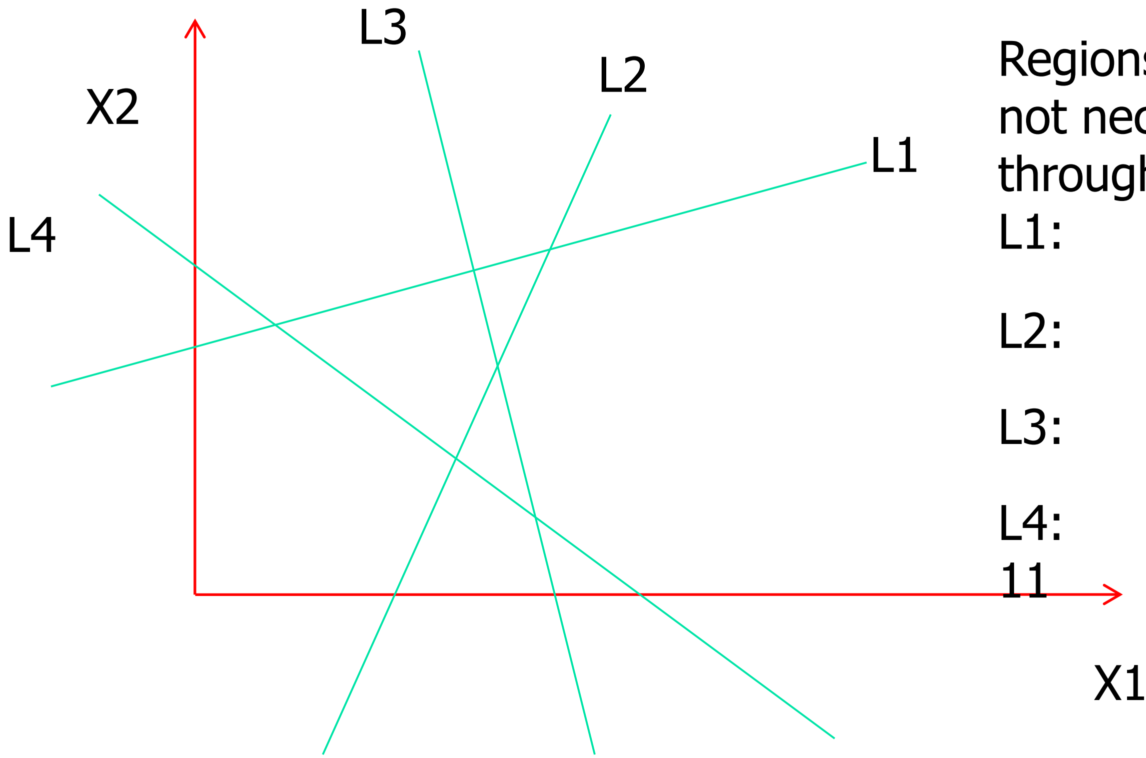
Co-ordinate Spaces

We work in the $<X_1, X_2>$ space or the $<w_1, w_2, \Theta>$ space

General equation of a Hyperplane:
$\Sigma W_i X_i = \Theta$



Regions produced by lines



Regions produced by lines not necessarily passing through origin

L1: 2

L2: 2+2 = 4

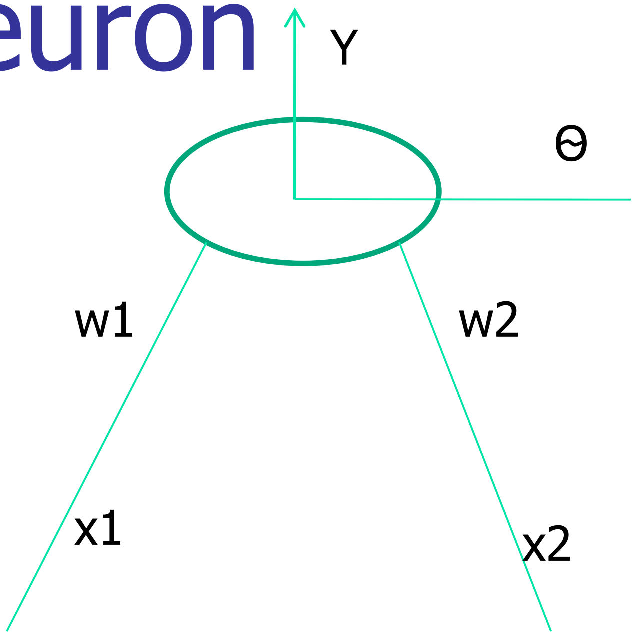
L3: 2+2+3 = 7

L4: 2+2+3+4 = 11

New regions created = Number of intersections on the incoming line by the original lines

Total number of regions = Original number of regions + New regions created

Number of computable functions by a neuron



$$w1 * x1 + w2 * x2 = \theta$$

$$(0,0) \Rightarrow \theta = 0 : P1$$

$$(0,1) \Rightarrow w2 = \theta : P2$$

$$(1,0) \Rightarrow w1 = \theta : P3$$

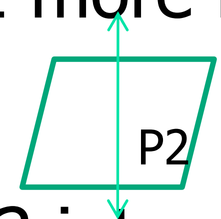
$$(1,1) \Rightarrow w1 + w2 = \theta : P4$$

P1, P2, P3 and P4 are planes in the <W1,W2, Θ> space

Number of computable functions by a neuron (cont…)

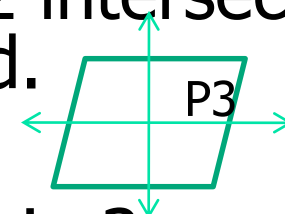
- P1 produces 2 regions
- P2 is intersected by P1 in a line. 2 more new regions are produced.

  Number of regions = 2+2 = 4



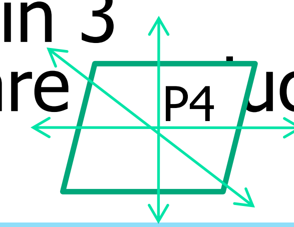
- P3 is intersected by P1 and P2 in 2 intersecting lines. 4 more regions are produced.

  Number of regions = 4 + 4 = 8



- P4 is intersected by P1, P2 and P3 in 3 intersecting lines. 6 more regions are produced.

  Number of regions = 8 + 6 = 14



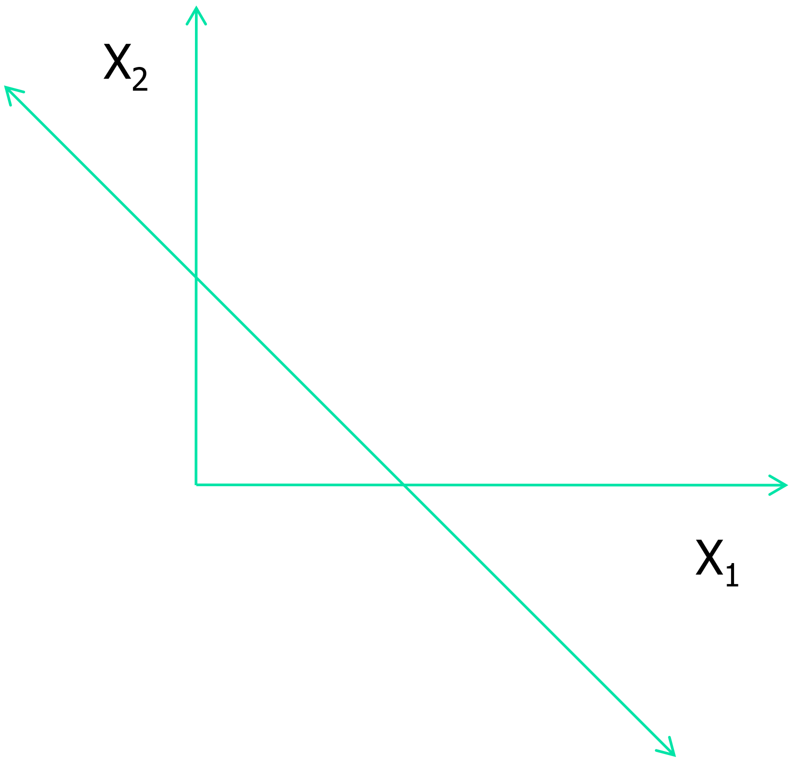
- Thus, a single neuron can compute 14 Boolean functions which are linearly separable.

Neurons compute the functions which are linearly separable.

Points in the same region

If
$W_1*X_1 + W_2*X_2 > \Theta$
$W_1'*X_1 + W_2'*X_2 > \Theta'$
Then

If $<W_1, W_2, \Theta>$ and $<W_1', W_2', \Theta'>$ share a region then they compute the same function



No. of Regions produced by Hyperplanes

Number of regions founded by n hyperplanes in d-dim passing through origin is given by the following recurrence relation

$$R_{n,d} = R_{n-1,d} + R_{n-1,d-1}$$

we use generating function as an operating function

Boundary condition:

$$R_{1,d} = 2$$

1 hyperplane in d-dim

$$R_{n,1} = 2$$

n hyperplanes in 1-dim,

Reduce to n points thru origin

which is line through origin.

The generating function is

$$f(x,y) = \sum_{n=1}^{\infty} \sum_{d=1}^{\infty} R_{n,d} \cdot x^n y^d$$

From the recurrence relation we have,

$$R_{n,d} - R_{n-1,d} - R_{n-1,d-1} = 0$$

$R_{n-1,d}$ corresponds to 'shifting' n by 1 place, => multiplication by x

$R_{n-1,d-1}$ corresponds to 'shifting' n and d by 1 place => multiplication by xy

On expanding $f(x,y)$ we get

$$\begin{aligned} f(x,y) = {} & R_{1,1} \cdot xy + R_{1,2} \cdot x y^2 + R_{1,3} \cdot x y^3 + \ldots + R_{1,d} \cdot x y^d + \ldots\ldots\infty \\ & + R_{2,1} \cdot x^2 y + R_{2,2} \cdot x^2 y^2 + R_{2,3} \cdot x^2 y^3 + \ldots + R_{2,d} \cdot x^2 y^d + \ldots\ldots\infty \\ & \ldots\ldots \\ & + R_{n,1} \cdot x^n y + R_{n,2} \cdot x^n y^2 + R_{n,3} \cdot x^n y^3 + \ldots + R_{n,d} \cdot x^n y^d + \ldots\ldots\infty \end{aligned}$$

$$f(x,y)=\sum_{n=1}^{\infty}\sum_{d=1}^{\infty}R_{n,d}\cdot x^{n}y^{d}$$

$$x\cdot f(x,y)=\sum_{n=1}^{\infty}\sum_{d=1}^{\infty}R_{n,d}\cdot x^{n+1}y^{d}=\sum_{n=2}^{\infty}\sum_{d=1}^{\infty}R_{n-1,d}\cdot x^{n}y^{d}$$

$$xy\cdot f(x,y)=\sum_{n=1}^{\infty}\sum_{d=1}^{\infty}R_{n,d}\cdot x^{n+1}y^{d+1}=\sum_{n=2}^{\infty}\sum_{d=2}^{\infty}R_{n-1,d-1}\cdot x^{n}y^{d}$$

$$x\cdot f(x,y)=\sum_{n=2}^{\infty}\sum_{d=2}^{\infty}R_{n-1,d}\cdot x^{n}y^{d}+\sum_{n=2}^{\infty}R_{n-1,1}\cdot x^{n}y$$

$$=\sum_{n=2}^{\infty}\sum_{d=2}^{\infty}R_{n-1,d}\cdot x^{n}y^{d}+2\cdot\sum_{n=2}^{\infty}x^{n}y$$

$$f(x,y)=\sum_{n=1}^{\infty}\sum_{d=1}^{\infty} R_{n,d}\cdot x^n y^d$$

$$=\sum_{n=2}^{\infty}\sum_{d=2}^{\infty} R_{n,d}\cdot x^n y^d+\sum_{d=1}^{\infty} R_{1,d}\cdot xy^d+\sum_{n=1}^{\infty} R_{n,1}\cdot x^n y-R_{1,1}\cdot xy$$

$$=\sum_{n=2}^{\infty}\sum_{d=2}^{\infty} R_{n,d}\cdot x^n y^d+2x\cdot\sum_{d=1}^{\infty} xy^d+2y\cdot\sum_{n=1}^{\infty} x^n y-2xy$$

extra x and y in above is mistake.

After all this expansion,

$$f(x,y)-x\cdot f(x,y)-xy\cdot f(x,y)$$

$$=\sum_{n=2}^{\infty}\sum_{d=2}^{\infty}(R_{n,d}-R_{n-1,d}-R_{n-1,d-1})x^n y^d$$

X^n X^n

$$+2y\cdot\sum_{n=1}^{\infty} x^d-2xy-2y\cdot\sum_{n=2}^{\infty} x+2x\cdot\sum_{d=1}^{\infty} y^d$$

$$=2x\cdot\sum_{d=1}^{\infty} y^d$$

since other two terms become zero

This implies

$$[1 - x - xy]\, f(x, y) = 2x \cdot \sum_{d=1}^{\infty} y^d$$

$$f(x, y) = \frac{1}{[1 - x(1 + y)]} \cdot 2x \cdot \sum_{d=1}^{\infty} y^d$$

$$= 2x \cdot [y + y^2 + y^3 + \ldots + y^d + \ldots..\infty] \cdot$$

$$[1 + x(1 + y) + x^2 (1 + y)^2 + \ldots + x^d (1 + y)^d + \ldots..\infty]$$

also we have,

$$f(x, y) = \sum_{n=1}^{\infty} \sum_{d=1}^{\infty} R_{n,\,d} \cdot x^n y^d$$

Comparing coefficients of each term in RHS we get, Comparing co-efficients we get

Regions produced.

$$R_{n,d} = 2\sum_{i=0}^{d-1} C_i^{n-1}$$

which is of order 2^n for n inputs.

Perceptron training

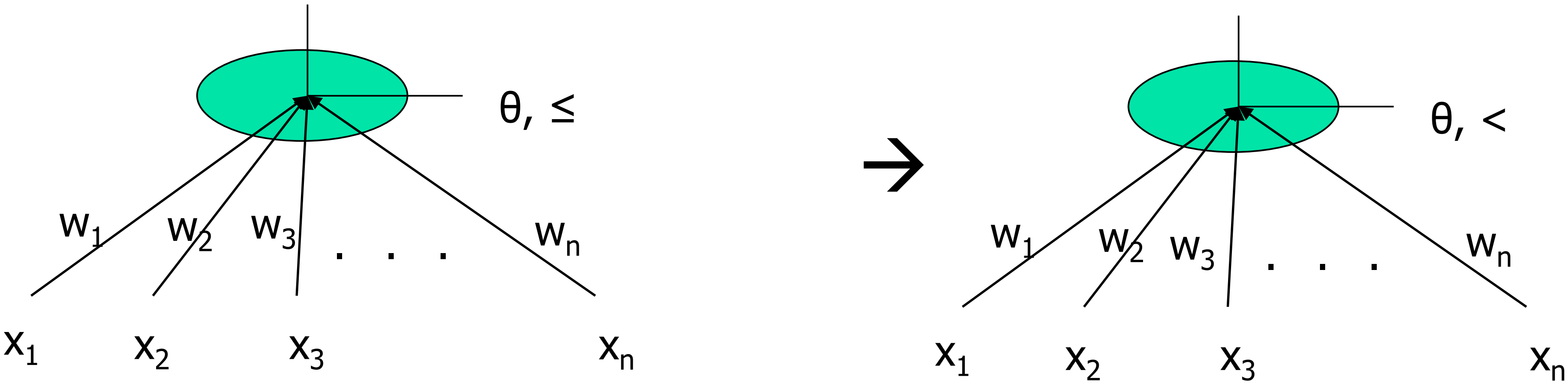
Perceptron Training Algorithm (PTA)

Preprocessing:

1. The computation law is modified to

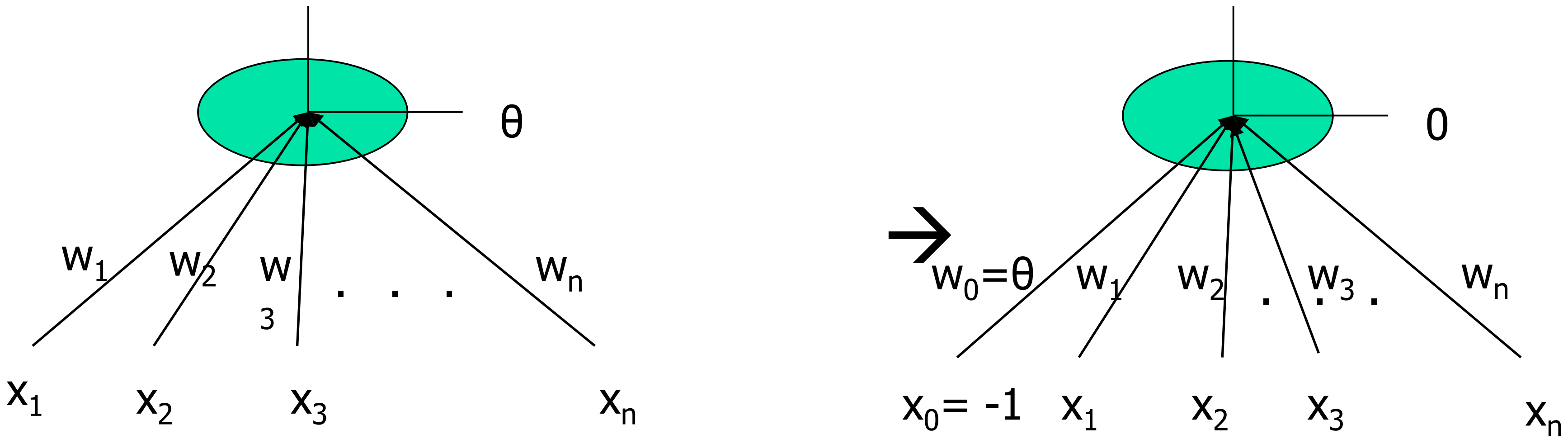
$$y = 1 \text{ if } \sum w_i x_i > \theta$$

$$y = o \text{ if } \sum w_i x_i < \theta$$



PTA – preprocessing cont…

2. Absorb θ as a weight



3. Negate all the zero-class examples

So that we've only one case to check for: greater than zero.

Example to demonstrate preprocessing

- **OR perceptron**

1-class <1,1> , <1,0> , <0,1>

0-class <0,0>

Augmented x vectors:- theta absorbed.

1-class <-1,1,1> , <-1,1,0> , <-1,0,1>

0-class <-1,0,0>

Negate 0-class:- <1,0,0>

Example to demonstrate preprocessing cont..

Now the vectors are

	x_0	x_1	x_2
X_1	-1	0	1
X_2	-1	1	0
X_3	-1	1	1
X_4	1	0	0

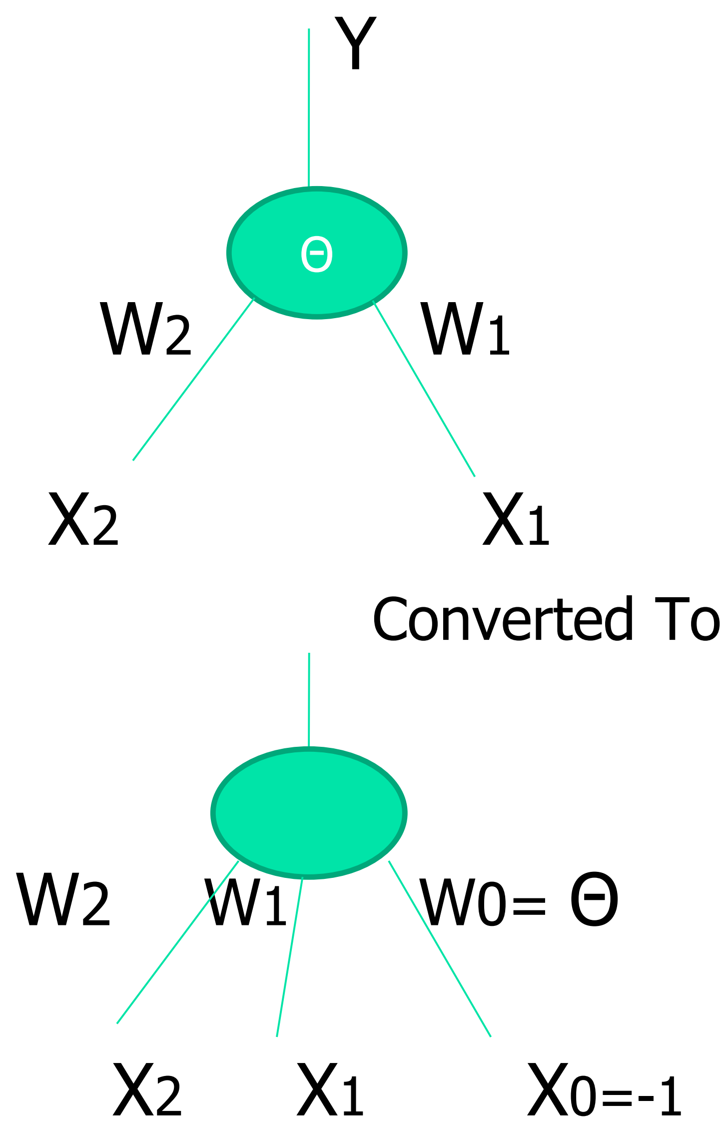
Perceptron Training Algorithm

1. Start with a random value of w
   ex: <0,0,0…>
2. Test for $wx_i > 0$
   If the test succeeds for i=1,2,…n
   then return w
3. Modify w, $w_{next} = w_{prev} + x_{fail}$

PTA on NAND

NAND:

X_2	X_1	Y
0	0	1
0	1	1
1	0	1
1	1	0



Preprocessing

NAND Augmented:

X_2	X_1	X_0	Y
0	0	-1	1
0	1	-1	1
1	0	-1	1
1	1	-1	0

NAND-0 class Negated

	X_2	X_1	X_0
V_0:	0	0	-1
V_1:	0	1	-1
V_2:	1	0	-1
V_3:	-1	-1	1

Vectors for which
W=<W2 W1 W0> has to
be found such that
W. Vi > 0

Use inequalities to show inconsistency.

PTA Algo steps

Algorithm:

1. Initialize and Keep adding the failed vectors until W. Vi > 0 is true.

Step 0: $W = \langle 0, 0, 0 \rangle$

$W_1 = \langle 0, 0, 0 \rangle + \langle 0, 0, -1 \rangle$ {V_0 Fails}

$= \langle 0, 0, -1 \rangle$

$W_2 = \langle 0, 0, -1 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

$= \langle -1, -1, 0 \rangle$

$W_3 = \langle -1, -1, 0 \rangle + \langle 0, 0, -1 \rangle$ {V_0 Fails}

$= \langle -1, -1, -1 \rangle$

$W_4 = \langle -1, -1, -1 \rangle + \langle 0, 1, -1 \rangle$ {V_1 Fails}

$= \langle -1, 0, -2 \rangle$

Trying convergence

$W_5 = \langle -1, 0, -2 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

$= \langle -2, -1, -1 \rangle$

$W_6 = \langle -2, -1, -1 \rangle + \langle 0, 1, -1 \rangle$ {V_1 Fails}

$= \langle -2, 0, -2 \rangle$

$W_7 = \langle -2, 0, -2 \rangle + \langle 1, 0, -1 \rangle$ {V_0 Fails}

$= \langle -1, 0, -3 \rangle$

$W_8 = \langle -1, 0, -3 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

$= \langle -2, -1, -2 \rangle$

$W_9 = \langle -2, -1, -2 \rangle + \langle 1, 0, -1 \rangle$ {V_2 Fails}

$= \langle -1, -1, -3 \rangle$

Trying convergence

$W_{10} = \langle -1, -1, -3 \rangle + \langle -1, -1, 1 \rangle$ $\{V_3$ Fails$\}$

$= \langle -2, -2, -2 \rangle$

$W_{11} = \langle -2, -2, -2 \rangle + \langle 0, 1, -1 \rangle$ $\{V_1$ Fails$\}$

$= \langle -2, -1, -3 \rangle$

$W_{12} = \langle -2, -1, -3 \rangle + \langle -1, -1, 1 \rangle$ $\{V_3$ Fails$\}$

$= \langle -3, -2, -2 \rangle$

$W_{13} = \langle -3, -2, -2 \rangle + \langle 0, 1, -1 \rangle$ $\{V_1$ Fails$\}$

$= \langle -3, -1, -3 \rangle$

$W_{14} = \langle -3, -1, -3 \rangle + \langle 0, 1, -1 \rangle$ $\{V_2$ Fails$\}$

$= \langle -2, -1, -4 \rangle$

$W15 = \langle -2, -1, -4 \rangle + \langle -1, -1, 1 \rangle$ {V3 Fails}

$= \langle -3, -2, -3 \rangle$

$W16 = \langle -3, -2, -3 \rangle + \langle 1, 0, -1 \rangle$ {V2 Fails}

$= \langle -2, -2, -4 \rangle$

$W17 = \langle -2, -2, -4 \rangle + \langle -1, -1, 1 \rangle$ {V3 Fails}

$= \langle -3, -3, -3 \rangle$

$W18 = \langle -3, -3, -3 \rangle + \langle 0, 1, -1 \rangle$ {V1 Fails}

$= \langle -3, -2, -4 \rangle$

$W2 = -3, \quad W1 = -2, \quad W0 = \Theta = -4$

Succeeds for all vectors

PTA convergence

Statement of Convergence of PTA

- Statement:

  Whatever be the initial choice of weights and whatever be the vector chosen for testing, PTA converges if the vectors are from a linearly separable function.

Proof of Convergence of PTA

- Suppose w_n is the weight vector at the n^{th} step of the algorithm.
- At the beginning, the weight vector is w_0
- Go from w_i to w_{i+1} when a vector X_j fails the test $w_i X_j > 0$ and update w_i as

$$w_{i+1} = w_i + X_j$$

- Since Xjs form a linearly separable function,

$$\exists\, w^* \text{ s.t. } w^* X_j > 0 \ \forall j$$

Proof of Convergence of PTA (cntd.)

- Consider the expression

$$G(w_n) = \frac{w_n \cdot w^*}{|w_n|}$$

Important point to prove convergence is to think of this G(wn) function and work with it's boundness. It is bounded that's why PTA has to converge. Requirement: linearly separable function and you're done.

where w_n = weight at nth iteration

- $$G(w_n) = \frac{|w_n| \cdot |w^*| \cdot \cos\theta}{|w_n|}$$

where θ = angle between w_n and w^*

- $G(w_n) = |w^*| \cdot \cos\theta$
- $G(w_n) \le |w^*|$ (as $-1 \le \cos\theta \le 1$)

Behavior of Numerator of G

$w_n \,.\, w^* \;=\; (w_{n-1} + X^{n-1}_{fail}) \,.\, w^*$

- $= w_{n-1} \,.\, w^* + X^{n-1}_{fail} \,.\, w^*$
- $= (w_{n-2} + X^{n-2}_{fail}) \,.\, w^* + X^{n-1}_{fail} \,.\, w^* \,.....$
- $= w_0 \,.\, w^* + (X^0_{fail} + X^1_{fail} + ... + X^{n-1}_{fail}) \,.\, w^*$

  $w^*.X^i_{fail}$ is always positive: note carefully

- Suppose $|X_j| \geq \delta$, where δ is the minimum magnitude.
- Num of G $\geq |w_0 \,.\, w^*| + n\,\delta \,.\, |w^*|$
- So, numerator of G grows with n.

Behavior of Denominator of G

- $|w_n| = \sqrt{(w_n \,.\, w_n)}$ {the sq root extends over the whole expression}

$= \sqrt{(w_{n-1} + X^{n-1}_{fail})^2}$

$= \sqrt{(w_{n-1})^2 + 2.\, w_{n-1.}\, X^{n-1}_{fail} + (X^{n-1}_{fail})^2}$

$\leq \sqrt{(w_{n-1})^2 + (X^{n-1}_{fail})^2}$ (as $w_{n-1.}\, X^{n-1}_{fail} \leq 0$)

$\leq \sqrt{(w_0)^2 + (X^0_{fail})^2 + (X^1_{fail})^2 + \ldots + (X^{n-1}_{fail})^2}$

- $|X_j| \leq \rho$ (max magnitude)
- So, Denom $\leq \sqrt{(w_0)^2 + n\rho^2}$

Some Observations

- Numerator of G grows as n
- Denominator of G grows as $\sqrt{n}$

  => Numerator grows faster than denominator
- If PTA does not terminate, $G(w_n)$ values will become unbounded.

Some Observations contd.

- But, as $|G(w_n)| \leq |w^*|$ which is finite, this is impossible!
- Hence, PTA has to converge.
- Proof is due to Marvin Minsky.