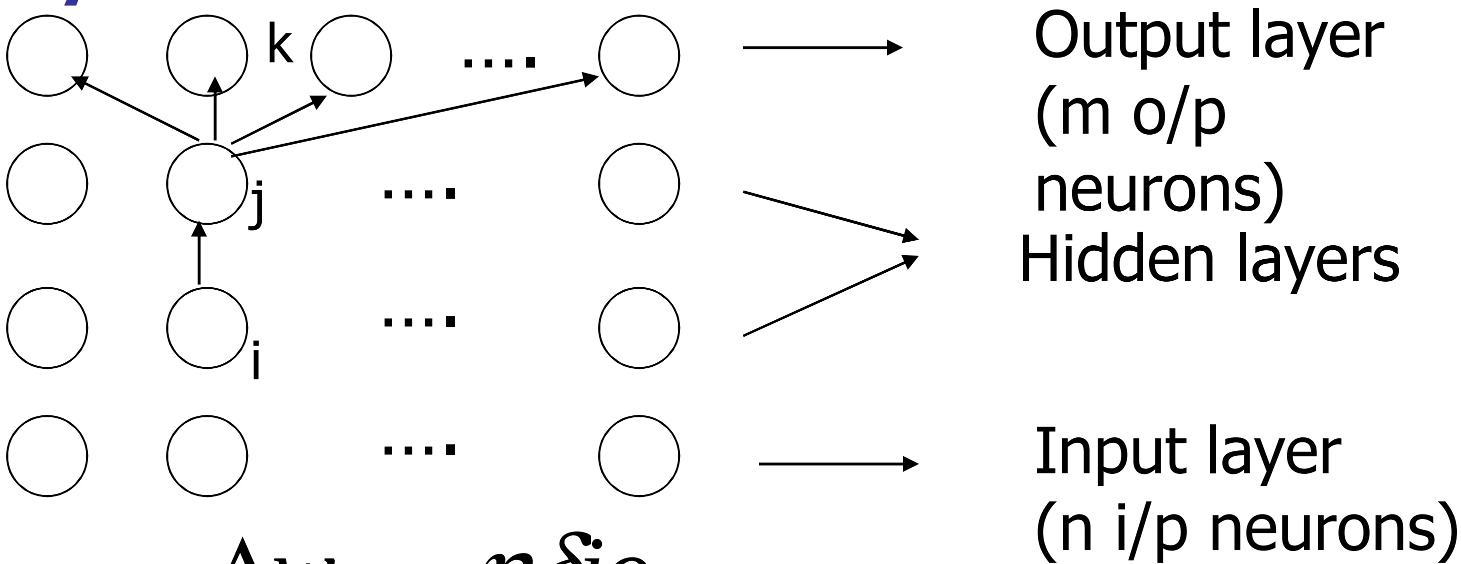
CS344: Introduction to Artificial Intelligence
(associated lab: CS386)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Lecture 36, 37: Hardness of training feed forward neural nets

11th and 12th April, 2011

Backpropagation for hidden layers



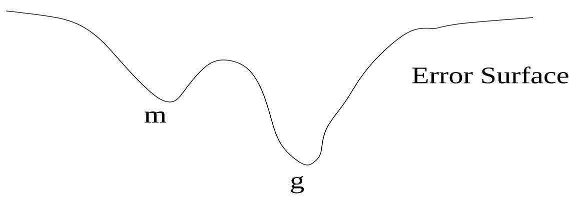
$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i$$

Local Minima

Due to the Greedy nature of BP, it can get stuck in local minimum *m* and will never be able to reach the global minimum *g* as the error can only decrease by weight change.



m- local minima, g- global minima

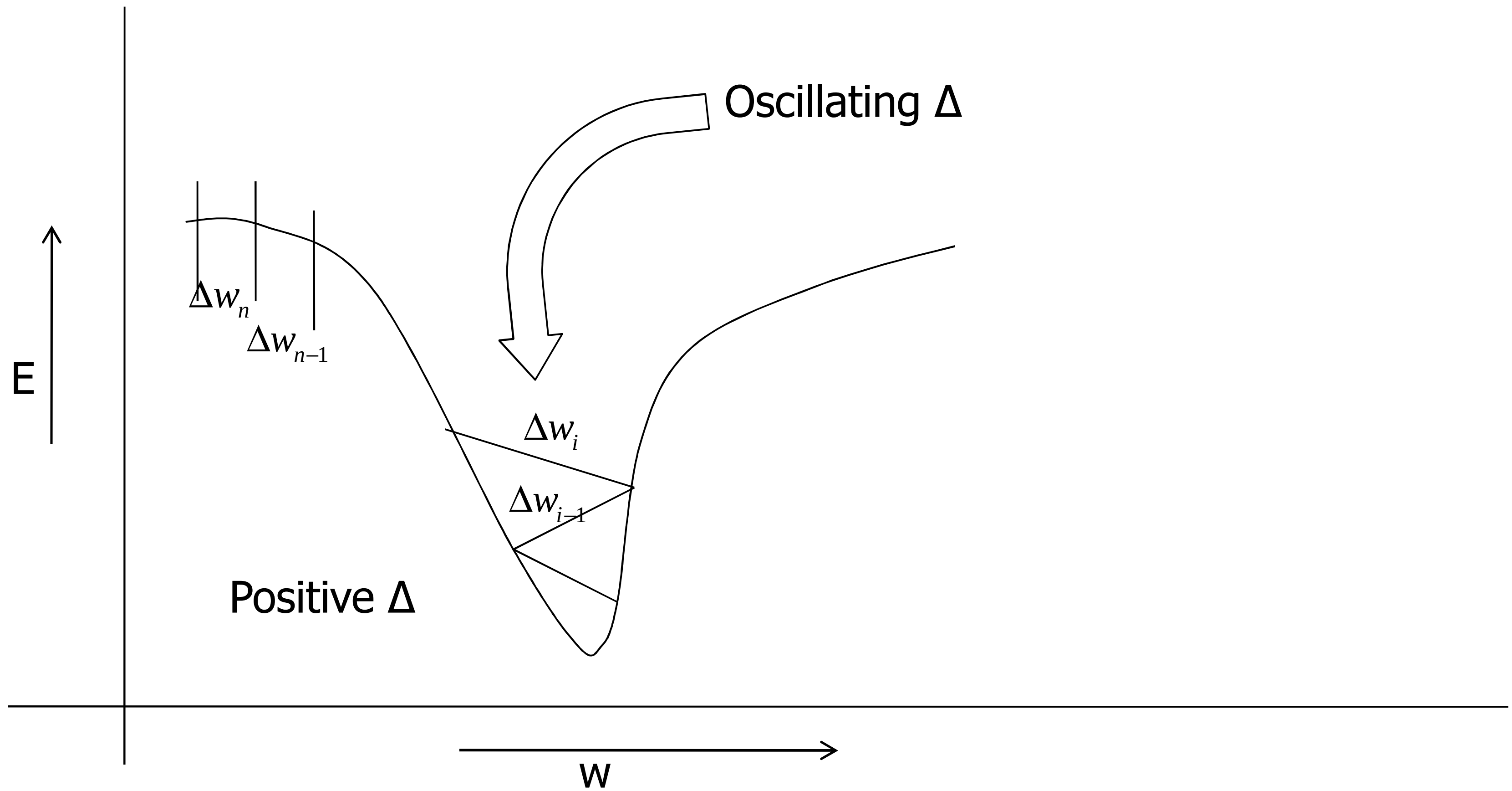
Figure- Getting Stuck in local minimum

Momentum factor

1. Introduce momentum factor.

$$(\Delta w_{ji})_{nth-iteration} = \eta \delta_j O_i + \beta (\Delta w_{ji})_{(n-1)th-iteration}$$

- Accelerates the movement out of the trough.
- Dampens oscillation inside the trough.
- Choosing β : If β is large, we may jump over the minimum.


Oscillating Δ
E
Δw_n
Δw_{n-1}
Δw_i
Δw_{i-1}
Positive Δ
w

Momentum factor

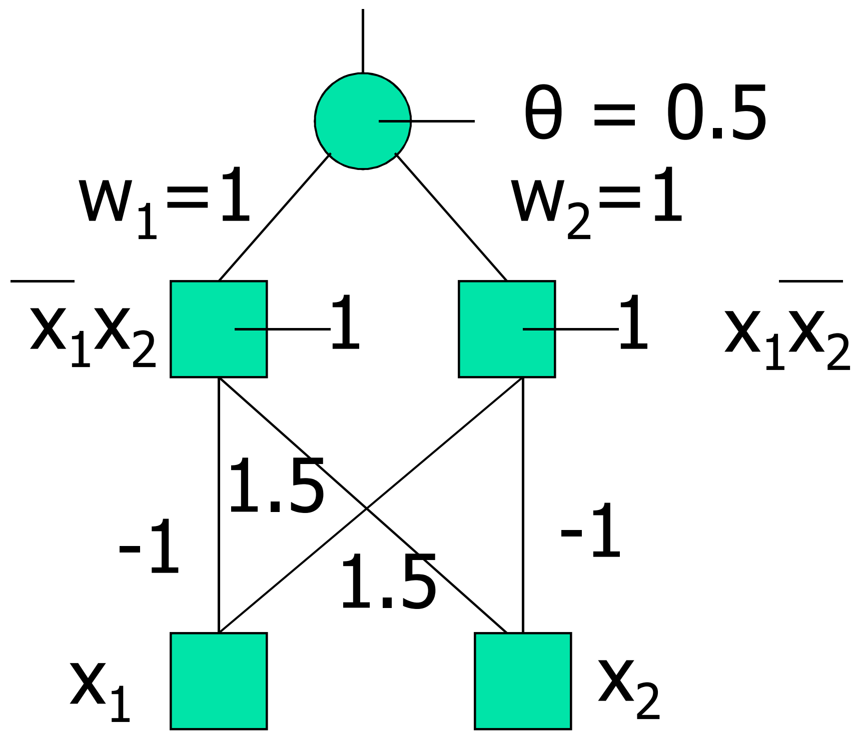
- If the momentum factor is very large

  $$\Delta w = (1 + \beta + \beta^2 + \beta^3 + \cdots)$$

  (GP series of β)
- β is *learning rate* (lies between 0 and 1)
- η is *momentum factor* (lies between 0 and 1)
- Generally, $\beta = \frac{1}{10} \times \eta$

Symmetry breaking

- If mapping demands different weights, but we start with the same weights everywhere, then BP will never converge.

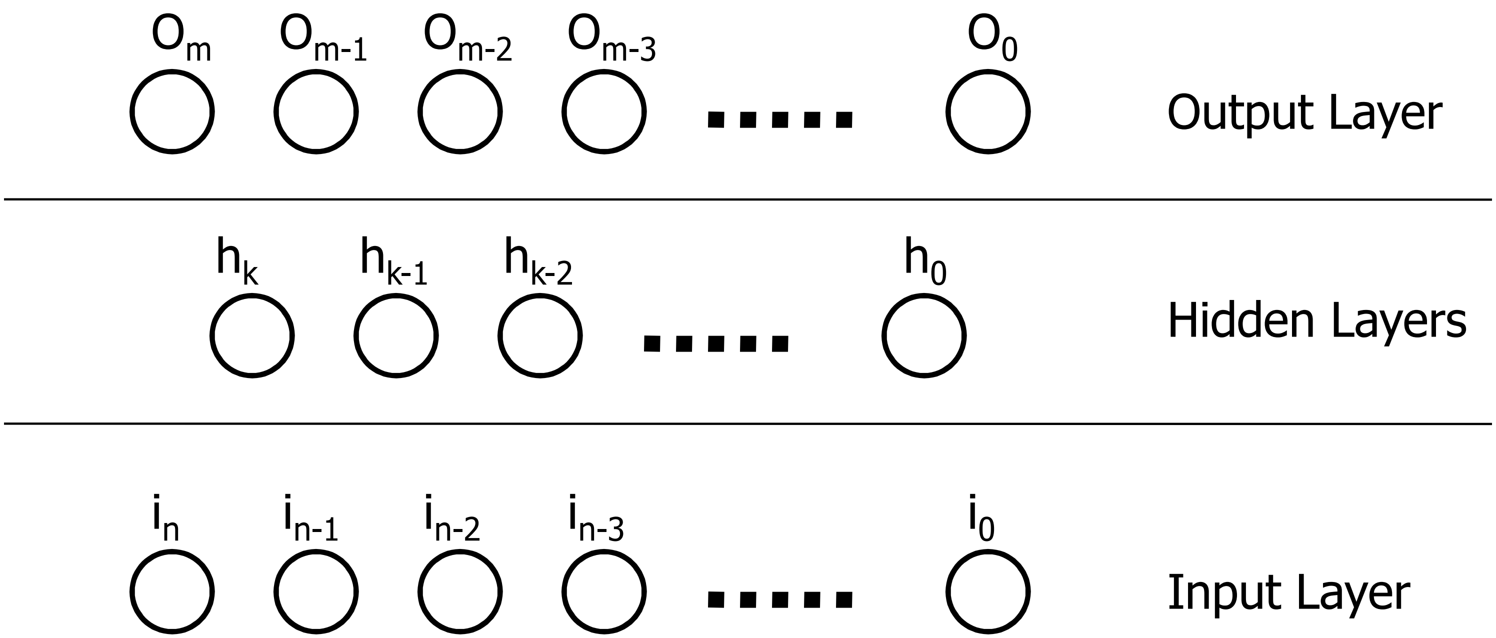


XOR n/w: if we started with identical weight everywhere, BP will not converge

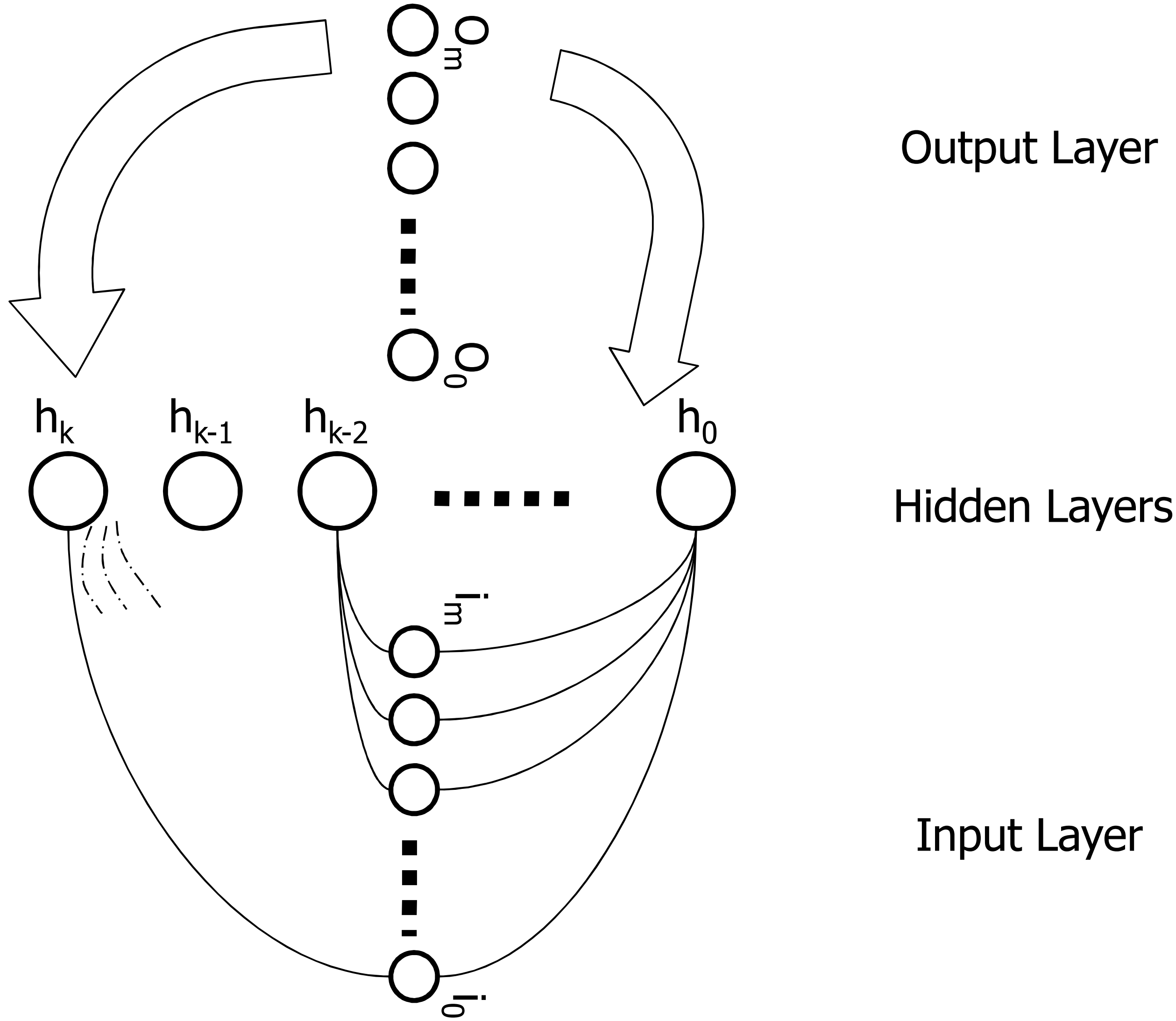
Symmetry breaking: simplest case

- If all the weights are same initially they will remain same over iterations

Symmetry Breaking: general case



Output layer and Input Layer form an axis.
Around the axis symmetry needs to be broken



Training of FF NN takes time!

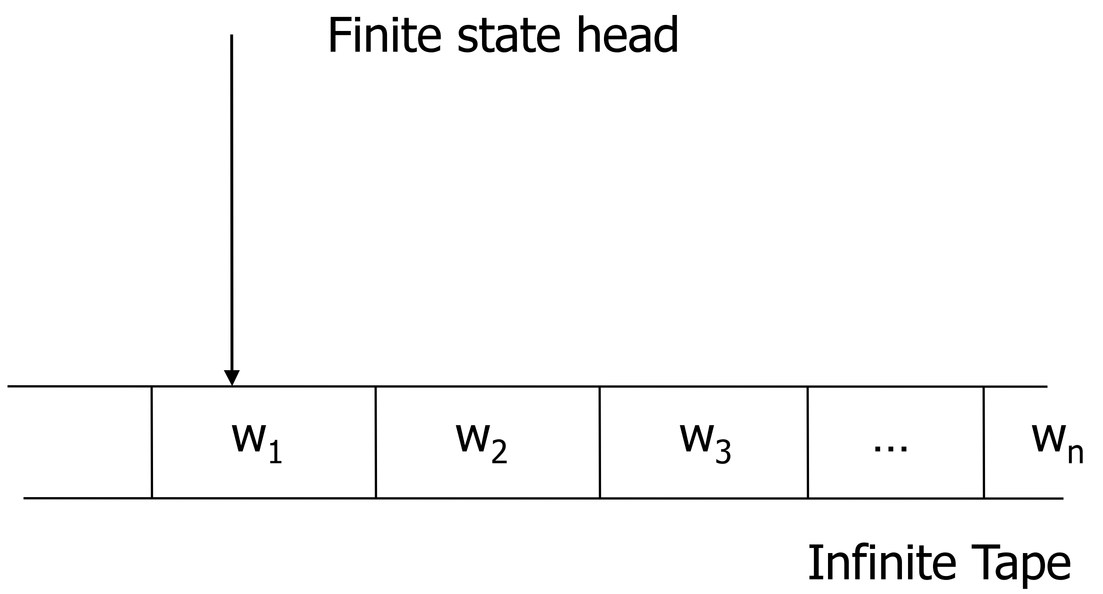
- BP + FFNN combination applied for many problems from diverse disciplines
- Consistent observation: the training takes time as the problem size increases
- Is there a hardness hidden soemwhere?

Hardness of Training Feedforward NN

- NP-completeness result:
  - *Avrim Blum, Ronald L. Rivest: Training a 3-node neural network is NP-complete. Neural Networks 5(1): 117-127 (1992)*Showed that the *loading problem* is hard
- As the number of training example increases, so does the training time EXPONENTIALLY

A primer on NP-completeness theory

Turing Machine



Formal Definition (vide: Hopcroft and Ullmann, 1978)

- A Turing machine is a 7-tuple $\langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
  - ***Q*** is a finite set of *states*
  - ***Γ*** is a finite set of the *tape alphabet/symbols*
  - ***b*** is the *blank symbol* (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
  - ***Σ***, a subset of ***Γ*** not including ***b*** is the set of *input symbols*
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a partial function called the *transition function*, where ***L*** is left shift, ***R*** is right shift.
  - $q_0 \in Q$ is the *initial state*
  - ***F*** is the set of *final* or *accepting states*

Non-deterministic and Deterministic Turing Machines

If δ is to a number of possibilities

$$\boldsymbol{\delta : Q \times \Gamma \rightarrow \{Q \times \Gamma \times \{L, R\}\}}$$

Then the TM is an NDTM; else it is a DTM

Decision problems

- Problems whose answer is *yes/no*
- For example,
  - *Hamilton Circuit:* Does an undirected graph have a path that visits every node and comes back to the starting node?
  - *Subset sum*: Given a finite set of integers, is there a subset of them that sums to 0?

The sets *NP* and *P*

- Suppose for a decision problem, **an NDTM is found** that takes time polynomial in the *length* of the input, then we say that the said problem is ***in NP***
- If, however, **a DTM is found** that takes time polynomial in the *length* of the input, then we say that the said problem is ***in P***

Relation between *P* and *NP*

- Clearly,
  - *P* is a *subset of NP*
- Is *P* a *proper subset of NP?*
- That is the *P = NP* question

The concept of NP-completeness
(informal definition)

- A problem is said to be *NP-complete,* if
  - ***It is in NP, and***
  - ***A known NP-complete problem is reducible TO it.***
- The 'first' NP-complete problem is
  - *satisfiability*: Given a Boolean Formula in Conjunctive Normal Form (CNF), does is have a satisfying assignment, *i.e.*, a set of 0-1 values for the constituting literals that makes the formula evaluate to 1? (even the restricted version of this problem- *3-sat-* is NP-complete)

Example of 3-sat

- $(x_1 + x_2 + x'_3)(x'_1 + x_3)$ is satisfiable: $x_2 = 1$ and $x_3 = 1$
- $x_1(x_2 + x_3)x'_1$ is not satisfiable.

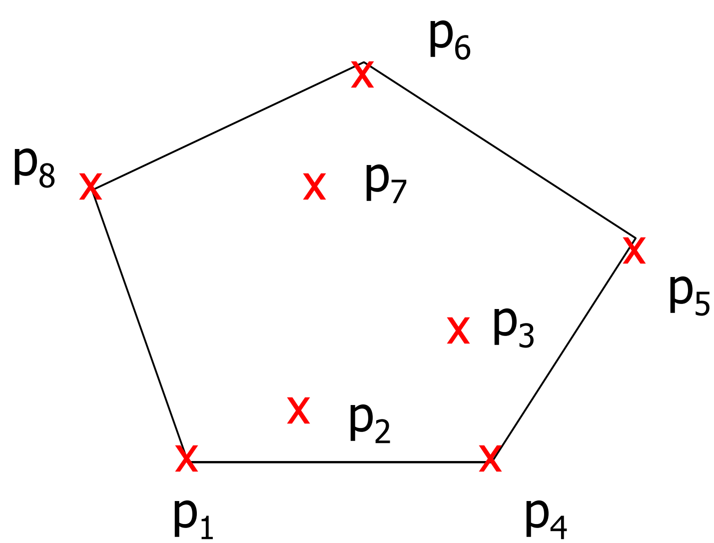
{x'_I means complement of x_i}

Numerous problems have been proven to be NP-complete

- The procedure is always the same:
- Take *an instance* of a *known* NP-complete problem; let this be *p.*
- Show a *polynomial time Reduction* of *p* TO an instance *q* of the problem whose status is being investigated.
- Show that the answer to *q* is *yes*, if and only if the answer to *p* is *yes.*

Clarifying the notion of *Reduction*

- Convex Hull problem:
  - Given a set of points on the two dimensional plane, find the *convex hull* of the points



P_1, p_4, p_5, p_6 and p_8 are on the convex hull

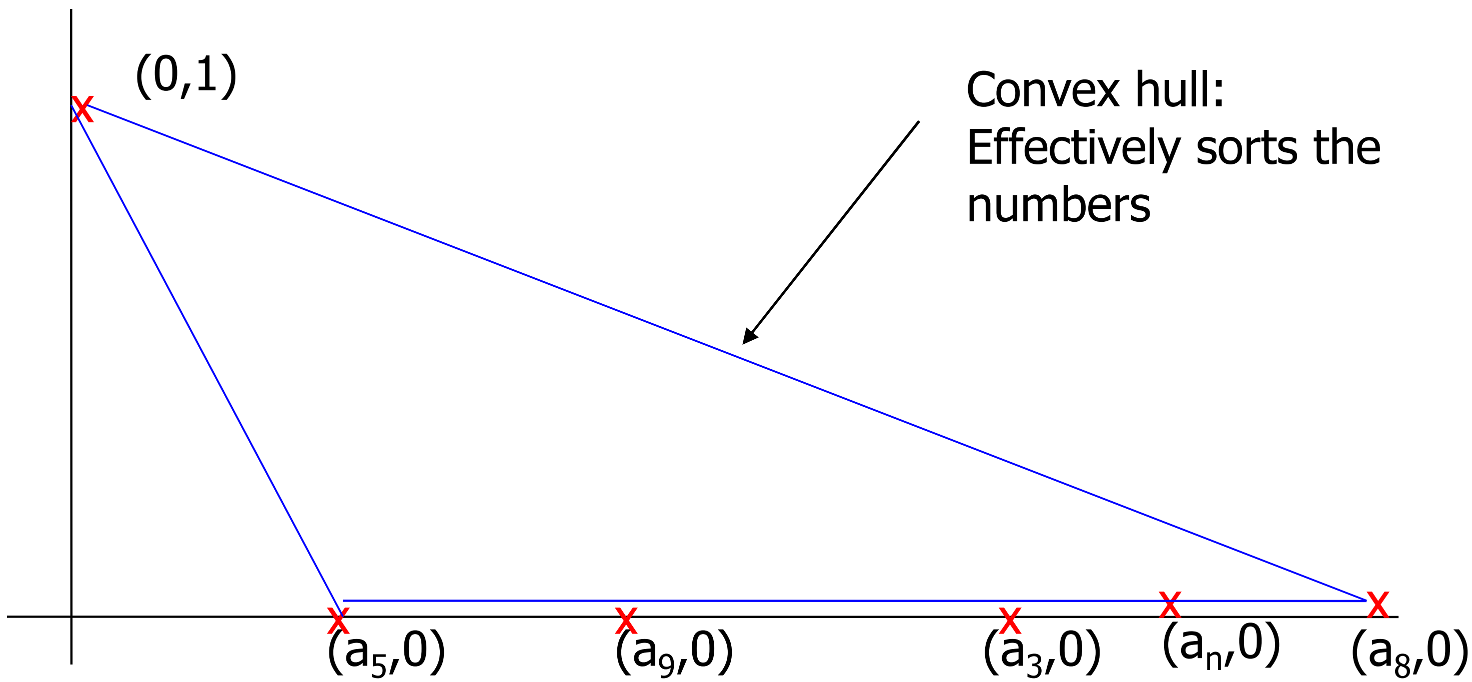
Complexity of convex hull finding problem

- We will show that this is *O(nlogn).*
- Method used is *Reduction.*
- The most important first step: *choose the right problem.*
- We take *sorting* whose complexity is known to be *O(nlogn)*

Reduce Sorting to Convex hull (*caution: NOT THE OTHER WAY*)

- Take *n* numbers $a_1, a_2, a_3, \ldots, a_n$ which are to be sorted.
- This is an instance of a sorting problem.
- ***From*** this obtain an instance of a convex hull problem.
- Find the convex hull of the set of points
  - $\langle 0,1\rangle, \langle a_1,0\rangle, \langle a_2,0\rangle, \langle a_3,0\rangle, \ldots, \langle a_n,0\rangle$
- This transformation takes *linear time* in the length of the input

Pictorially…



Convex hull finding is $O(nlogn)$

- If the complexity is lower, *sorting too has lower complexity*
- Because by the linear time procedure shown, ANY instance of the sorting problem can be converted to an instance of the CH problem and solved.
- *This is not possible.*
- Hence CH is $O(nlogn)$

Important remarks on reduction

Problem

Input Situation

i_1
i_2
i_3
.
.
.
i_m

Algorithm

A_1
A_2
.
.
.
A_n

Case Scenario : $\langle A_p, i_q \rangle$ Algorithm A_p on input i_q

Worst Case of best algorithm

- Case Scenario : $<A_p, i_q>$

  Algorithm A_p on input i_p
- Worst Case of best algorithm

  $<A^{\uparrow}, i_{\downarrow}>$
- Time complexity $O(|i_{\downarrow}|)$

  $|i_{\downarrow}|$ length of $i_{\downarrow}$

Example

Sorting (ascending)

Input

<3, 1.5, 9, 11, … >

<4, 2, 1, 5, … >

etc

Algorithm

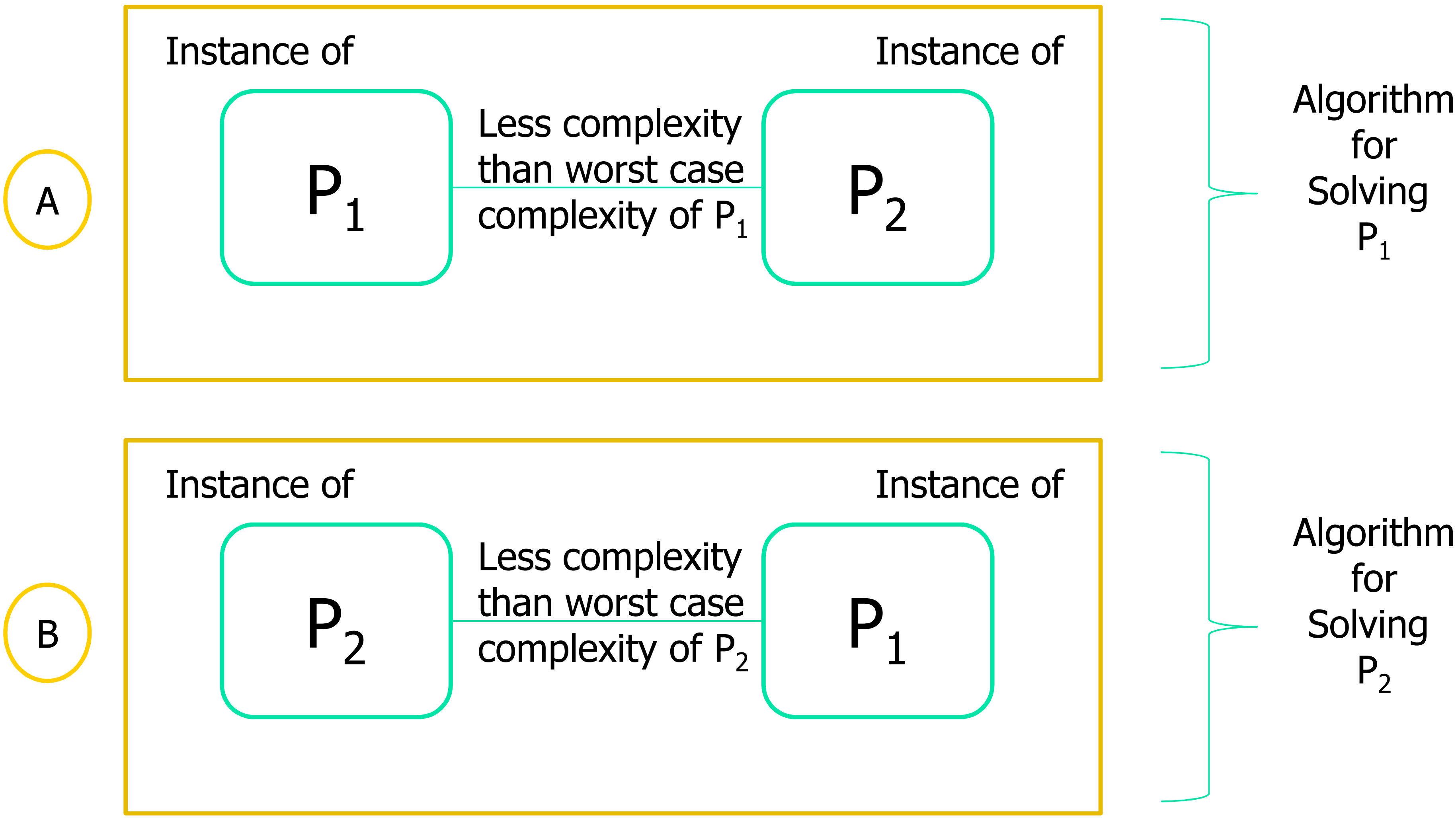
Bubble Sort

Heap Sort

Merge Sort

Best Algorithm : Quicksort
Worst Case: Already sorted sequence

Transformations



P_1

1. Sorting
2. Set Splitting

P_2

1. Convex Hull
2. Linear Confinement

We know the worst case complexity of P_1

Worst case complexity of P_1

$P_2 \rightarrow P_1$: new algorithm for solving P_2

- For any problem

  Situation A when an algorithm is discovered, and its worst case complexity calculated, the effort will continuously be to find a better algorithm. That is to improve upon the worst case complexity.

  Situation B Find a problem P_1 whose worst case complexity is known and transform it to the unknown problem with less complexity. That puts a seal on how much improvement can be done on the worst case complexity.

Worst case complexity of best complexity $O(n\log n)$

Example : Sorting

Worst complexity of bubble sort $O(n^2)$

<P, A, I>:
<Problem, Algorithm, Input>:
the trinity of complexity theory

Training of 1 hidden layer 2 neuron feed forward NN is NP-complete

Numerous problems have been proven to be NP-complete

- The procedure is always the same:
- Take *an instance* of a *known* NP-complete problem; let this be *p.*
- Show a *polynomial time Reduction* of *p* TO an instance *q* of the problem whose status is being investigated.
- Show that the answer to *q* is *yes*, if and only if the answer to *p* is *yes.*

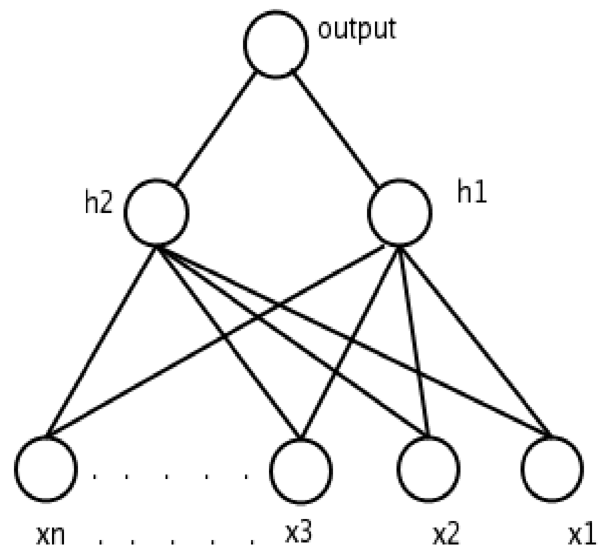
Training of NN

- Training of Neural Network is NP-hard
- This can be proved by the NP-completeness theory

- Question
  - Can a set of examples be loaded onto a Feed Forward Neural Network efficiently?

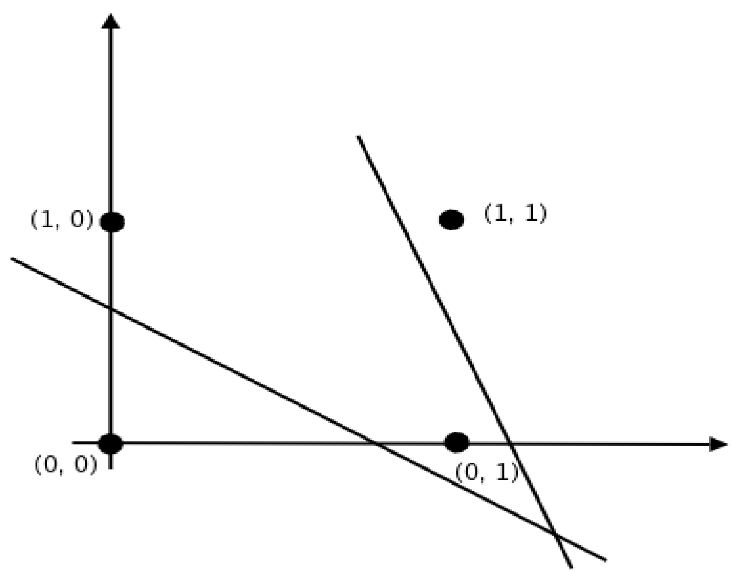
Architecture

- We study a special architecture.
- Train the neural network called 3-node neural network of feed forward type.
- ALL the neurons are 0-1 threshold neurons



Architecture

- h_1 and h_2 are hidden neurons
- They set up hyperplanes in the (n+1) dimensions space.



Confinement Problem

- Can two hyperplanes be set which confine ALL and only the positive points?
- Positive Linear Confinement problem is NP-Complete.
- Training of positive and negative points needs solving the CONFINEMENT PROBLEM.

Solving with Set Splitting Problem

- Set Splitting Problem
- Statement:
  - Given a set S of n elements $e_1, e_2, \ldots, e_n$ and a set of subsets of S called as concepts denoted by $c_1, c_2, \ldots, c_m$, does there exist a splitting of S
  - i.e. are there two sets S_1 (subset of S) and S_2 (subset of S) and none of $c_1, c_2, \ldots, c_m$ is subset of S_1 or S_2

Set Splitting Problem: example

- Example

$S = \{s_1, s_2, s_3\}$

$c_1 = \{s_1, s_2\}$, $c_2 = \{s_2, s_3\}$

Splitting exists

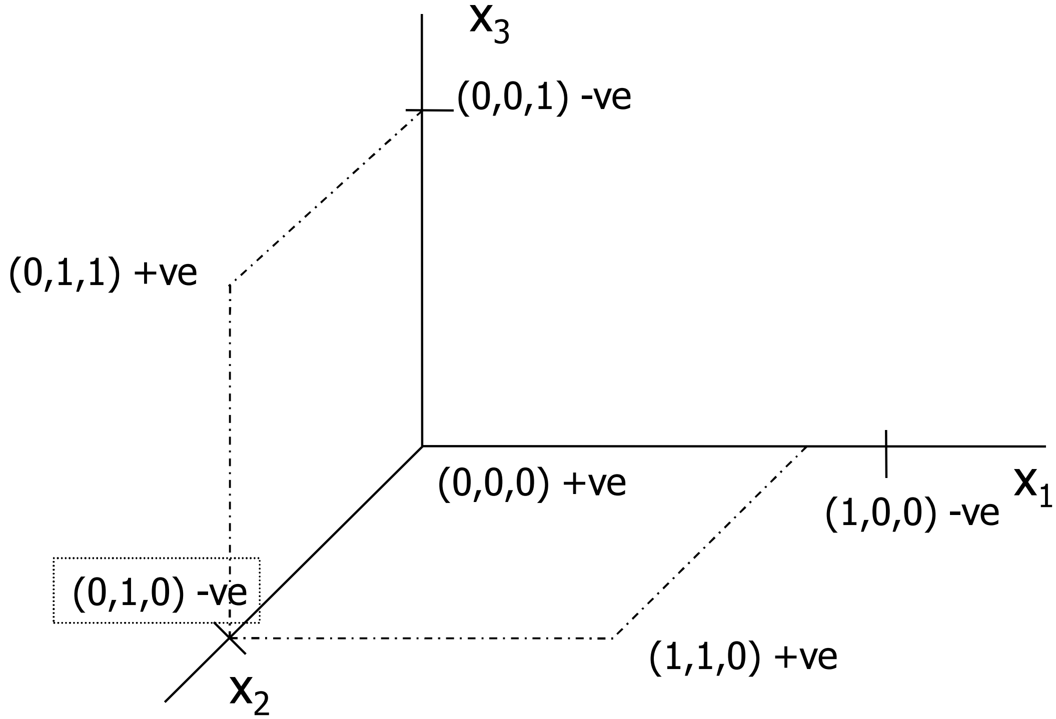
$S_1 = \{s_1, s_3\}$, $S_2 = \{s_2\}$

Transformation

- For *n* elements in S, set up an *n*-dimensional space.
- Corresponding to each element mark a negative point at unit distance in the axes.
- Mark the origin as positive
- For each concept mark a point as positive.

Transformation

- $S = \{s_1, s_2, s_3\}$
- $c_1 = \{s_1, s_2\}$, $c_2 = \{s_2, s_3\}$



Proving the transformation

- Statement
  - *Set-splitting problem has a solution if and only if positive linear confinement problem has a solution.*
- Proof in two parts: **if part** and **only if** part
- If part
  - *Given* Set-splitting problem has a solution.
  - *To show* that the constructed Positive Linear Confinement (PLC) problem has a solution
  - *i.e.* to show that since S_1 and S_2 exist, P_1 and P_2 exist which confine the positive points

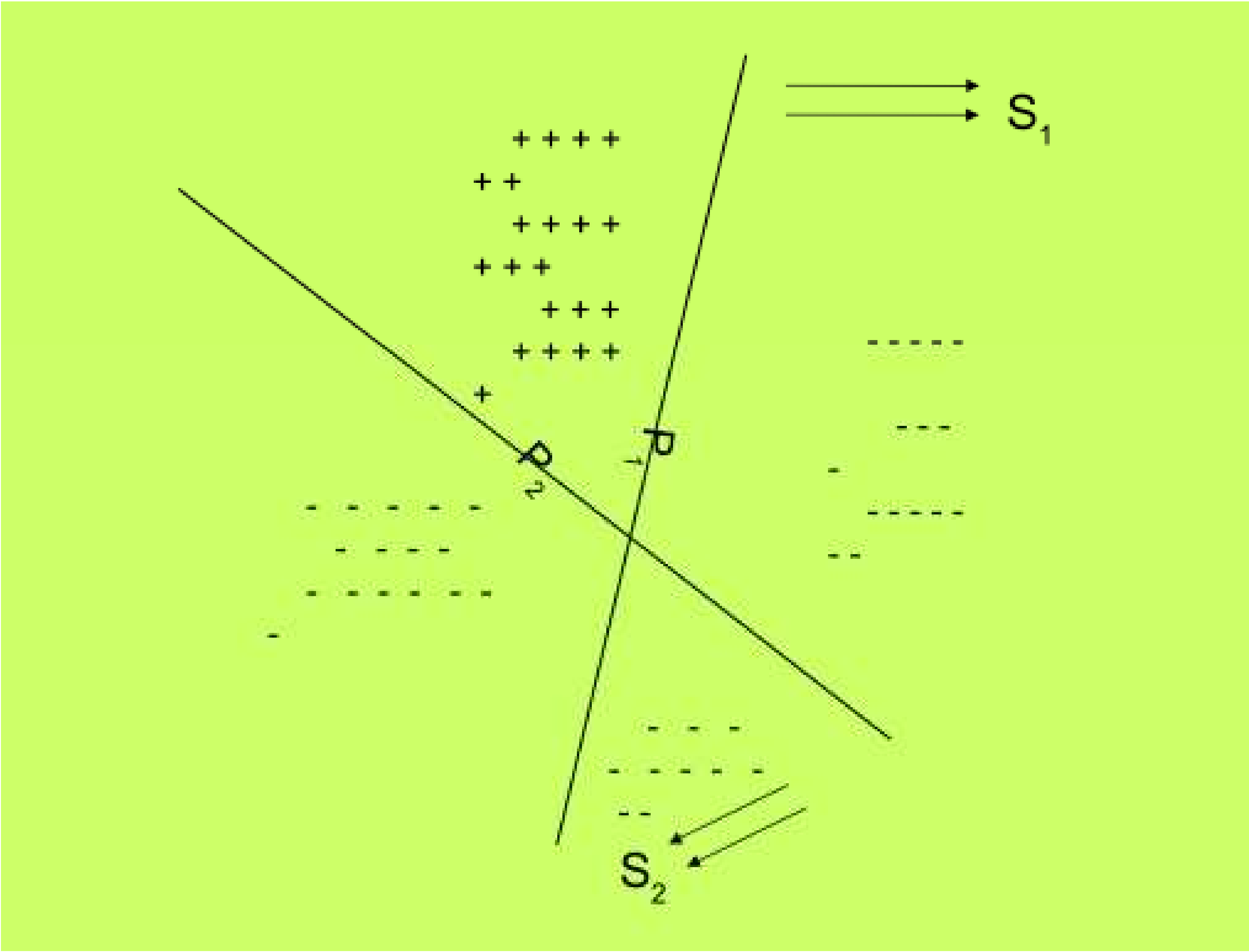
Proof – If part

- P_1 and P_2 are as follows:
  - $P_1 : a_1x_1 + a_2x_2 + \ldots + a_nx_n = -1/2$ -- Eqn A
  - $P_2 : b_1x_1 + b_2x_2 + \ldots + b_nx_n = -1/2$ -- Eqn B

$$a_i = -1, \quad \text{if } s_i \in S_1$$
$$= n, \quad \text{otherwise}$$

$$b_i = -1, \quad \text{if } s_i \in S_2$$
$$= n, \quad \text{otherwise}$$

Representative Diagram



Proof (If part) – Positive points

- For origin (a +ve point), plugging in $x_1 = 0 = x_2 = \ldots = x_n$ into P_1 we get, $0 > -1/2$
- For other points
  - +ve points correspond to c_i's
  - Suppose c_i contains elements $\{s_1^i, s_2^i, \ldots, s_n^i\}$, then at least one of the s_j^i cannot be in S_1

    $\therefore$ co-efficient of $x_j^i = n$,

    $\therefore$ LHS $> -1/2$
- Thus +ve points for each c_i belong to the same side of P_1 as the origin.
- Similarly for P_2.

Proof (If part) – Negative points

- -ve points are the unit distance points on the axes
  - They have only one bit as 1.
  - Elements in S_1 give rise to m_1 -ve points.
  - Elements in S_2 give rise to m_2 -ve points.
- -ve points corresponding to S_1
  - If $q_i \varepsilon S_1$ *then* x_i *in* P_1 *must have co-efficient -1*

    $\therefore LHS = -1 < -1/2$

What has been proved

- Origin (+ve point) is on one side of P_1
- +ve points corresponding to c_i*'s* are on the same side as the origin.
- -ve points corresponding to S_1 are on the opposite side of P_1

Illustrative Example

- Example
  - $S = \{s_1, s_2, s_3\}$
  - $c_1 = \{s_1, s_2\}$, $c_2 = \{s_2, s_3\}$
  - Splitting : $S_1 = \{s_1, s_3\}$, $S_2 = \{s_2\}$
- +ve points:
  - (<0, 0, 0>,+), (<1, 1, 0>,+), (<0, 1, 1>,+)
- -ve points:
  - (<1, 0, 0>,-), (<0, 1, 0>,-), (<0, 0, 1>,-)

Example (contd.)

- The constructed planes are:
- P_1 :
  - $a_1x_1 + a_2x_2 + a_3x_3 = -1/2$
  - $-x_1 + 3x_2 - x_3 = -1/2$
- P_2:
  - $b_1x_1 + b_2x_2 + b_3x_3 = -1/2$
  - $3x_1 - x_2 + 3x_3 = -1/2$

Example (contd.)

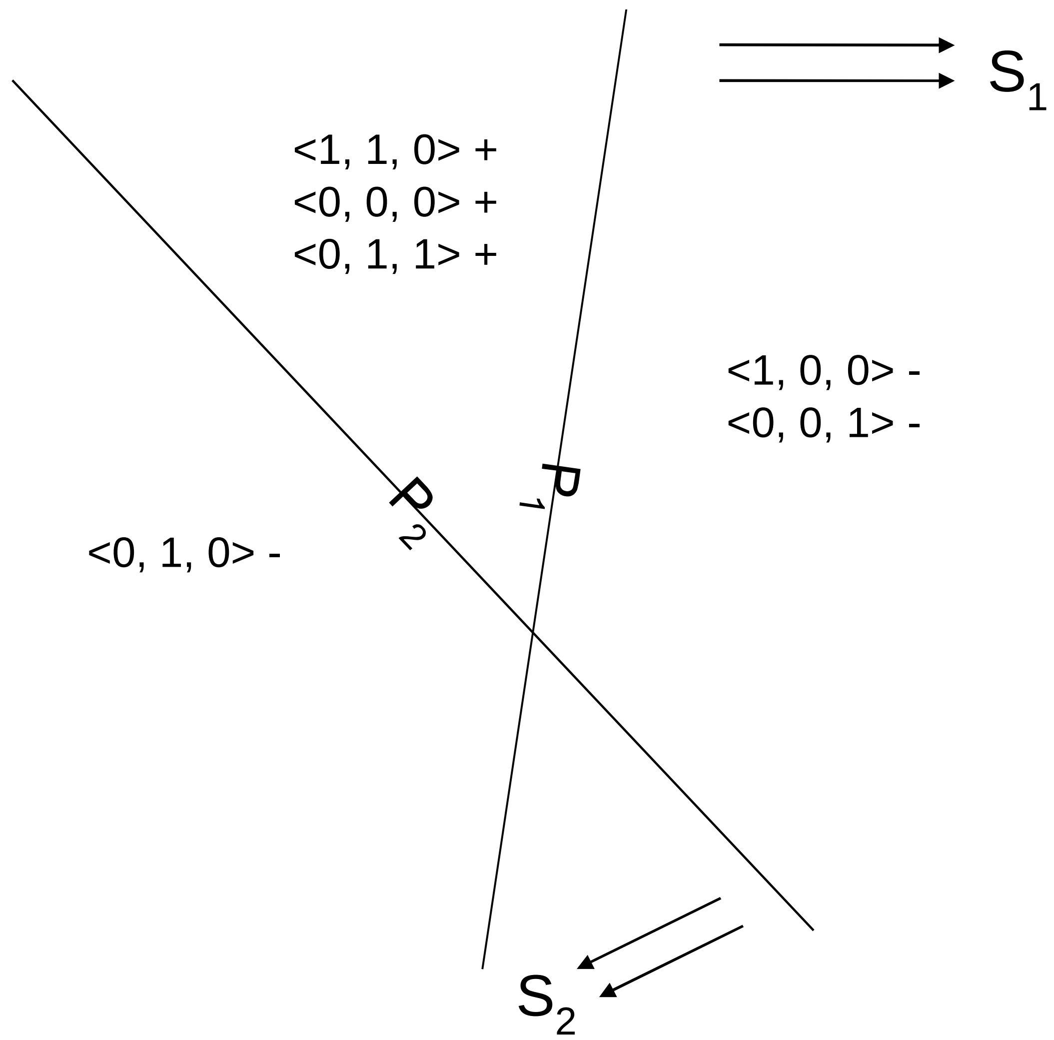
- P_1: $-x_1 + 3x_2 - x_3 = -1/2$
- <0, 0, 0>: LHS = 0 > -1/2,
  - ∴ <0, 0, 0> is +ve pt (similarly, <1,1,0> and <0,1,1> are classified as +ve)
- <1, 0, 0>: LHS = -1 < -1/2,
  - ∴ <1, 0, 0> is -ve pt
- <0, 0, 1>: LHS = -1 < -1/2,
  - ∴ <0, 0, 1> is -ve pt

But <0,1,0> is classified as +ve, i.e., cannot classify the point of S_2.

Example (contd.)

- $P_2 : 3x_1 - x_2 + 3x_3 = -1/2$
- <0, 0, 0> : LHS = 0 > -1/2
  - ∴ <0, 0, 0> is +ve pt
- <1, 1, 0> : LHS = 2 > -1/2
  - ∴ <1, 1, 0> is +ve pt
- <0, 1, 1> : LHS = 2 > -1/2
  - ∴ <0, 1, 1> is +ve pt
- <0, 1, 0> : -1 < -1/2
  - ∴ <0, 1, 0> is -ve pt

Graphic for Example



Proof – Only if part

- Given +ve and -ve points constructed from the set-splitting problem, two hyperplanes P_1 and P_2 have been found which do positive linear confinement
- To show that S can be split into S_1 and S_2

Proof - Only if part (contd.)

- Let the two planes be:
  - $P_1: a_1x_1 + a_2x_2 + \dots + a_nx_n = \theta_1$
  - $P_2 : b_1x_1 + b_2x_2 + \dots + b_nx_n = \theta_2$
- Then,
  - S_1 = {elements corresponding to -ve points separated by P_1}
  - S_2 = {elements corresponding to -ve points separated by P_2}

Proof - Only if part (contd.)

- Since P_1 and P_2 *take care of* **all** -ve points, their union is equal to S ... (proof obvious)
- **To show:** No c_i is a subset of S_1 and S_2
- *i.e.*, there is in c_i at least one element $\notin S_1$

  -- *Statement (A)*

Proof - Only if part (contd.)

- Suppose $c_i \subset S_1$, then every element in c_i is contained in S_1
- *Let* $e_1^i, e_2^i, \ldots, e_{mi}^i$ *be the elements of* c_i *corresponding to each element*
- *Evaluating for each co-efficient, we get,*
  - $a_1 < \theta_1, \quad a_2 < \theta_1, \ \ldots, \ a_{mi} < \theta_1$ -- *(1)*
  - *But* $a_1 + a_2 + \ldots + a_m > \theta_1$ -- *(2)*
  - *and* $0 > \theta_1$ -- *(3)*
- *CONTRADICTION*

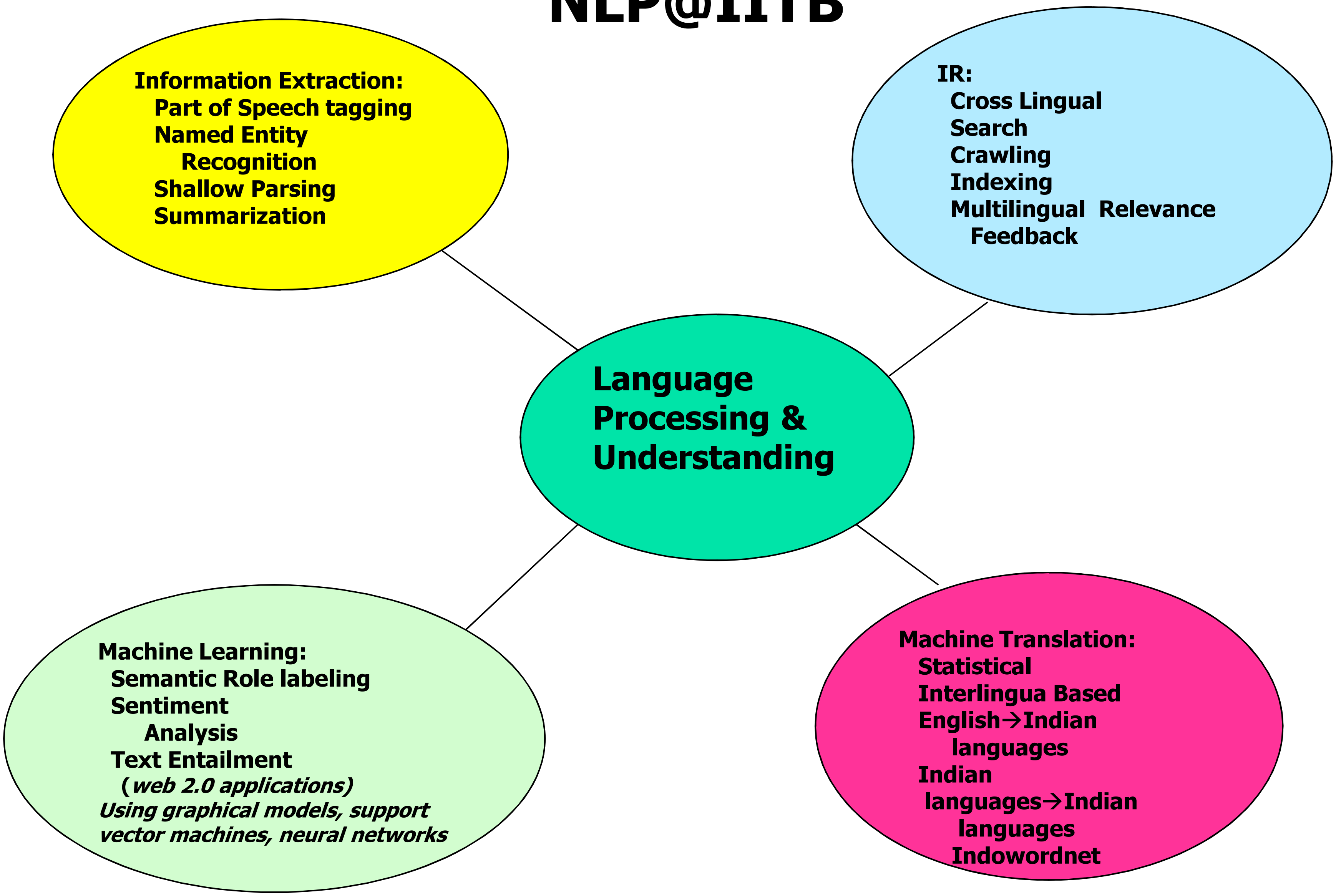
What has been shown

- Positive Linear Confinement is NP-complete.
- Confinement on any set of points of one kind is NP-complete (easy to show)
- The architecture is special- only one hidden layer with two nodes
- The neurons are special, 0-1 threshold neurons, NOT sigmoid
- Hence, can we generalize and say that FF NN training is NP-complete?
- Not rigorously, perhaps; but strongly indicated

Summing up

- Some milestones covered
  - A* Search
  - Predicate Calculus, Resolution, Prolog
  - HMM, Inferencing, Training
  - Perceptron, Back propagation, NP-completeness of NN Training
- Lab: to reinforce understanding of lectures
- Important topics left out: Planning, IR (advanced course next sem)
- Seminars: breadth and exposure
- Lectures: Foundation and depth

NLP@IITB



Resources: http://www.cfilt.iitb.ac.in
Publications: http://www.cse.iitb.ac.in/~pb

Linguistics is the eye and computation the body