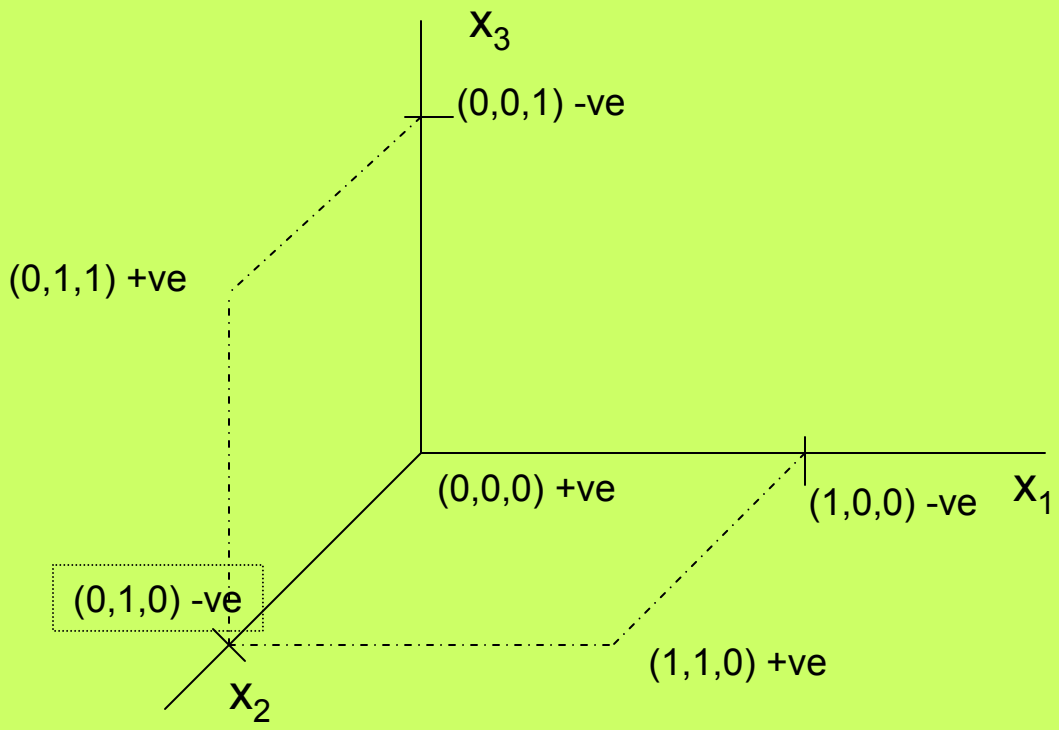
CS623: Introduction to Computing with Neural Nets *(lecture-9)*

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Transformation from set-splting to positive linear confinement: for proving NP-completeness of the latter

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



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 of *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

$$a_i = -1, \quad \text{if } s_i \in S_1$$
$$= n, \quad \text{otherwise}$$
$$b_i = -1, \quad \text{if } s_i \in S_2$$
$$= n, \quad \text{otherwise}$$

What is 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

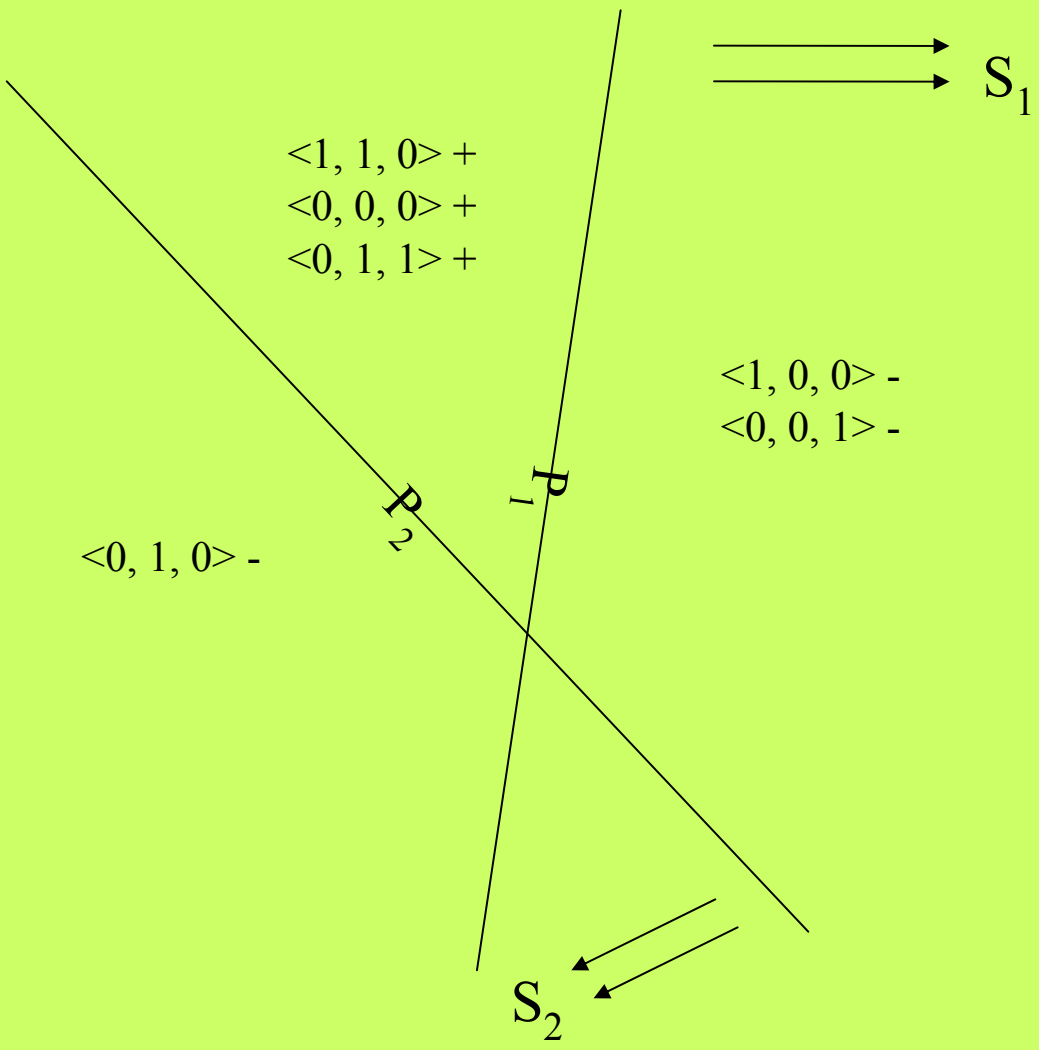
Illustrative Example

- 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:
  - $b_1x_1 + b_2x_2 + b_3x_3 = -1/2$
  - $3x_1 - x_2 + 3x_3 = -1/2$

Graphic for Example



Proof of *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: $a_1x_1 + a_2x_2 + \ldots + a_nx_n = \theta_1$
  - P_2 : $b_1x_1 + b_2x_2 + \ldots + b_nx_n = \theta_2$
- 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.

- 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*

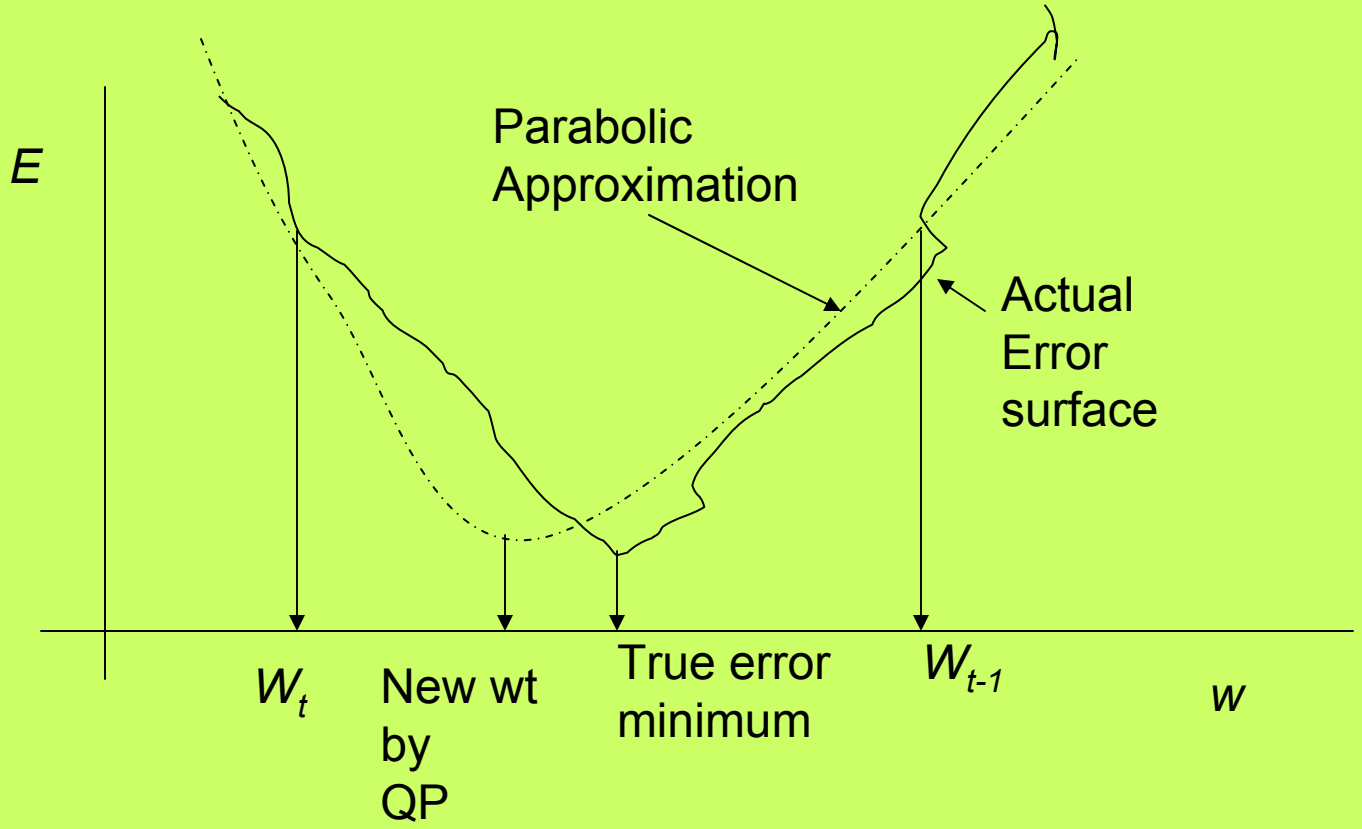
What has been shown

- 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

Accelerating Backpropagation

Quickprop: *Fahlman, '88*

- Assume a parabolic approximation to the error surface



Quickprop basically makes use of the second derivative

- $E = aw^2 + bw + c$
- First derivative, $E' = \delta E / \delta w = 2aw + b$
- $E'(t-1) = 2aw(t-1) + b$ --- (1)
- $E'(t) = 2aw(t) + b$ --- (2)
- Solving,
  - $a = [E'(t) - E'(t-1)]/2[w(t) - w(t-1)]$
  - $B = E'(t) - [E'(t) - E'(t-1)]w(t)/[w(t) - w(t-1)]$

Next weight *w(t+1)*

- This is found by minimizing *E(t+1)*
- Set *E'(t+1)* to 0
- *2aw(t+1) + b = 0*
- Substituting for *a* and *b*
  - *w(t+1)-w(t)= [w(t)-w(t-1)] X [E'(t)]/[E'(t-1)-E'(t)]*
  - *i.e.* $\Delta w(t) = \Delta w(t-1) \dfrac{E'(t)}{E'(t-1)-E'(t)}$

How to fix the hidden layer

By trial and error mostly

- No theory yet
- Read papers by Eric Baum
- Heuristics exist like the *mean* of the sizes of the i/p and the o/p layers (arithmetic, geometric and harmonic)

Grow the network: *Tiling Algorithm*

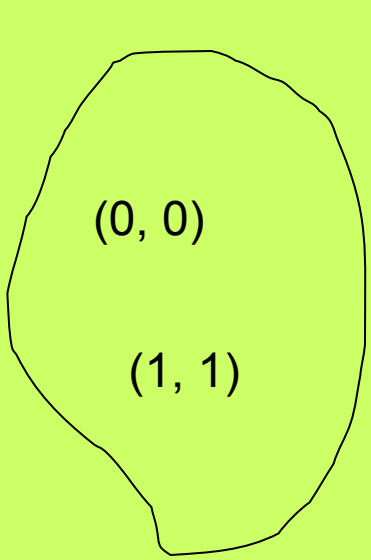
- A kind of divide and conquer strategy
- Given the *classes in the data*, run the perceptron training algorithm
- If linearly separable, convergence without any hidden layer
- If not, do as well as you can (*pocket* algorithm*)*
- This will produce classes with misclassified points

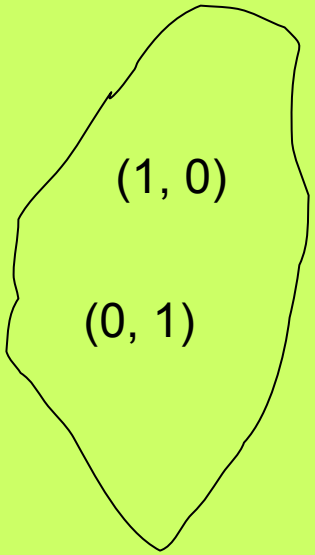
Tiling Algorithm *(contd)*

- Take the class with misclassified points and break into subclasses which contain no *outliers*
- Run PTA again *after recruiting* the required number of *perceptrons*
- Do this until *homogenous* classes are obtained
- Apply the same procedure for the first hidden layer to obtain the second hidden layer and so on

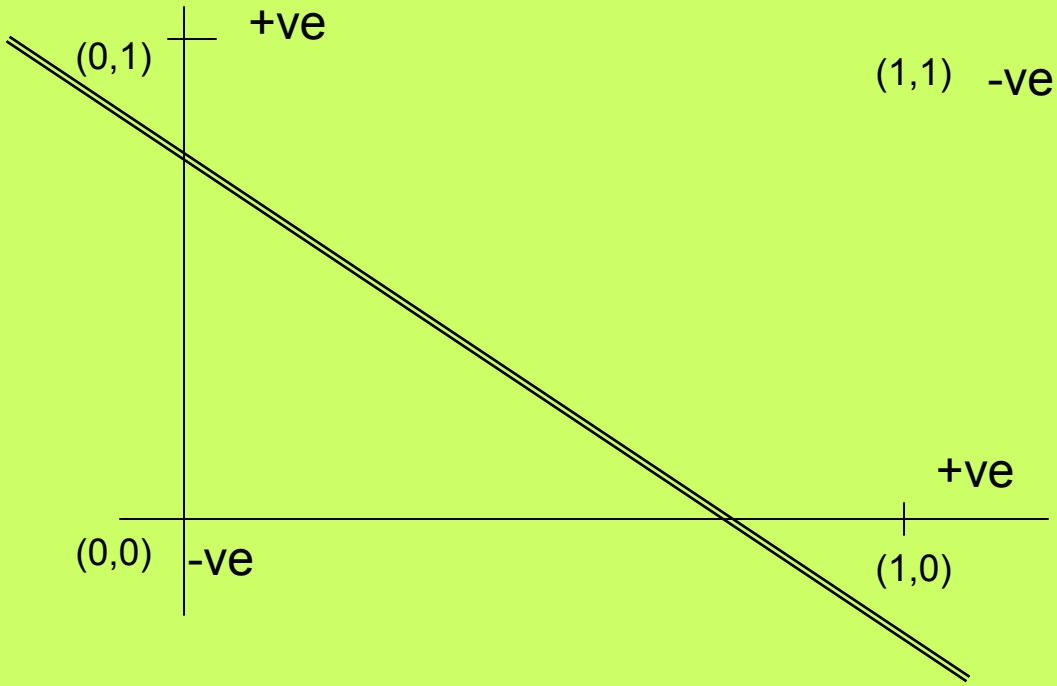
Illustration

- XOR problem
- Classes are

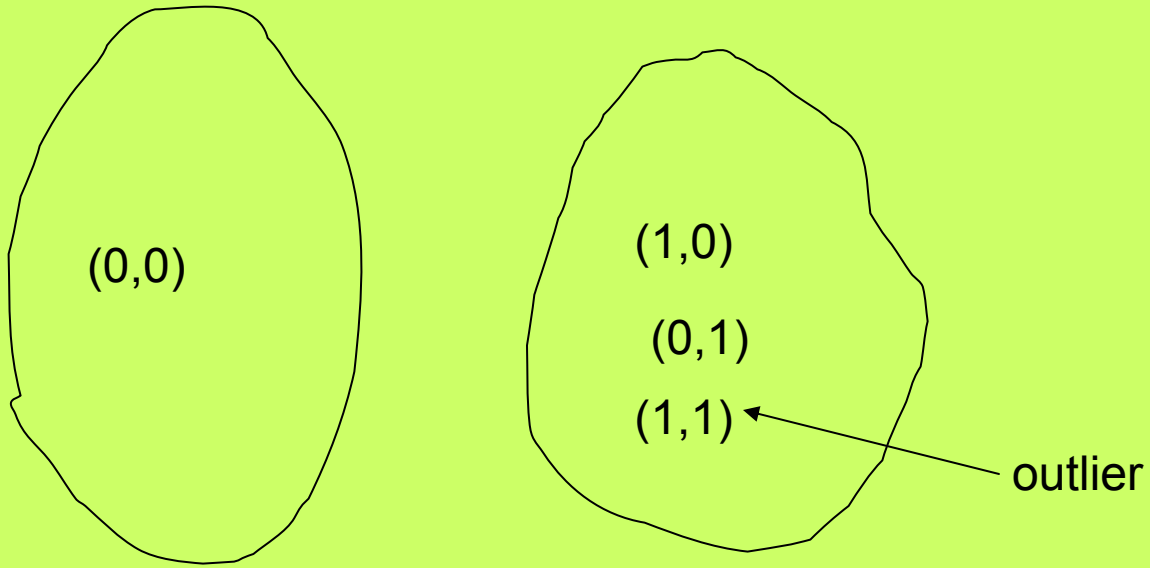




As best a classification as possible

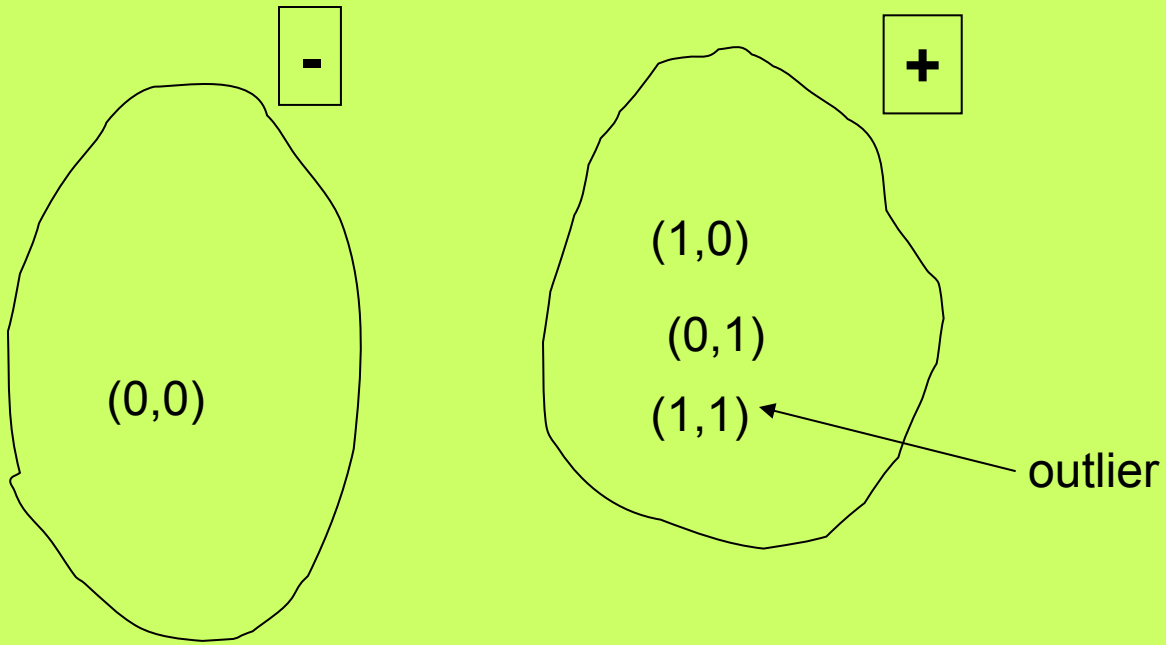


Classes with error



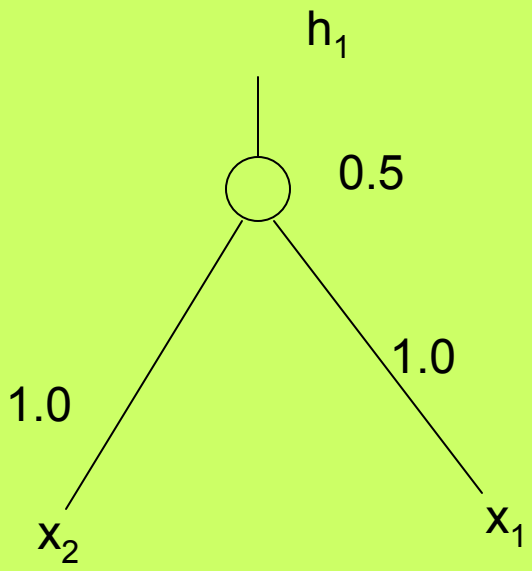
How to achieve this classification

- Give the labels as shown: eqv to an OR problem

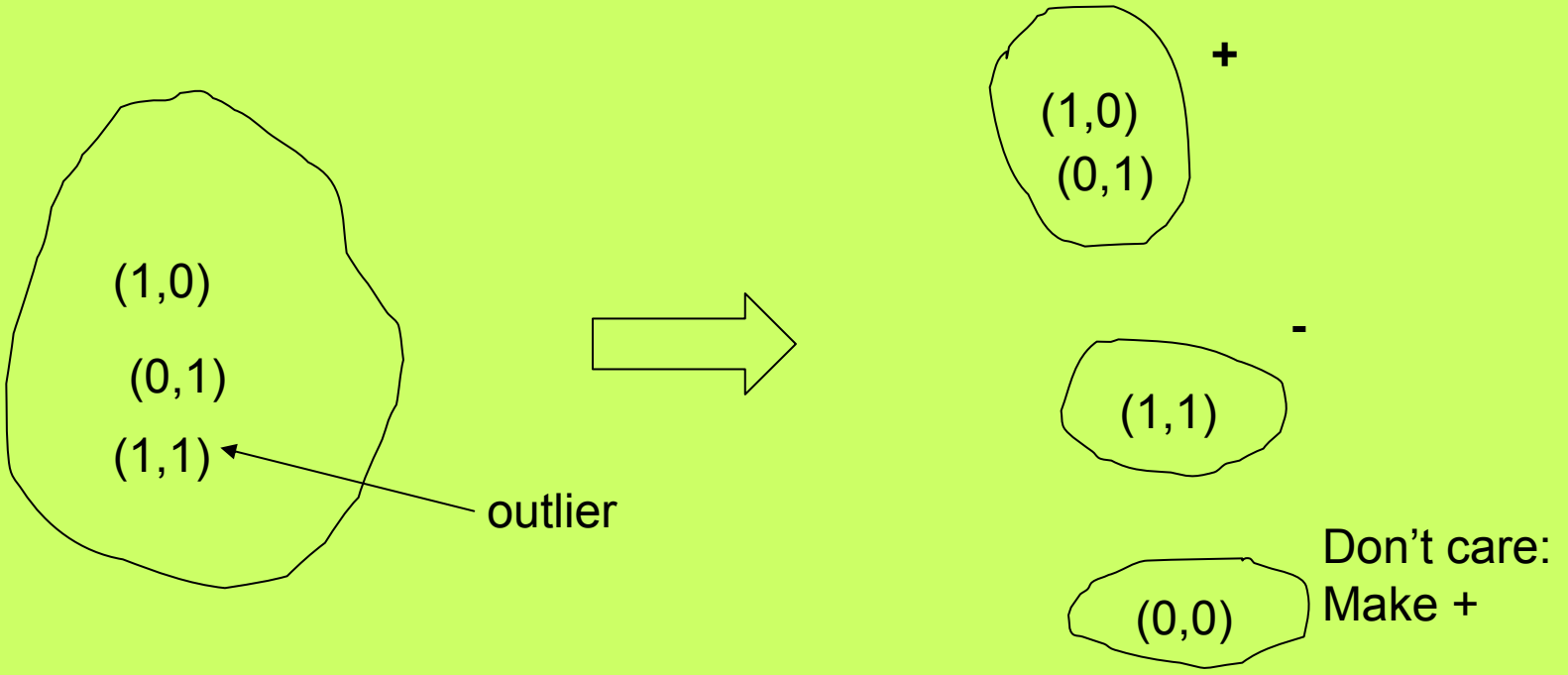


The partially developed n/w

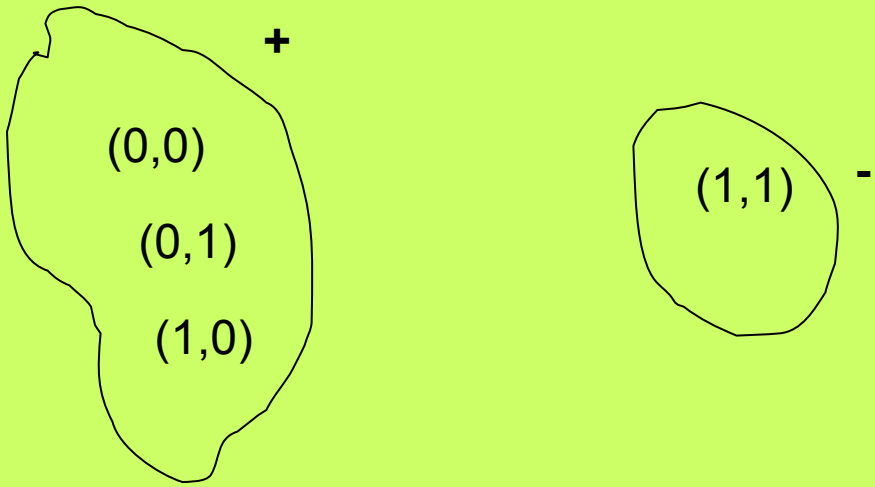
- Get the first neuron in the hidden layer, which computes OR



Break the incorrect class

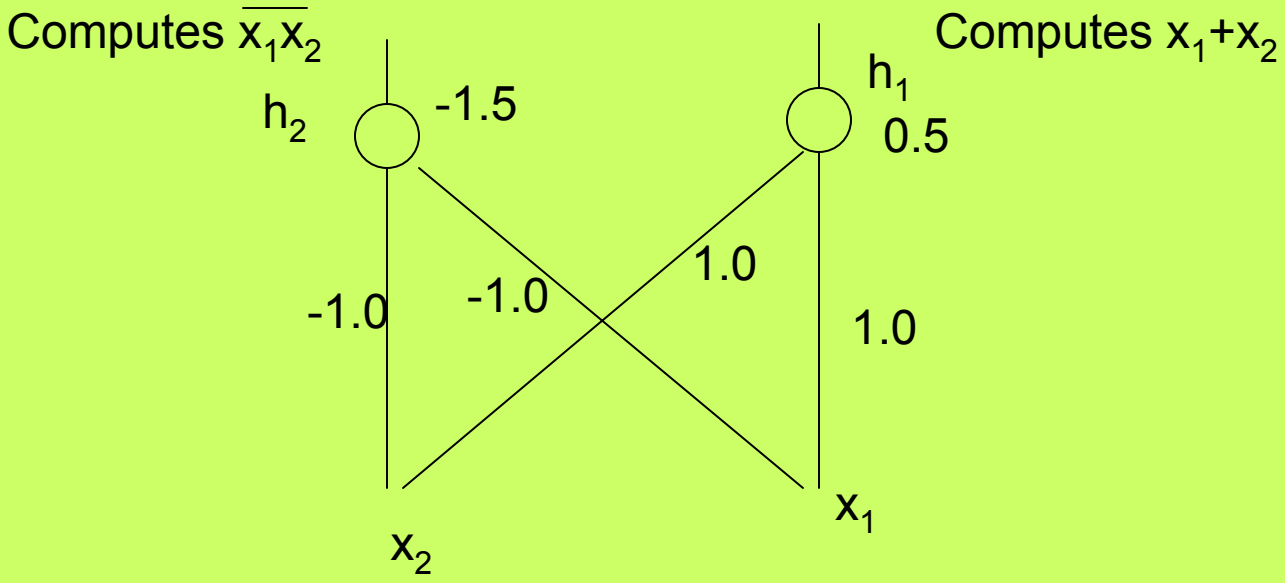


Solve classification for h_2



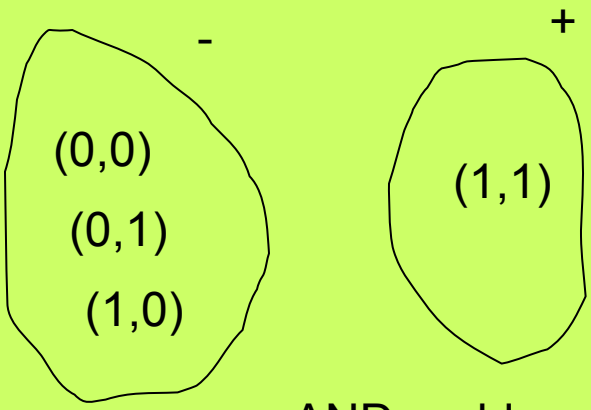
This is $\overline{x_1 x_2}$

Next stage of the n/w



Getting the output layer

- Solve a tiling algo problem for the hidden layer

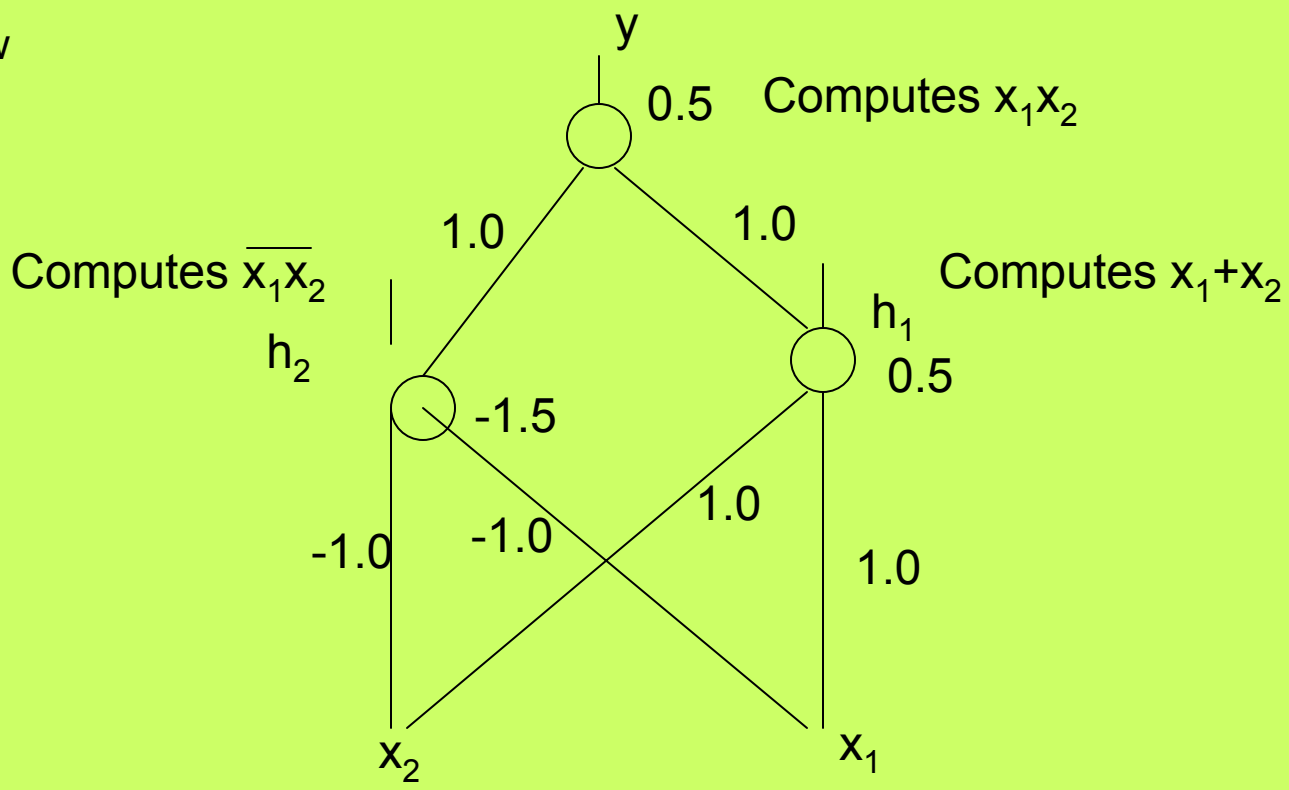


AND problem

x_2	x_1	h_1 (x_1+x_2)	h_1 $\overline{x_1 x_2}$	y
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Final n/w

- AND n/w



Lab exercise

Implement the tiling algorithm and run it for

1. XOR
2. Majority
3. IRIS data