NN Basics Tutorial 2

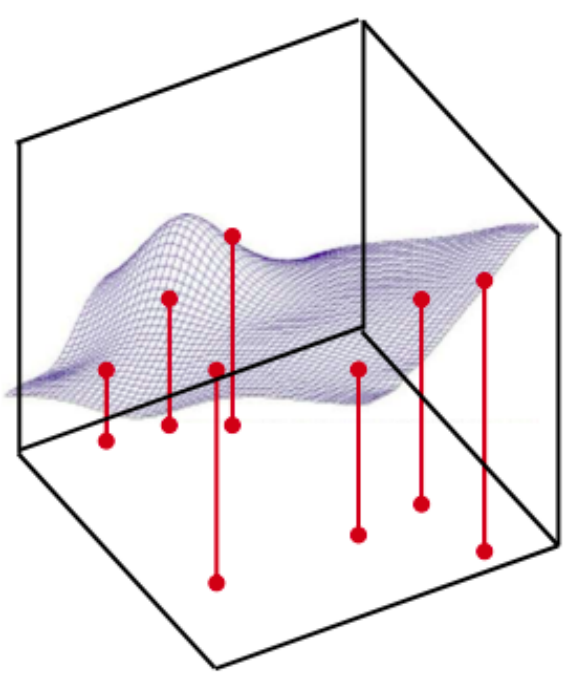
Tirtharaj Dash

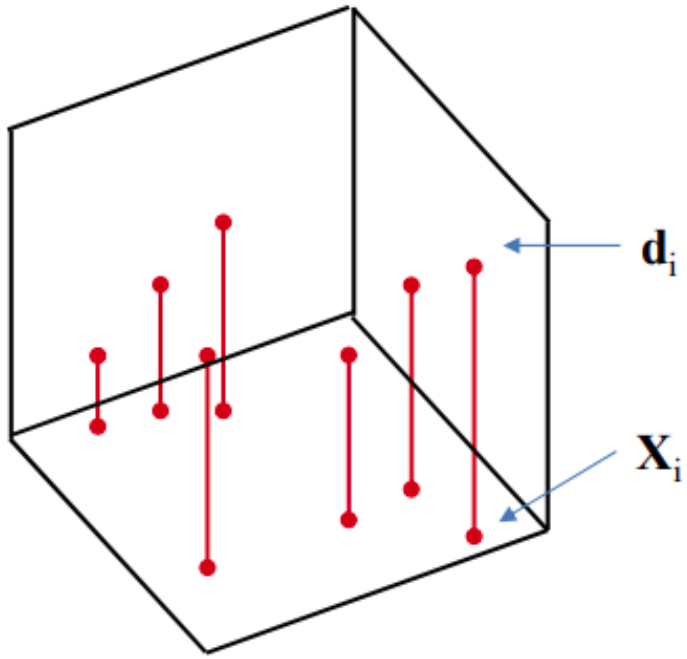
Dept. of CS & IS and APPCAIR
BITS Pilani, Goa Campus

August 28, 2021

What is our intention?

- We have a dataset of few samples of data points (or instances), and the goal is to learn the entire function from these few samples.
- What do we really mean by the above:





- Given these $\mathbf{x}_i$'s and and d_i's (d_is are also denoted as y_i's, called true outputs or desired outputs), we try to approximate the true (blue) function $g(\mathbf{x})$ using a neural network $f(\mathbf{X}, \mathbf{w})$.

Perceptron I

A simple modification for convenience: During an initial lecture we motivated the Perceptron network via Logistic Regression, where we showed the use of the sigmoid activation function for Binary classification. However, here we will use the standard Perceptron model based on a hard-threshold.

What is the difference? The only difference is that in the former, the percetron makes a 'soft' decision (refers to the probability estimates), and in the latter case, the perceptron makes a hard decision (refers to strict condition check against a threshold).

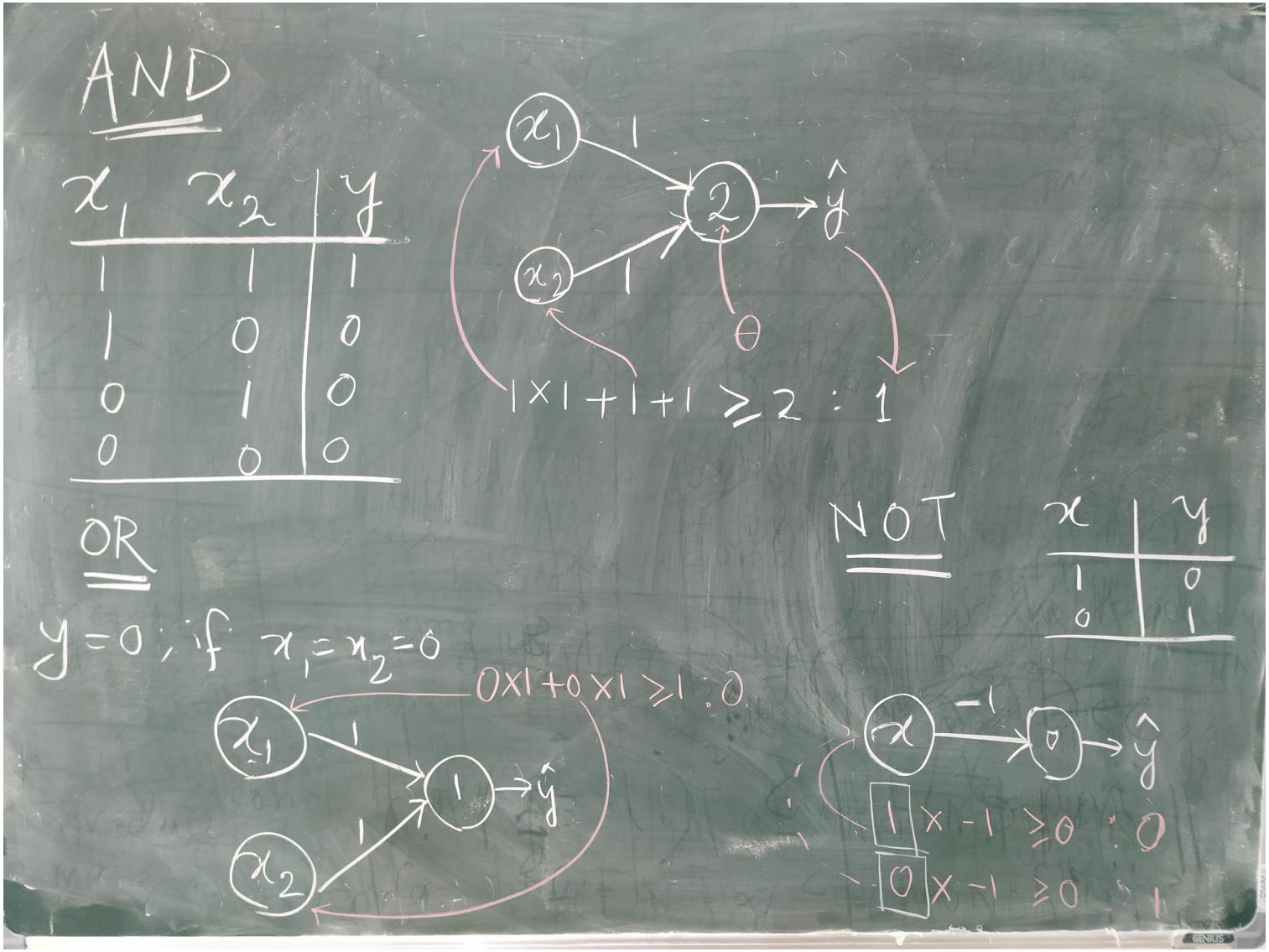
Perceptron II

1. For modelling a Perceptron, assume a hard-threshold function for activation at the output neuron i.e.

$$\hat{y} = \begin{cases} 1 & \text{if } \sum \mathbf{w} \cdot \mathbf{x} \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

Can it model simple Boolean gate operations (AND, OR, NOT)?

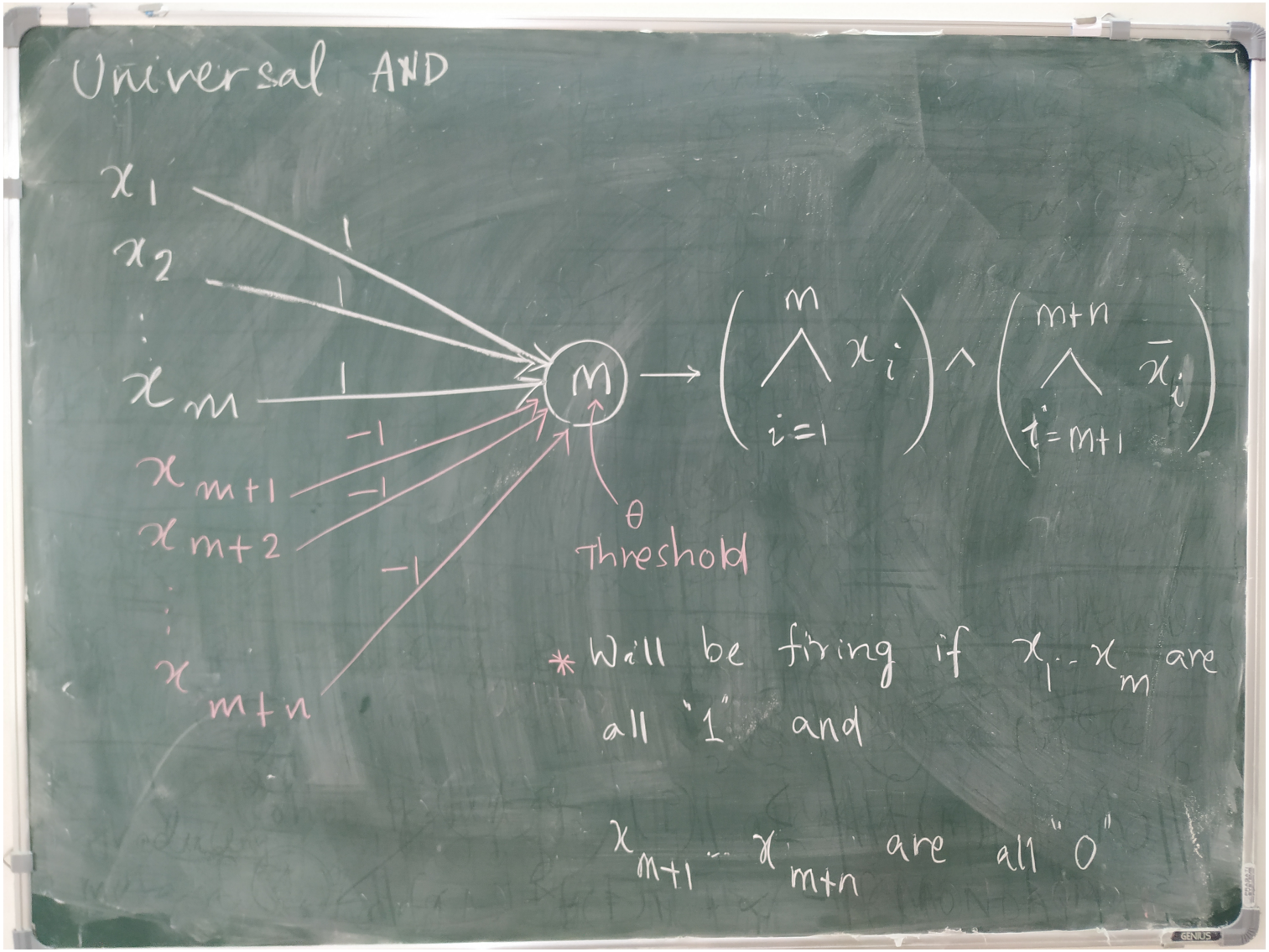
Perceptron III



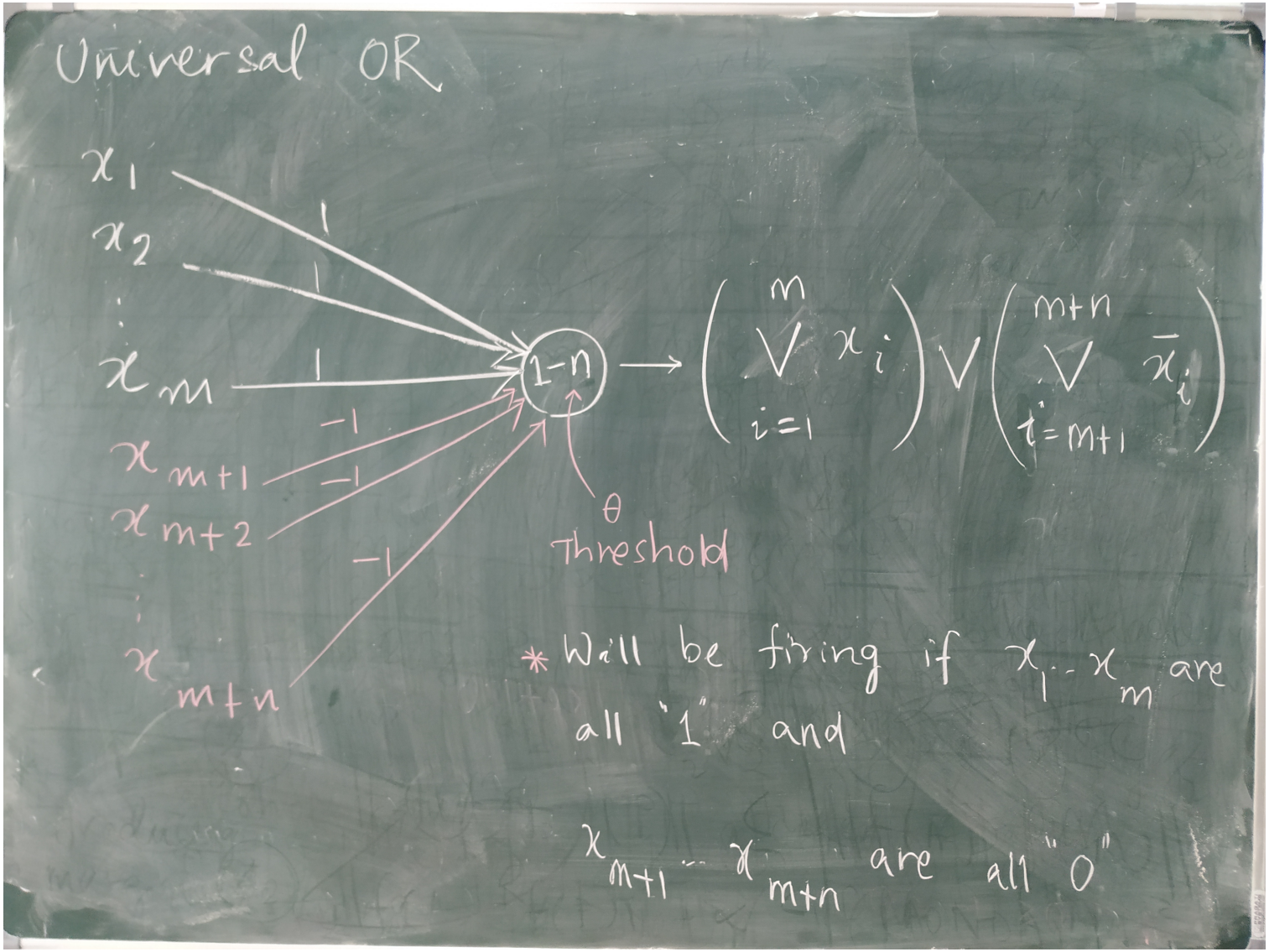
Perceptron IV

2. Similar to the previous question, model the following gate operations: (a) Universal AND; (b) Universal OR.

Perceptron V



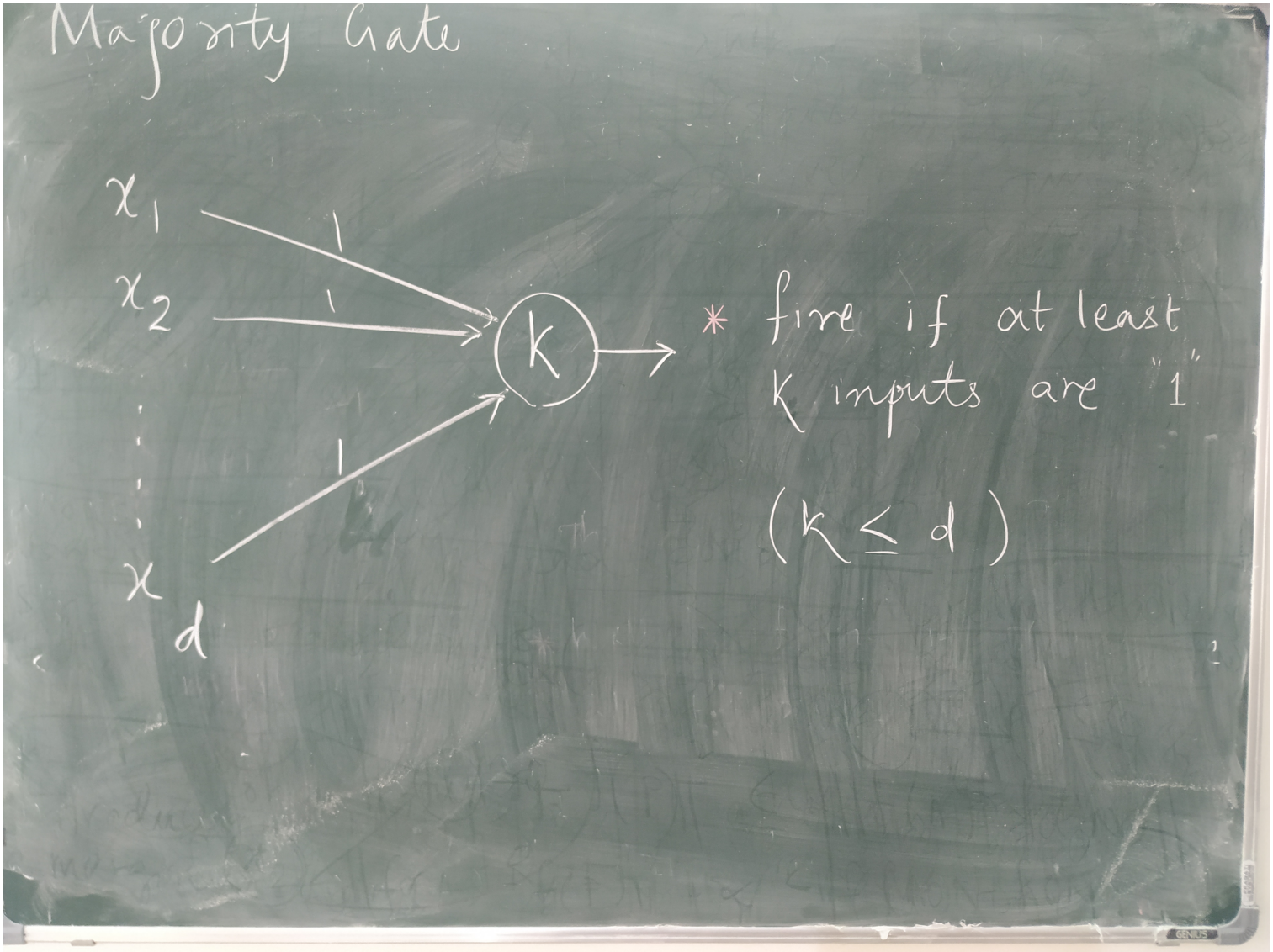
Perceptron VI



Perceptron VII

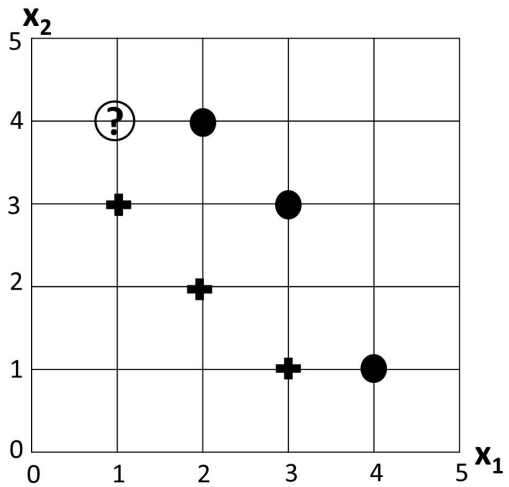
3. Model a majority gate operation using Perceptron. The model should fire if at least k (out of d) inputs are active.

Perceptron VIII



Perceptron IX

4. The following data points represented as $(x_1,\ x_2)$ are provided to you. Data points are: Positive: (1, 3) (2, 2) (3, 1) and Negative: (2, 4) (3, 3) (4, 1). Data points are classified as either $+1$ or -1. An unknown point is located at (1, 4). The goal is to learn a hard-threshold perceptron. That is: $\hat{y} = sign(\mathbf{w}^T\mathbf{x})$, where $sign(\cdot)$ is the sign function.
   Answer the following questions.



Perceptron X

(a) Assume that the points are introduced to perceptron in the following order. Simulate one iteration of the perceptron algorithm with a learning rate (η) of 0.5 and an initial weight vector of $(w_1, w_2, w_0) = (3, 3, 30)$. Fill the following table for your weight vector in the same format.

Data point (x_1, x_2)	w_1	w_2	w_0 (bias)
	3	3	30
(1, 3), +1	?	?	?
(2, 2), +1	?	?	?
(3, 1), +1	?	?	?
(2, 4), –1	?	?	?
(3, 3), –1	?	?	?
(4, 1), –1	?	?	?

(b) What is the equation of your perceptron obtained in part (a)?

(c) Is your perceptron a correct linear separator of the given data?

(d) What is the class label returned by your learned perceptron for the unknown data point (1, 4)? What is the distance of this point from the learned perceptron?

Perceptron XI

Answer. (a) It is a strict-threshold perceptron. The loss function is:

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

where, $\hat{y} = sign(\mathbf{w}\mathsf{T}\mathbf{x}) = sign(w_1 x_1 + w_2 x_2 + w_0)$.

The update equation is (based on the smooth approximation to the gradient of the loss; as explained in the tutorial):

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \times \frac{\partial L}{\partial \mathbf{w}} \\ &\leftarrow \mathbf{w} + \eta \times (y - \hat{y}) \cdot [x_1, x_2, +1] \end{aligned}$$

Perceptron XII

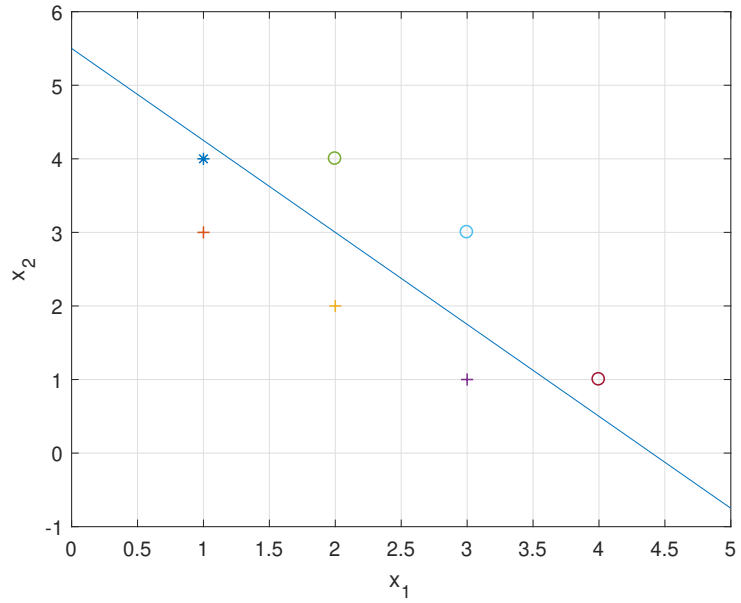
The weight vector after introduction of each example is:

w_1	w_2	w_0
3	3	30
3	3	30
3	3	30
3	3	30
1	–1	29
–2	–4	28
–6	–5	27

- Equation of perceptron is $w_1 x_1 + w_2 x_2 + w_0 = 0$ (Put the values of these parameters)

Perceptron XIII

(c) Yes. See the plot below:

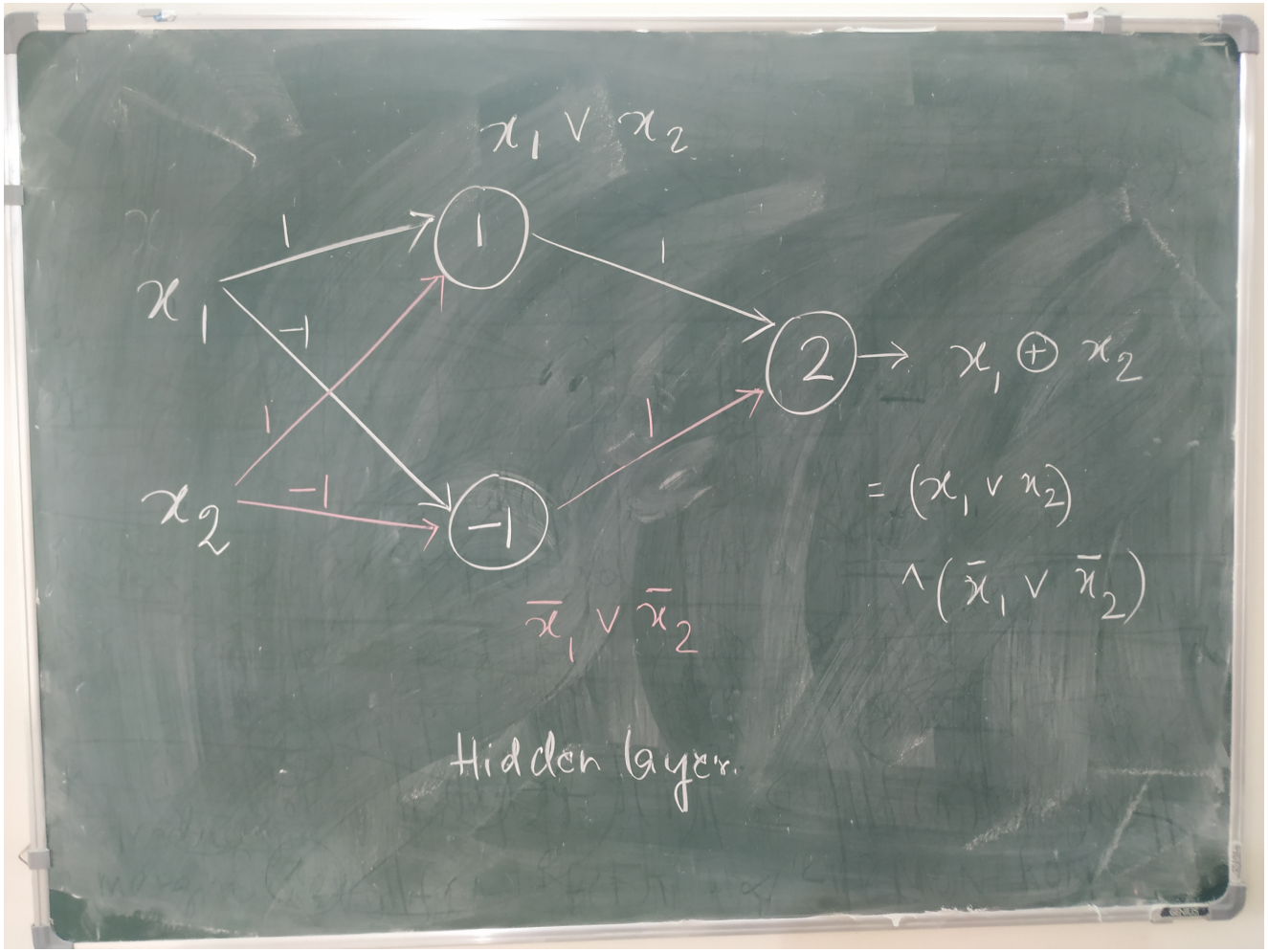


(d) $\hat{y} = +1$. This is correct from the visuals. Thd distance is $d(perceptron, (1, 4)) = \frac{|w_1 \times 1 + w_2 \times 4 + w_0|}{\sqrt{(-6)^2 + (-5)^2}} = 0.1280$.

MLPs I

5. Model XOR gate using a 2-layered MLP network with hard-threshold.

MLPs II

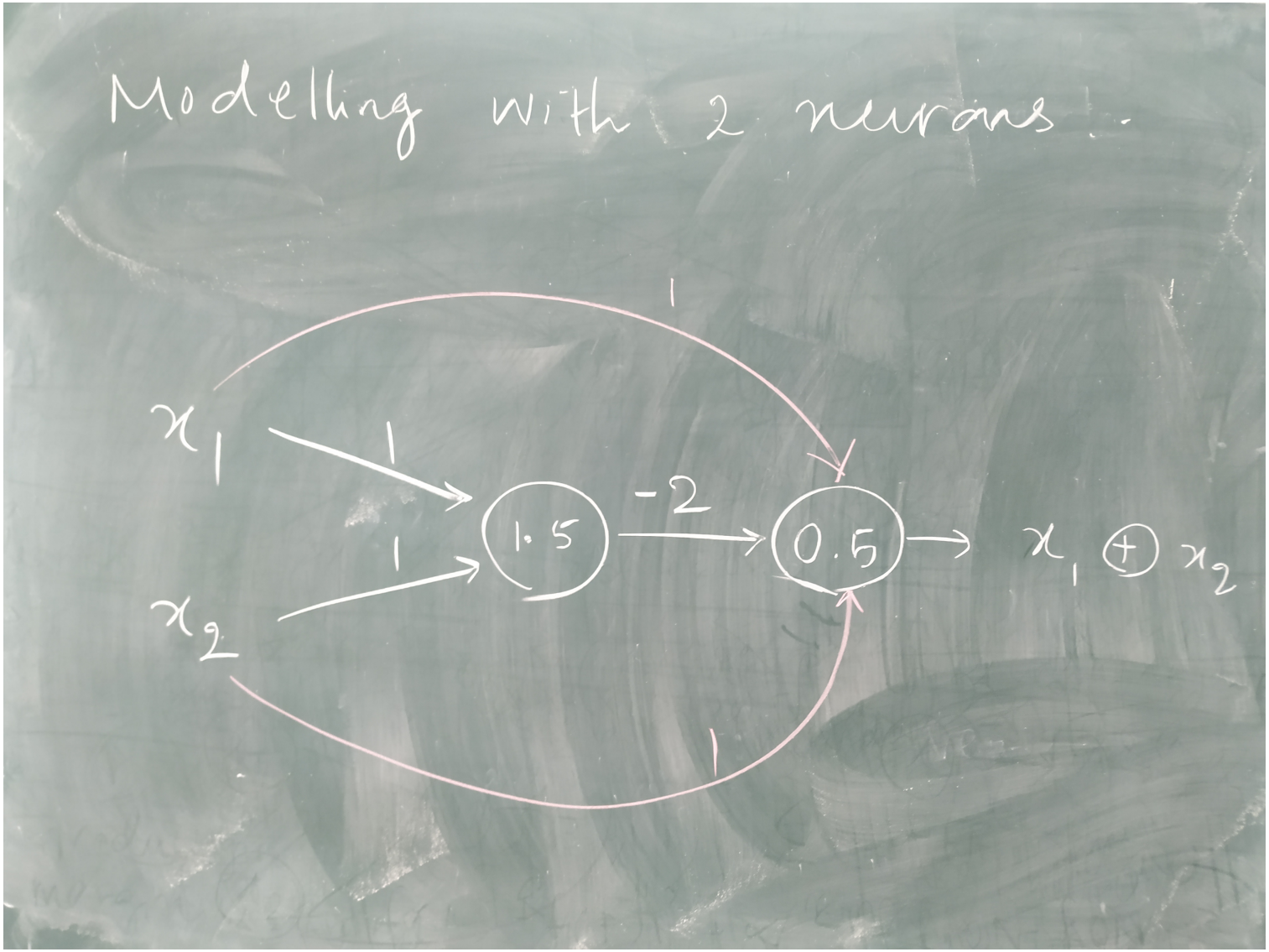


Correction This is modelling XNOR.

MLPs III

6. Can XOR gate be done with lesser (< 3) number of neurons?

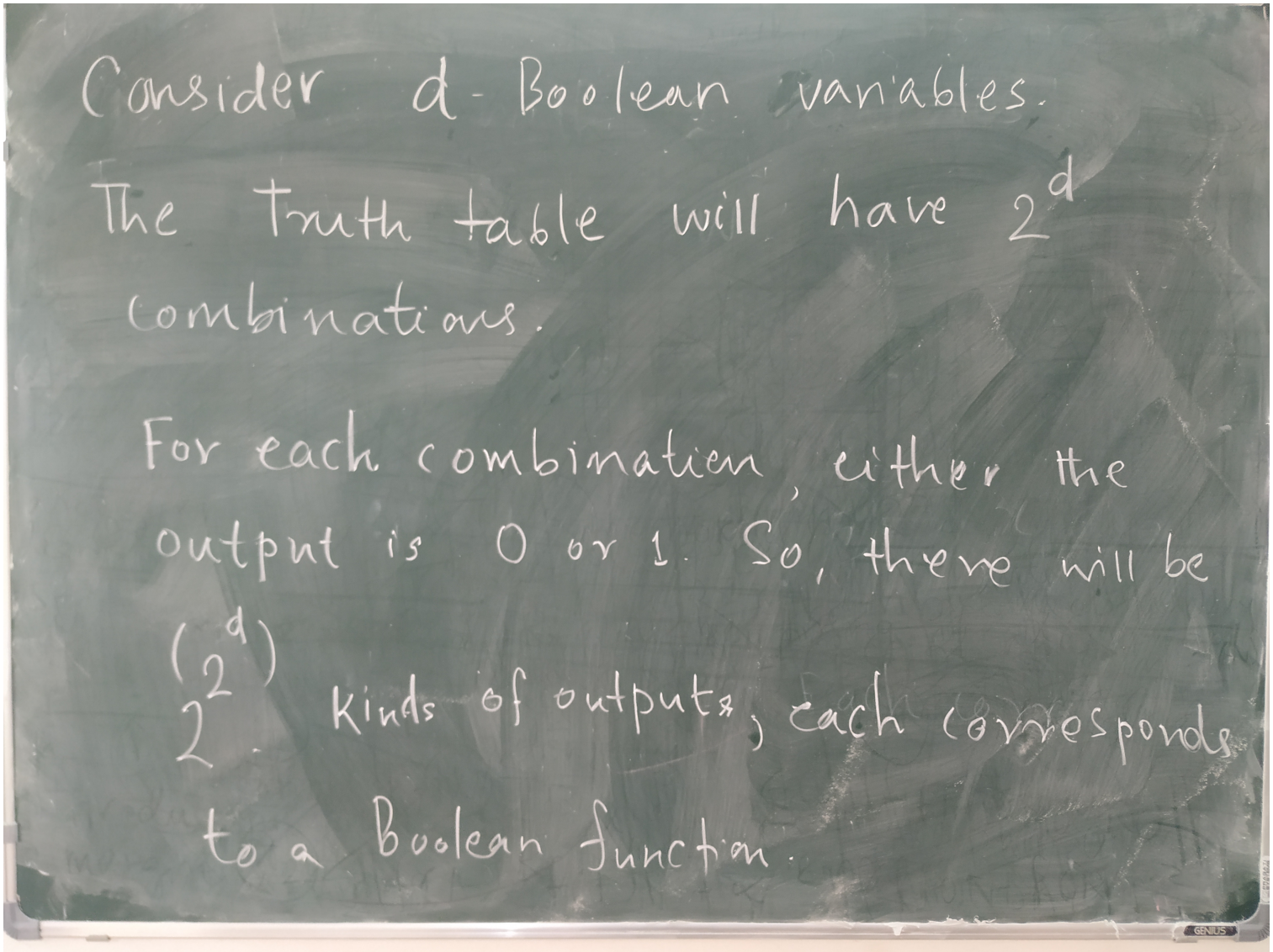
MLPs IV



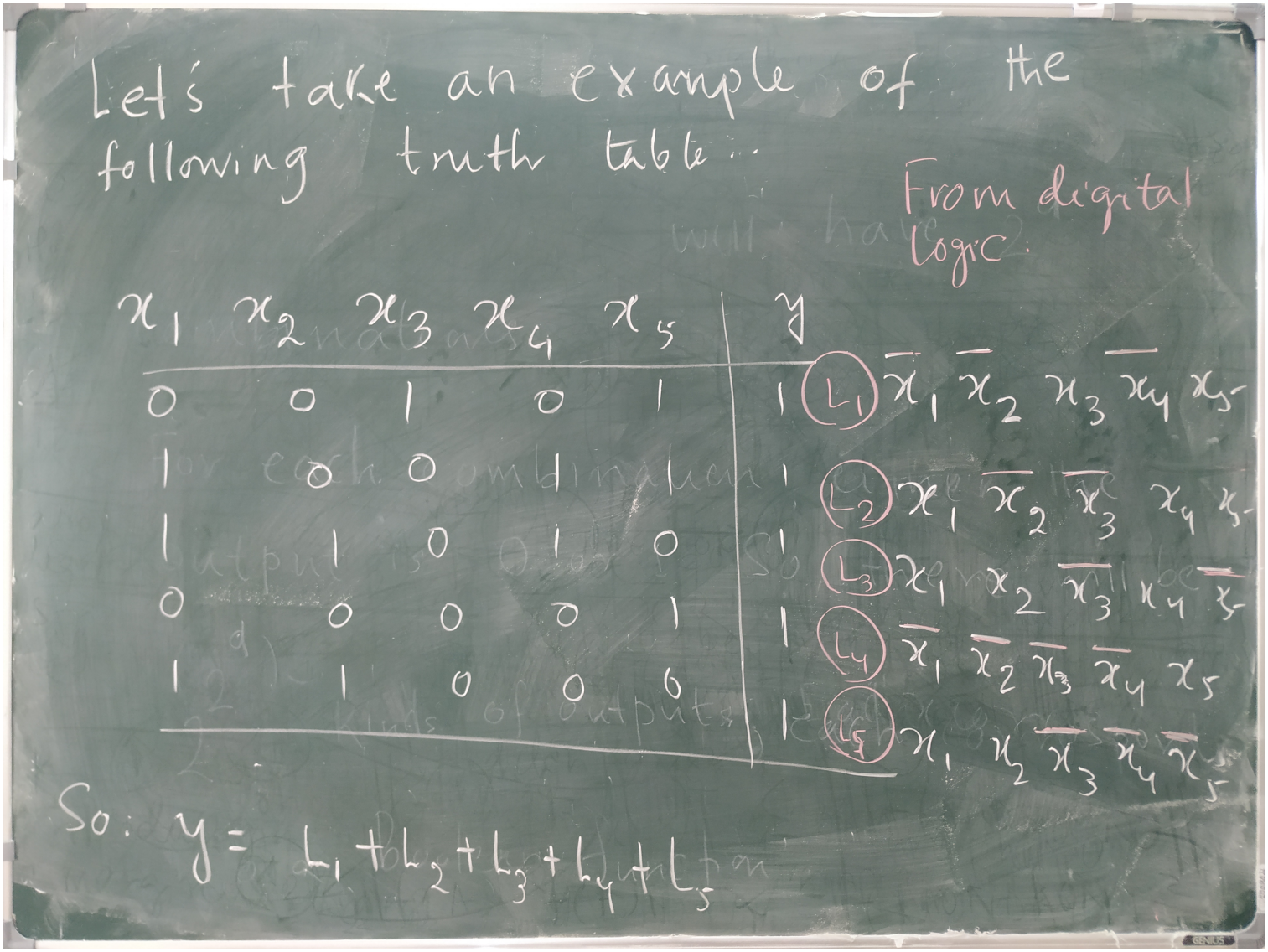
MLPs V

7. A one-hidden-layer MLP is a Universal Boolean function. Verify this with an example.

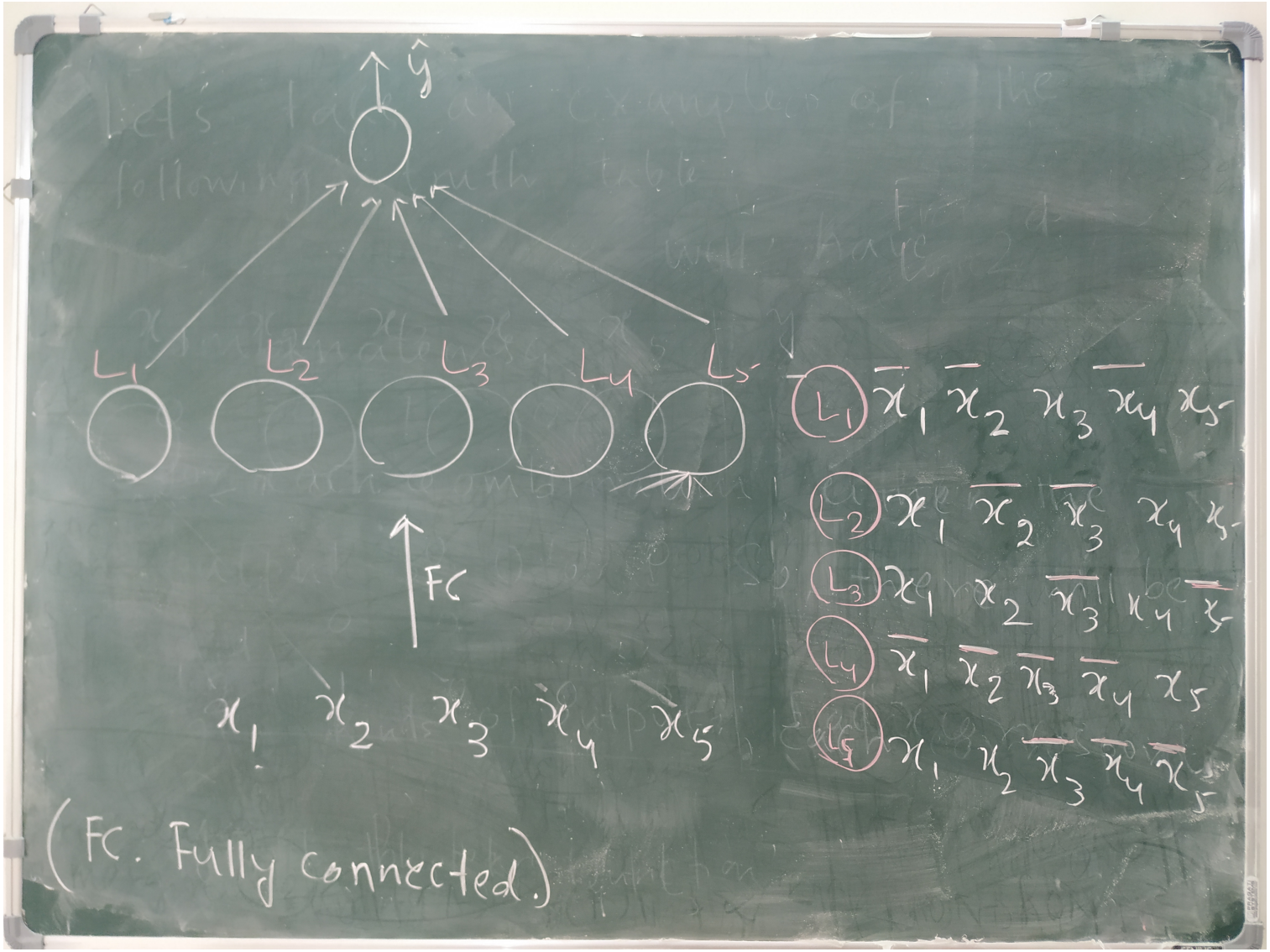
MLPs VI


Consider d - Boolean variables.
The truth table will have 2^d combinations.
For each combination, either the output is 0 or 1. So, there will be $2^{(2^d)}$ kinds of outputs, each corresponds to a Boolean function.

MLPs VII


Let's take an example of the following truth table..
From digital logic.
x_1 x_2 x_3 x_4 x_5 | y
0 0 1 0 1 | 1
1 0 0 1 1 | 1
1 1 0 1 0 | 1
0 0 0 0 1 | 1
1 1 0 0 0 | 1
L_1 $\bar{x}_1 \bar{x}_2 x_3 \bar{x}_4 x_5$
L_2 $x_1 \bar{x}_2 \bar{x}_3 x_4 x_5$
L_3 $x_1 x_2 \bar{x}_3 \bar{x}_4 \bar{x}_5$
L_4 $\bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 x_5$
L_5 $x_1 x_2 \bar{x}_3 \bar{x}_4 \bar{x}_5$
So: $y = L_1 + L_2 + L_3 + L_4 + L_5$

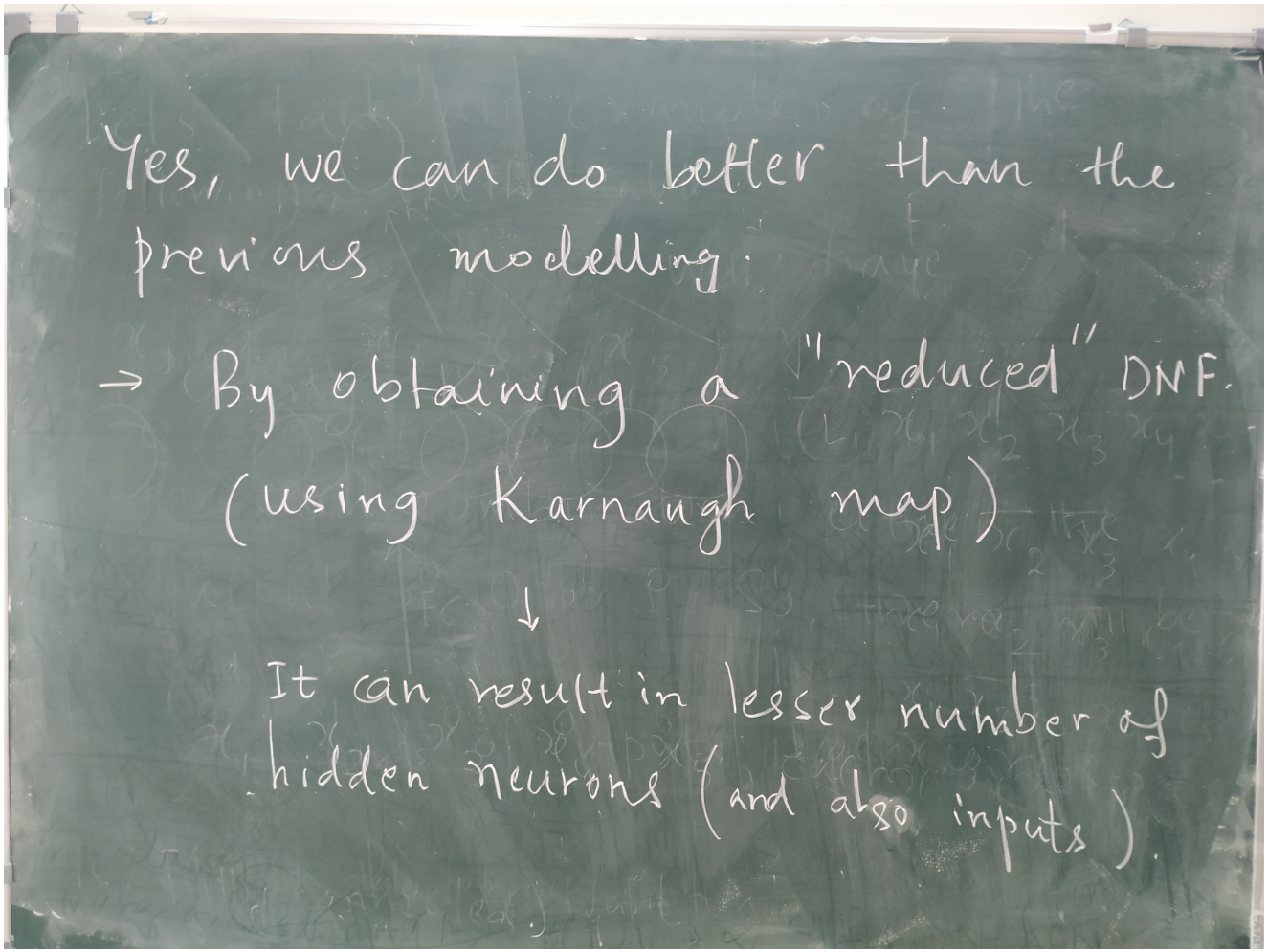
MLPs VIII


ŷ
L1
L2
L3
L4
L5
FC
x1 x2 x3 x4 x5
(FC. Fully connected.)
L1 x̄1 x̄2 x3 x̄4 x5
L2 x1 x̄2 x̄3 x4 x5
L3 x1 x2 x̄3 x4 x̄5
L4 x̄1 x̄2 x̄3 x̄4 x5
L5 x1 x2 x̄3 x̄4 x̄5

MLPs IX

8. For your given example, can you do better than your previous modelling?

MLPs X


Yes, we can do better than the previous modelling.
→ By obtaining a "reduced" DNF.
(using Karnaugh map)
↓
It can result in lesser number of hidden neurons (and also inputs).

MLPs XI

9. Consider a softmax activation function at the output layer (with k units) of a neural network. Here, the real-valued outputs are converted into probabilities as follows:

$$\hat{y}_i = \frac{\exp(v_i)}{\sum_{i=1}^{k} \exp(v_i)}; \ \forall \ i = \{1, 2, \ldots, k\}$$

   a. Show that the value of $\frac{\partial \hat{y}_i}{\partial v_j}$ is:
      - $\hat{y}_i(1 - \hat{y}_i)$, when $i = j$.
      - $-\hat{y}_i \hat{y}_j$, when $i \neq j$.

(b) Use the above result to show the correctness of the loss derivative:

$$\frac{\partial L}{\partial v_i} = \hat{y}_i - y_i$$

Assume that we are using the cross-entropy loss $L = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$, where $y_i \in \{0, 1\}$ is the one-hot encoded class label over different values of $i \in \{1, 2, \ldots, k\}$. In one-hot encoding, class 1 is represented as $(1, 0, 0, \ldots, k-1$ such $0s)$, and class 2 is represented as $(0, 1, 0, \ldots, k-2$ such $0s)$.

Answer. (a) When $i = j$:

$$\frac{\partial \hat{y}_i}{\partial v_j} = \frac{\left(\sum_{i=1}^{k} \exp(v_i)\right) \exp(v_j) - \exp(v_j)\exp(v_j)}{\left(\sum_{i=1}^{k} \exp(v_i)\right)^2}$$

$$= \frac{\exp(v_i)}{\sum_{i=1}^{k} \exp(v_i)} - \frac{\exp(v_j)^2}{\left(\sum_{i=1}^{k} \exp(v_i)\right)^2}$$

$$= \hat{y}_i - (\hat{y}_i)^2$$

$$= \hat{y}_i(1 - \hat{y}_i)$$

When $i \neq j$:

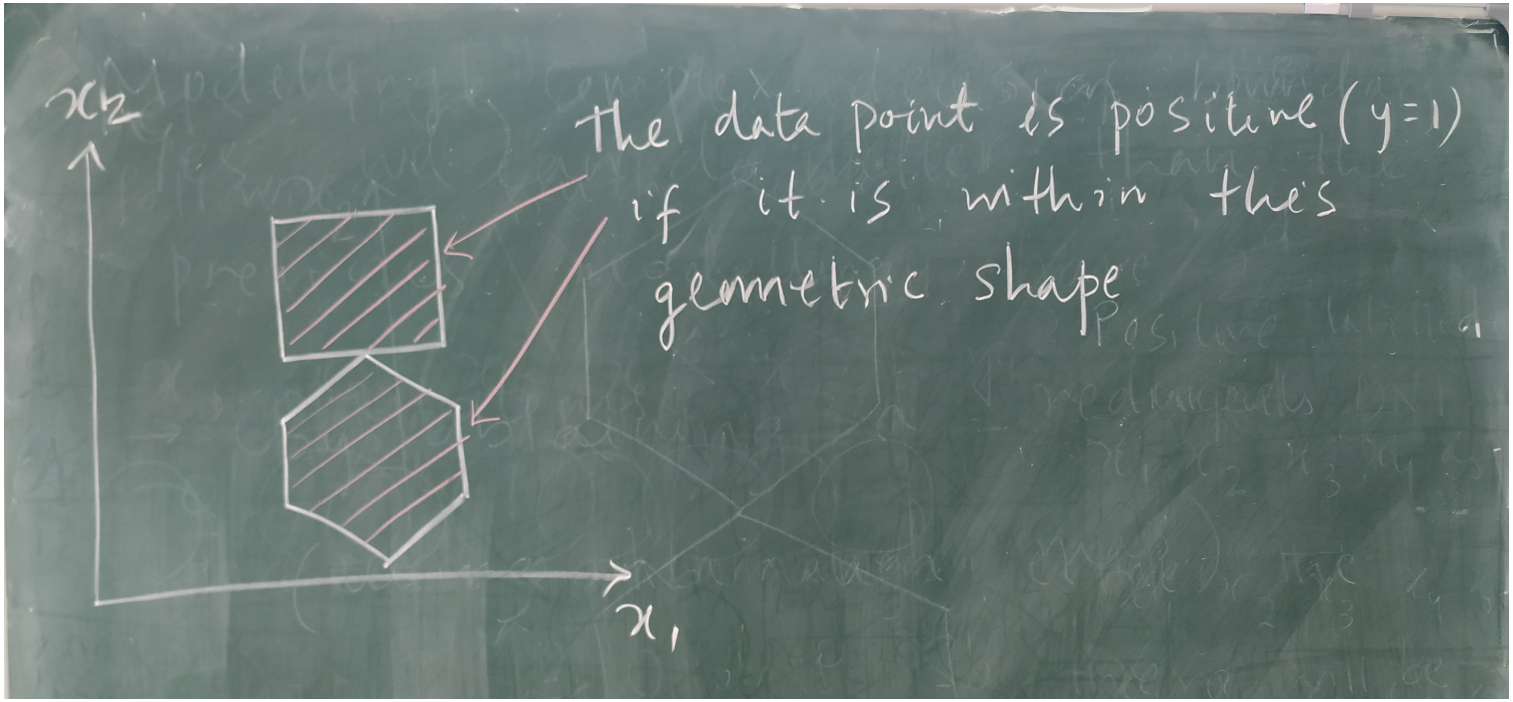
$$\begin{aligned}
\frac{\partial \hat{y}_i}{\partial v_j} &= \frac{0 - \exp(v_i)\exp(v_j)}{\left(\sum_{i=1}^{k} \exp(v_i)\right)^2} \\
&= -\frac{\exp(v_i)}{\sum_{i=1}^{k} \exp(v_i)} \frac{\exp(v_j)}{\sum_{i=1}^{k} \exp(v_i)} \\
&= -\hat{y}_i \hat{y}_j
\end{aligned}$$

(b) $\frac{\partial L}{\partial v_i}$ can be expressed as

$$\begin{aligned}
\frac{\partial L}{\partial v_i} &= \sum_{j=1}^{k} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial v_i} \\
&= \sum_{j \neq i} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial v_i} + \sum_{j=i} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial v_i} \\
&= \sum_{j \neq i} \frac{y_i}{\hat{y}_j}(-\hat{y}_i \hat{y}_j) - \frac{y_i}{\hat{y}_j} \hat{y}_i (1 - \hat{y}_i) \\
&= -y_i + \hat{y}_i \hat{y}_i + \sum_{j \neq i} \hat{y}_i \hat{y}_j \\
&= -y_i + \sum_{j=1}^{k} \hat{y}_i \hat{y}_j \\
&= -y_i + \hat{y}_i \sum_{j=1}^{k} \hat{y}_j
\end{aligned}$$

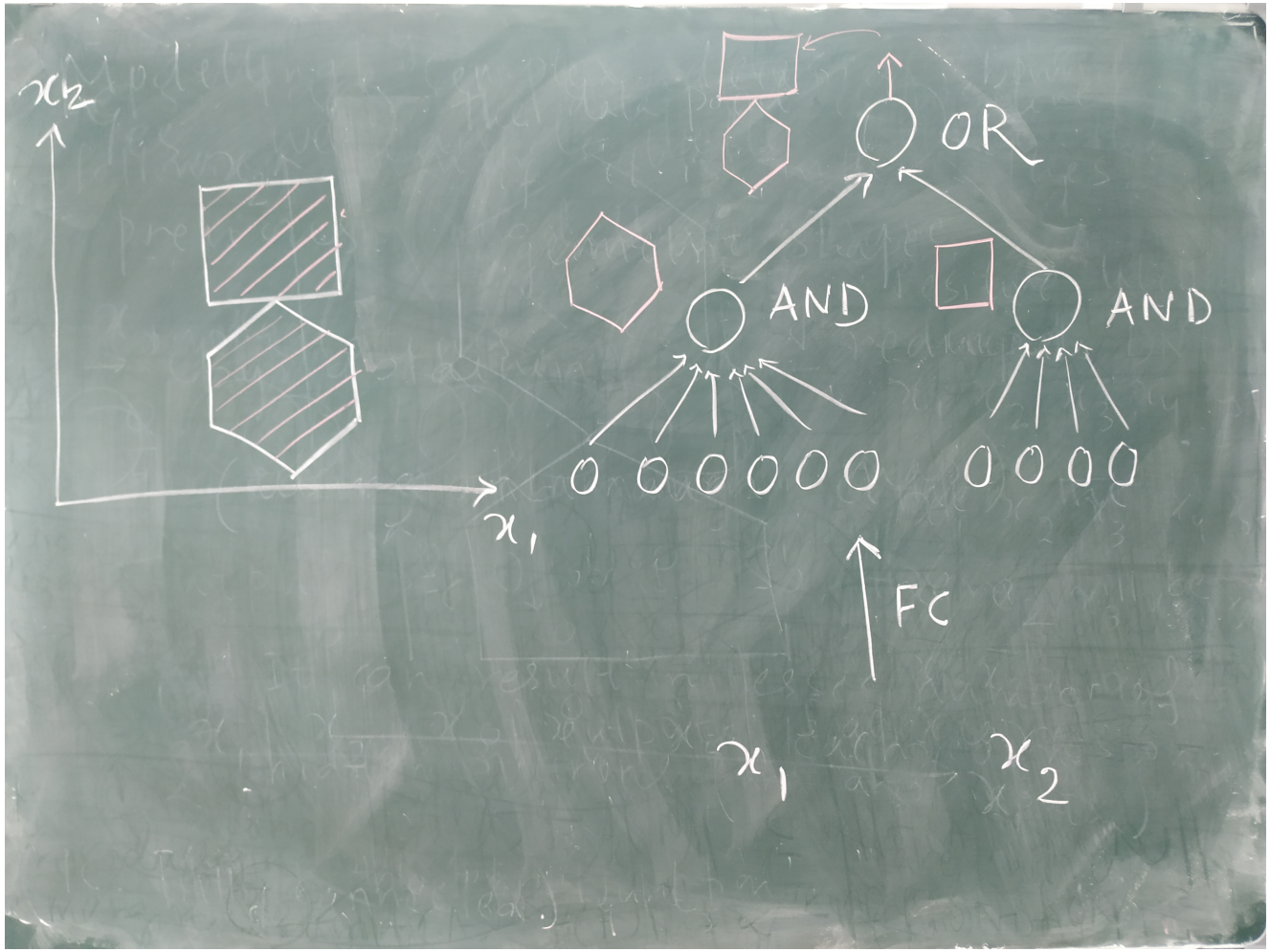
MLP Decision Boundary I

10. Model the following decision boundary using MLP.

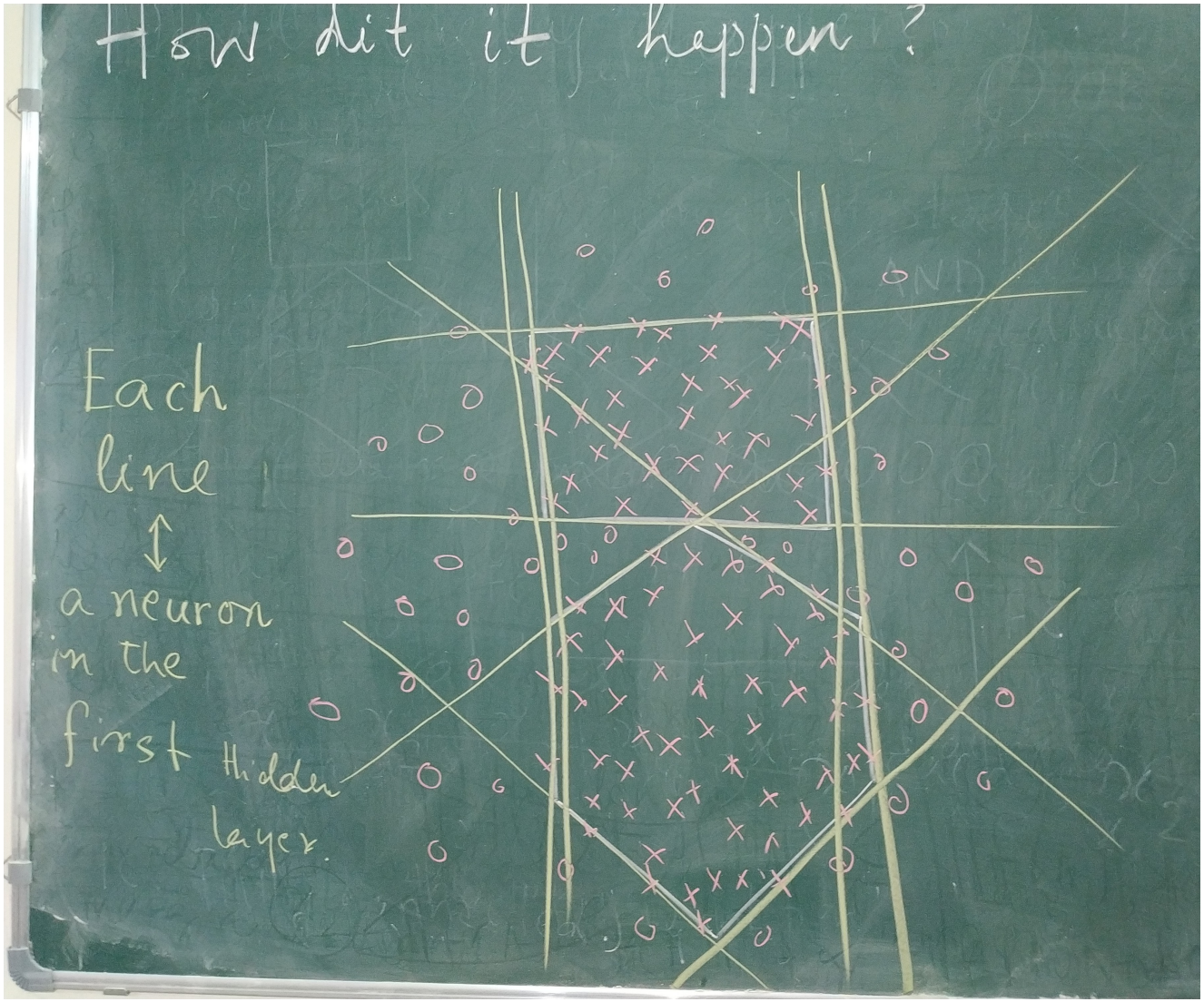


MLP Decision Boundary II

Answer. Here is the rough sketch:



MLP Decision Boundary III



MLP as Universal Approximator I

11. Consider a single-hidden layer MLP with parameters $\mathbf{w}$ and b, and hidden layer activation $g(\cdot)$. Assuming we are free to choose any $\mathbf{w}$, b and g, which functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ can an MLP arbitrarily approximate?

MLP as Universal Approximator II

Answer. Neural networks can emulate any continuous function as long as the activation functions g are not algebraic polynomials almost everywhere. Since a mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ can be computed by m mappings, $f_j : \mathbb{R}^n \to \mathbb{R}$ (w.l.o.g $f : \mathbb{R}^n \to \mathbb{R}$)

A single-hidden layer MLP is then:

$$f_j(\mathbf{x}) = \sum_{i=1}^{k} w_j^{(2)} g\left(\mathbf{w}_j^{(1)} \cdot \mathbf{x} + b\right)$$

MLP as Universal Approximator III

The following are from: Leshno et al. (1993): Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6), 861-867.

- By choosing $\mathbf{w}^{(1)}$ $j = 1, 2, \ldots,$ we can create an infinite set of basis functions for $f(\mathbf{x})$: This set forms a basis for the set of all continuous functions
- The activation g is not simply non-linear as we often find in some online blogs. It is **non-polynomial**.
  - E.g. $g(x) = x^2$ will not give a universal approximator MLP.