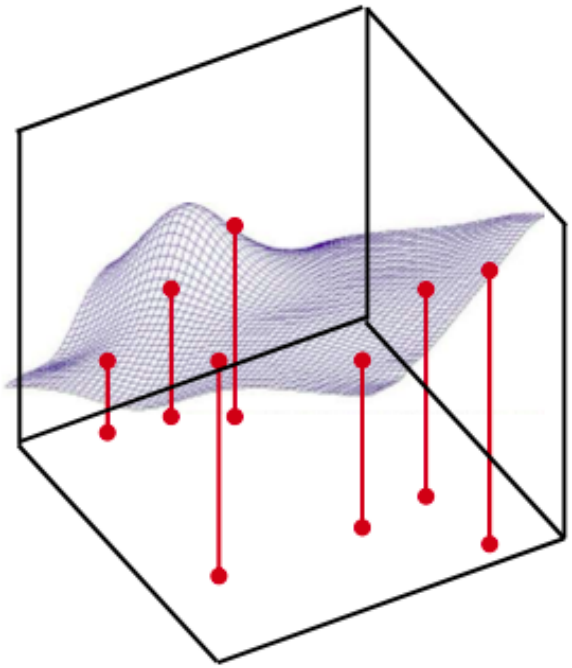
Deep Learning Refresher

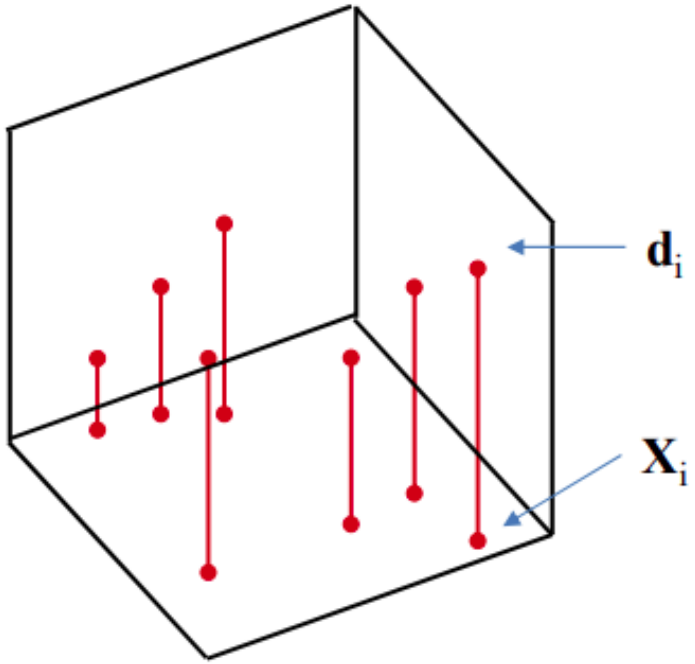
Week 2

Tirtharaj Dash

Tutorial I

- 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:





Tutorial II

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

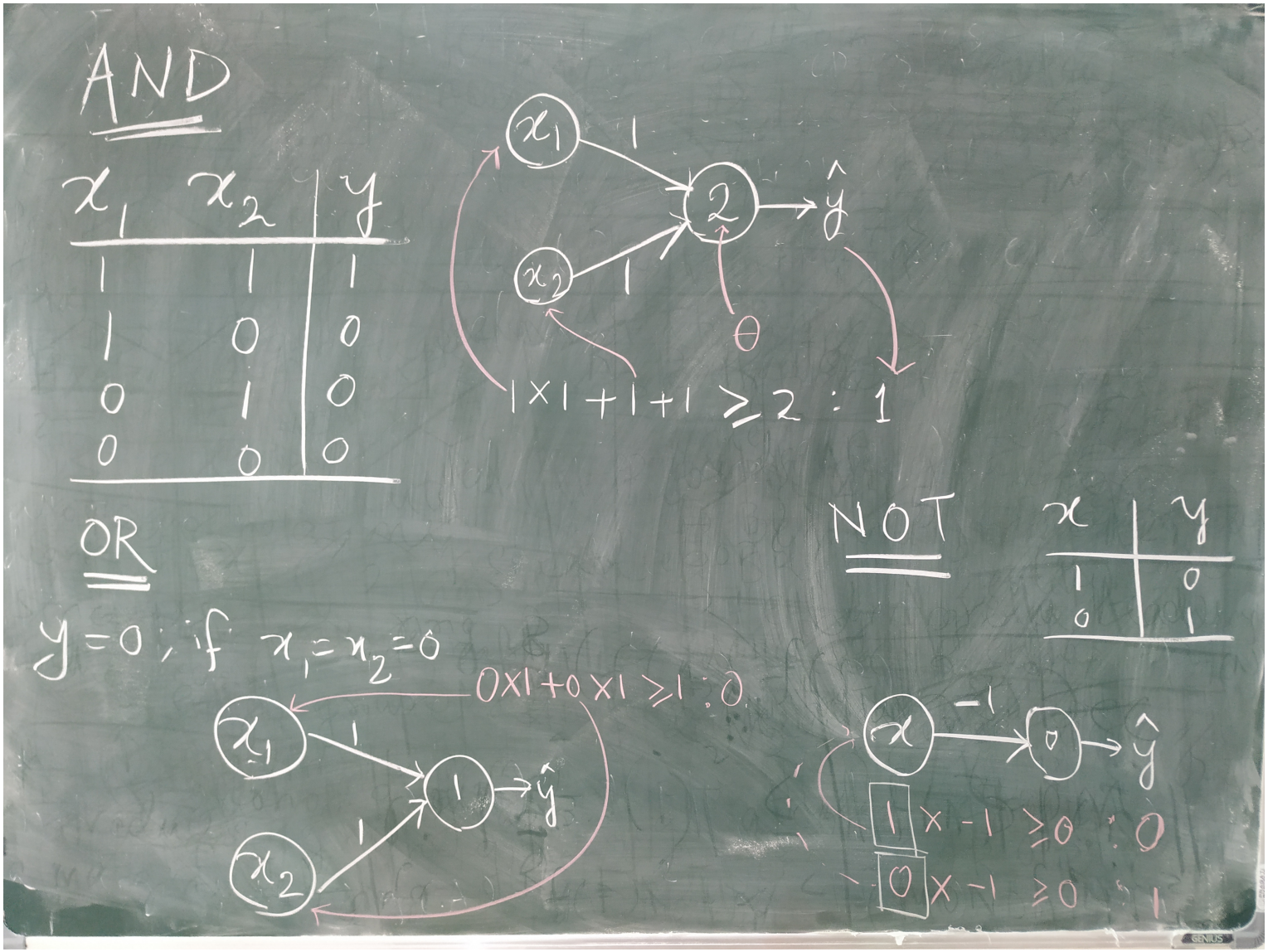
Tutorial III

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)?

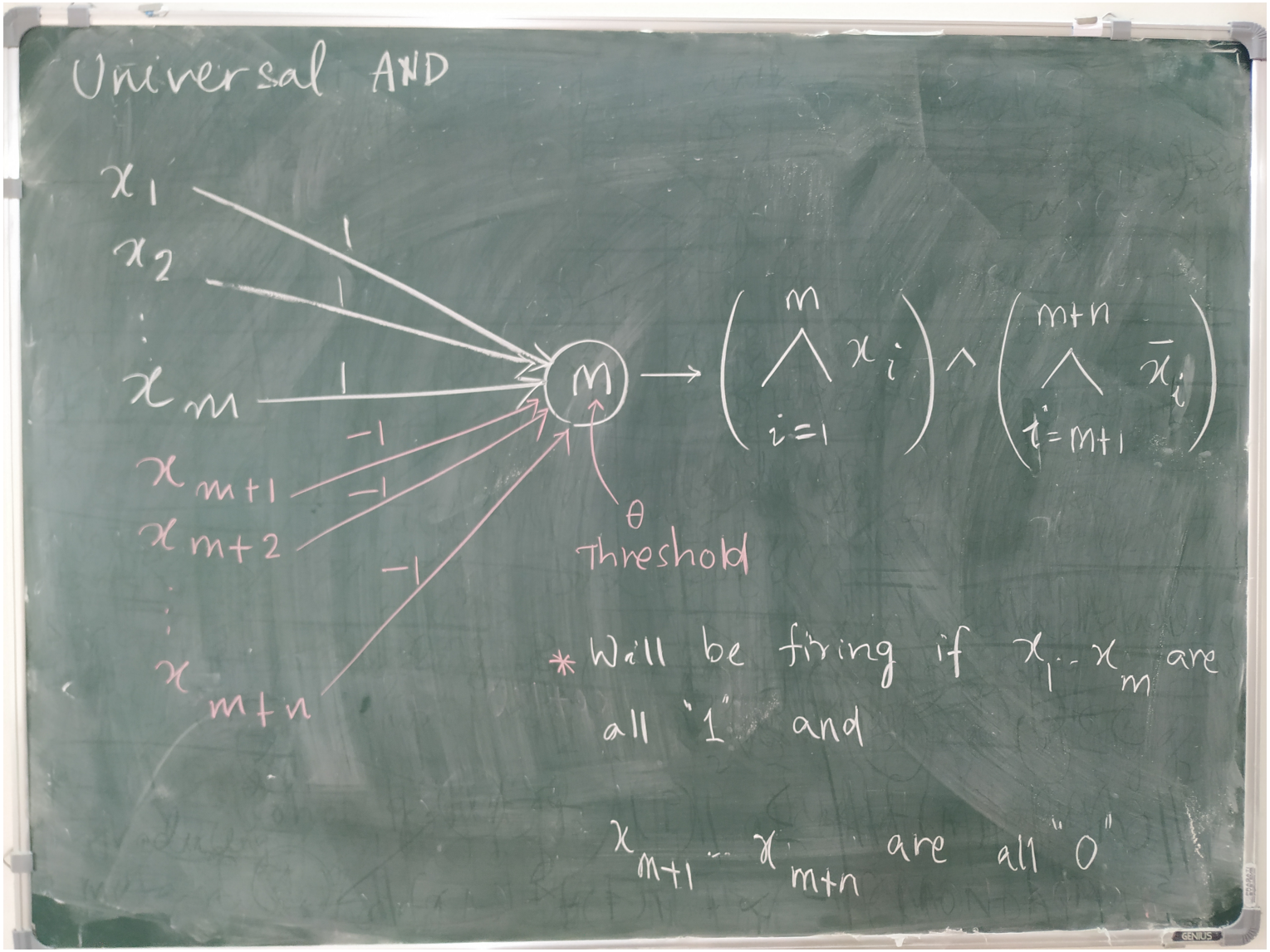
Tutorial IV


AND
x_1 x_2 y
1 1 1
1 0 0
0 1 0
0 0 0
x_1 1
x_2 1
2
$\hat{y}$
θ
$1\times1+1+1 \geq 2 : 1$
OR
$y=0$; if $x_1=x_2=0$
$0\times1+0\times1 \geq 1 : 0$
x_1 1
x_2 1
1
$\hat{y}$
NOT
x y
1 0
0 1
x -1
0
$\hat{y}$
$1\times-1 \geq 0 : 0$
$0\times-1 \geq 0 : 1$

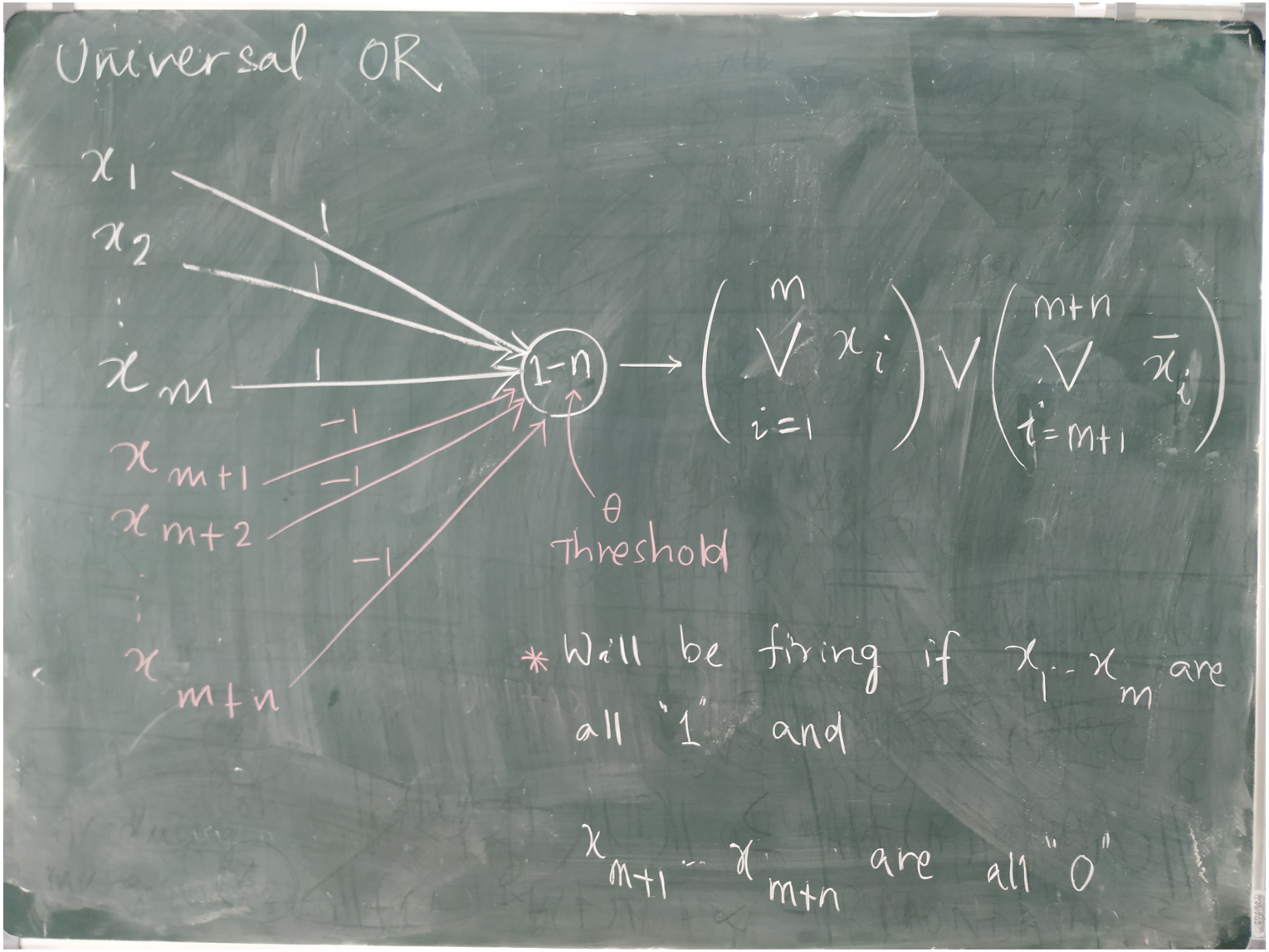
Tutorial V

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

Tutorial VI


Universal AND
x_1
x_2
x_m
x_{m+1}
x_{m+2}
x_{m+n}
1
1
1
−1
−1
−1
m
θ
Threshold
$\left(\bigwedge_{i=1}^{m} x_i\right) \wedge \left(\bigwedge_{i=m+1}^{m+n} \bar{x}_i\right)$
* Will be firing if $x_1 \ldots x_m$ are all "1" and
$x_{m+1} \ldots x_{m+n}$ are all "0"

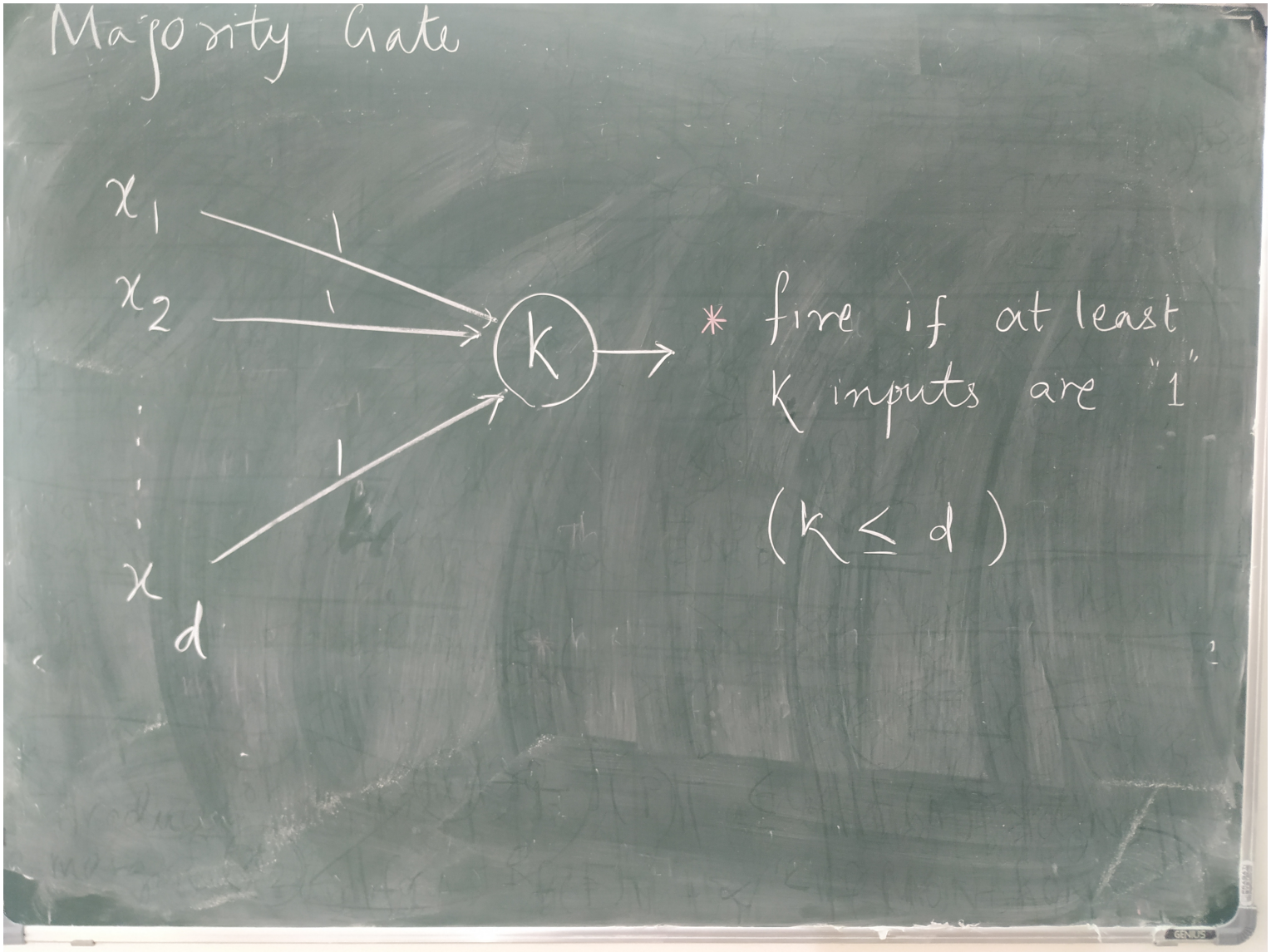
Tutorial VII



Tutorial VIII

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

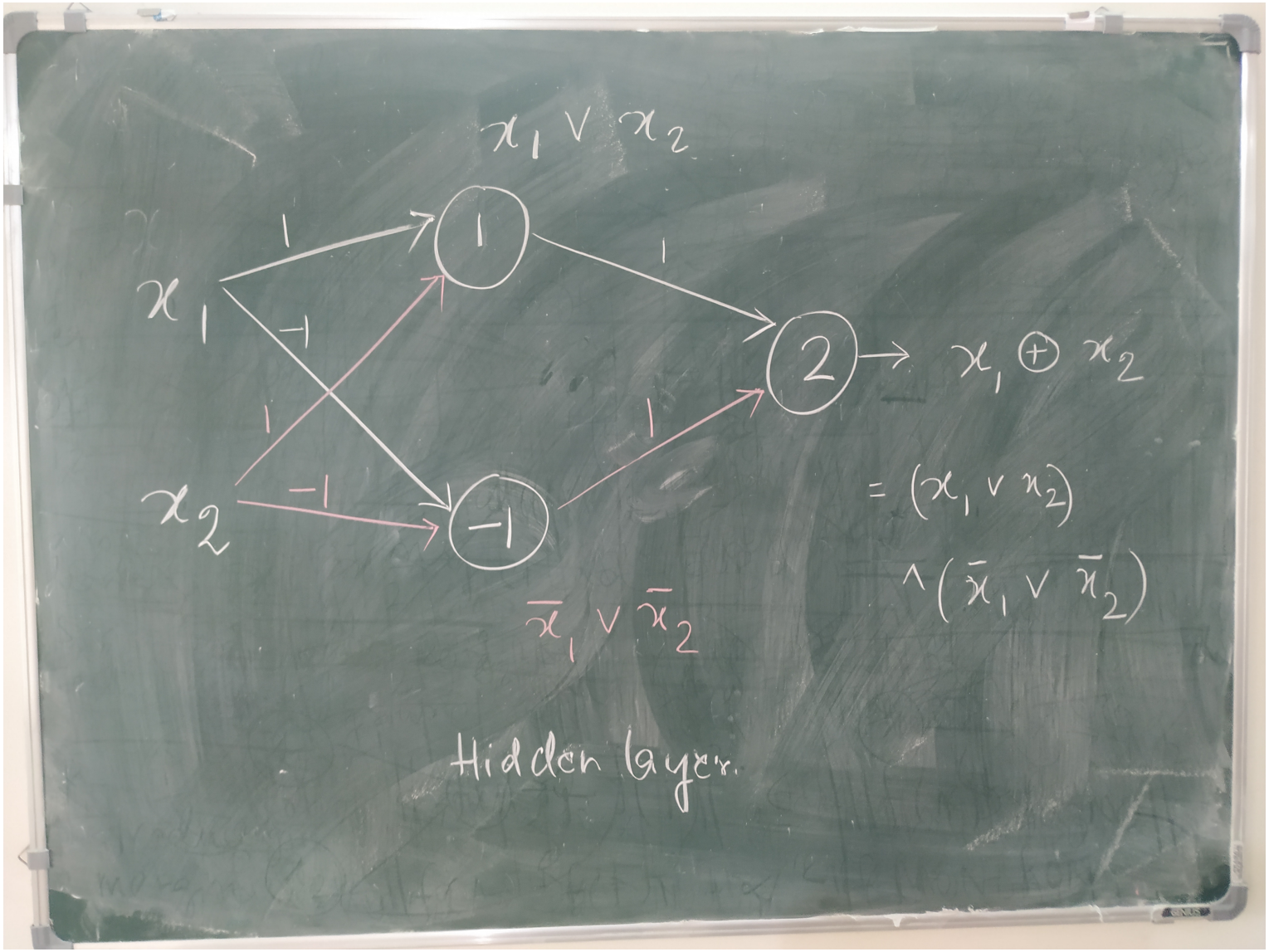
Tutorial IX


Majority Gate
x_1
x_2
x_d
1
1
1
k
* fire if at least k inputs are "1"
$(k \leq d)$

Tutorial X

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

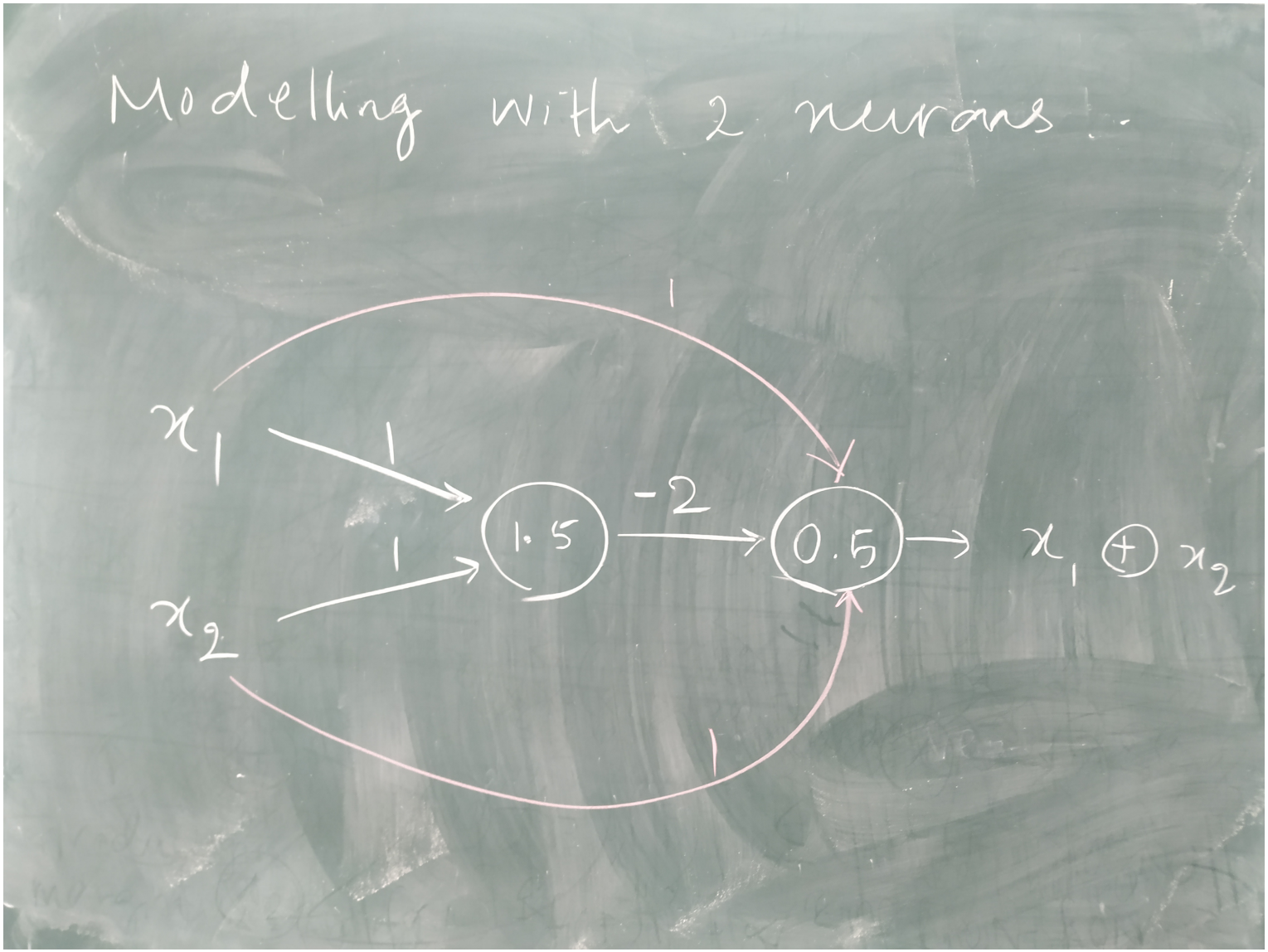
Tutorial XI


$x_1 \vee x_2$
1
1
x_1
-1
1
2
$x_1 \oplus x_2$
$= (x_1 \vee x_2)$
$\wedge (\bar{x}_1 \vee \bar{x}_2)$
x_2
-1
1
-1
$\bar{x}_1 \vee \bar{x}_2$
Hidden layer.

Tutorial XII

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

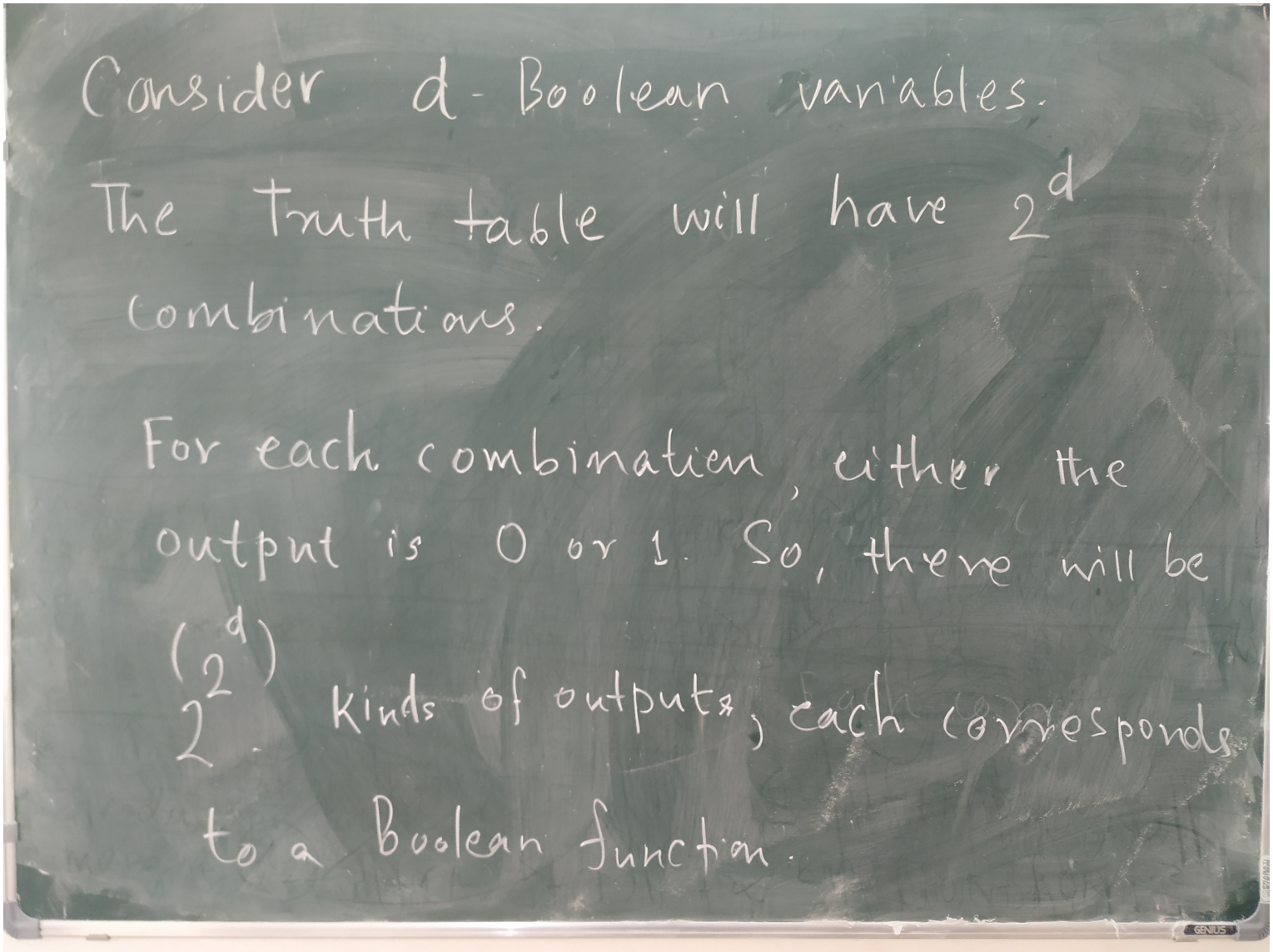
Tutorial XIII



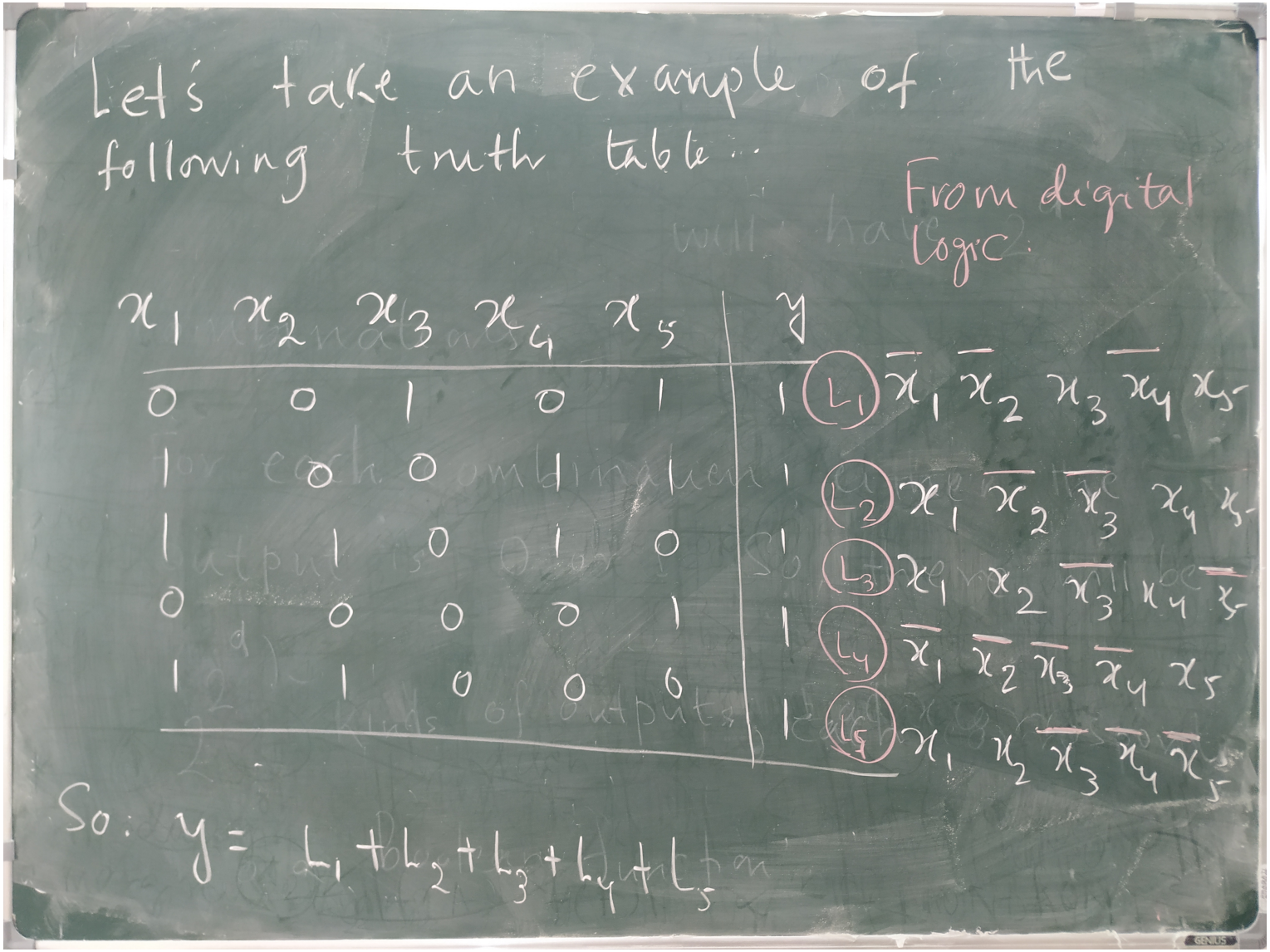
Tutorial XIV

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

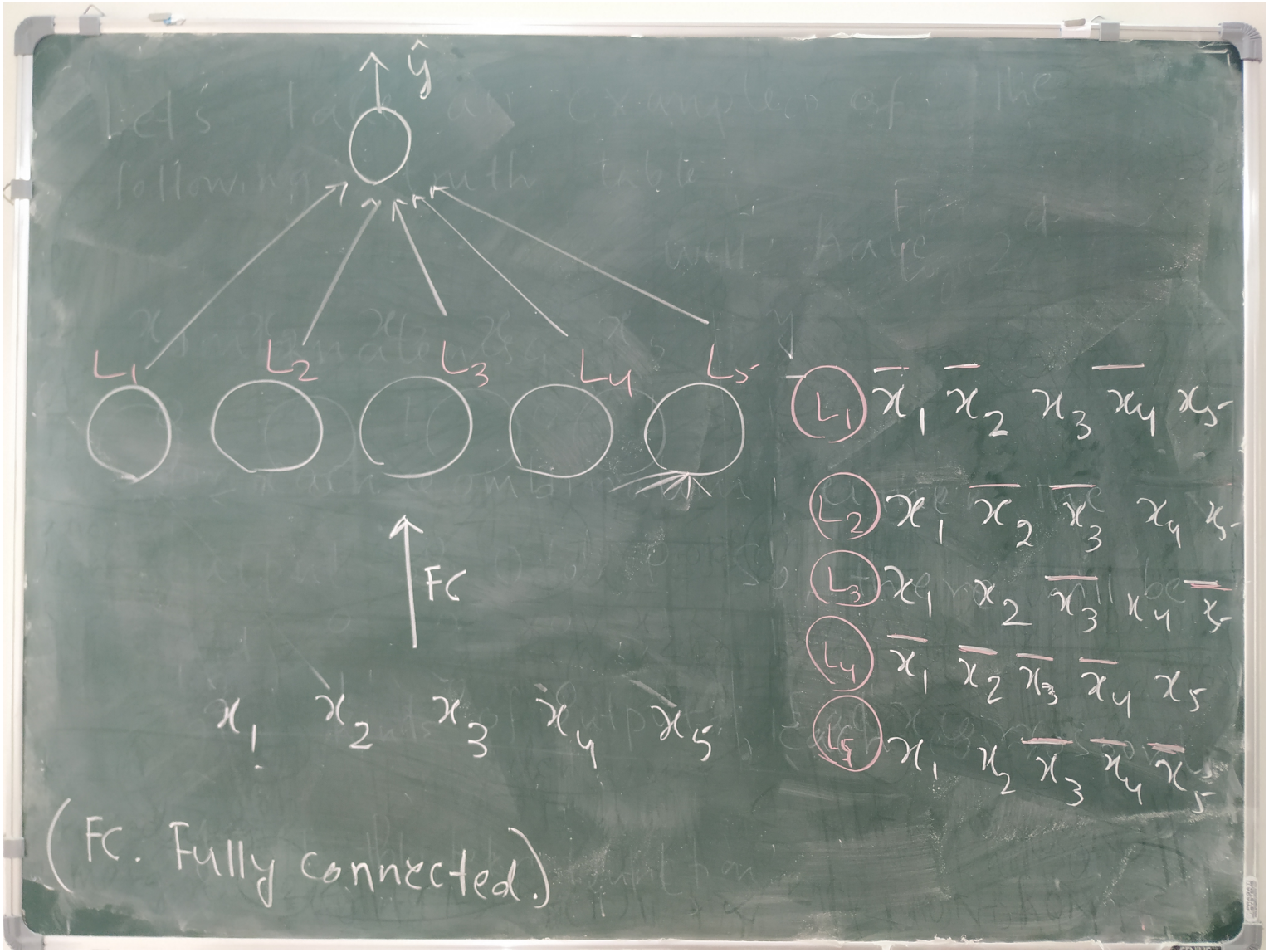
Tutorial XV


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^d) 2^{2^d} kinds of outputs, each corresponds to a Boolean function.

Tutorial XVI



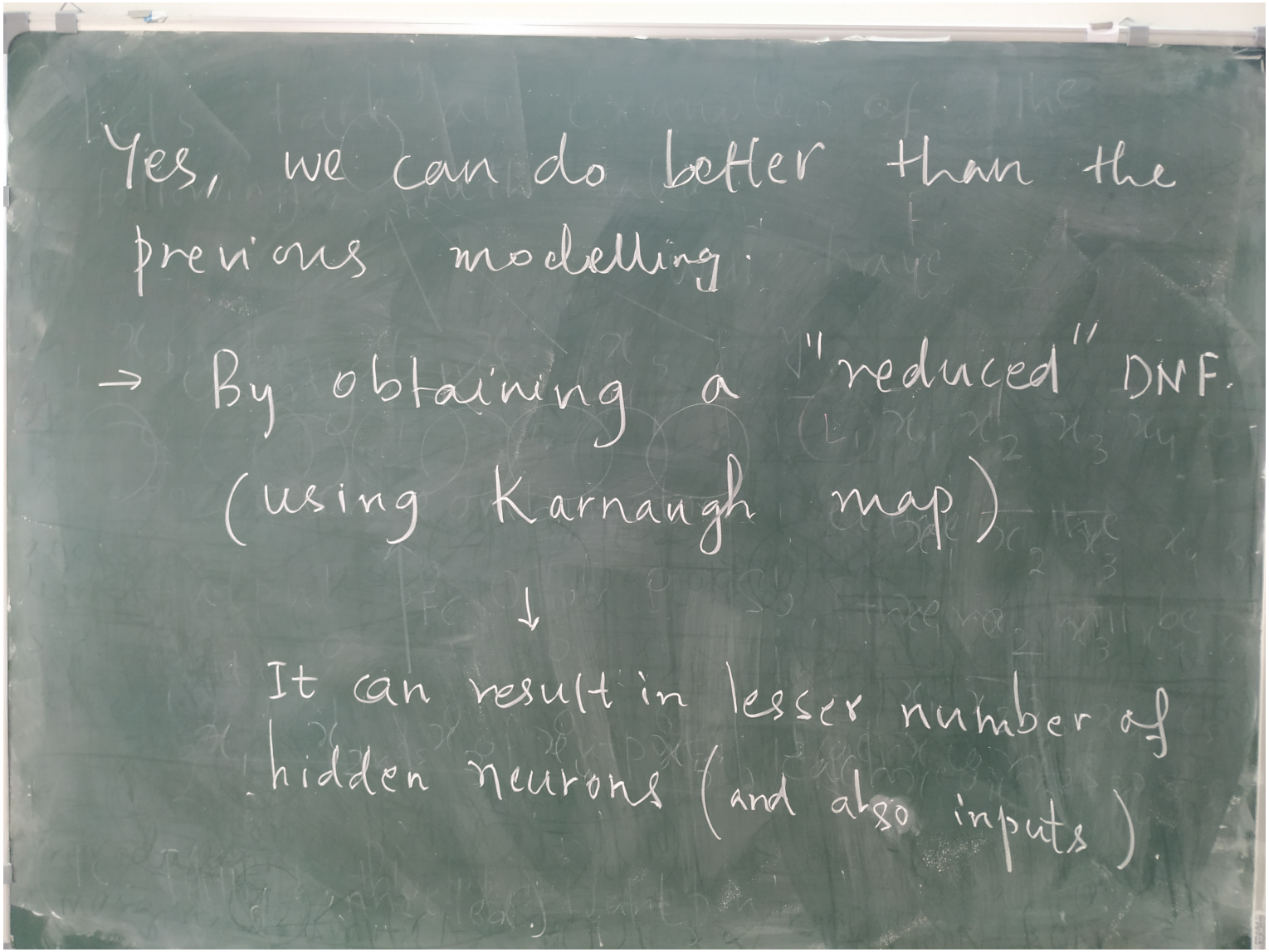
Tutorial XVII



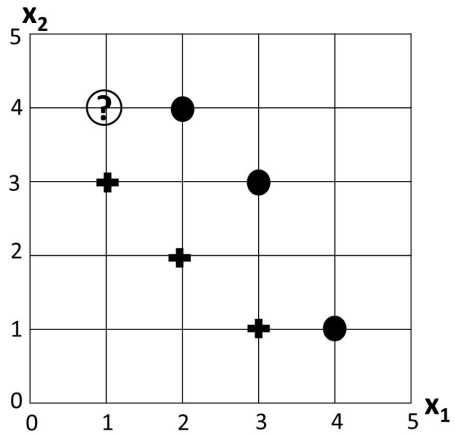
Tutorial XVIII

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

Tutorial XIX



8. 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 $(f(\mathbf{x}, \mathbf{w}) > 0?\ +1\ :\ -1)$ and classify the unknown data point by the learned perceptron. For this, answer the following questions.



Tutorial XXI

(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?

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

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

where, $\hat{y} = f(\mathbf{w}, \mathbf{x}) = w_1 x_1 + w_2 x_2 + w0$.
The update equation is:

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

The weight vector after introduction of each example is:

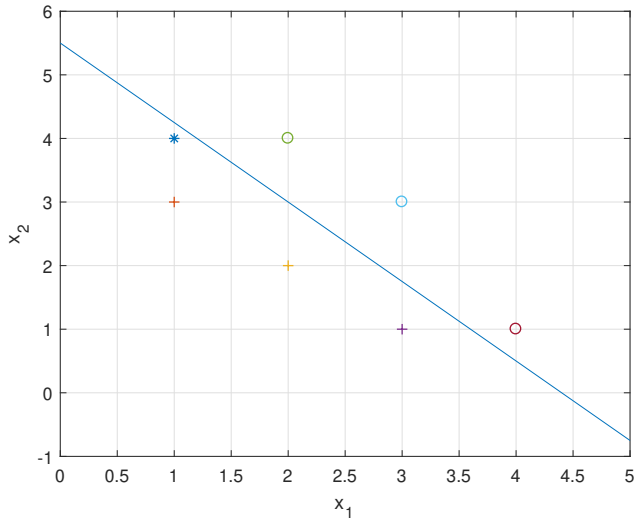
Tutorial XXIII

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

(b) Equation of perceptron is $w_1 x_1 + w_2 x_2 + w0 = 0$
(Put the values of these parameters)

(c) Yes. See the plot below

Tutorial XXIV



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

Tutorial XXV

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 $(1, 0, 0, \ldots, k-1 \text{ such } 0s)$, and class 2 is represented as $(0, 1, 0, \ldots, k-2 \text{ such } 0s)$.

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 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.

Basic Optimisation I

Gradient of a scalar function Let $\mathbf{x} = [x_1, \ldots, x_n]^\mathsf{T}$ and let $f(\mathbf{x})$ be a **scalar function** of $\mathbf{x}$. Then the derivative of $f(\mathbf{x})$ w.r.t. $\mathbf{x}$, called the **gradient vector** or **gradient** of $f(\mathbf{x})$ is a column vector denoted by

$$\nabla_{\mathbf{x}} f(\mathbf{x}) \text{ or } \nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right]^\mathsf{T}$$

Basic Optimisation II

Gradient of a vector function Let $\mathbf{x} = [x_1, \ldots, x_n]^\mathsf{T}$ and let $\mathbf{f}(\mathbf{x})$ be a **vector function** of $\mathbf{x}$, denoted by $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}, \ldots, f_m(\mathbf{x}]^\mathsf{T}$. Then, the derivative of $\mathbf{f}(\mathbf{x})$ w.r.t. $\mathbf{x}$, called the **Jacobian matrix** or **Jacobian** of $\mathbf{f}(\mathbf{x})$, is an $m \times n$ matrix denoted by

$$\mathbf{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla_{\mathbf{x}}^\mathsf{T} f_1(\mathbf{x}) \\ \vdots \\ \nabla_{\mathbf{x}}^\mathsf{T} f_m(\mathbf{x}) \end{bmatrix}$$

Basic Optimisation III

Hessian of a scalar function Let $\mathbf{x} = [x_1, \ldots, x_n]^\mathsf{T}$ and let $f(\mathbf{x})$ be a **scalar function** of $\mathbf{x}$. Then the second derivative of $f(\mathbf{x})$, called the **Hessian matrix** or **Hessian** of $f(\mathbf{x})$, is an $n \times n$ matrix denoted by

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

which is:

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial}{\partial x_1}\left(\frac{\partial f}{\partial x_1}\right) & \frac{\partial}{\partial x_1}\left(\frac{\partial f}{\partial x_2}\right) & \cdots & \frac{\partial}{\partial x_1}\left(\frac{\partial f}{\partial x_n}\right) \\ \frac{\partial}{\partial x_2}\left(\frac{\partial f}{\partial x_1}\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial f}{\partial x_2}\right) & \cdots & \frac{\partial}{\partial x_2}\left(\frac{\partial f}{\partial x_n}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_n}\left(\frac{\partial f}{\partial x_1}\right) & \frac{\partial}{\partial x_n}\left(\frac{\partial f}{\partial x_2}\right) & \cdots & \frac{\partial}{\partial x_n}\left(\frac{\partial f}{\partial x_n}\right) \end{bmatrix} = \begin{bmatrix} \nabla_{\mathbf{x}}^\mathsf{T} \frac{\partial f}{\partial x_1} \\ \vdots \\ \nabla_{\mathbf{x}}^\mathsf{T} \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Basic Optimisation IV

Gradient of a function (1) Let $\mathbf{c} = [c_1, \ldots, c_n]^\mathsf{T}$ and $\mathbf{x} = [x_1, \ldots, x_n]^\mathsf{T}$. Then the gradient of a linear scalar function $f(\mathbf{x}) = \mathbf{c}^\mathsf{T}\mathbf{x} = \mathbf{x}^\mathsf{T}\mathbf{c}$ w.r.t. $\mathbf{c}$

$$\nabla_{\mathbf{c}} f(\mathbf{x}) = \mathbf{x}$$

Gradient of a function (2) If $f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{x}$, then

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = 2\mathbf{x}$$

Gradient of a function (3) If $f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x}$, then

$$\nabla_{\mathbf{x}} f = 2\mathbf{A}\mathbf{x}$$

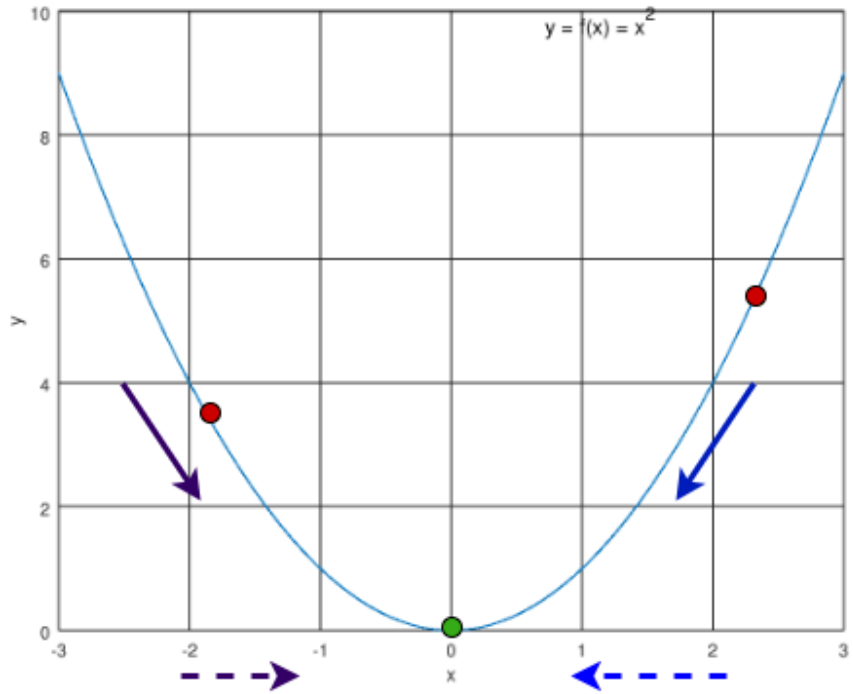
Basic Optimisation V

Let's look at minimisation problems for functions that are continuous and differentiable.

- If the derivative of the function is positive, the function is increasing.
  - Dont move in that direction, because youll be moving away from a minimum.
- If the derivative of the function is negative, the function is decreasing.
  - Keep going, since youre getting closer to a minimum.

Basic Optimisation VI

Let $f(x) = x^2$. The function looks like this:



The arrows show movement of next functional value, and the dotted arrows show the corresponding direction of movement of x.

Basic Optimisation VII

Here is a very simple gradient descent procedure:

1. Initialize x to some value
2. **while** stopping criterion is not met
   - 2.1 Calculate the gradient of the function, $\nabla_x f$
   - 2.2 $x := x - \eta \nabla_x f$
3. **return** x

Notice step 2.2. above: x will move right, if $\nabla_x f$ is negative, and it will move left, if $\nabla_x f$ is positive.

Basic Optimisation VIII

12. Using gradient descent, obtain the value of x that minimizes $f(x) = (x-2)^2 - 5$. Starting value of $x = 3$ and $\eta = 1$.

Answer. Derivative of f w.r.t. x: $\nabla f = 2(x-2)$

- $x = 3$: $\nabla f|_{x=3} = 2$; $x = 3 - 2 = 1$; $f(1) = -4$
- $x = 1$: $\nabla f|_{x=1} = -2$; $x = 1 - (-2) = 3$; $f(3) = -4$.
- ... gets repeated.

Basic Optimisation IX

13. Solve the same question with same starting point, but with $\eta = 0.5$.

Answer. Derivative of f w.r.t. x: $\nabla f = 2(x-2)$

- $x = 3$: $\nabla f|_{x=3} = 2$; $x = 3 - 0.5 \times 2 = 2$; $f(2) = -5$
- $x = 2$: $\nabla f|_{x=2} = 0$; $x = 2 - 0.5 \times 0 = 2$; $f(2) = -5$.
- $x = 2$: $\nabla f|_{x=2} = 0$; $x = 2 - 0.5 \times 0 = 2$; $f(2) = -5$.
- Value of f doesn't change further. So, stopping criterion met. Return $x = 2$. This is same as the exact solution i.e. Find root of $\nabla f = 0$.

Basic Optimisation X

Gradient descent is guaranteed to eventually find a local minimum if:

- the learning rate is set appropriately (sometimes, using adaptive learning rate); $\eta \in [0.0001, 1]$.
- a finite local minimum exists (i.e. the function doesn't keep decreasing forever).

Basic Optimisation XI

Various stopping criteria for gradient descent:

- Stop when the norm of the gradient is below some threshold, θ

$$||\nabla f|| < \theta$$

  This is checking the distant the gradient is from the origin, $\mathbf{0}$.

- Maximum number of iterations is reached.

Basic Optimisation XII

It is straightforward to extend the gradient descent procedure to scalar functions with multiple variables.

14. Let $f(x_1, x_2) = 3x_1^2 - 2x_1x_2 + x_2^2 - 5$. Initial values $x_1 = 1$, $x_2 = 1$. Fix $\eta = 1$.

Answer. Present value of f: $f(1, 1) = 3 - 2 + 1 - 5 = -3$.
The partial derivatives are:

$$\nabla_{x_1} f = 6x_1 - 2x_2$$
$$\nabla_{x_2} f = 2x_2 - 2x_1$$

Basic Optimisation XIII

Update the present $x_{1,2}$:

$$\begin{aligned} x_1 &= x_1 - \eta \nabla_{x_1} f \\ &= 1 - (6 - 2) = -3 \\ x_2 &= x_2 - \eta \nabla_{x_2} f \\ &= 1 - (2 - 2) = 1 \end{aligned}$$

New value of f: $f(-3, 1) = 29$. Update the present $x_{1,2}$ using gradients:

$$\begin{aligned} x_1 &= x_1 - \eta \nabla_{x_1} f \\ &= -3 - (-18 - 2) = 17 \\ x_2 &= x_2 - \eta \nabla_{x_2} f \\ &= 1 - (-6 - 2) = 9 \end{aligned}$$

New value of f: $f(17, 9) = 637$.

Basic Optimisation XIV

15. Solve the above question with $\eta = 0.1$.

Answer. Update the present $x_{1,2}$:

$$x_1 = 1 - 0.1(6 - 2) = 0.6$$
$$x_2 = 1 - 0.1(2 - 2) = 1$$

New value of f: $f(0.6, 1) = -4.12$. Update the present $x_{1,2}$ using gradients:

$$x_1 = 0.6 - 0.1(3.6 - 2) = 0.44$$
$$x_2 = 1 - 0.1(2 - 1.2) = 0.92$$

New value of f: $f(0.44, 0.92) = -4.38$.