Neural Networks

(Course: Introduction to Data Science)

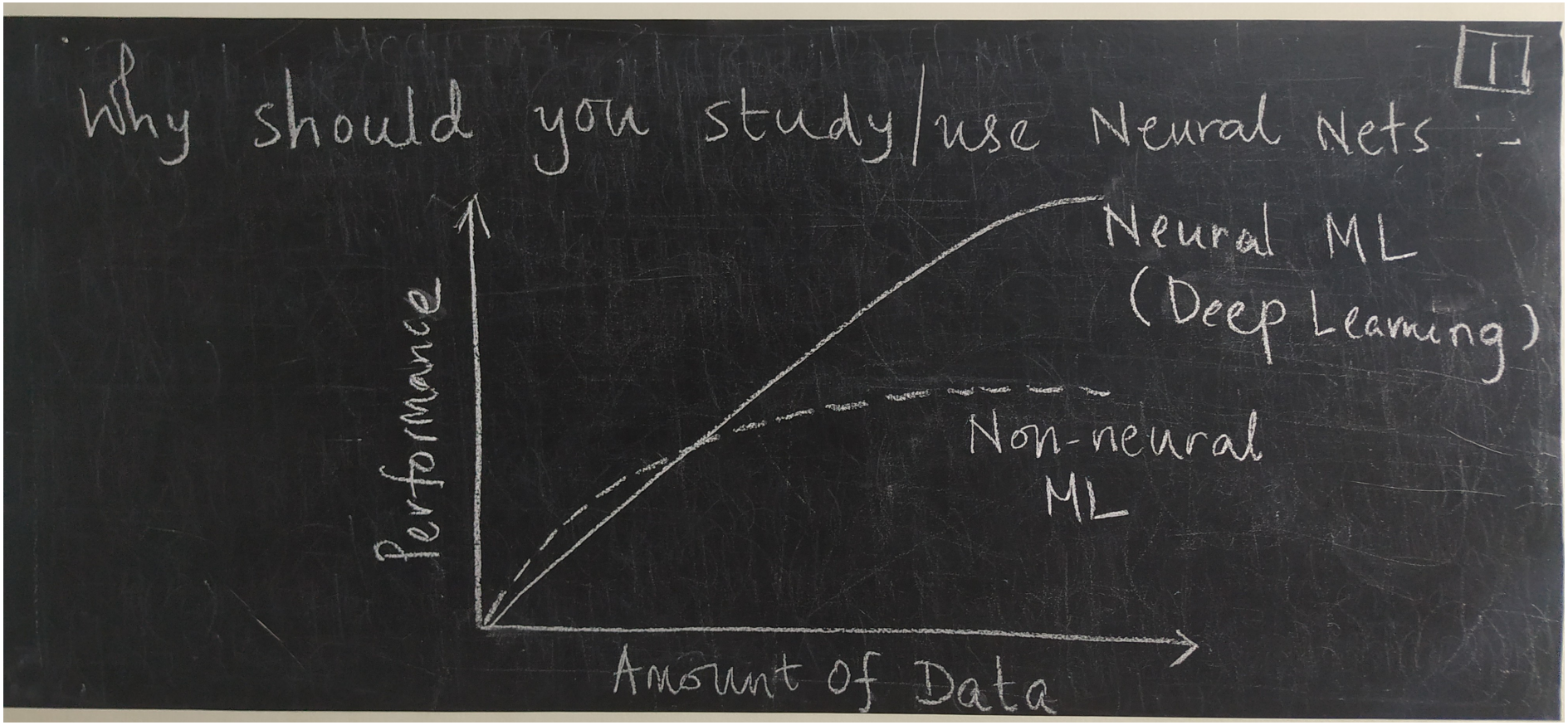
Tirtharaj Dash

BITS Pilani, K.K. Birla Goa Campus

tirtharaj@goa.bits-pilani.ac.in

October 30, 2020

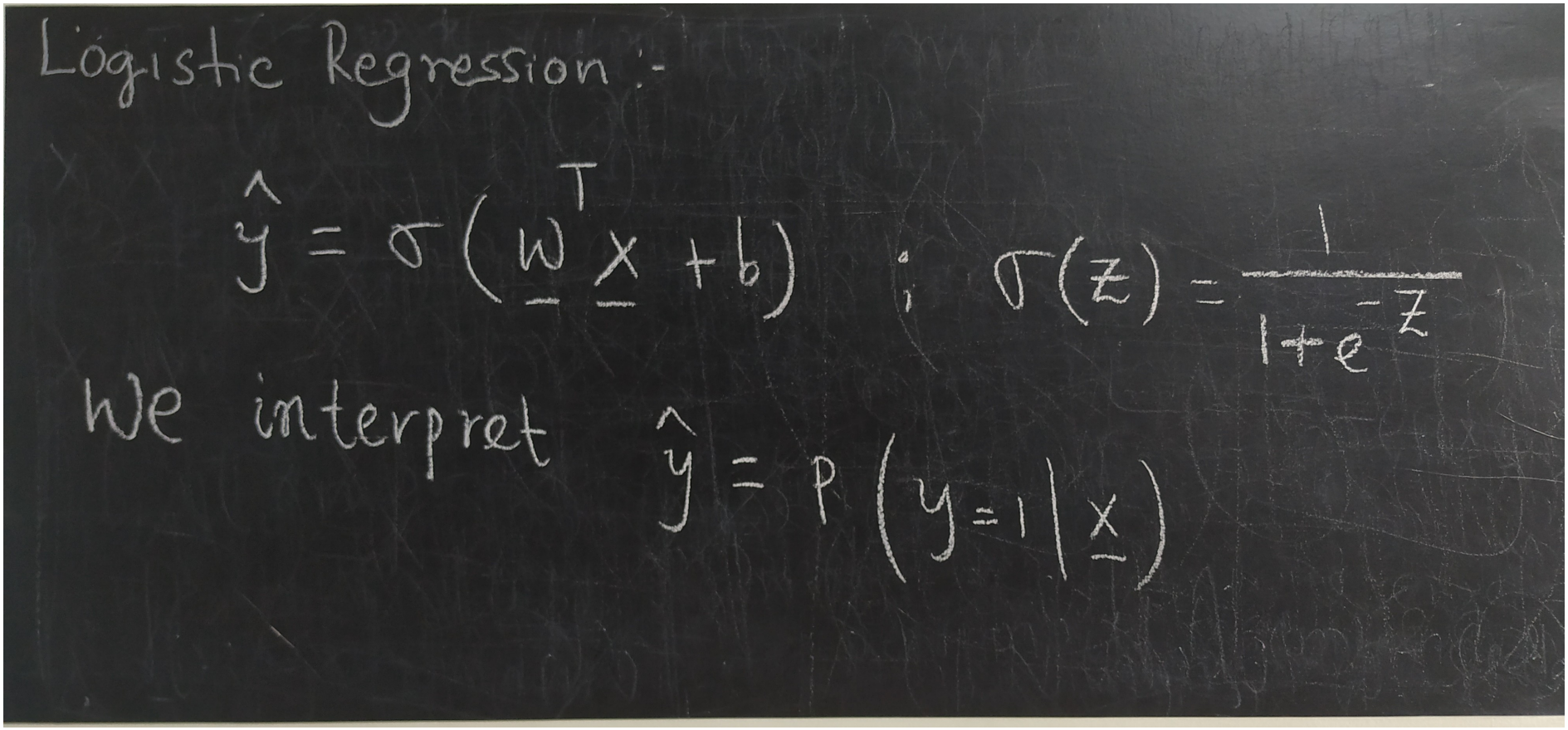
Why study NNs?



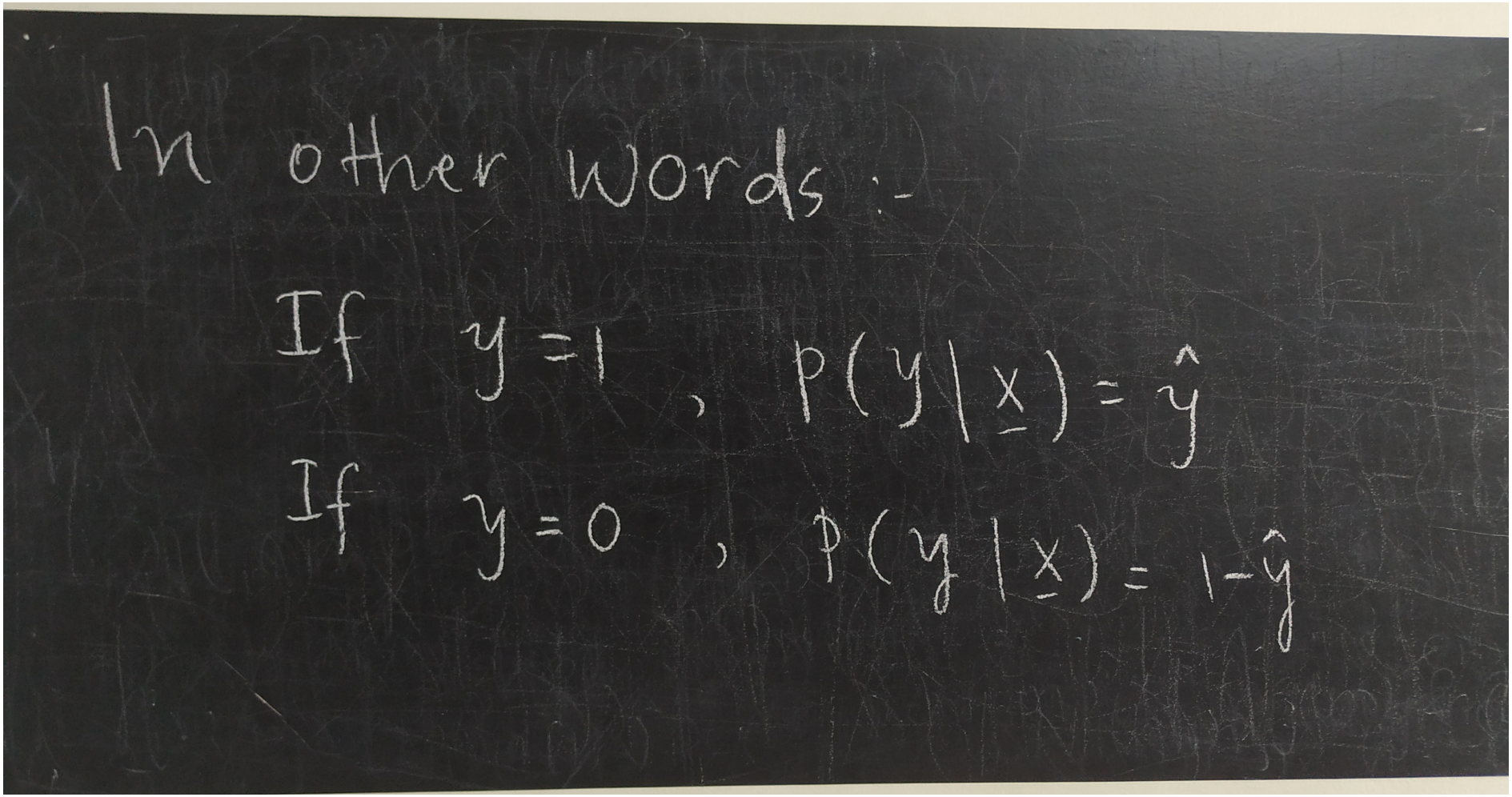
In this lecture:

- Logistic Regression as a NN (Perceptron)
- Issues with perceptron
- Layered NNs (Multilayer Perceptron)
- Activation functions

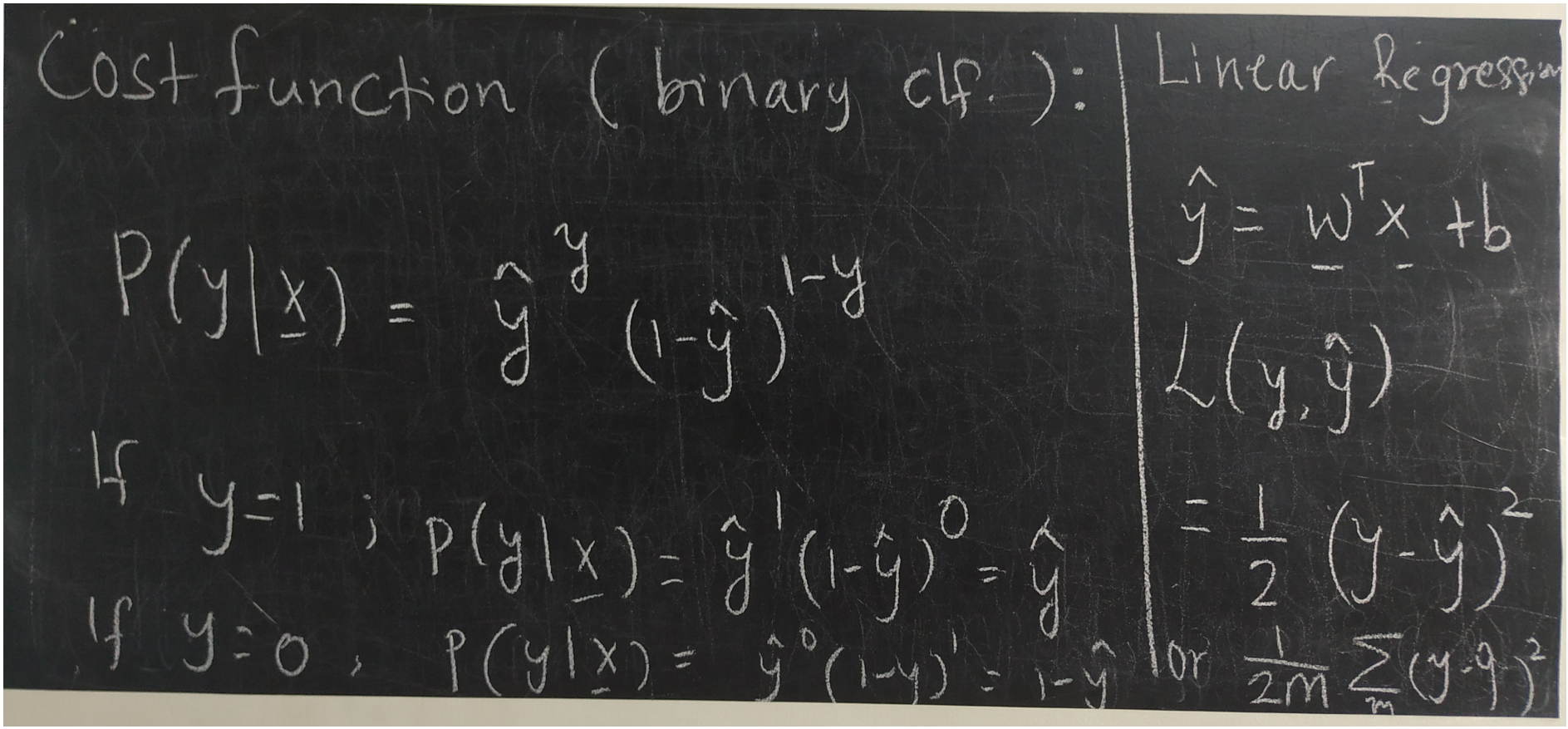
Logistic Regression as NN I



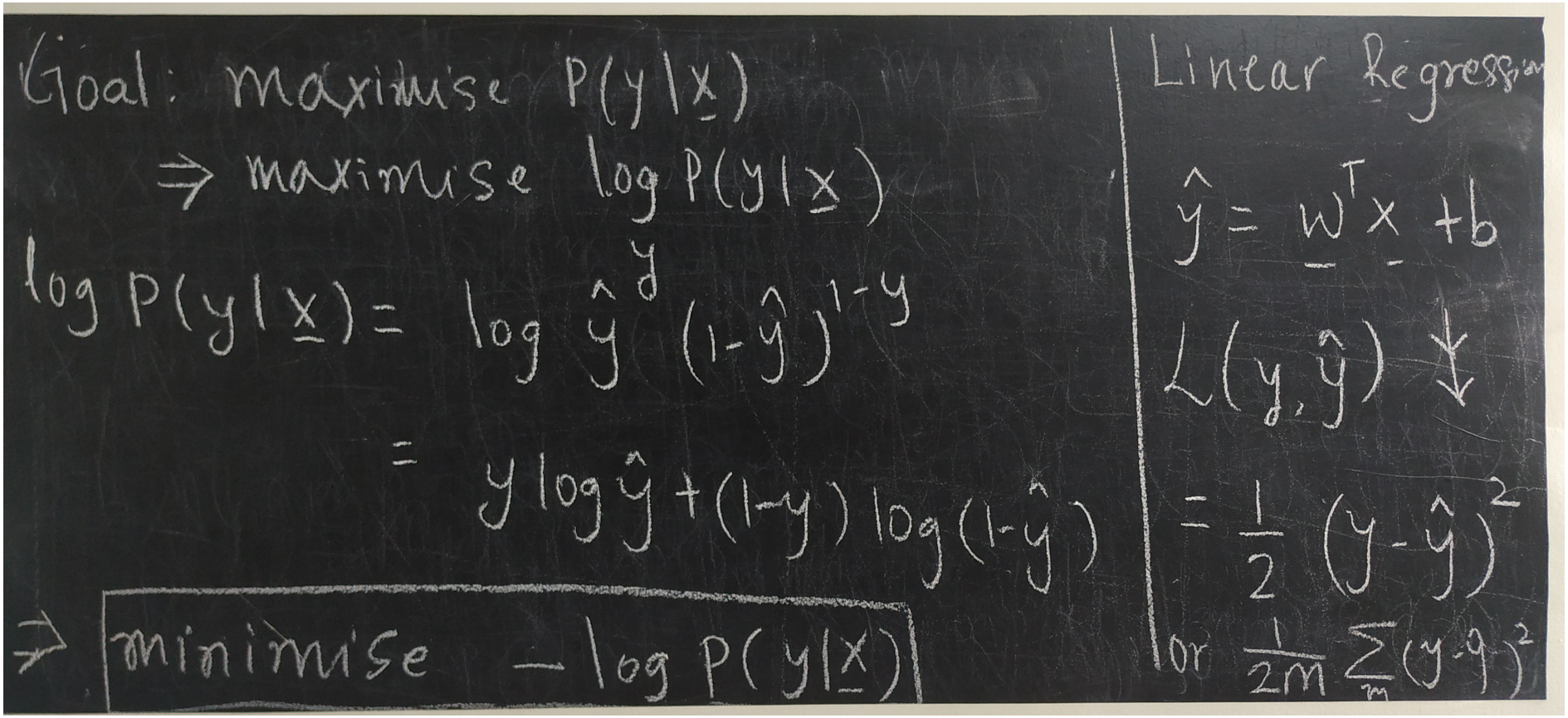
Logistic Regression as NN II



Logistic Regression as NN III



Logistic Regression as NN IV

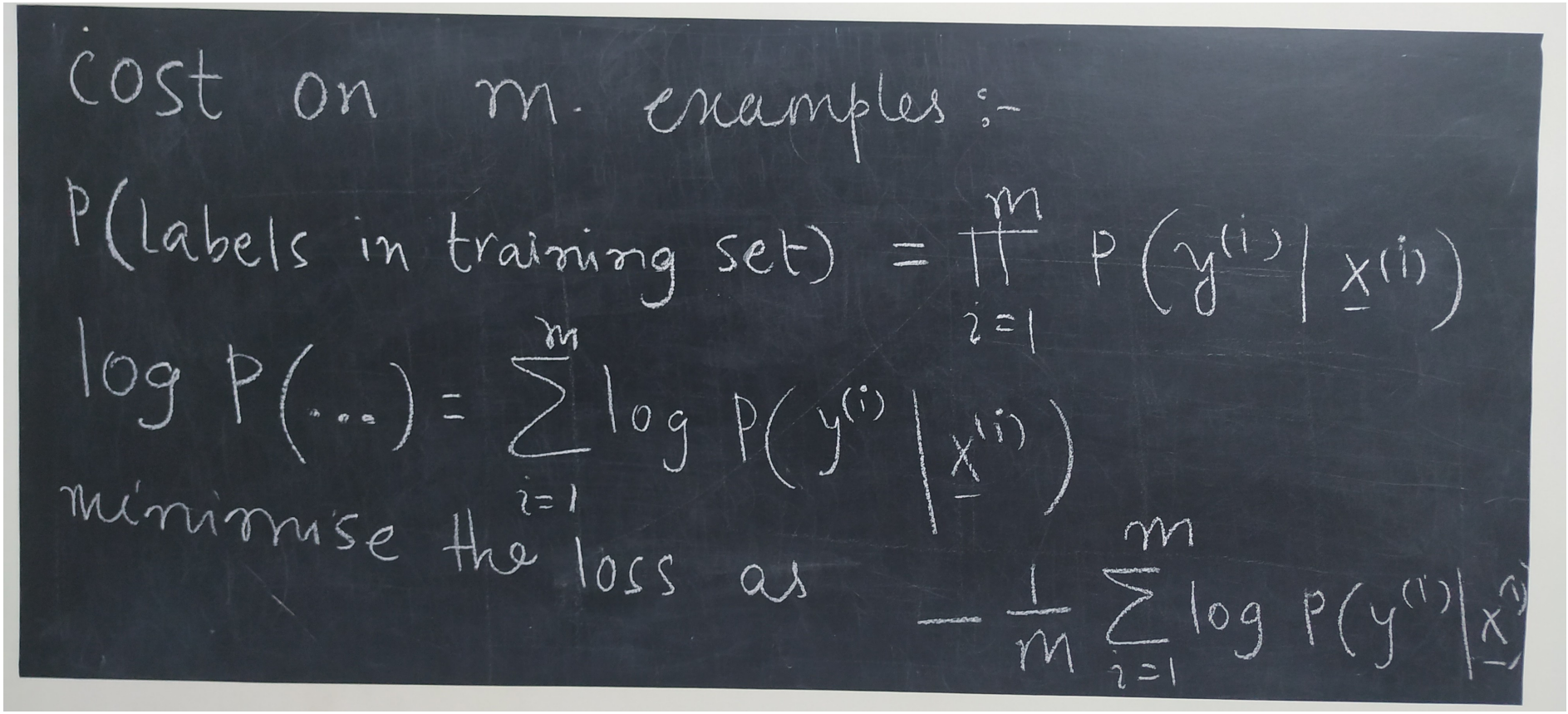


The cost function: $-\{y \log(\hat{y}) + (1-y) \log(1-\hat{y})\}$ is known as **cross-entropy loss function**.

Logistic Regression as NN V

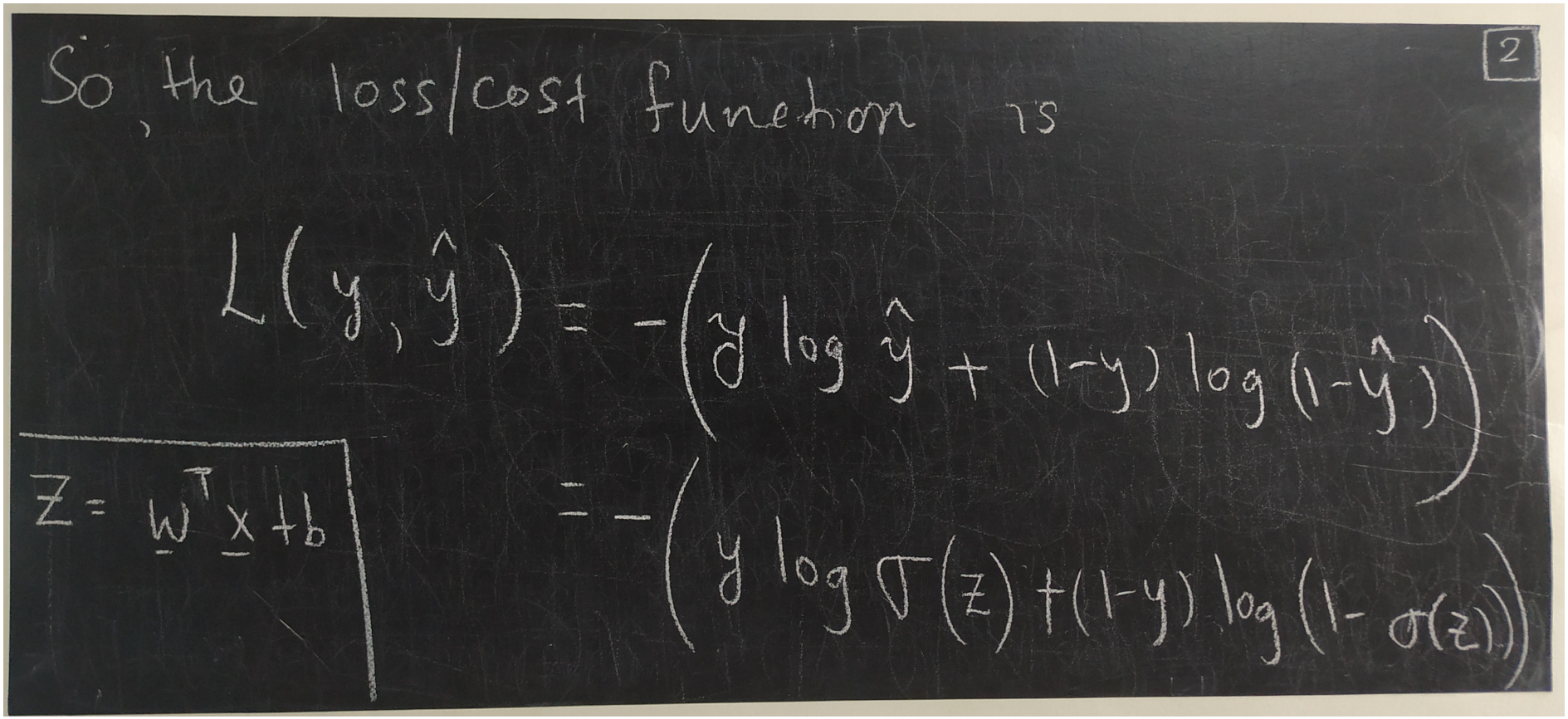
- Cross-entropy function computes the entropy between two different probability distribution.
- As $\hat{y} \in [0, 1]$, we can see that $(\hat{y}, 1 - \hat{y})$ is the probability distribution obtained from the logistic model, given any example x or $\underline{x}$.
- *Where is the other probabibility distribution?* We can treat the labels as a probability distribution. Any example x is associated with a label 1 or 0 (1 or 2, or whatever). When we convert these to one-hot encoded vector we get the labelling class 1: [1,0], class 2:[0,1]. This itself is a probability distribution. Therefore, each example is associated with a true probability distribution that the logistic model tries to reach.

Logistic Regression as NN VI



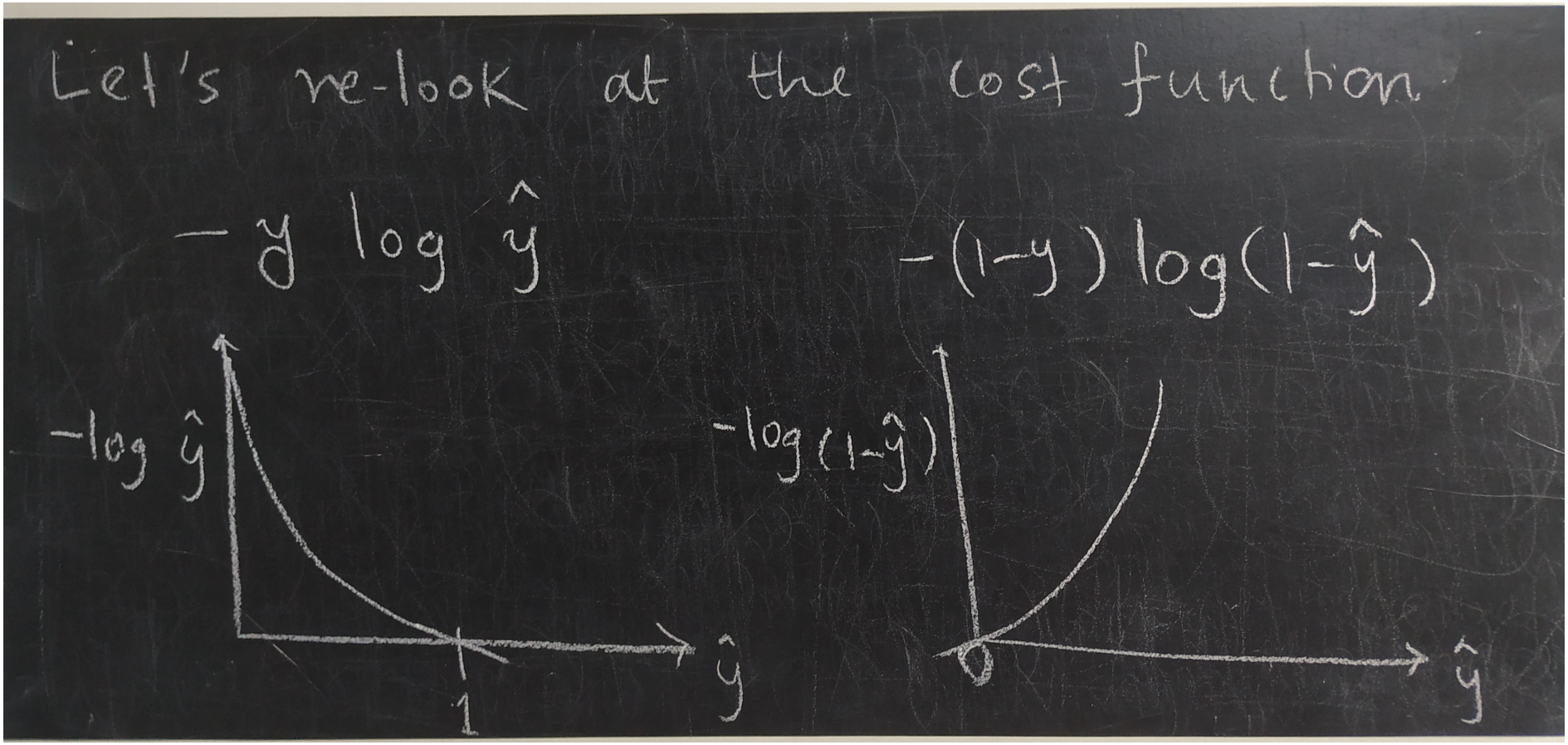
Logistic Regression as NN VII

For convenience, let's stay with the loss for a single example:

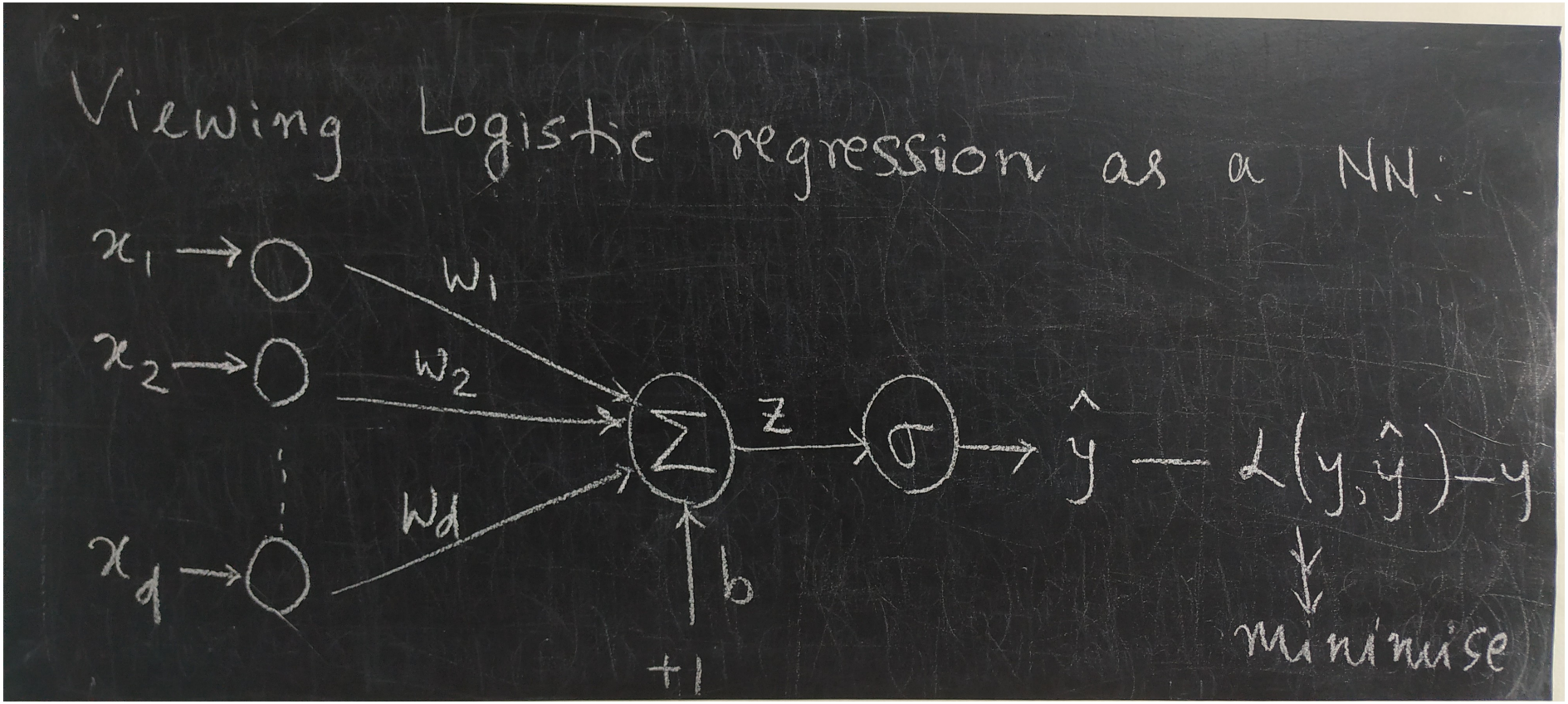


Logistic Regression as NN VIII

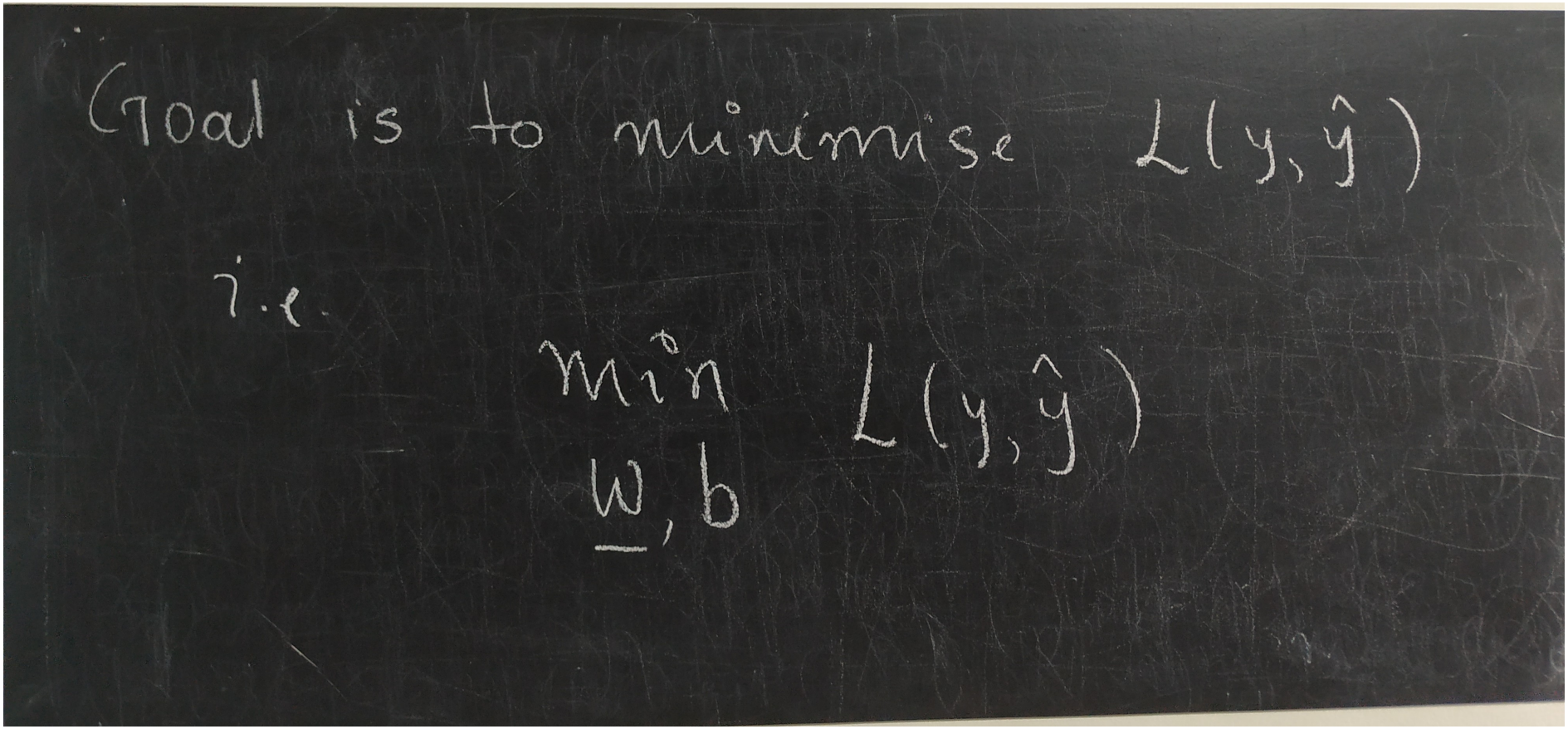
What do these two terms in the loss try to achieve?



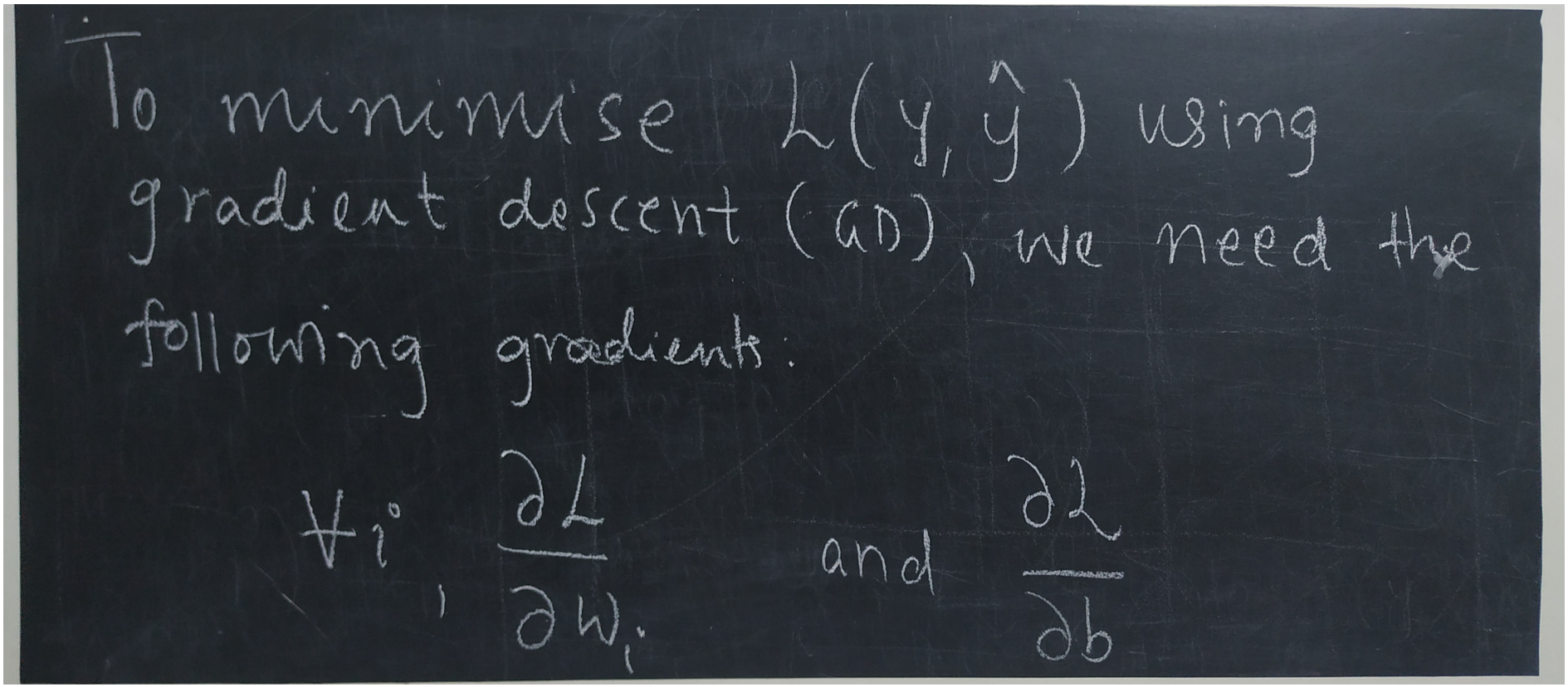
Logistic Regression as NN IX



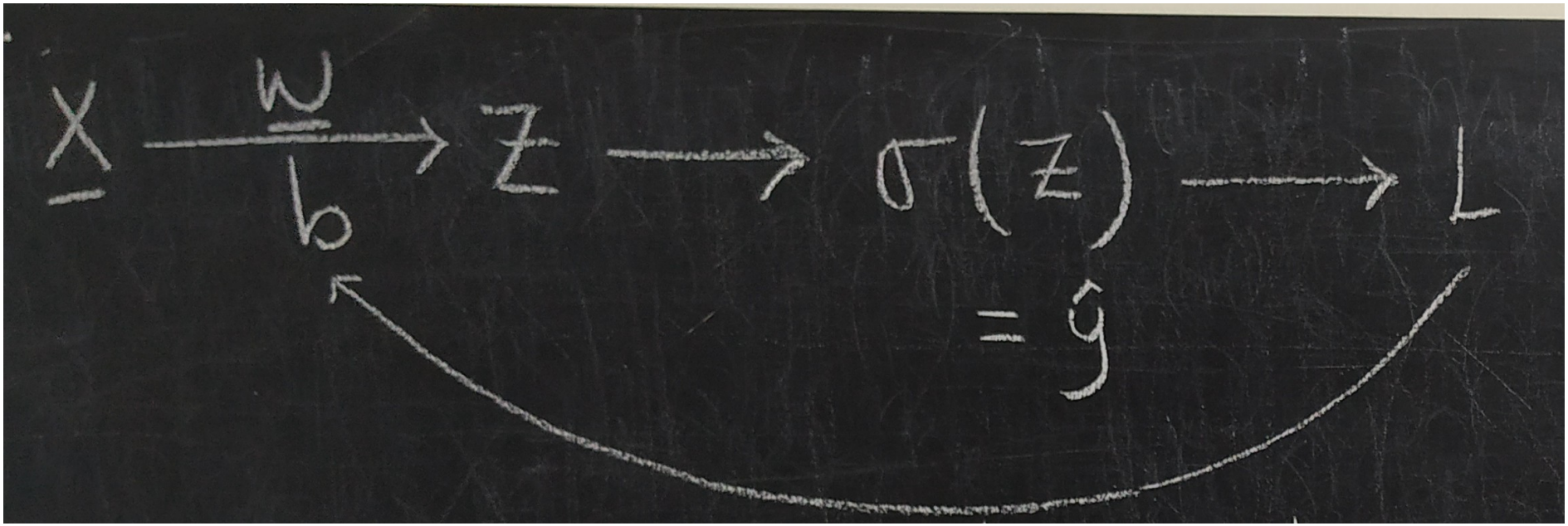
Logistic Regression as NN X



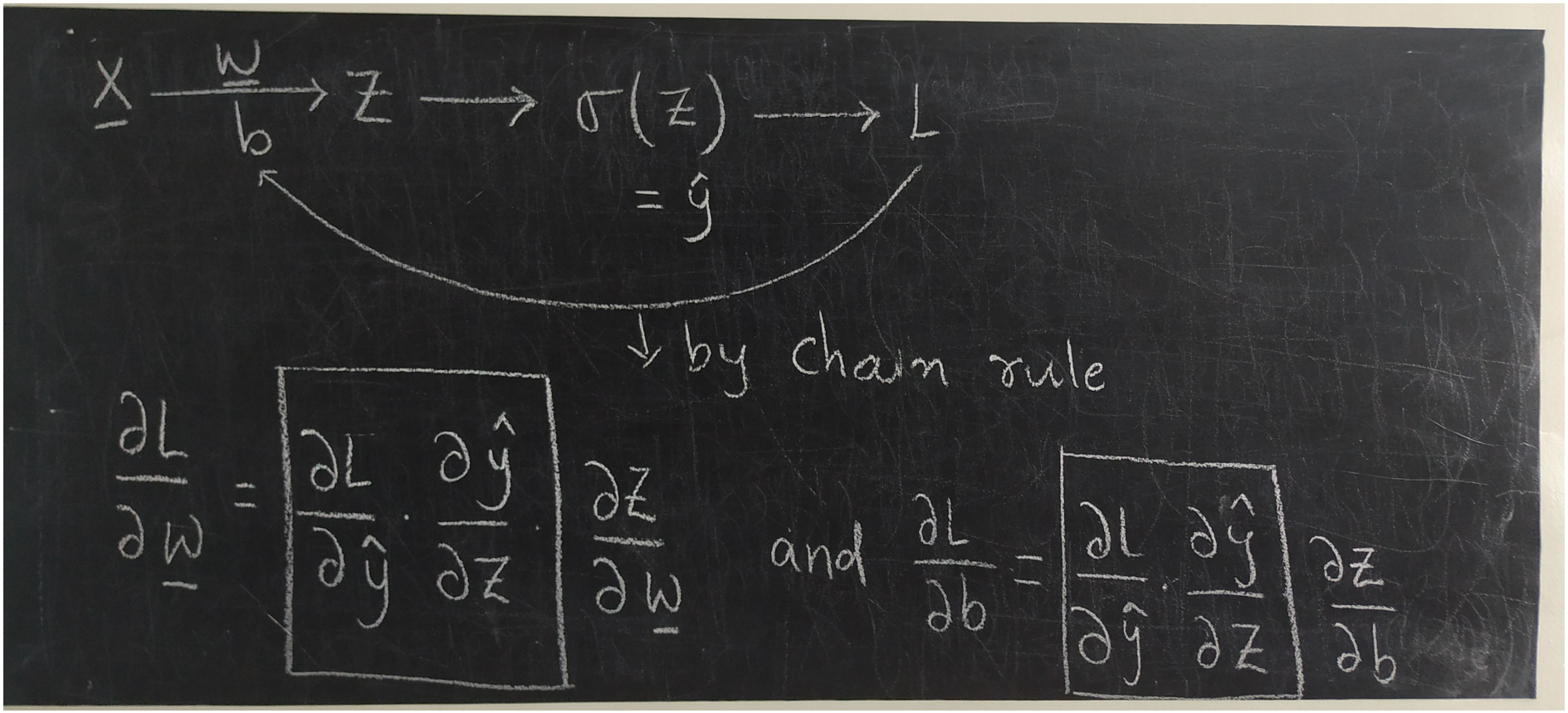
Logistic Regression as NN XI



Logistic Regression as NN XII



Logistic Regression as NN XIII



Logistic Regression as NN XIV

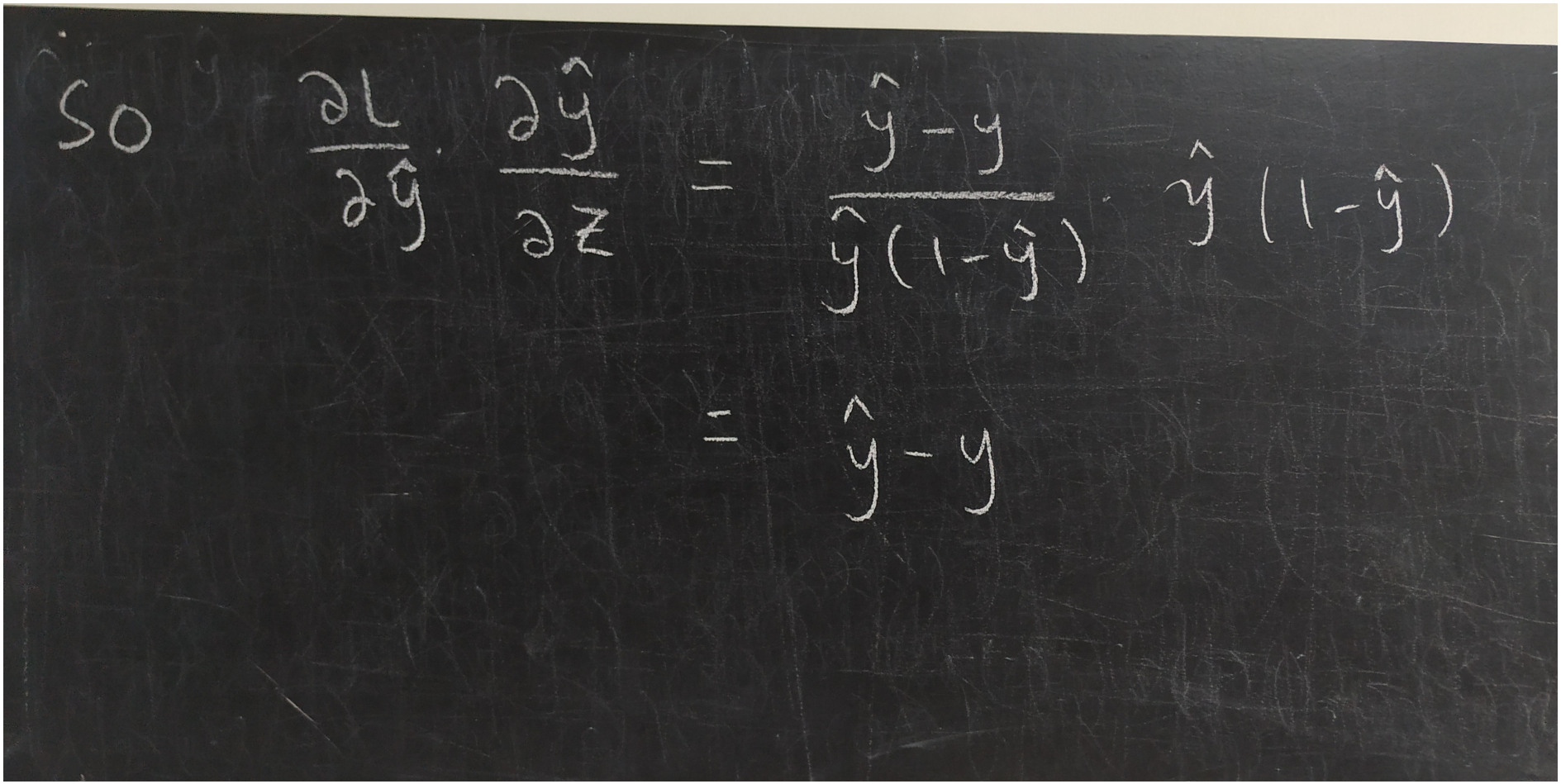
$$L = -y \log \hat{y} - (1-y) \log (1-\hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

as $\frac{\partial}{\partial z}\sigma(z) = \sigma(z)(1-\sigma(z))$

Logistic Regression as NN XV

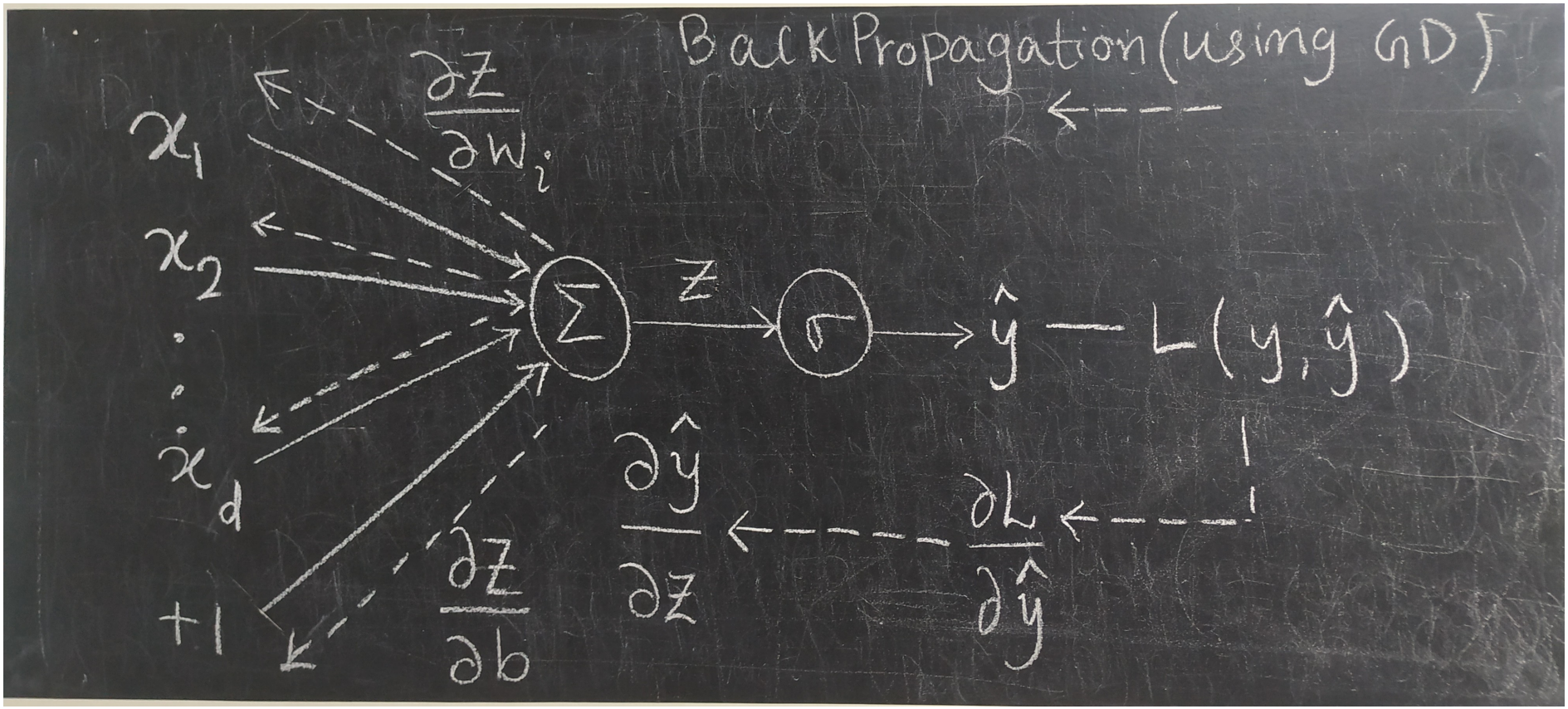


Logistic Regression as NN XVI

$$\frac{\partial L}{\partial w_i} = (\hat{y} - y)\frac{\partial z}{\partial w_i} = (\hat{y} - y)\, x_i$$

$$\frac{\partial L}{\partial b} = (\hat{y} - y)\frac{\partial z}{\partial b} = (\hat{y} - y) \cdot 1$$

Logistic Regression as NN XVII



Logistic Regression as NN XVIII

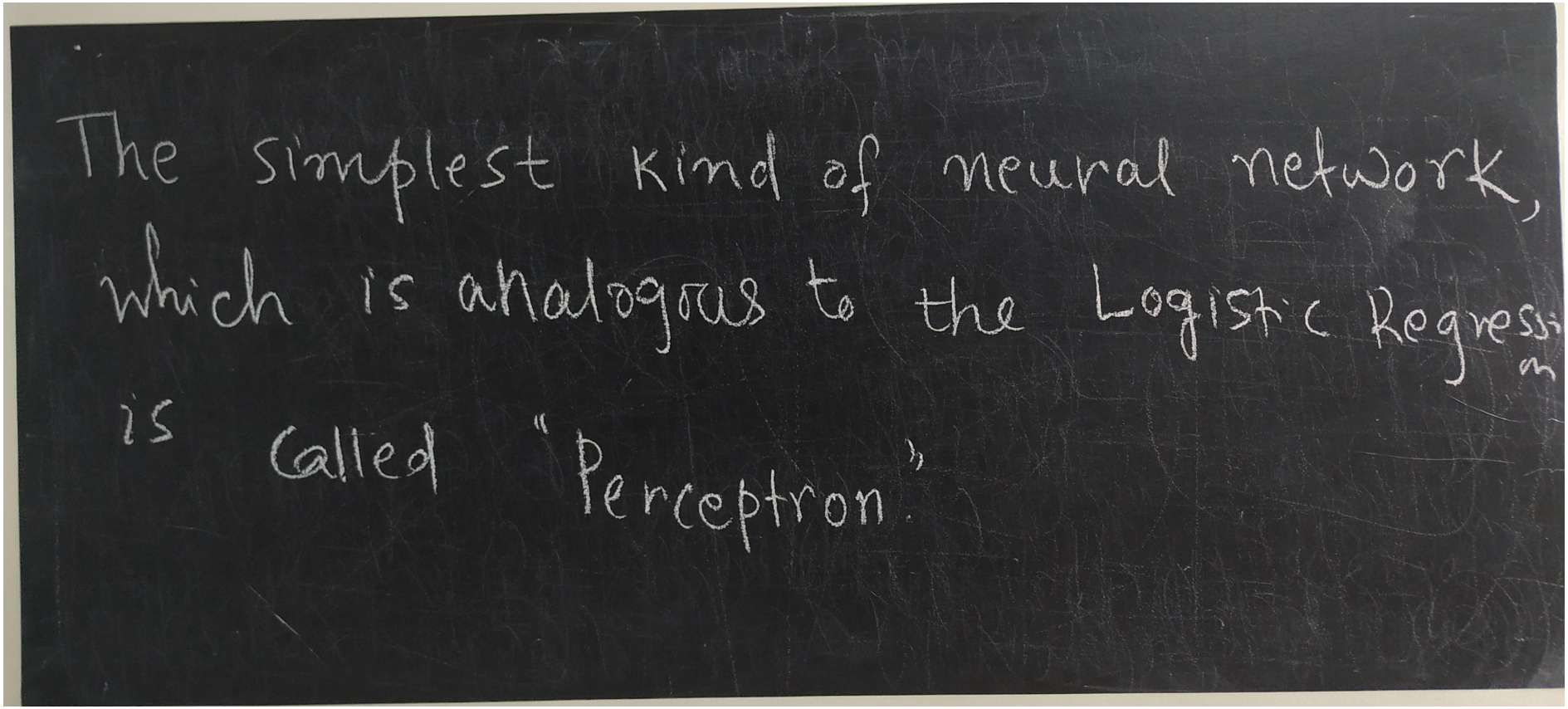
To update the parameters:

$$w_i = w_i - \alpha \frac{\partial \mathcal{L}}{\partial w_i}, \ \forall i \in \{1, \ldots, d\}$$

$$b = b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

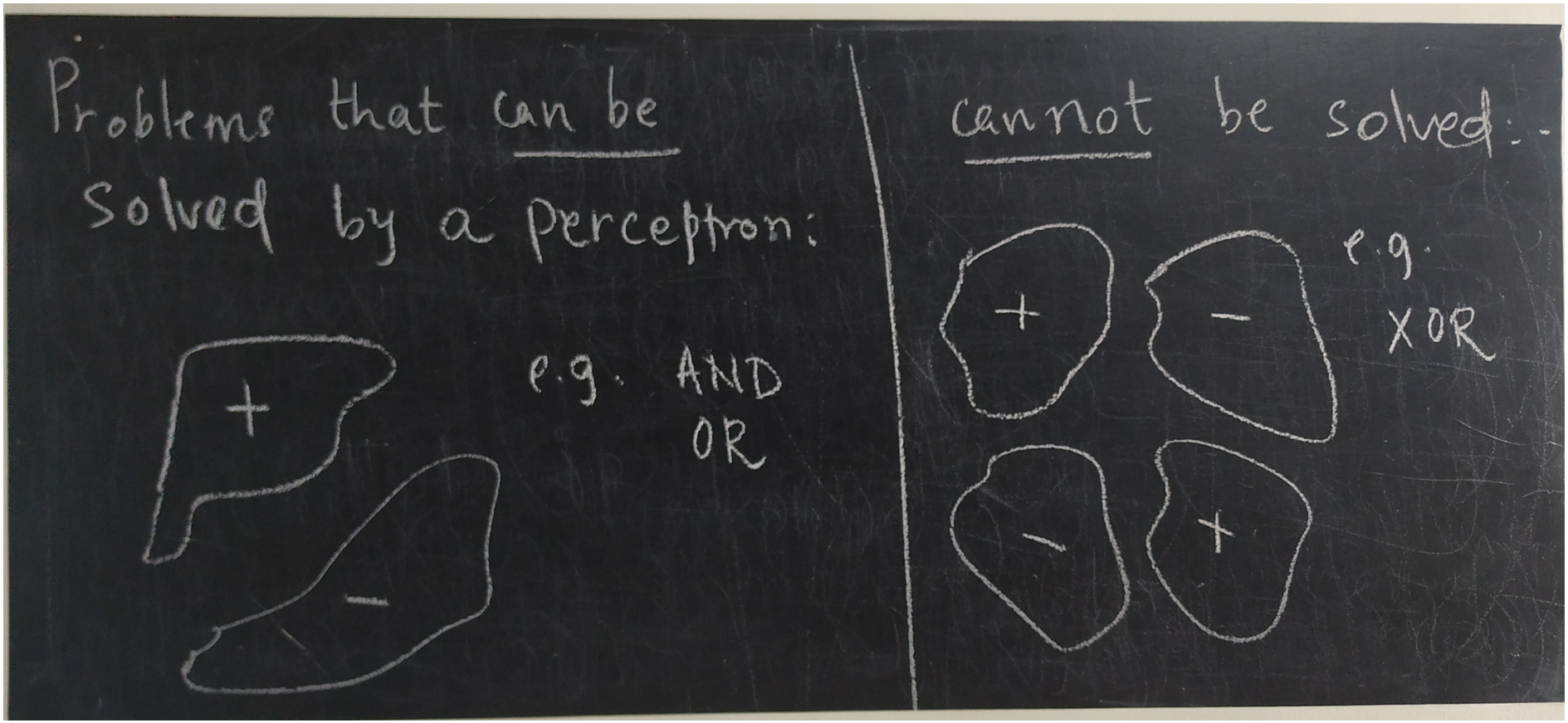
where, $\alpha \in (0, 1]$ is the learning rate.

Logistic Regression as NN XIX

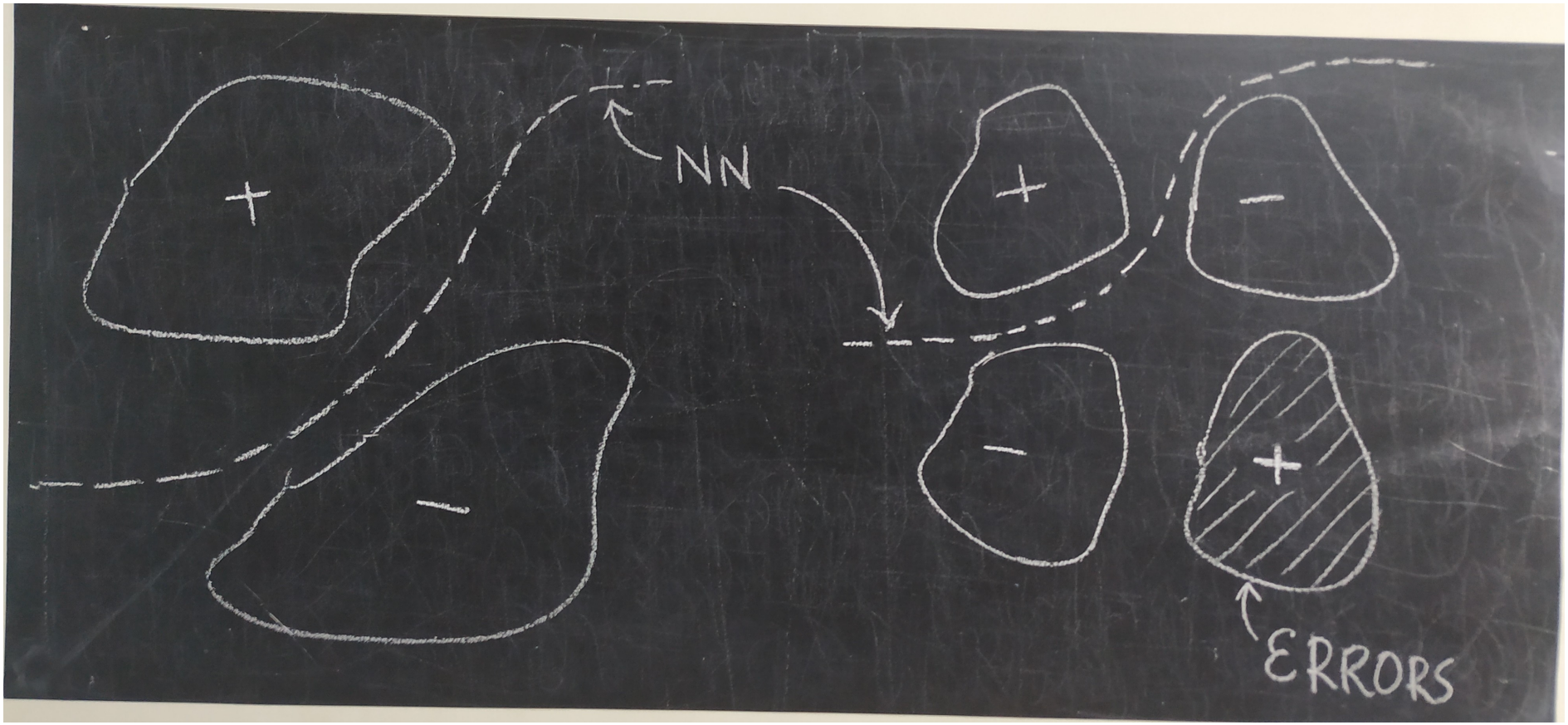


In the NN world, these parameters w and b are called **synaptic weights** or "simply" **weights**; z is called the **net input**. The function that operates on the net input in any neuron is called **activation function**.

Issues with Perceptron I



Issues with Perceptron II



- Many real-world problems are of second kind.

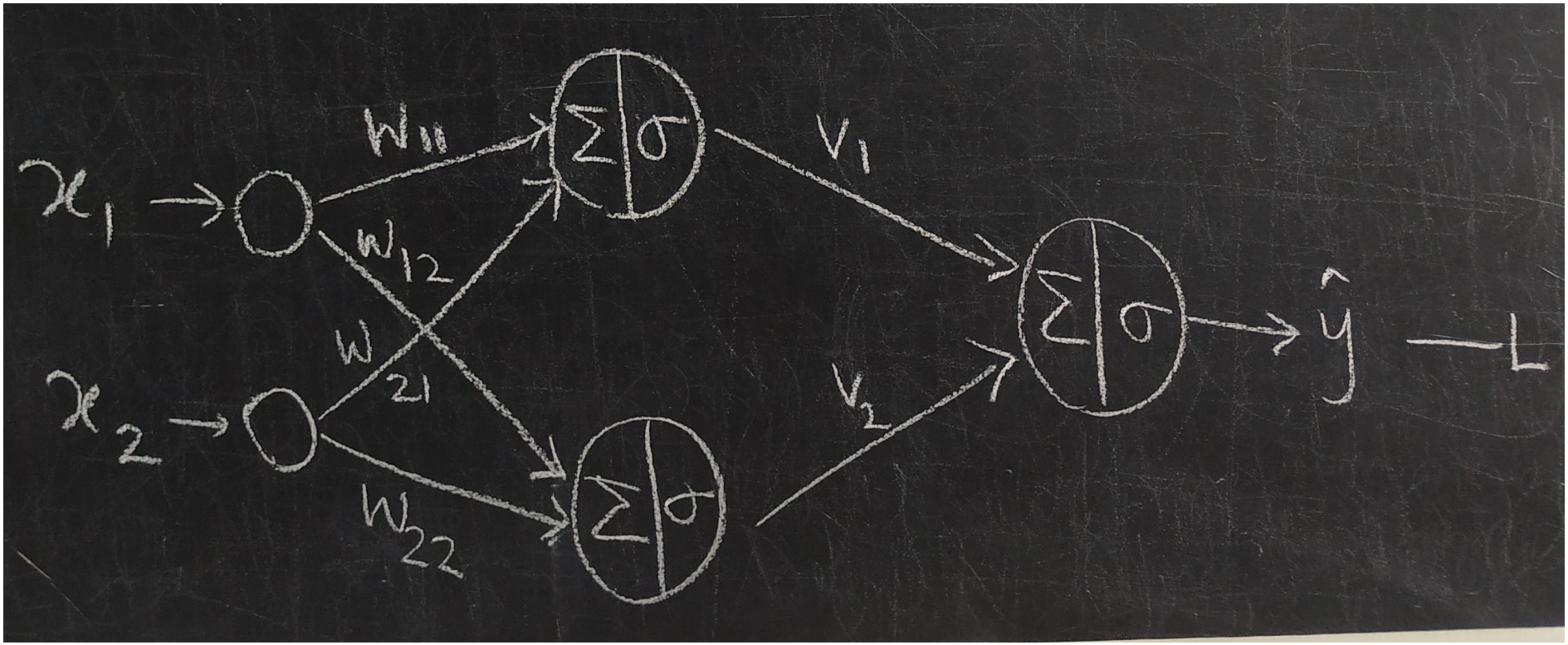
Solution

Represent the features in a transformed feature space on which learning the model will become easier:

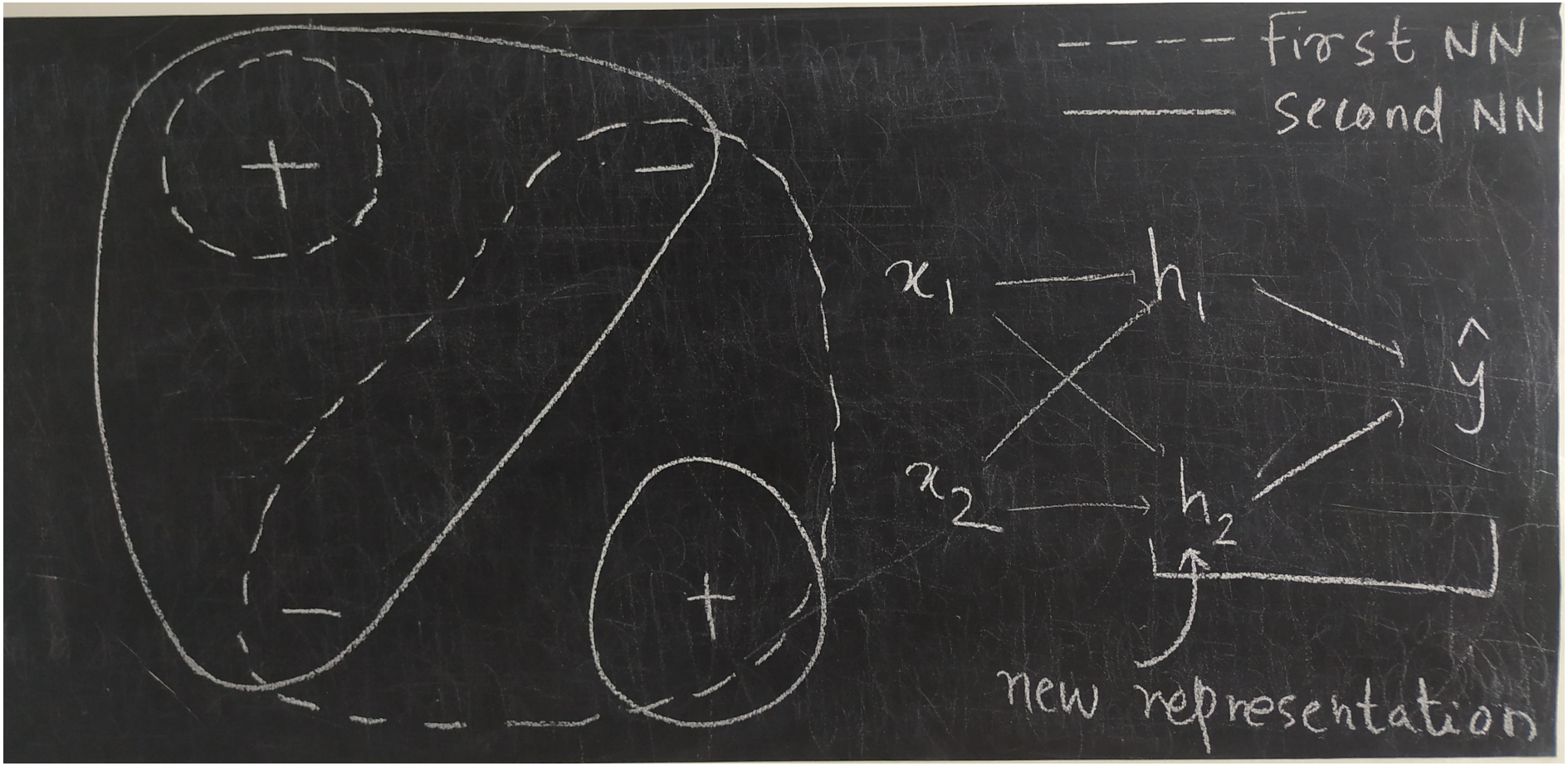
We will be restricting our study in this course to a multilayered neural network that achieves this: Multilayer Perceptron

Multilayer NNs I

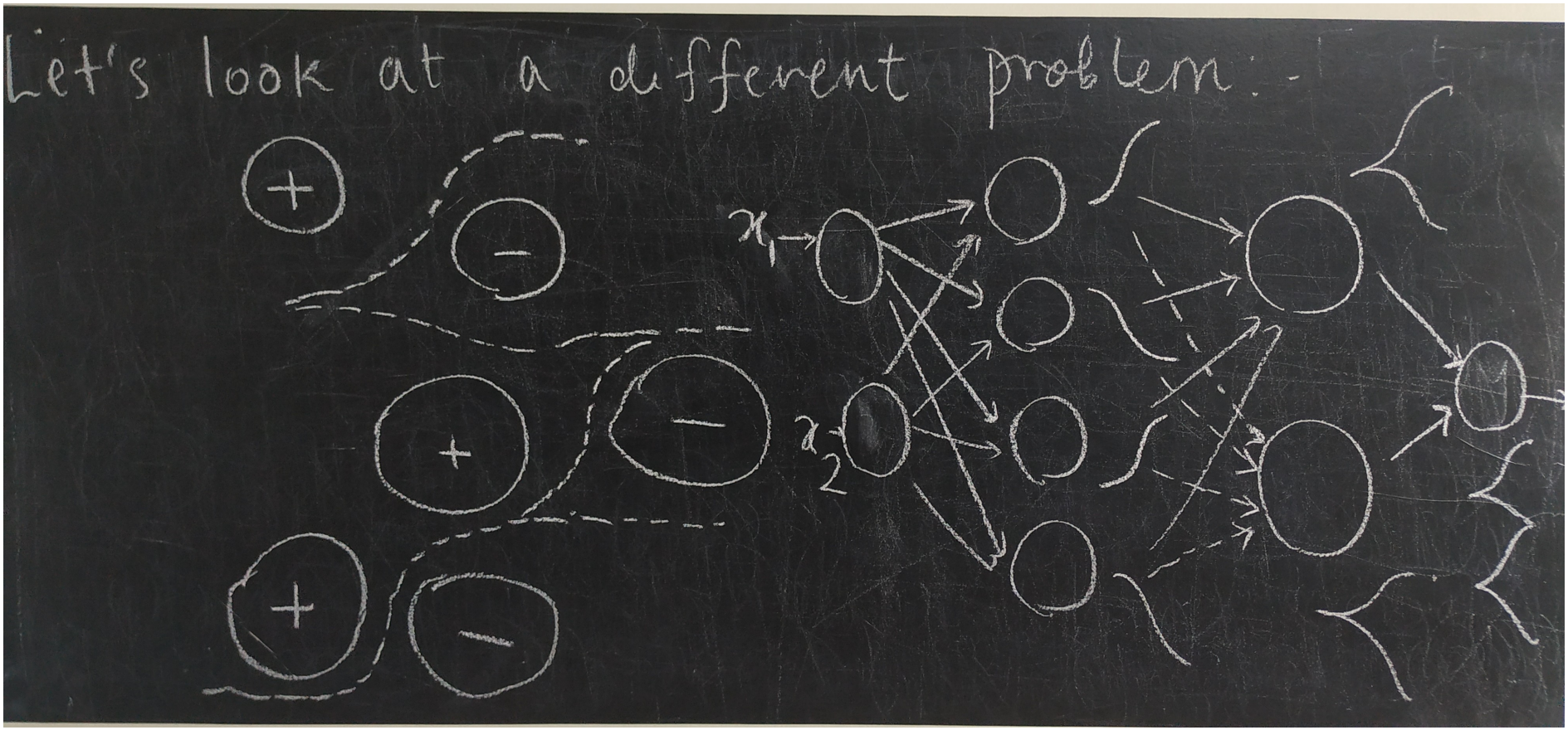
Muliple levels of transformation of features using layers:



Multilayer NNs II



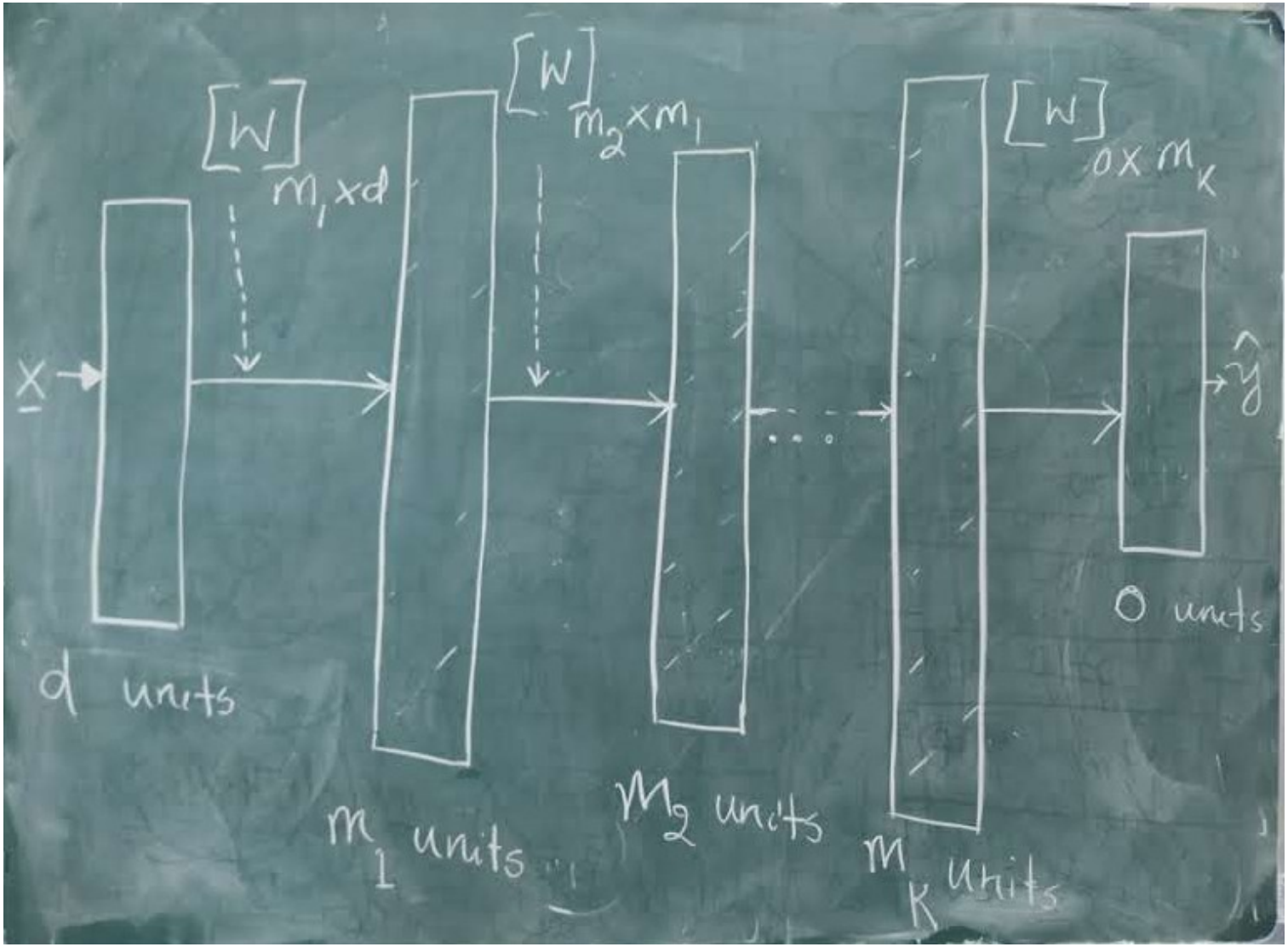
Multilayer NNs III



Multilayer NNs IV

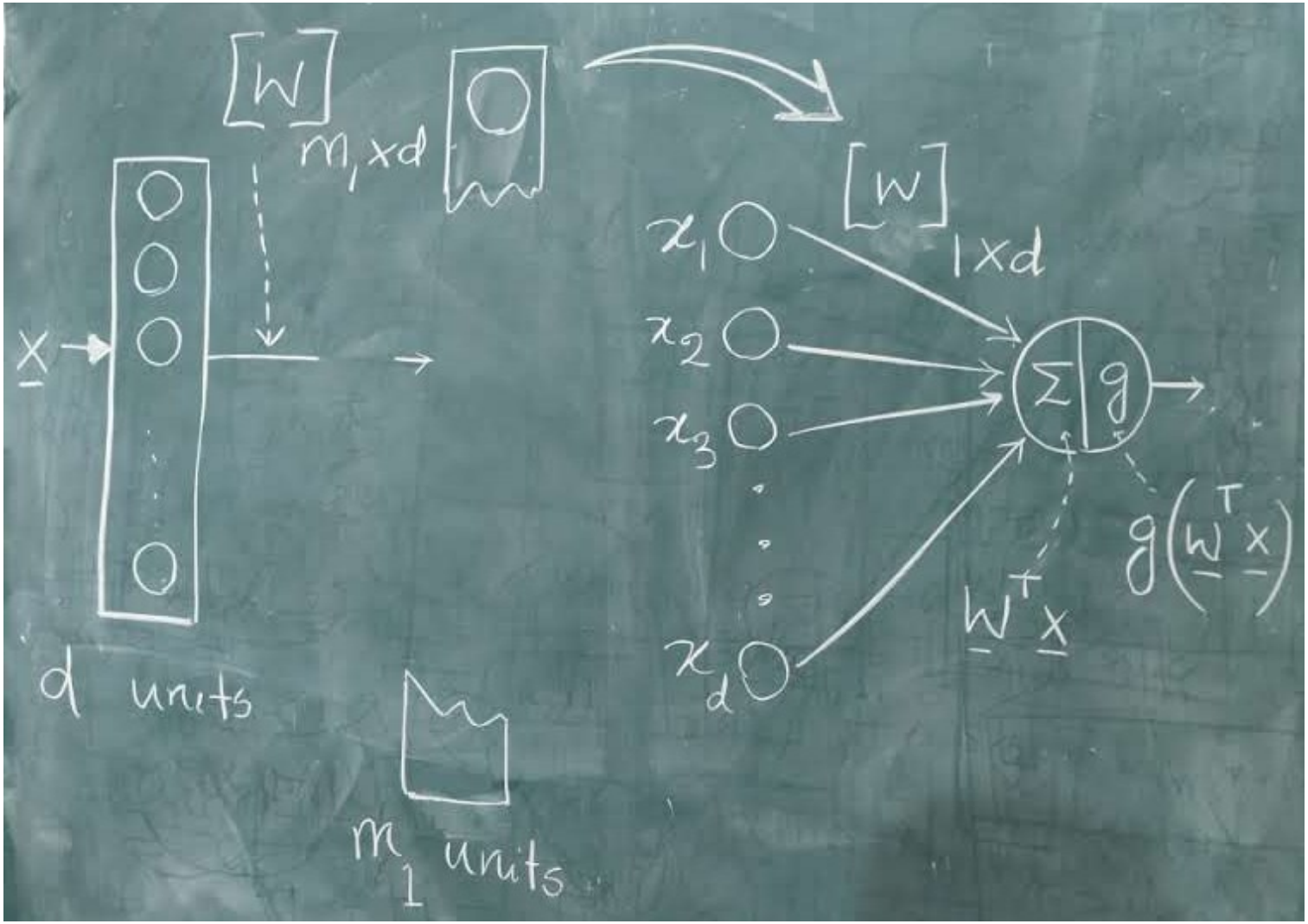
A neural network that is based on the this idea is 'Multilayer Perceptron (MLP)'.

Multilayer NNs V



Multilayer NNs VI

Visualising a single neuron in a hidden layer:



Multilayer NNs VII

Some difficulties concerning the structure of an MLP:

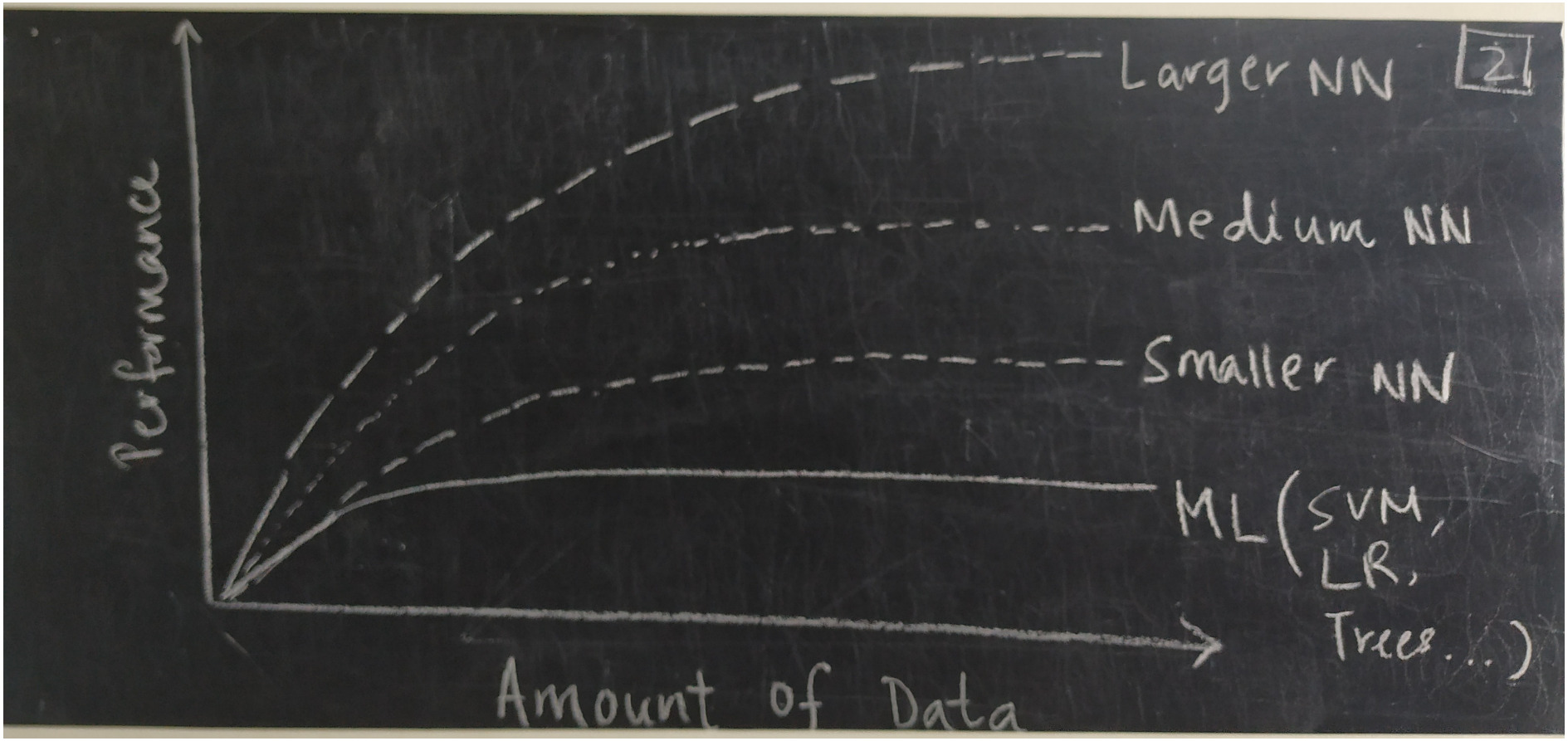
1. How many hidden layers for a given problem?
2. How many hidden units per layer?
3. What activation function to choose?

Multilayer NNs VIII

Hidden layers and units:

1. The hidden layers behave like feature detectors.
2. Early layers detect low-level features, and later layers detect high-level features.
3. Both these parameters can be considered as hyperparameters and they can be tuned using cross-validation method.
4. These parameters influence the performance of the model.

Multilayer NNs IX



Multilayer NNs X

Activation function:

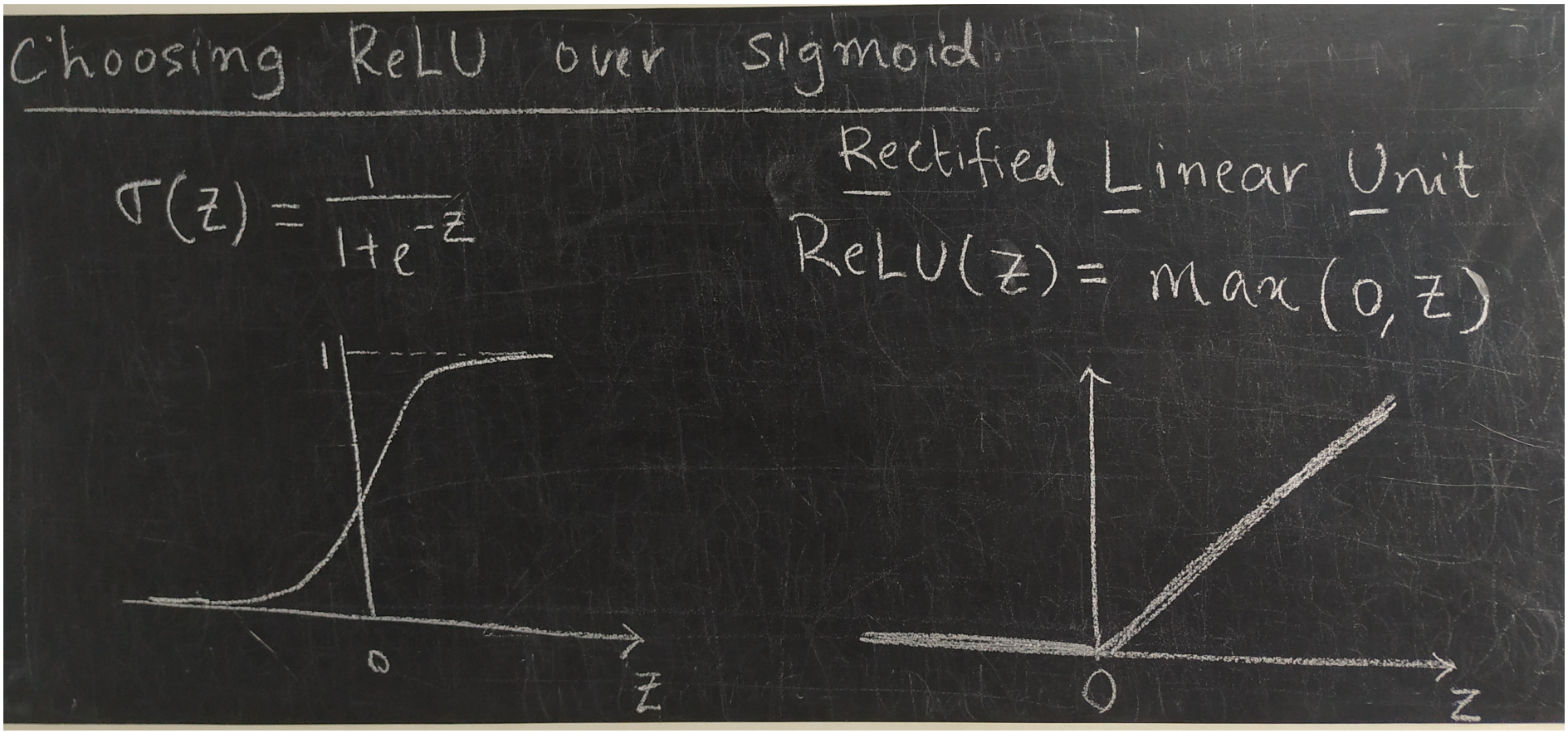
- No activation for inputs
- Output activation depends on the target domain:

Problem	# of output units	$g(\cdot)$
$y \in \mathbb{R}$	1	linear
$y \in \{0, 1\}$	1	sigmoid
$y \in \{0, 1, \ldots, c\}$	c	softmax

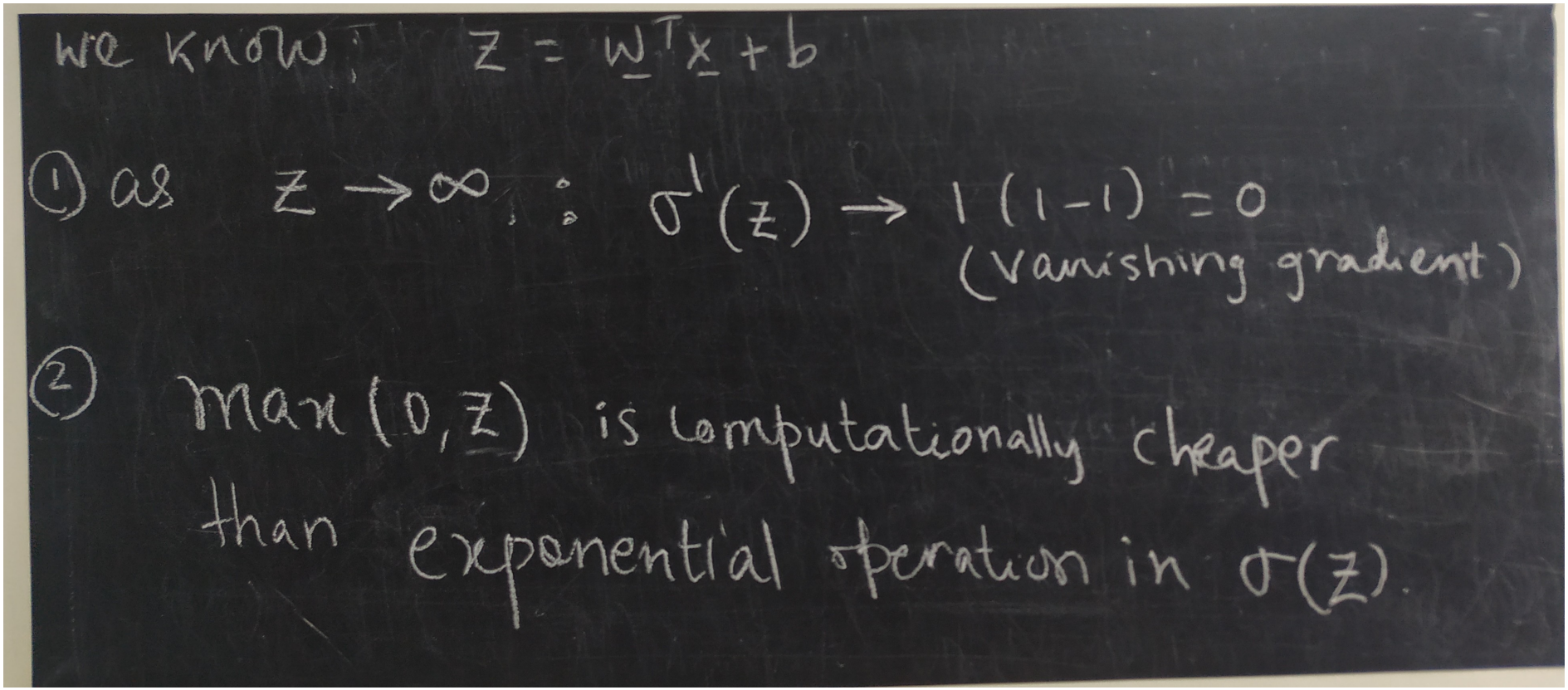
- Hidden unit activations are non-linear:
  - sigmoid
  - tanh
  - ReLU and its variants

Multilayer NNs XI

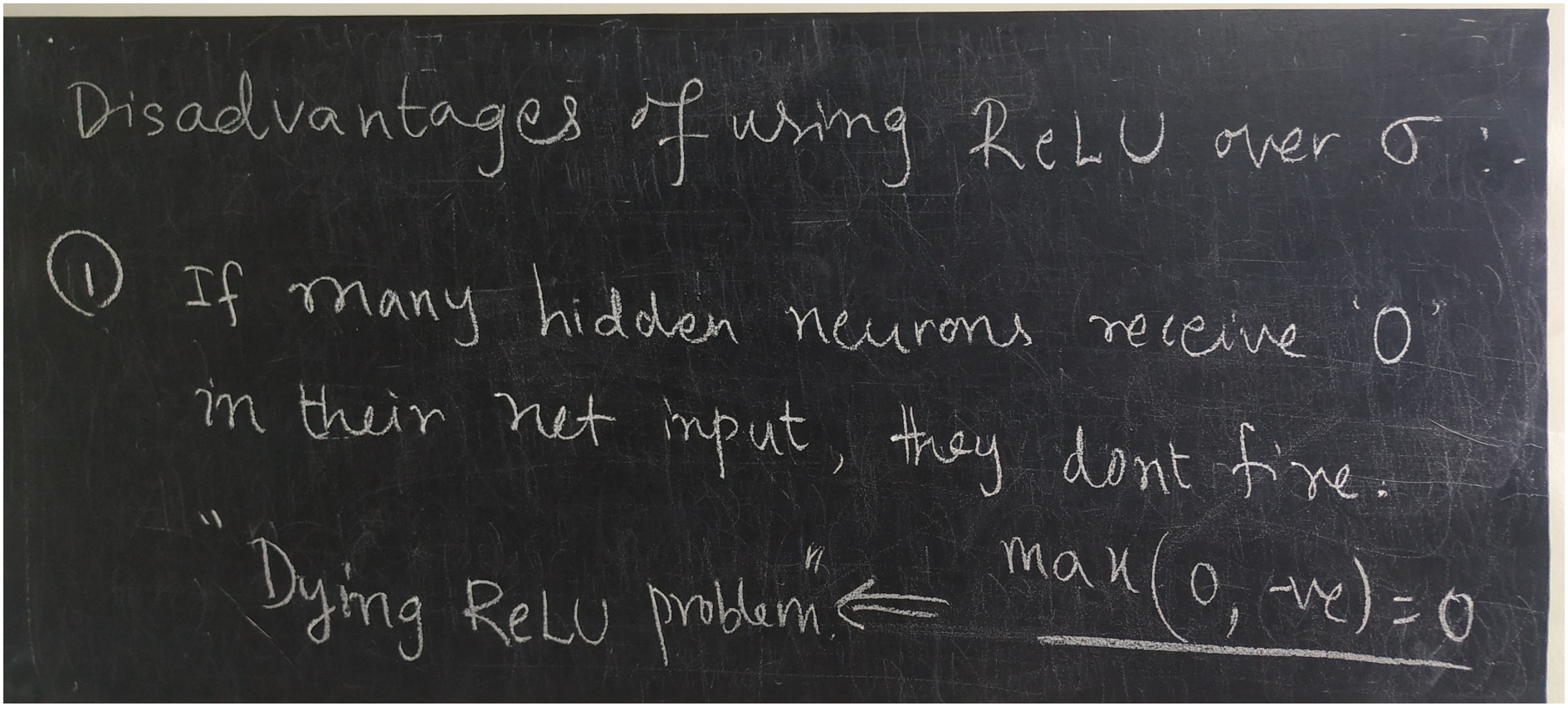
ReLU activation:



Multilayer NNs XII



Multilayer NNs XIII



Multilayer NNs XIV

Two variants of ReLU function:

- Leaky ReLU:

$$\text{LeakyReLU}(z) = \max(0.01z, z)$$

- Parametric ReLU:

$$\text{pReLU}(z) = \max(\alpha z, z)$$

  The parameter α is learned using an optimiser.

Multilayer NNs XV

Weights:

The synaptic weights (ws) for each layer of an MLP are optimised using gradient descent. The procedure is popularly known as 'Backpropagation'.

Mathematics behind 'BackPropagation' is beyond the scope of this course. However, we will do practical problem solving by implementing a neural network using any popular Python library such as Scikit-learn or Tensorflow-Keras or PyTorch.