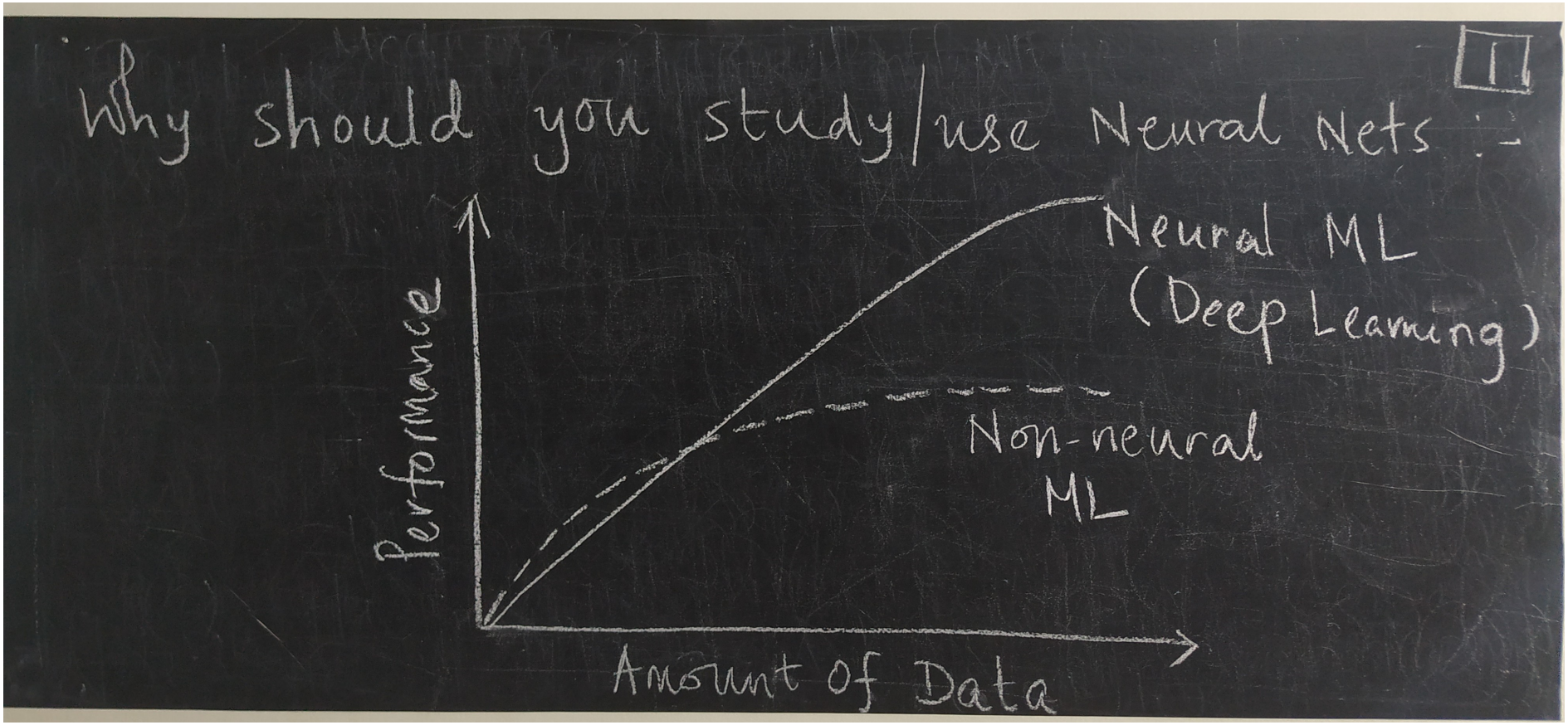
NN Basics Tutorial 1

Tirtharaj Dash

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

August 31, 2021

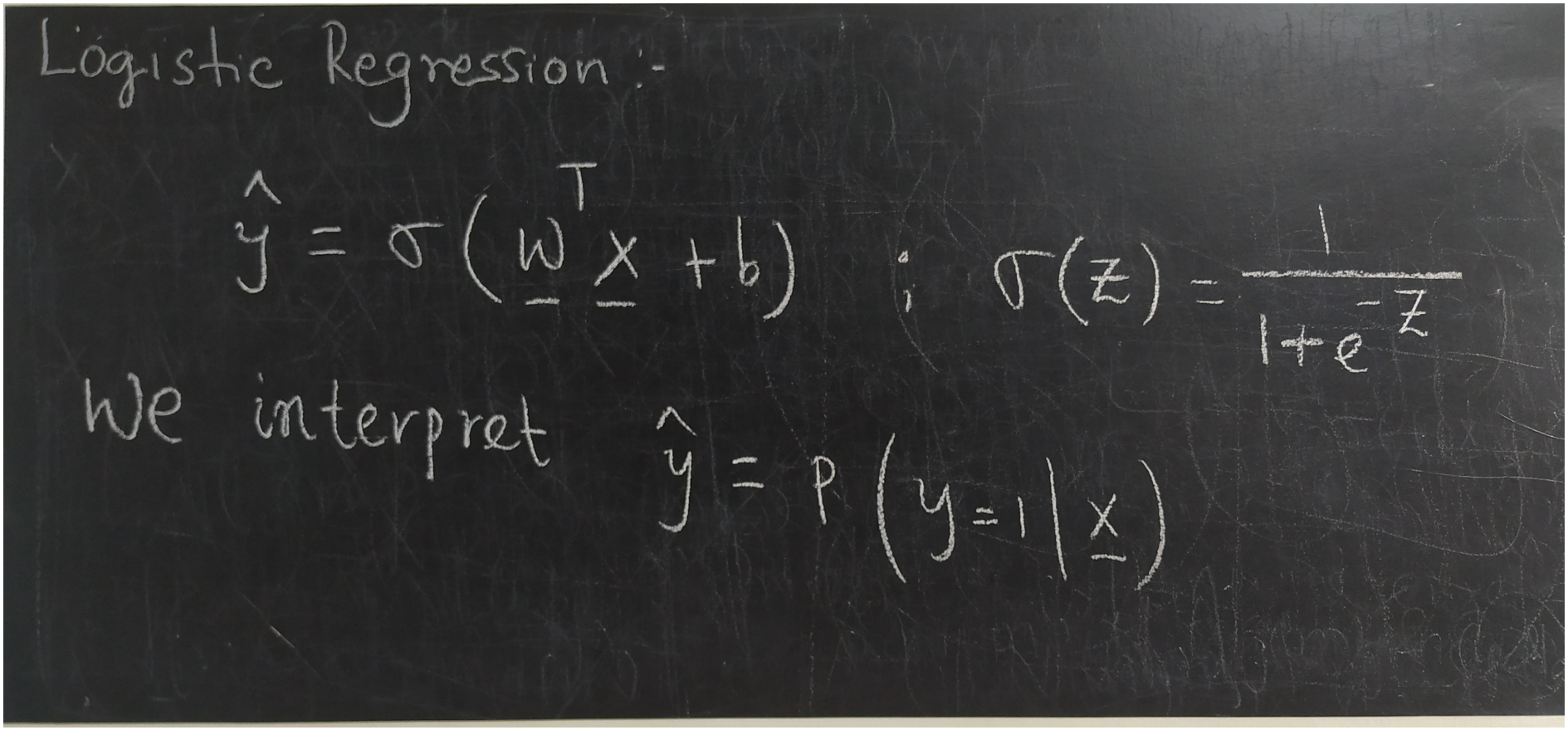
Why NN/DL?


Why should you study/use Neural Nets :-
Performance
Neural ML
(Deep Learning)
Non-neural
ML
Amount of Data

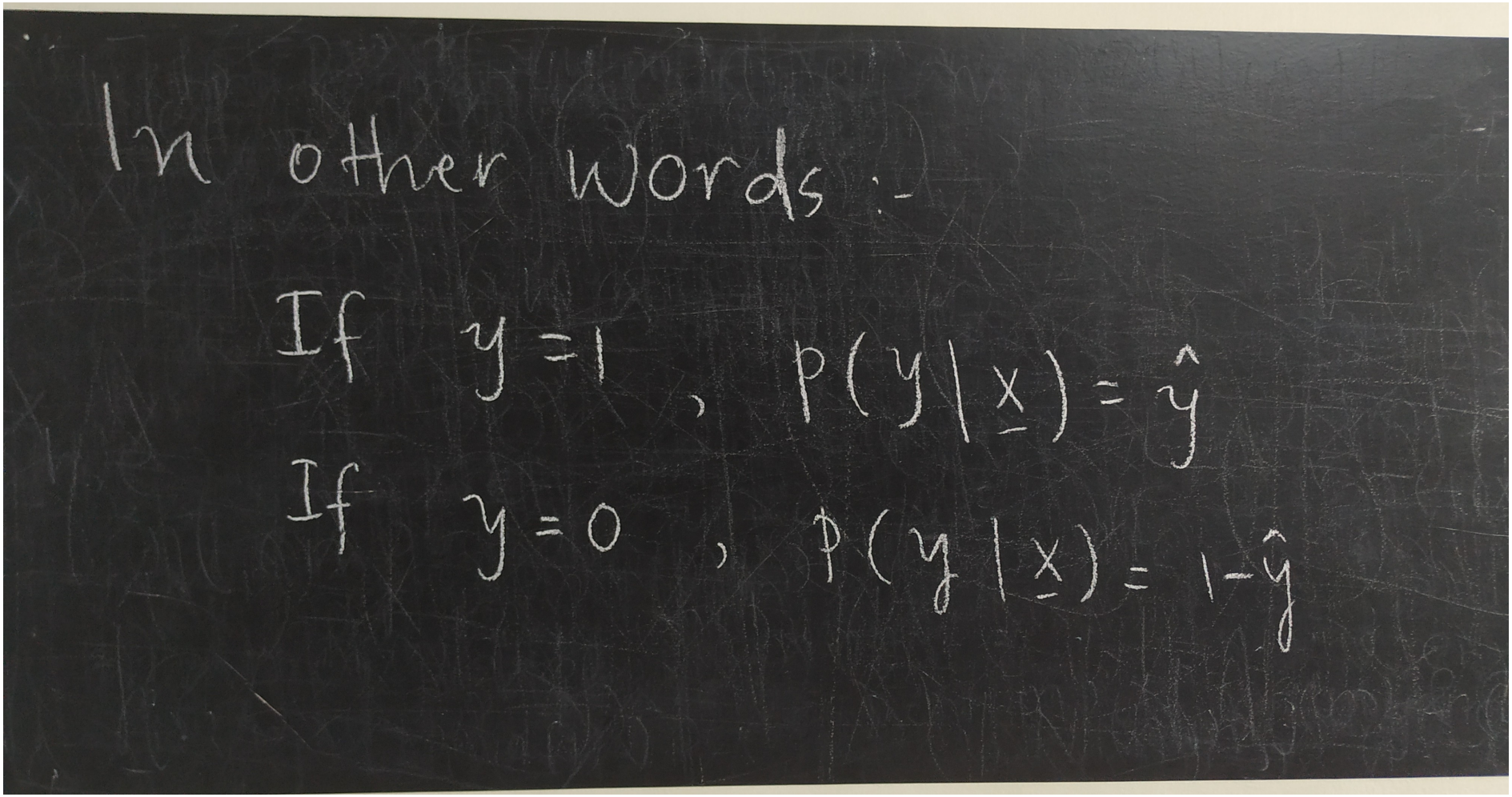
In this tutorial series:

- Logistic Regression as a NN (Perceptron)
- Issues with perceptron
- Kernel trick
- Layered NNs (Multilayer Perceptron)
  - Backpropagation – (skipped to lecture)
- Activation functions

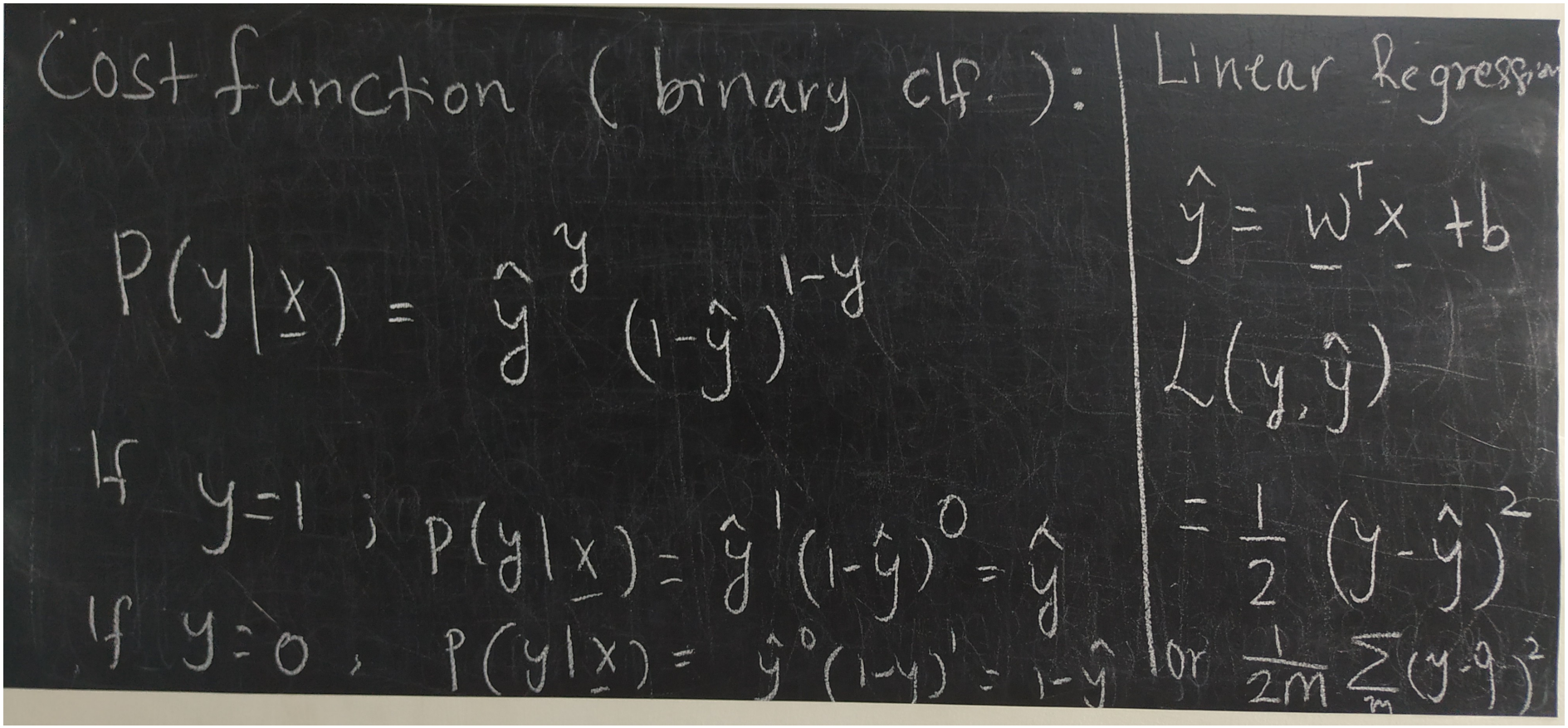
Logistic Regression as NN I



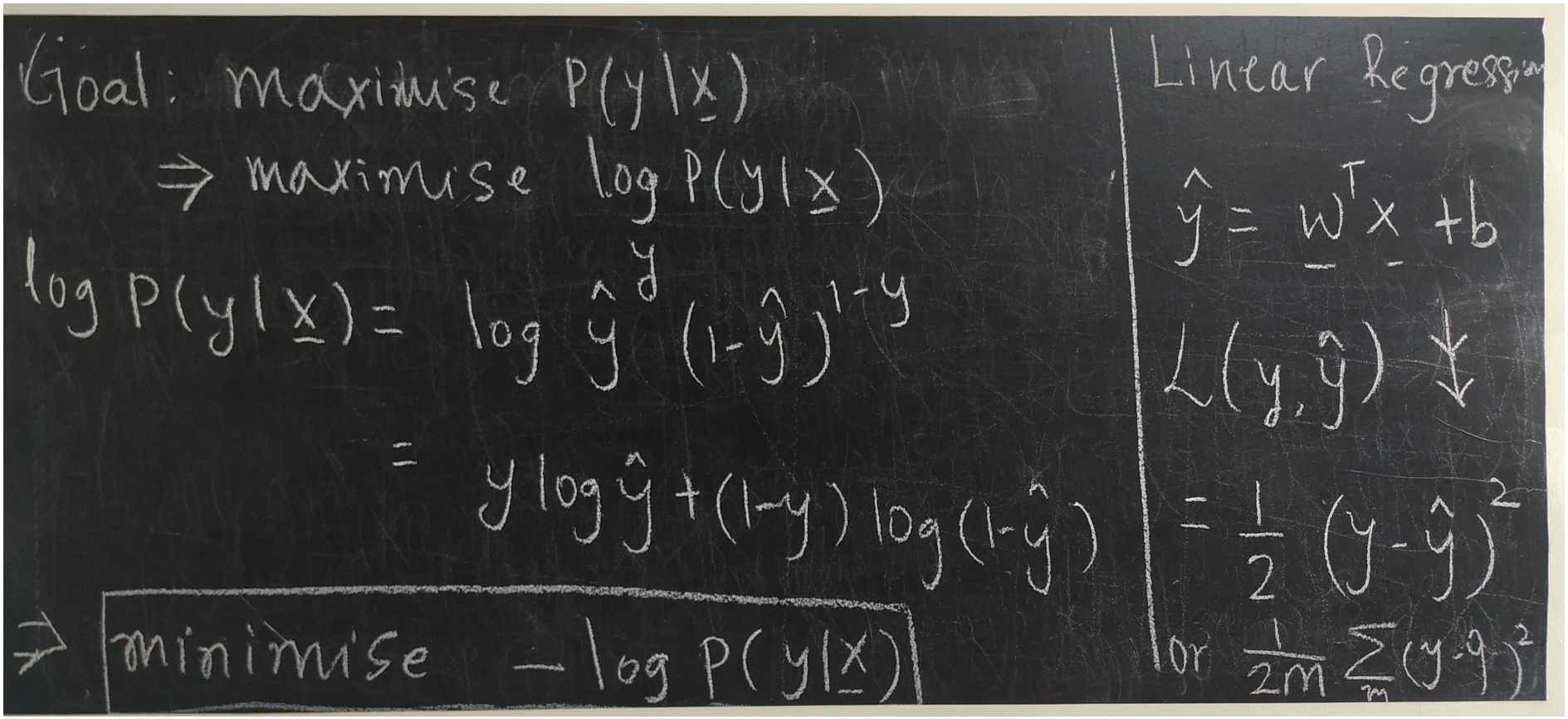
Logistic Regression as NN II



Logistic Regression as NN III



Logistic Regression as NN IV

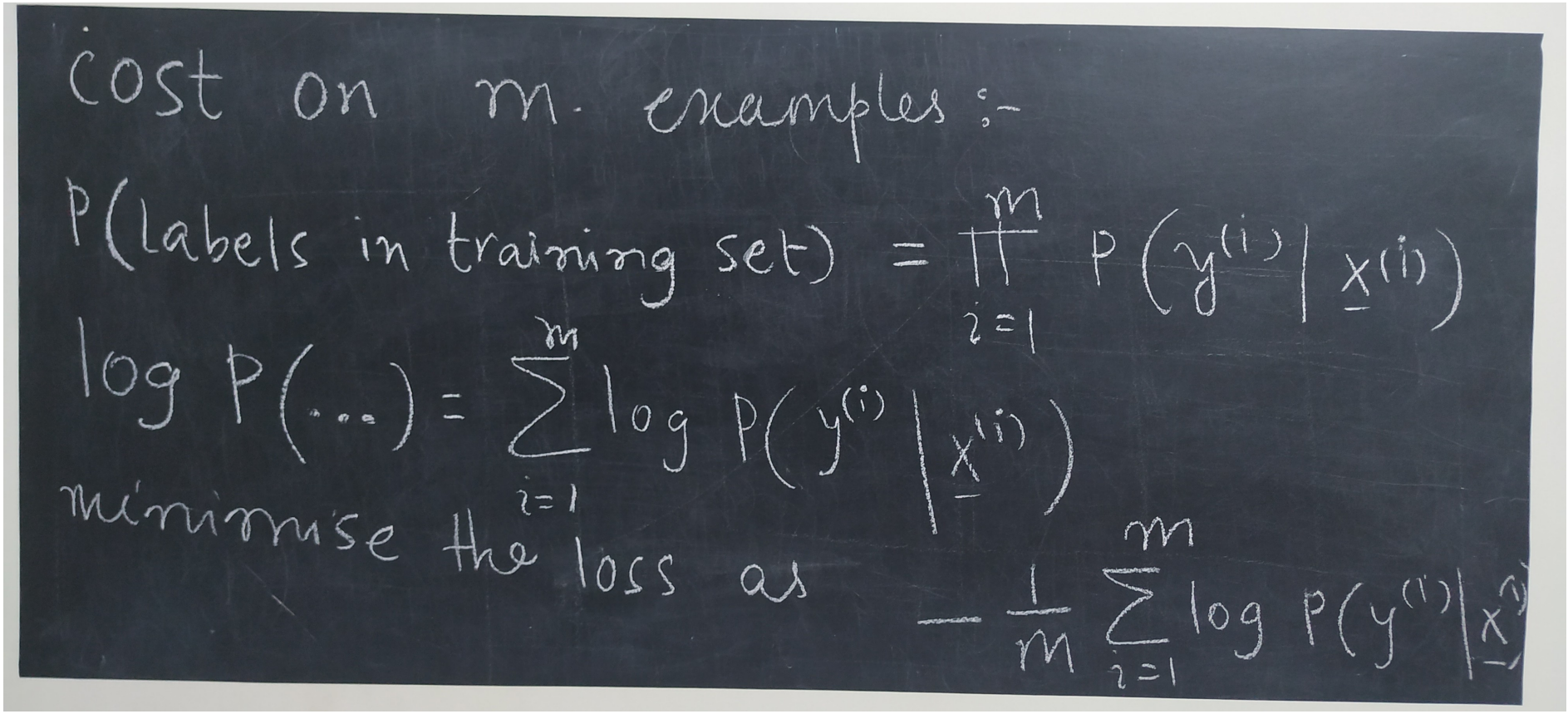


The cost function: $-\{y \log(\hat{y}) + (1-y) \log(1-\hat{y})\}$ is known as **cross-entropy loss function**.

Logistic Regression as NN V

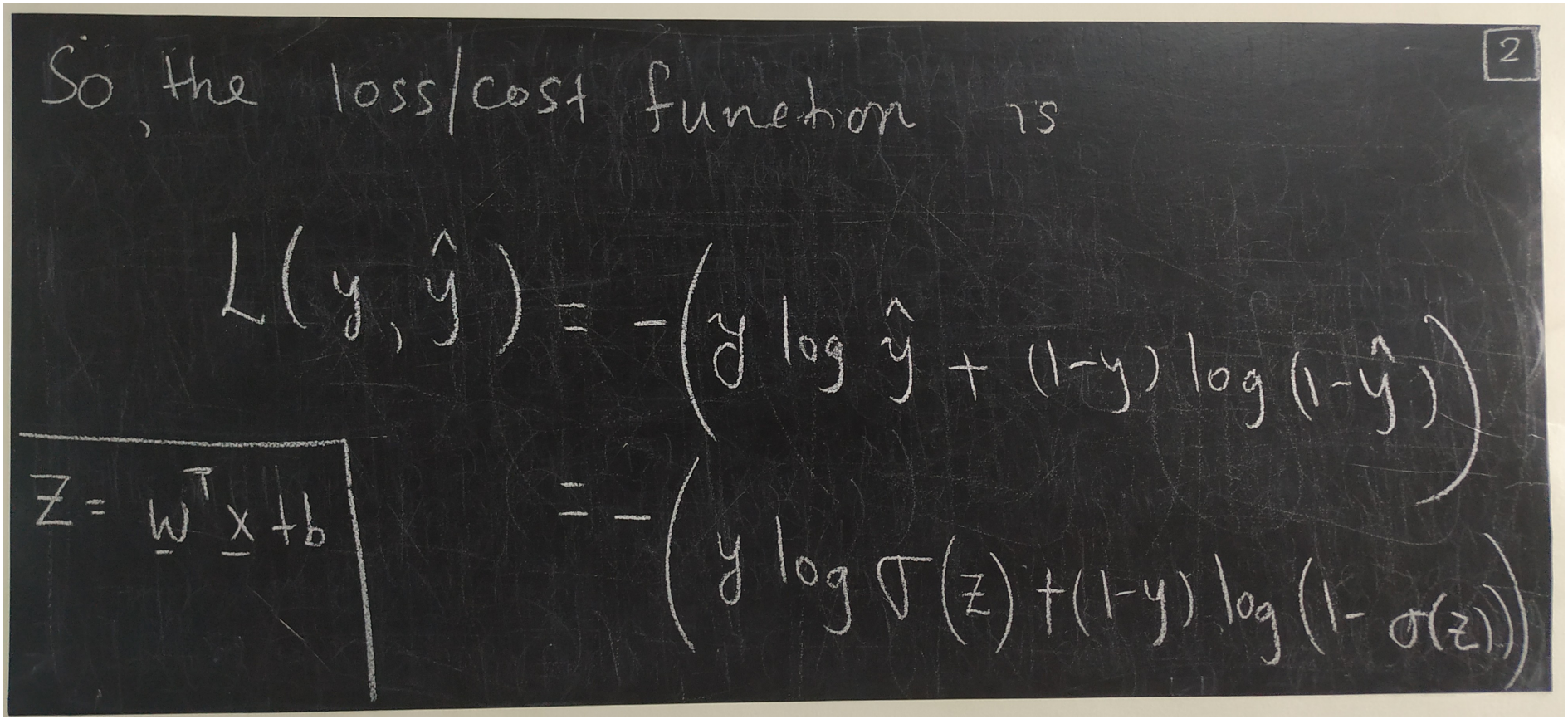
- Cross-entropy function computes the entropy between two different probability distribution.
- As $\hat{y} \in [0, 1]$, we can see that $(\hat{y}, 1 - \hat{y})$ is the probability distribution obtained from the logistic model, given any example $\mathbf{x}$ or $\underline{x}$.
- *Where is the other probabibility distribution?* We can treat the labels as a probability distribution. Any example $\mathbf{x}$ 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



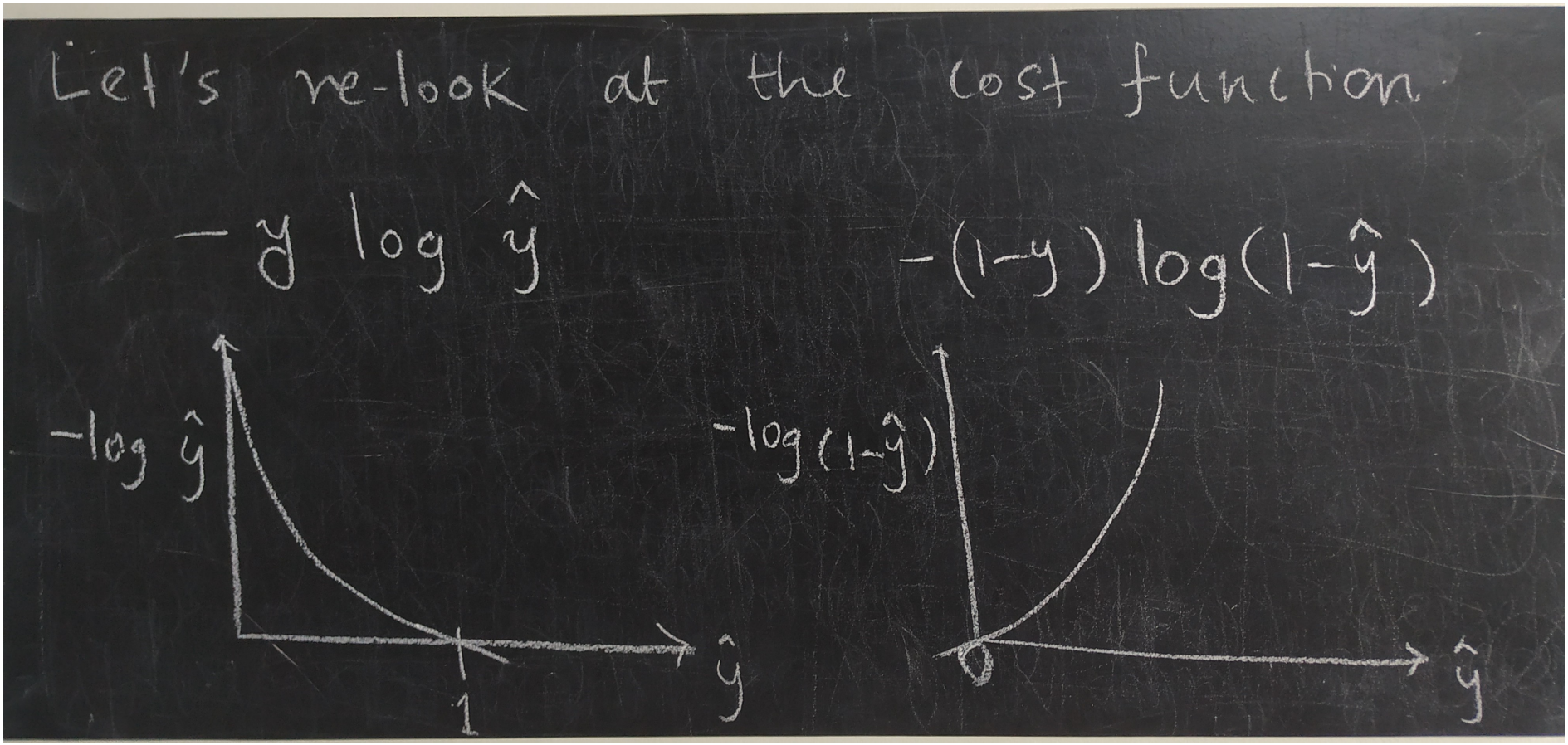
Logistic Regression as NN VII

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

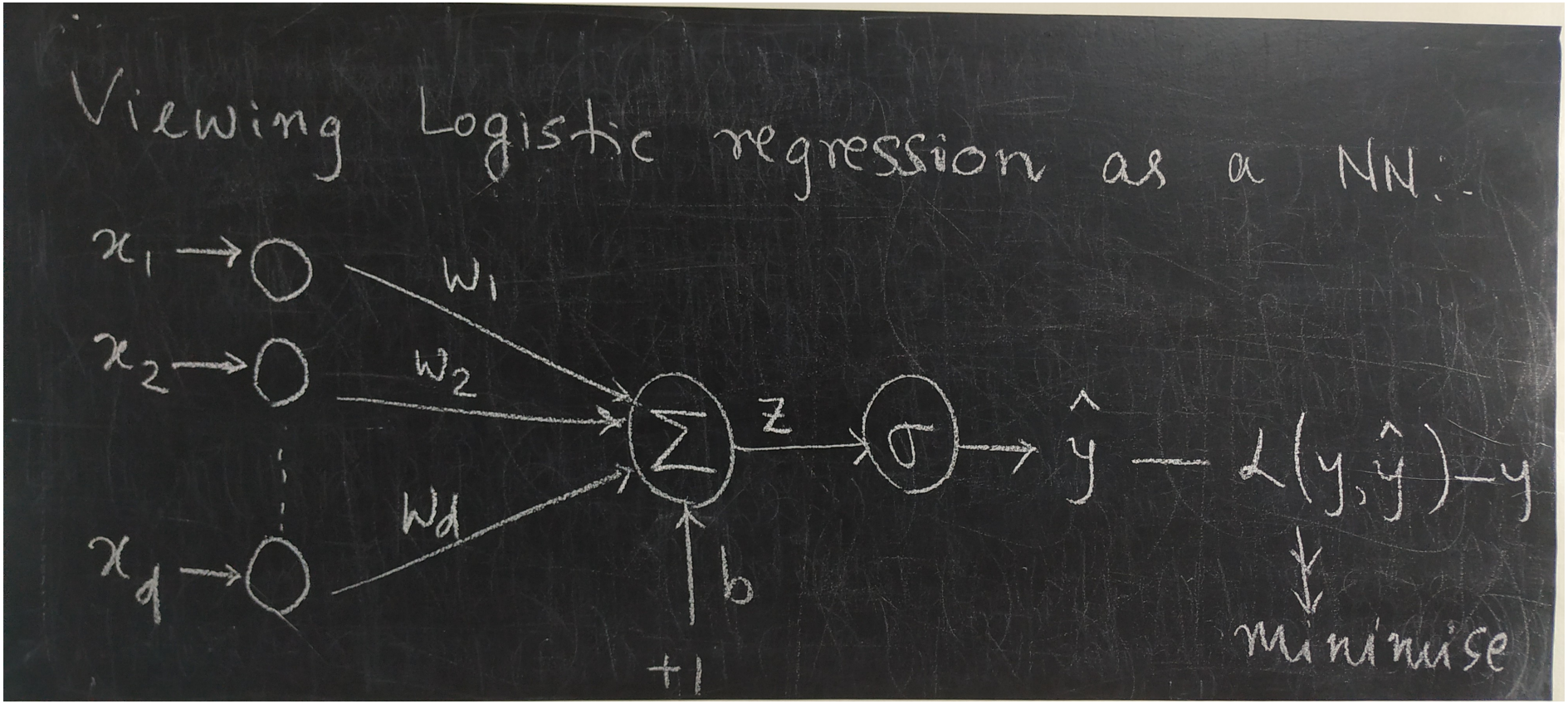


Logistic Regression as NN VIII

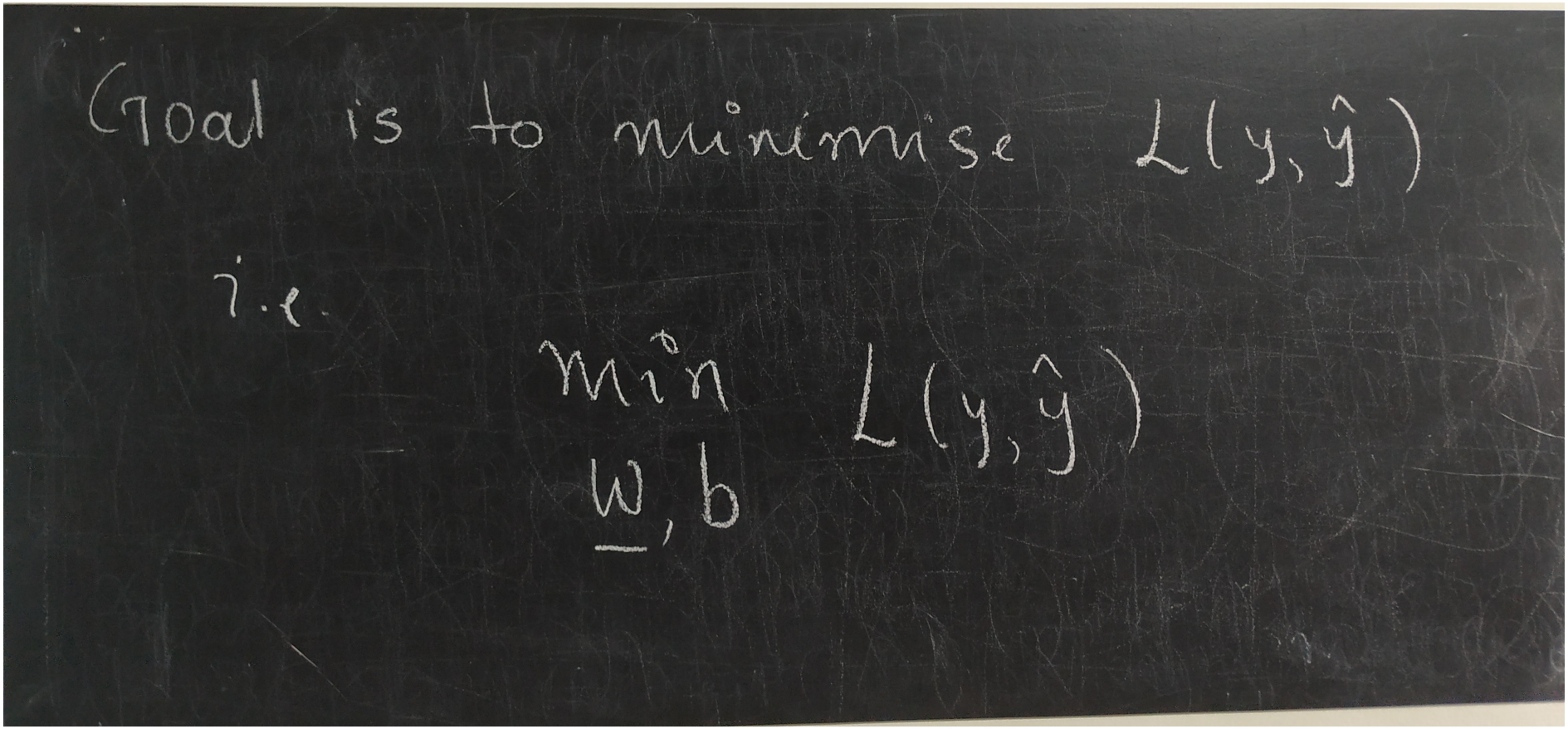
What do these two terms in the loss try to achieve?



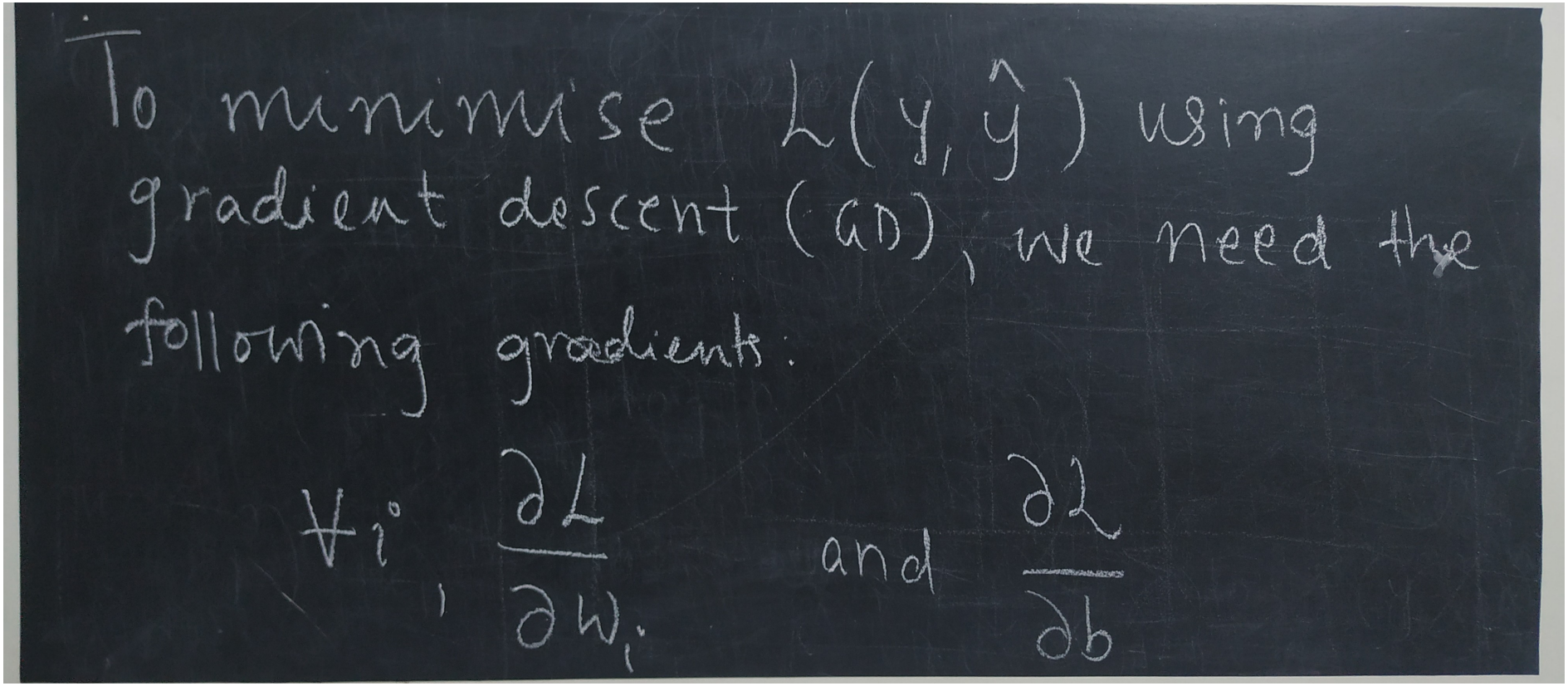
Logistic Regression as NN IX



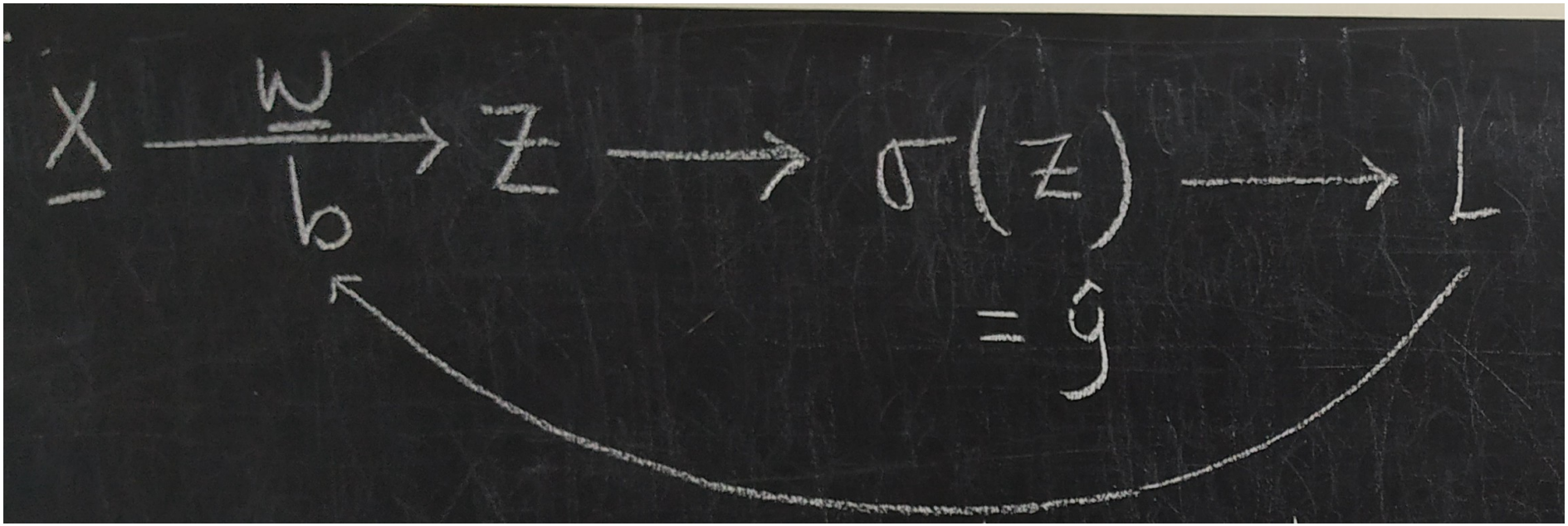
Logistic Regression as NN X



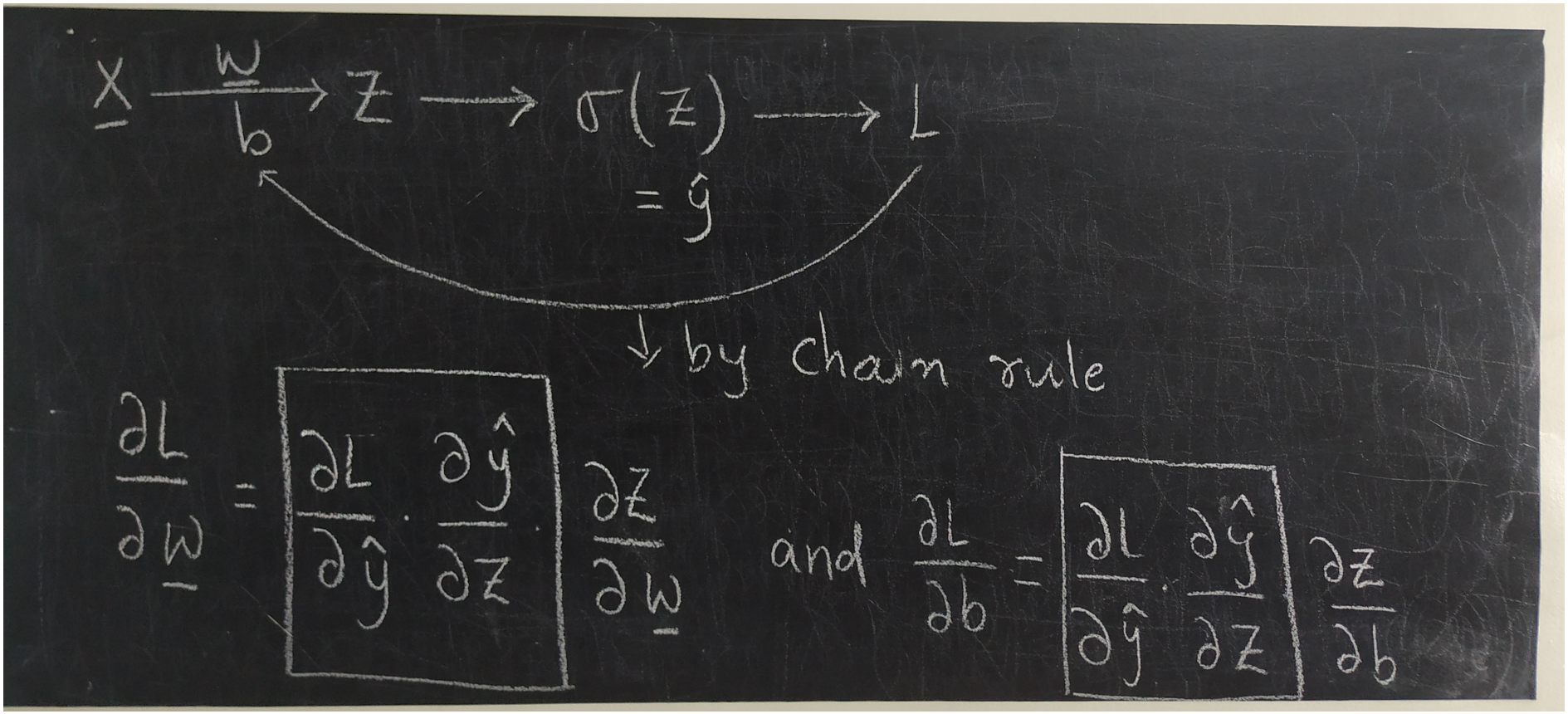
Logistic Regression as NN XI



Logistic Regression as NN XII



Logistic Regression as NN XIII



Logistic Regression as NN XIV

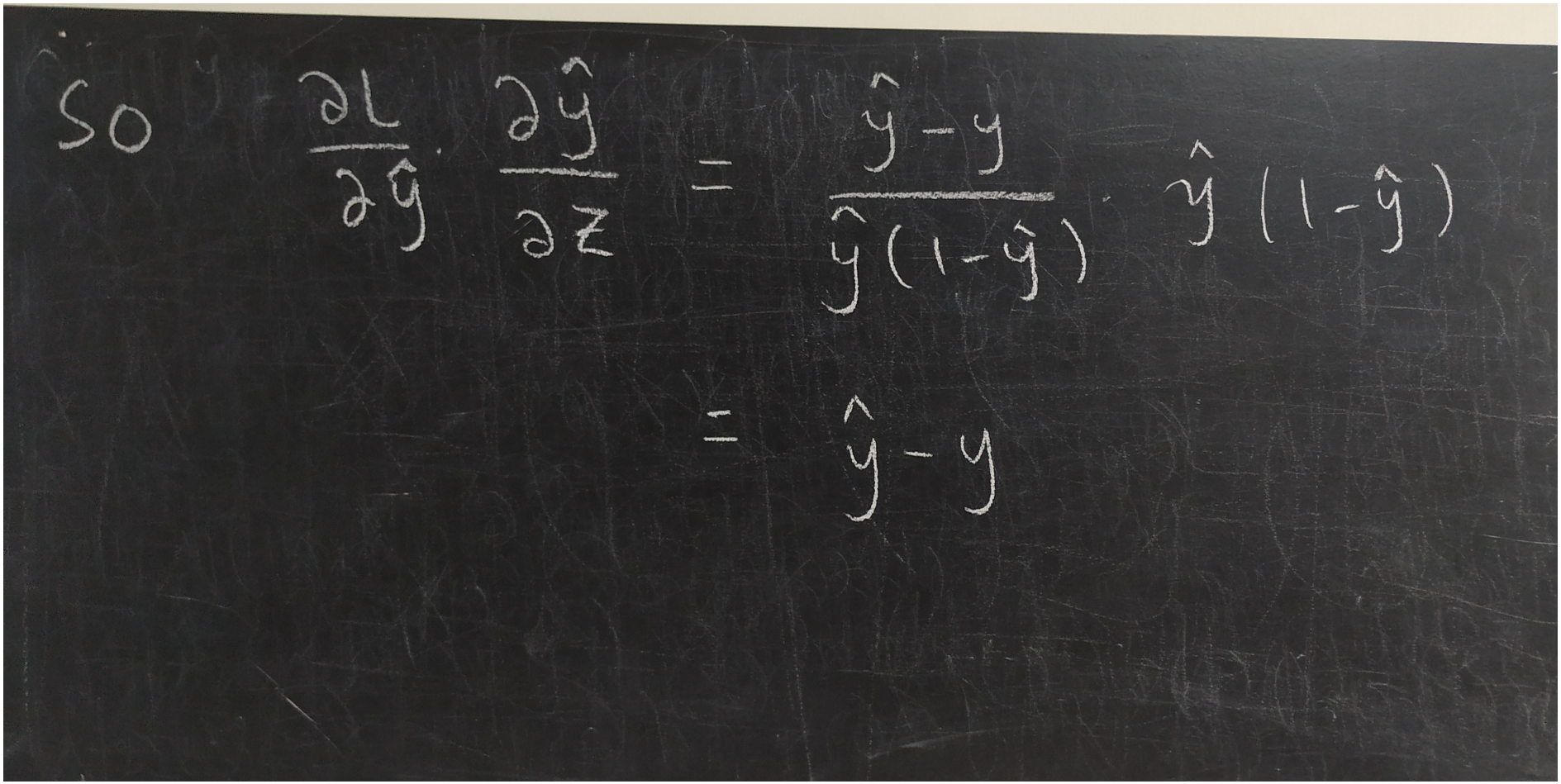
$$L = -y \log \hat{y} - (1-y) \log (1-\hat{y})$$

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

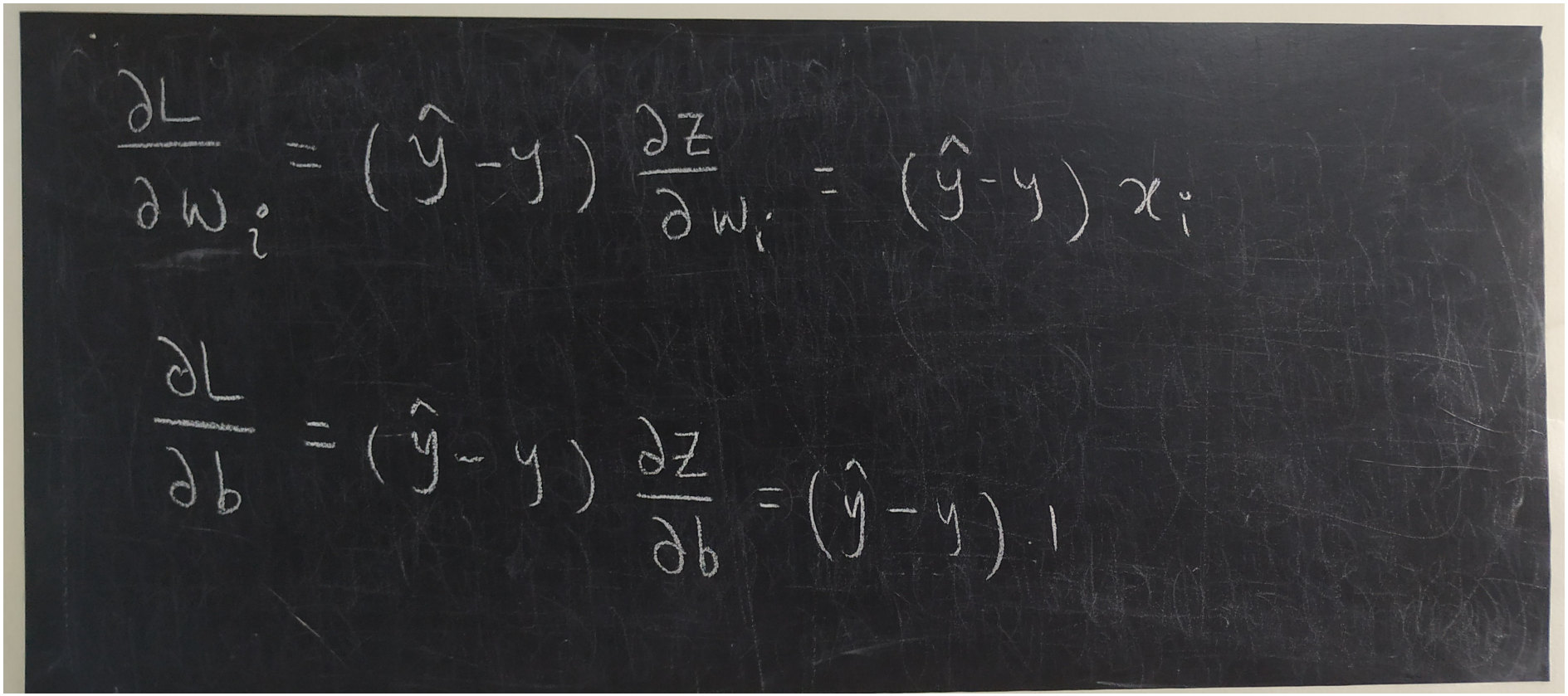
$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

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

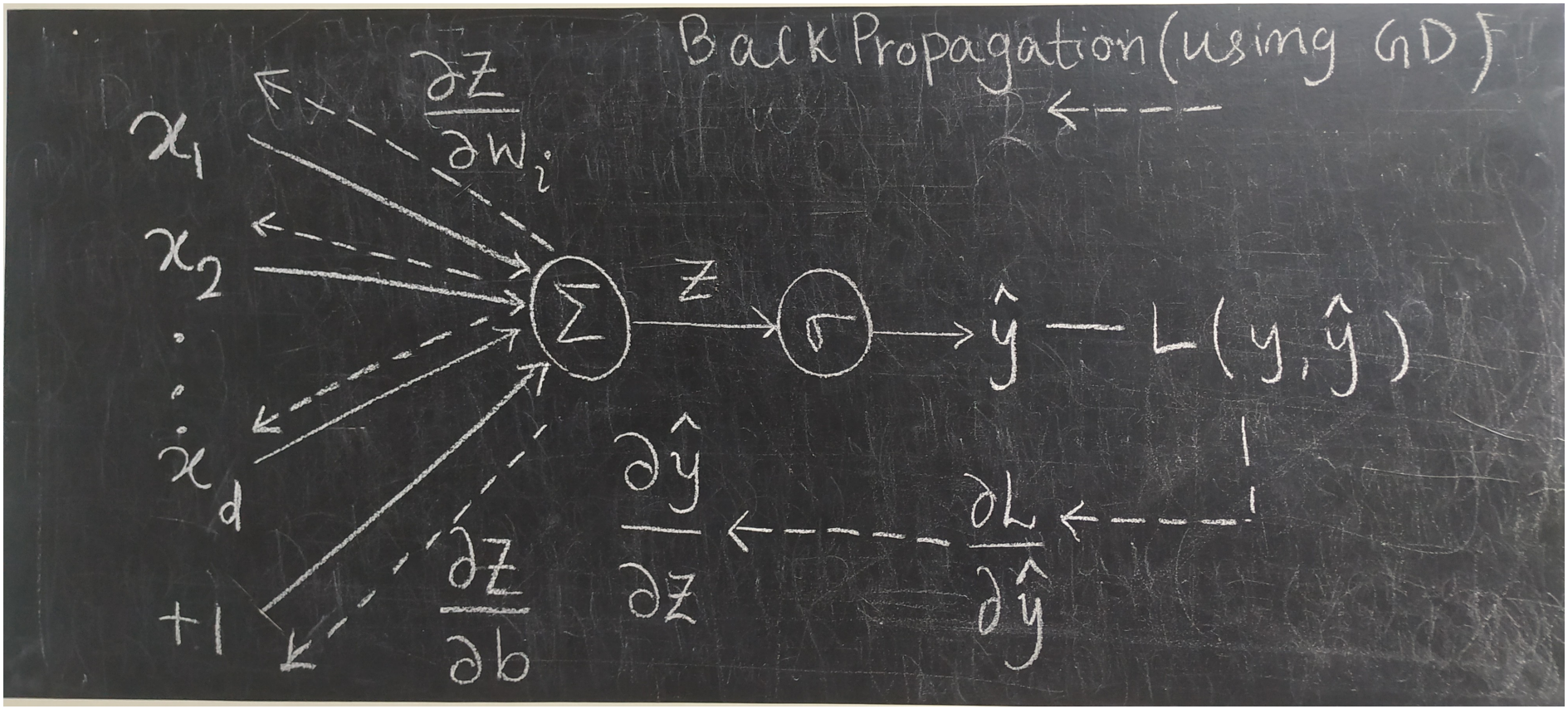
Logistic Regression as NN XV



Logistic Regression as NN XVI



Logistic Regression as NN XVII



Logistic Regression as NN XVIII

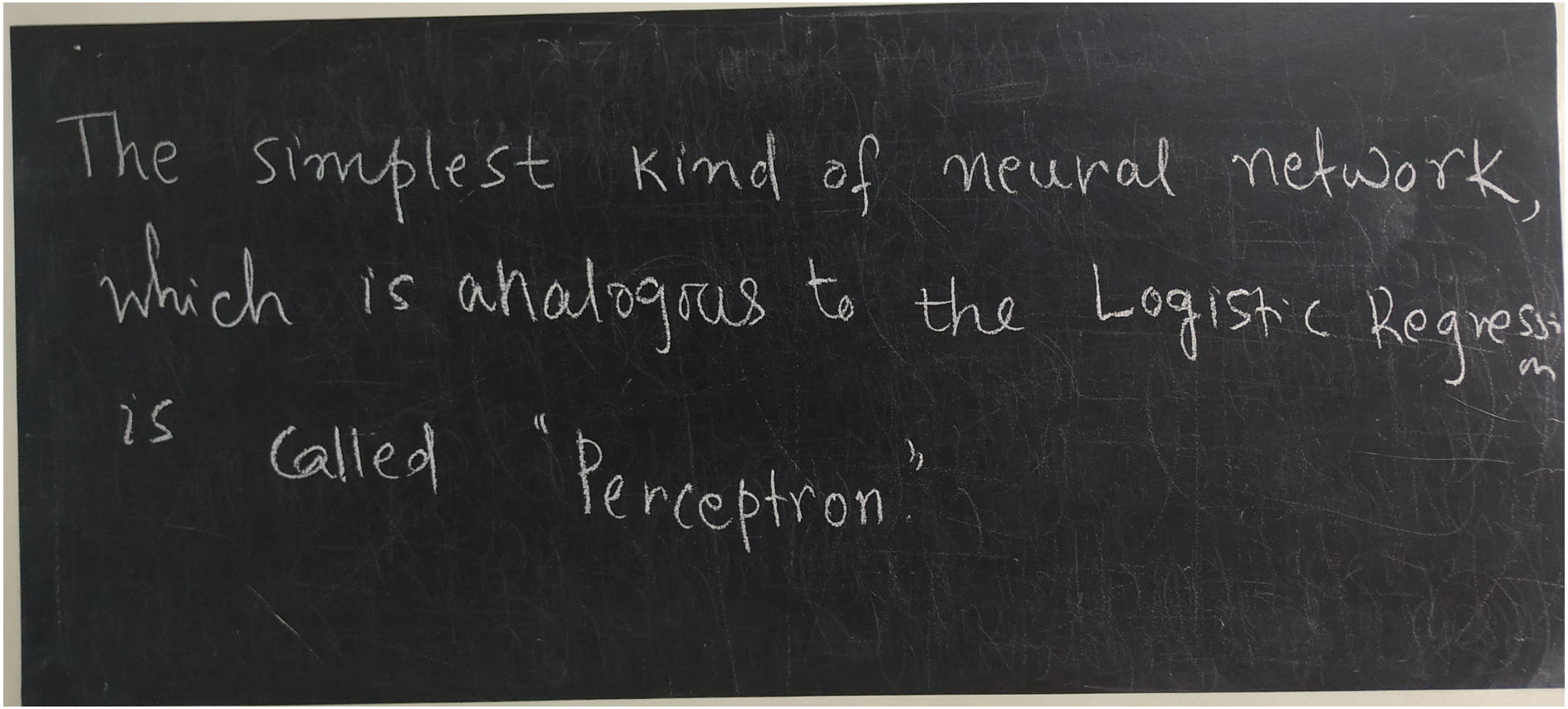
To update the parameters:

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

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

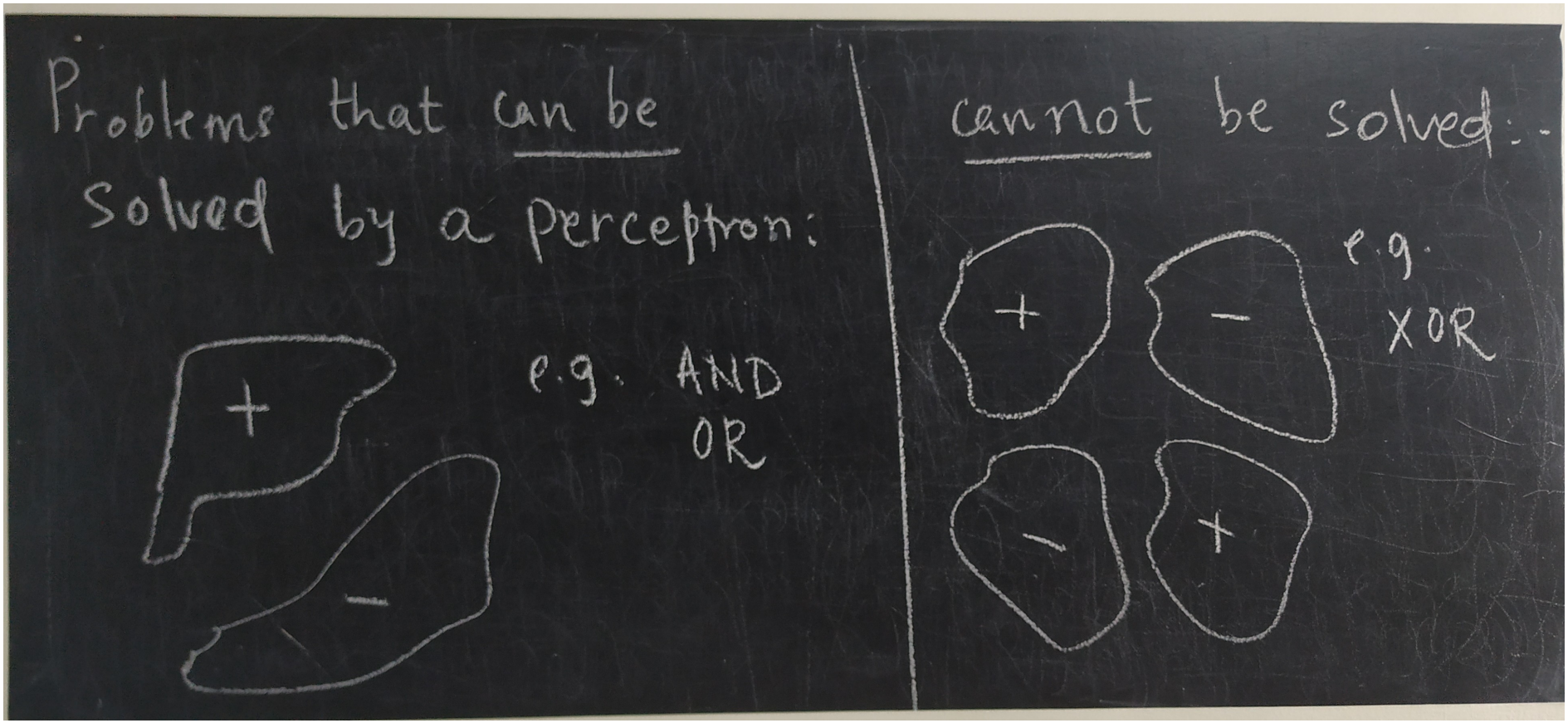
where, $\alpha \in (0, 1]$ is the learning rate.

Logistic Regression as NN XIX

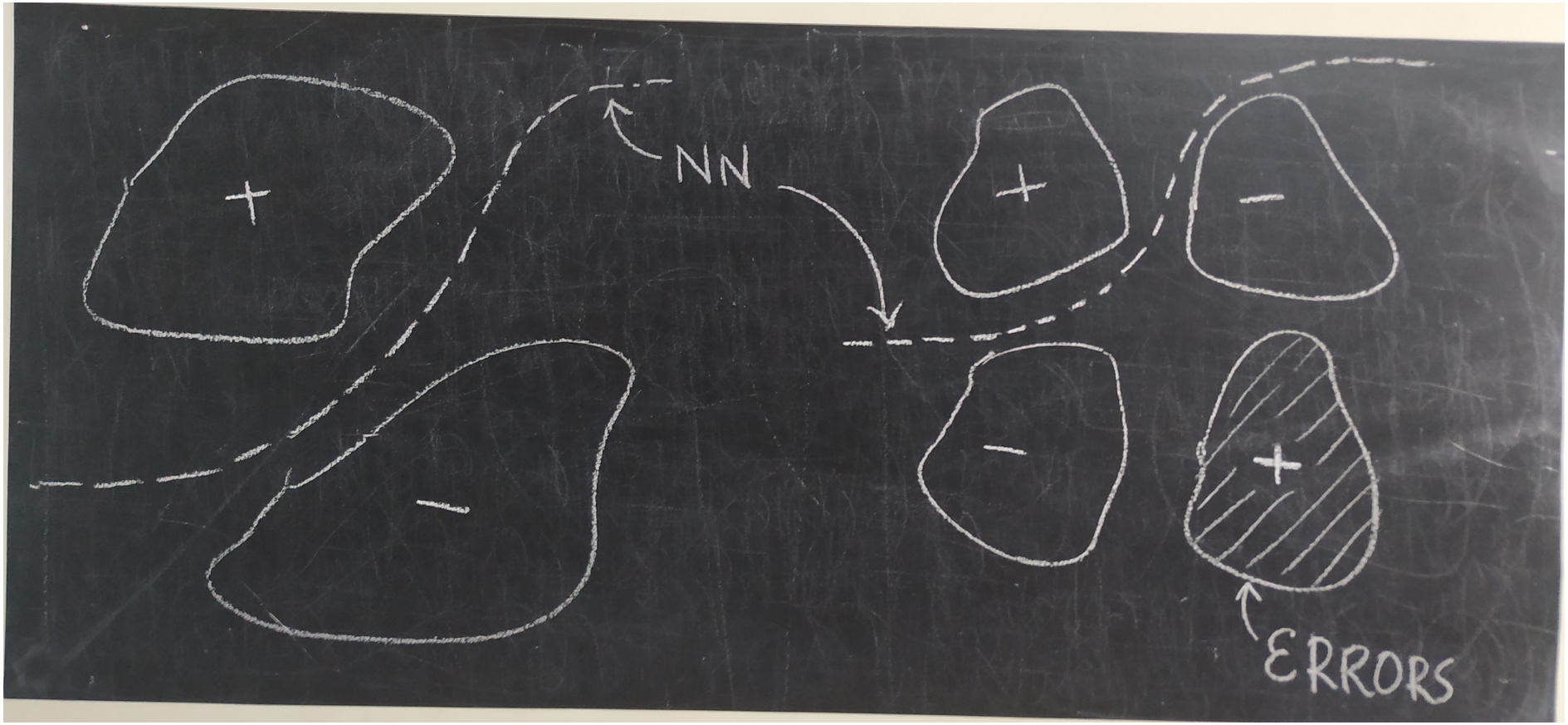


In the NN world, these parameters **w** and b are called **synaptic weights** or "simply" **weights**; z is called the **net input**. The function that operates on the net input in any neuron is called **activation function**.

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:

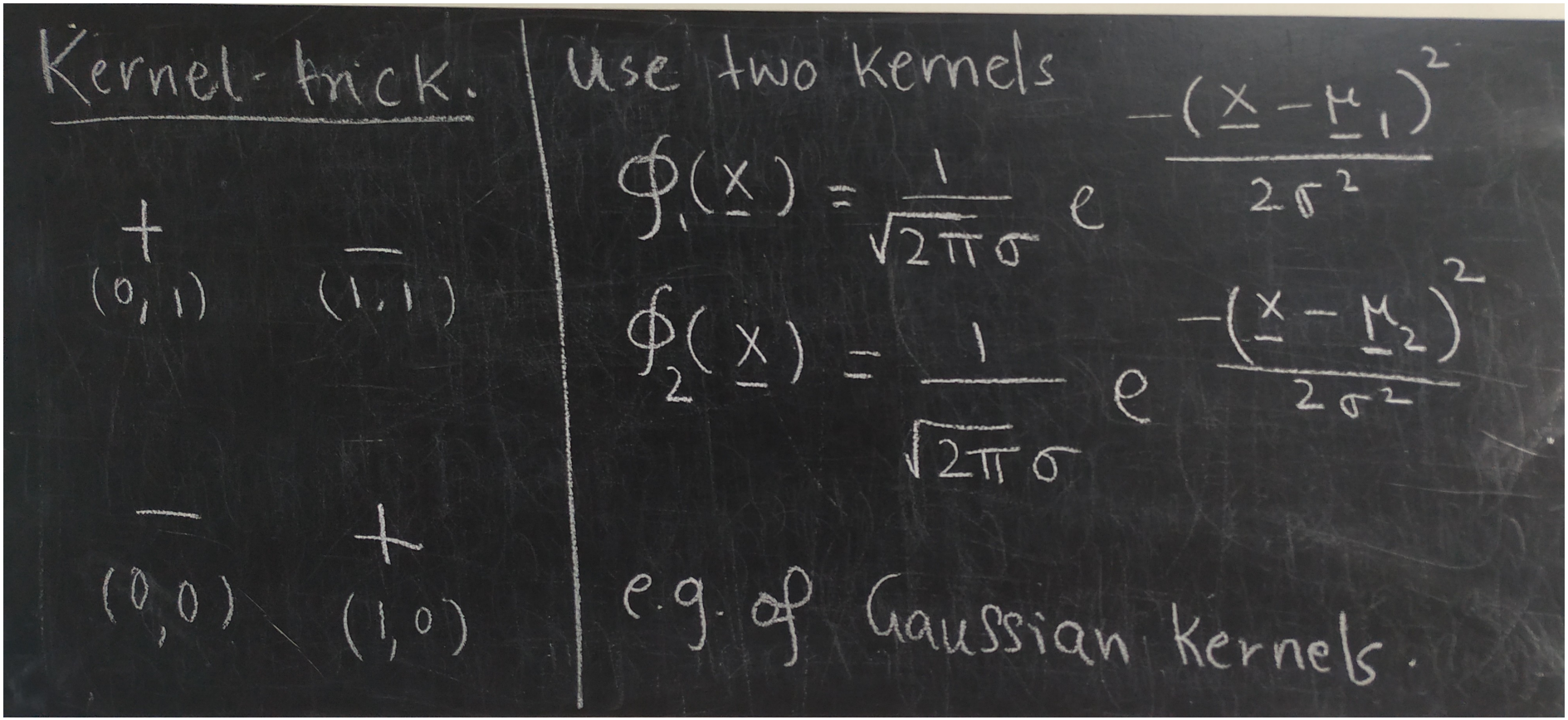
- Using kernel trick (for less-complex problem)
- By multiple layers of feature transformation (for hard problems)

This forms the basis for:

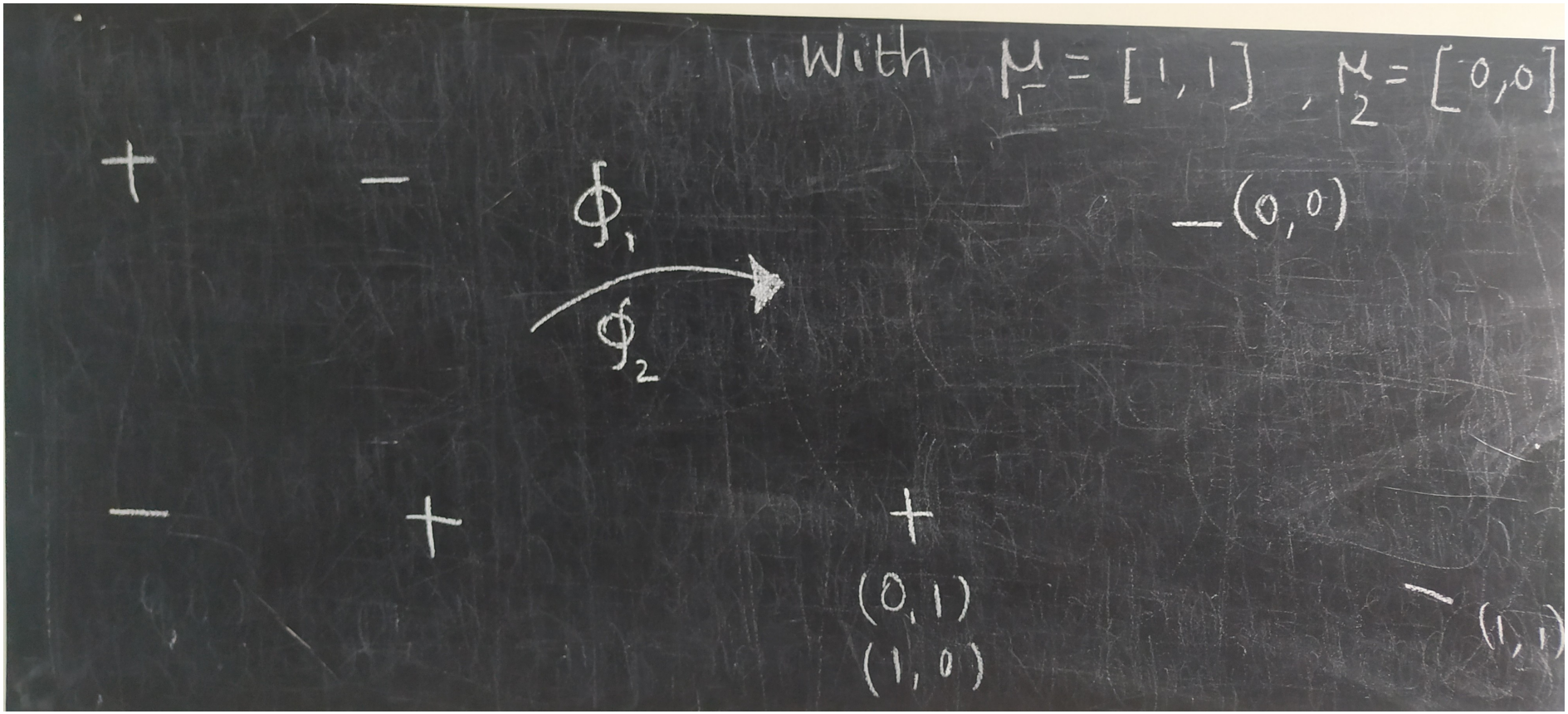
- Kernelised Neural Nets (e.g. Radial-basis Function Nets)
- Multilayered Neural Nets (e.g. Multilayer Perceptron)

Kernel Trick I

We limit our discussion to a particular kind of kernel only.



Kernel Trick II



Much of theory on RBF-Nets (next) are based on RBF kernels.

RBF-Net I

- Here, we solve the problem of classifying nonlinearly separable patterns by proceeding in a hybrid manner using two stages:
  - The first stages transforms the set of nonlinearly separable samples (patterns) into a new set, which under certain conditions, are more likely to be linearly separable in the new sample space (Mathematically justified by Cover's paper in 1965)
  - The second stage completes the solution to the prescribed classification problem by using least-squares estimation (or any other method).

RBF-Net II

- RBF networks are has a single hidden layer and an output layer.
  - The inputs take the inputs in **x**
  - The hidden units, applies a nonlinear transformation from the input space to the hidden (feature) space. [For most applications, the dimensionality of this layer should be high, so as to satisfy the Cover's theorem] (*stage 1*)
  - The output layer produces the final ouptut of the network that is then compared with the true output (*stage 2*).

RBF-Net III

Cover's Theorem (1965): A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.

RBF-Net IV

- Recall that the models that we have studied so far is:

$$\mathbf{w}^{\mathsf{T}}\mathbf{x} = 0$$

  where, $\mathbf{w}$ and $\mathbf{x}$ are d-dimensional vectors.
- Now, in stage 1, we first transform $\mathbf{x}$ via a non-linear kernel $\varphi(\cdot)$, and in stage 2, we build the model:

$$\mathbf{w}^{\mathsf{T}}\varphi(\mathbf{x}) = 0$$

  where, φ contains several radial-basis functions $\{\phi_1, \ldots, \phi_m\}$.
- Also, $d < m < N$, where N is the number of data points. (Why this value of m? – See next)

RBF-Net V

The radial-basis function ϕ_k computes a scalar value based on the similarity of a datapoint $\mathbf{x}_i$ and the center $\boldsymbol{\mu}_k$:

$$\phi_k(\mathbf{x}_i) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{||\mathbf{x}_i - \boldsymbol{\mu}_k||^2}{2\sigma_k^2}}$$

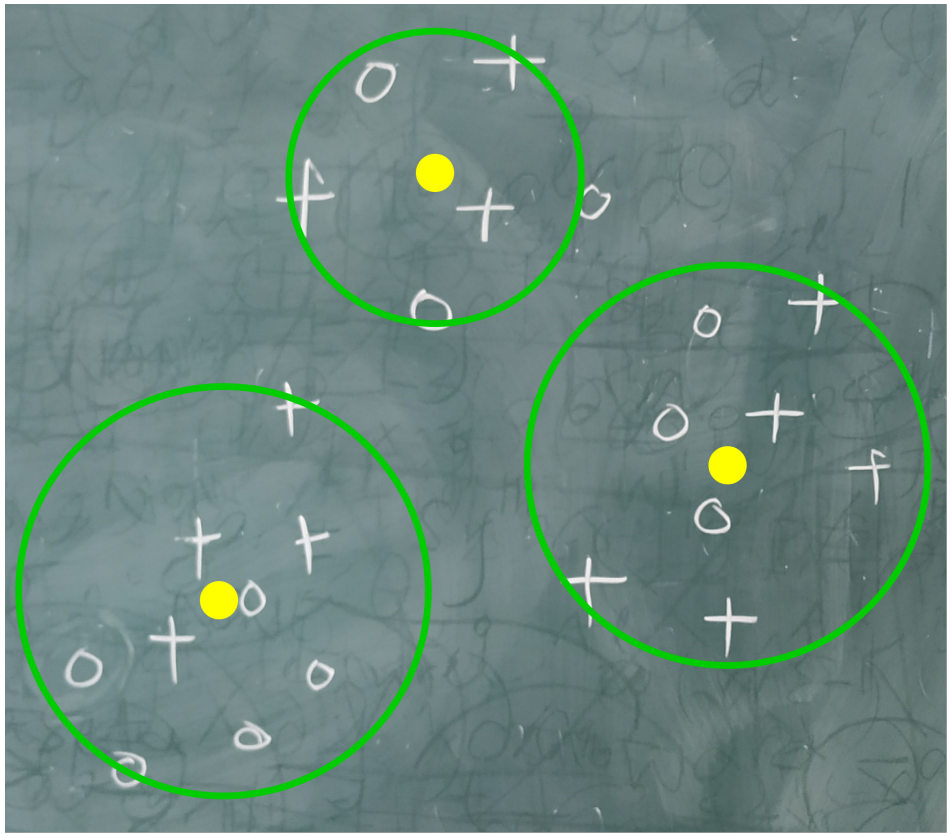
RBF-Net VI

Case $m = N$ Here, it is suggesting that each datapoint be a center $\boldsymbol{\mu}_i = \mathbf{x}_i$; $1 \leq i \leq m$.

(This was the initial proposal for RBF-nets, which were based on interpolation theory)

RBF-Net VII

Case $m < N$ Here, it is suggesting that a representative of a group be a center $\boldsymbol{\mu}_i = \mathbf{c}_i$; $1 \leq i \leq m$.



RBF-Net VIII

Parameter: m and $\boldsymbol{\mu}$

There is a close relationship between obtaining these two parameters:

- Use k-means clustering on data $\mathbf{X}$ with $k = m$.
- The centroids($\mathbf{c}_i$s) obtained after the convergence of k-means are chosen to be $\boldsymbol{\mu}_i$s.

The k-means algorithm can be based on:

- a fixed k and $k > d$
- there are various methods to obtain a good value for k such as the elbow method, silhouette method or cross-validation.

RBF-Net IX

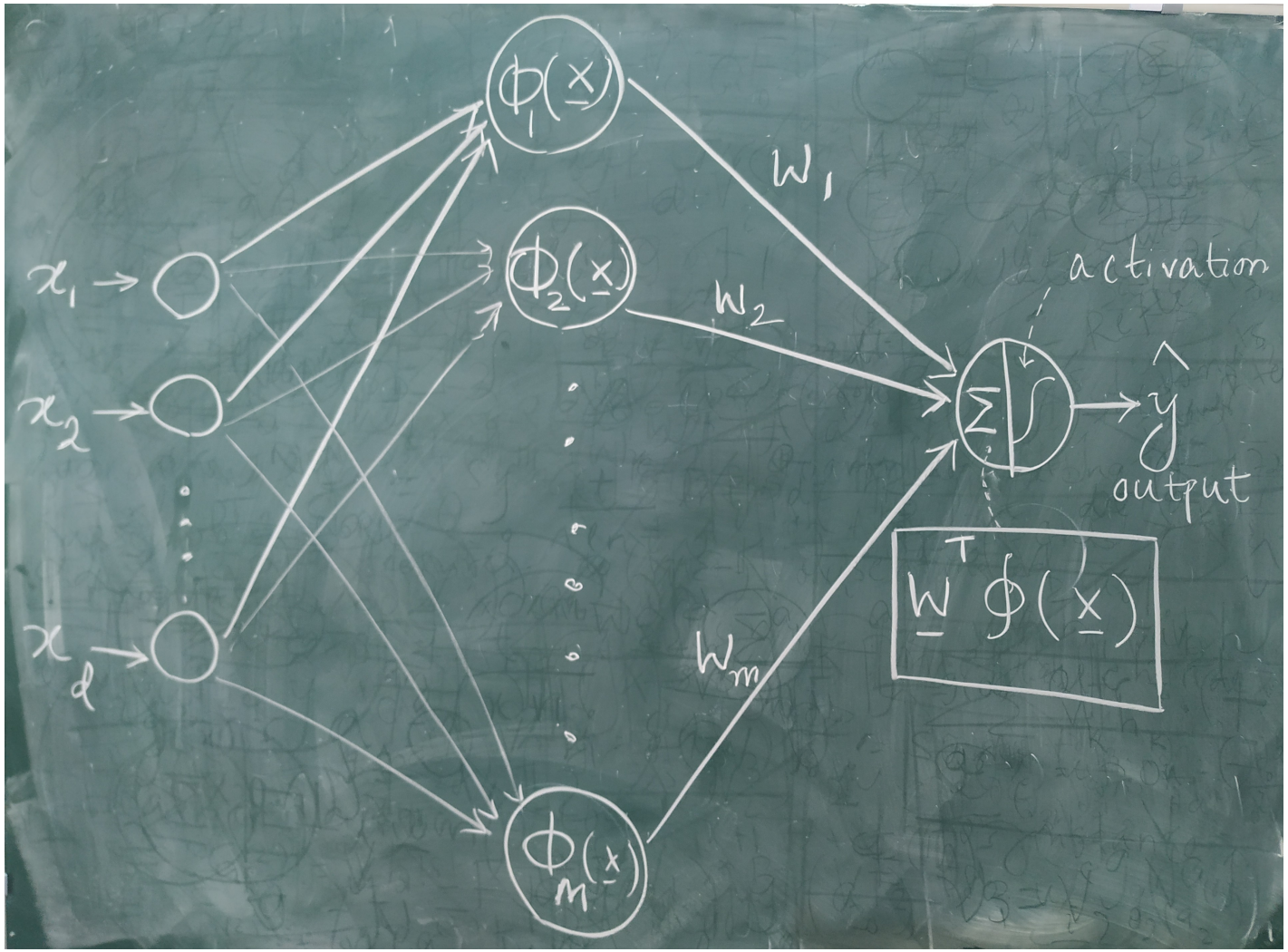
Parameter: σ

Note that so far we have not discussed about σs in the basis functions. We note these points:

- The σ parameter behaves as a 'radius' (hence the name "radial basis") of the circular spread from the center $\mathbf{c}$ or $\boldsymbol{\mu}$.
- These can be obtained from the k-means clusters discussed in the previous slide.
- These parameters are fixed to same value for all the basis functions $\phi_1, \ldots, \phi_k$.
- A typical value is 1.
- Other computationally complicated alternatives can be chosen such as treating σ (for a fixed value) or each σ_i (for $1 \leq i \leq k$) as *hyperparameters* and tuning it via cross-validation.

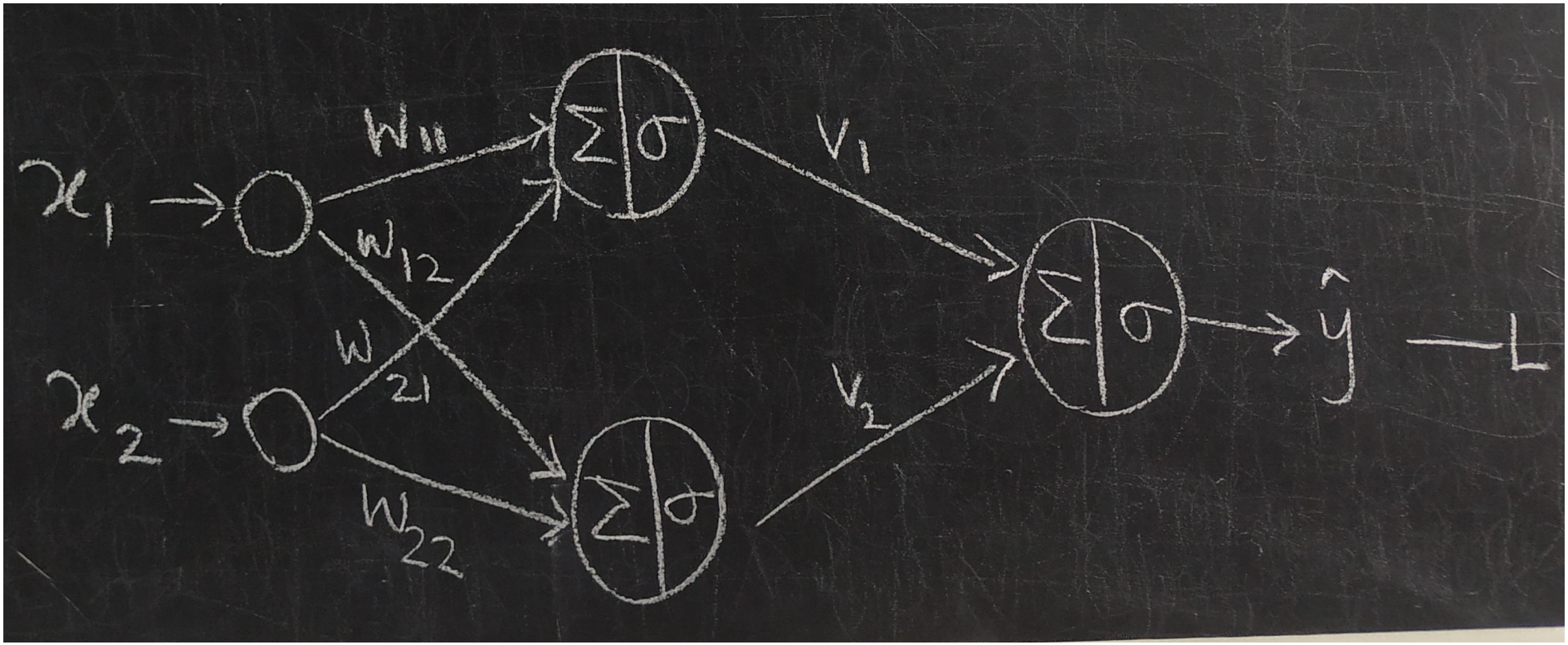
RBF-Net X

Final structure of RBF net

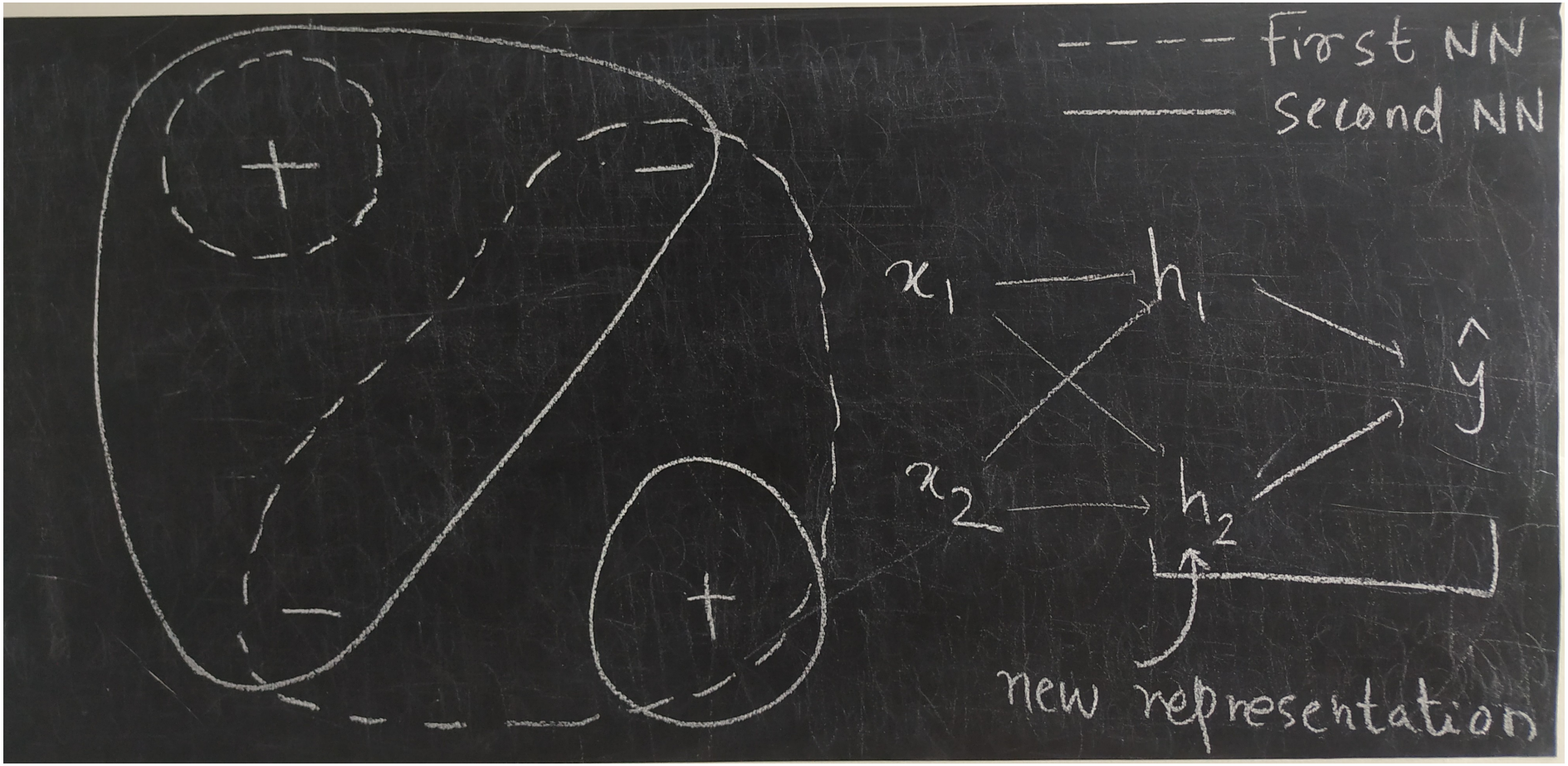


Multilayer NNs I

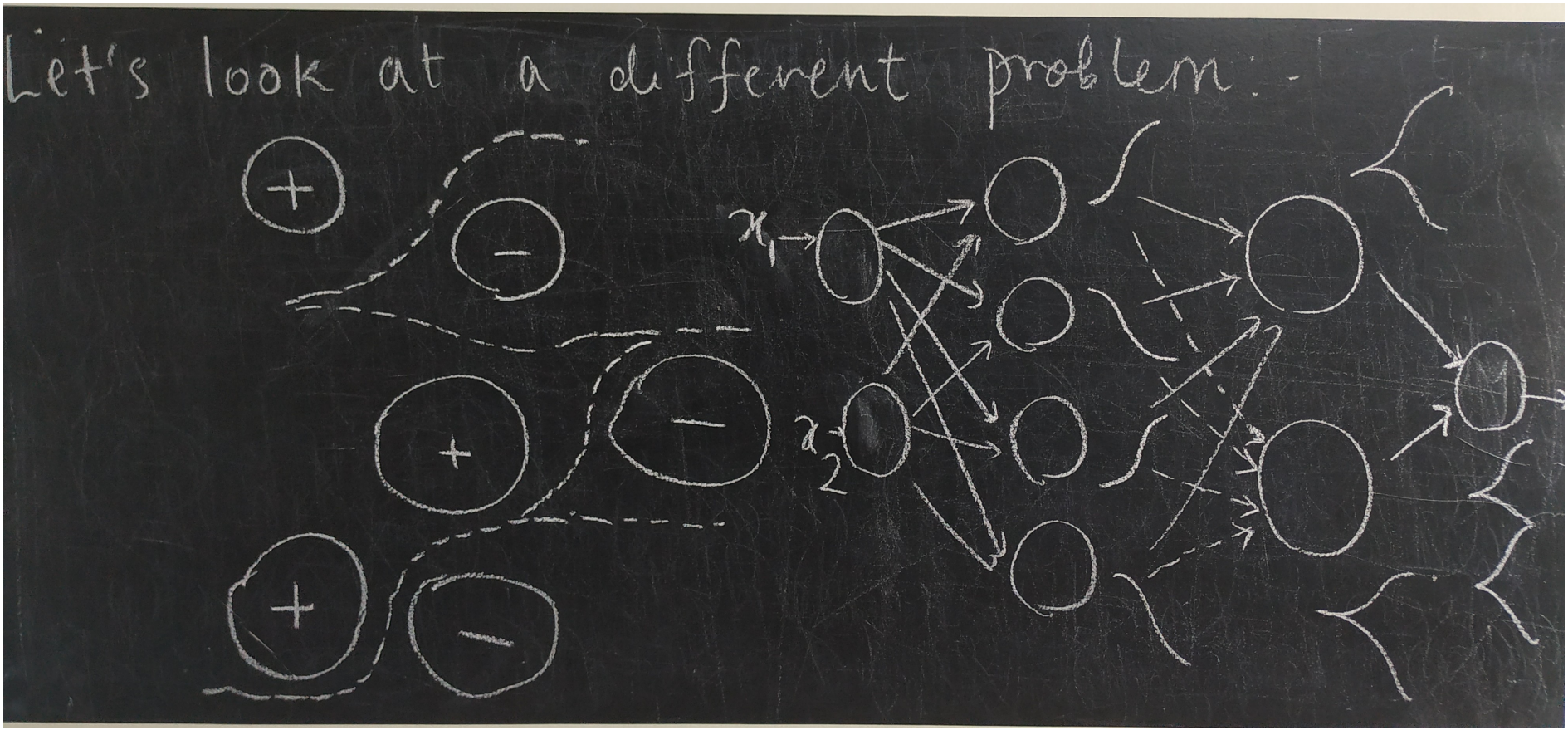
Muliple levels of transformation of features using layers:



Multilayer NNs II

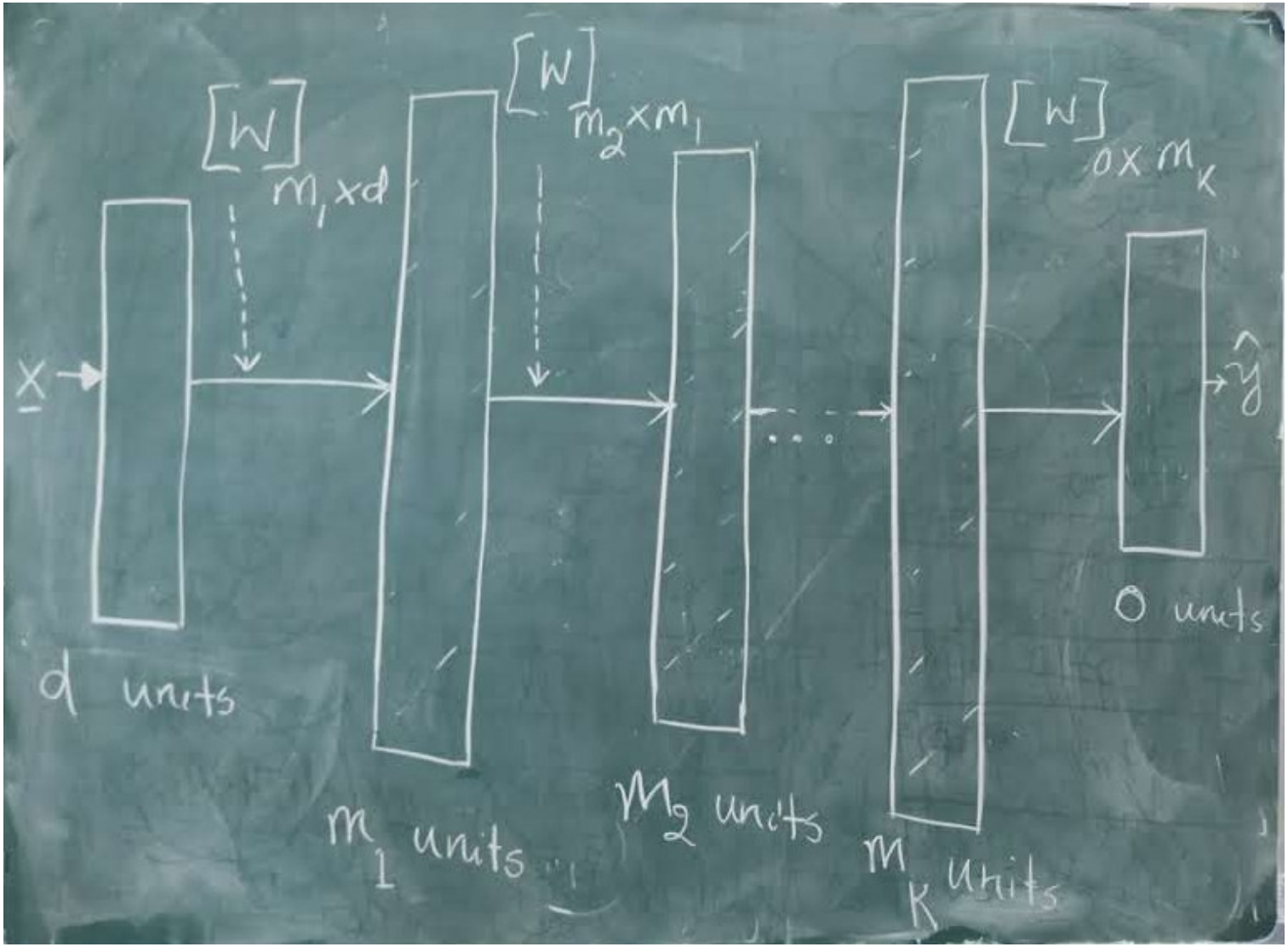


Multilayer NNs III



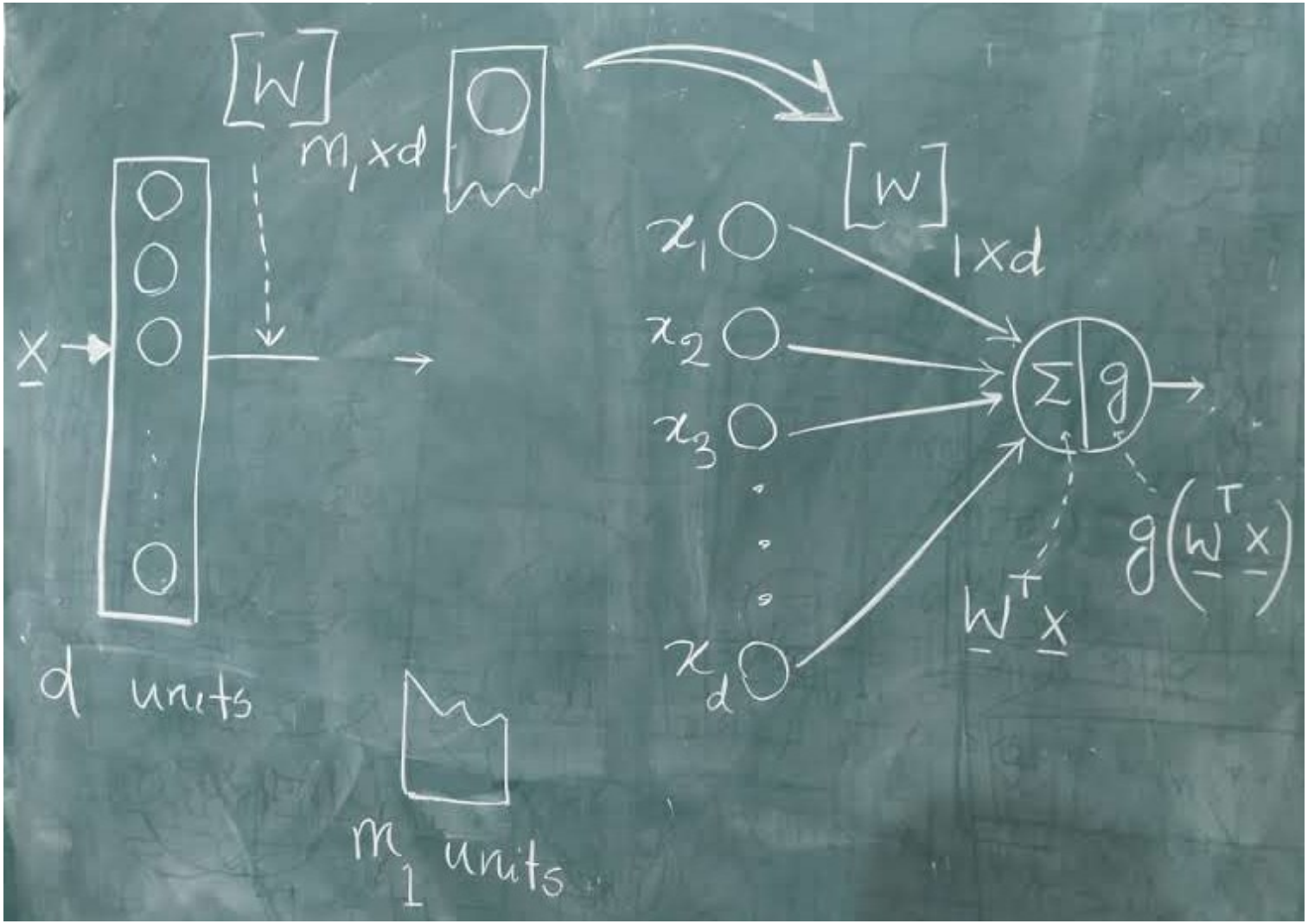
A neural network that is based on the this idea is 'Multilayer Perceptron (MLP)'.

MLPs I


[W]
$m_1 \times d$
[W]
$m_2 \times m_1$
[W]
$o \times m_K$
$\underline{x}$
$\hat{y}$
d units
m_1 units
m_2 units
m_K units
o units

MLPs II

Visualising a single neuron in a hidden layer:



MLPs III

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?

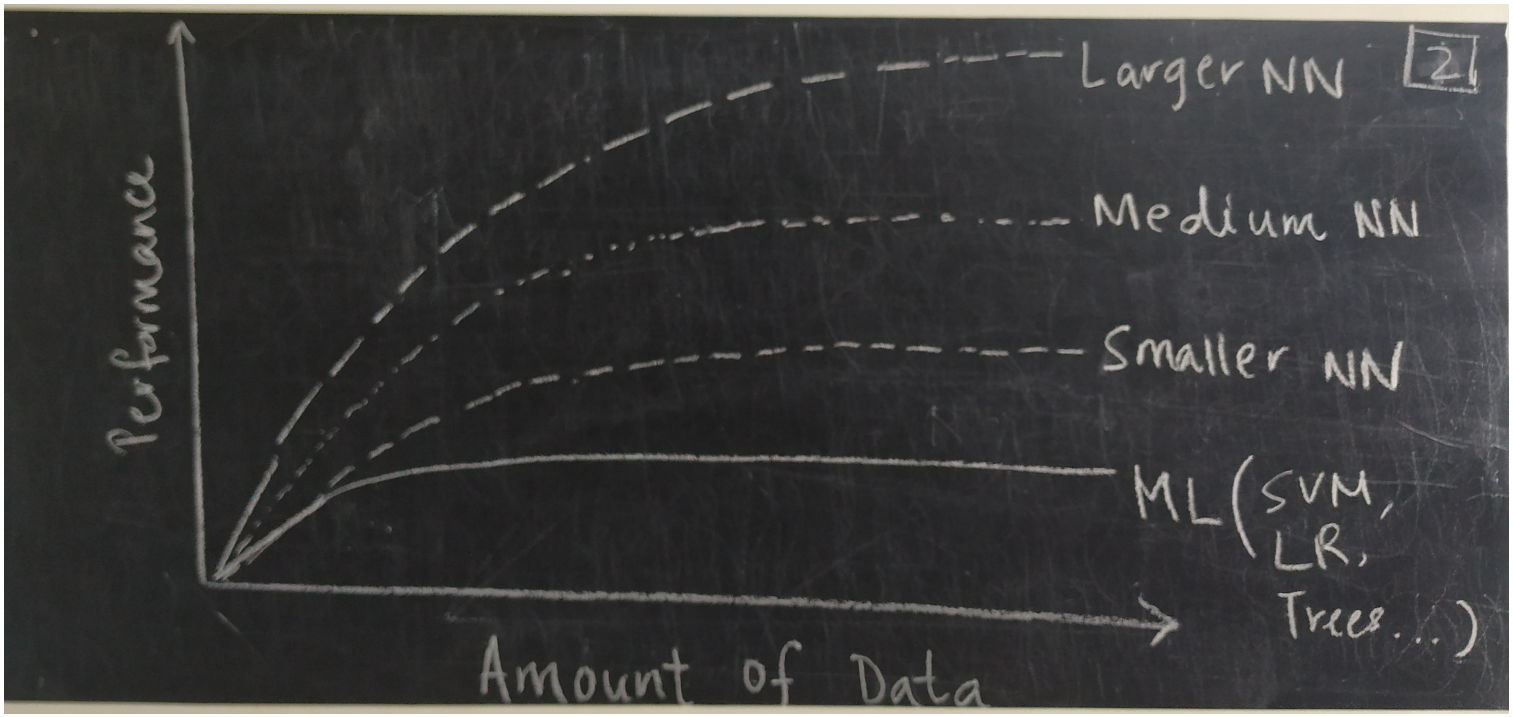
Let's look each of these next.

MLPs IV

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.

MLPs V



(In practice, this may not always turn out to be true. This figure is just for the sake of explaining the notion of *depth* and *width* of a deep net.)

MLPs VI

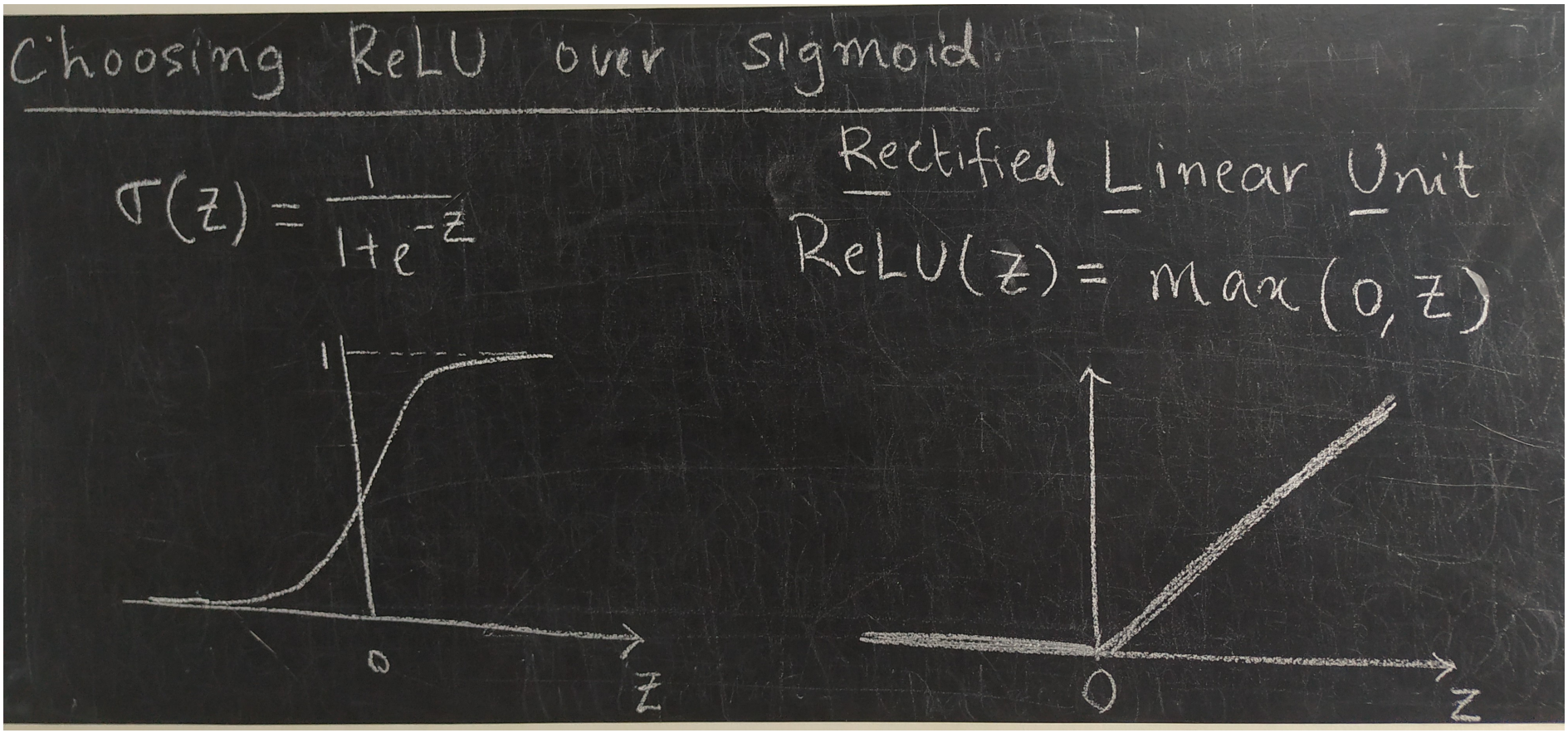
Activation function:

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

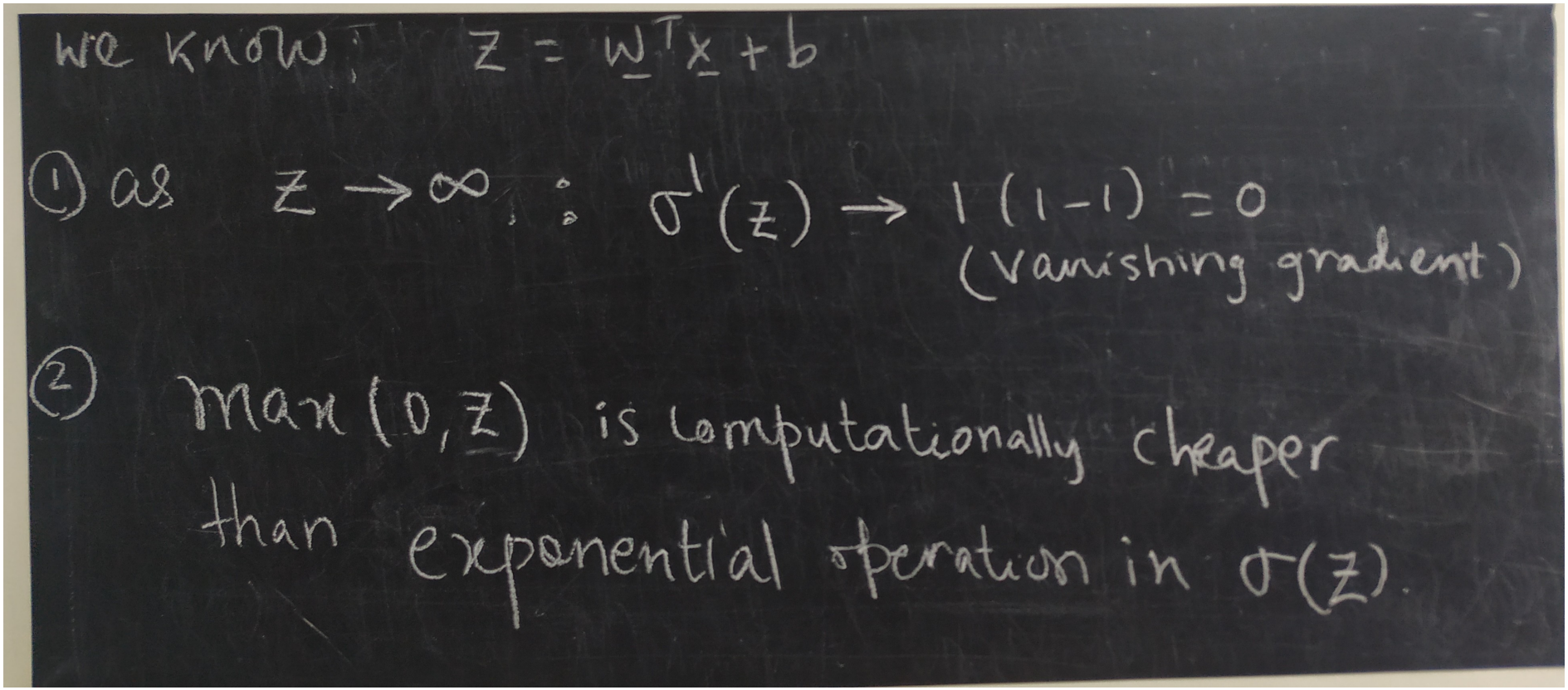
Problem	# of output units	$g(\cdot)$
$y \in \mathbb{R}$	1	linear
$y \in \{0, 1\}$	1	sigmoid
$y \in \{1, \ldots, c\}$	c	softmax

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

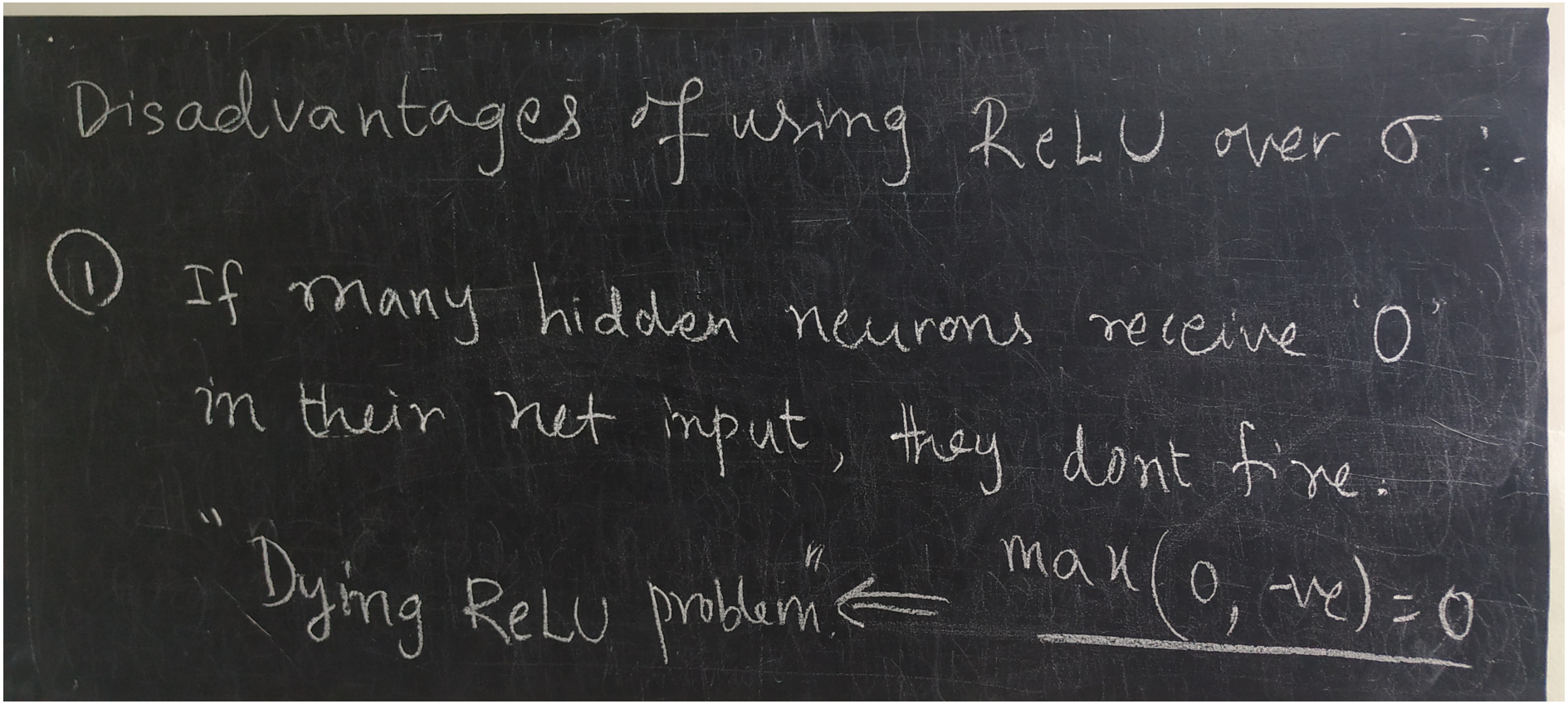
ReLU activation:



MLPs VIII


we know; $z = \underline{w}^T \underline{x} + b$
① as $z \to \infty$; $\sigma'(z) \to 1(1-1) = 0$
(Vanishing gradient)
② $\max(0, z)$ is computationally cheaper than exponential operation in $\sigma(z)$.

MLPs IX


Disadvantages of using ReLU over σ :-
① If many hidden neurons receive '0' in their net input, they dont fire.
"Dying ReLU problem" ⟸ max(0, -ve) = 0

MLPs X

Two variants of ReLU function:

- Leaky ReLU:

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

- Parametric ReLU:

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

MLPs XI

Weights:

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

Training using Backpropagation

- In the lecture, we will derive the parameter update rules using error backpropagation. There, we will see how simple chain-rule of derivative calculus can be so instrumental in training deep networks.
- We will also see how various activation function in the hidden layers have significant influence on the multilayered neural net training. A brief overview of these is provided in the following slides.