Title: Machine Learning (2021/2022)

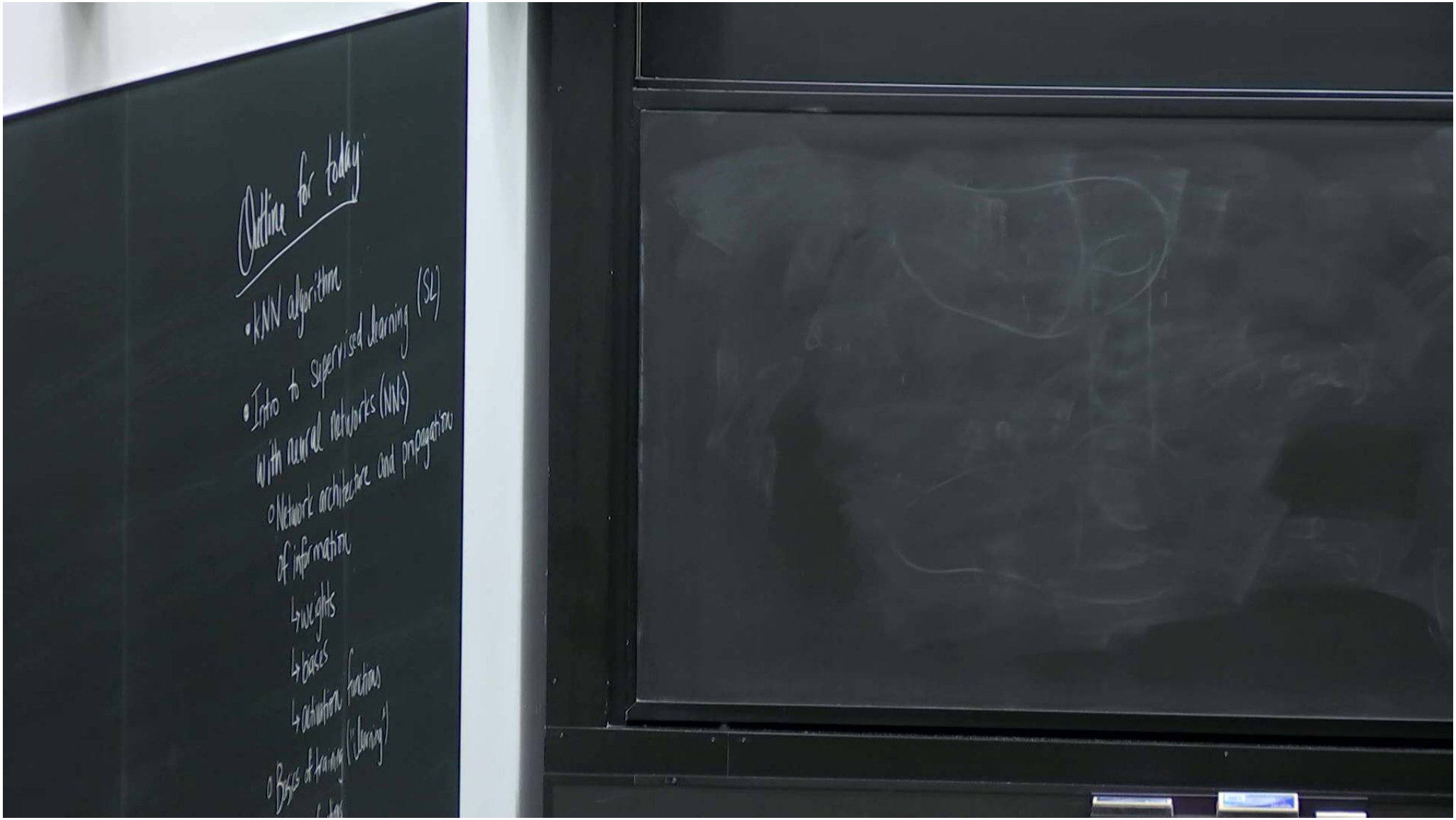
Speakers: Lauren Hayward

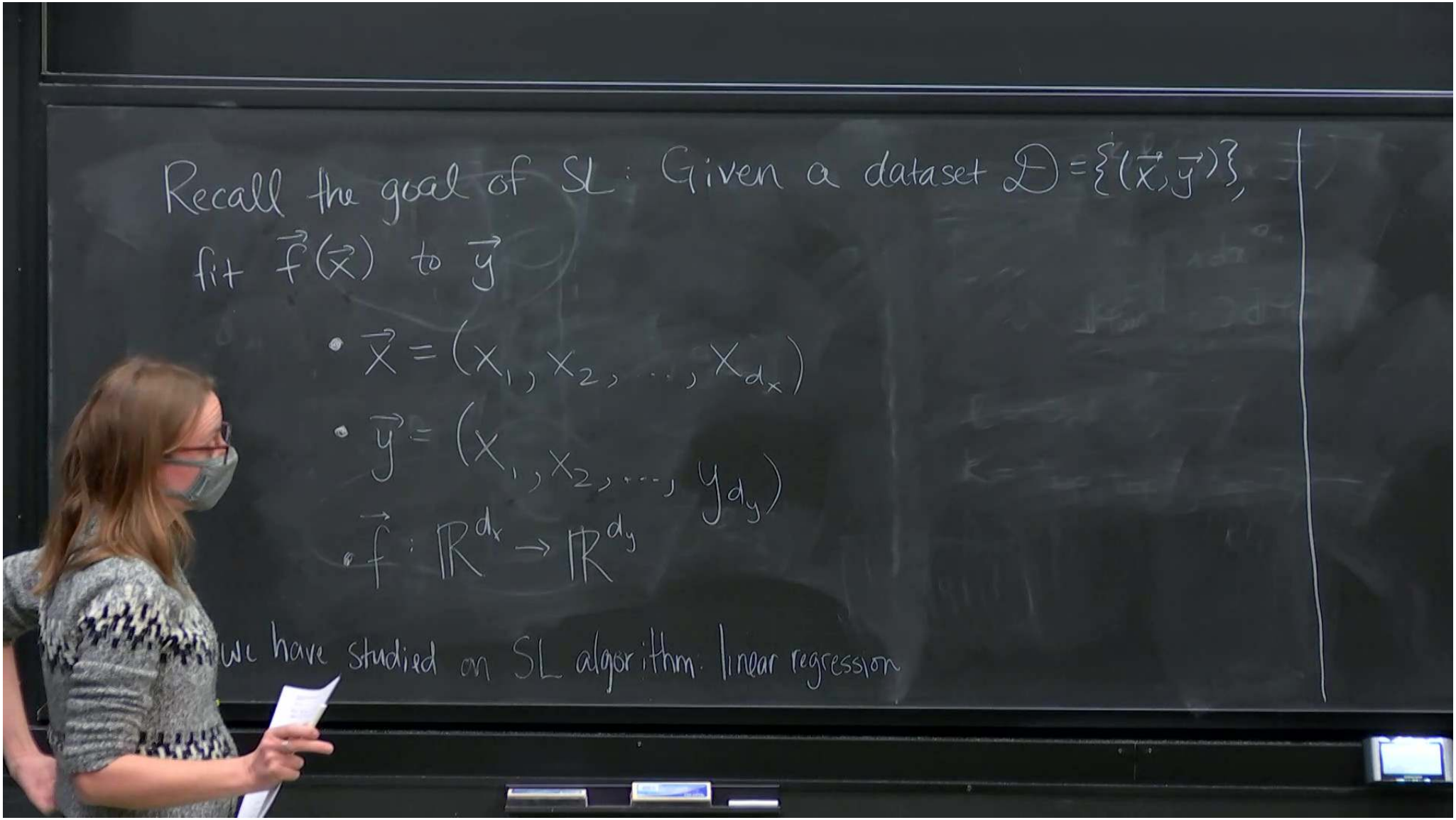
Collection: Machine Learning (2021/2022)

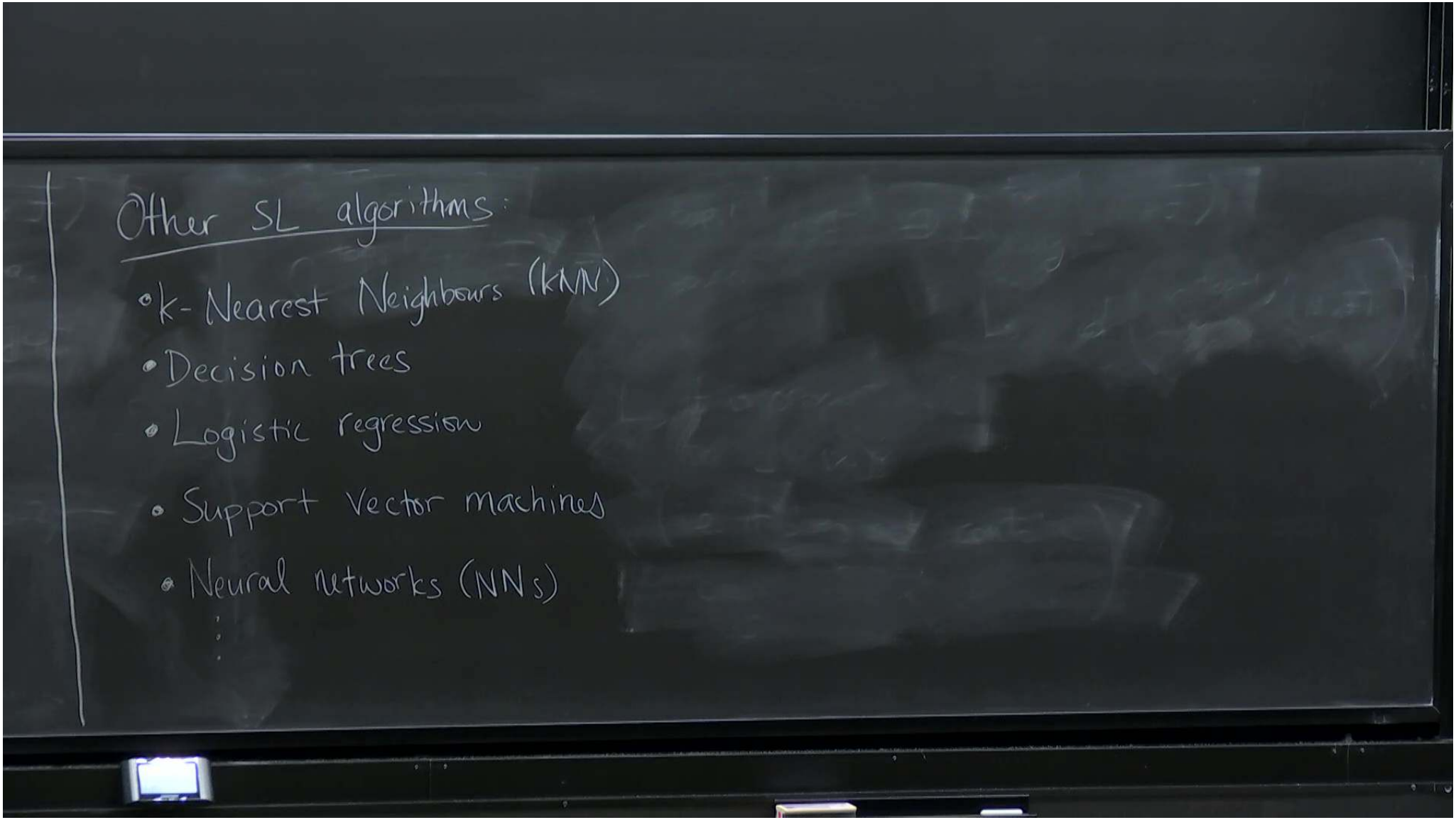
Date: April 11, 2022 - 11:30 AM

URL: https://pirsa.org/22040067

Abstract: This course is designed to introduce modern machine learning techniques for studying classical and quantum many-body problems encountered in condensed matter, quantum information, and related fields of physics. Lectures will focus on introducing machine learning algorithms and discussing how they can be applied to solve problem in statistical physics. Tutorials and homework assignments will concentrate on developing programming skills to study the problems presented in lecture.


Outline for today:
• kNN algorithm
• Intro to Supervised learning (SL)
with neural networks (NNs)
○ Network architecture and propagation
of information
↳ weights
↳ biases
↳ activation functions
○ Basics of training ("learning")


Recall the goal of SL: Given a dataset $\mathcal{D} = \{(\vec{x}, \vec{y})\}$,
fit $\vec{f}(\vec{x})$ to $\vec{y}$
$\vec{x} = (x_1, x_2, \ldots, x_{d_x})$
$\vec{y} = (x_1, x_2, \ldots, y_{d_y})$
$\vec{f} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$
we have studied on SL algorithm: linear regression


Other SL algorithms:
• k-Nearest Neighbours (kNN)
• Decision trees
• Logistic regression
• Support Vector machines
• Neural networks (NNs)
⋮

kNN

For a dataset $\mathcal{D} = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^{M}$

To determine $f(\vec{x})$ for kNN:

↳ Step 1: Calculate the distance $d(\vec{x}, \vec{x}^{(i)}) = \|\vec{x} - \vec{x}^{(i)}\|_2$ for all $\vec{x}^{(i)}$ in $\mathcal{D}$

↳ Find the k closest points to $\vec{x}$. Store these as $\vec{x}^{*(j)}$ and their labels as $\vec{y}^{*(j)}$ $(1 \leq j \leq k)$

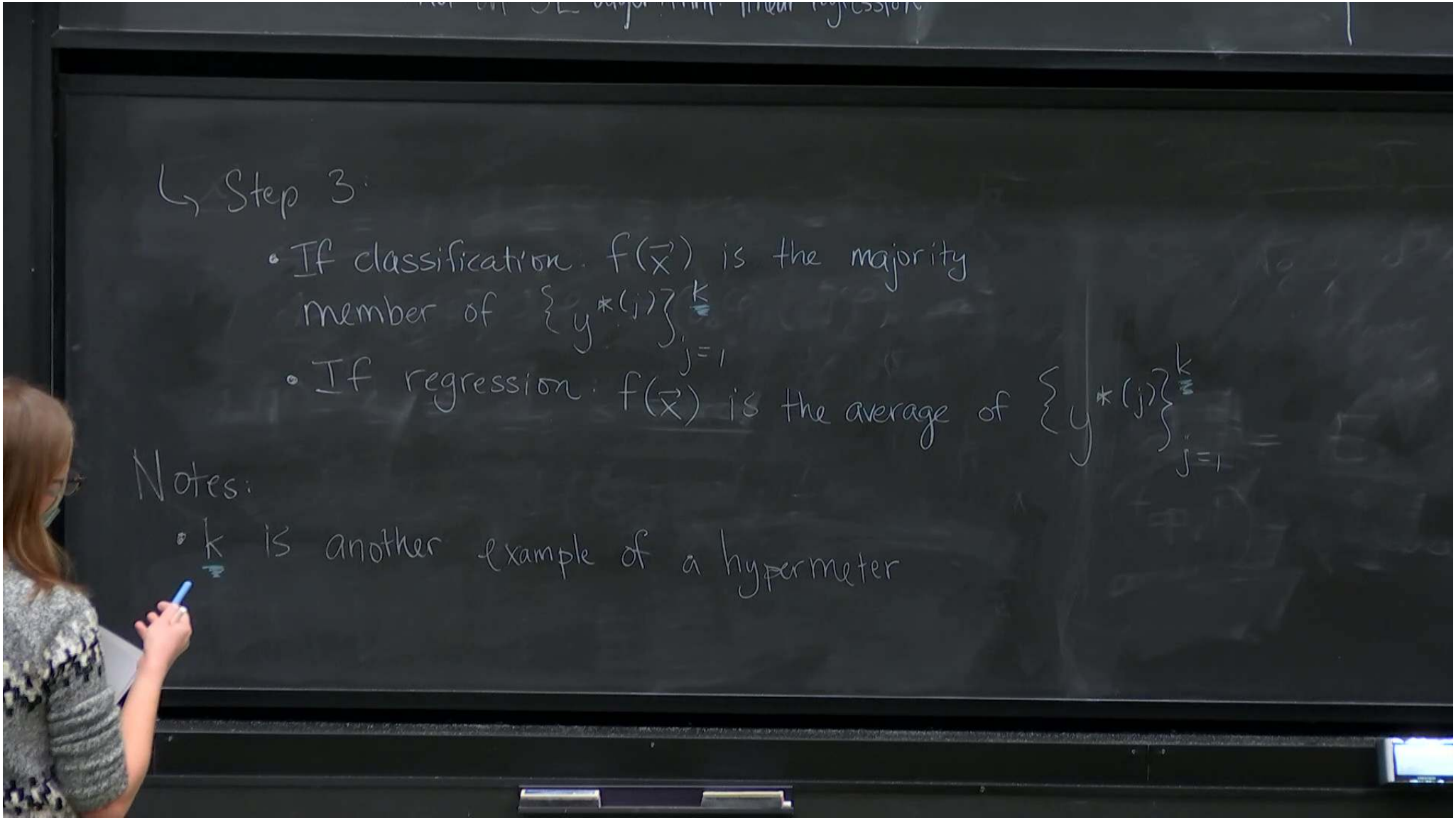
kNN

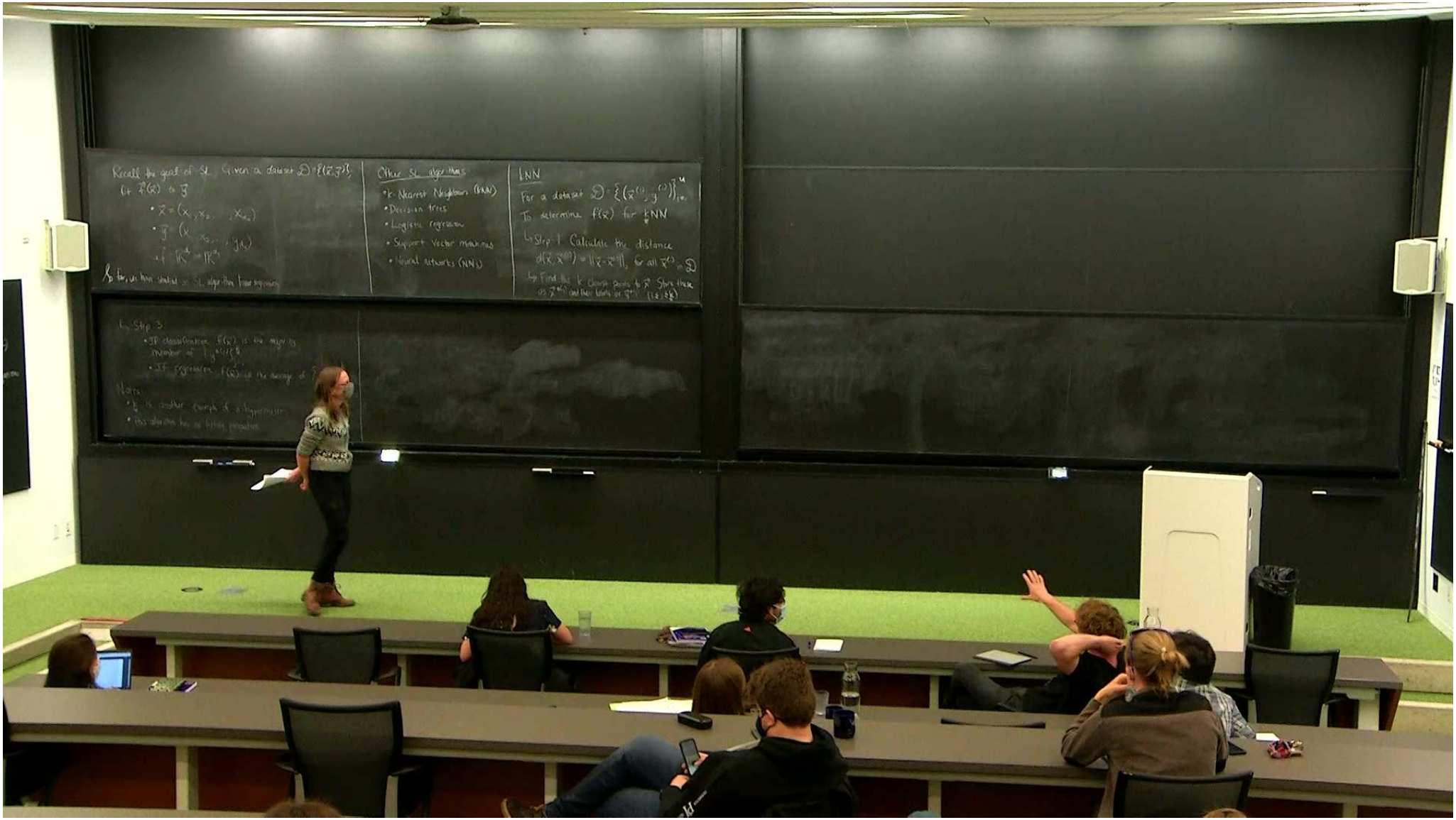
For a dataset $\mathcal{D} = \{(\vec{x}^{(i)}, y^{(i)})\}_{i=1}^{M}$

To determine $f(\vec{x})$ for kNN:

↳ Step 1: Calculate the distance $d(\vec{x}, \vec{x}^{(i)}) = \|\vec{x} - \vec{x}^{(i)}\|_2$ for all $\vec{x}^{(i)}$ in $\mathcal{D}$

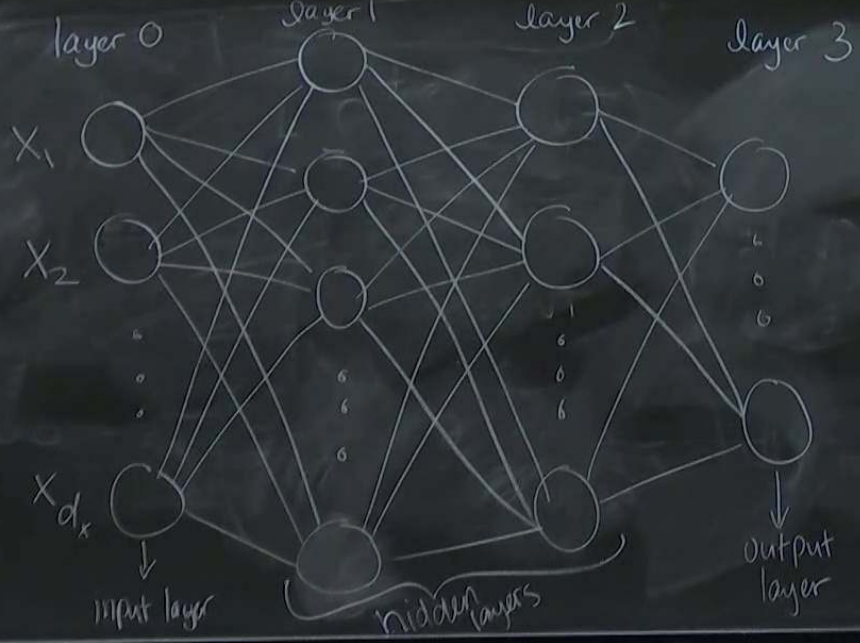
↳ Find the k closest points to $\vec{x}$. Store these as $\vec{x}^{*(j)}$ and their labels as $\vec{y}^{*(j)}$ $(1 \leq j \leq k)$


↳ Step 3:
• If classification: $f(\vec{x})$ is the majority member of $\{y^{*(j)}\}_{j=1}^{k}$
• If regression: $f(\vec{x})$ is the average of $\{y^{*(j)}\}_{j=1}^{k}$
Notes:
• k is another example of a hypermeter

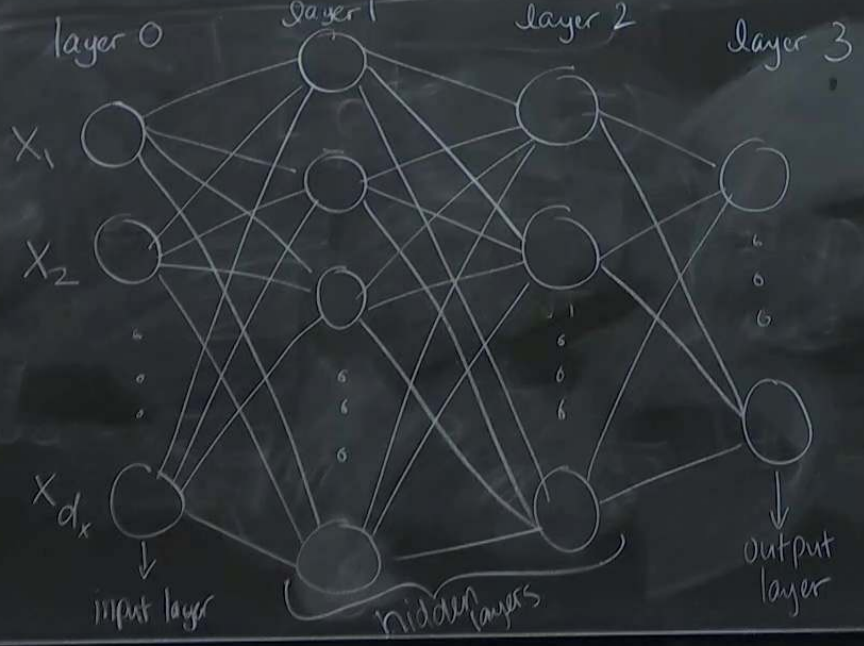

Other SL algorithms
• k- Nearest Neighbours (kNN)
• Decision trees
• Logistic regression
• Support Vector machines
• Neural networks (NNs)
kNN

⇒ Find the k closest points to $\vec{x}$. Store these as $\vec{x}^{*(j)}$ and their labels as $\vec{y}^{*(j)}$ $(1 \le j \le k)$

Feedforward Neural Networks



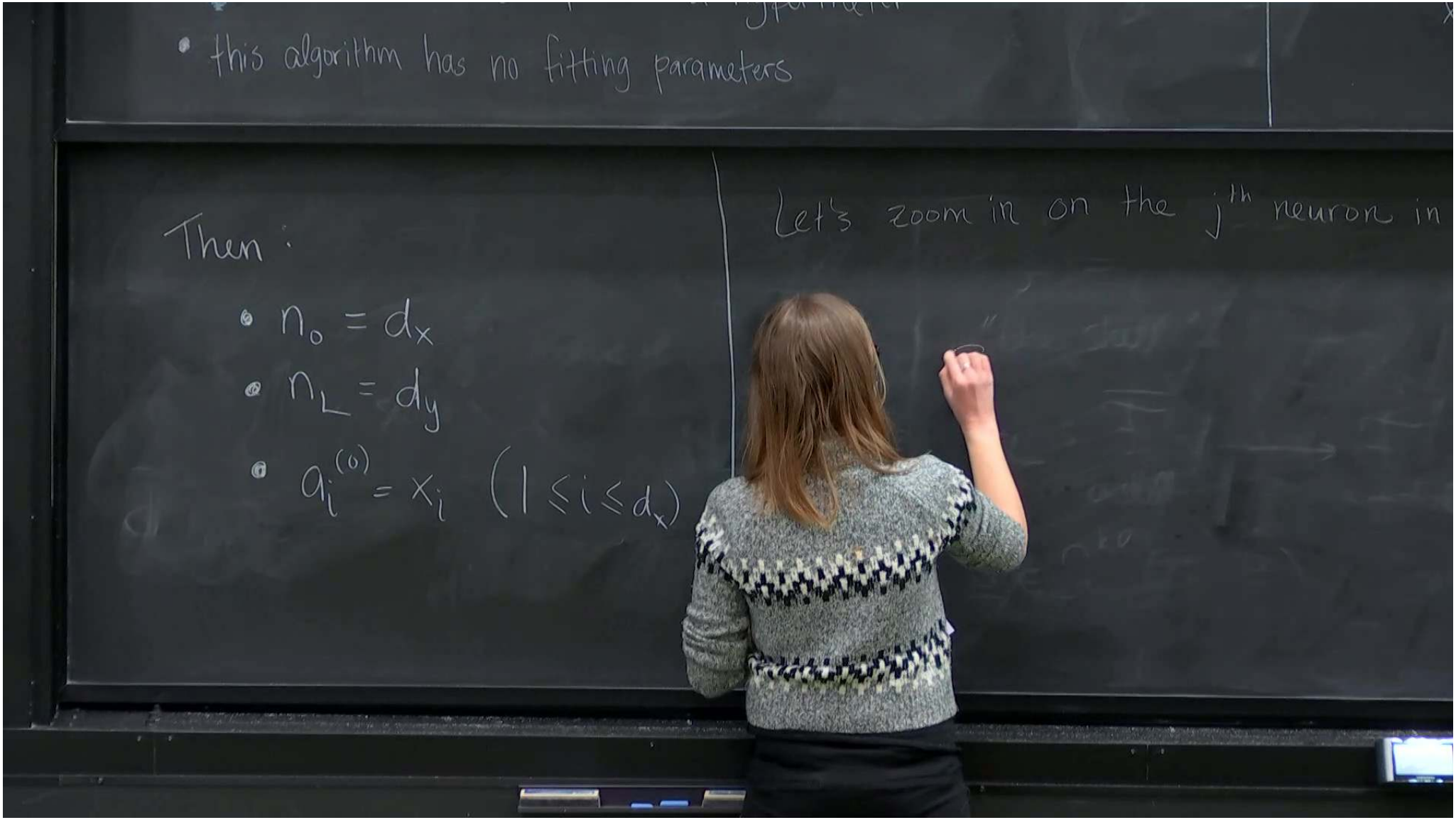
Feedforward Neural Networks



Each ○ is a "neurons"

Notation:

- $n_\ell \equiv$ # of neurons in layer ℓ
- $L \equiv$ largest value of ℓ (eg: $L = 3$ here)
- $a_j^{(\ell)} \equiv$ output from the j^{th} neuron in layer ℓ


this algorithm has no fitting parameters
Then:
$n_0 = d_x$
$n_L = d_y$
$a_i^{(0)} = x_i \quad (1 \leq i \leq d_x)$
Let's zoom in on the j^{th} neuron in

ng parameters

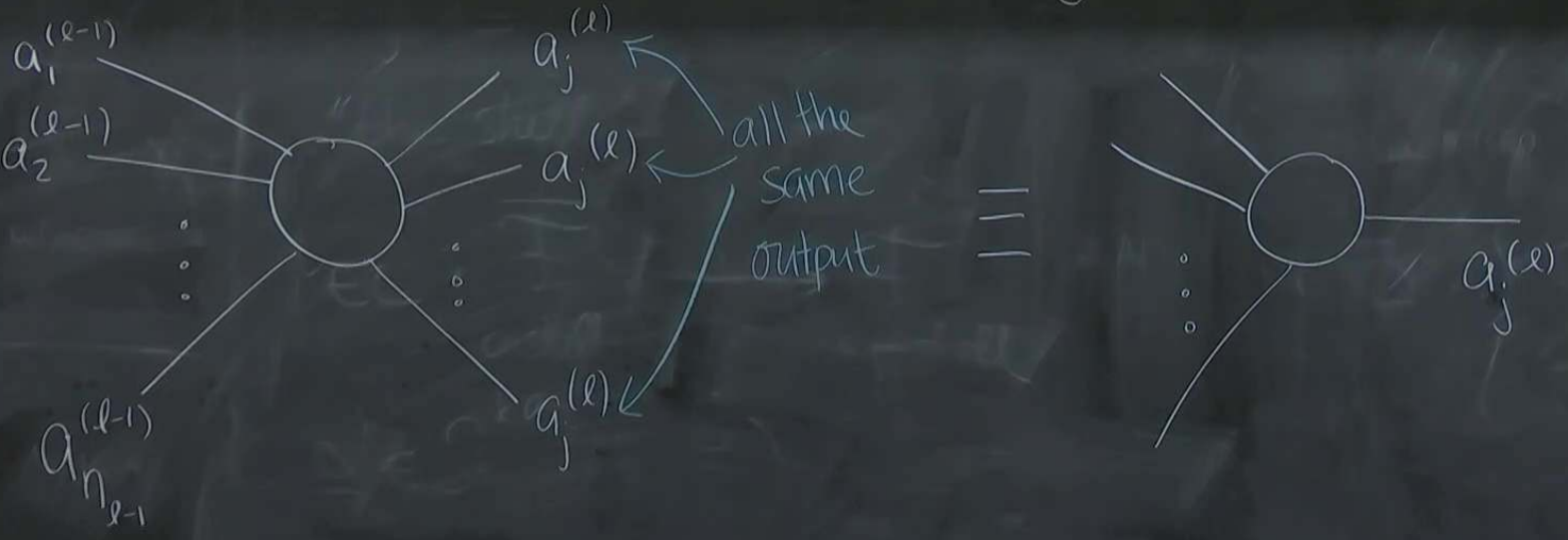
d_x

input layer

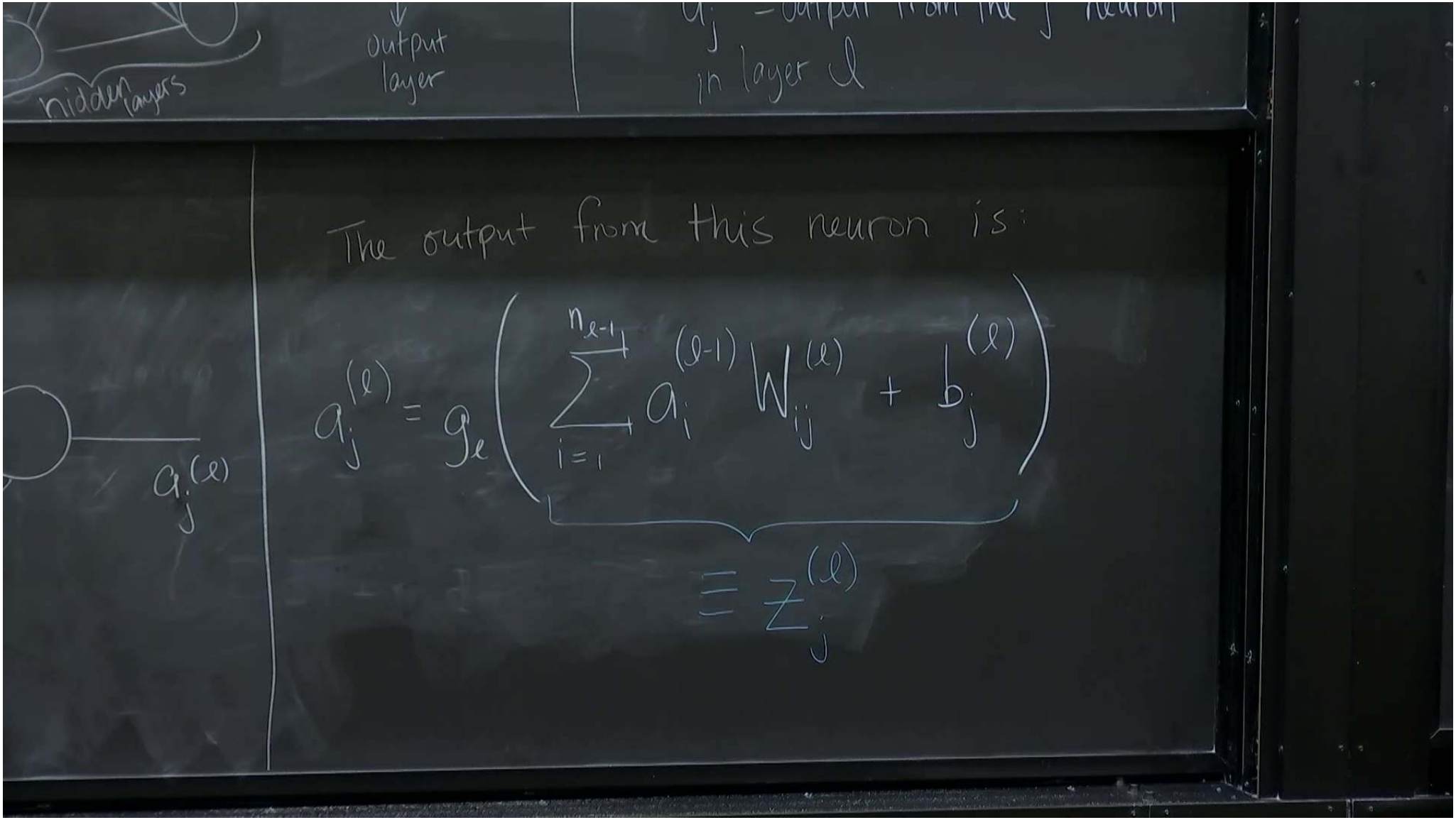
hidden layers

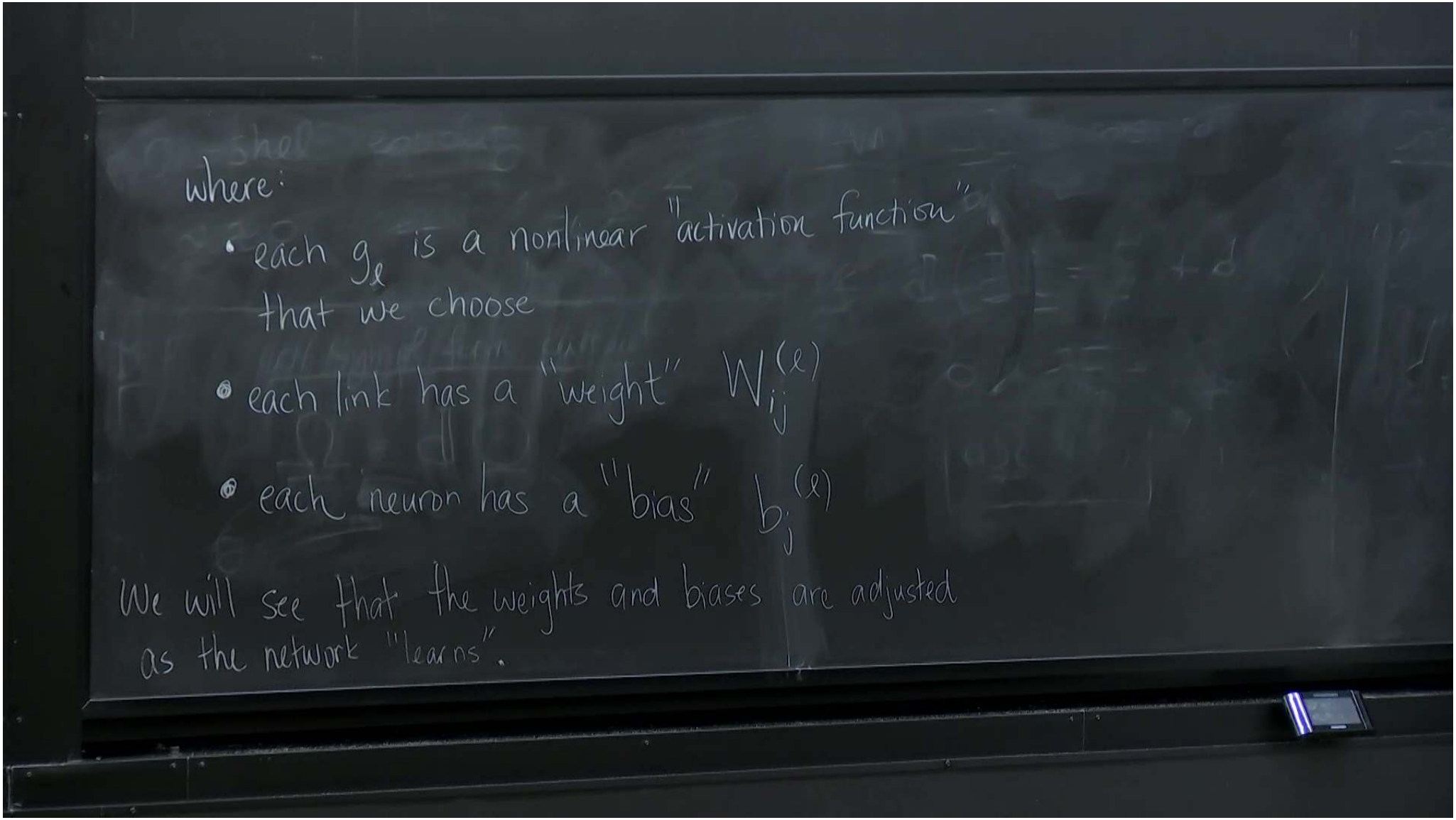
output layer

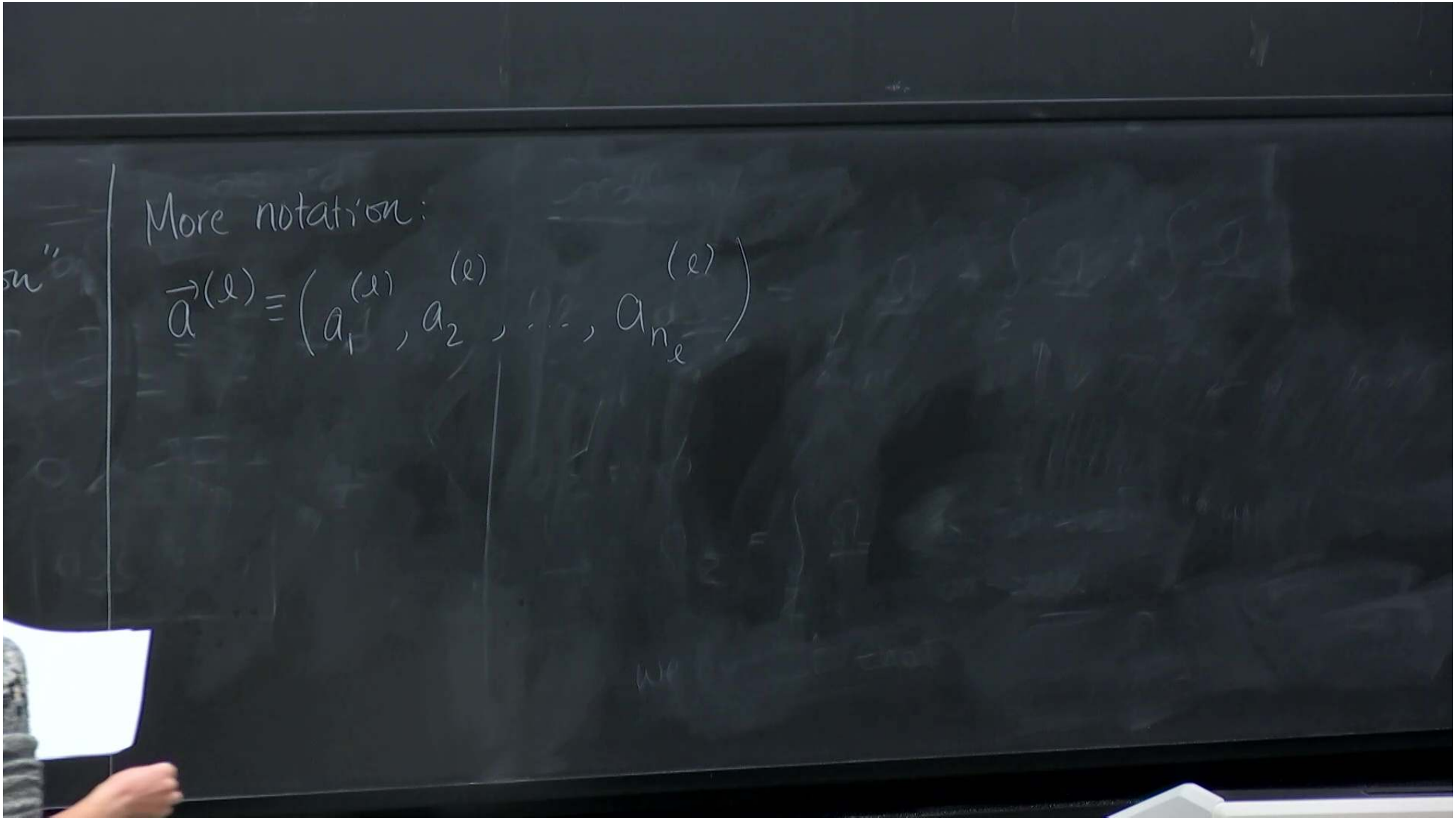
Let's zoom in on the j^{th} neuron in layer l

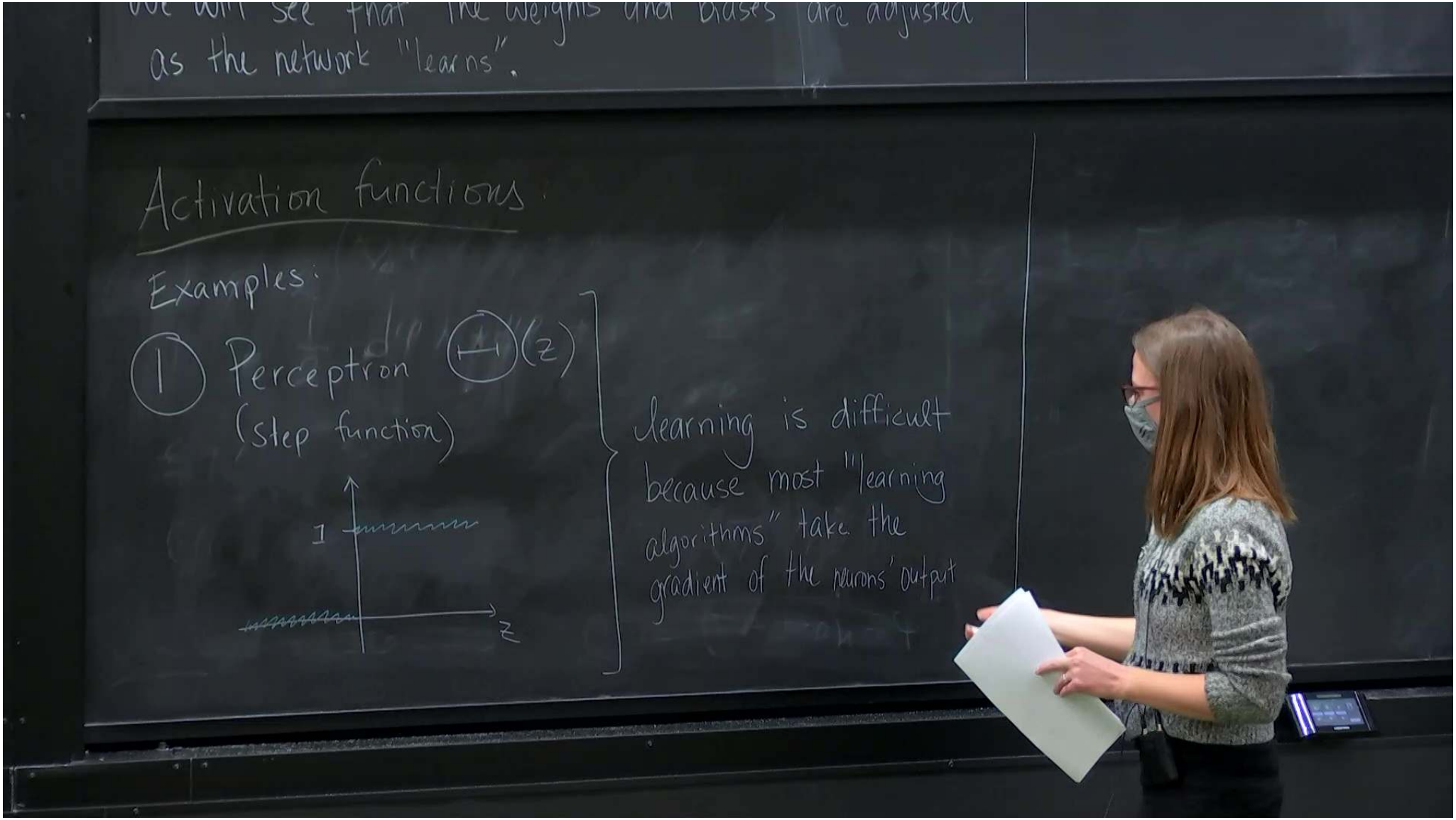


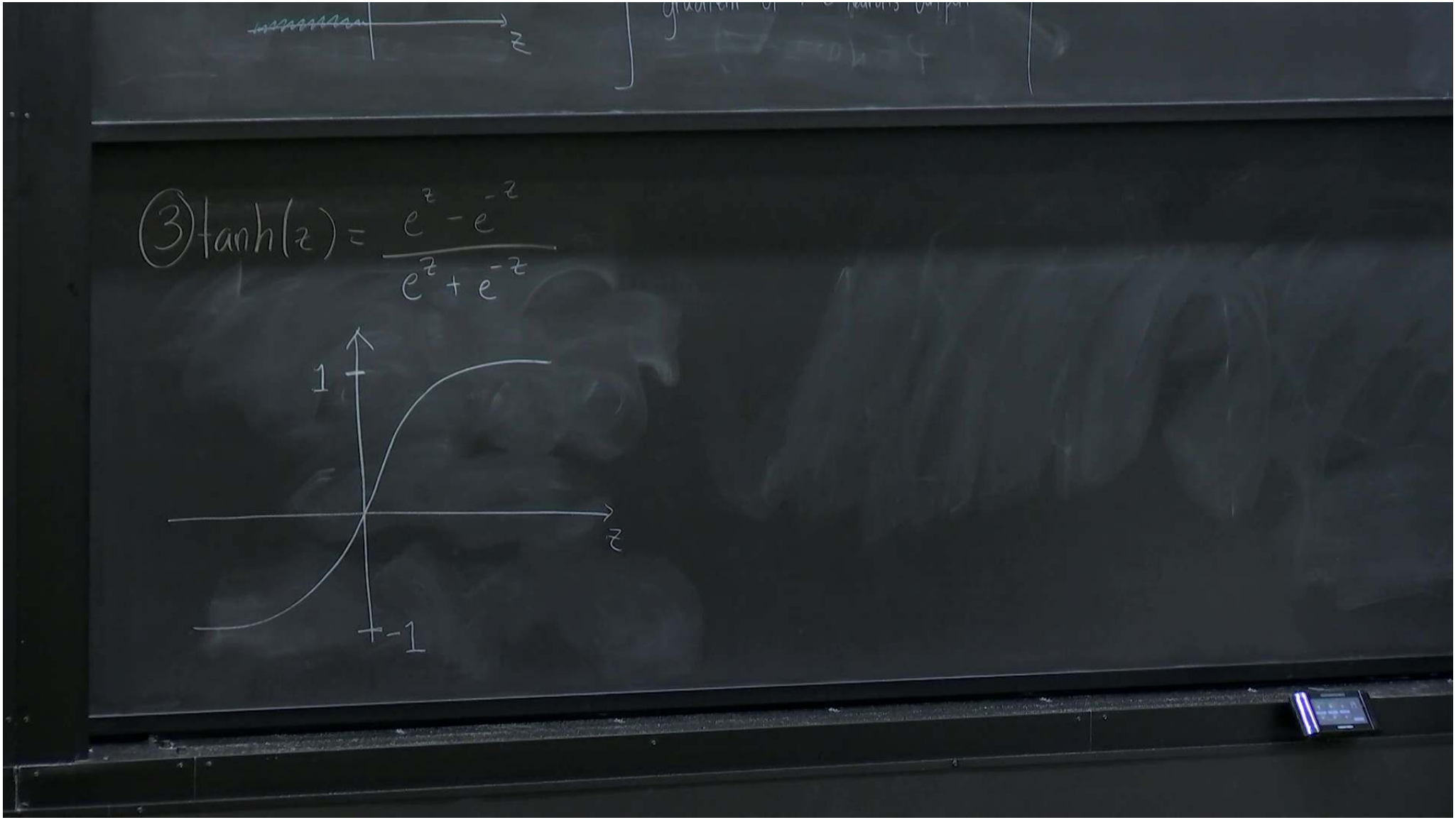
$\leq d_x$)

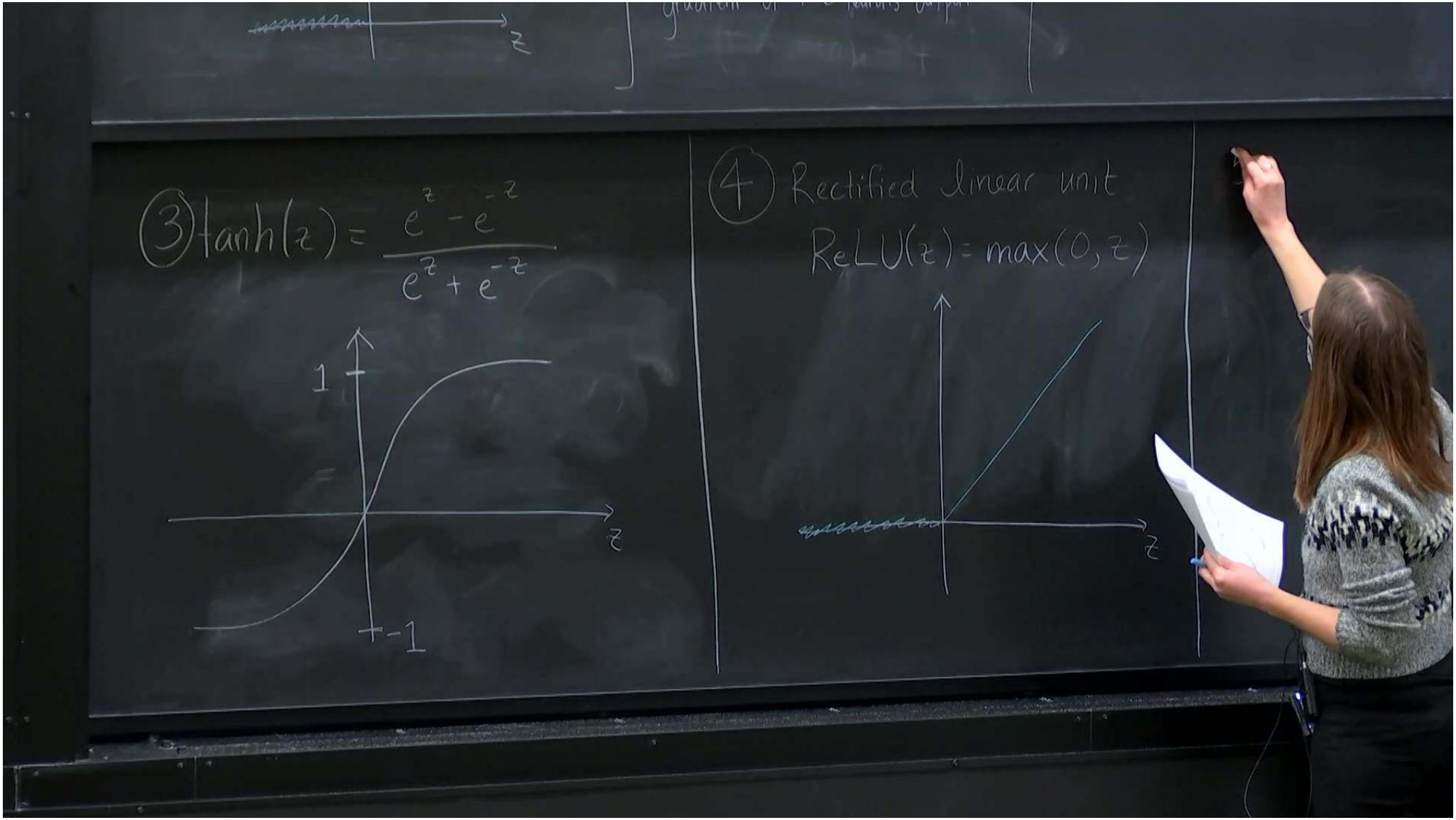

hidden layers
output layer
a_j = output from the j neuron in layer l
The output from this neuron is:
$a_j^{(l)} = g_l\left(\sum_{i=1}^{n_{l-1}} a_i^{(l-1)} W_{ij}^{(l)} + b_j^{(l)}\right)$
$\equiv z_j^{(l)}$
$a_j^{(l)}$

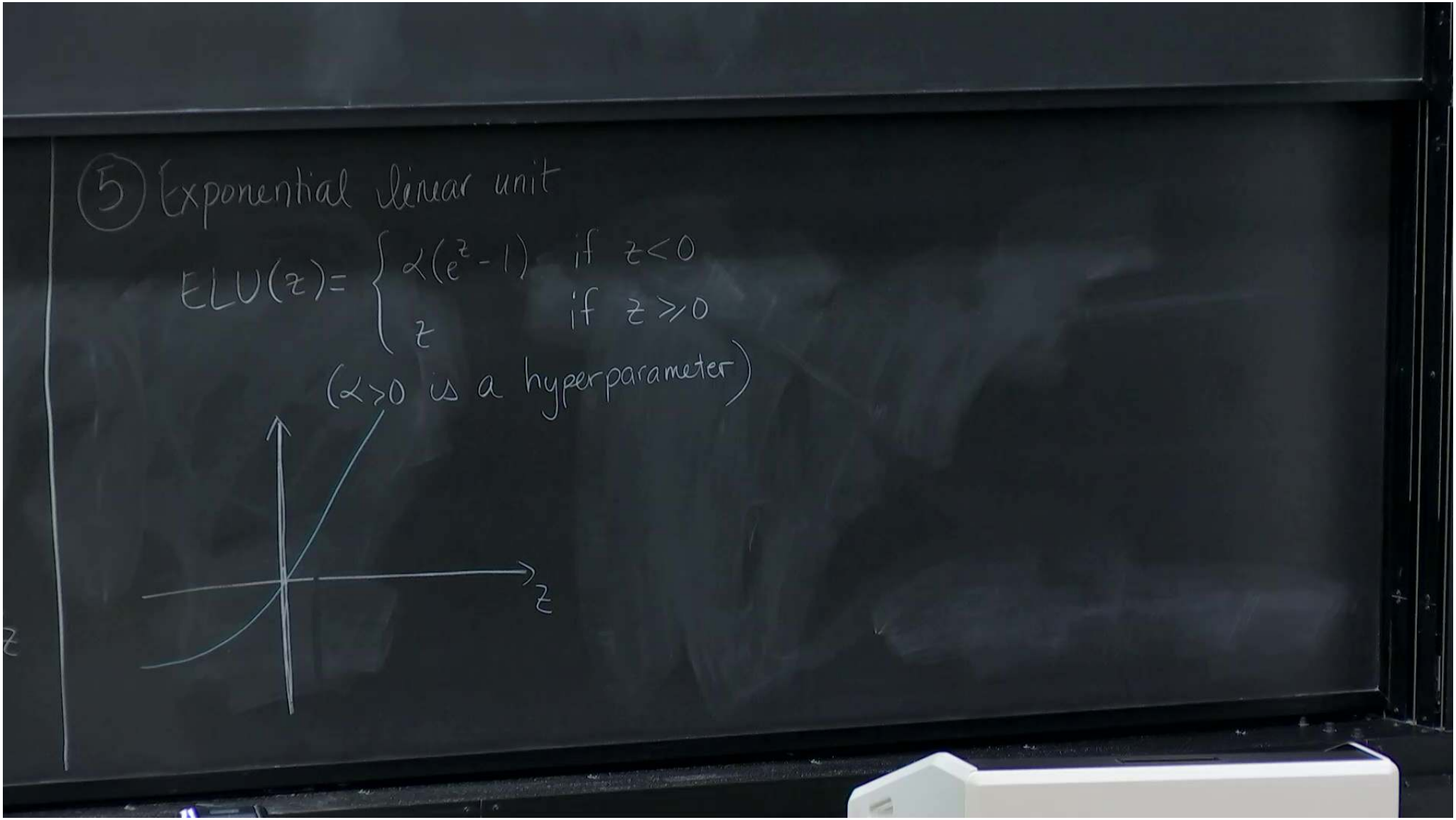

where:
• each g_ℓ is a nonlinear "activation function" that we choose
• each link has a "weight" $W_{ij}^{(\ell)}$
• each neuron has a "bias" $b_j^{(\ell)}$
We will see that the weights and biases are adjusted as the network "learns".


More notation:
$\vec{a}^{(\ell)} \equiv (a_1^{(\ell)}, a_2^{(\ell)}, \ldots, a_{n_\ell}^{(\ell)})$


as the network "learns".
Activation functions:
Examples:
1 Perceptron (step function)
1
z
learning is difficult because most "learning algorithms" take the gradient of the neurons' output


z
③ tanh(z) = (e^z − e^−z) / (e^z + e^−z)
1
z
−1


z
③ tanh(z) = (e^z − e^−z) / (e^z + e^−z)
1
z
−1
④ Rectified linear unit
ReLU(z) = max(0, z)
z


5 Exponential linear unit
ELU(z) = { α(e^z − 1) if z < 0
z if z ≥ 0
(α > 0 is a hyperparameter)
z

For classification with $d_y > 1$, it is common to apply the softmax activation function to the output layer

$$\left[\text{softmax}(z_1, z_2, \ldots, z_{n_\ell})\right]_i = \frac{e^{z_i}}{\sum_{j=1}^{n_\ell} e^{z_j}} \quad (1 \le i \le n_\ell)$$

Feedfor

x_1

x_2

x_d

For classification with $d_y > 1$, it is common to apply the softmax activation function to the output layer

$$\left[\text{Softmax}(z_1, z_2, \ldots, z_{n_e})\right]_i = \frac{e^{z_i}}{\sum_{j=1}^{n_e} e^{z_j}} \quad (1 \le i \le n_e)$$

Outputs can be interpreted as probabilities

Feedfor

x_1

x_2

x_d