Title: Machine Learning (2021/2022)

Speakers: Lauren Hayward

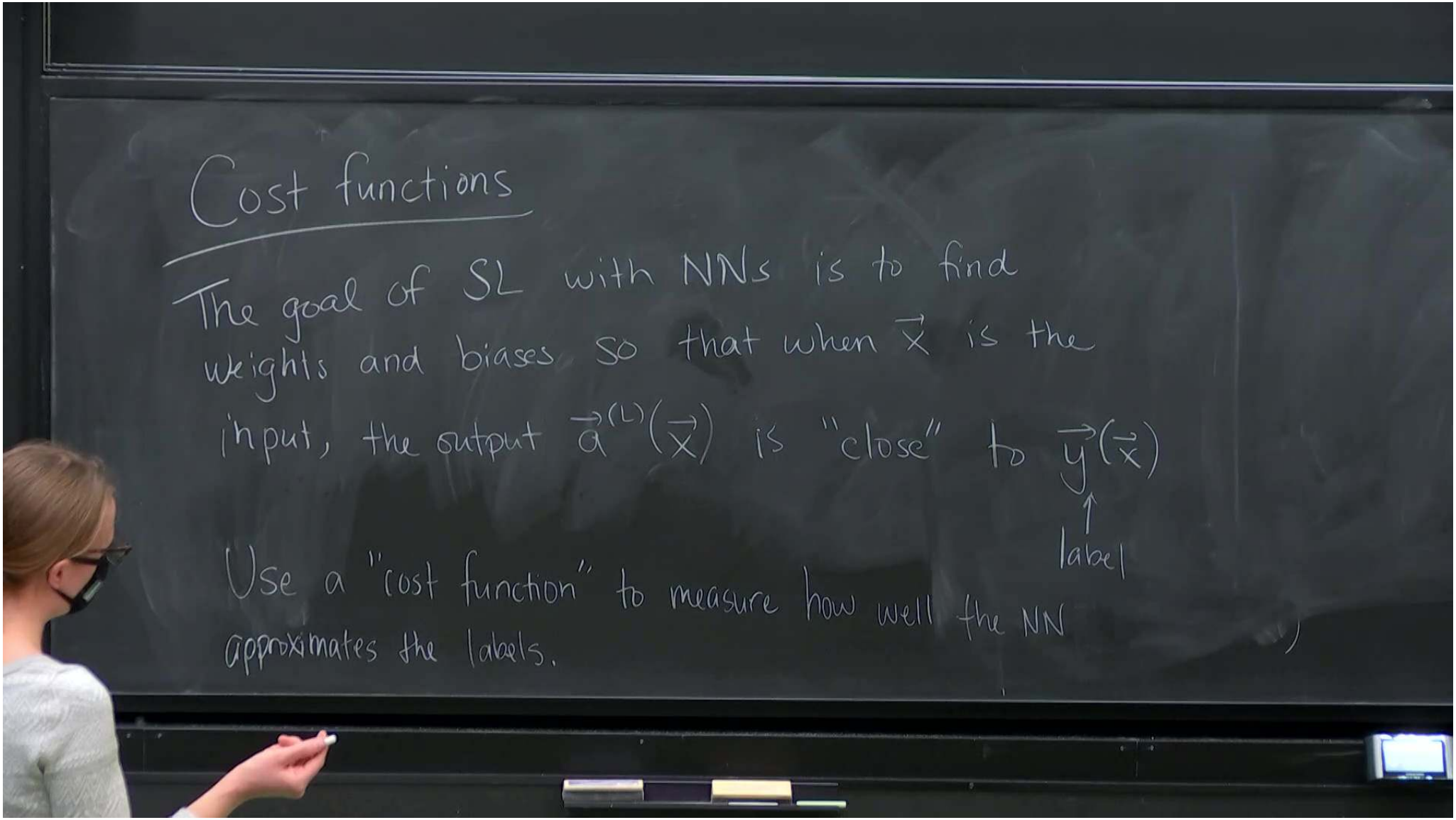
Collection: Machine Learning (2021/2022)

Date: April 12, 2022 - 11:30 AM

URL: https://pirsa.org/22040068

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.

Last time: Architecture of neural networks (NNs) for supervised learning (SL)
Questions from last time:
Outline for today: Basics of NN training ("learning")
• cost functions
• learning algorithms
• backpropagation
Cost functions


Cost functions
The goal of SL with NNs is to find weights and biases so that when $\vec{x}$ is the input, the output $\vec{a}^{(L)}(\vec{x})$ is "close" to $\vec{y}(\vec{x})$
label
Use a "cost function" to measure how well the NN approximates the labels.

Different cost functions are possible, such as:

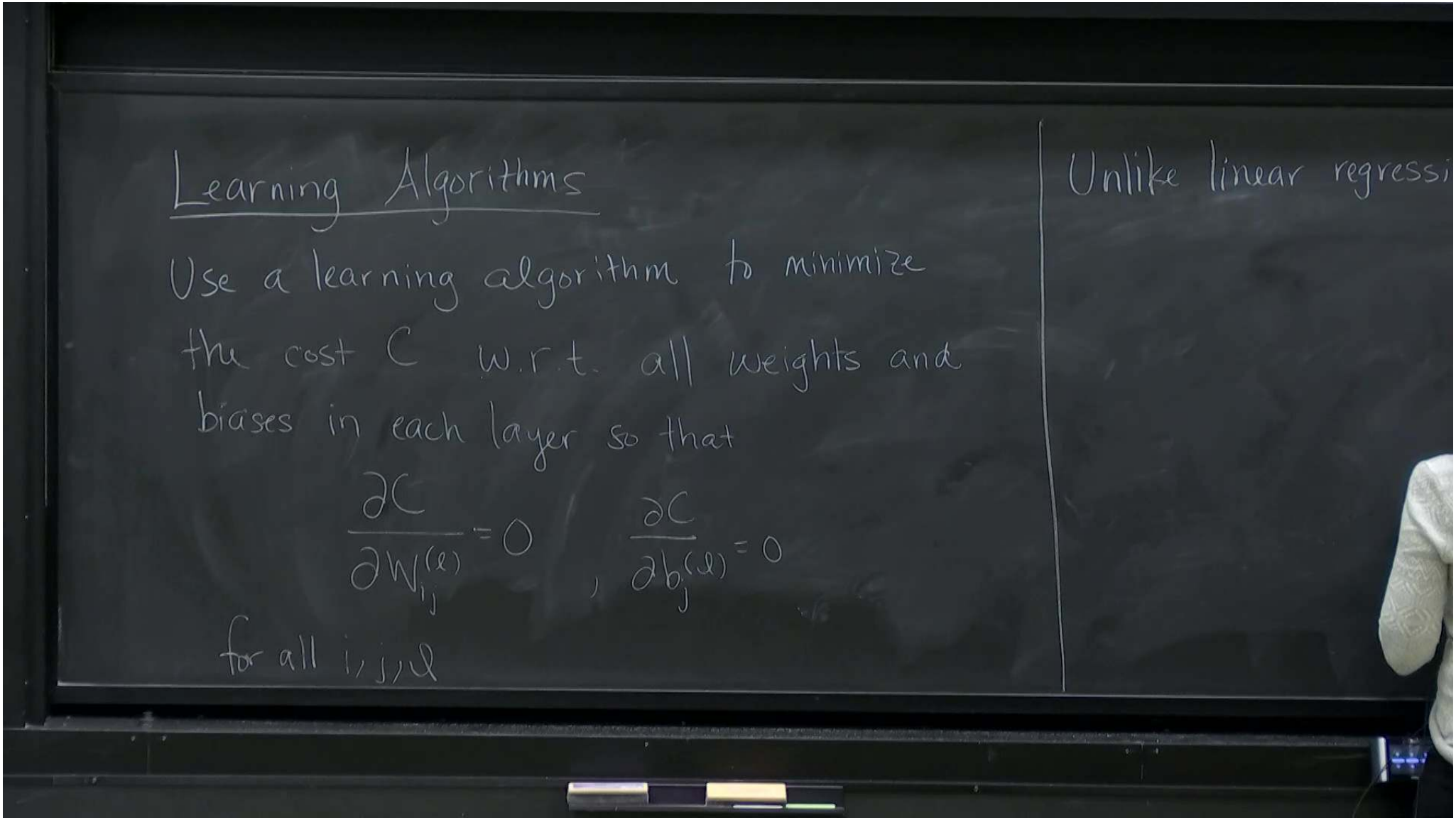
(1) Mean-squared error (MSE)

$$C_{MSE} = \frac{1}{2|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} \underbrace{\sum_{i=1}^{n_L} \left[a_i^{(L)}(\vec{x}) - y_i(\vec{x}) \right]^2}_{= \| \vec{a}^{(L)}(\vec{x}) - \vec{y}(\vec{x}) \|^2}$$

(2) Cross entropy (CE)

(for $0 \le y_i(\vec{x}) \le 1$, $0 \le a_i^{(L)}(\vec{x}) \le 1$)

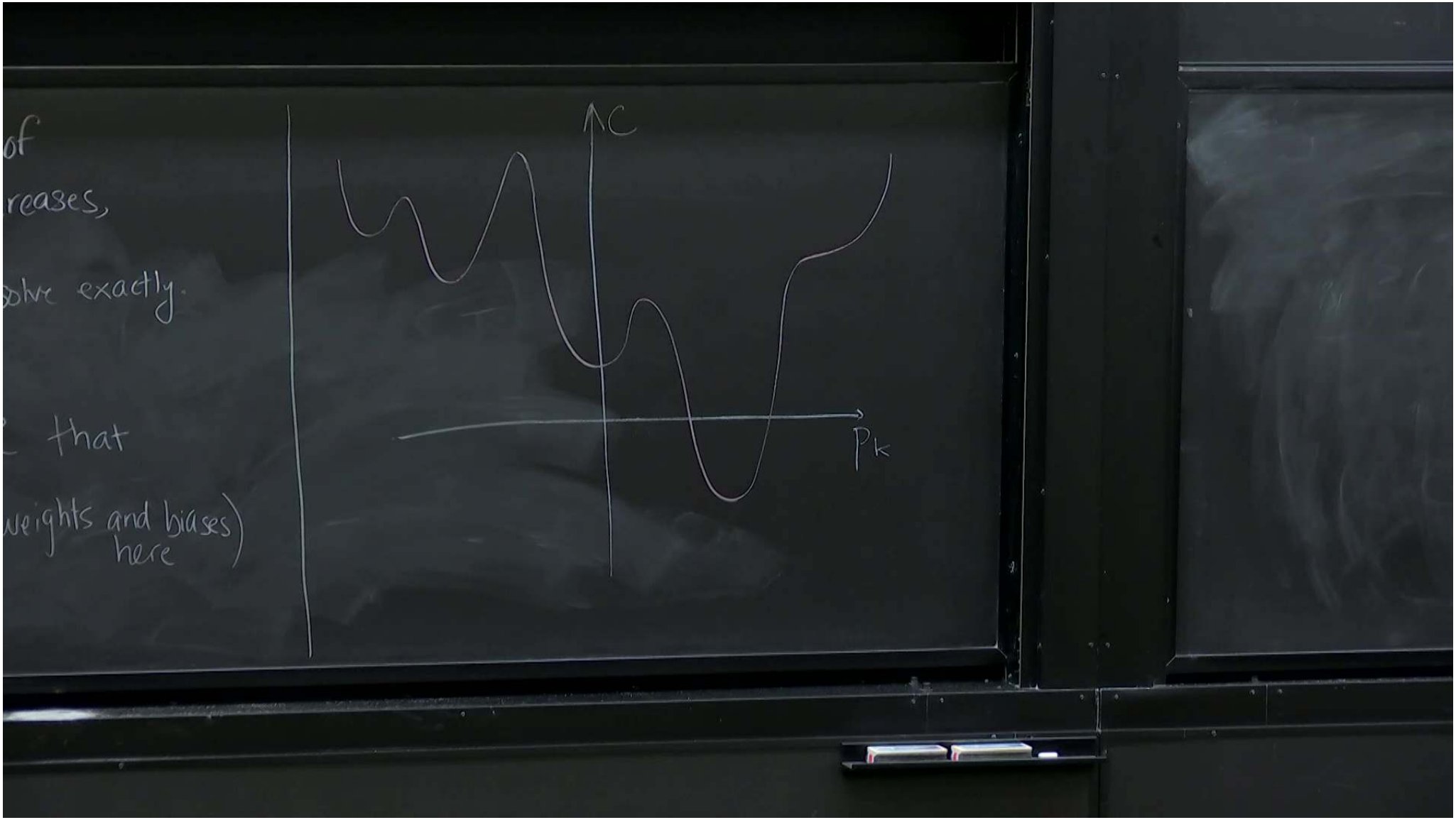
$$C_{CE} = \frac{-1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} \sum_{i=1}^{n_L} \left[y_i(\vec{x}) \log a_i^{(L)}(\vec{x}) + \left(1 - y_i(\vec{x})\right) \log\left(1 - a_i^{(L)}(\vec{x})\right) \right]$$


Learning Algorithms
Use a learning algorithm to minimize the cost C w.r.t. all weights and biases in each layer so that
$\frac{\partial C}{\partial w_{ij}^{(l)}} = 0$, $\frac{\partial C}{\partial b_j^{(l)}} = 0$
for all i, j, l
Unlike linear regressi

Unlike linear regression, as the # of NN parameters (weights & biases) increases, it quickly becomes impossible to solve exactly.

More generally, consider a cost C that depends on params. $p_1, p_2, \ldots, p_N$ (weights and biases here)

We want $\frac{\partial C}{\partial p_k} = 0$ for $1 \leq k \leq N$


of
reases,
olve exactly.
that
eights and biases)
here
C
P_k

for all i, j, k

We want $\frac{\partial C}{\partial p_k} =$

Recall gradient descent (GD): update the parameters as:

$$\vec{p} \rightarrow \vec{p} - \eta \vec{\nabla}_{\vec{p}} C$$

η is the learning rate hyperparam

Let's examine some alternatives/improvements

Stochastic

Assume C

We want $\frac{\partial C}{\partial p_k} = 0$ for $1 \leq k \leq N$

Stochastic GD (SGD)

Assume C can be written as $C = \frac{1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} c(\vec{x})$ (MSE and CE both satisfy this)

Idea: appoximate C and $\vec{\nabla}_{\vec{p}} C$ by summing over a random "mini-batch" $\mathcal{B}$ of $M_{\mathcal{B}}$ data points (instead of the whole dataset).

Benefits:
- faster
- introduces randomness, which can decrease the chance of getting stuck in a local min.

Adding momentum

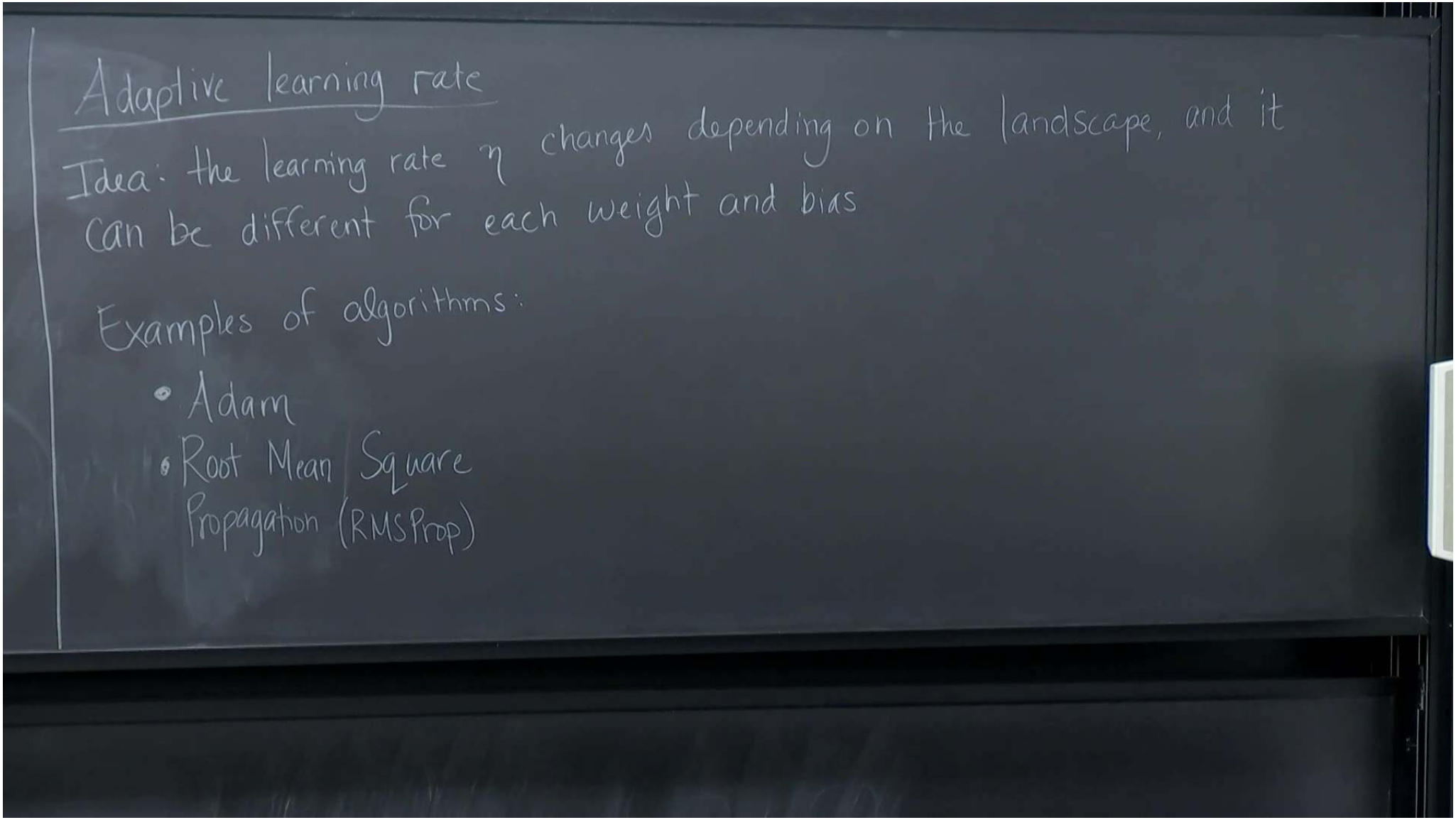
Idea: keep some "memory" of previous gradients

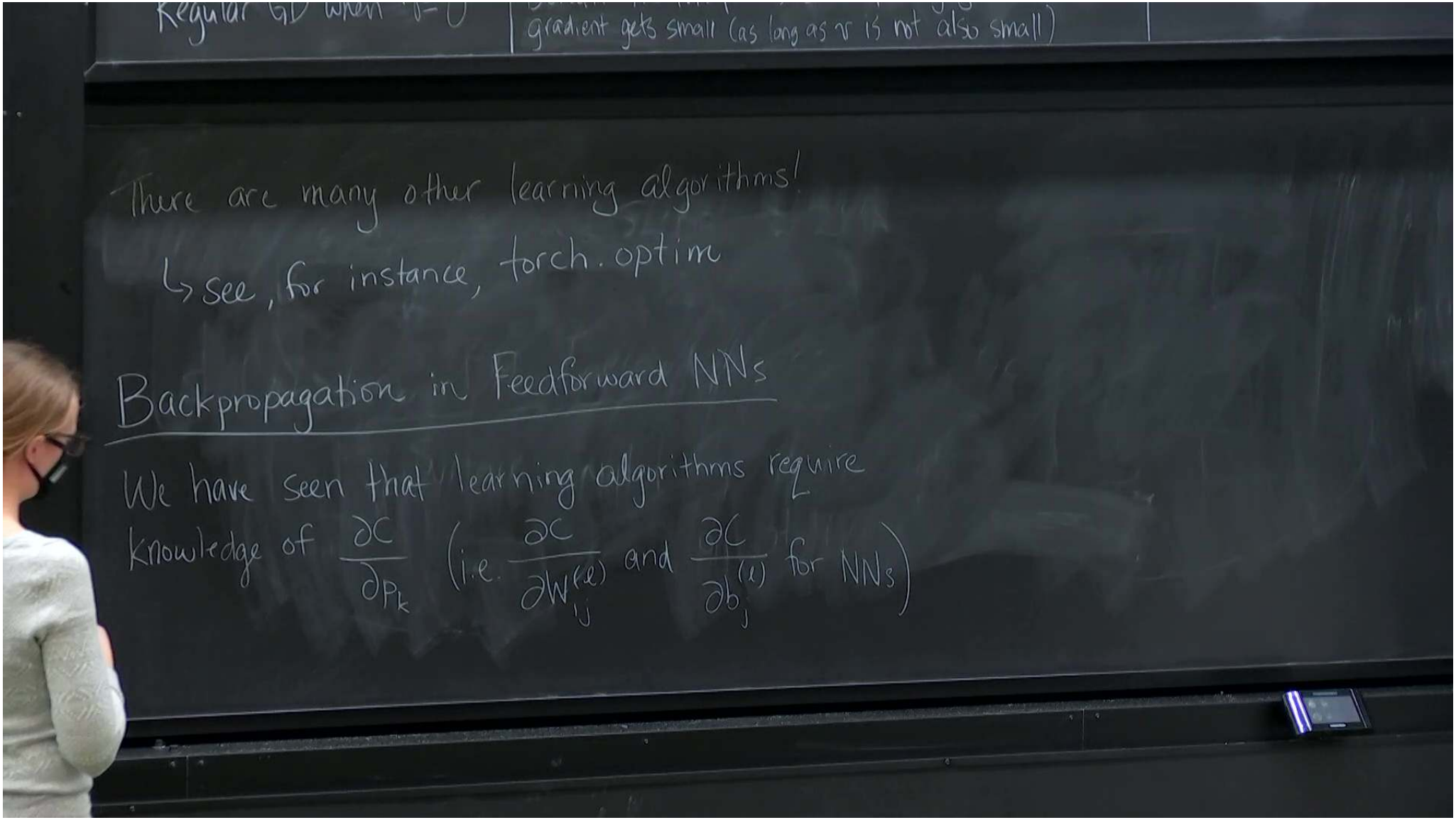
update: $\vec{v} \longrightarrow \gamma \vec{v} - \eta \vec{\nabla}_{\vec{p}} C$

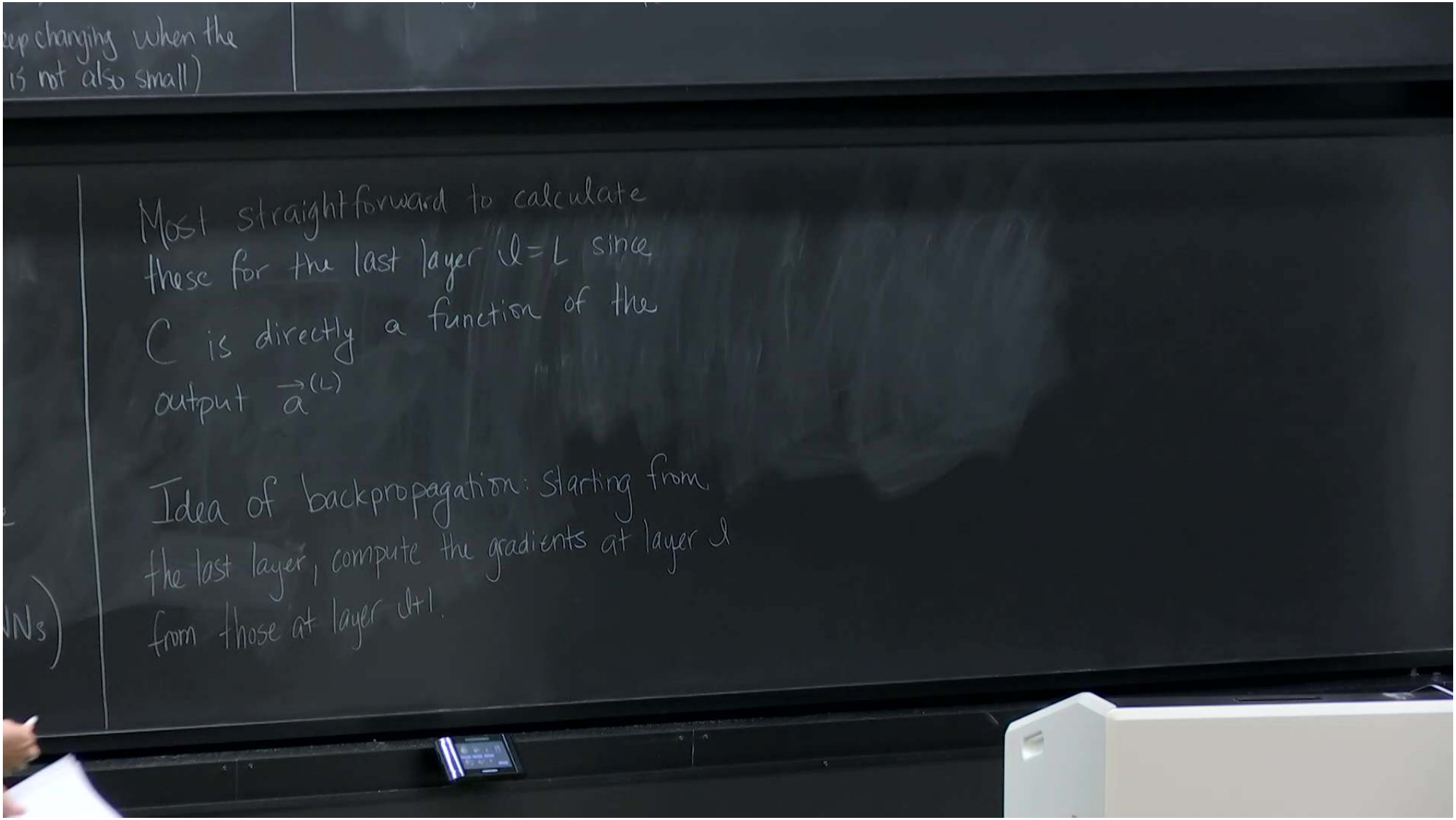
$\vec{p} \longrightarrow \vec{p} + \vec{v}$

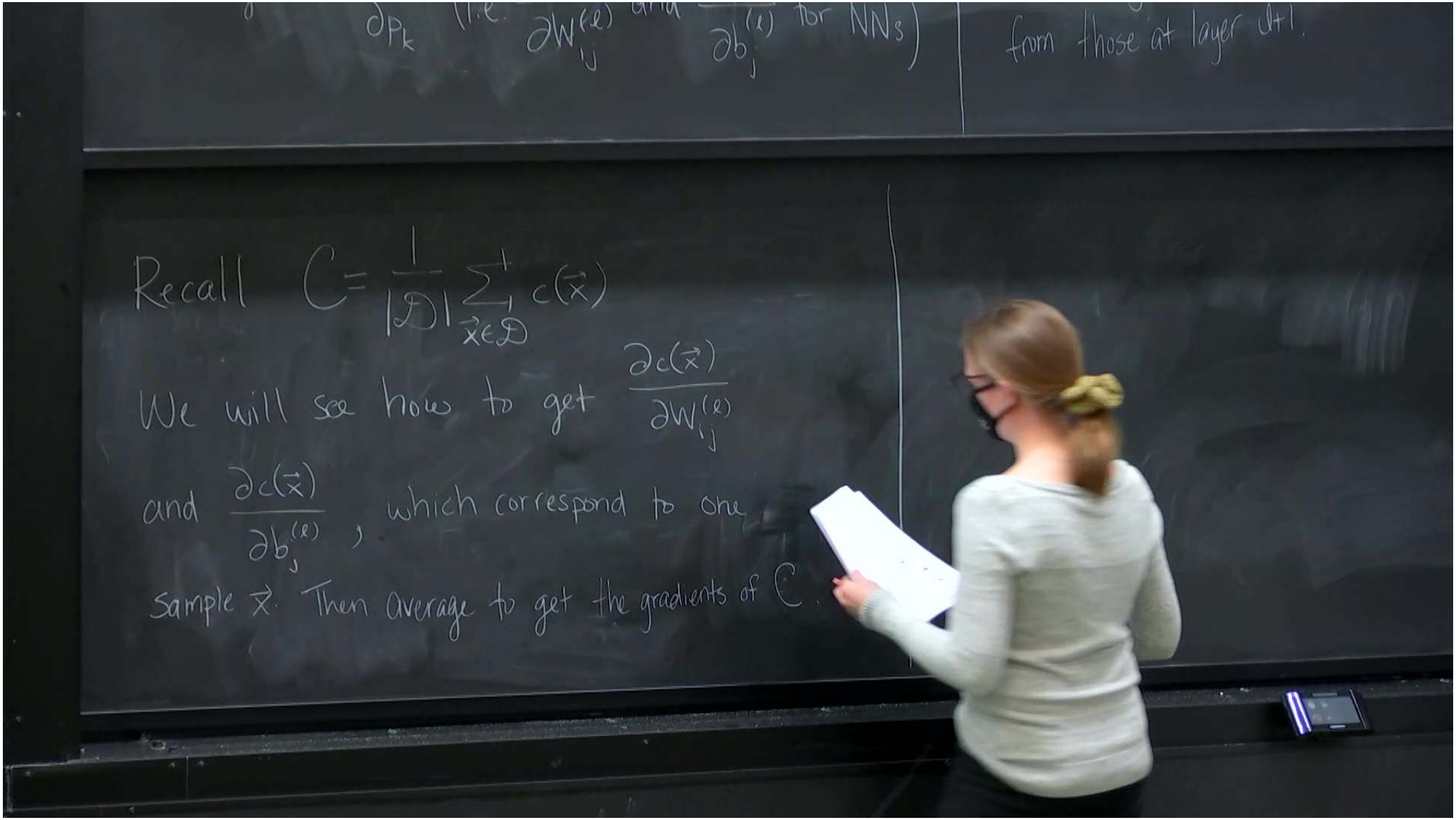
where γ is the "momentum hyperparam" $(0 \leq \gamma < 1)$

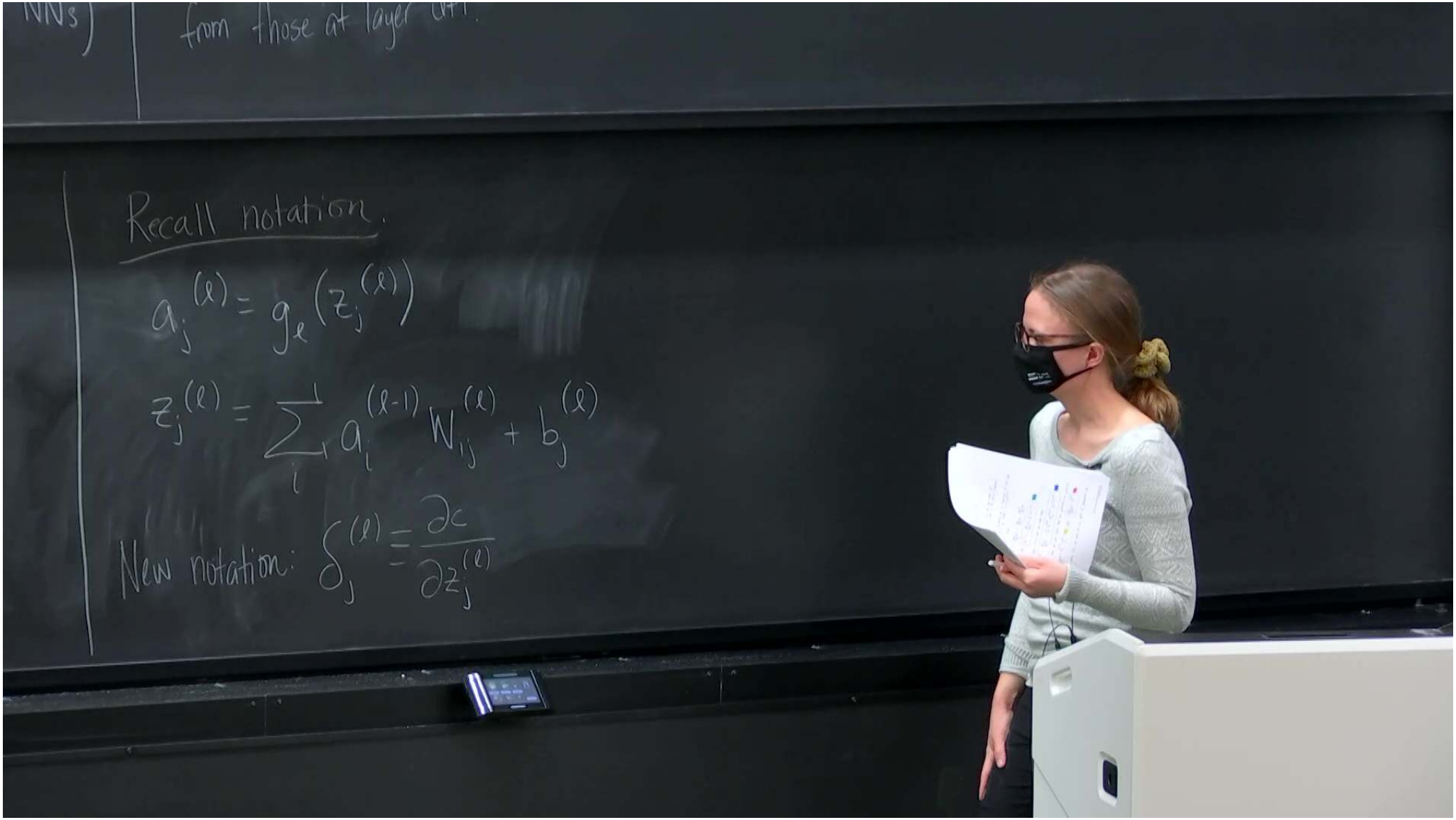
Regular GD when $\gamma = 0$ | Benefit: the NN params. can keep changing when the gradient gets small (as long as v is not also small)

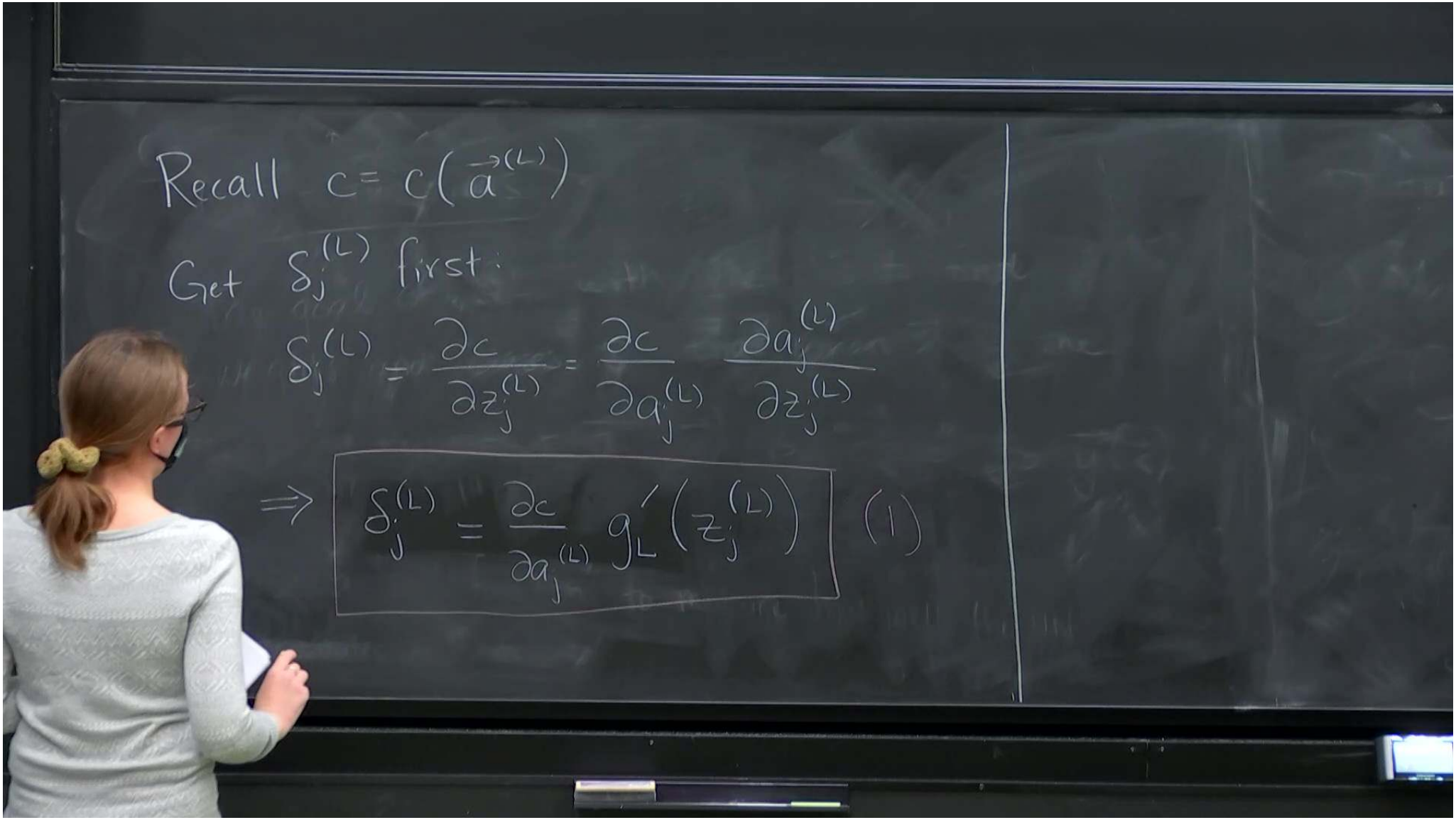

Adaptive learning rate
Idea: the learning rate η changes depending on the landscape, and it can be different for each weight and bias
Examples of algorithms:
• Adam
• Root Mean Square Propagation (RMSProp)


Regular GD when
gradient gets small (as long as r is not also small)
There are many other learning algorithms!
↳ see, for instance, torch.optim
Backpropagation in Feedforward NNs
We have seen that learning algorithms require
knowledge of $\frac{\partial C}{\partial p_k}$ (i.e. $\frac{\partial C}{\partial w_{ij}^{(\ell)}}$ and $\frac{\partial C}{\partial b_j^{(\ell)}}$ for NNs)


ep changing when the
is not also small)
Most straightforward to calculate
these for the last layer $l = L$ since
C is directly a function of the
output $\vec{a}^{(L)}$
Idea of backpropagation: starting from
the last layer, compute the gradients at layer l
from those at layer $l+1$.


∂P_k (i.e. $\frac{\partial W_{ij}^{(\ell)}}{\partial b_j^{(\ell)}}$ for NNs)
from those at layer $\ell+1$.
Recall $C = \frac{1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} c(\vec{x})$
We will see how to get $\frac{\partial c(\vec{x})}{\partial W_{ij}^{(\ell)}}$
and $\frac{\partial c(\vec{x})}{\partial b_j^{(\ell)}}$, which correspond to one
sample $\vec{x}$. Then average to get the gradients of C.


NNs)
from those at layer
Recall notation.
$a_j^{(\ell)} = g_\ell(z_j^{(\ell)})$
$z_j^{(\ell)} = \sum_i a_i^{(\ell-1)} W_{ij}^{(\ell)} + b_j^{(\ell)}$
New notation: $\delta_j^{(\ell)} = \frac{\partial c}{\partial z_j^{(\ell)}}$

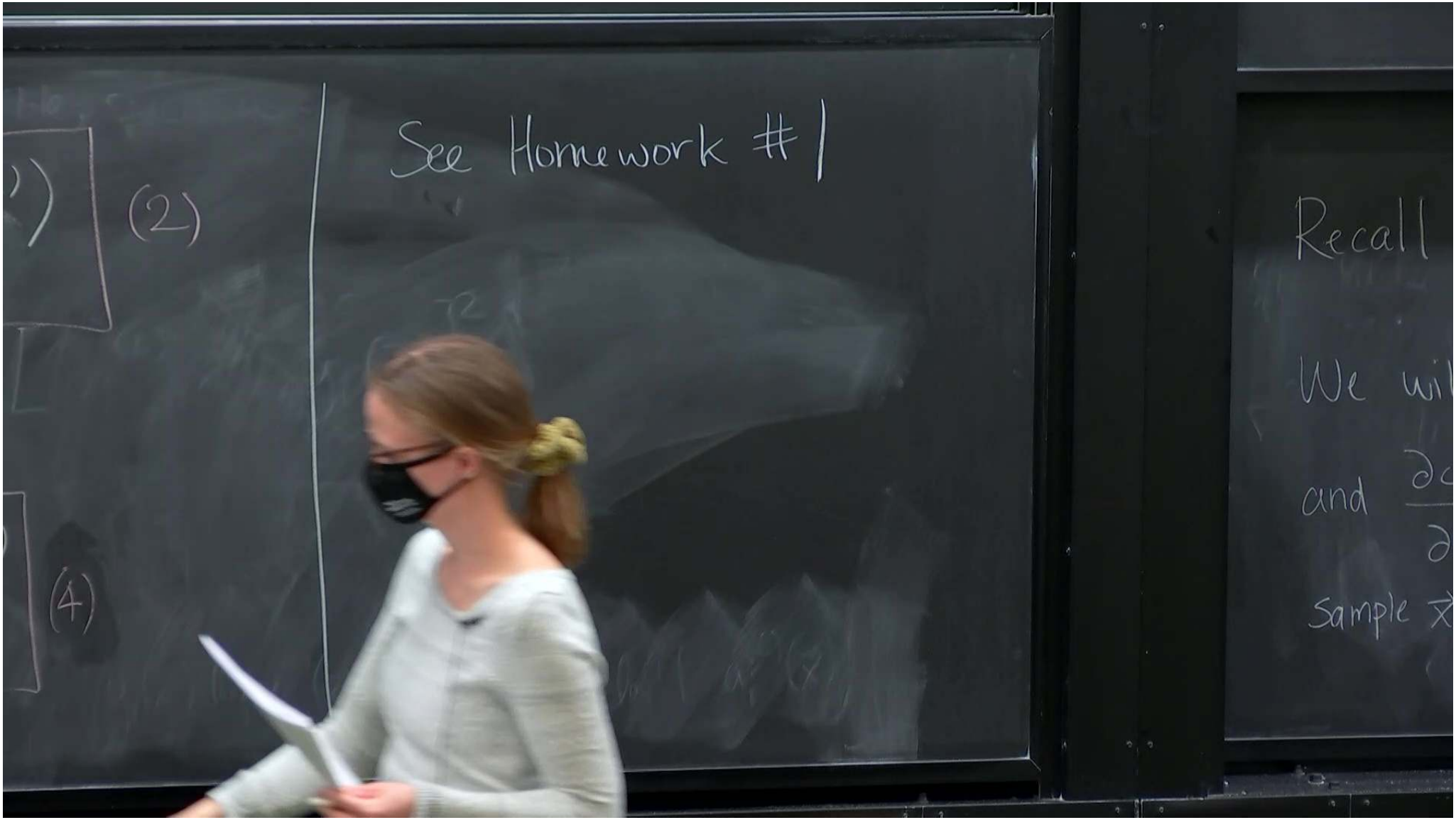

Recall $c = c(\vec{a}^{(L)})$
Get $\delta_j^{(L)}$ first:
$\delta_j^{(L)} = \frac{\partial c}{\partial z_j^{(L)}} = \frac{\partial c}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$
$\Rightarrow \boxed{\delta_j^{(L)} = \frac{\partial c}{\partial a_j^{(L)}} g_L'(z_j^{(L)})}$ (1)

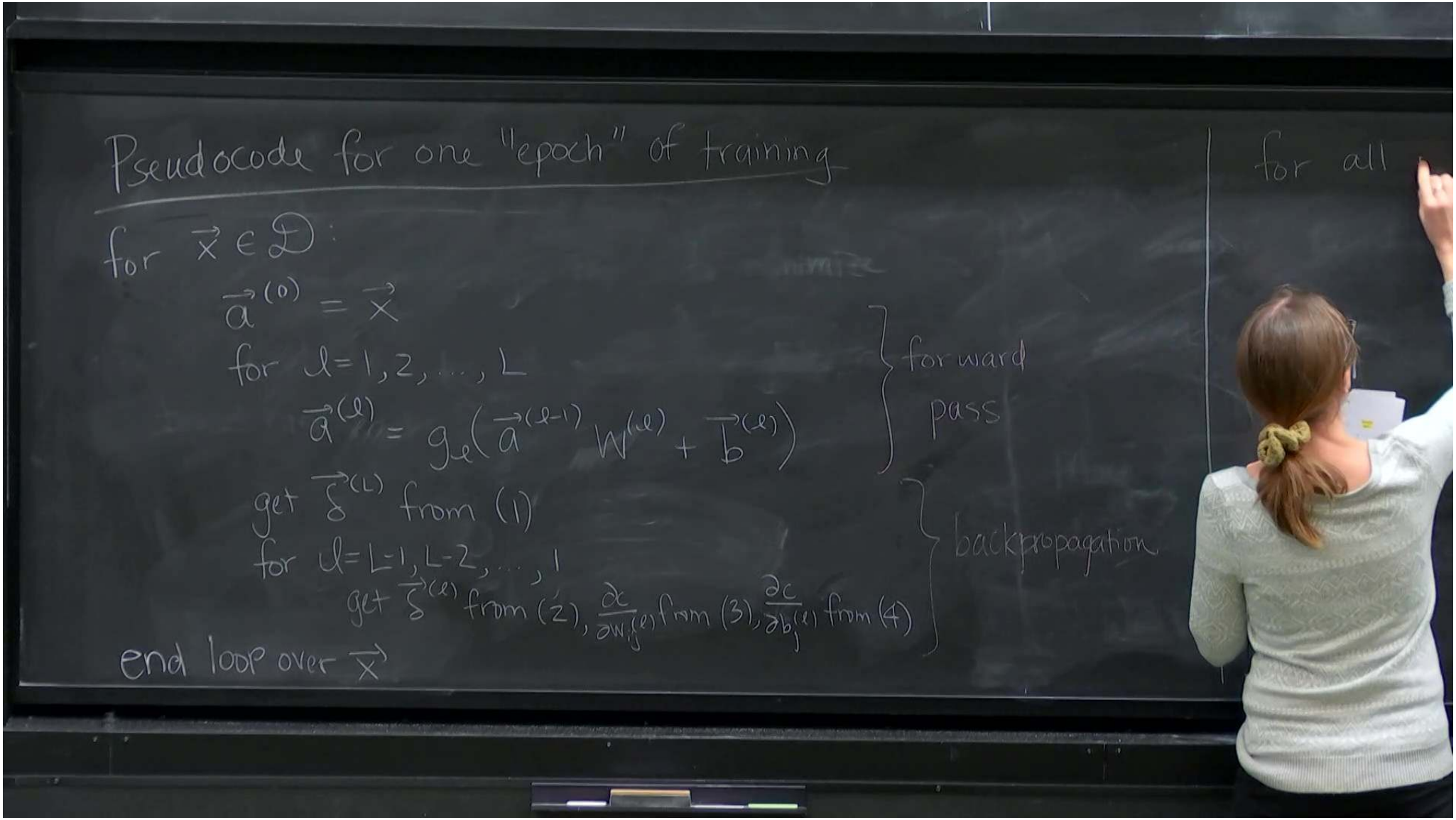
For layer $l<L$, one can show:

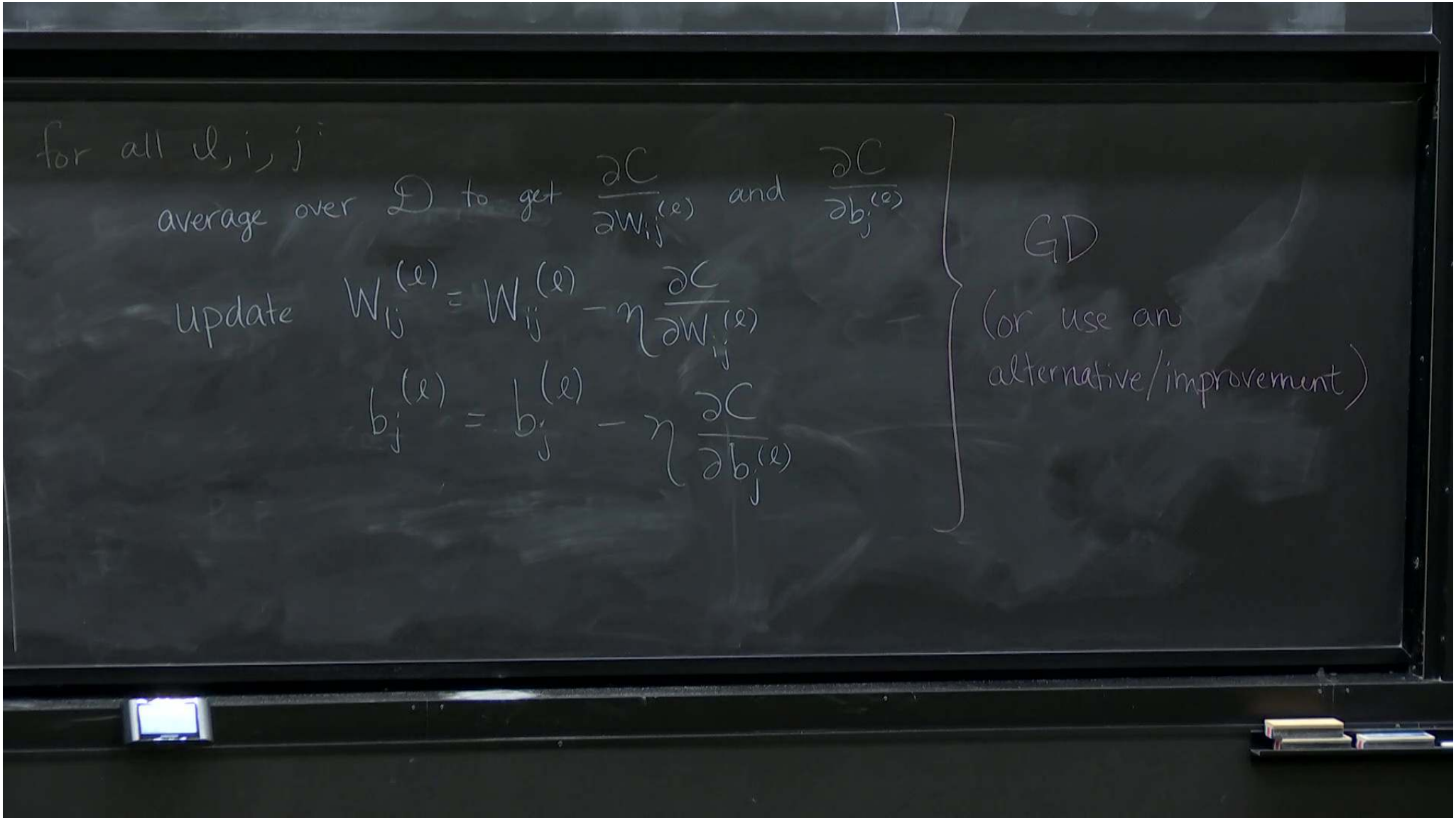
$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_k \delta_k^{(l+1)} W_{kj}^{(l+1)\,T} g_l'(z_j^{(l)}) \quad (2)$$

Then:

$$\frac{\partial C}{\partial W_{ij}^{(l)}} = a_i^{(l-1)} \delta_j^{(l)} \quad (3) \quad \text{and} \quad \frac{\partial C}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad (4)$$


See Homework #1
(2)
(4)
Recall
We wi
and
Sample


Pseudocode for one "epoch" of training
for $\vec{x} \in \mathcal{D}$:
$\vec{a}^{(0)} = \vec{x}$
for $l = 1, 2, \ldots, L$
$\vec{a}^{(l)} = g_l(\vec{a}^{(l-1)} W^{(l)} + \vec{b}^{(l)})$
forward pass
get $\vec{\delta}^{(L)}$ from (1)
for $l = L-1, L-2, \ldots, 1$
get $\vec{\delta}^{(l)}$ from (2), $\frac{\partial C}{\partial w_{jk}^{(l)}}$ from (3), $\frac{\partial C}{\partial b_j^{(l)}}$ from (4)
backpropagation
end loop over $\vec{x}$
for all


for all ℓ, i, j
average over $\mathcal{D}$ to get $\frac{\partial C}{\partial w_{ij}^{(\ell)}}$ and $\frac{\partial C}{\partial b_j^{(\ell)}}$
update $w_{ij}^{(\ell)} = w_{ij}^{(\ell)} - \eta \frac{\partial C}{\partial w_{ij}^{(\ell)}}$
$b_j^{(\ell)} = b_j^{(\ell)} - \eta \frac{\partial C}{\partial b_j^{(\ell)}}$
GD
(or use an alternative/improvement)