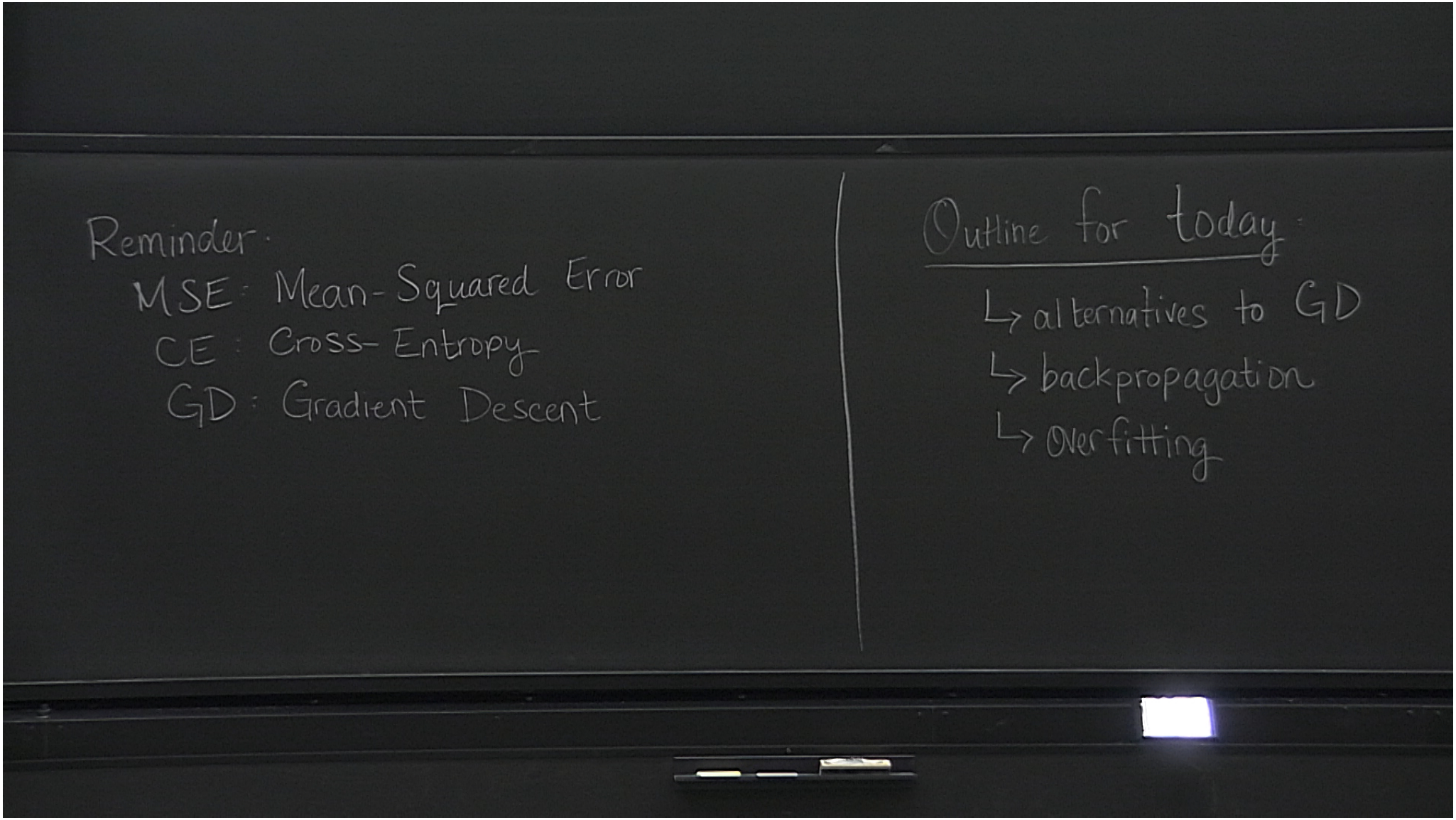
Title: PSI 2017/2018 - Machine Learning for Many Body Physics - Lecture 3

Date: Apr 11, 2018 09:00 AM

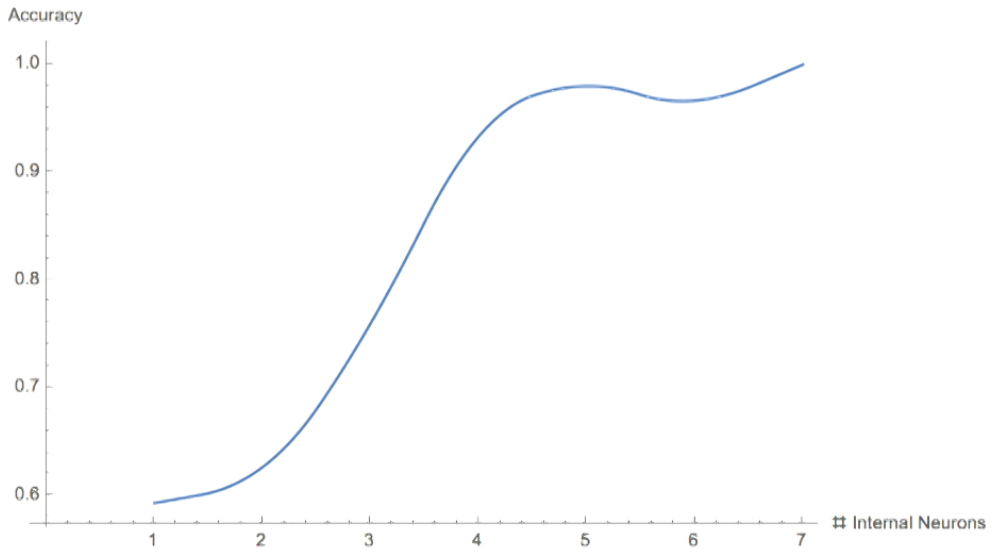
URL: http://pirsa.org/18040056

Abstract:


Reminder:
MSE: Mean-Squared Error
CE: Cross-Entropy
GD: Gradient Descent
Outline for today:
↳ alternatives to GD
↳ backpropagation
↳ Overfitting

Group 1

Accuracy vs. the number of neurons in the hidden layer

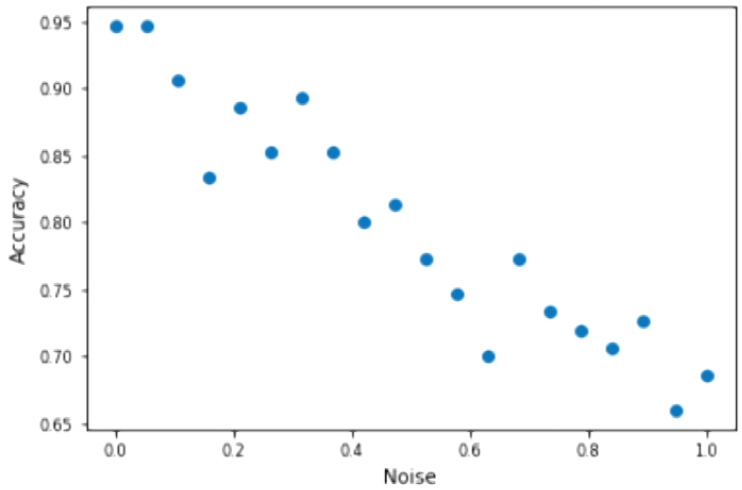


Carlos, Cheryne, Kfir, Vladimir

PERIMETER SCHOLARS INTERNATIONAL

Group 2

Accuracy vs. the magnitude of noise in the data

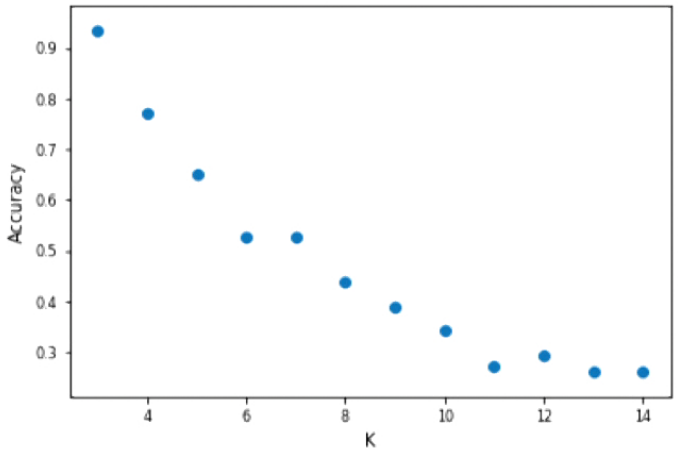


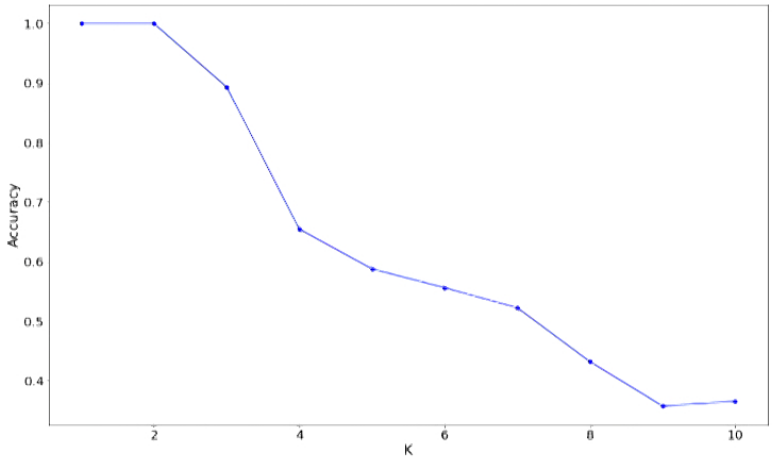
Ahmed, Gwyneth, Lei, Matt

PERIMETER SCHOLARS INTERNATIONAL

Group 3

Accuracy vs. the number of different labels (branches)

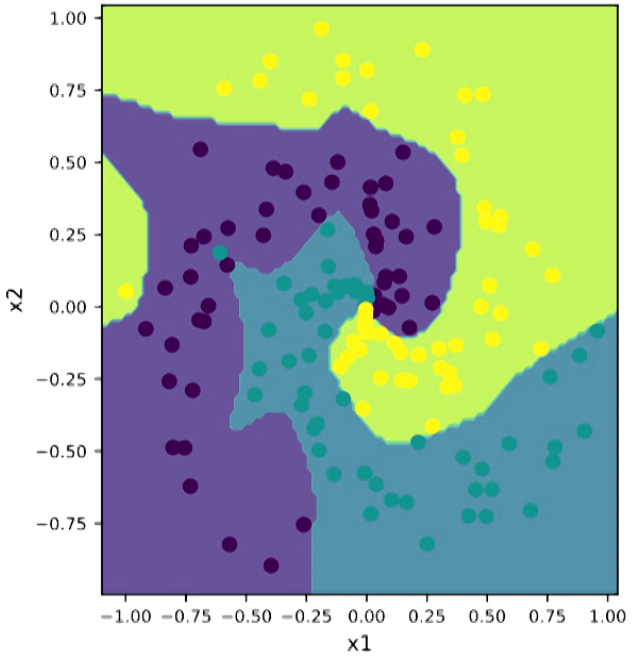


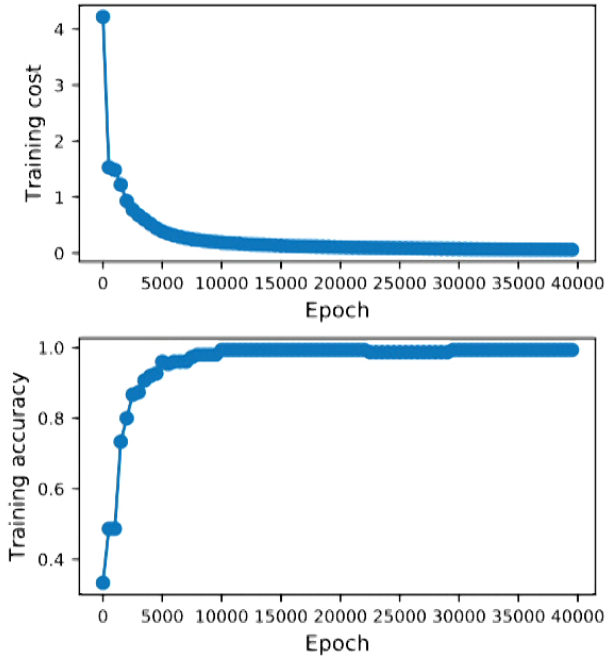


Lusine, Qi, Stavros, TC

PERIMETER SCHOLARS INTERNATIONAL

```
mag_noise = 0.5
nH = 100
```





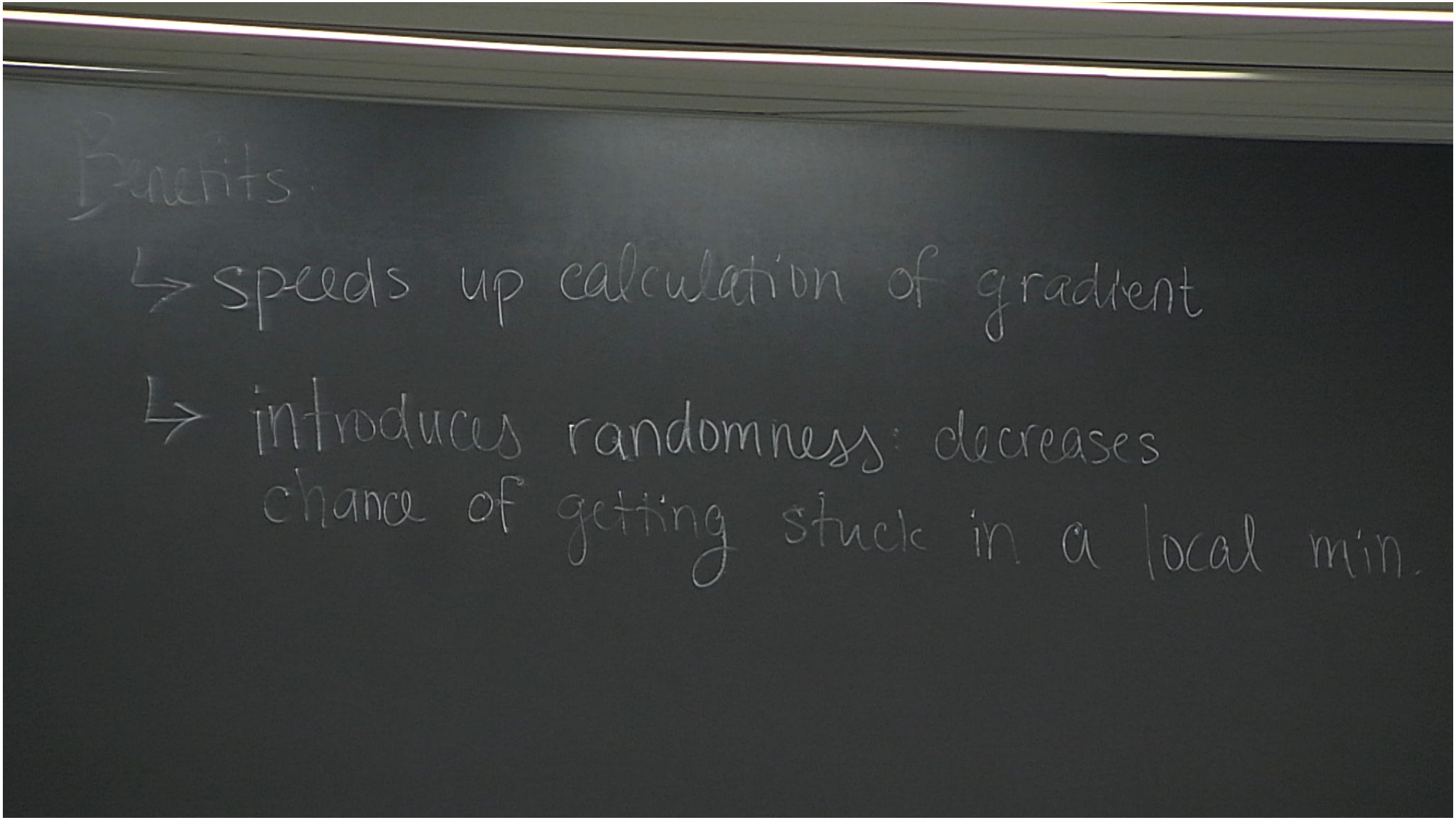
PERIMETER SCHOLARS INTERNATIONAL

Alternatives to GD

(1) Stochastic GD (SGD)

Assume $C = \frac{1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} c(\vec{x})$

Idea: Approximate the gradient $\vec{\nabla}_{\vec{p}} C$ by summing over a randomly-chosen "mini-batch" $\mathcal{B}$ of M samples (instead of summing over all of the dataset $\mathcal{D}$)

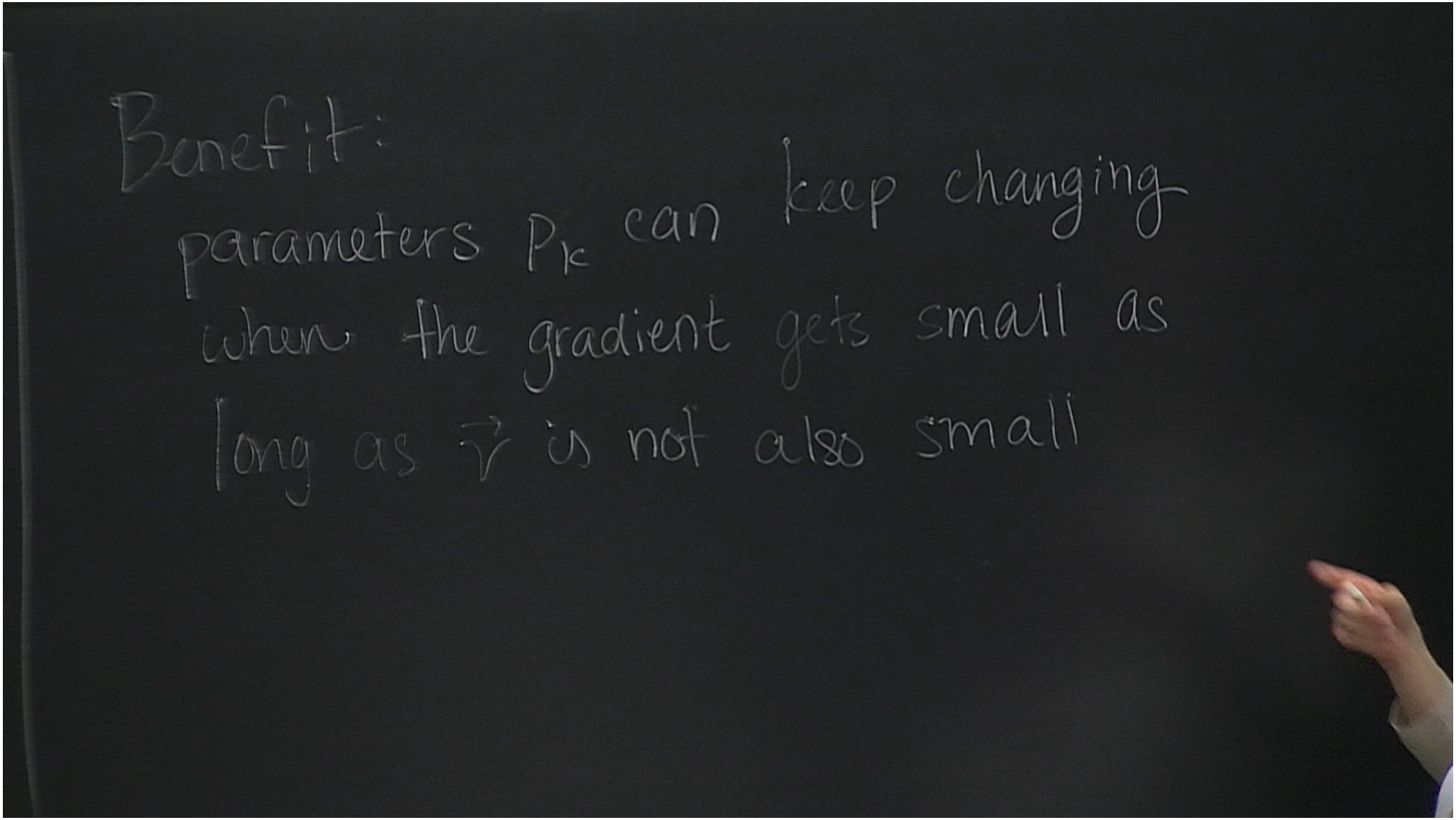

Benefits:
↳ speeds up calculation of gradient
↳ introduces randomness: decreases chance of getting stuck in a local min.

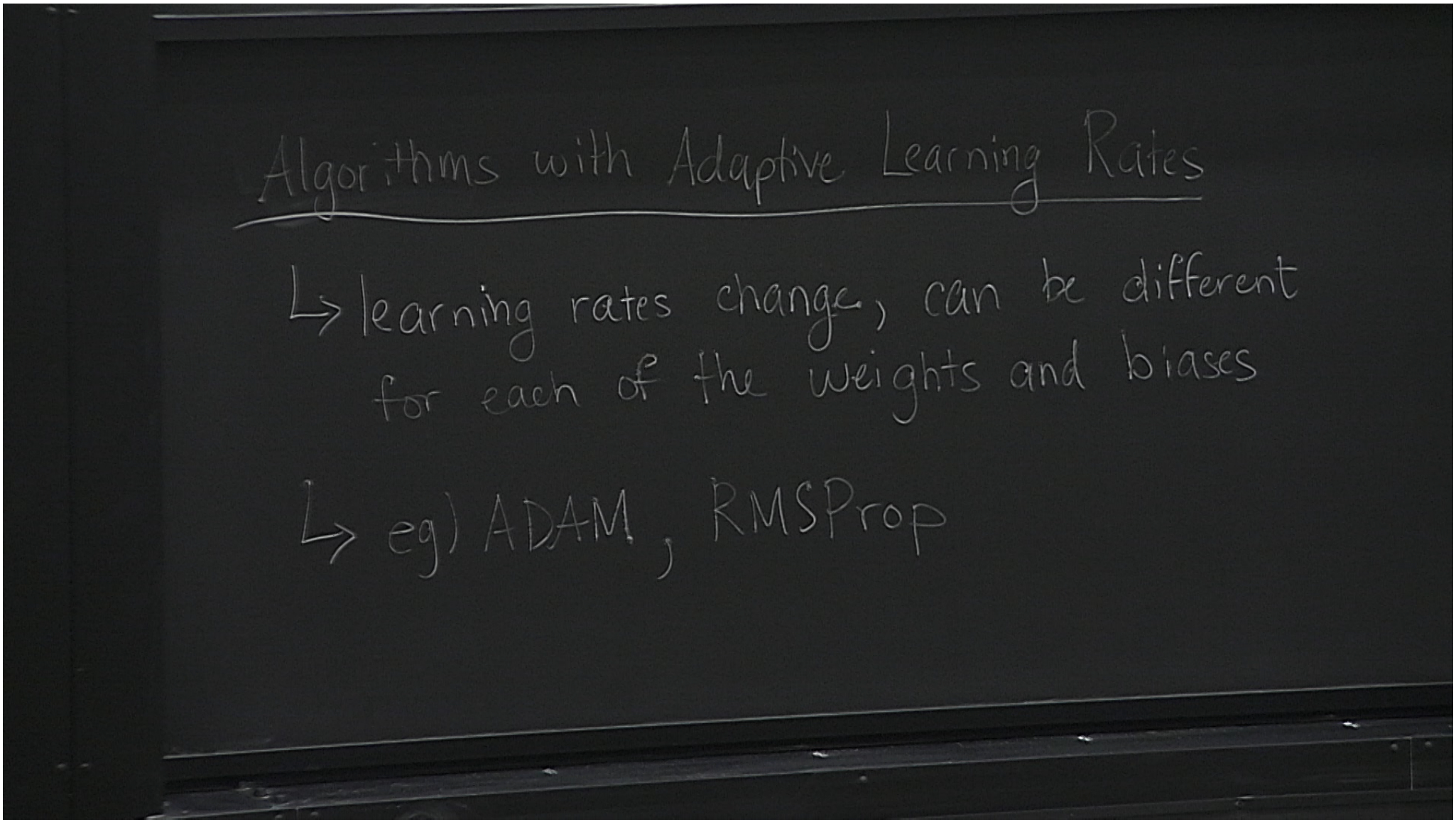
Adding momentum

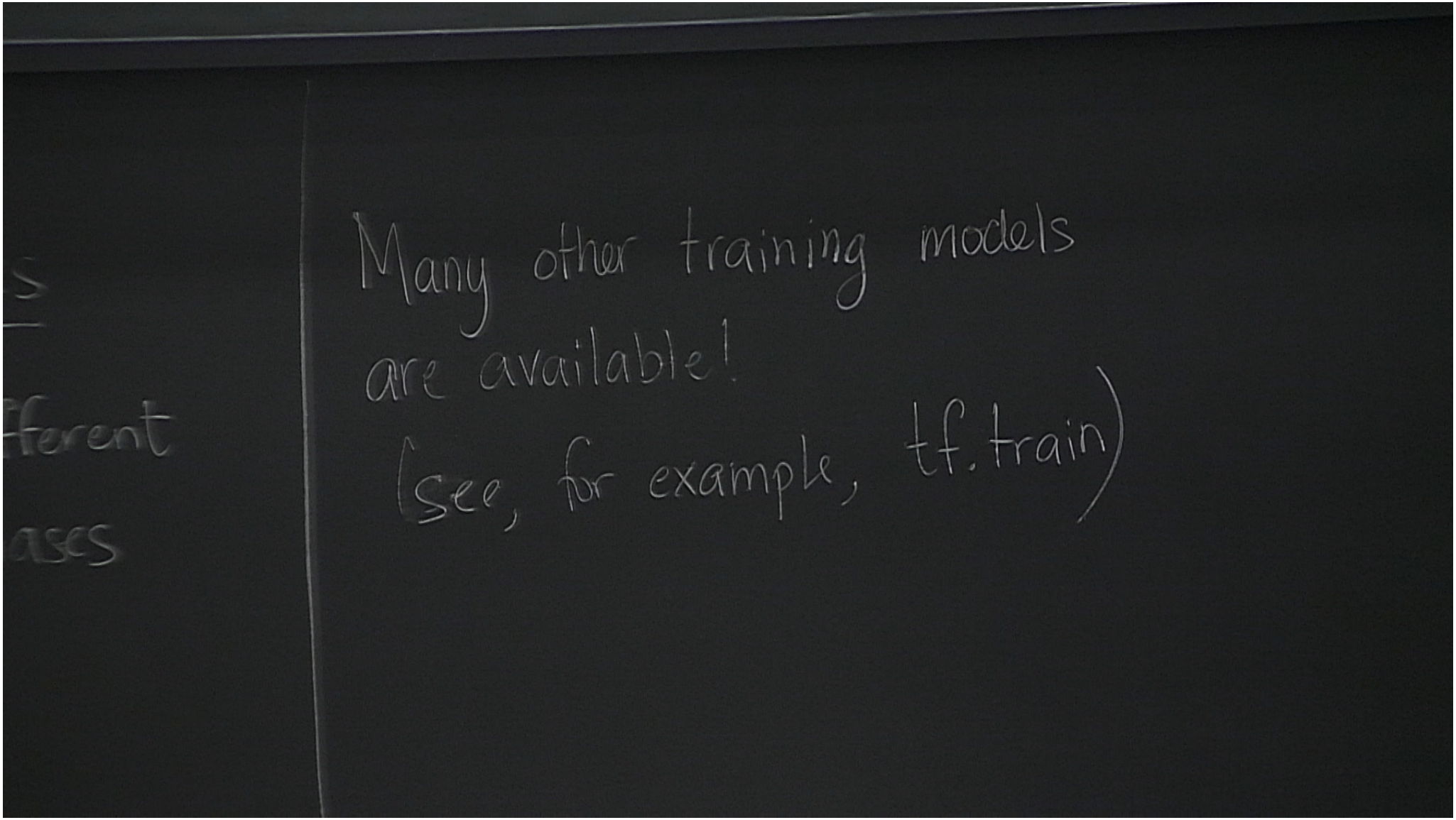
↳ keep some "memory" $\vec{v}$ of previous gradients

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

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


Benefit:
parameters p_k can keep changing
when the gradient gets small as
long as $\vec{v}$ is not also small


Algorithms with Adaptive Learning Rates
↳ learning rates change, can be different for each of the weights and biases
↳ eg) ADAM, RMSProp


Many other training models
are available!
(see, for example, tf.train)

Backpropagatation

↳ used to calculate the gradients needed for training

Recall $C = \frac{1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} c(\vec{x})$

We will see how to calculate $\frac{\partial c(\vec{x})}{\partial W_{ij}^{(\ell)}}$ and $\frac{\partial c(\vec{x})}{\partial b_j^{(\ell)}}$ (corresponding to one training sample $\vec{x}$).

Then we average over many samples to get gradients of C

Recall: $a_j^{(\ell)} = g_\ell(z_j^{(\ell)})$

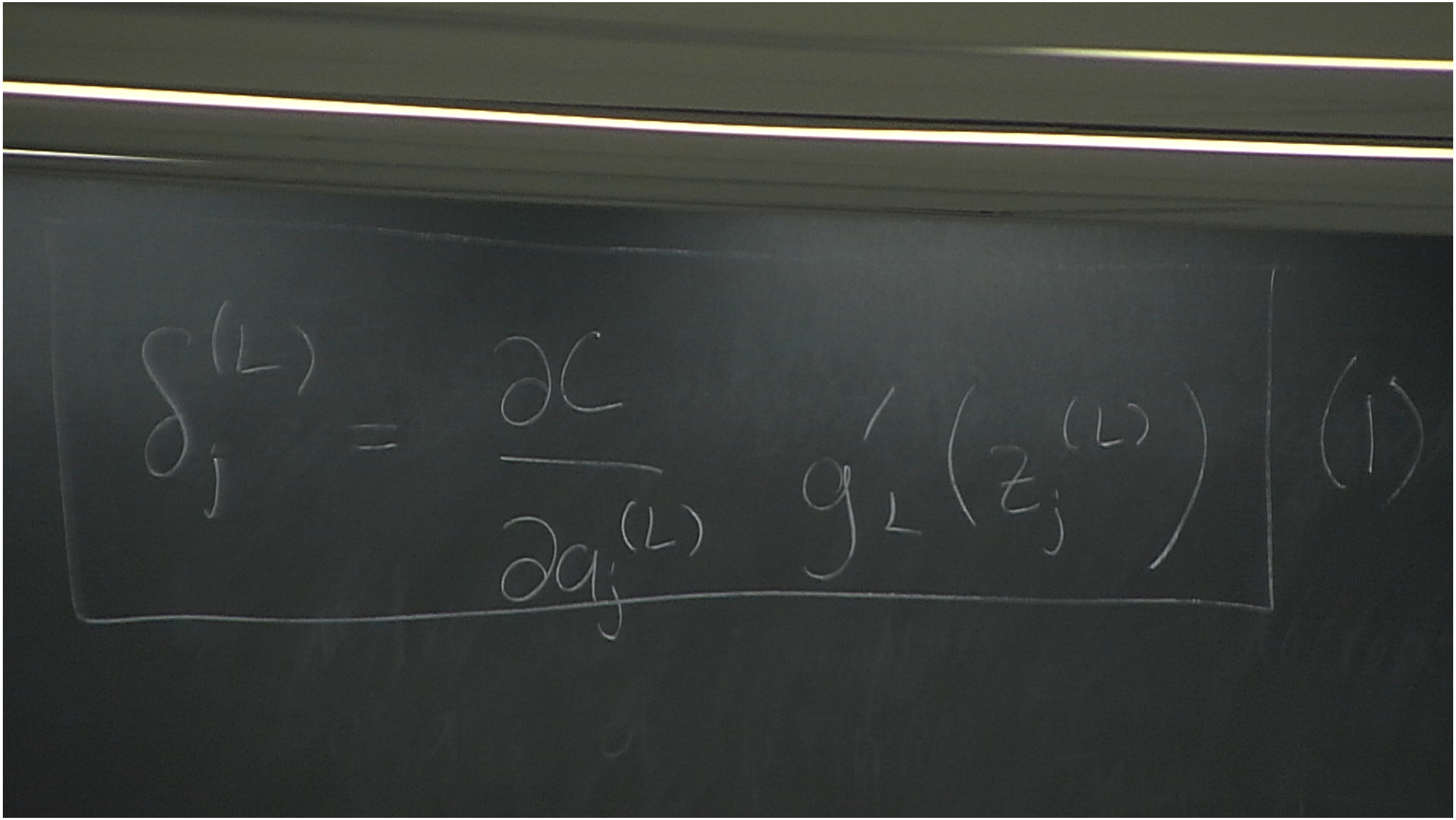
$$z_j^{(\ell)} = \sum_i a_i^{(\ell-1)} W_{ij}^{(\ell)} + b_j^{(\ell)}$$

esponding

Define $\delta_j^{(\ell)} = \frac{\partial C}{\partial z_j^{(\ell)}}$

C is written as a function of the last layer's output $a^{(L)}$: $C = C(a^{(L)})$

For the last layer: $\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = \frac{\partial C}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$


$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} g'_L(z_j^{(L)})$ (1)

$$\frac{\partial z_j^{(L)}}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$$

For layer $l<L$:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_k \frac{\partial C}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \ldots$$

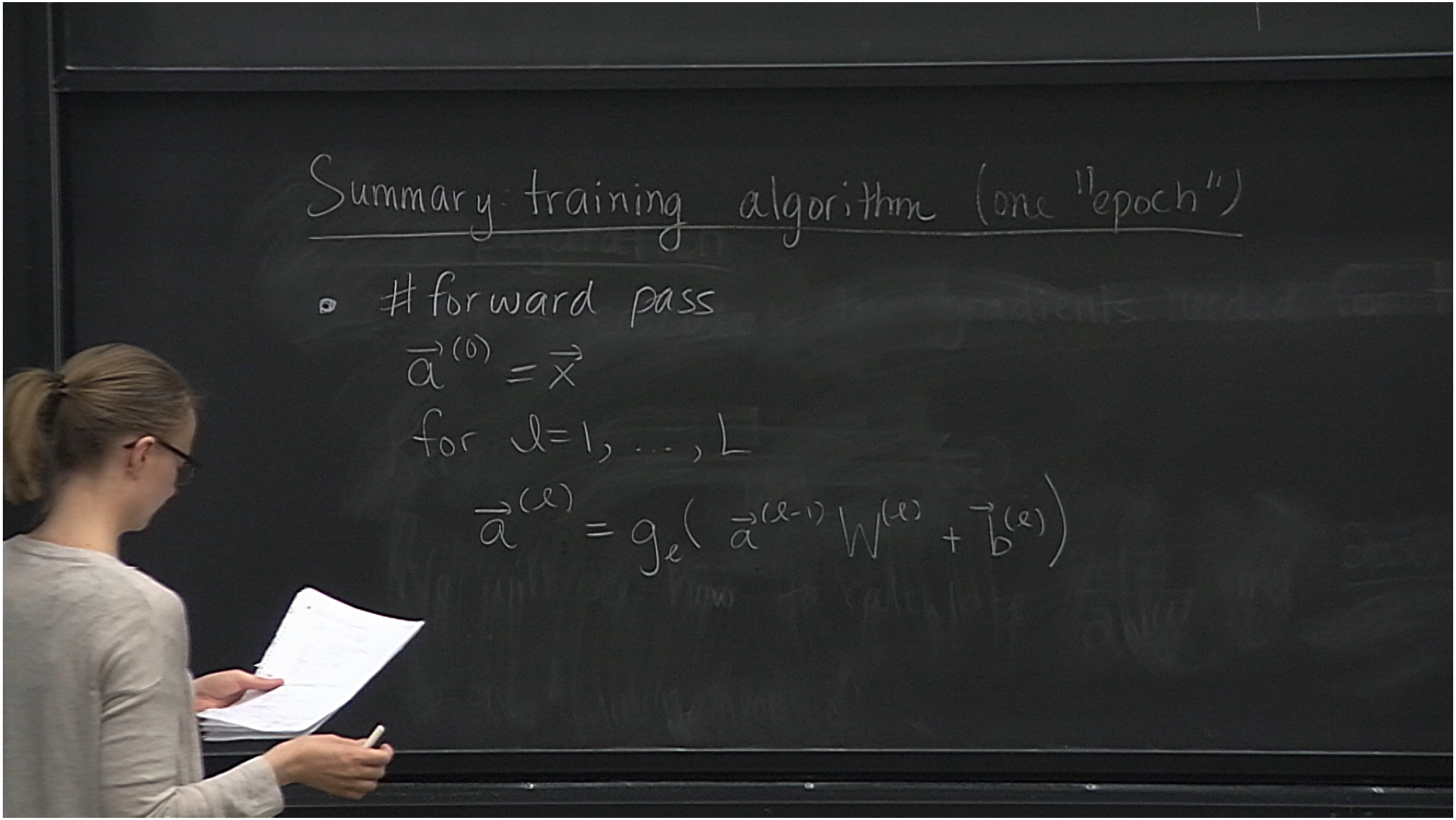
show

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

Now one can show that

$$\frac{\partial C}{\partial w_{ij}^{(\ell)}} = a_i^{(\ell-1)} \delta_j^{(\ell)} \quad (3)$$

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)} \quad (4)$$


Summary: training algorithm (one "epoch")
forward pass
$\vec{a}^{(0)} = \vec{x}$
for $l = 1, \ldots, L$
$\vec{a}^{(l)} = g_l(\vec{a}^{(l-1)} W^{(l)} + \vec{b}^{(l)})$

- #backpropagation

  get $\vec{\delta}^{(L)}$ from (1)

  for $l = L-1, \ldots, 1$:

  get $\vec{\delta}^{(l)}$ from (2)

  get $\frac{\partial C}{\partial W_{ij}^{(l)}}$ and $\frac{\partial C}{\partial b_j^{(l)}}$ from (3) & (4)

- #update: $W_{ij}^{(l)} = W_{ij}^{(l)} - \eta \frac{\partial C}{\partial W_{ij}^{(l)}}$

  $b_j^{(l)} = b_j^{(l)} - \eta \frac{\partial C}{\partial b_j^{(l)}}$

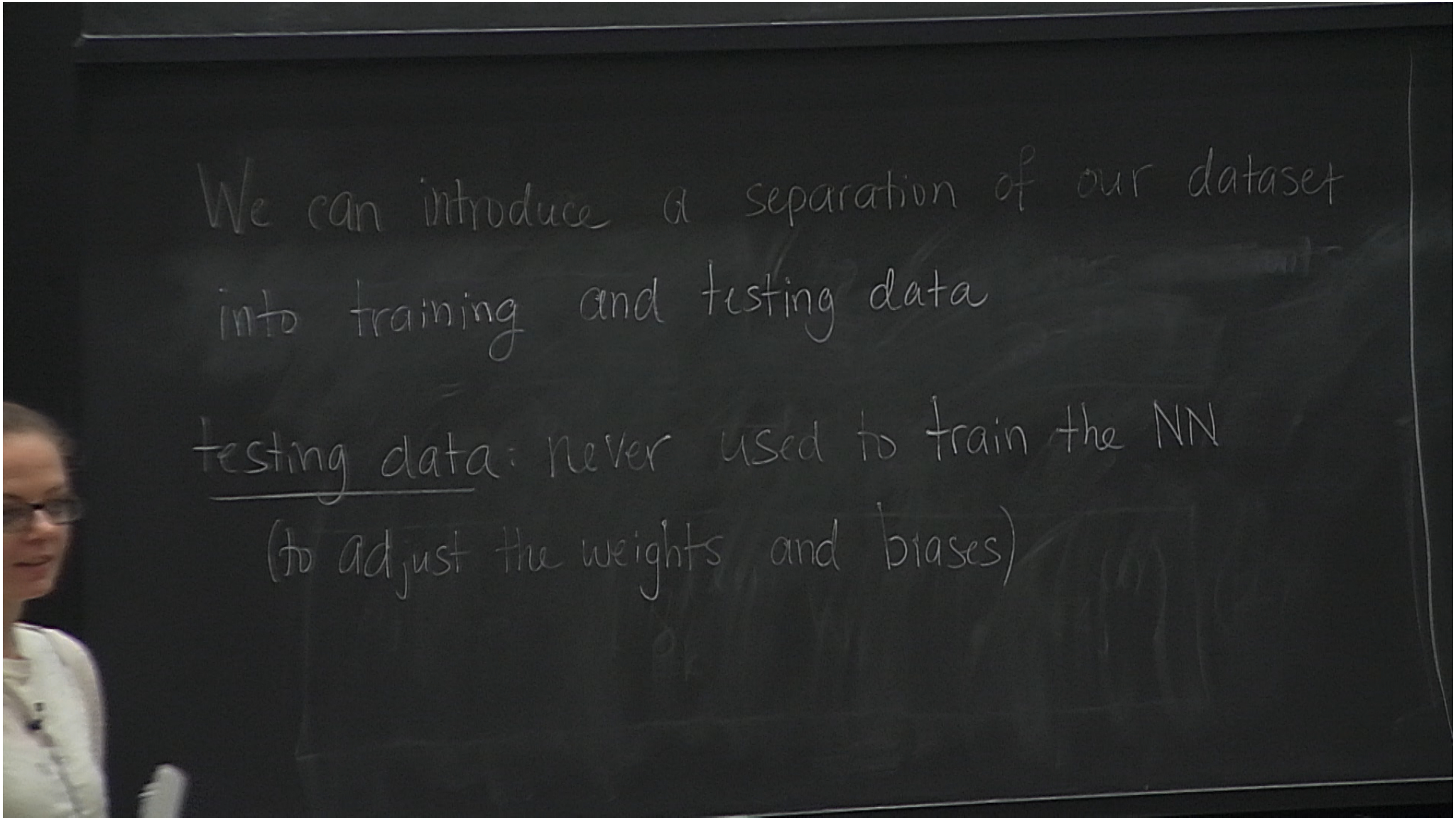
  } OR use an alternative to GD

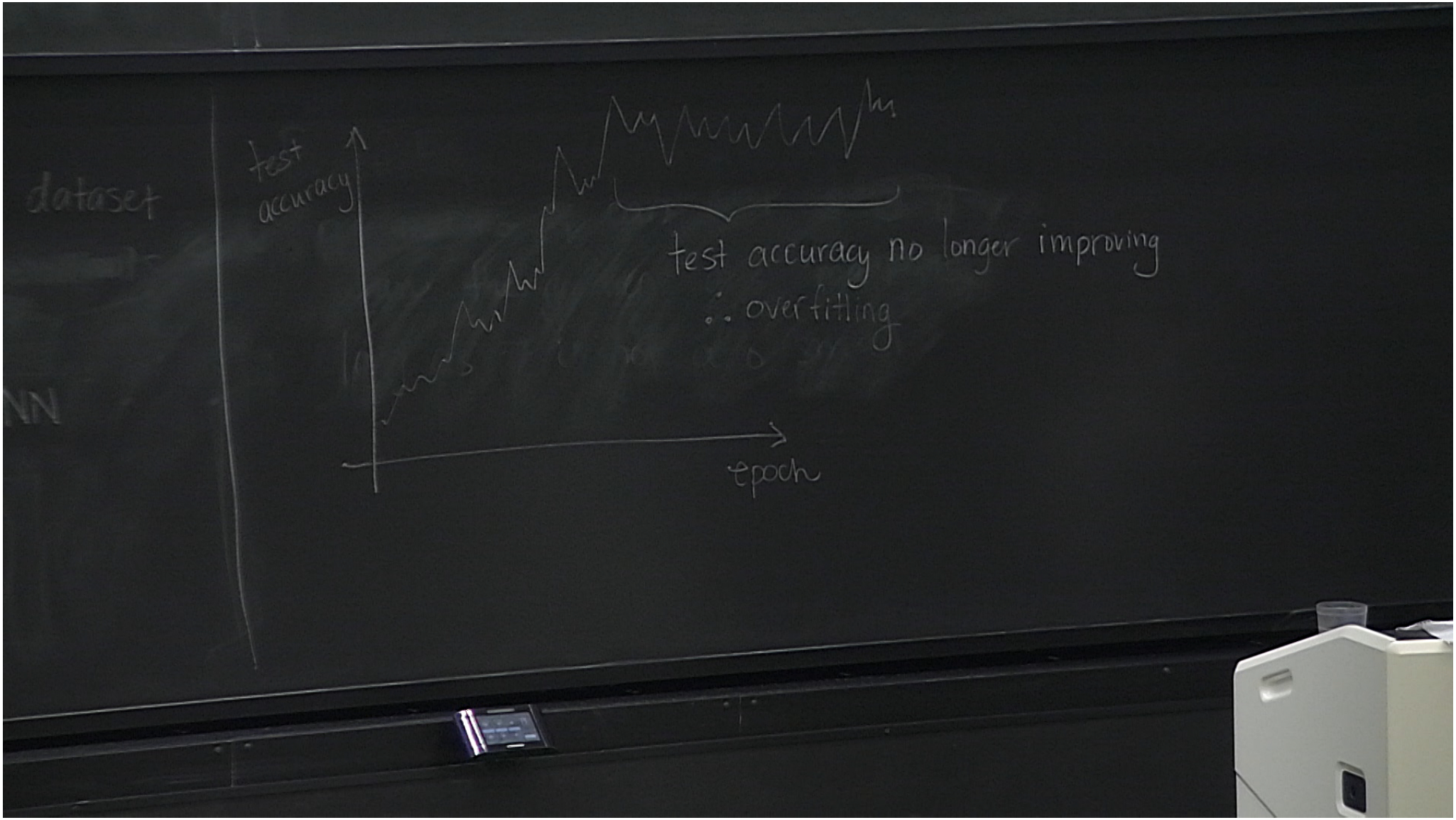
Overfitting

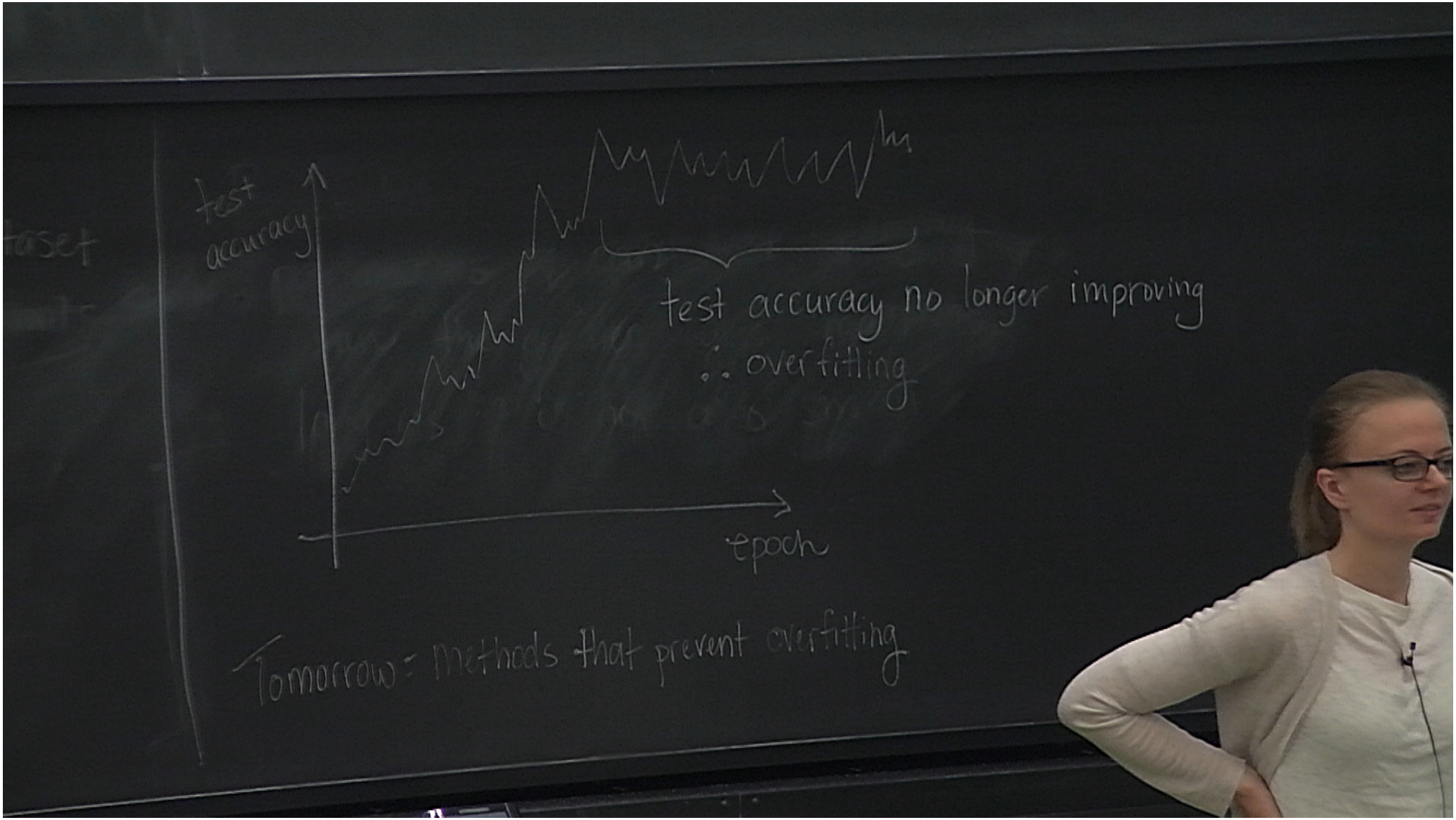
In Tutorial #1, we saw that C could be decreasing and (in some cases) the NN could get ~100% accuracy on the training data.

(% of label correctly predicted)

However, the NN was clearly overfitting!


We can introduce a separation of our dataset into training and testing data
testing data: never used to train the NN
(to adjust the weights and biases)


dataset
test accuracy
test accuracy no longer improving
∴ overfitting
epoch


test accuracy
test accuracy no longer improving
∴ overfitting
epoch
Tomorrow: methods that prevent overfitting