Title: PSI 2018/2019 - Machine Learning - Lecture 3

Speakers: Lauren Hayward Sierens

Collection: PSI 2018/2019 - Machine Learning (Hayward Sierens)

Date: March 27, 2019 - 9:00 AM

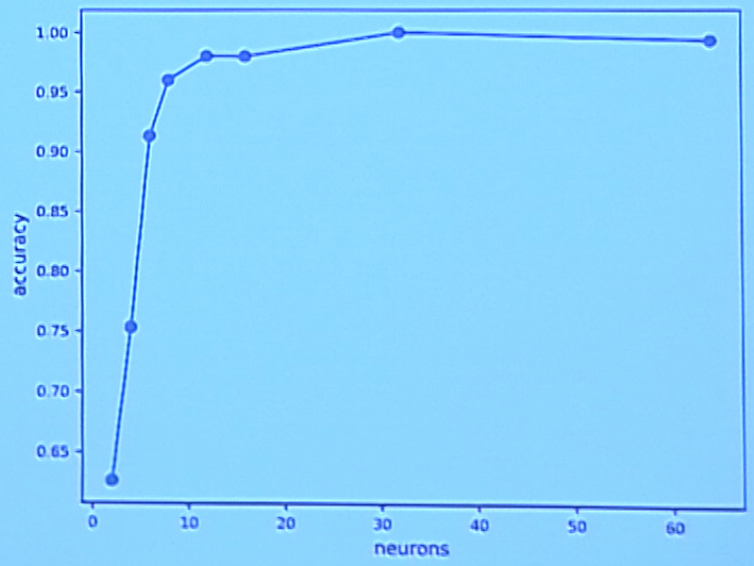
URL: http://pirsa.org/19030055

Outline for today

- more learning algorithms: alternatives to gradient descent (GD)
- backpropagation
- overfitting

Tutorial 1, Problem 2

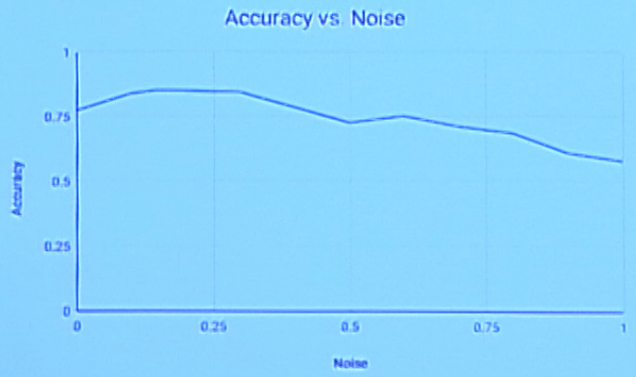
Accuracy vs. the number of neurons in the hidden layer

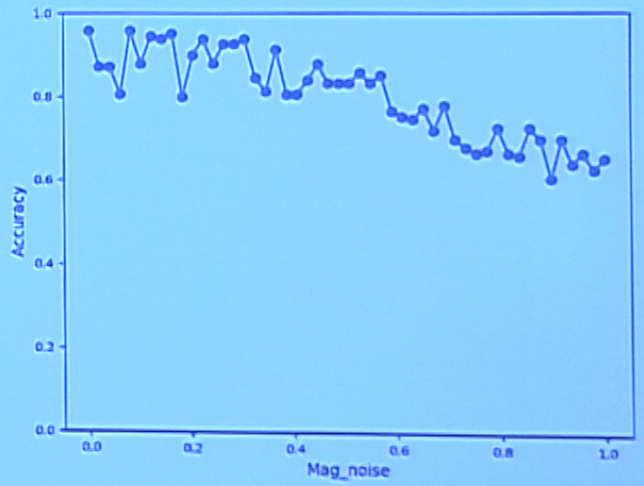


PERIMETER SCHOLARS INTERNATIONAL

Tutorial 1, Problem 2

Accuracy vs. the magnitude of noise in the data

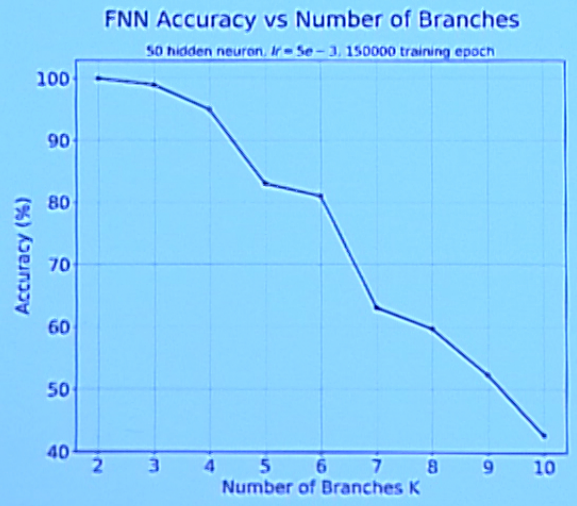


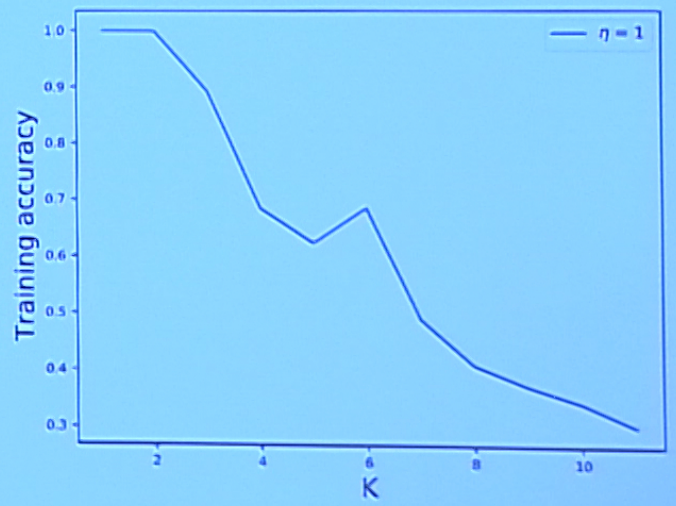


PERIMETER SCHOLARS INTERNATIONAL

Tutorial 1, Problem 2

Accuracy vs. the number of different labels (branches)

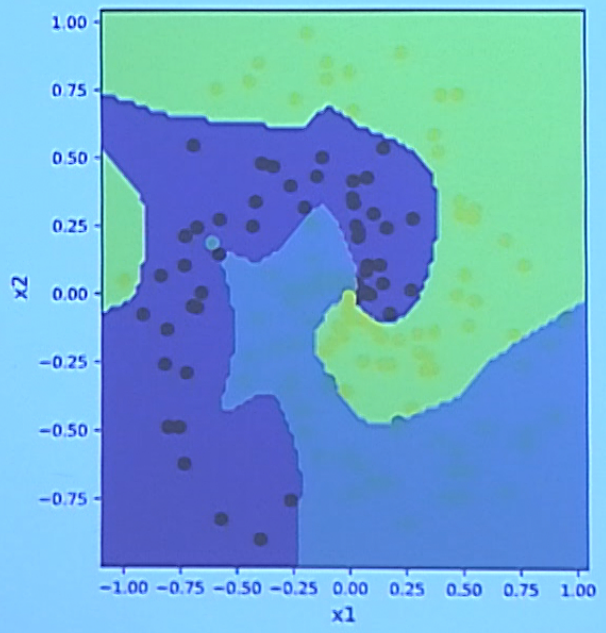


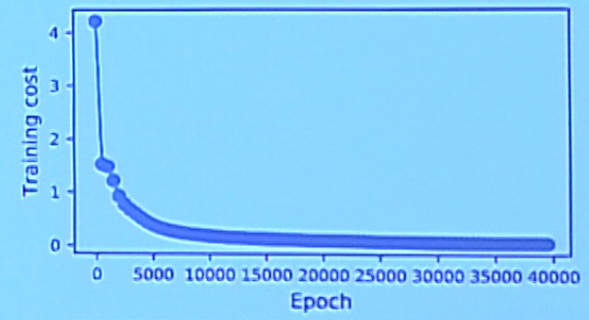


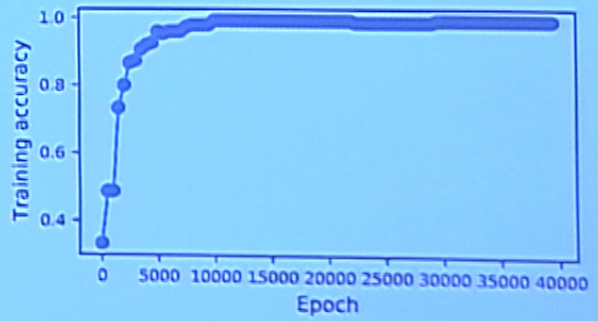
PERIMETER SCHOLARS INTERNATIONAL

Overfitting

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





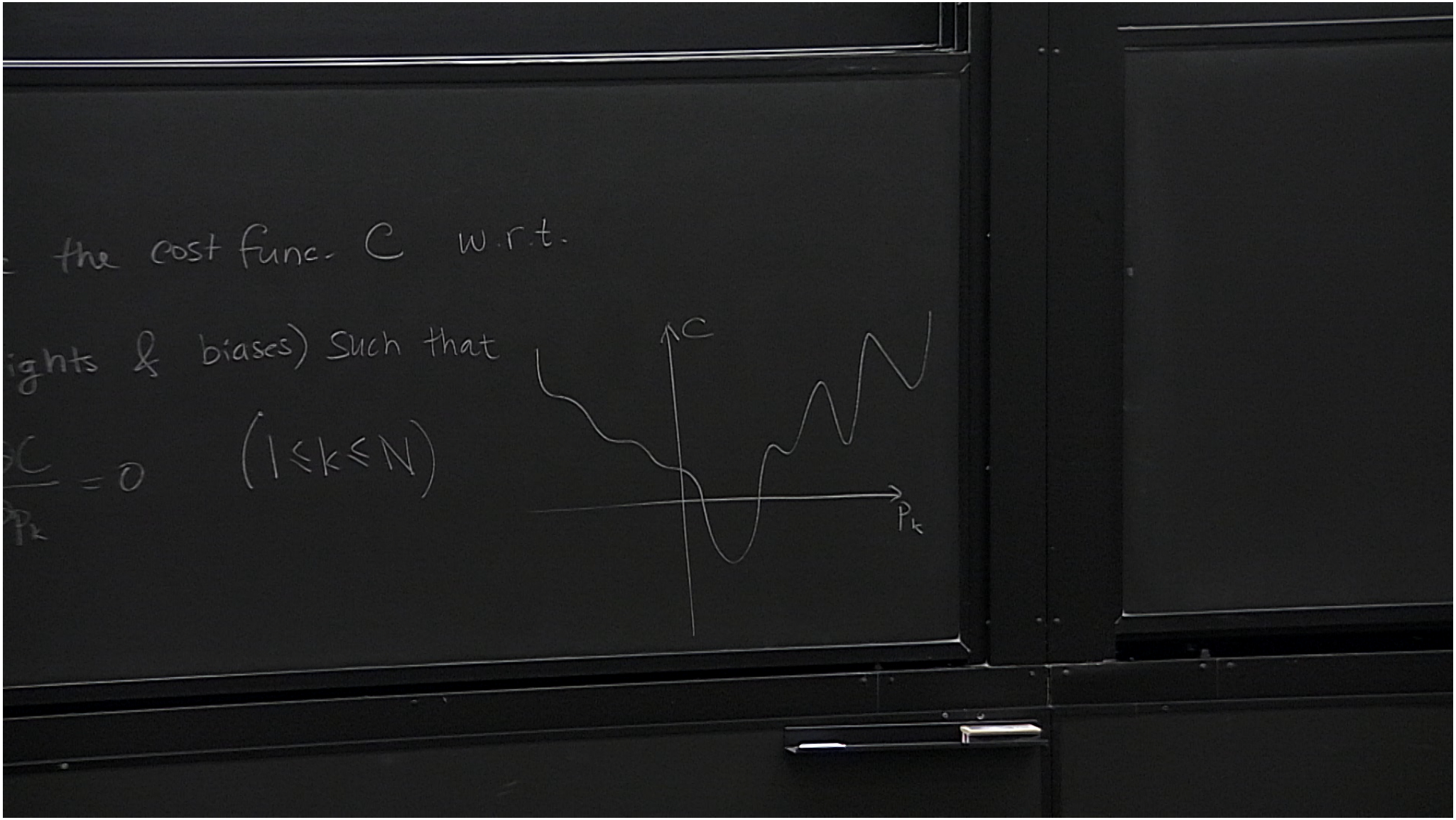


PERIMETER SCHOLARS INTERNATIONAL

Learning Algorithms

Review. we want to minimize the cost func. C w.r.t. the params $p_1, \ldots, p_N$ (weights & biases) such that

$$\frac{\partial C}{\partial p_k} = 0 \quad (1 \leq k \leq N)$$


the cost func. C w.r.t.
ights & biases) such that
$\frac{\partial C}{\partial P_k} = 0 \quad (1 \leq k \leq N)$
C
P_k

GD: update the params as:

$$\vec{p} \to \vec{p} - \underbrace{\eta}_{\text{learning rate}} \vec{\nabla}_{\vec{p}} C$$

Stochastic GD (SGD)

Assume C can be written as:

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

Idea: Approximate C and $\vec{\nabla}_{\vec{p}} C$ by summing over a randomly-chosen "mini-batch" $\mathcal{B}$ of M data points.

Benefit

as:

$\nabla_{\vec{p}} C$ by

"mini-batch" $\mathcal{B}$

Benefits:

- Speeds up calculation of gradient
- introduces a new source of randomness, which can decrease the chance of getting stuck.

Note: regula

Adding momentum:

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

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

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

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

of previous

Note: regular GD when $\gamma = 0$

Benefit: the NN params can keep changing when the gradient gets flat (as long as γ is not also small)

$\vec{v}_0 = 0 \quad \vec{p}_0 = \vec{p}_{init}$

$\vec{v}_1 = -\eta \vec{\nabla}_{\vec{p}_0} C$

$\vec{p}_1 = \vec{p}_{init} - \eta \vec{\nabla}_{\vec{p}_0} C$

$\vec{v}_2 = \gamma(-\eta \vec{\nabla}_{\vec{p}_0} C) - \eta \vec{\nabla}_{\vec{p}_1} C$

$\vec{p}_2 = \vec{p}_{init} - \eta \vec{\nabla}_{\vec{p}_0} C + \vec{v}_2$

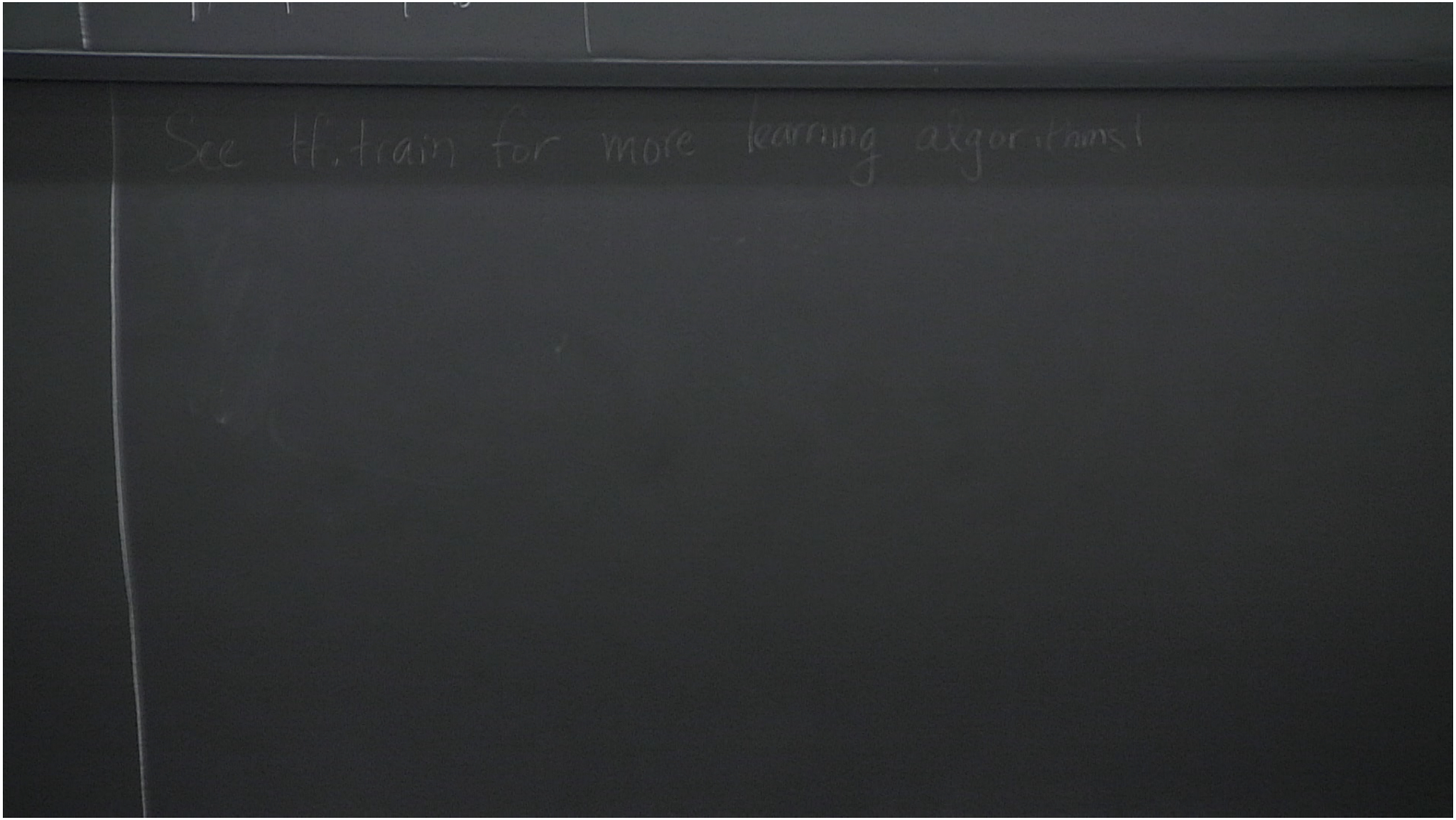
m." $(0 \leq \gamma < 1)$

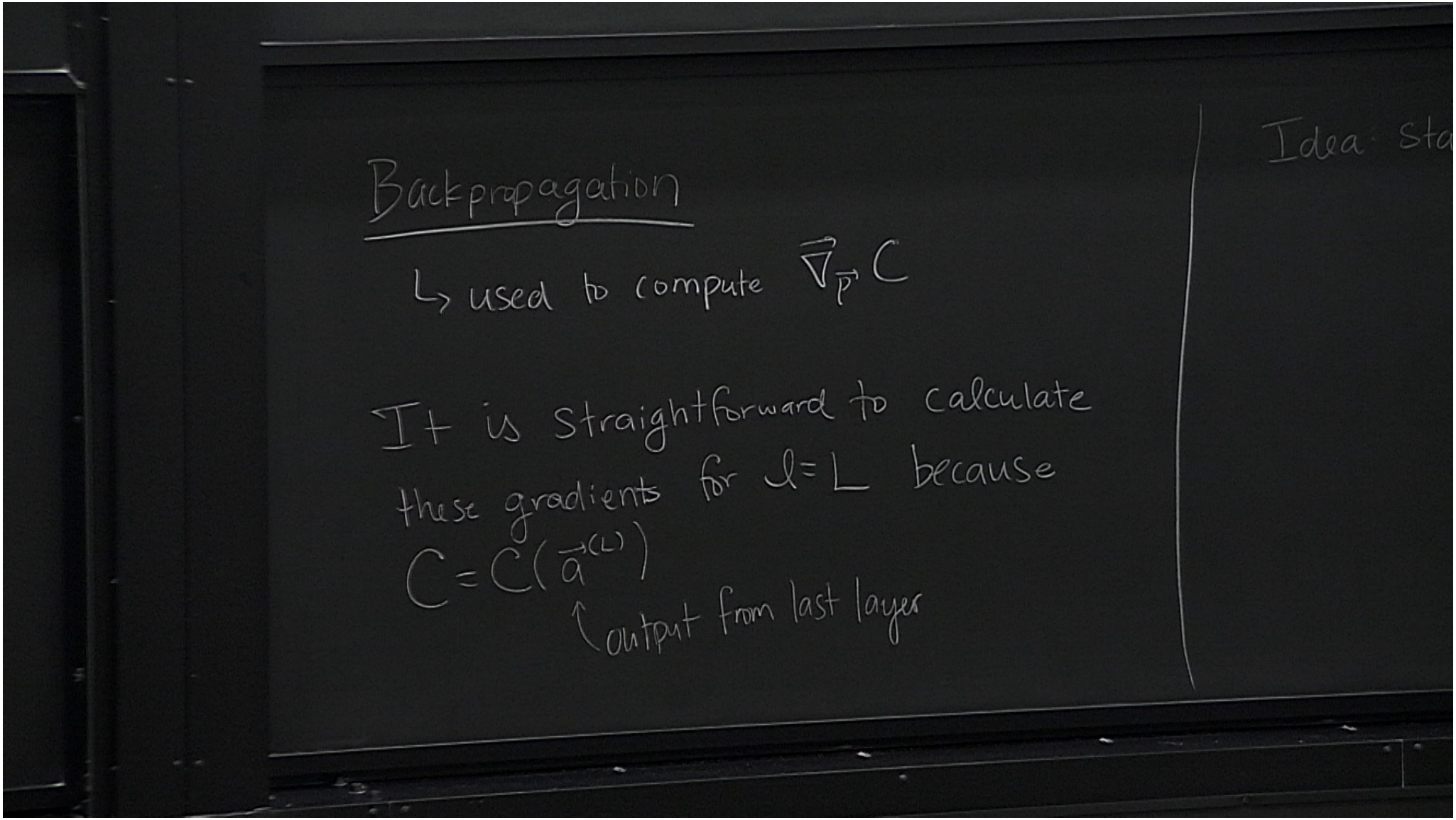
Adaptive learning rate

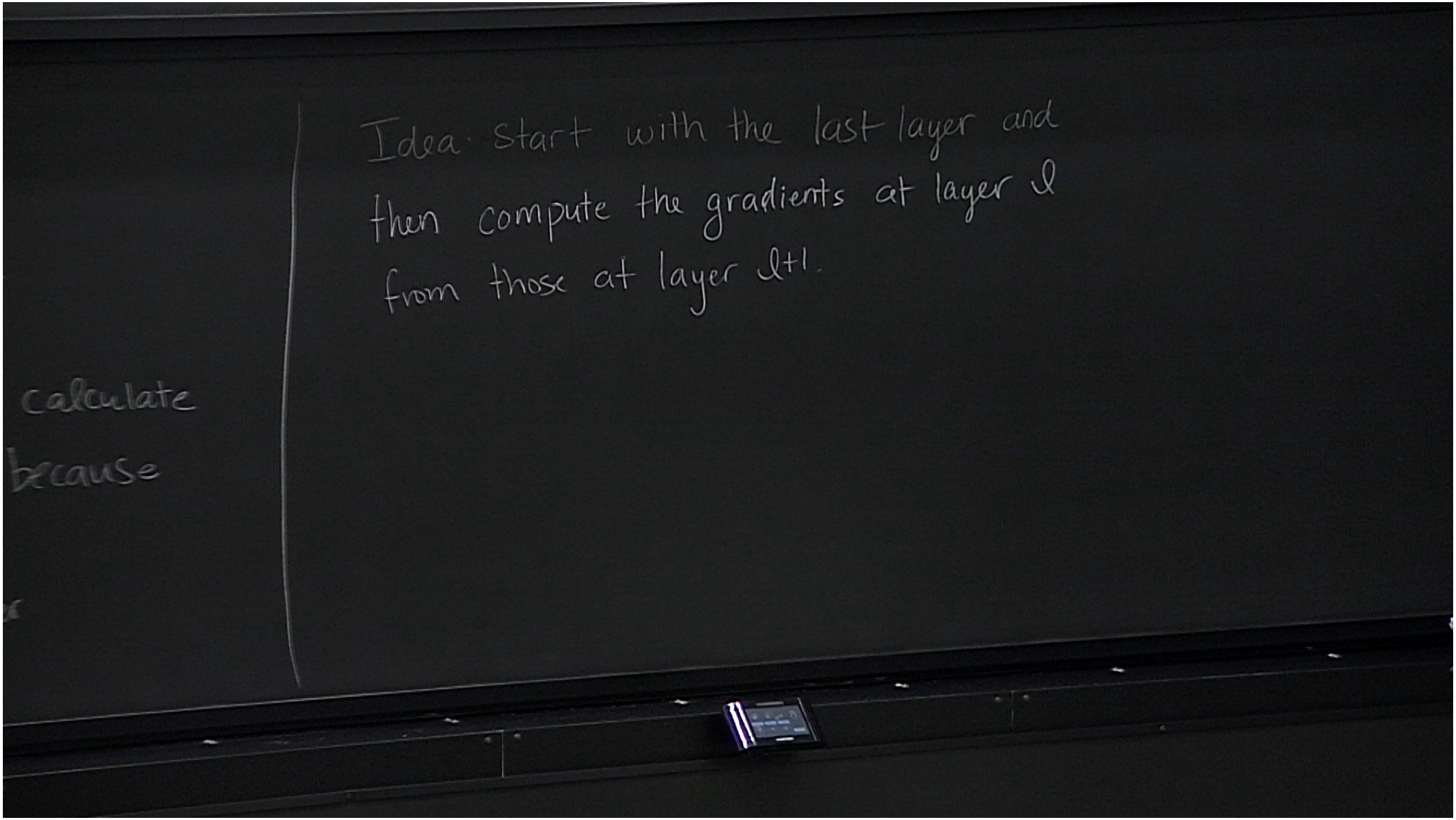
Idea: change η depending on the landscape, and allow it to be different for various weights & biases.

Examples of algorithms:

- Adam
- Root Mean Square Propagation (RMSProp)


See tf.train for more learning algorithms!


Backpropagation
↳ used to compute $\vec{\nabla}_{\vec{p}} C$
It is straightforward to calculate these gradients for $l = L$ because
$C = C(\vec{a}^{(L)})$
↑ output from last layer
Idea: sta


Idea: start with the last layer and
then compute the gradients at layer l
from those at layer l+1.
calculate
because

$\vec{p}_1 = \vec{p}_{init} - \eta \vec{\nabla}_{\vec{p}_0} C$ | $\vec{p}_2 = \vec{p}_{init} - \eta \vec{\nabla}_{\vec{p}_0} C + \vec{v}_2$

Idea: start with the last layer and then compute the gradients at layer ℓ from those at layer $\ell+1$.

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

Let's study the gradients $\frac{\partial c(\vec{x})}{\partial W_{ij}^{(\ell)}}$ and $\frac{\partial c(\vec{x})}{\partial b_j^{(\ell)}}$ corresponding to <u>one</u> data point

Recall:

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

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

Now define: $\delta_j^{(\ell)} \equiv \frac{\partial C}{\partial z_j^{(\ell)}}$

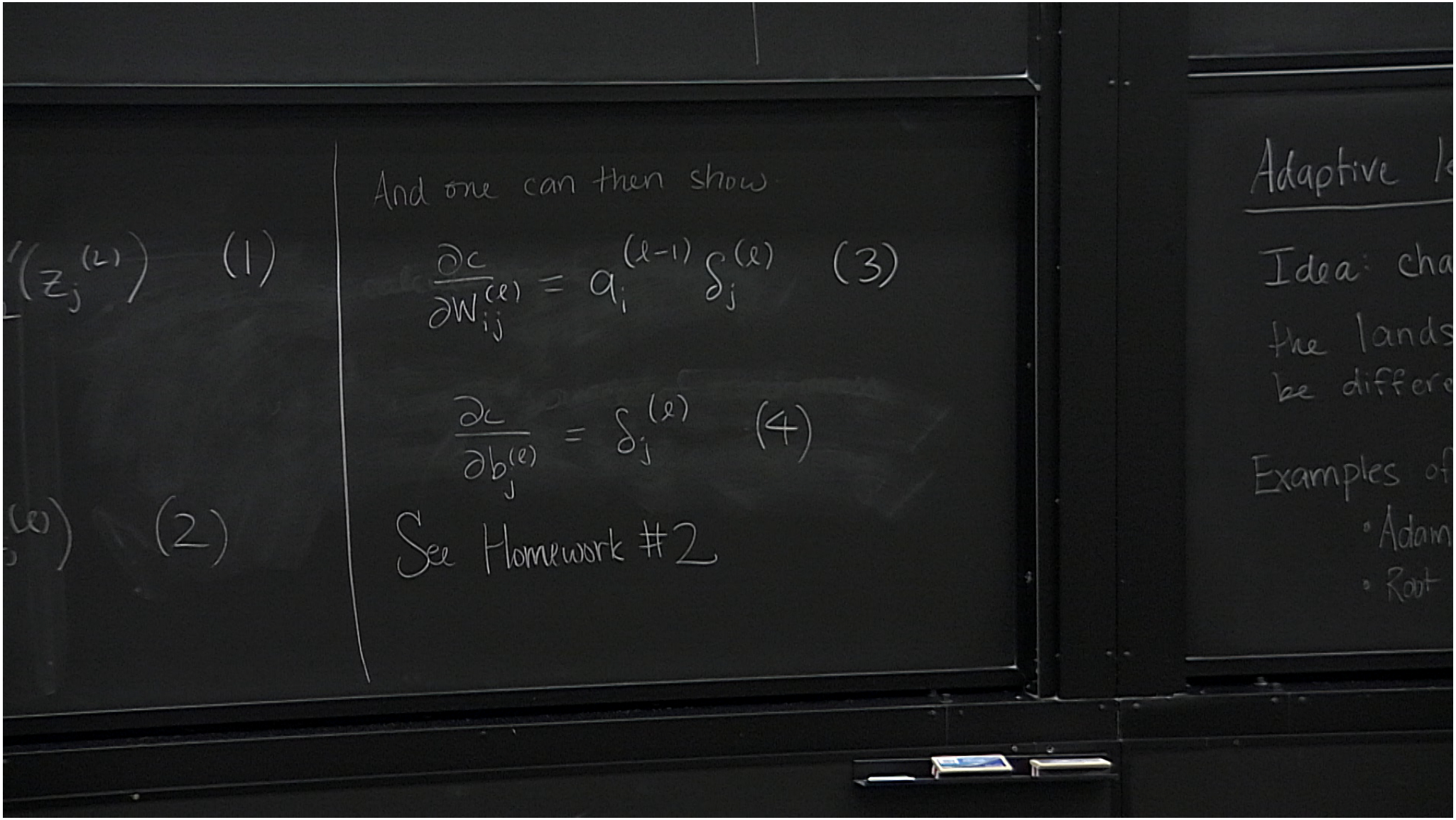
$+ b_j^{(\ell)}$

First get $\delta_j^{(L)}$:

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

Then for $\ell < L$, one can show:

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


$\sigma'(z_j^{(L)})$ (1)
$\delta^{(L)}$) (2)
And one can then show
$\frac{\partial C}{\partial W_{ij}^{(\ell)}} = a_i^{(\ell-1)} \delta_j^{(\ell)}$ (3)
$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}$ (4)
See Homework #2
Adaptive
Idea: cha
the lands
be differe
Examples of
• Adam
• Root

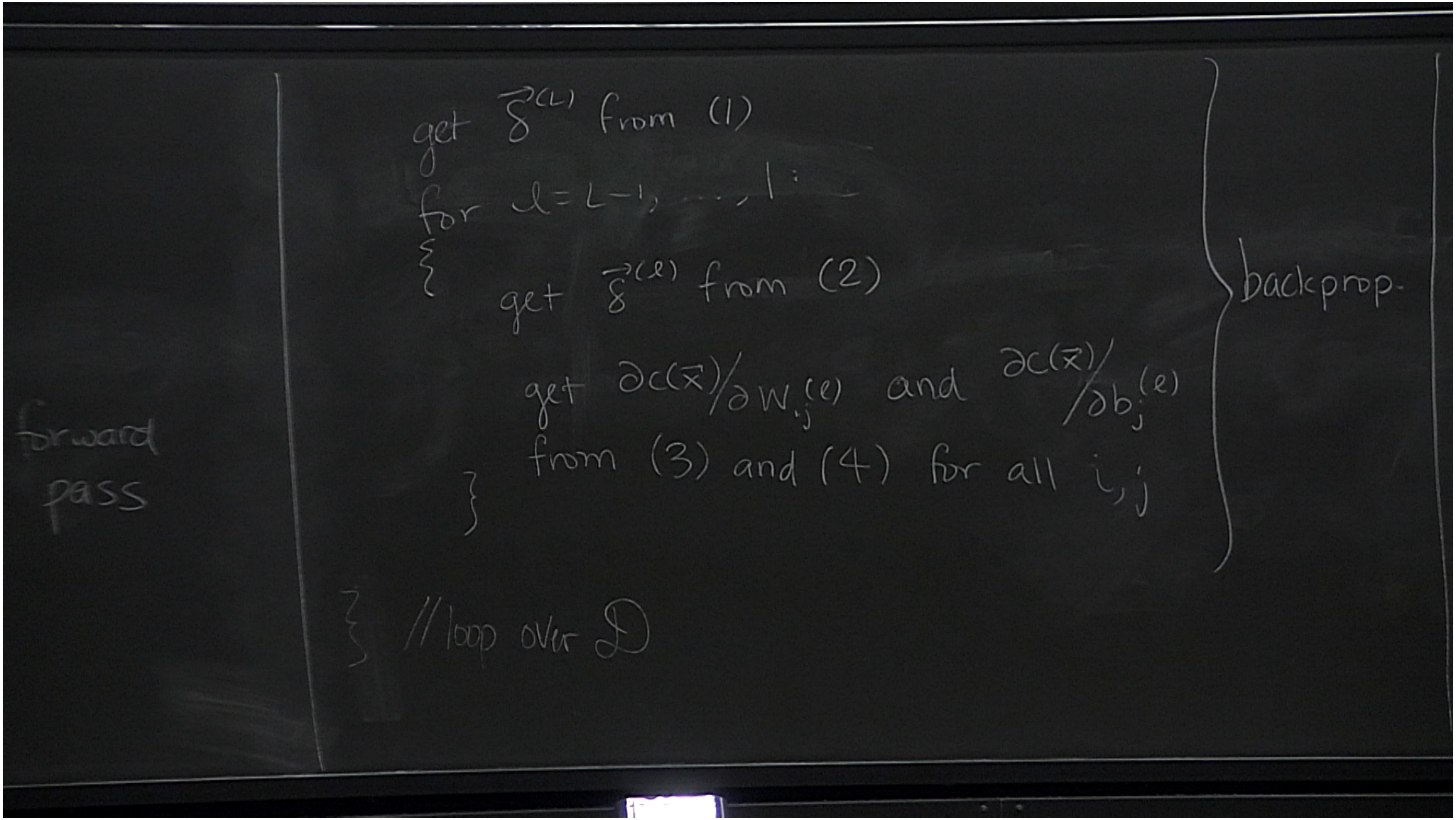
Pseudocode for one "epoch" of training

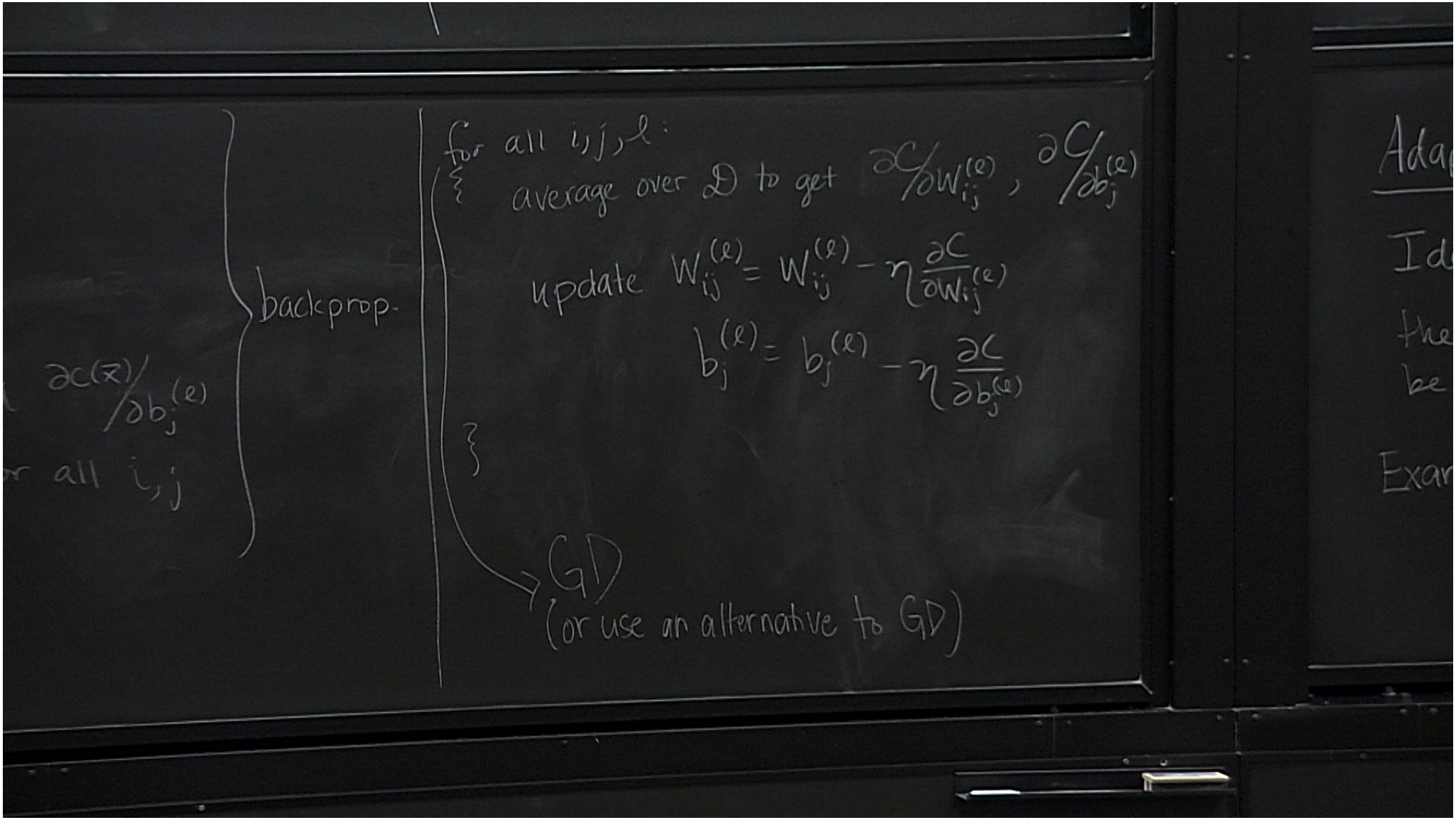
for $\vec{x}$ in $\mathcal{D}$ {

$\vec{a}^{(0)} = \vec{x}$

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

$\vec{a}^{(l)} = g_l(\vec{a}^{(l-1)} W^{(l)} + \vec{b}^{(l)})$ } forward pass


get $\vec{\delta}^{(L)}$ from (1)
for $l = L-1, \ldots, 1$:
{
get $\vec{\delta}^{(l)}$ from (2)
get $\partial C(\vec{x})/\partial w_{ij}^{(l)}$ and $\partial C(\vec{x})/\partial b_j^{(l)}$ from (3) and (4) for all i, j
}
backprop.
forward pass
} // loop over $\mathcal{D}$


$\partial C(x)/\partial b_j^{(\ell)}$
or all i,j
backprop.
for all i,j,ℓ:
{
average over $\mathcal{D}$ to get $\partial C/\partial W_{ij}^{(\ell)}$, $\partial C/\partial b_j^{(\ell)}$
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 to GD)
Ada
Id
the
be
Exar

Overfitting

In SL, "accuracy" refers to the percentage of labels the NN predicts correctly

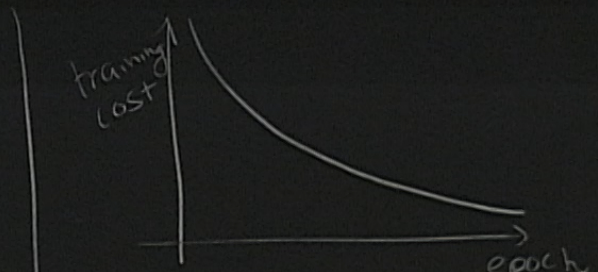
In Tut. 1, we saw cases where C was decreasing and the NN achieved 100% accuracy but was overfitting

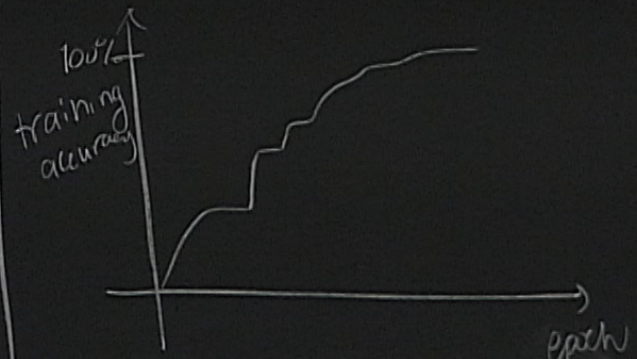
to the percentage

correctly

where C was decreasing

uracy but was overfitting





epoch

epoch

We can segment the dataset $\mathcal{D}$ into "training" and "testing" data

Training data: uses to adjust the weights & biases

Testing data: never used in training

