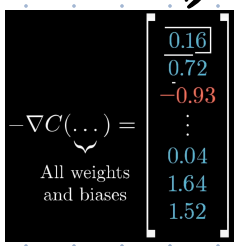
back propagation
↳ algorithm to compute Gradient
↳ propergate backwards = spreading backwards

Steps of back-prop:

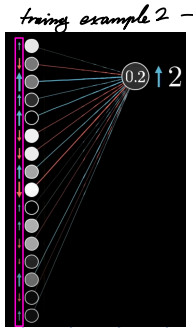
1. forward propagate a training example

2. evaluate it's quality / accuracy:
   ↳ compute Cost (Sum of Squared difference from output to correct)
   The Cost Vector encodes:
   ↳ if weight / biases should increase / decrease
   ↳ the importance of each weight / bias

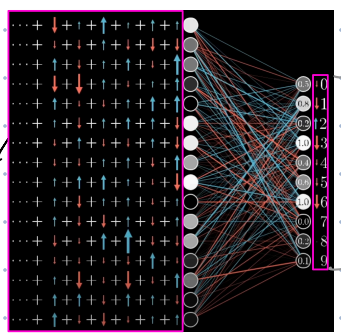


if weight has high number the network is more sensitive for it
↓
where is biggest bang for buck

3. Go backwards through the layers & shift the weights



→ add all shifts for total Gradient



→ indicates how to shift weight

Sum of Shifts
↳ repeat this for all layers

4. repeat the process of discovering the shifts for other training examples

5. take the average shift over all training data for weights of new Gradient
   ↳ if it would just have the shifts for "training example 2" it just could predict handwritten 2's → model would be overfitted

	2	5	0	4	1	9	⋯	Average over all training data
w_0	−0.08	+0.02	−0.02	+0.11	−0.05	−0.14	⋯	→ −0.08
w_1	−0.11	+0.11	+0.07	+0.02	+0.09	+0.05	⋯	→ +0.12
w_2	−0.07	−0.04	−0.01	+0.02	+0.13	−0.15	⋯	→ −0.06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
$w_{13,001}$	+0.13	+0.08	−0.06	−0.09	−0.02	+0.04	⋯	→ +0.04

$$\Longrightarrow -\nabla C(\vec{w}) = \begin{bmatrix} -0{,}08 \\ +0{,}12 \\ -0{,}06 \\ \vdots \\ +0{,}04 \end{bmatrix} = \text{new / trained Gradient}$$

6. evaluate the accuracy of new Gradient