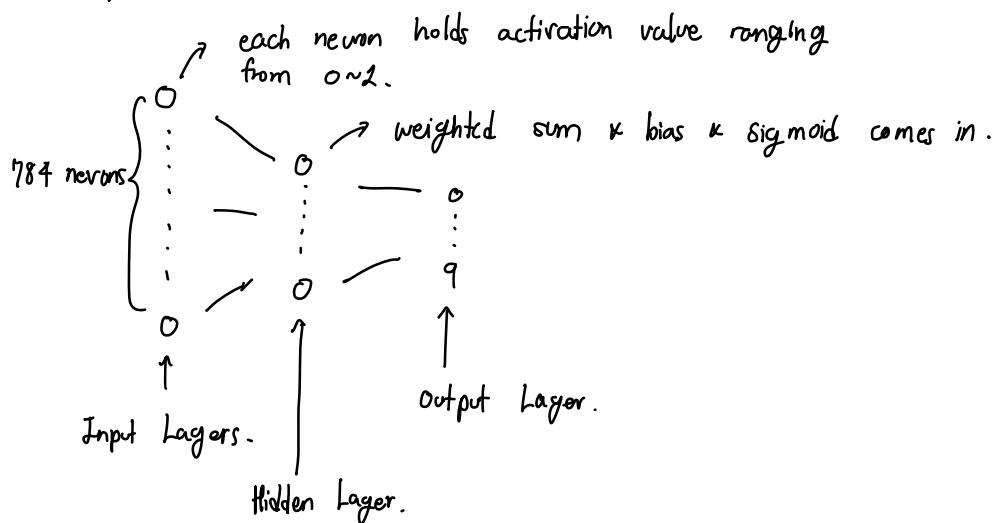


Recap: Chapter 2.



If 784 neurons w/ 2 arbitrary Hidden Layers w/ 16 neurons each,

Then network has about $13,002 = 784 \times 16 + 16 \times 16 + 16 \times 10$ weights

$16 + 16 + 10$
biases

Controls what network does.

Chapter 2: Gradient Descent.

Considering $\sigma(w_1a_1 + w_2a_2 + \dots + w_na_n + b)$,

0
1
2
3
4
5
6
7
8
9

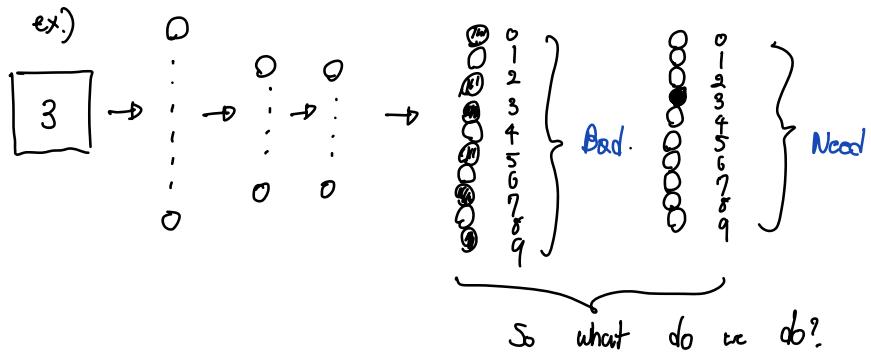
- this equation shows that conceptually, each neuron is connected to all of the neurons in the previous layer.
- weights in the weighted sum defining each activation are like the strengths of each neuron connection.
- bias is the indication whether those neurons tend to be active or inactive.

Now, How do we find these weights & biases?

1. Initialize randomly

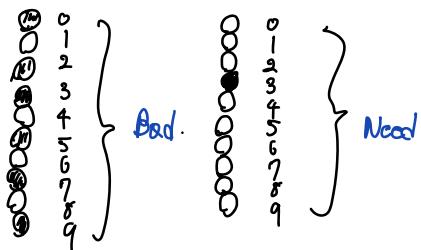
↳ will perform terribly because it is all random.

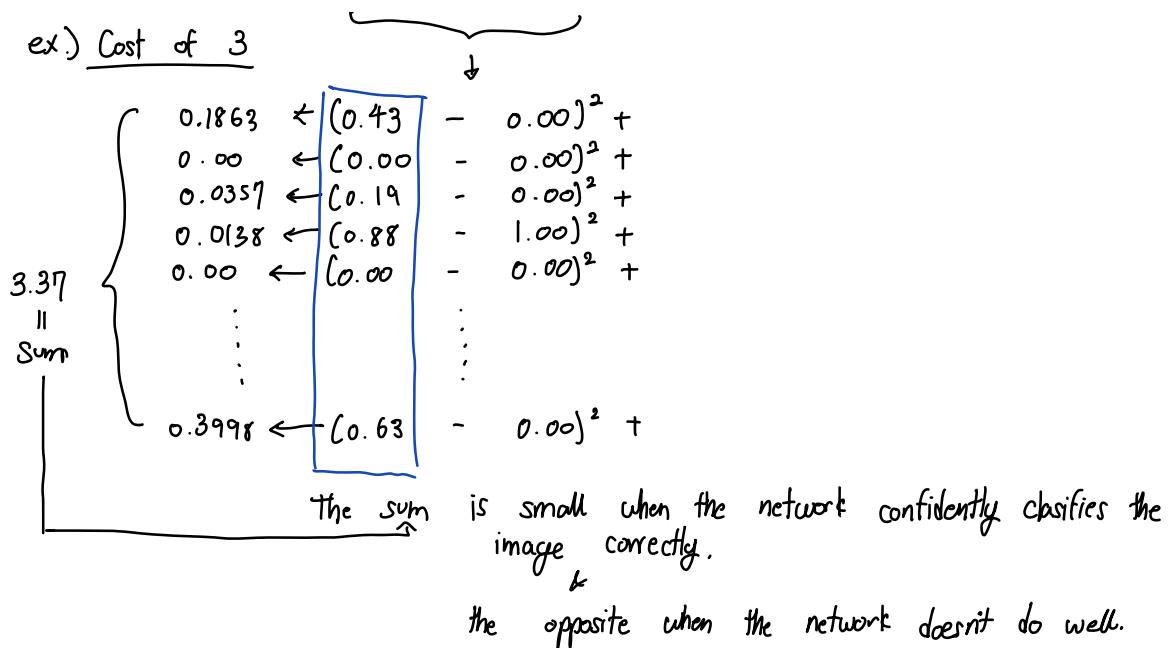
↳ we need to define "cost" function



2. Find "cost"

1. add up the squares of differences of both Bad and Need.





3. Average the cost of all training data.

- from above, using Bad & Need and its sum, we find the average cost of many data (image in this case) given.

Complicated!

in Neural Network function. from our example.

Input : 784 numbers (pixels)

Output : 10 numbers

Parameters : 13,002 weights / biases

in Cost Function

Input : 13,002 weights / biases

Output : 1 number (The Cost)

Parameters : Many, many, many training examples.

* So How do we change the weights & biases since it gives poor result?

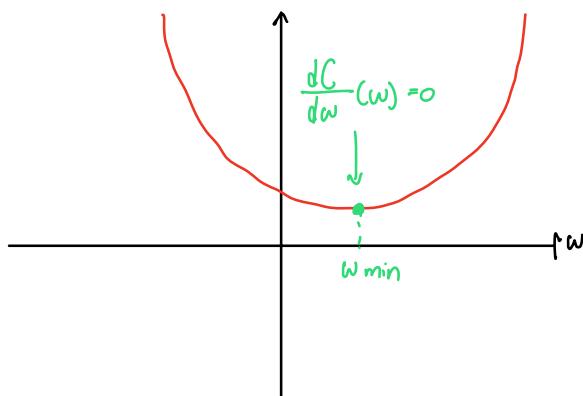
* 4. Gradient Descent *

ex.) Cost Function

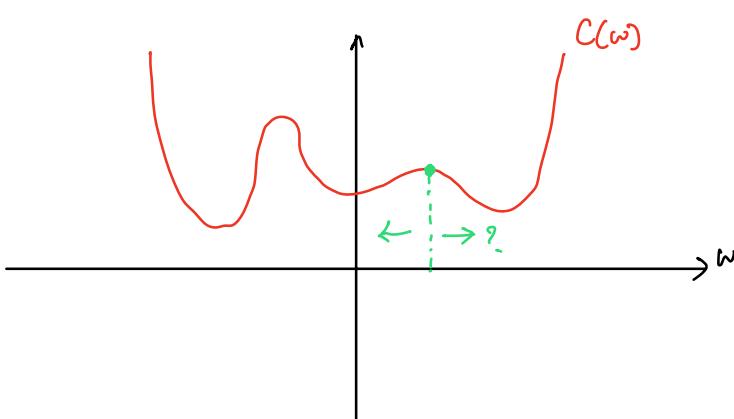
$$C(\underbrace{w_1, w_2, \dots, w_{13,02}}_{\text{Weights and Biases}})$$

imagine a simple function $C(w)$.

single input \Rightarrow single output.



This is not always feasible (possible)



So! start at any input,
and see which "Direction"

\downarrow
Slope of a function

- if slope > 0 , shift left.
- if slope < 0 , shift right.

continue doing this repeatedly, you can find the local minimum
of the function.

imagine a ball rolling on top of the curve.

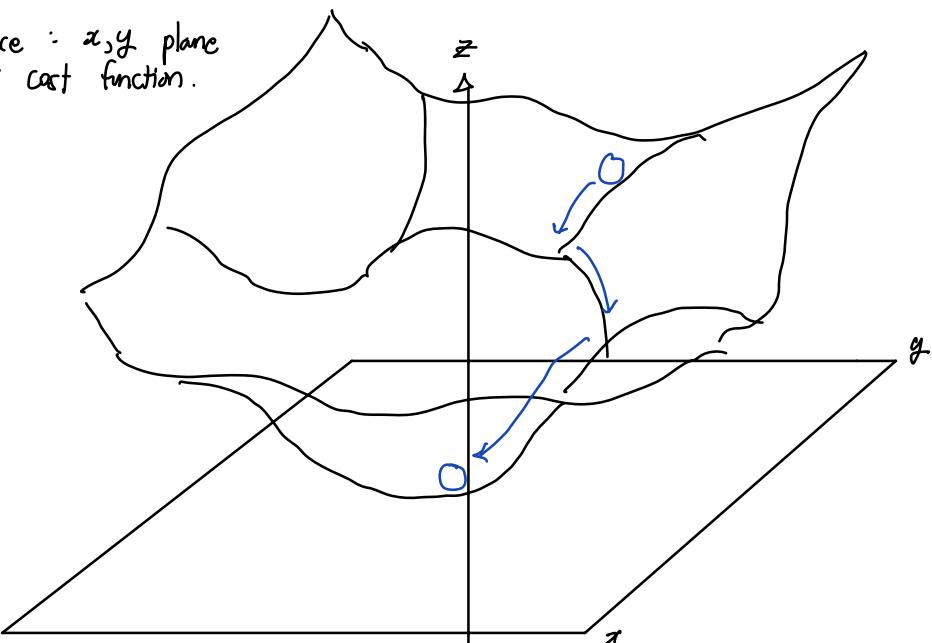
- but there are many valleys that you may

land on, depending on where you roll the ball down hill.

↳ no guarantee that the ball rolled on the smallest value of the cost function.

But using this idea and finding a slope closer to 0, we can find the closest minimum.

Input space : x, y plane
graph : cost function.



Q: Which direction decreases $c(x, y)$ most quickly?

- what's the downhill direction?

A: "Gradient" gives the direction of steepest increase.

$$\nabla C(x, y)$$

and take the negative of this to find the down hill.

and the length shows how steep the steep is.

} Multivariable calculus.

1. Compute ∇C

2. Small step in $-\nabla C$ direction

3. Repeat

13,002 weights & biases.

How to 'nudge' all weights & biases.

$$\vec{w} = \begin{bmatrix} 2.25 \\ -1.57 \\ \vdots \\ -1.16 \\ 3.82 \end{bmatrix} \rightarrow -\nabla C(\vec{w}) = \begin{bmatrix} 0.18 \\ 0.45 \\ -0.57 \\ \vdots \\ 0.40 \\ -0.32 \end{bmatrix}$$

\uparrow
 + add $-\nabla C(\vec{w})$ to \vec{w} to adjust

Then! the in our cost function

$$\left\{ \begin{array}{l} 0.1863 \leftarrow (0.43) - 0.00)^2 + \\ 0.00 \leftarrow (0.00) - 0.00)^2 + \\ 0.0357 \leftarrow (0.19) - 0.00)^2 + \\ 0.0138 \leftarrow (0.88) - 1.00)^2 + \\ 0.00 \leftarrow (0.00) - 0.00)^2 + \\ \vdots \\ 0.3998 \leftarrow (0.63) - 0.00)^2 + \end{array} \right.$$

\downarrow

$$\left. \begin{array}{l} \leftarrow (0.02) - 0.00)^2 + \\ \text{close to } 0 \leftarrow (0.04) - 0.00)^2 + \\ \leftarrow (0.09) - 0.00)^2 + \\ \leftarrow (0.98) - 1.00)^2 + \\ \leftarrow (0.06) - 0.00)^2 + \\ \vdots \\ \leftarrow (0.02) - 0.00)^2 + \end{array} \right\}$$

Comparing this effectively
 is known as backpropagation.

what is Gradient descent?

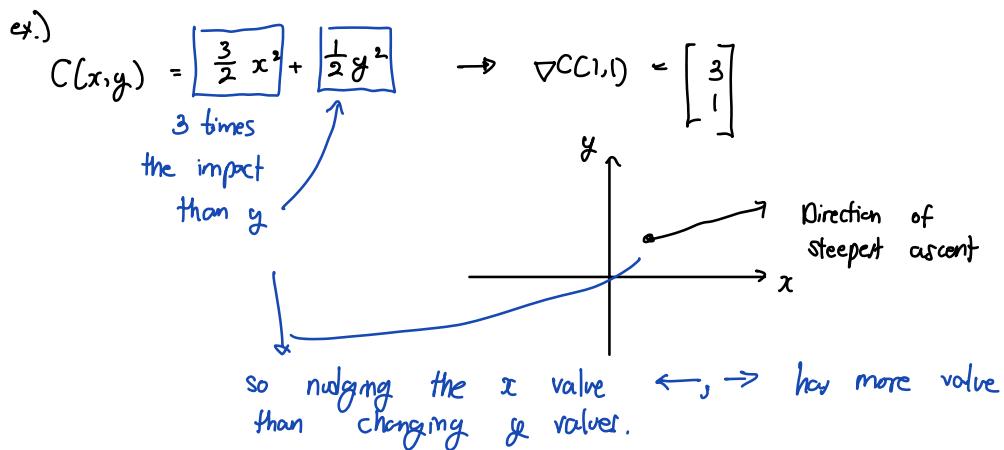
- Way to converge toward some local minimum of a cost function.

$$\vec{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$$

$$-\nabla C(\vec{W}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \\ 0.16 \end{bmatrix}$$

w_0 should increase somewhat
 w_1 should increase a little
 w_2 should decrease a lot

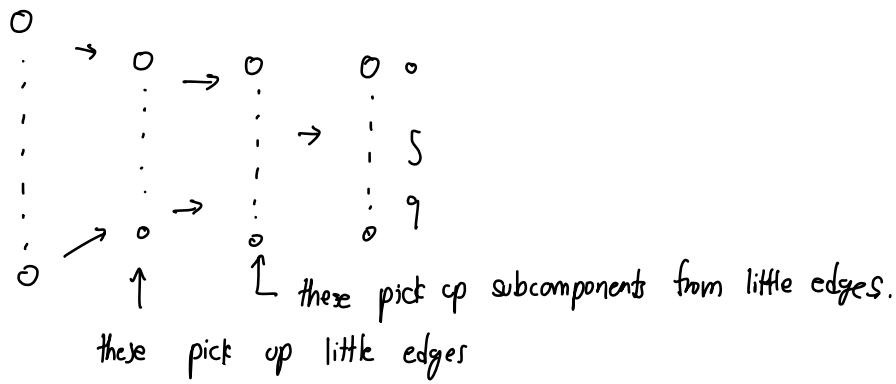
$w_{13,000}$ should increase a lot
 $w_{13,001}$ should decrease somewhat
 $w_{13,002}$ should increase a little



5. How well does this work?

$$\frac{\text{number correct}}{\text{total}} = \text{value.}$$

Before we said that in a perfect world,



* reality is, that it doesn't work like that at all.

↳ it almost look random! (loose patterns)

because it works in the 13,002 dimensional space
of possible weights and biases, it just finds a little
local minimum

↳ if you input a random image, (like a blurry looking
black and white picture), system actually gives a really
confident nonsense answer.

↳ because it has no idea how to draw a digit it can
only classify them. due to constrained training setup.

This is "old technology" aka "Multilayer perceptron"