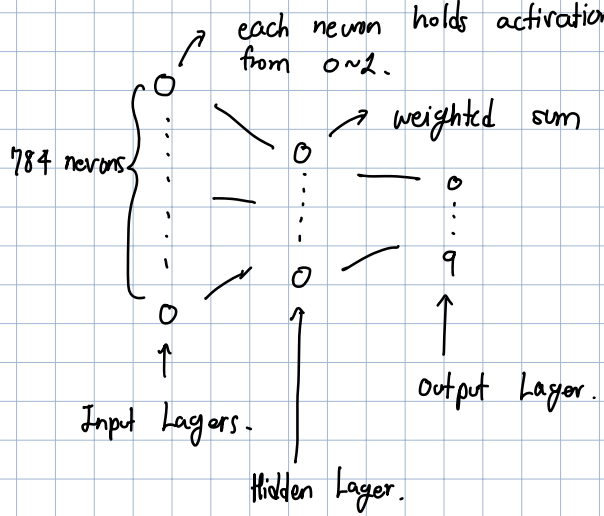
Recap: Chapter 1.



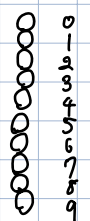
If 784 neurons w/ 2 arbitrary hidden layers w/ 16 neurons each,

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 $\sigma(w_1 a_1 + w_2 a_2 + \dots + w_n a_n + b)$,



- 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 neuron tend to be active or inactive.

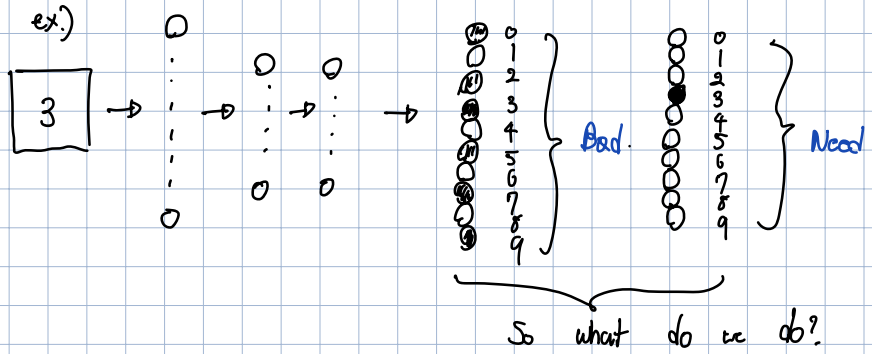
Now, How do we find these weights & biases?

1. Initialize randomly

↳ will perform terribly because it is all random.

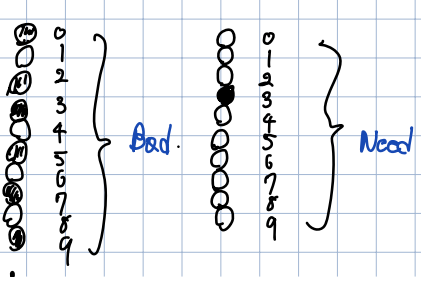
↳ we need to define "cost" function

ex:)

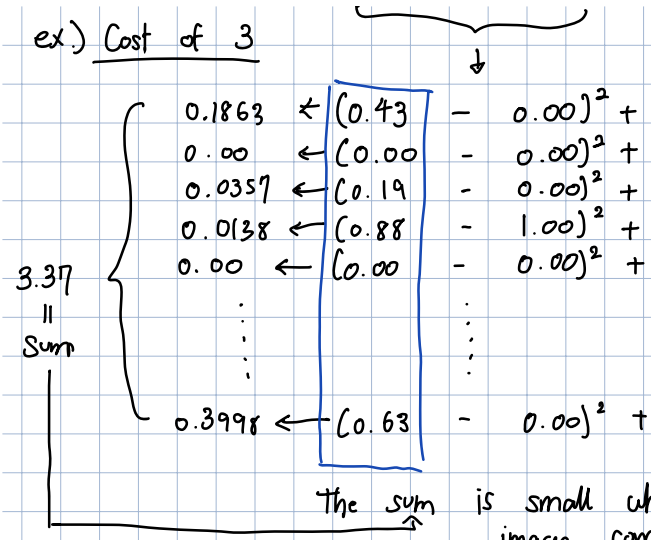


2. Find "cost"

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



ex.) Cost of 3



The sum is small when the network confidently clasifies the image correctly.

&

the opposite when the network doesn't do well.

3. Average the cost of all training data.

- from above, using Bad & Need and it's 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.

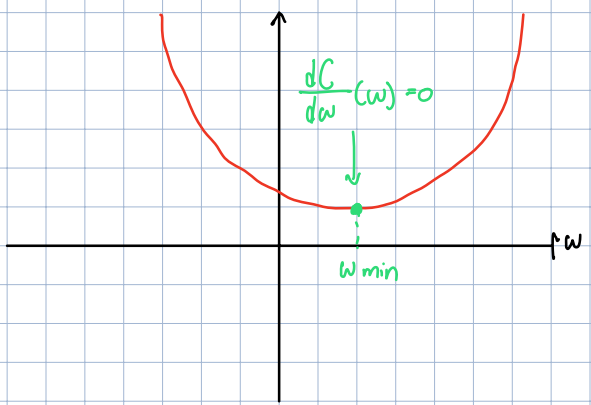
ex.) Cost Function

$C(w_1, w_2, \ldots, w_{13,002})$

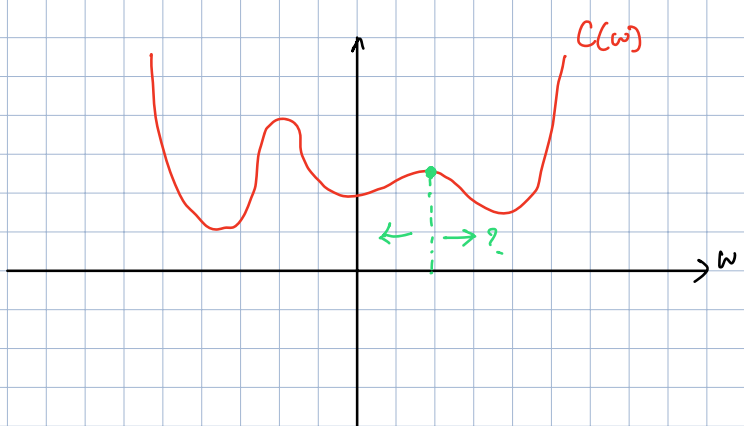
Weights and Biases.

imagine a simple function $C(w)$.

single input ⇒ single output.



This is not always feasible (possible)



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

↓

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.

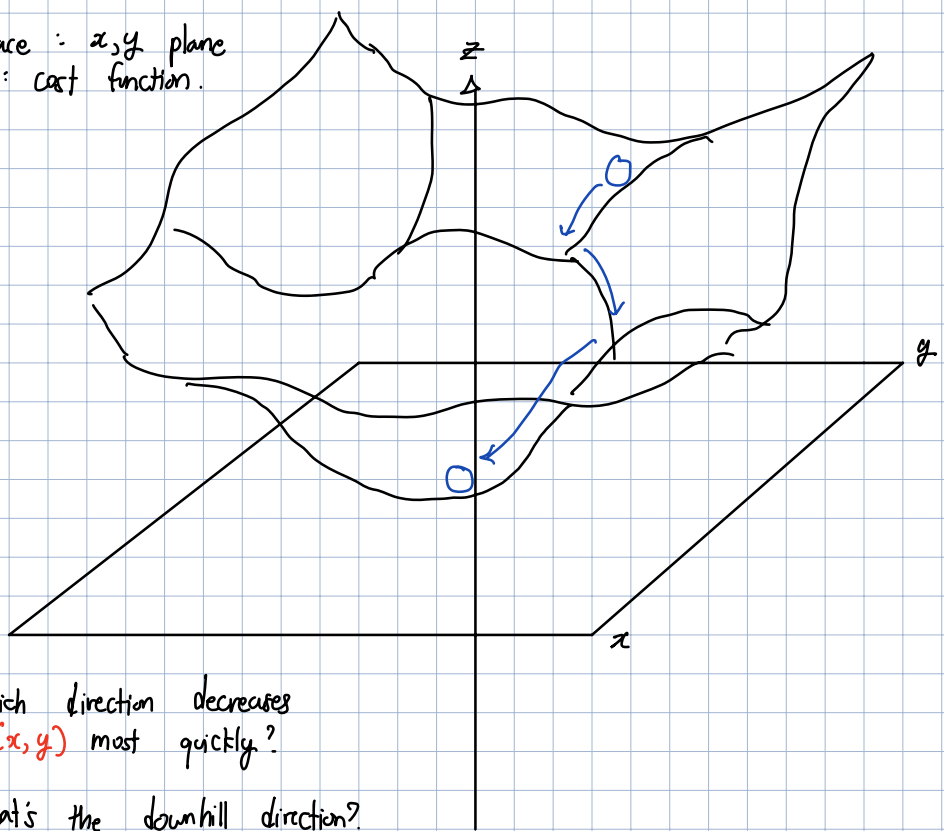
- but there are many valleys that you may

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.

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

④ add $-\nabla C(\vec{w})$ to $\vec{w}$ to adjust

Then! the in our cost function

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

$$\downarrow$$

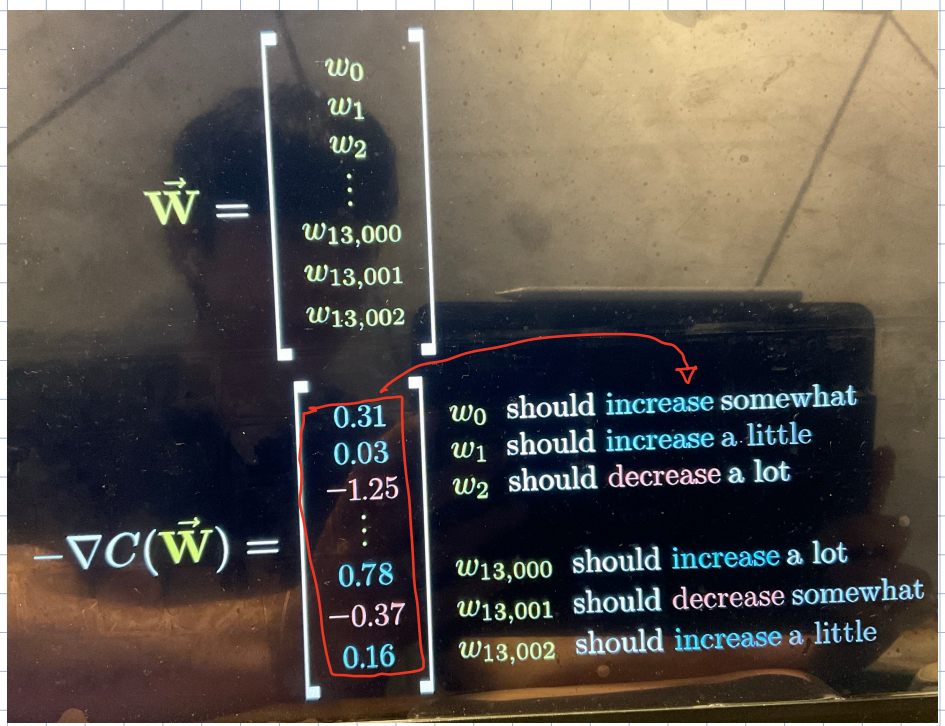
$$\left\{\begin{array}{rl} \leftarrow & (0.02 - 0.00)^2 + \\ \leftarrow & (0.04 - 0.00)^2 + \\ \leftarrow & (0.09 - 0.00)^2 + \\ \leftarrow & (0.98 - 1.00)^2 + \\ \leftarrow & (0.06 - 0.00)^2 + \\ \vdots & \\ \leftarrow & (0.02 - 0.00)^2 + \end{array}\right.$$

Close to 0

Computing this effectively is known as <u>backpropagation</u>.

What is Gradient descent?

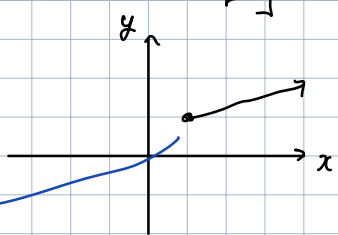
- Way to converge toward some local minimum of a cost function.



ex.)

$C(x,y) = \frac{3}{2}x^2 + \frac{1}{2}y^2 \rightarrow \nabla C(1,1) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$

3 times the impact than y

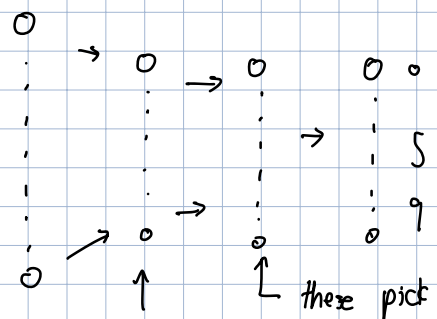


so nudging the x value ←, → has more value than changing y values.

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 nonsence 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"