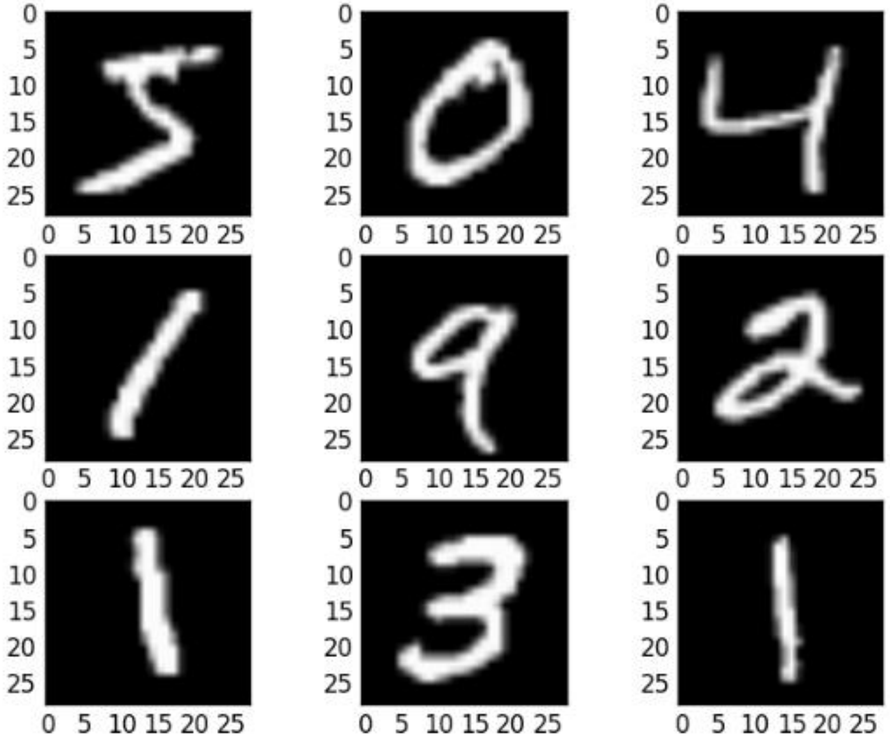
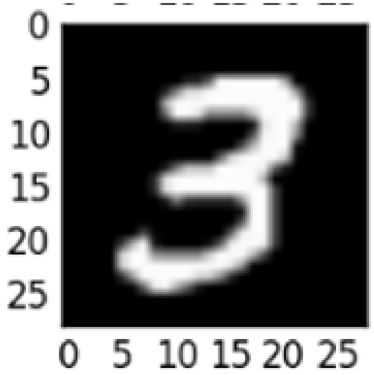
Neural Network Foundation

Tianchu.Zhao@uts.edu.au

MNIST

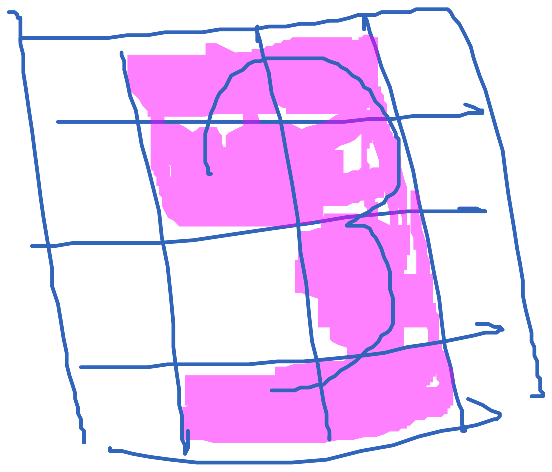
- MNIST (Modified National Institute of Standards and Technology database)
- is a large database that contains handwritten digits images



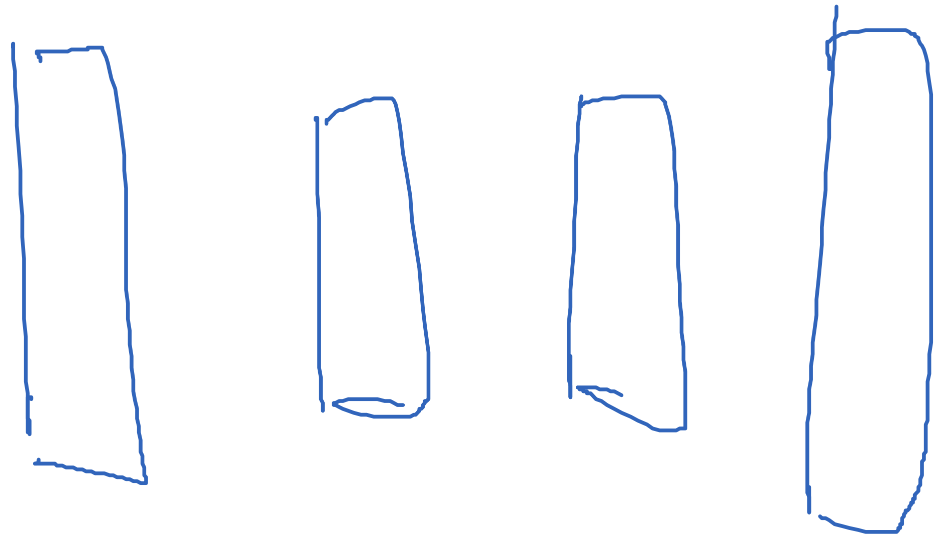


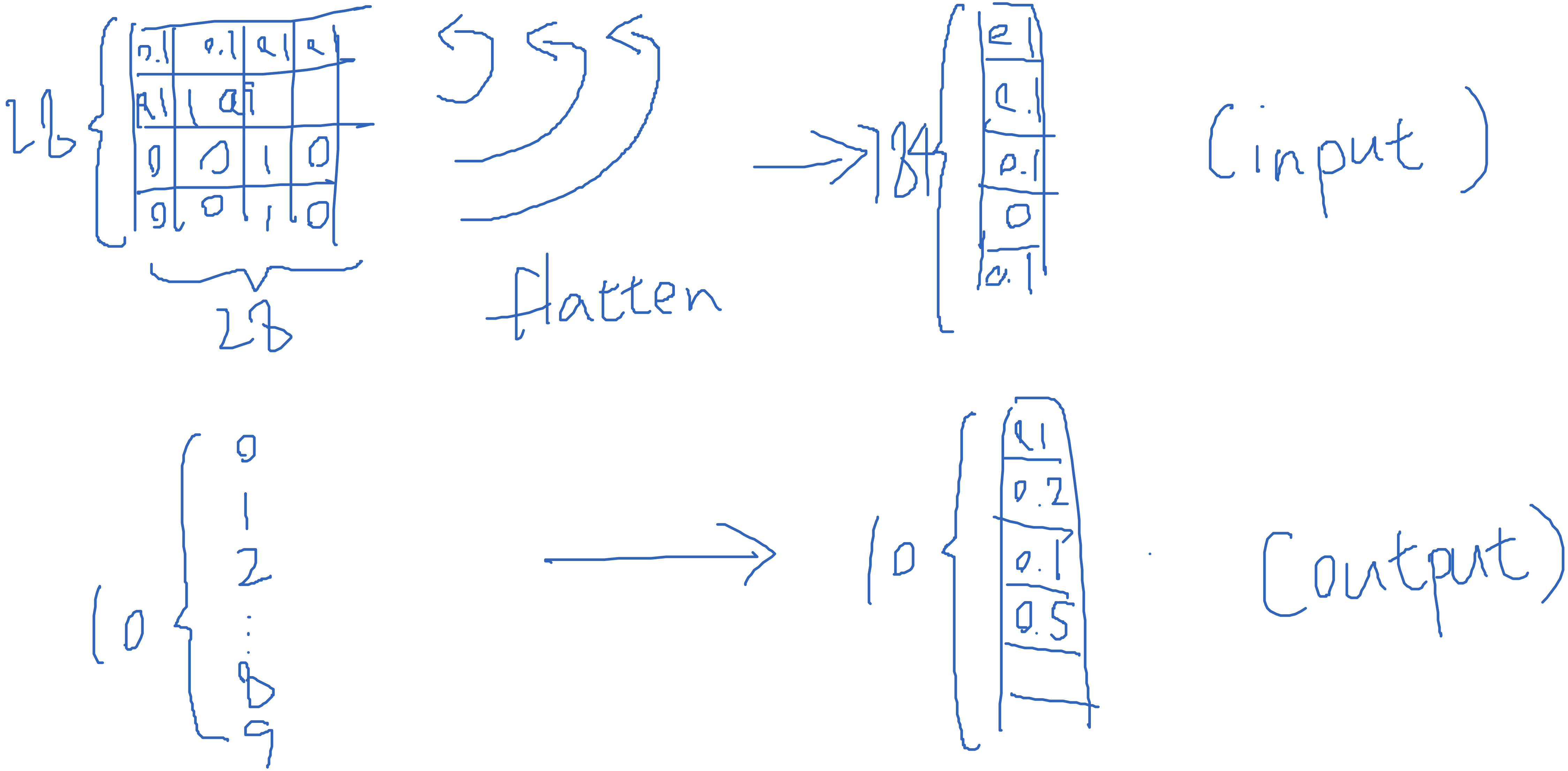
→ 3

- we can see our eye's retina as a matrix and when we see 3
- the corresponding locations lights up

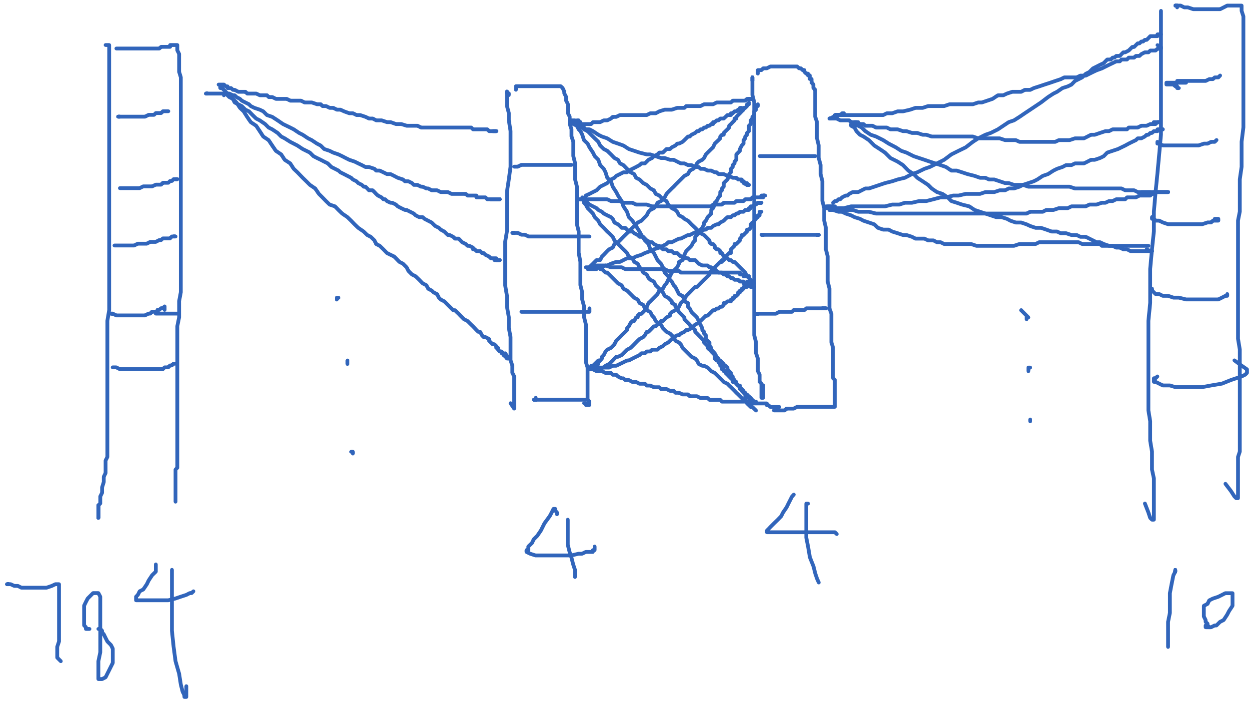


- Now we want to build a computer program that is able to do the same
- Neural Network

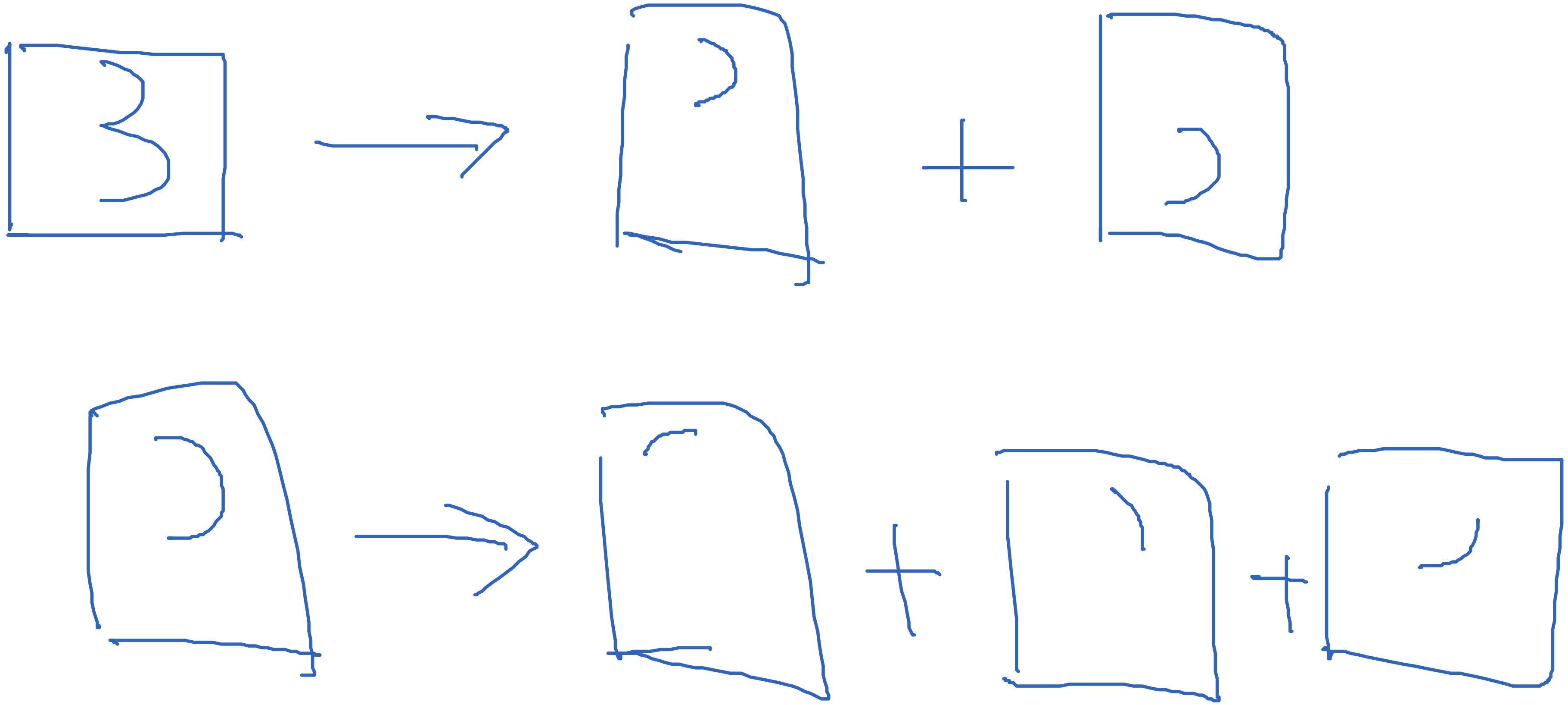



28
28
flatten
784
(input)
10
0
1
2
6
9
10
(output)

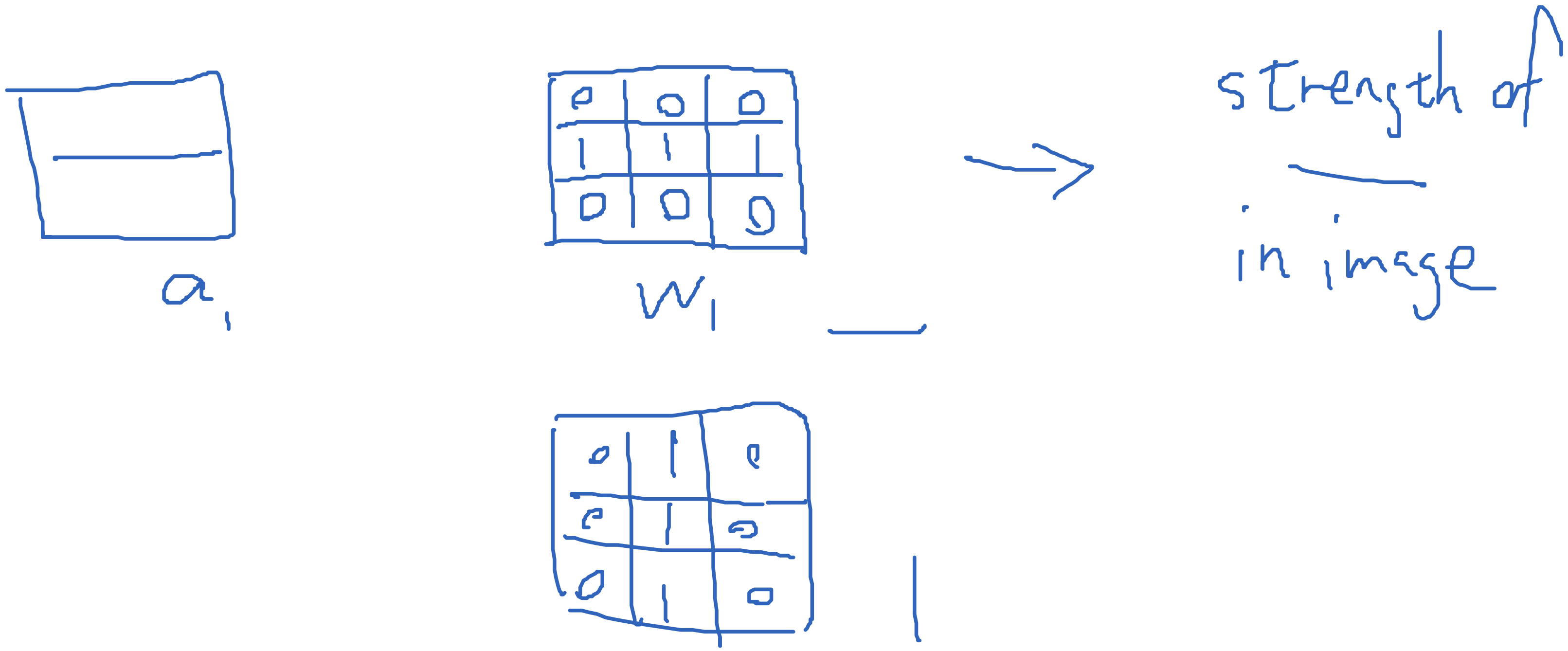
- the neurons in hidden layer and the number of hidden layer can be any arbitrary number
- you need experimentation to determine what's the best for the task



An explanation of how we recognise 3



- we hope that the hidden layer will do the same

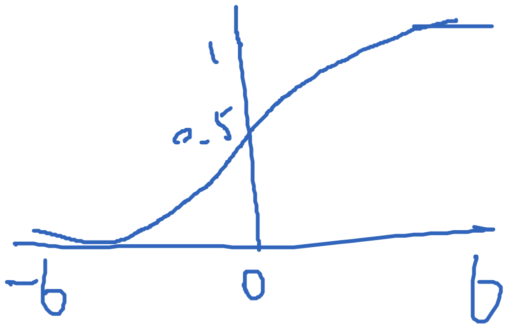


- we can compute weighted sum

$$w_1 a_1 + w_2 a_2 + \cdots$$

- we can also compare weighted sum, so that we know which is more related to the desire number
- to do this we use a function called activation function
  - eg sidmoid

$$\sigma : \frac{1}{1+e^{-x}}$$



$$x \in R \quad \sigma \in (0,1)$$

- then we have

$$\sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 \ldots)$$

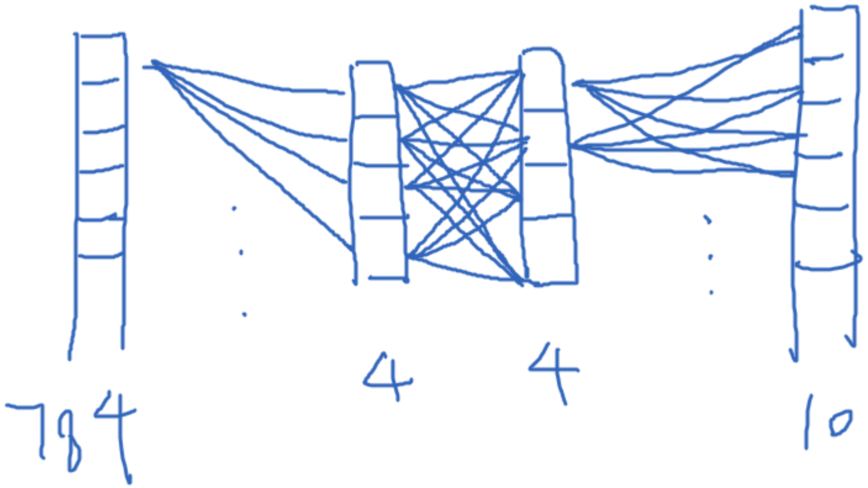
- we can also say that we only want this to be activated when weighted sum is > 5

$$\sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 \ldots -5)$$

- This represents the compute flow from 1 layer to anther layer

Parameters(weights)

- In total we have the following parameters



weights: 784x4 4x4 4x10 = 3192

bias: 4 4 10 = 18

= 3210

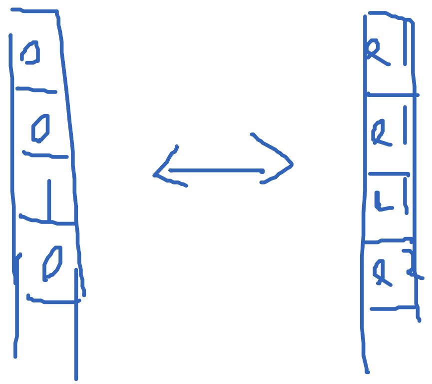
- Manually configure all 3210 parameters is are not feasible
- Thus we have machine learning
- The learning part is to let the computer find the appropriate weight and bias

- We first need to compute the output
- We first initialise all weights randomly and feed the data layer by layer to obtain output, this is called forward pass

$$\boxed{3} \to a^1 \to a^2 \to \text{output}$$

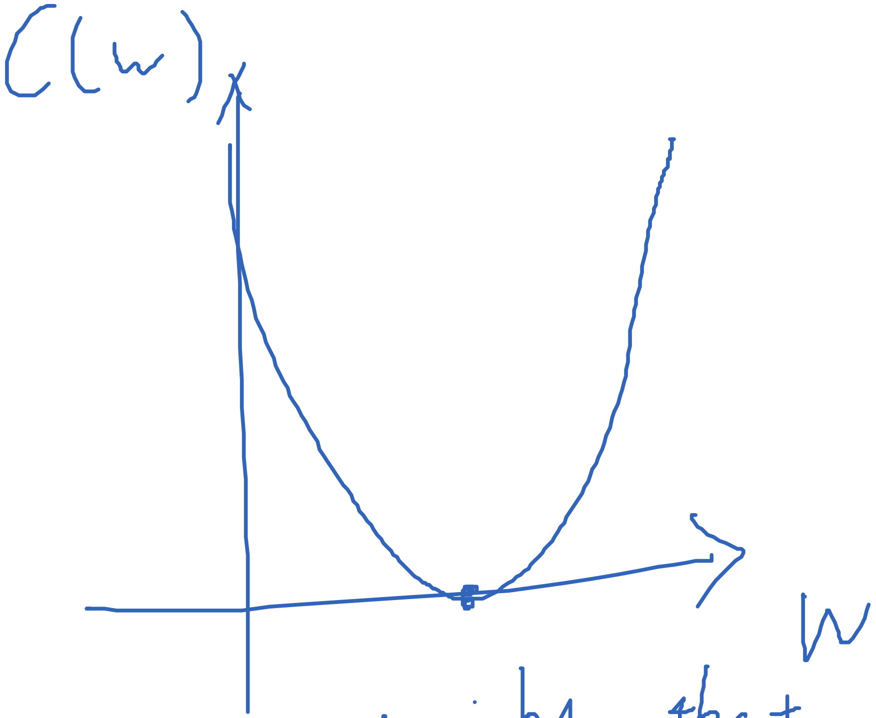
output: $\begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.2 \end{bmatrix} \to 4$

- the output is random, because random weights
- to let the model learn the parameters, we need to know how much the current network is not doing right



$\sum (\hat{y} - y)^2 \rightarrow$ cost of 1 training example

Simple relation between weight and cost



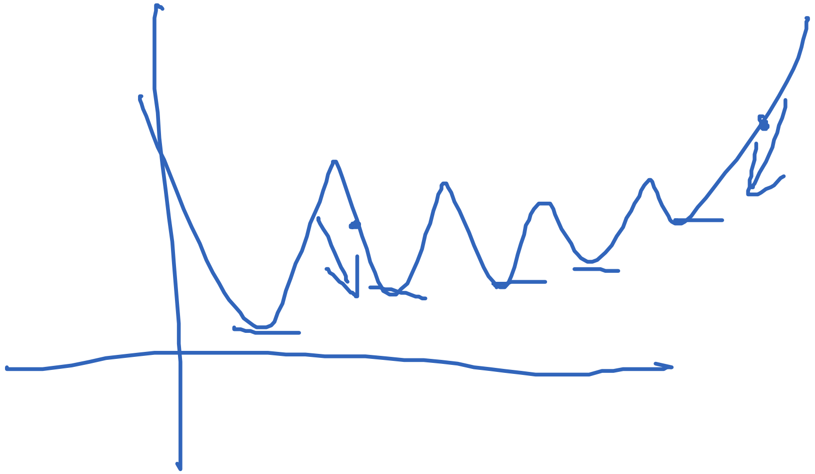
weight that optimise C(w)

find

- we can find out the derivative of C with respect to w
- derivative which measures the how much given parameters affect steepness of the function
- a partial derivative, which measures the how much one parameter affect steepness of the function holding other parameters constant
- the minimal Cost has the following properties

$$\frac{dC}{dw} C(w) = 0$$

- In reality the Cost vs weights graph is a lot more complex



- We can’t calculate the minimum value of C straight away
- We use a strategy called gradient descent

- The idea is
- shift left if the slope is positive
- shift right if the slope is negative
- These will lower the cost, and once the Cost gets smaller, this means we are closer to the real prediction
- repeat this step will reach the local minima

$$\vec{w} \begin{bmatrix} \\ \\ \\ \end{bmatrix} \Rightarrow \underbrace{-\nabla C(\vec{w})}_{} = \begin{bmatrix} 0.7 \\ -0.5 \\ \vdots \end{bmatrix}$$

negative gradient

each weight has different gradient

- The algorithm for computing this gradient efficiently is called back propagation

forward pass $a^0 \rightarrow a^1 \rightarrow a^2 \rightarrow a^3$

backprop $a^{L-3} \xleftarrow[w^{L-2}\, b^{L-2}]{} a^{L-2} \xleftarrow[w^{L-1}\, b^{L-1}]{} a^{L-1} \xleftarrow[w^{L}\, b^{L}]{} a^{L}$

$$\begin{cases} \text{Cost } C = (\underline{a^L} - y)^2 \\ a^L = \sigma(z^L) \\ z^L = w^L \underline{a^{L-1}} + b^L \end{cases}$$

$$\underline{w^L b^L} \cdot a^{L-1} \rightarrow z^L \rightarrow a^L \rightarrow \underline{C}$$

$$\frac{dC}{dw^L} = \frac{dz^L}{dw^L} \frac{da^L}{dz^L} \frac{dC}{da^L}$$

$$C = (a^L - y)^2$$

$$a^L = \sigma(z^L) \longrightarrow \frac{1}{1+e^{-z^L}}$$

$$z^L = w^L a^{L-1} + b^L$$

$$\frac{dC}{da^L} = 2(a^L - y) \qquad \left[\frac{1}{u(x)}\right]' = \frac{u'(x)}{u(x)^2}$$

$$\frac{da^L}{dz^L} = \sigma'(z^L) \longrightarrow \frac{d}{dz}\left[\frac{1}{1+e^{-z}}\right] = \frac{-\frac{d}{dz}\left[1+e^{-z}\right]}{(1+e^{-z})^2}$$

$$= -\frac{-1 \times e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{e^z}{(e^z+1)^2}$$

$$\frac{dz^L}{dw^L} = a^{L-1}$$

we have

$$\frac{dC}{dw^{(L)}} = a^{L-1} \cdot \sigma'(z^L) \cdot 2(a^L - y)$$

- this is just for 1 training example,
- such that this image want to change this parameter this way, another image want to change this parameter another way
- to change weight so that it is suitable for all training examples we average them

$$\frac{dC}{dw^L} = \frac{1}{n} \sum_{k=1}^{n} \frac{\partial C_k}{\partial w^L}$$

- Put everything together, we have

forward pass $a^0 \rightarrow a^1 \rightarrow a^2 \rightarrow a^3$

backprop $a^{L-3} \xleftarrow[w^{L-2}\, b^{L-2}]{} a^{L-2} \xleftarrow[w^{L-1}\, b^{L-1}]{} a^{L-1} \xleftarrow[w^{L}\, b^{L}]{} a^L$

$$
\begin{tikzcd}
x \arrow[r] & z^{L-2}\ a^{L-2} \arrow[r] & z^{L-1}\ a^{L-1} \arrow[r] & z^{L}\ a^{L} \arrow[r] & C \\
w^{L-2} \arrow[ur] & w^{L-1} \arrow[ur] & w^{L} \arrow[ur] & & \\
b^{L-2} \arrow[uur] & b^{L-1} \arrow[uur] & b^{L} \arrow[uur] & y \arrow[uur] &
\end{tikzcd}
$$