DMML 12 Mar 2019

Neural networks

Perceptron:

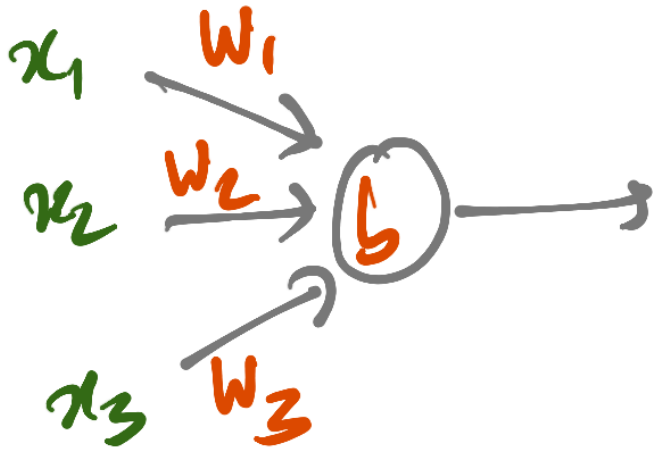
$$w \cdot x > \text{threshold } t \rightarrow 1$$

$$\leq t \rightarrow 0$$

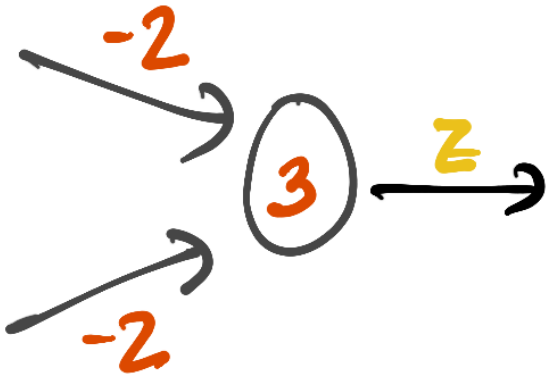
bias $b = -$ threshold t

$$w \cdot x + b > 0 \rightarrow 1$$

$$w \cdot x + b \leq 0 \rightarrow 0$$



Perceptron for NAND function



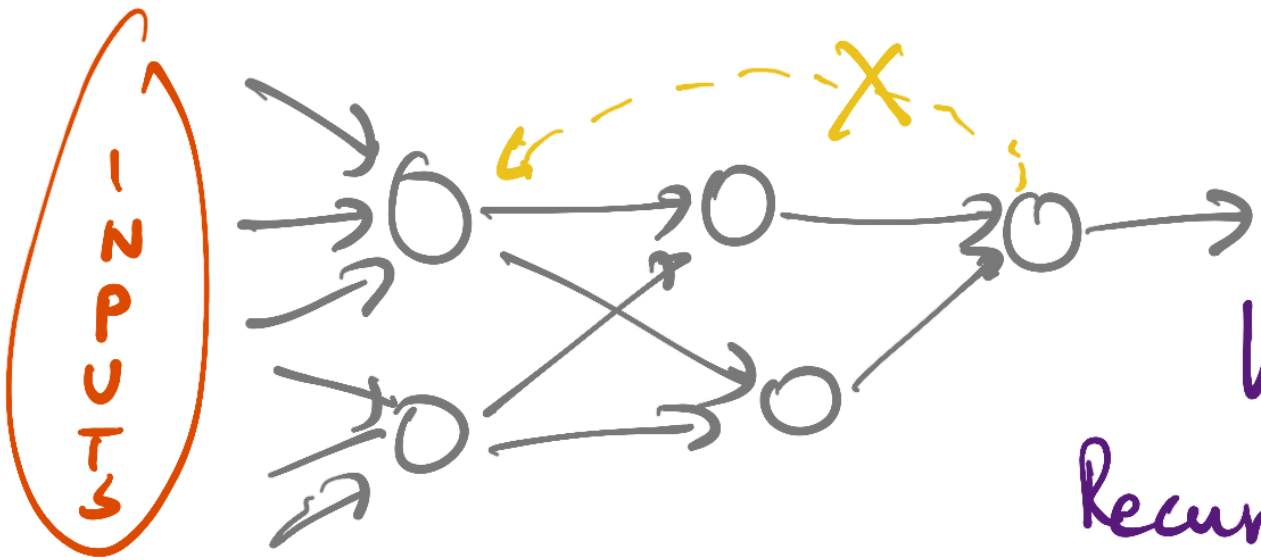
x_1	x_2	z	output
0	0	3	1
0	1	1	1
1	0	1	1
1	1	-1	0

$\neg (x_1 \wedge x_2)$

At the level of boolean functions, NAND is universal

→ Networks of perceptrons can compute any boolean function

Network

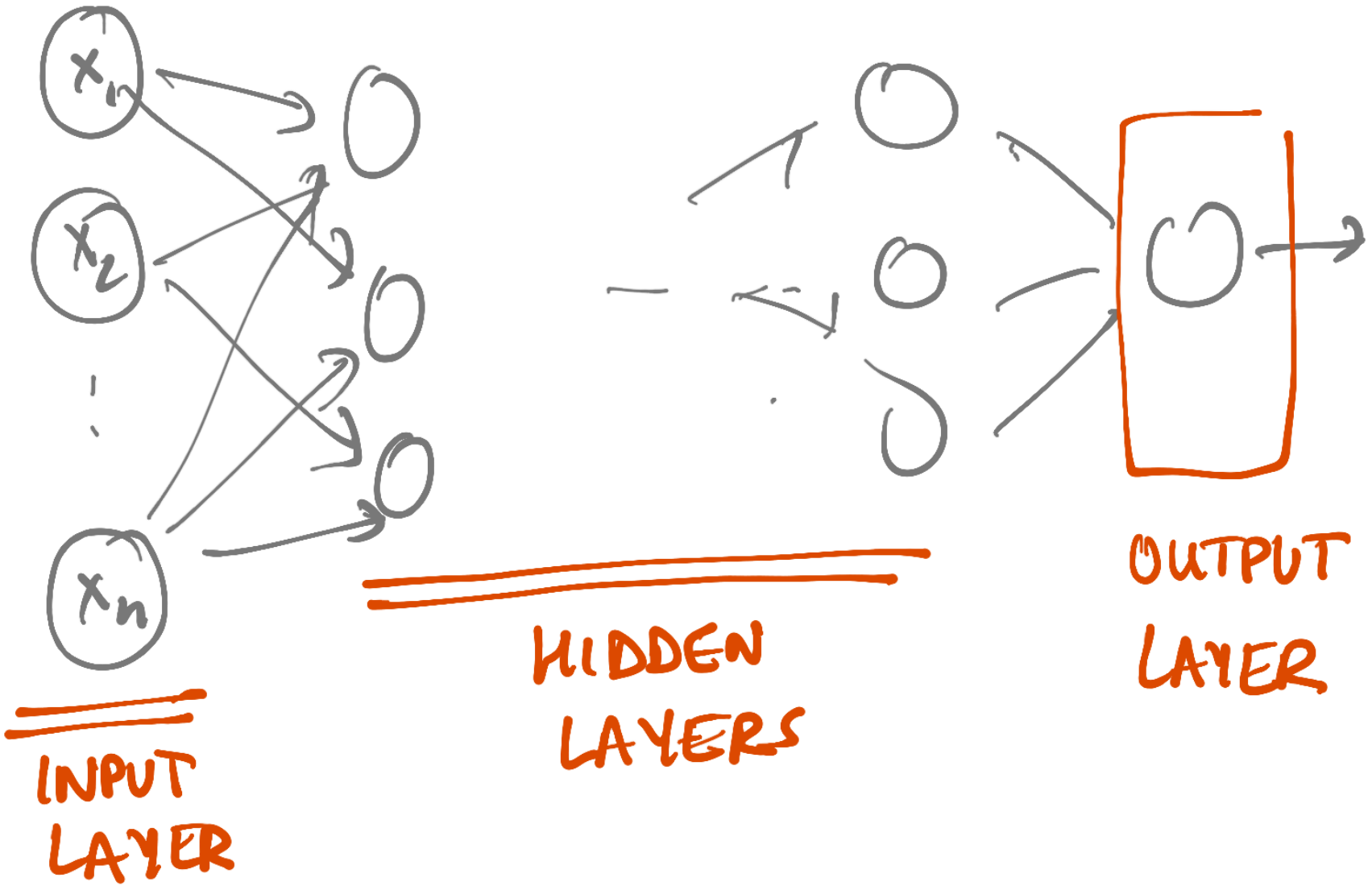


Acyclic

With cycles: Recurrent NN

Convention

Input layer of pseudo-perceptrons



What determines behaviour of network?

- Each node has w_i's & b
- Collectively, all weights & biases determine output

Single node

- learn weights & bias from training data
  - Iterative update (perceptron algo)
  - Optimize gap (SVM)

Only practical approach is gradient descent

Cost = Error Mean Square Error

$$\frac{1}{2n}\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2$$

$\downarrow$

w_i, b

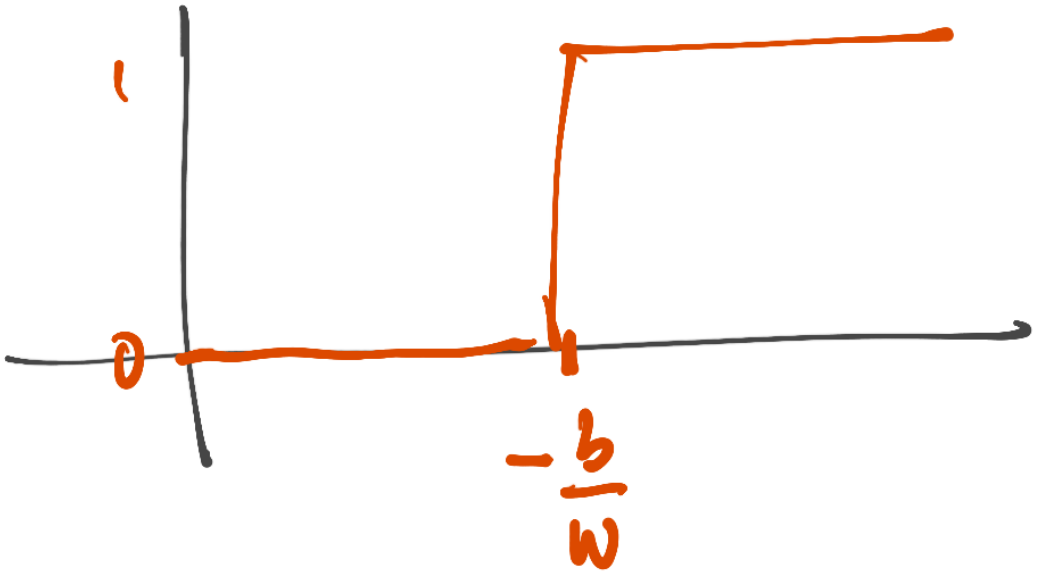
$$\Delta C = \alpha_1 \Delta w_1 + \alpha_2 \Delta w_2 \cdots + \alpha_{n+1} \Delta b$$

$$\frac{\partial C}{\partial w_1} \quad \frac{\partial C}{\partial w_2} \quad \cdots \quad \frac{\partial C}{\partial b}$$

Step function

$W \cdot x + b > 0 \rightarrow 1$

$W \cdot x + b \leq 0 \rightarrow 0$



Output is not "stable" at boundary $-\frac{b}{w}$

Instead, use sigmoid

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

$z \to \infty, \sigma(z) \to 1$

$z \to -\infty \; \sigma(z) \to 0$

Smooth step

Slope of the sigmoid is proportional to $\|w\|$

But now $\frac{\partial C}{\partial w_i}$, $\frac{\partial C}{\partial b}$ are better behaved

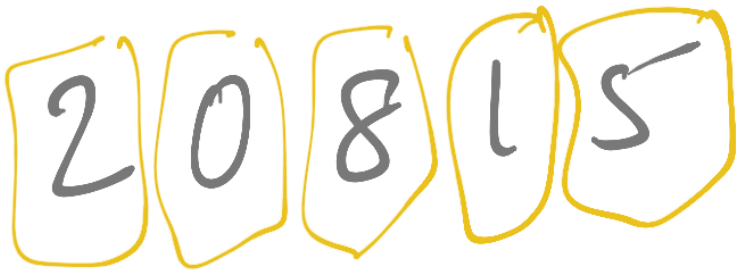
How it is applied in practice

Handwritten digits 0 1 2 ... 9

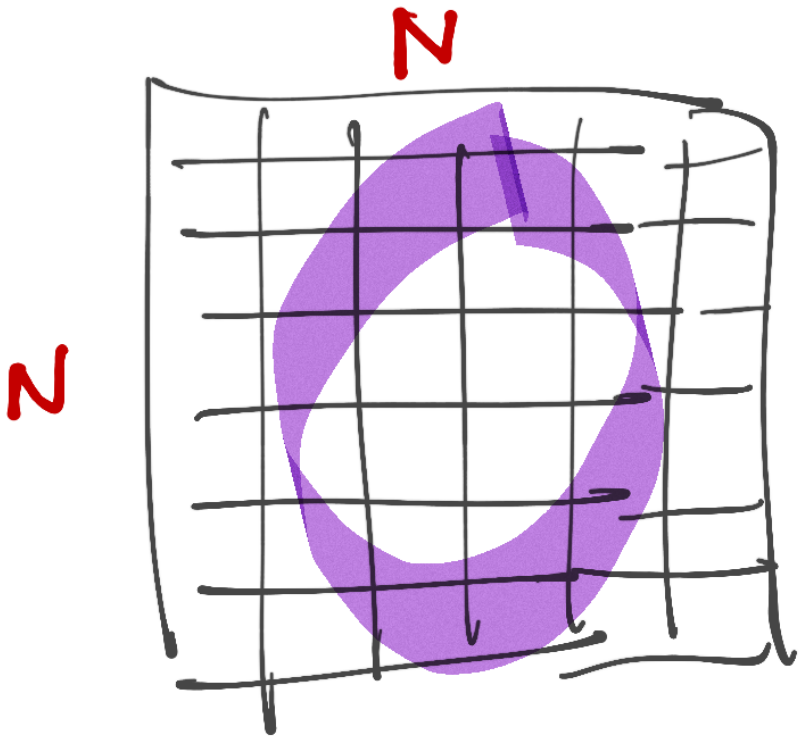
US Postal service

Identify where the ZIP code is

Segment ZIP code into individual digits



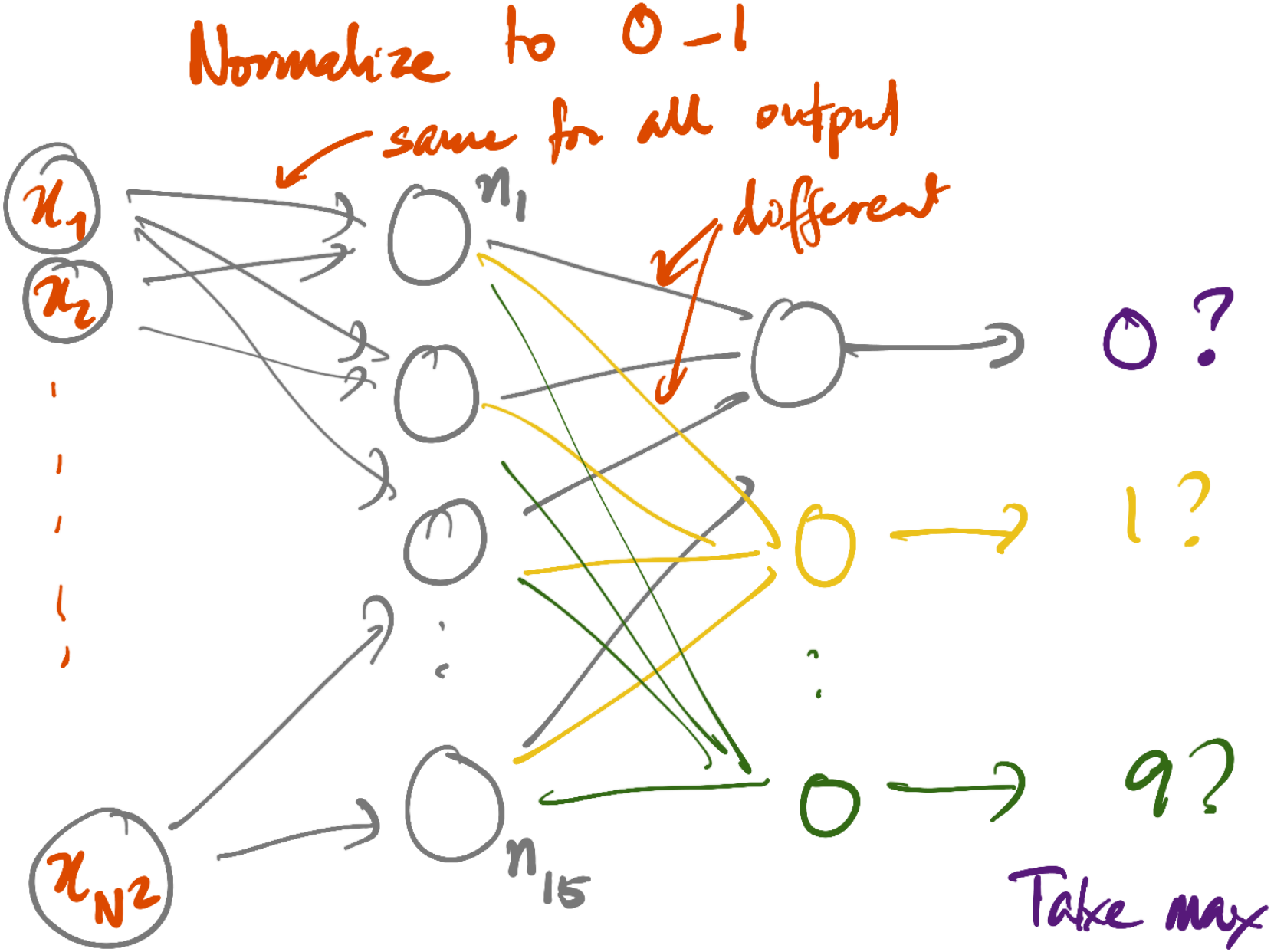
Digital scan of some resolution

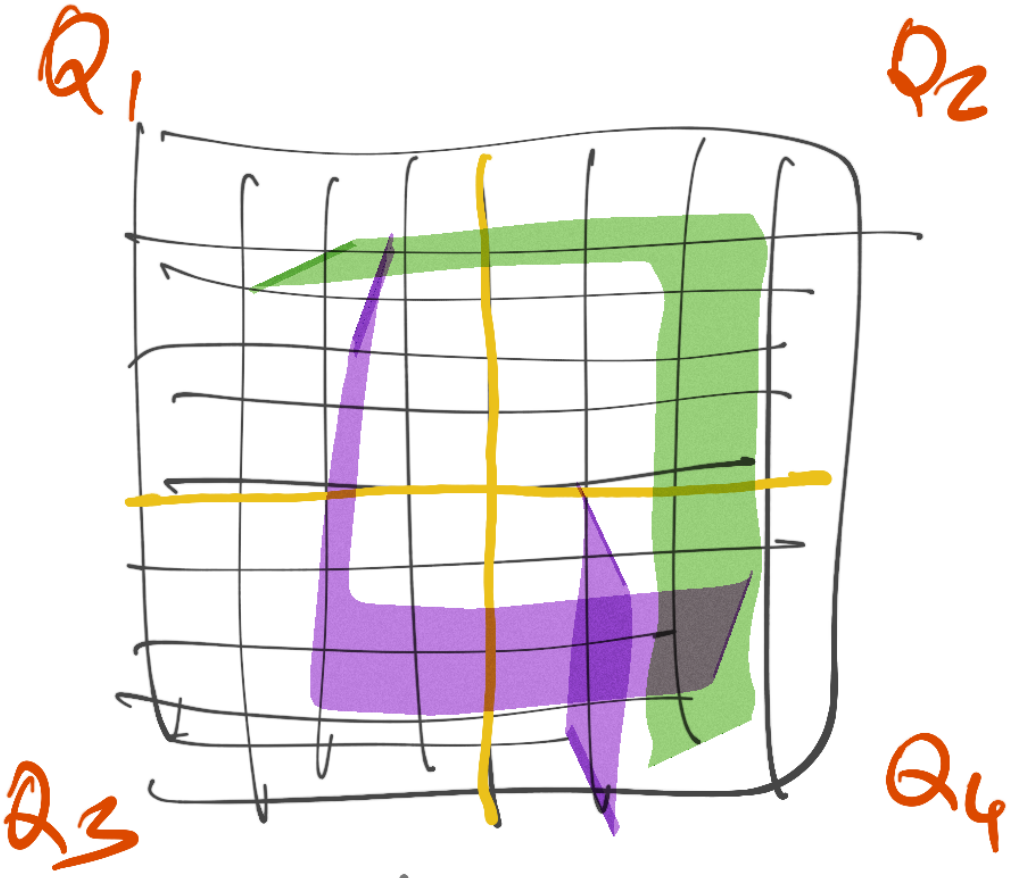


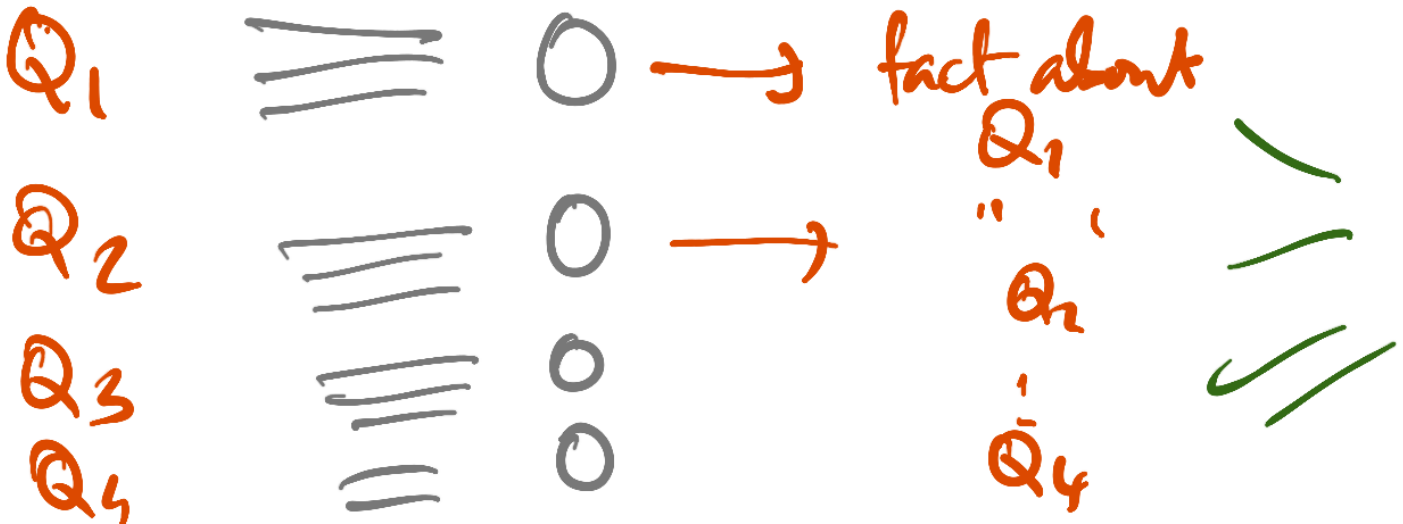
Each pixel is an input x_i

$$\underbrace{x_1, x_2 \cdots x_N}_{\text{row 1}}, x_{N+1}, \cdots, x_{N^2}$$

x_i : grayscale value 0 – 255




Q1
Q2
Q3
Q4
HIDDEN LAYER


Q1
Q2
Q3
Q4
fact about
Q1
"
Q2
Q4

Big technical step

- Implementing gradient descent efficiently
  - Large number of parameters with complex dependencies

Back Propagation

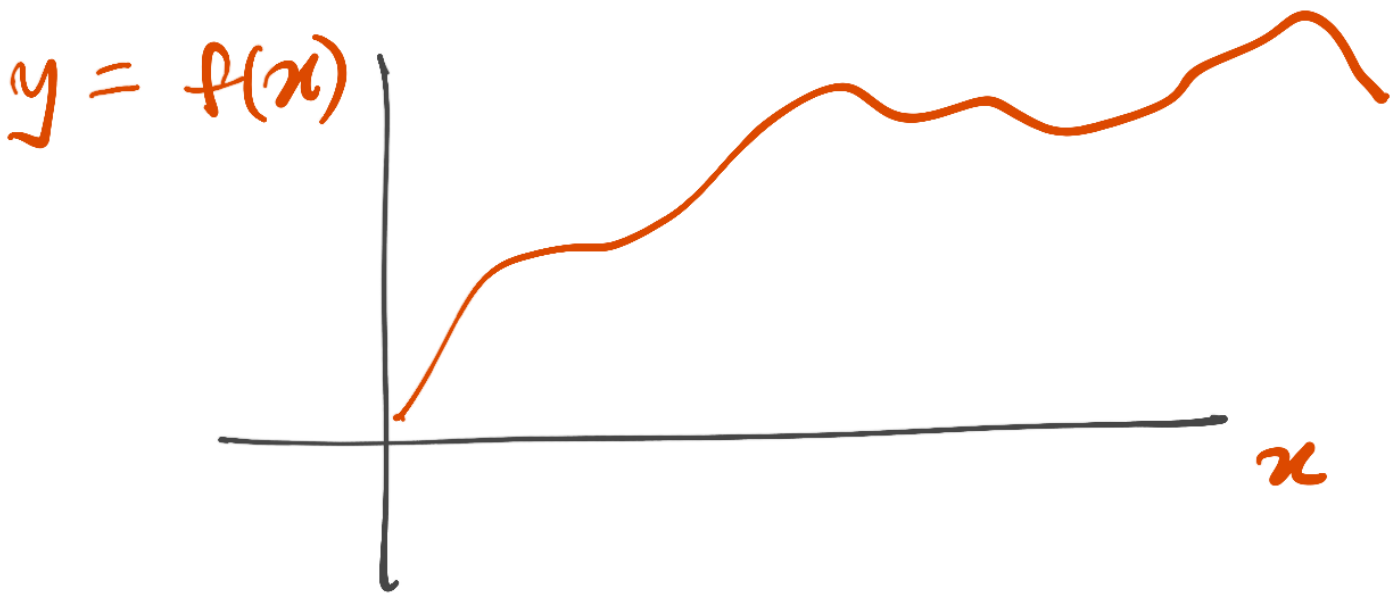
$\frac{\partial C}{\partial w_i}, \frac{\partial C}{\partial b}$ for internal nodes

- More parameters ⇒ more training data

Universality of the model

- Boolean context — NAND
- Grayscale inputs etc — not boolean

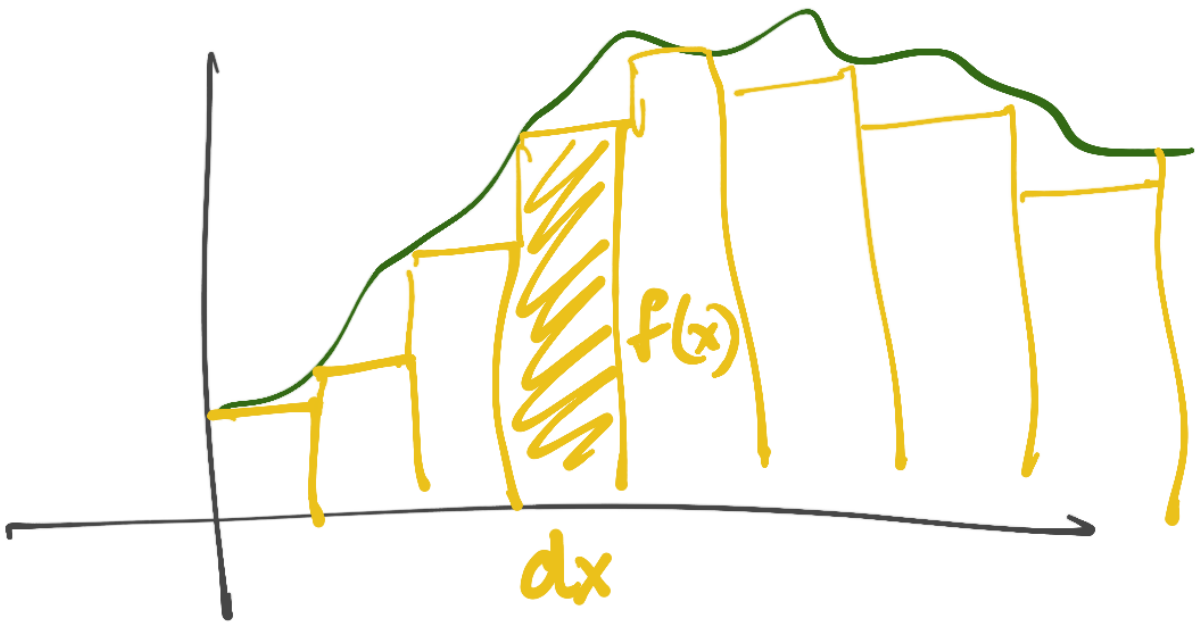
Simplest case: input is x_1 $(= x)$



Can I design a network that computes $f(x)$?

Recall calculus

Area under a curve?



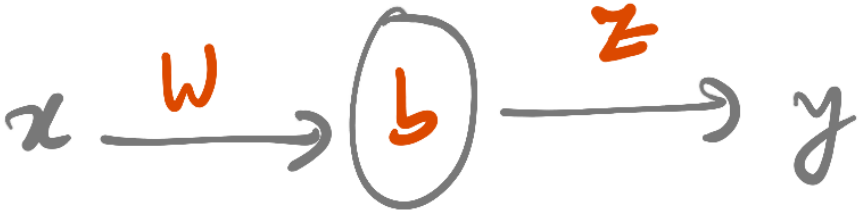
$f(x) \cdot dx$

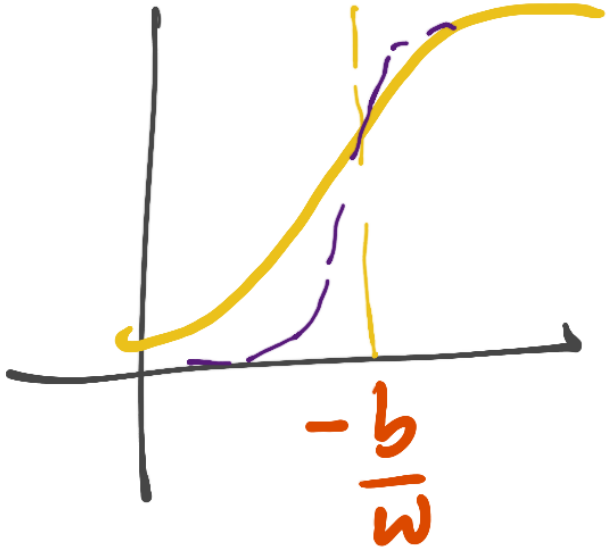
$dx \to 0$

this approximation converge

Similarly, produce a rectangular approximation of $f(x)$ using a neural network.

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





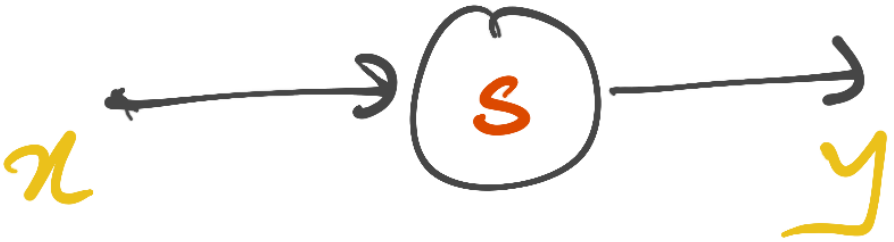
$W\uparrow \Rightarrow$ slope is steeper

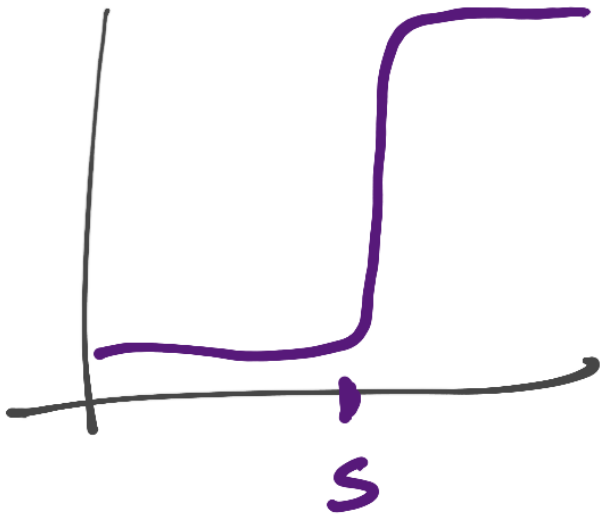
b determines the position of the step

$b\uparrow \Rightarrow$ step moves left

Alternatively,

$$s = -\frac{b}{w}$$

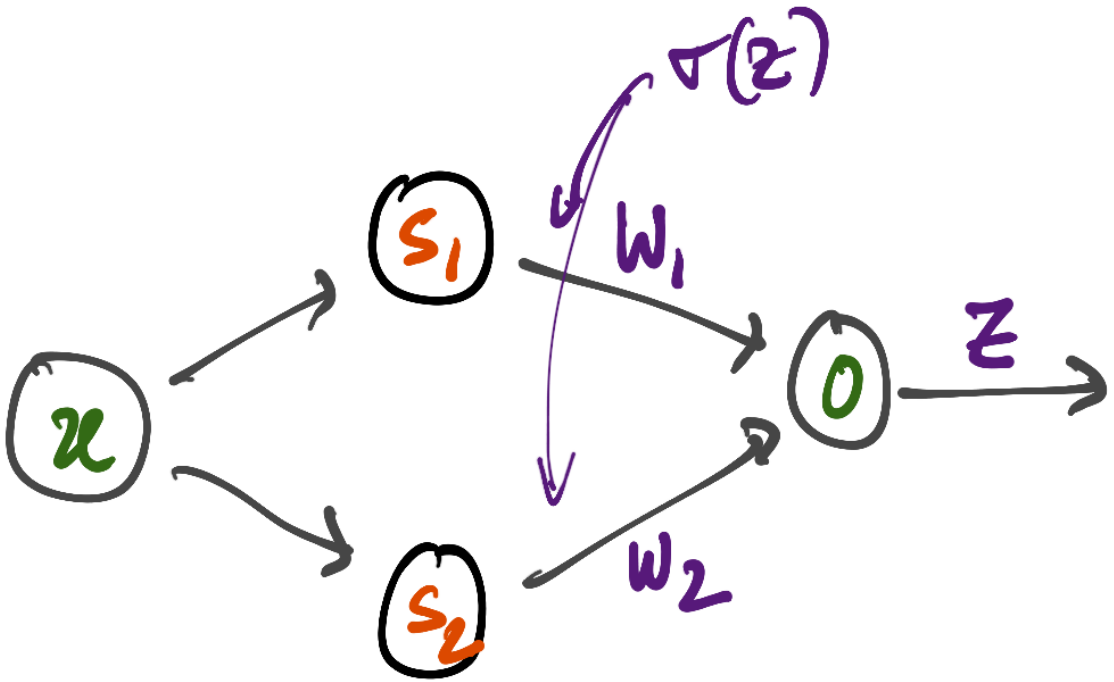


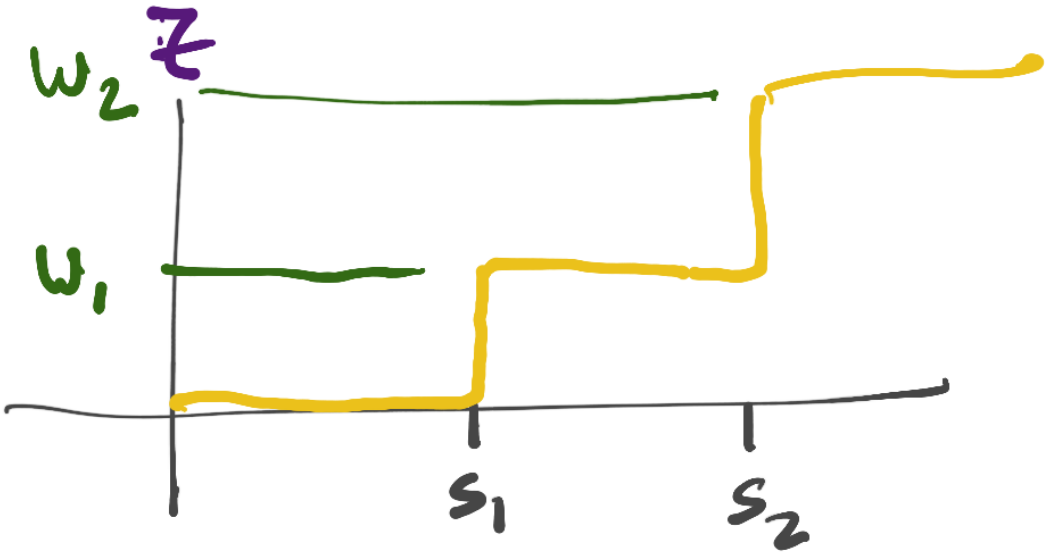


Assume w is large

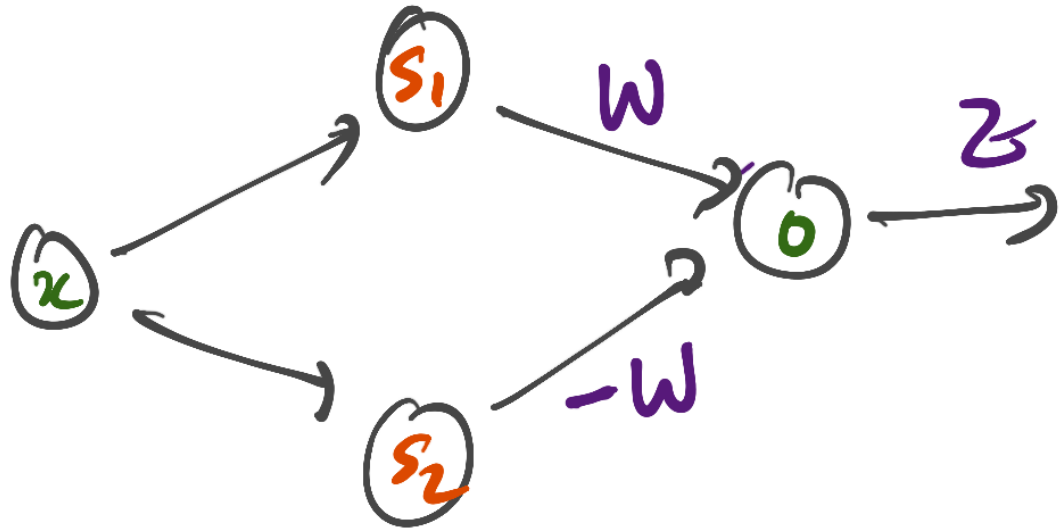
Nno

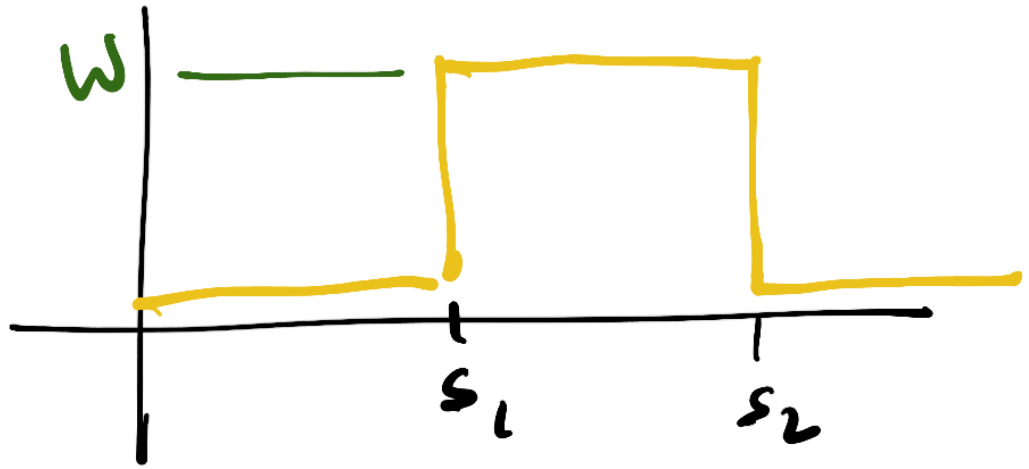
$s_1 < s_2$

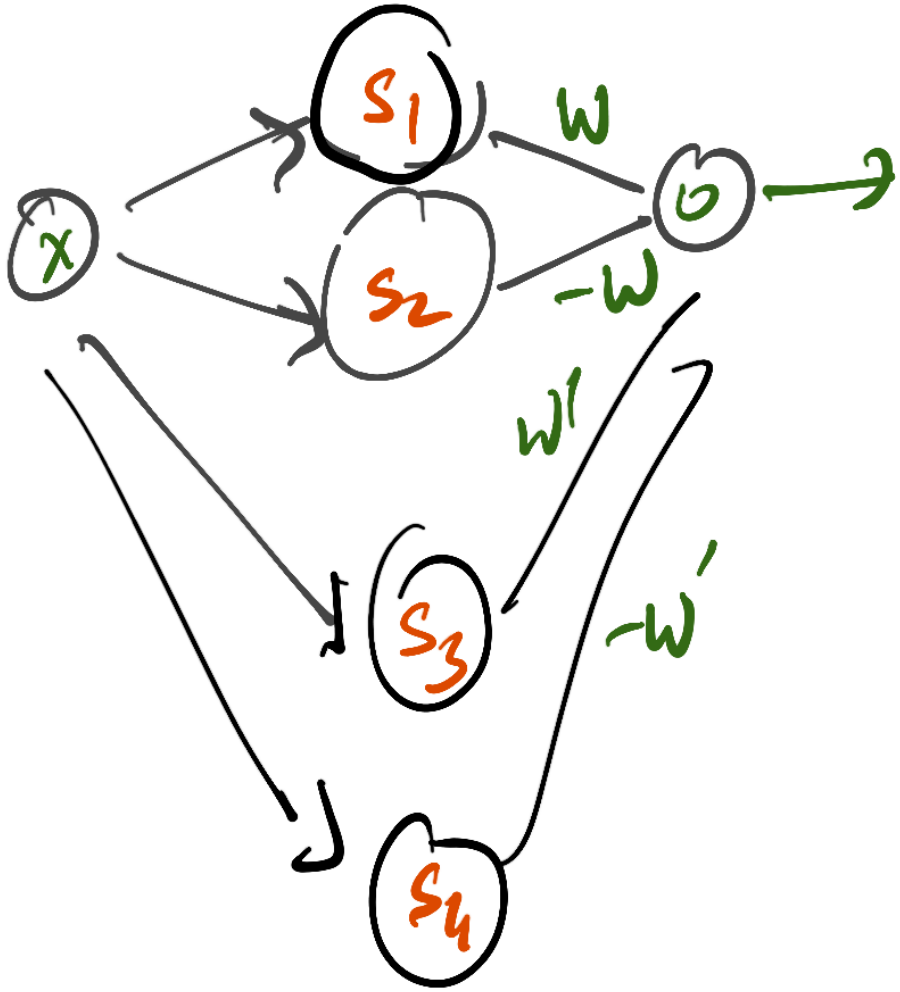


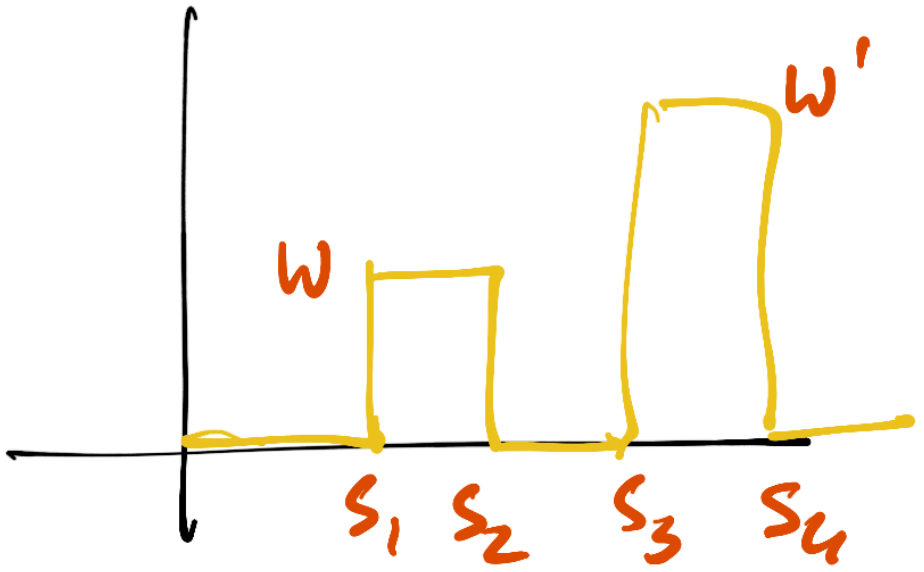


Therefore $s_1 < s_2$









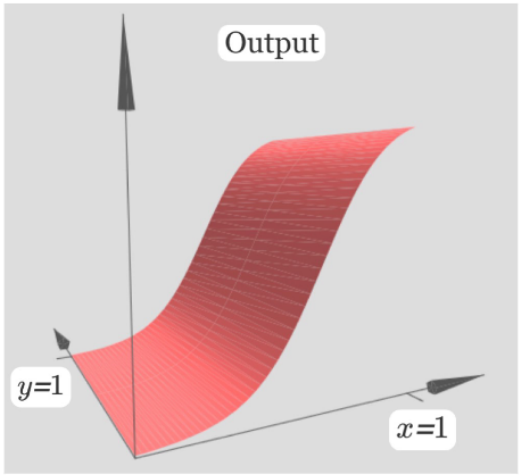
Quality of approx determined by $s_2 - s_1$ etc

Quality ↑ ⇒ size ↑

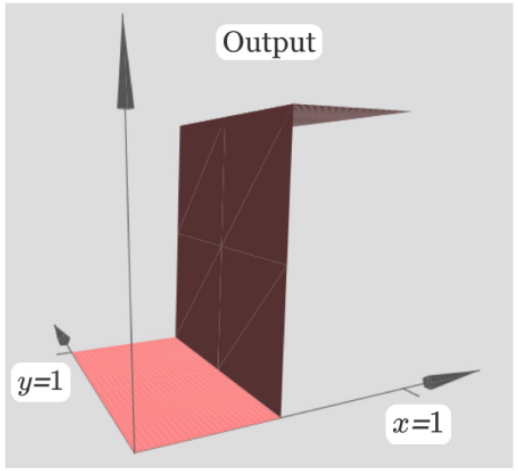
ONE HIDDEN LAYER

$y = f(x_1, x_2)$

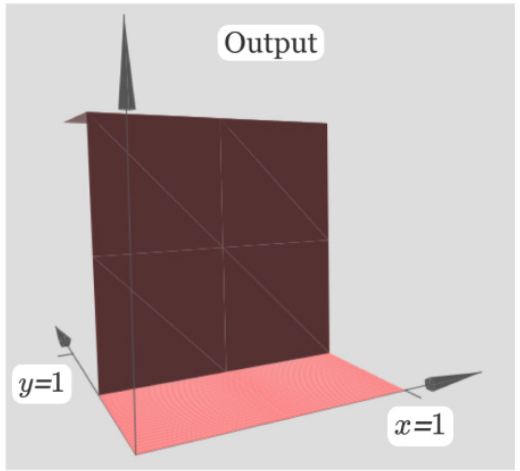
Seperately create steps in x_1 direction and x_2 direction



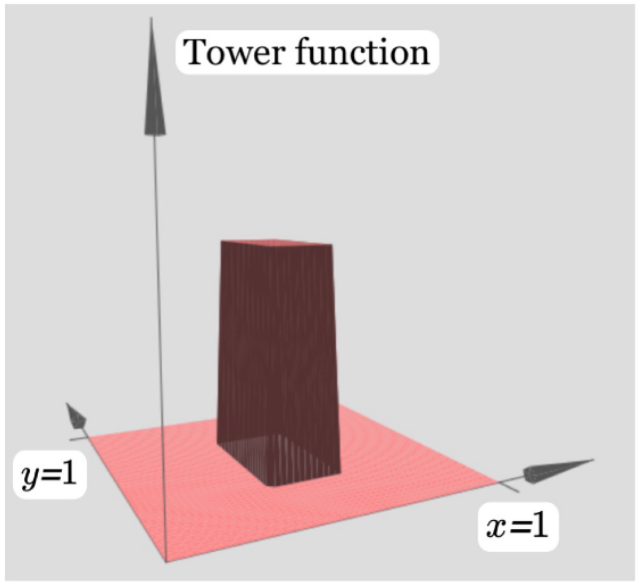
Surface to approximate



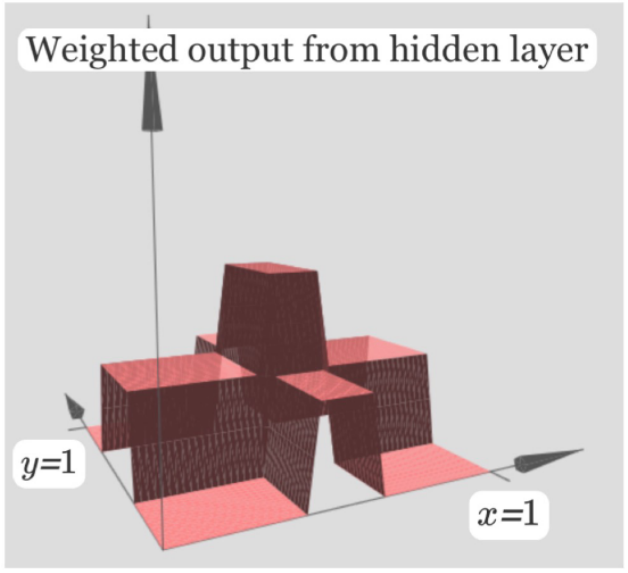
Step in x_1 direction



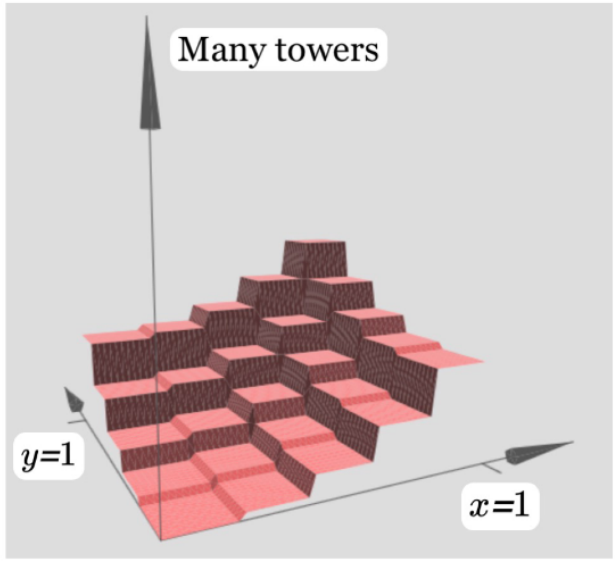
Step in x_2 direction



Combine x_1, x_2 steps to form cuboids



Stack cuboids



Cuboidal approximation of a surface