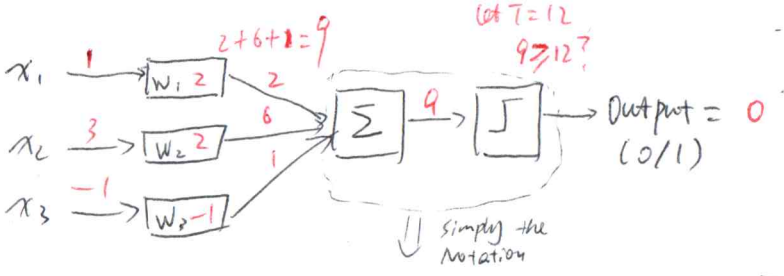
# Neural Network

Supervised ML

$\langle \vec{X}, Y \rangle \rightarrow$ [Alg] $\rightarrow$ Rule for classification $\langle \vec{X}, Y = ? \rangle$

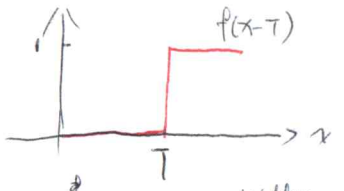
- kNN
- ID trees
- Neural Net
- SVM

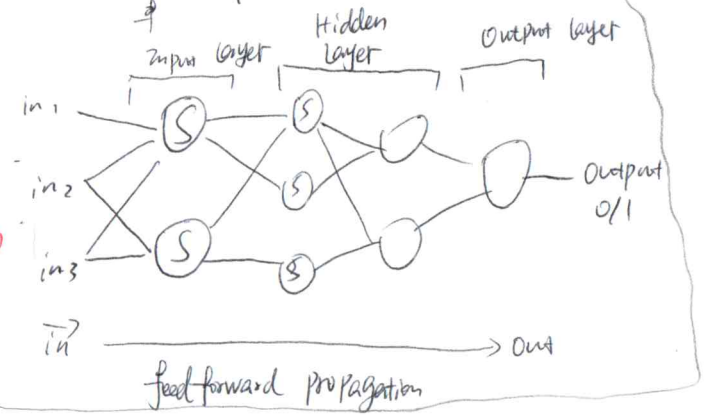


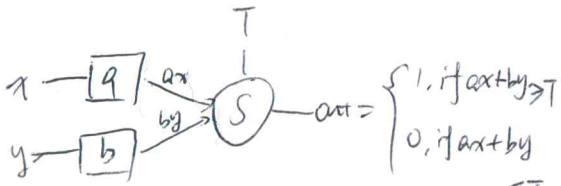

chain a bunch of these simple unit to build a larger network

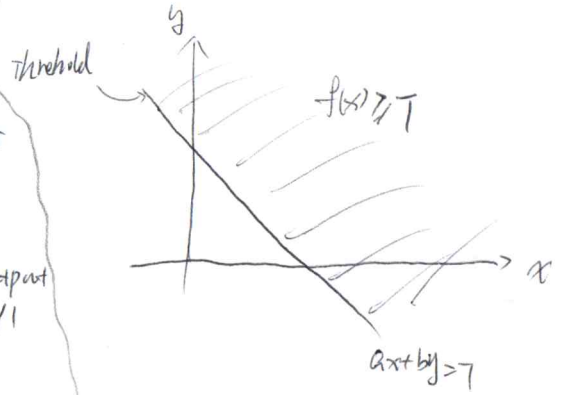
Threshold function:

$$\text{Stairstep}(x) = \begin{cases} 1, & \text{if } x \geq T=0 \\ 0, & \text{if } x < T=0 \end{cases}$$



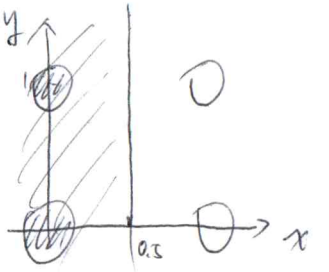



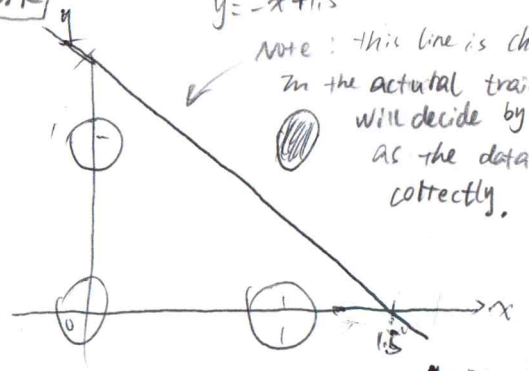

Not function

| x | y | ¬(x) |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



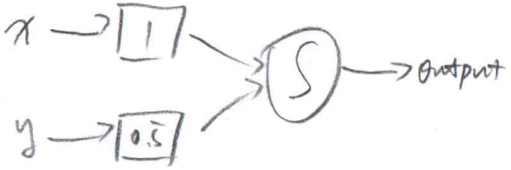

$x \leq 0.5$ (or $-x + 0y \geq -0.5$)

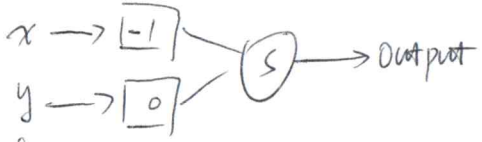
| x | y | x∧y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




Note: this line is chosen arbitrally, in the actural trainning, the NN will decide by itself, as long as the data were classified correctly.

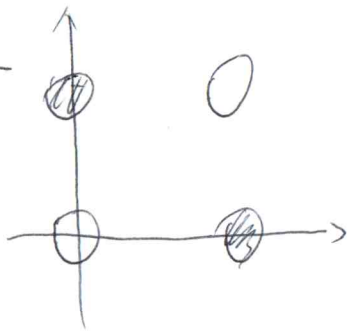
$y + x \geq 1.5$



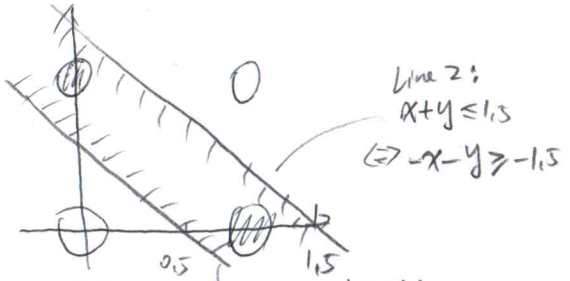




X or function

| x | y | x⊕y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

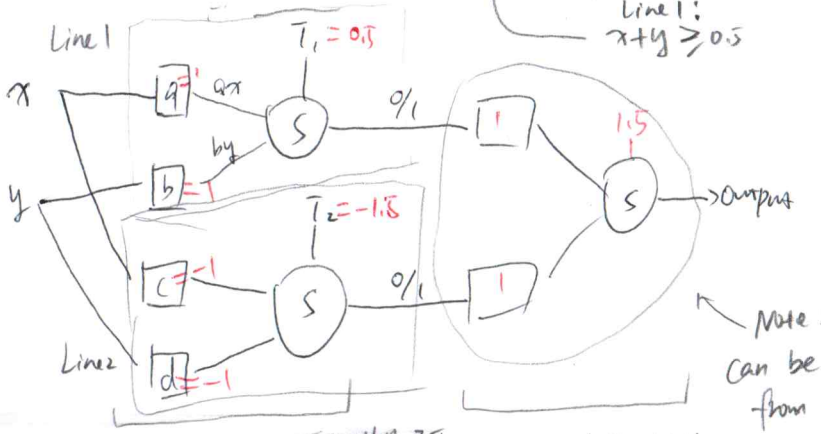


| $x$ | $y$ | $x \oplus y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




Line 2:
$x+y \le 1.5$
$\Leftrightarrow -x-y \ge -1.5$

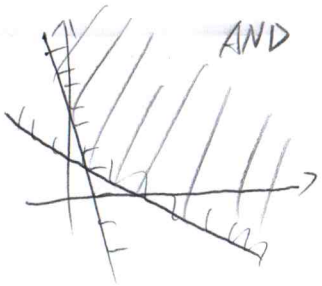
Line 1:
$x+y \ge 0.5$

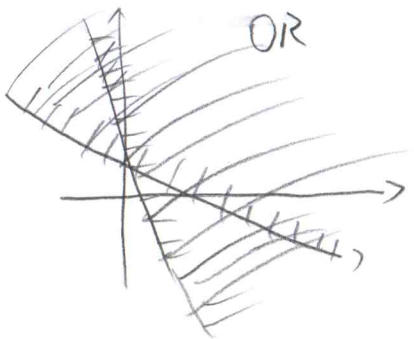



Note: this module can be copied directly from former work.

two line $\begin{cases} ax+by \ge T_1 \\ cx+dy \ge T_2 \end{cases}$ a logic unit

If logic unit = AND = OR




Let's test ~~out~~ design:

$x = 0, y = 0$

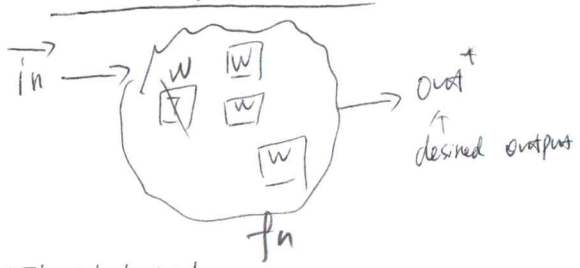
$\begin{cases} x+y = 0 \ge 0.5\ ? \Rightarrow 0 \\ -x-y = 0 \ge -1.5\ ? \Rightarrow 1 \end{cases}$ > AND → 0 ✓

$x = 0, y = 1$

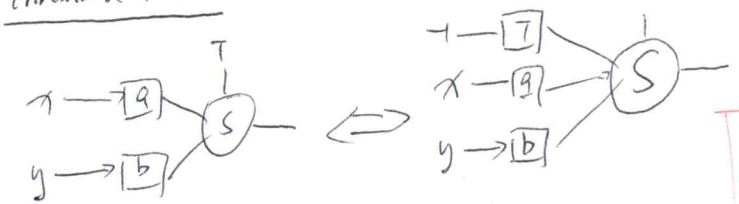
$\begin{cases} x+y = 1 \ge 0.5\ ? \Rightarrow 1 \\ -x-y = -1 \ge -1.5\ ? \Rightarrow 1 \end{cases}$ > AND → 1 ✓

# Neural Network

## Back propagation




## Threshold trick

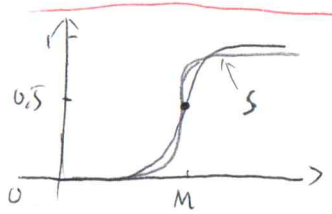



$x+y \geq T \iff x+y+(-1)T \geq 0$

Note: this is convenient in actual Trainning. Because we can treat threshold as Weigh Parameter as well, In Back-Prop, every parameters become Weight, so we just need to figure out a way to adjust weight.

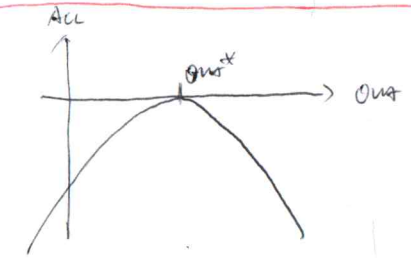
$$\text{Sigmoid}_{s,M}(x) = \frac{1}{1+e^{-s(x-M)}}$$

s: steepness, M: midpoint, e.g. $\frac{1}{1+e^{-x}}$




Note: You can't take derivative over stair step func, so approximated by Sigmoid.

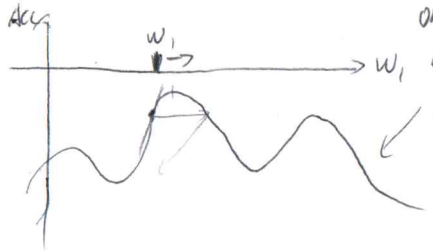
$$\text{Performance} = \text{Accuracy}(out^*, out) = -\frac{1}{2}(out^* - out)^2$$




Note: this give us a way to measure how far or closer we're to the actual output.

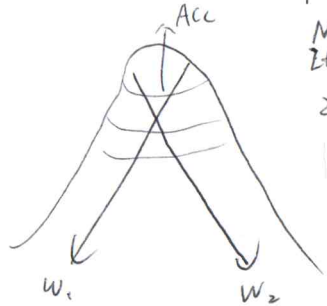
## Gradient ascent

$out = f_n(w_1, w_2, w_3 \ldots)$



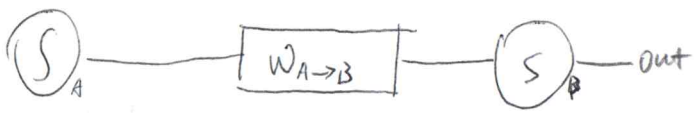

Note: If we only adjust one param and has all others fixed, our goal here is to increase the weight to get a higher Acc.

r = rate constant (step size)




Note: It's hard to draw over 2D, but the idea is same

e.g. Someone named John Conner built a NN, and he's trying to distinguish between Human shape terminator robots and real humans based on three input features: 1) whether misuse colloquial expressions. 2) whether has super strength. 3) Whether they're good with gun.

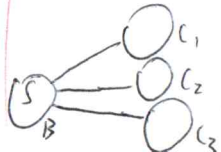



$$W_{A\to B,new} = W_{A\to B,old} + \Delta W_{A\to B}$$

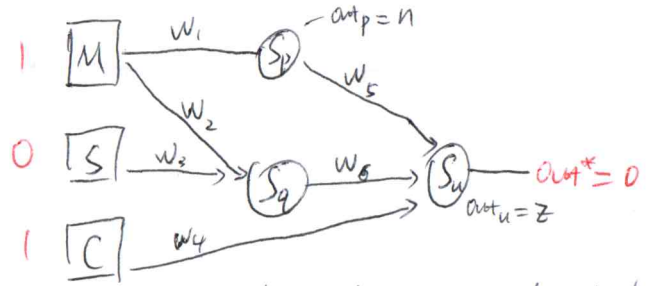
$$\Delta W_{A\to B} = r \cdot out_A \cdot \delta_B$$

$out_A$: the output of A; $\delta_B$: depends on the property of B.

$$\delta_B = \begin{cases} out_B(1-out_B)(out^* - out), & \text{if B is in final layer} \\ out_B(1-out_B)\sum_{C_i \in out\ going} W_{B\to C}\,\delta_C, & \text{if B is not in final layer} \end{cases}$$




Note: use Back-Prop to train NN, and use feed forward to classify points in NN.




Let's do an example to calculate the update weight for $W_1$:

$W_{1,new} = W_{1,old} + \Delta W_1$ (let $r = 10$)

$\Delta W_1 = r \cdot 1 \cdot \delta_p$ (in this case $out_A$ is not a neural, but an input)

$\delta_p = out_p(1-out_p)W_5\delta_u = n(1-n)W_5\delta_u$

$\delta_u = out_u(1-out_u)(out^* - out) = z(1-z)(0-z)$