Linearly Separable Boolean Functions

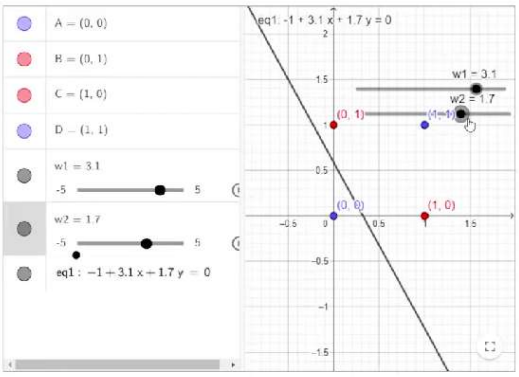
x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$w_0 + \sum_{i=1}^{2} w_i x_i < 0$



$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \Rightarrow w_0 < 0$

$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \Rightarrow w_2 \geq -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \Rightarrow w_1 \geq -w_0$

$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \Rightarrow w_1 + w_2 < -w_0$

2 & 3 conditions contradict condition 4

∴ it's not linearly separable

Most real world data will not be linearly separable & will always contain outliers!

A single perceptron cannot deal with such data.

However, a network of perceptrons can indeed!

How many boolean functions can you design from 2 inputs?

x_1	x_2	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Only 2 functions f_7 & f_{10} are not linearly separable

(XOR)(!XOR)

So for n inputs we can have: 2^{2^n} boolean functions

How many are not linearly separable?

So for n inputs we can have : 2

How many are not linearly separable ?

It's an unsolved problem !

Representation power of a network of perceptrons

We will see how to implement any boolean function using a network of perceptrons...

True = +1
False = −1

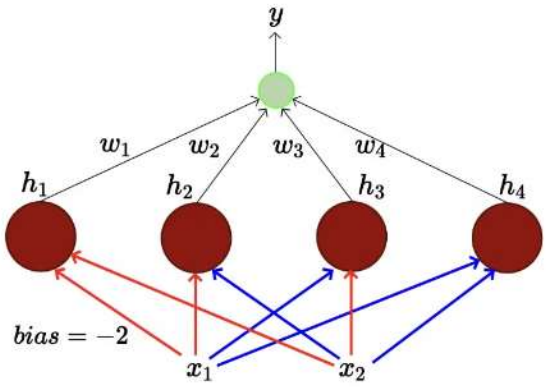
We will consider 2 inputs & 4 perceptrons

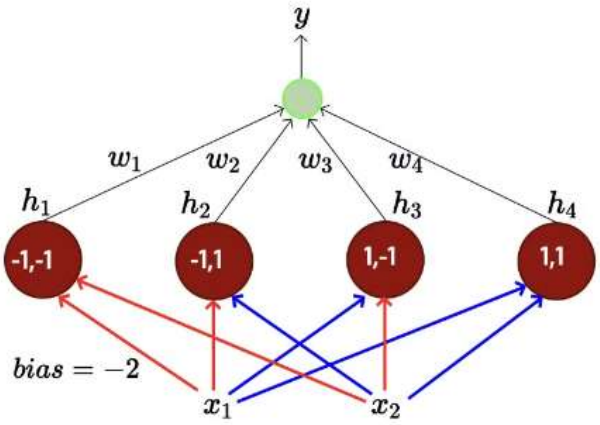
Each input is connected to all 4 perceptrons with specific weights

red $w = -1$,
blue $w = +1$
, bias $(w_0) = -2$

∴ perceptron will only fire if weighted sum of input will be ≥ 2

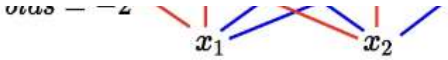
Each perceptron is connected to an output perceptron by weights (which needs to be learned)





Terminology

- This network has 3 layers
- The layer containing (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer is called the **output layer**
- The outputs of the 4 perceptrons are denoted by h_1, h_2, h_3, h_4
- Red & blue weights are called



- Red & blue weights are called layer 1 weights
- w_1, w_2, w_3, w_4 are layer 2 weights

We claim that this network can be used to implement any bookan function (linearly separable or not)

In other words: we can find w_1, w_2, w_3 & w_4 such that the truth table can represented by this network.

Each perceptron in the middle layer is gonna fire for a specific input.

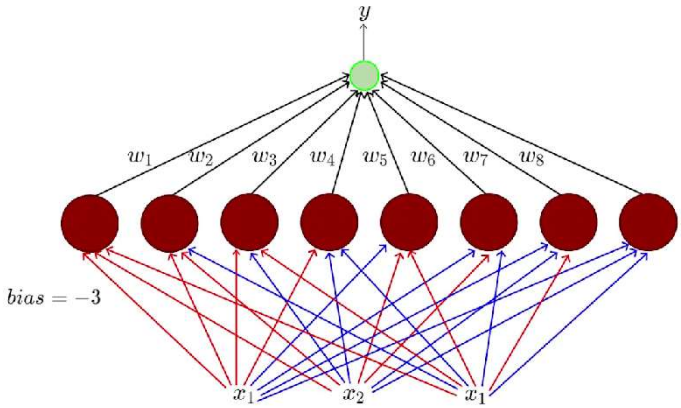
Let w_0 be the bias output of the neuron (i.e., it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	1	1	w_4

This results in the following four conditions to implement XOR: $w_1 < w_0$, $w_2 \geq w_0$, $w_3 \geq w_0$, $w_4 < w_0$

These conditions are not contradicting so we can choose proper values for them!

For 3 inputs:



what about n inputs*? 2^n perceptrons

what about n inputs ? 2^n perceptrons

containing 1 hidden layer

A network of 2^n+1 perceptrons is not necessary but sufficient

Networks of the form are called Multilayered Perceptrons

[MLP]

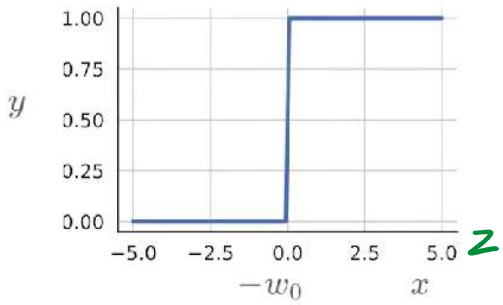
Sigmoid Neurons

We saw enough boolean functions!

What about arbitrary functions of the form: $y \in f(x)$ where $x \in \mathbb{R}^n$ & $y \in \mathbb{R}$?

Can we have a network which can (approximately) represent such functions?

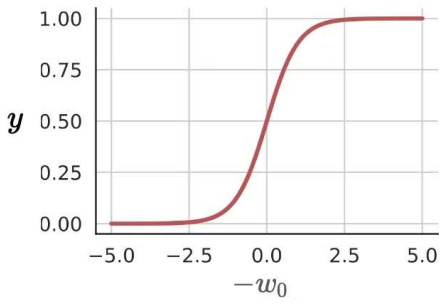
The threshold rule of perceptron is very harsh! It behaves like a step function



There is a sudden change in the decision (y) from 0 to 1 when we cross the threshold.

$$z = \sum_{i=1}^{n} w_i x_i$$

Introducing Sigmoid neurons where the output function is smoother



$$z = \sum_{i=1}^{n} w_i x_i$$

Here is one form of sigmoid function called logistic function:

$$y = \frac{1}{1 + \exp\left(-\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)\right)}$$

$w^T x$

$\sum_{i=1}$

$w^T x$

As y is no longer binary but is a real value b/w 0 & 1, it can be interpreted as probability

As $w^T x \to \infty$

$$y = \frac{1}{1+e^{-\infty}} = \frac{1}{1+\frac{1}{e^{\infty}}} = \frac{1}{1+0} = 1$$

As $w^T x \to -\infty$

$$y = \frac{1}{1+e^{\infty}} = \frac{1}{\infty} = 0$$

As $w^T x \to 0$

$$y = \frac{1}{1+e^{0}} = 0.5$$

Perceptrons

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i x_i \geq 0$$

$$0 \text{ if } \sum_{i=0}^{n} w_i x_i < 0$$

Not smooth, not differentiable at $(-w_0)$

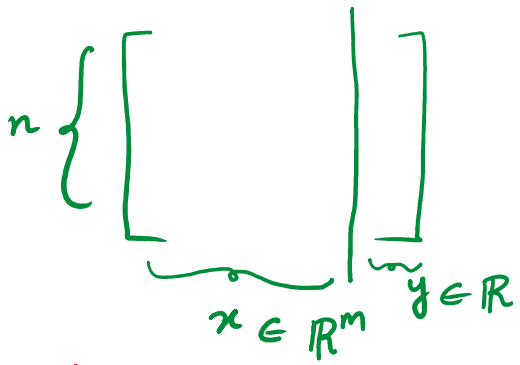
Sigmoid Neuron

$$y = \frac{1}{1+\exp\left(-\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)\right)}$$

Smooth, Differentiable

A typical Supervised Machine Learning Setup

Data: $\{x_i, y_i\}_{i=1}^{n}$



Model: Our approximation of the relation b/w x & y

eg: $y = \frac{1}{1+e^{-w^T x}}$ or $\hat{y} = w^T x$ or $\hat{y} = x^T W x$

$\therefore y = \hat{f}(x)$

Parameters: In the above examples w is the parameters

Learning Algorithm: An algorithm for learning the parameters w of the model

Objective/ Loss/ Error Function: To guide the learning algorithm to minimize the loss function

Example: Movie Going

Data: $\{x_i = \text{movie}, y_i = (\text{like/dislike})\}_{i=1}^{n}$

Model: Our approximation relation b/w x & y (probability of liking a movie)

$\hat{y} = \frac{1}{1+e^{-w^T x}}$ $\hat{y}$ -> approximation of y

Parameters: w

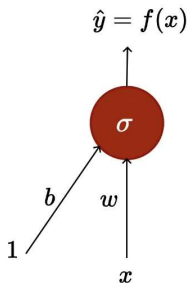
Parameters : w

Learning Algorithm: Gradient descent

Objective: $L(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

The learning algorithm should aim to find a w which minimizes the above function

Learning Parameters: (Infeasible) guess work



$$f(x) = \frac{1}{1+e^{-(wx+b)}}$$

This is the supervised learning setup we'll follow for learning parameters.

σ stands for sigmoid function (logistic function in this case)

There is only 1 input

b is bias (w_0)

Input: $\{x_i, y_i\}_{i=1}^{N}$ → N pairs of (x, y)

Objective: Find w & b such that

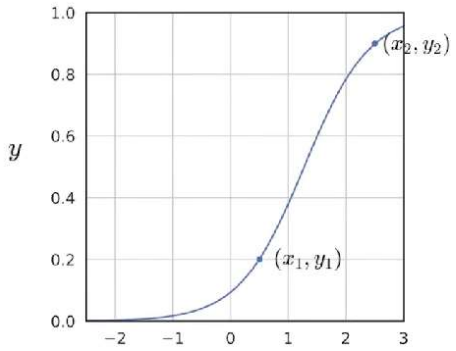
$$\underset{(w,b \in \mathbb{R})}{\text{minimize}} \; L(w,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(x_i)\right)^2$$

What does it mean to train the network?

Suppose we train with $(x,y) = (0.5, 0.2)$ & $(2.5, 0.9)$

At the end we are expected to find w^*, b^* such that $f(0.5) \to 0.2$ & $f(2.5) \to 0.9$

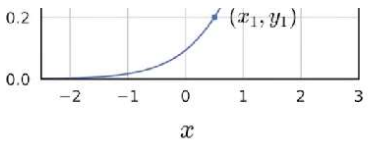
$$f(x) = \frac{1}{1+e^{-(wx+b)}}$$



Loss: $L(w,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(x_i)\right)^2$

$$= \frac{1}{2}\left[\{0.2 - f(0.5)\}^2 + \{0.9 - f(2.5)\}^2\right]$$

For different values of w & b,



w	b	Loss
0.50	0.00	0.0730
-0.10	0.00	0.14
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

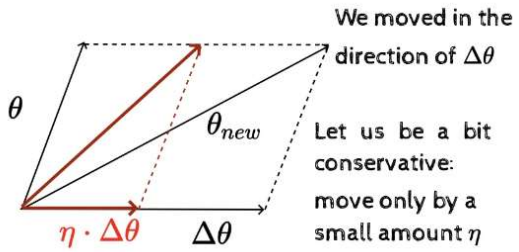
Taylor Series

Goal: Finding minimum error w/o brute force

Parameters: $\theta = [w, b]$

$\Delta\theta = [\Delta w, \Delta b]$

Change in Parameters w, b



$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$

what is the right $\Delta\theta$?

Answer comes from Taylor Series

Taylor series is a way of approximating any continuous differentiable function $L(w)$ using polynomials of degree n. The higher the degree, the better the approximation.

$$L(w) = L(w_0) + \frac{L'(w_0)}{1!}(w - w_0) + \frac{L''(w_0)}{2!}(w - w_0)^2 + \frac{L'''(w_0)}{3!}(w - w_0)^3 + \cdots$$

Linear approximation at n=1

$$L(w) = L(w_0) + \frac{L'(w_0)}{1!}(w - w_0)$$

Linear approximation at $n=2$

$$L(\omega) = L(\omega_0) + \frac{L'(\omega_0)}{1!}(\omega-\omega_0) + \frac{L''(\omega_0)}{2!}(\omega-\omega_0)^2$$

Gradient Descent

For ease of notation, let $\Delta\theta = u$, then from Taylor Series we have the following:

Hessian matrix

$$L(\theta + \eta u) = L(\theta) + \eta * u^T \nabla_\theta L(\theta) + \frac{\eta^2}{2!} * u^T \nabla_\theta^2 L(\theta) u + \cdots$$

Gradient → collection of partial derivatives

Gradient of Gradient is Hessian Matrix

eg :- $\theta = w^2 + b^2$

$$\begin{bmatrix} \frac{\partial \theta}{\partial w} \\ \frac{\partial \theta}{\partial b} \end{bmatrix} = \begin{bmatrix} 2w \\ 2b \end{bmatrix} \rightarrow \nabla_\theta L(\theta)$$

$$L(\theta + \eta u) = L(\theta) + \eta * u^T \nabla_\theta L(\theta)$$

[η is typically small, so $\eta^2, \eta^3 \rightarrow 0$]

The move ηu would be favourable only if,

$L(\theta) > L(\theta + \eta u)$

or, $L(\theta + \eta u) - L(\theta) < 0$ [new loss is smaller]

implying: $u^T \nabla_\theta L(\theta) < 0$

what is it's range?

Let β be the angle b/w u^T & $\nabla_\theta L(\theta)$,

we know: $-1 \leq \cos(\beta) = \frac{u^T \cdot \nabla_\theta L(\theta)}{\|u\| \|\nabla_\theta L(\theta)\|} \leq 1$

We know: $-1 \leq \cos(\beta) = \frac{u^T \cdot \nabla_\theta L(\theta)}{\|u^T\| * \|\nabla_\theta L(\theta)\|} \leq 1$ $\rightarrow k$ (let)

$$-k \leq k \cos\beta = u^T \cdot \nabla_\theta L(\theta) \leq k$$

Thus $L(\theta + \eta u) - L(\theta) = u^T \nabla_\theta L(\theta) = k * \cos(\beta)$
is the most negative when $\cos(\beta) = -1$
$\therefore \beta = 180°$

In other words, move to the opposite of the gradient

Parameter Updation Rule

$$w_{t+1} = w_t - \eta \nabla w_t$$
$$b_{t+1} = b_t - \eta \nabla b_t$$

$$\nabla w_t = \frac{\partial L(w,b)}{\partial L(w)} \text{ at } w = w_t, b = b_t$$

$$\nabla b_t = \frac{\partial L(w,b)}{\partial L(b)} \text{ at } w = w_t, b = b_t$$

Gradient Descent: Weight Update Rule

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

$$\nabla w_t = \frac{\partial L(w,b)}{\partial L(w)} \text{ at } w = w_t,\ b = b_t$$

$$\nabla b_t = \frac{\partial L(w,b)}{\partial L(b)} \text{ at } w = w_t,\ b = b_t$$

Algorithm

$t \leftarrow 0$
max_iterations $\leftarrow$ 1000
$w, b \leftarrow$ initialize randomly
while $t <$ max_iterations do
  $w_{t+1} \leftarrow w_t - \eta \nabla w_t$
  $b_{t+1} \leftarrow b_t - \eta \nabla w_t$
  $t \leftarrow t+1$
end

```
max_iterations = 1000
w = random()
b = random()
while max_iterations:
  w = w - eta*dw
  b = b - eta*db
  max_iterations -= 1
```

what exactly is ∇w_t? How to compute it?

For just one point,

$$\nabla w = \frac{\partial}{\partial w}\left[\frac{1}{2} * (f(x) - y)^2\right]$$

$$= \frac{1}{2}\left[2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)\right]$$

$$f(x) = \frac{1}{1 + \exp(-wx+b)}$$

$$\frac{\partial}{\partial w}(f(x)-y)]$$

$$= (f(x)-y) * \frac{\partial}{\partial w} f(x)$$

$$= (f(x)-y) * \frac{\partial}{\partial w}\left[\frac{1}{1+e^{-(wx+b)}}\right] \quad \text{— (1)}$$

y is a constant & it's partial derivative will be 0

$$\frac{\partial}{\partial w}\left[\frac{1}{1+e^{-(wx+b)}}\right] = \frac{1}{1+e^{-(wx+b)}} \frac{\partial}{\partial w}\left(e^{-(wx+b)}\right)$$

$$= \frac{-1}{1+e^{-(wx+b)}} * e^{-(wx+b)} * \frac{\partial}{\partial w}\left(-(wx+b)\right)$$

$$= \frac{-1}{1+e^{-(wx+b)}} * \frac{e^{-(wx+b)}}{1+e^{-(wx+b)}} * -x$$

$$= \frac{1}{1+e^{-(wx+b)}} * \frac{e^{-(wx+b)}}{1+e^{-(wx+b)}} * x$$

$$= f(x)(1-f(x))\,x$$

Substituting this in eqn (1) we get,

$$\nabla w = (f(x)-y) * f(x) * (1-f(x)) * x$$

for one point.

For n points,

$$\nabla w = \sum_{i=1}^{n} \left[(f(x_i)-y_i) * f(x_i) * (1-f(x_i)) * x_i\right]$$

Similarly,

Similarly,

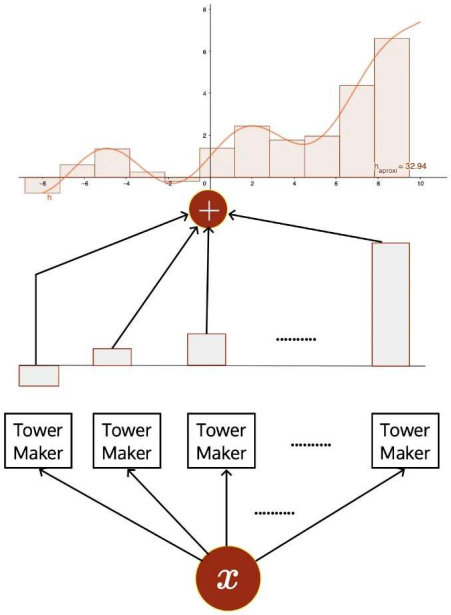
$$\nabla b = \sum_{i=1}^{n} \left[(f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) \right]$$

```
import numpy as np
X = [0.5,2.5]
Y = [0.2,0.9]

def f(x,w,b):
  return 1/(1+np.exp(-(w*x+b)))

def error(w,b):
  err = 0.0
  for x,y in zip(X,Y):
    fx = f(x,w,b)
    err += (fx-y)**2
  return 0.5*err

def grad_b(x,w,b,y):
  fx = f(x,w,b)
  return (fx-y)*fx*(1-fx)

def grad_w(x,w,b,y):
  fx = f(x,w,b)
  return (fx-y)*fx*(1-fx)*x

def do_gradient_descent():

  w,b,eta,max_epochs = -2,-2,1.0,1000

  for i in range(max_epochs):
    dw,db = 0,0
    for x,y in zip(X,Y):
      dw += grad_w(x,w,b,y)
      db += grad_b(x,w,b,y)

    w = w - eta*dw
    b = b - eta*db
```

Representation of Power of a Multilayer Network of Sigmoid Neurons

A multilayer network of neurons with a single hidden layer can be used to approximate any continuous function to any desired precision.

In other words, there is a guarantee that for any function $f(x): \mathbb{R}^n \to \mathbb{R}^m$, we can always find a neural network (with 1 hidden layer containing enough neurons) whose output $g(x)$ satisfies $|g(x) - f(x)| < \varepsilon$

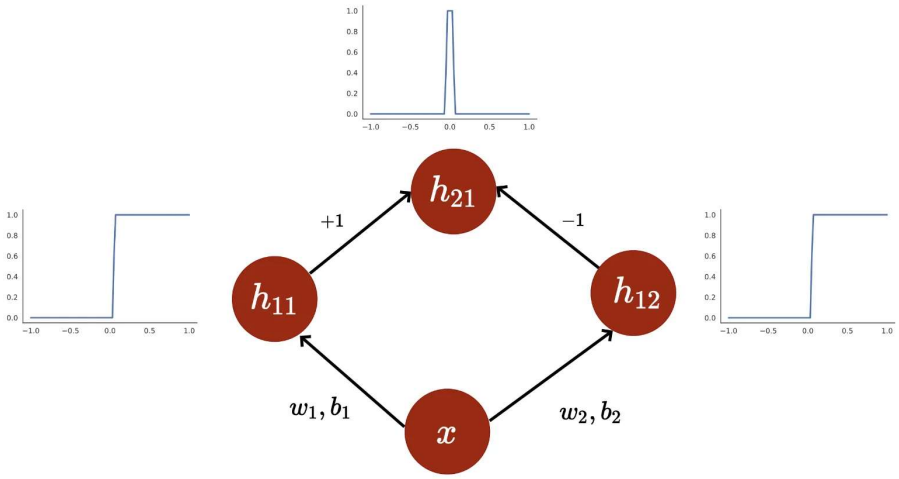


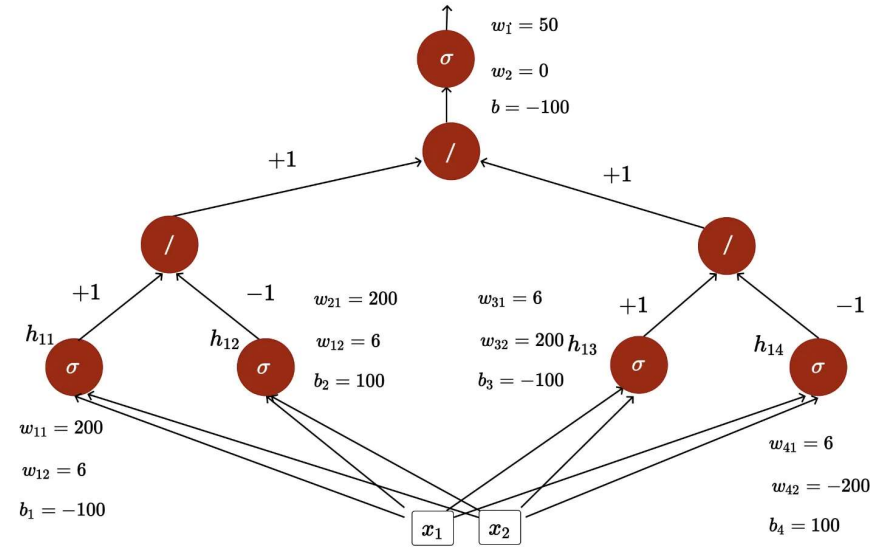
Any function can be **represented** by summing **shifted** and **scaled** versions of tower functions

Suppose there is a black box which takes the original input (x) and constructs these tower functions

We can then have a simple network which can just add them up to approximate the function

Our job now is to figure out what is inside this blackbox


h_{21}
$+1$
-1
h_{11}
h_{12}
w_1, b_1
x
w_2, b_2


$w_1 = 50$
$w_2 = 0$
$b = -100$
σ
/
$+1$
$+1$
$+1$
-1
$+1$
-1
h_{11}
h_{12}
h_{13}
h_{14}
$w_{21} = 200$
$w_{12} = 6$
$b_2 = 100$
$w_{31} = 6$
$w_{32} = 200$
$b_3 = -100$
$w_{11} = 200$
$w_{12} = 6$
$b_1 = -100$
$w_{41} = 6$
$w_{42} = -200$
$b_4 = 100$
x_1
x_2