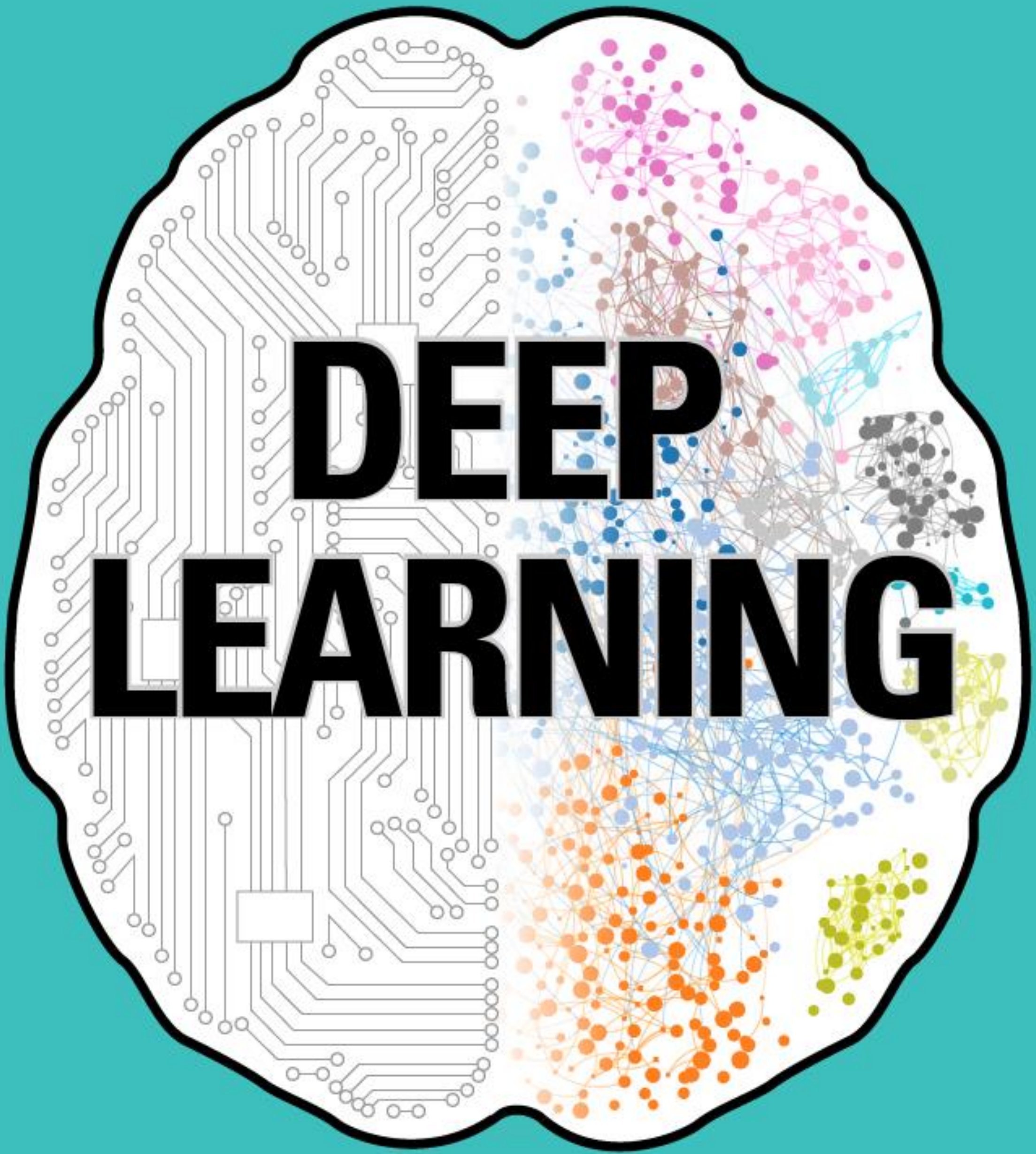


Neural Networks & Deep Learning

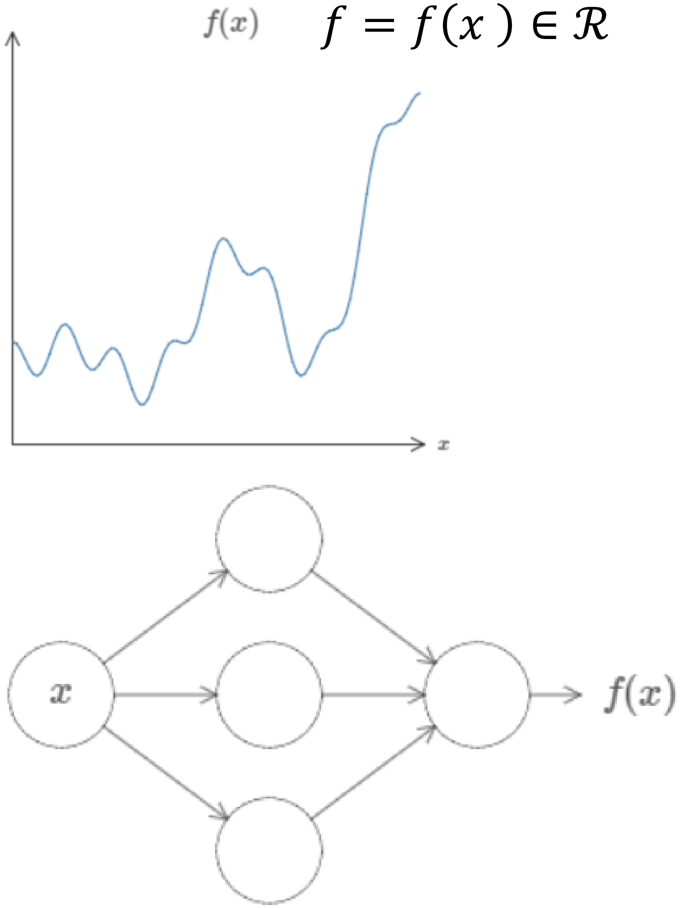
http://neuralnetworksanddeeplearning.com/chap4.html

The universality theorem

neural networks can compute any function at all.

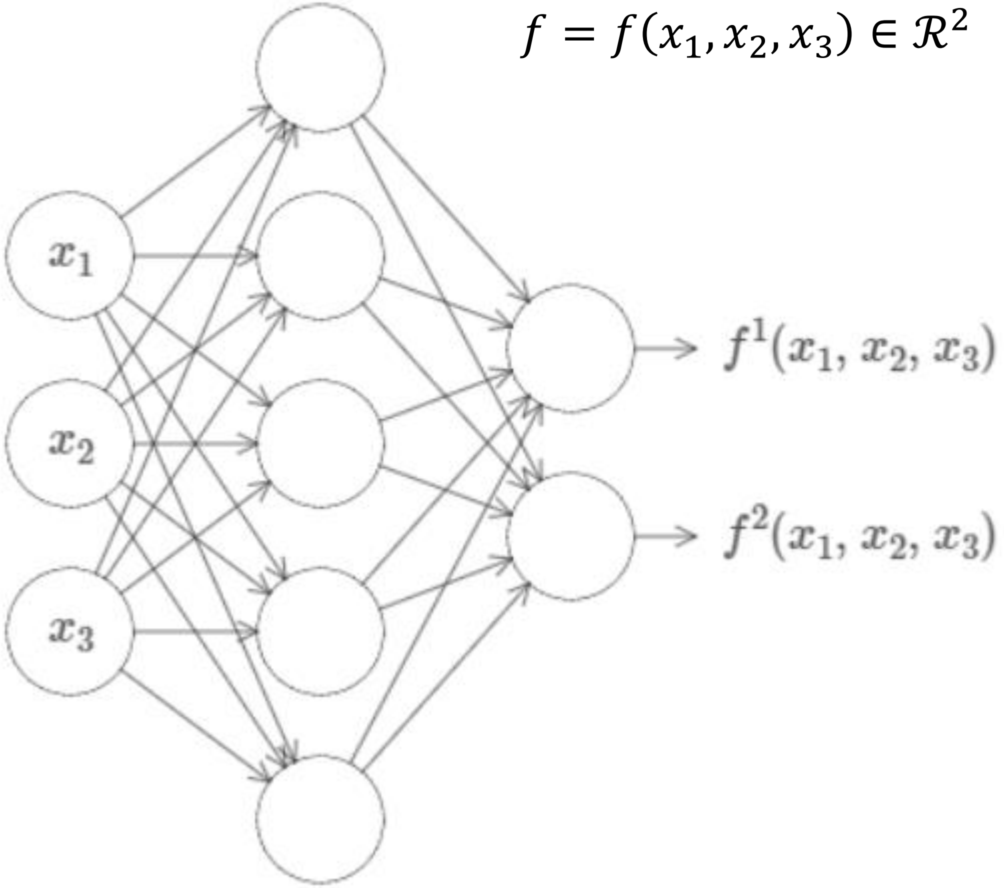
$f : \mathcal{R}^1 \rightarrow \mathcal{R}^1$

$f = f(x) \in \mathcal{R}$



$f : \mathcal{R}^3 \rightarrow \mathcal{R}^2$

$f = f(x_1, x_2, x_3) \in \mathcal{R}^2$



The universality theorem

Two caveats

1. approximation

First, this doesn't mean that a network can be used to *exactly* compute any function.
Rather, we can get an *approximation* that is as good as we want.
By increasing the number of hidden neurons we can improve the approximation.

2. continuous function

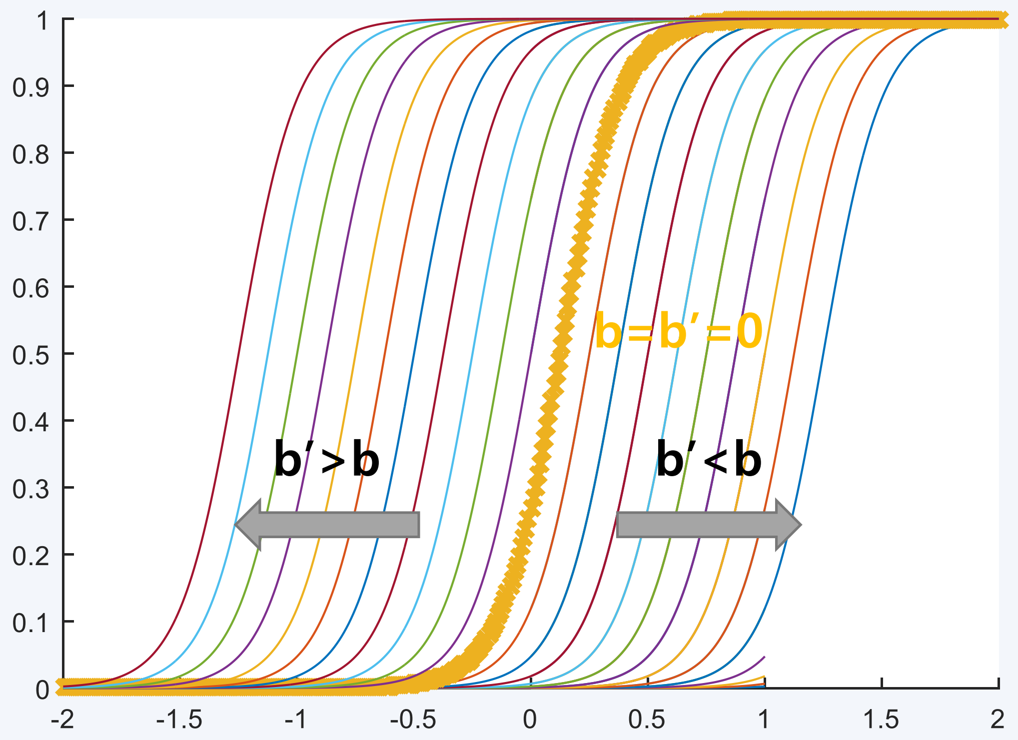
The second caveat is that the class of functions which can be approximated in the way described are the *continuous* functions. If a function is discontinuous, i.e., makes sudden, sharp jumps, then it won't in general be possible to approximate using a neural net.

Universality with one input and one output

Shape change by weight and bias

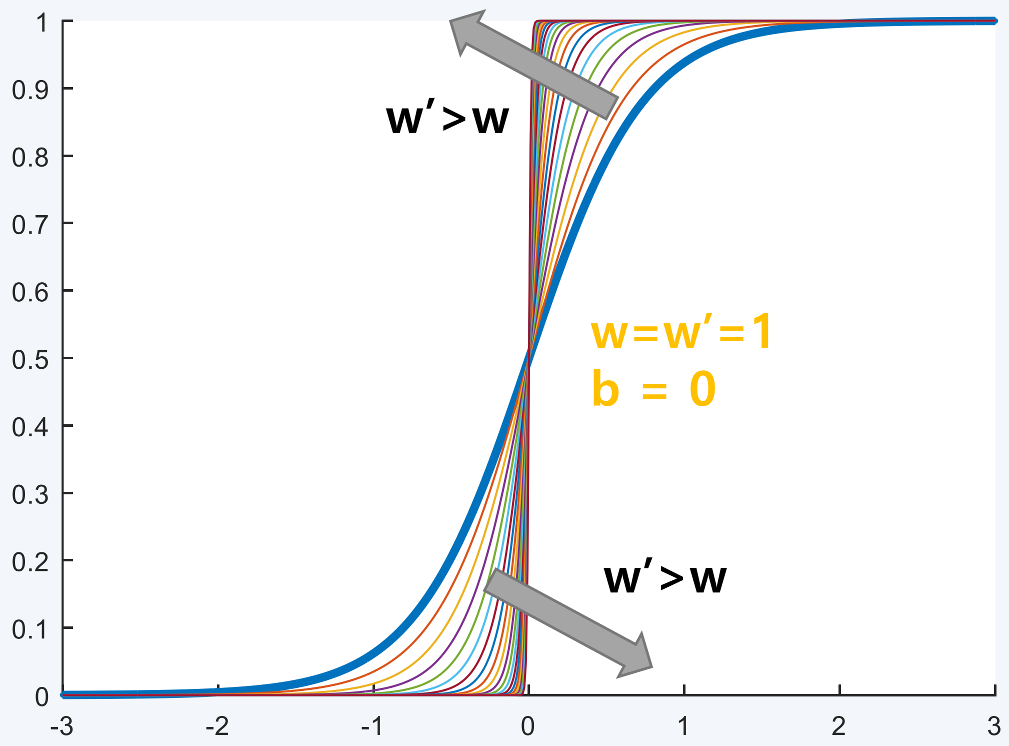
- As bias term increases, graph for output from hidden neuron goes left without shape change
- As bias term decreases, graph for output from hidden neuron goes right without shape change

$$\sigma(wx + b) = \sigma(z) \rightarrow \sigma(wx + b')$$



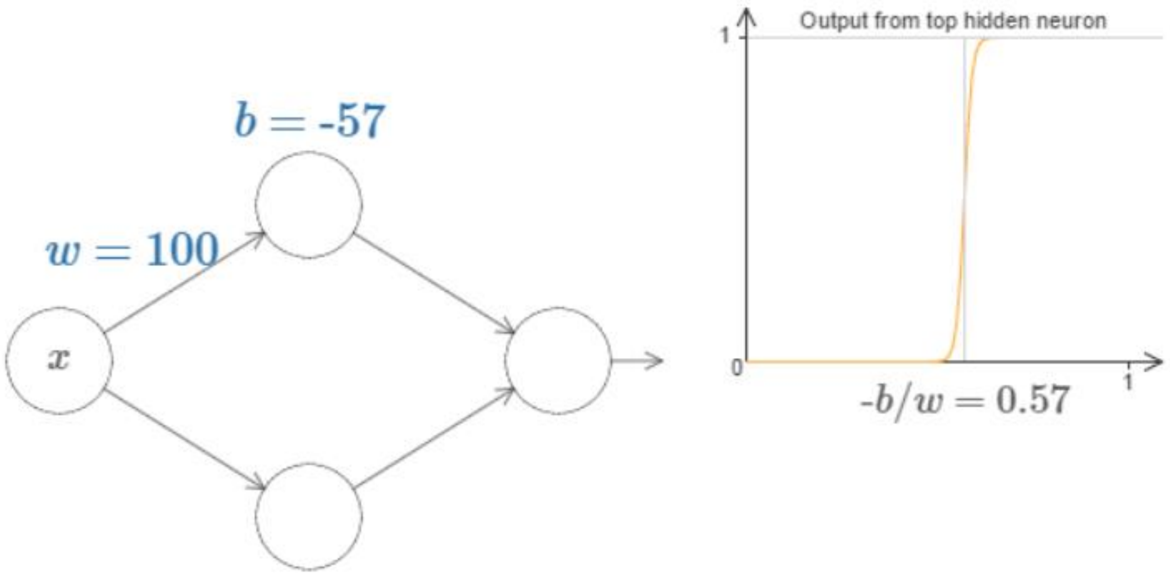
- As weight term increases, graph for output from hidden neuron changes its shape.
- The curve gets steeper, until eventually it begins to look like a step function

$$\sigma(wx + b) = \sigma(z) \rightarrow \sigma(w'x + b)$$



Universality with one input and one output

One hidden layer with two neurons



- It's actually quite a bit easier to work with step functions than general sigmoid functions.
- The reason is that in the output layer we add up contributions from all the hidden neurons.
- For this, we use 'w' with a very large – big enough that the step function is a very good approximation.
- If so, at what value of x does the step occur?

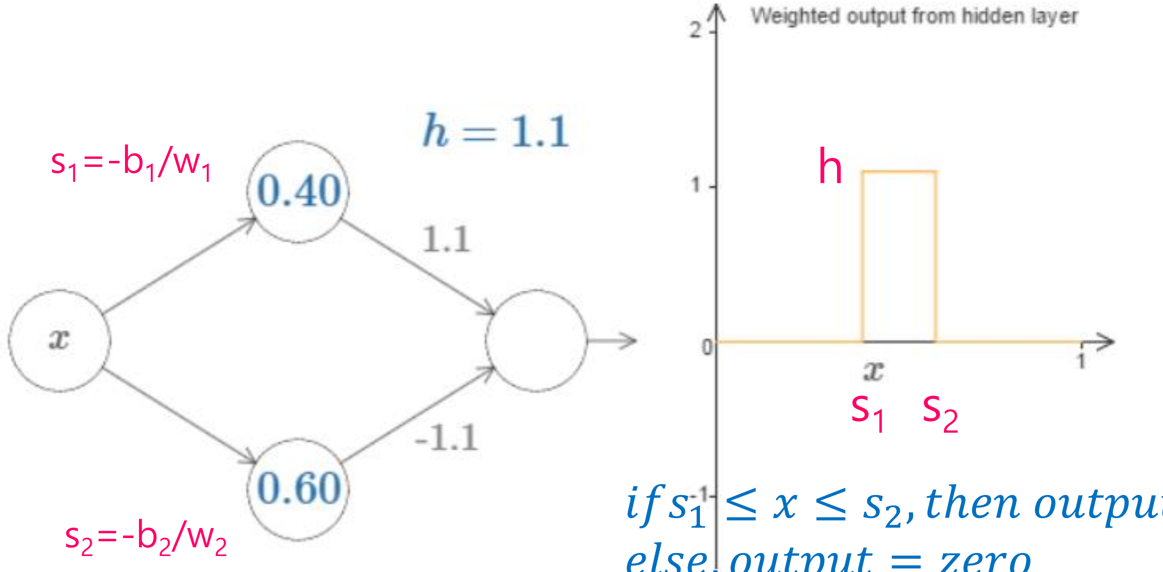
$$x = -b/w = s$$

Universality with one input and one output

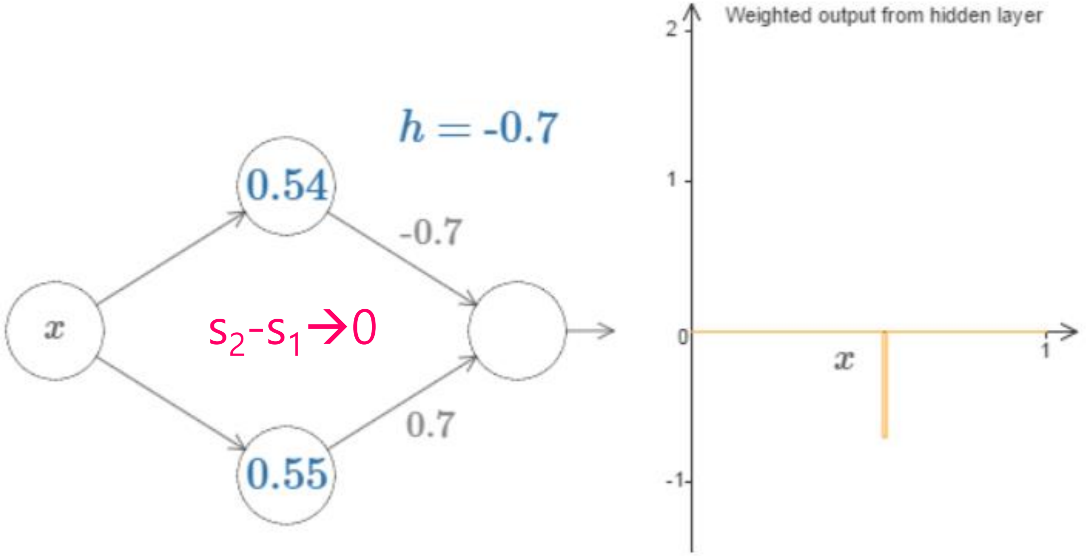
One hidden layer with two neurons

A pair of step-function like neurons with weights summing up to zero gives "bump" function

Bumb function	*Delta function*



$$if\, s_1 \le x \le s_2, then\ output = h,$$
$$else, output = zero$$



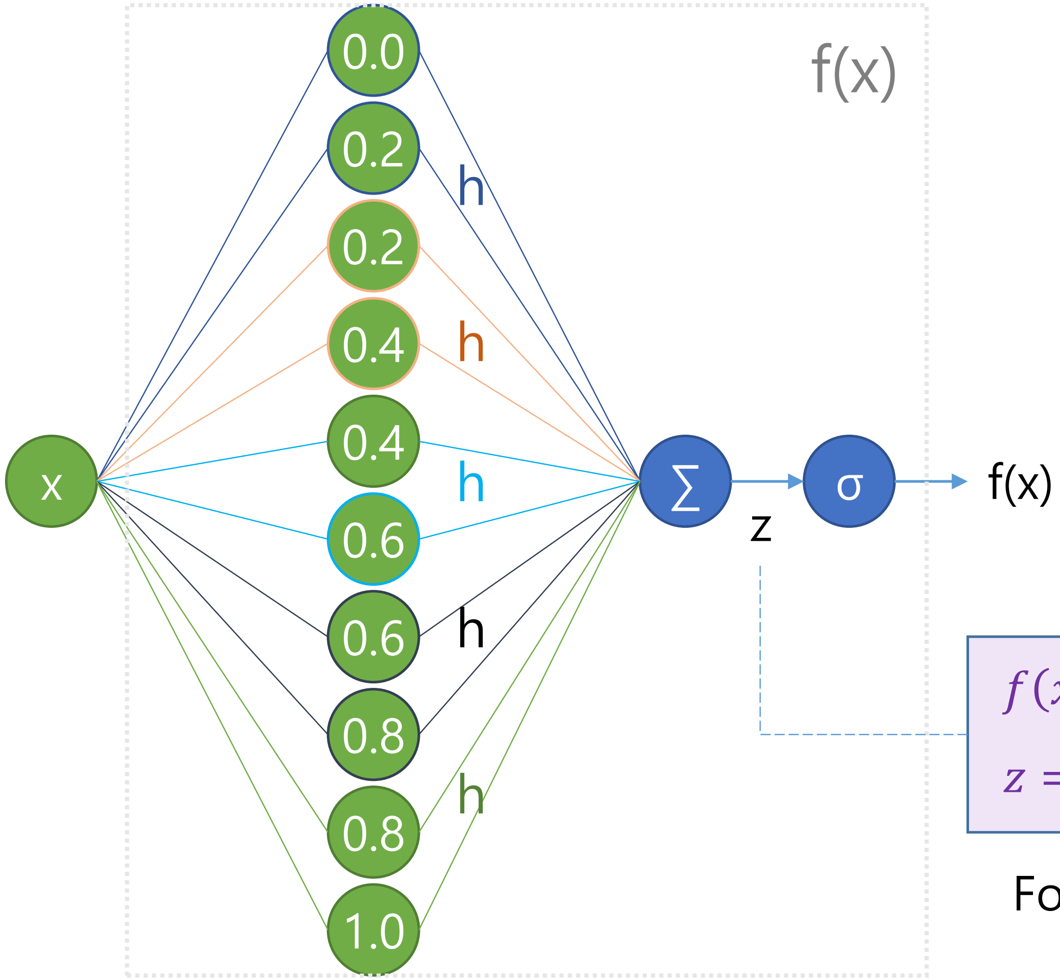
"bump" function is also called if-then-else statement

We can approximate any functions with may pairs of delta-function

Universality with one input and one output

Example

$$f(x) = 0.2 + 0.4x^2 + 0.3x\sin(15x) + 0.05\cos(50x)$$



- 5 pairs of neurons
- A combination of 5 step functions
- Only 'h' values are learned

$$f(x) = \sigma(z)$$

$$z = \sigma^{-1} \circ f(x)$$

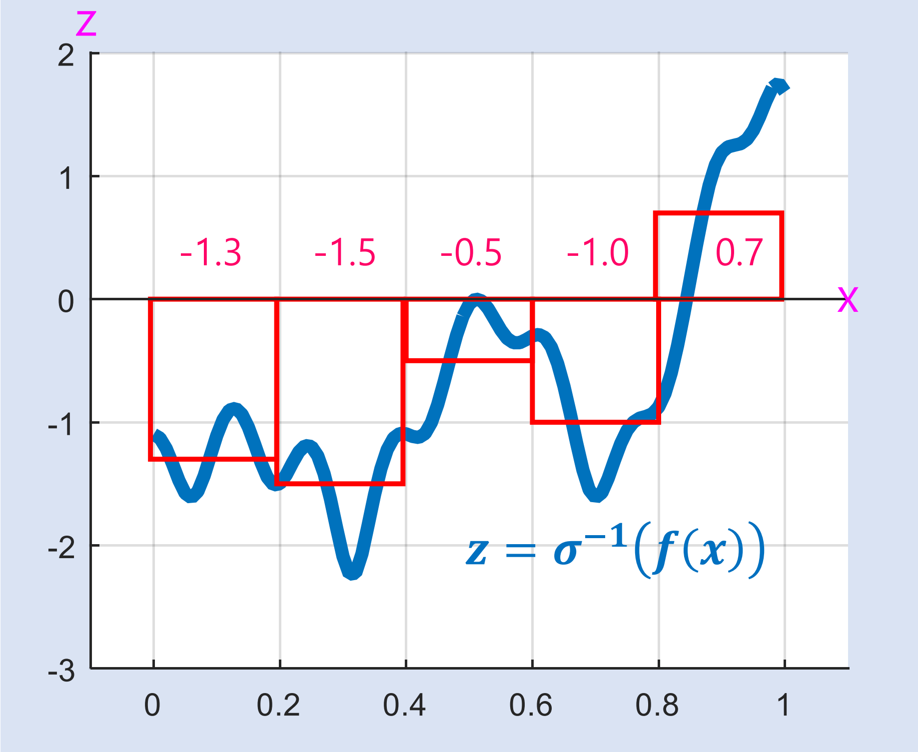
For the simplicity, we are focusing on 'z'

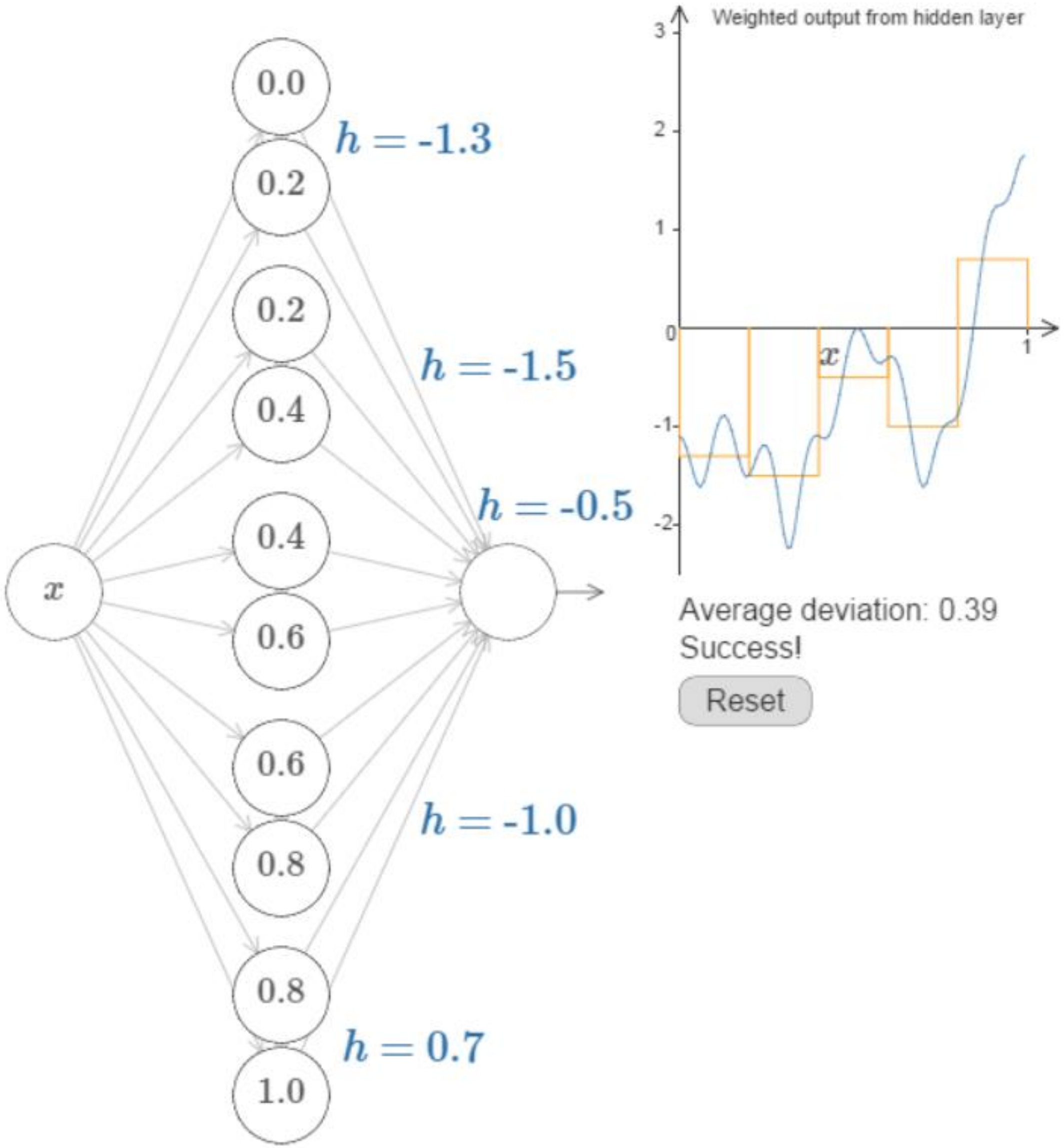
Universality with one input and one output

Example

$$f(x) = 0.2 + 0.4x^2 + 0.3x\sin(15x) + 0.05\cos(50x)$$

$$z = \sigma^{-1}(f(x)) = \sigma^{-1}(y) = ln\left(\frac{y}{1-y}\right)$$

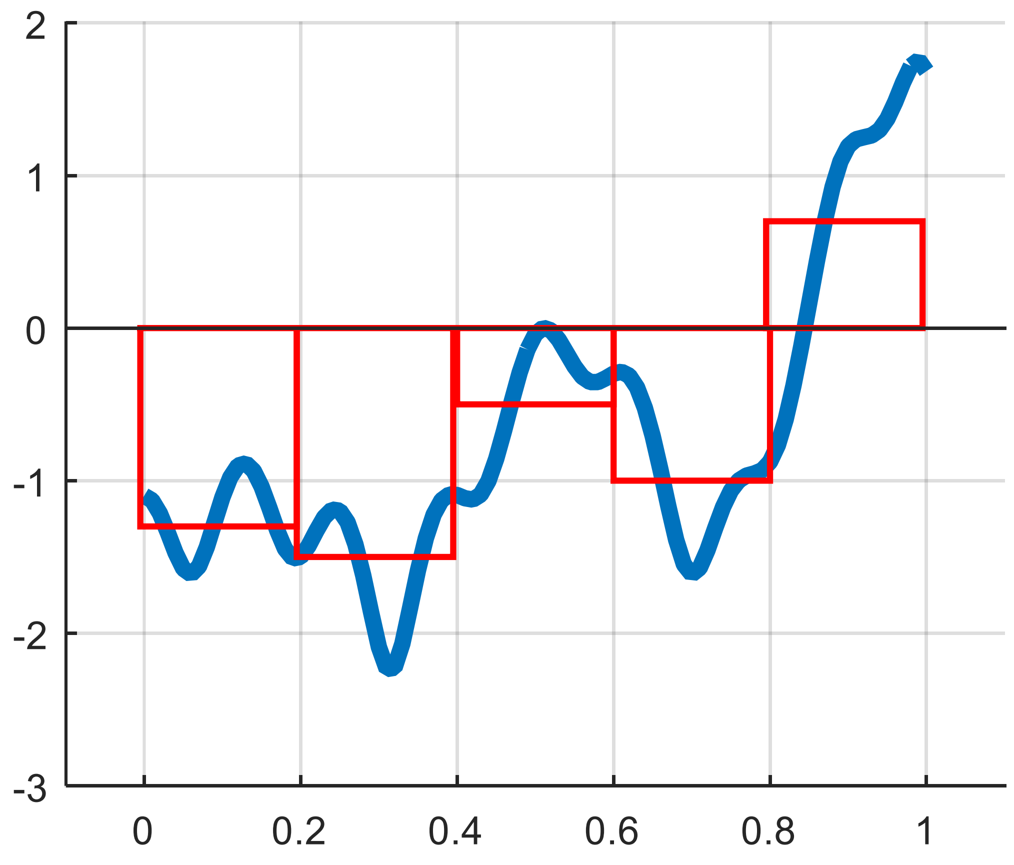




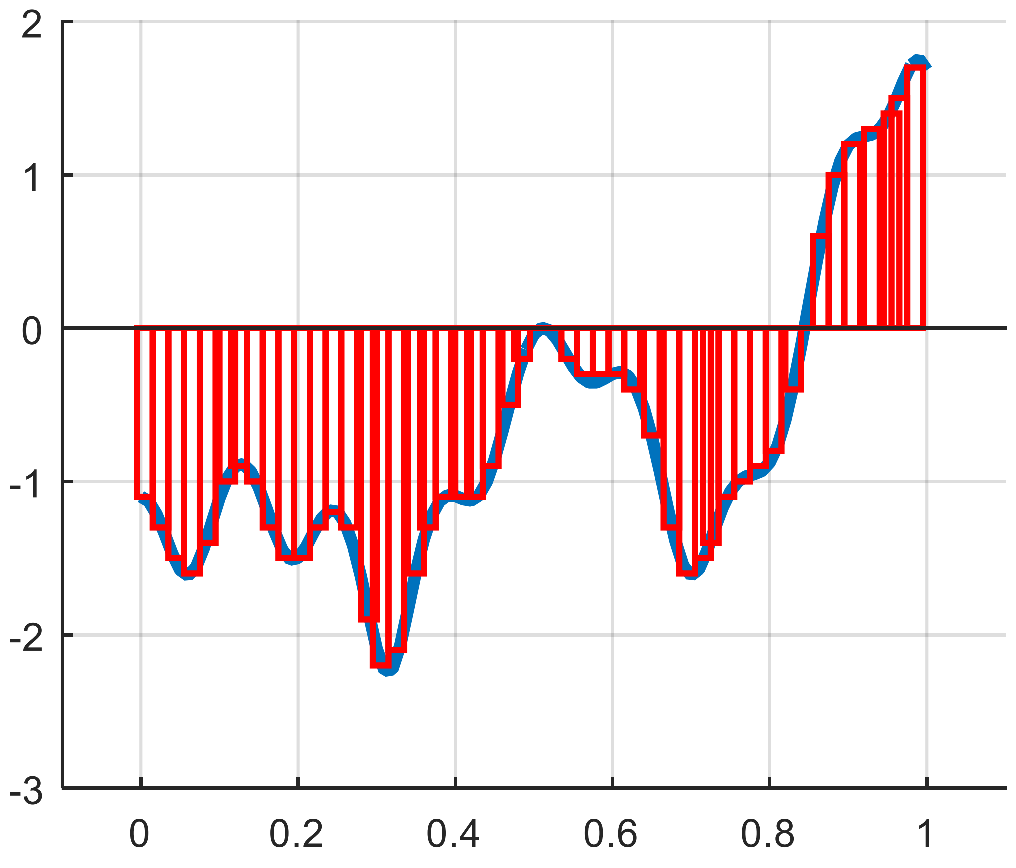
Universality with one input and one output

Example

5 pairs of neurons



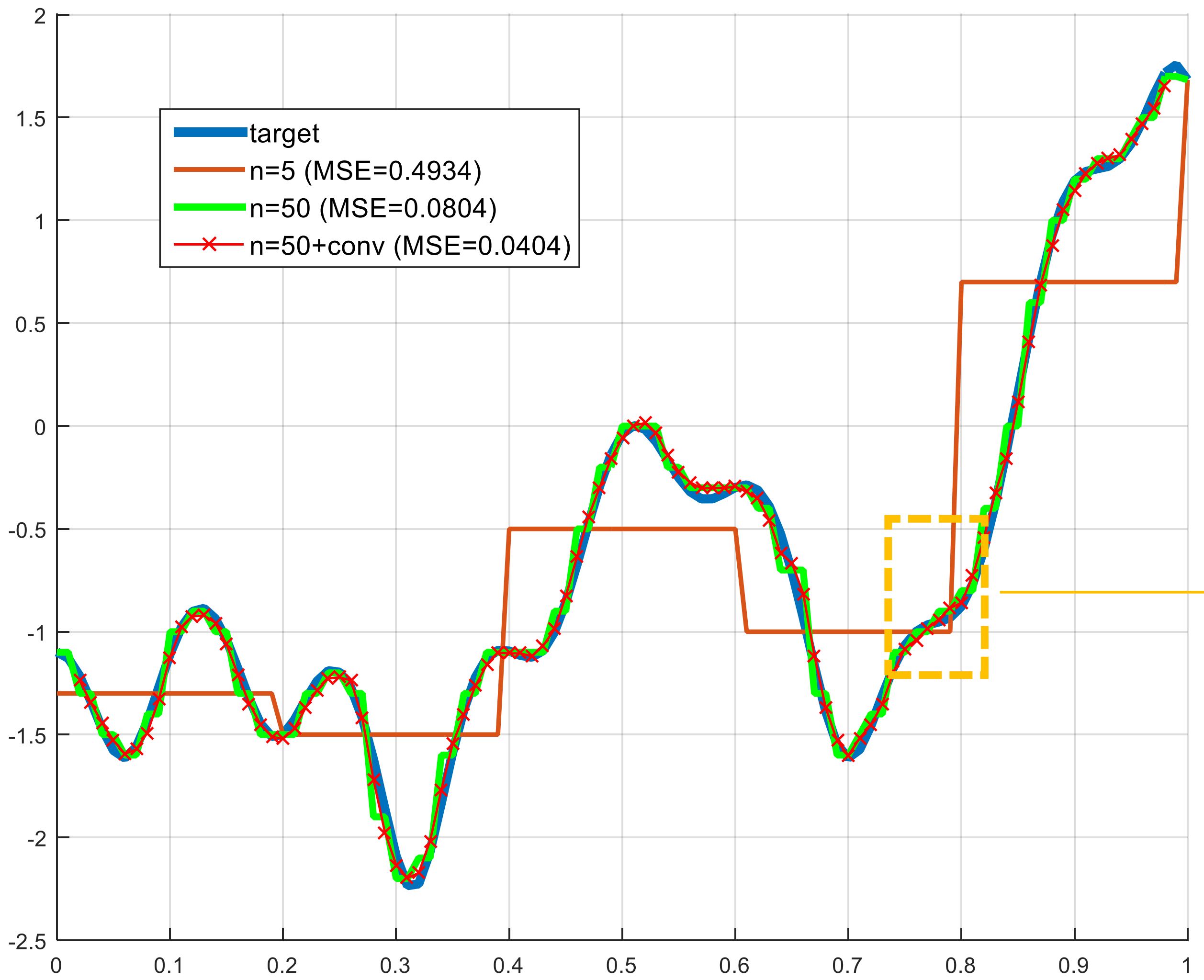
50 pairs of neurons

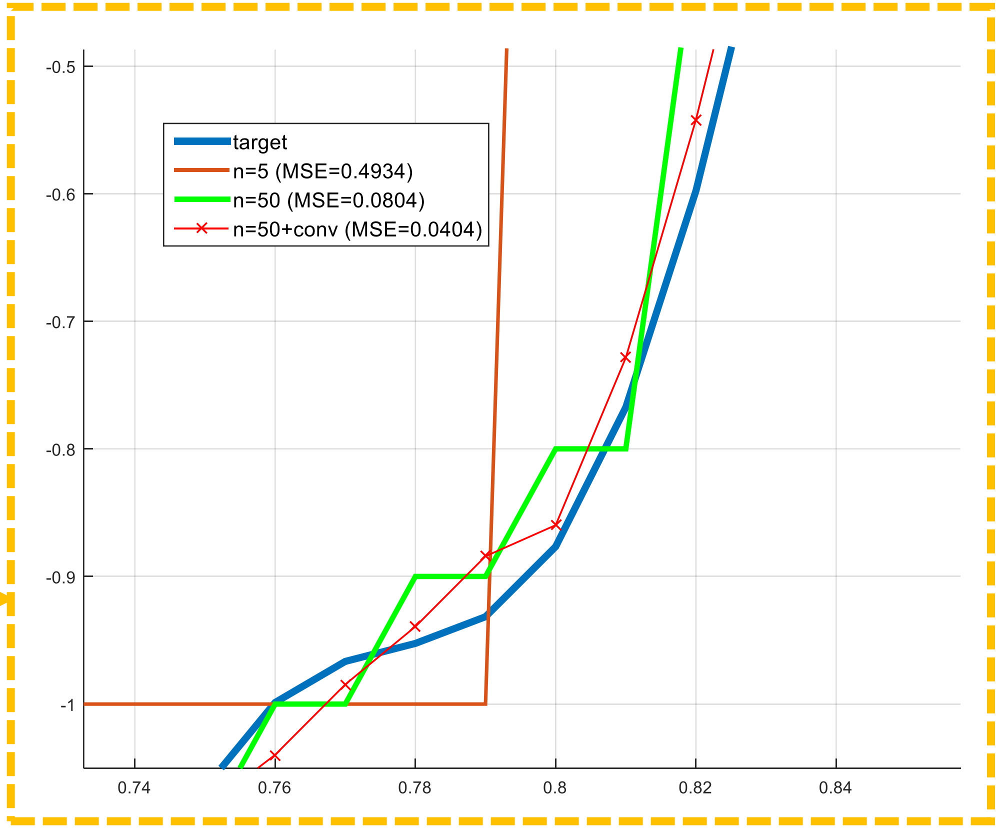


Universality with one input and one output

Example

How about adding a convolution layer??



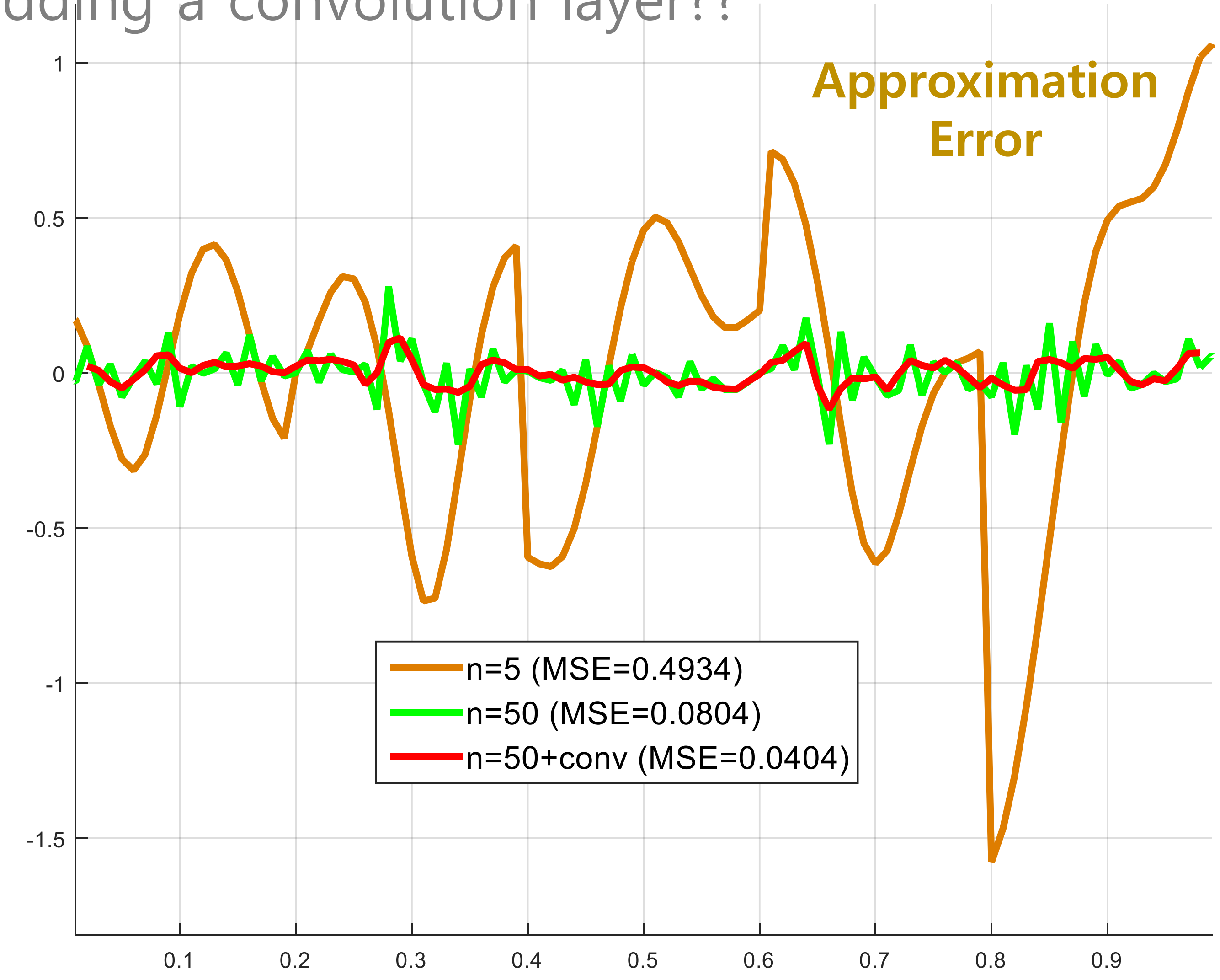


Magnified View

Universality with one input and one output

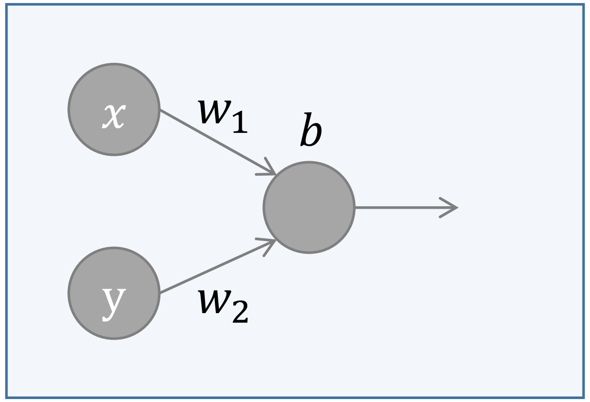
Example

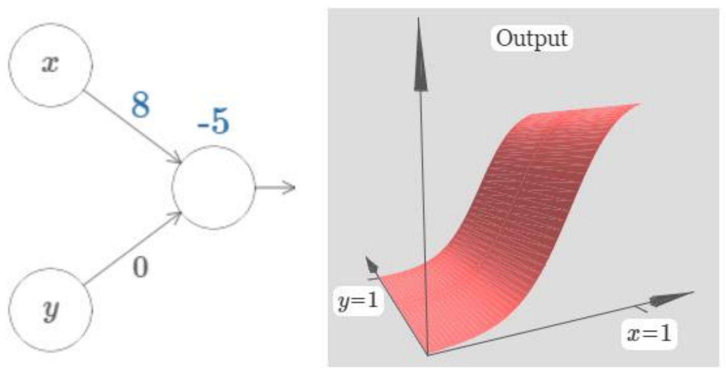
How about adding a convolution layer??



Many input variables

Two input variables



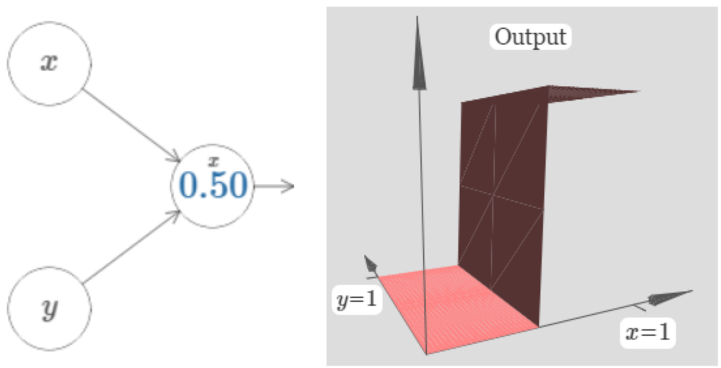


As bias term increases, graph for output from hidden neuron goes left on the x-axis without shape change
As bias term decreases, graph for output from hidden neuron goes right on the x-axis without shape change

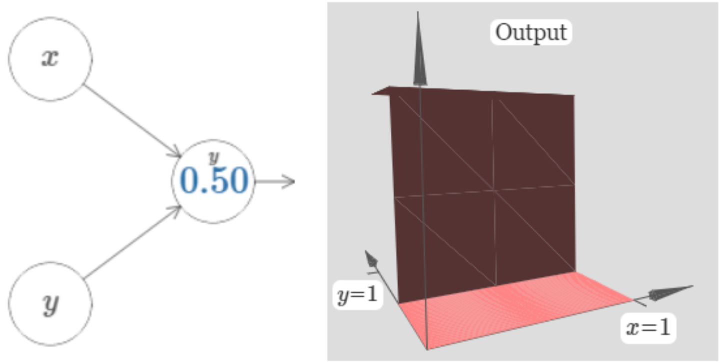
As weight term increases, graph for output from hidden neuron changes its shape.
(The curve gets steeper, until eventually it begins to look like a step function)

Same properties as in one input variable are observed!

For very large w_1 and zero w_2

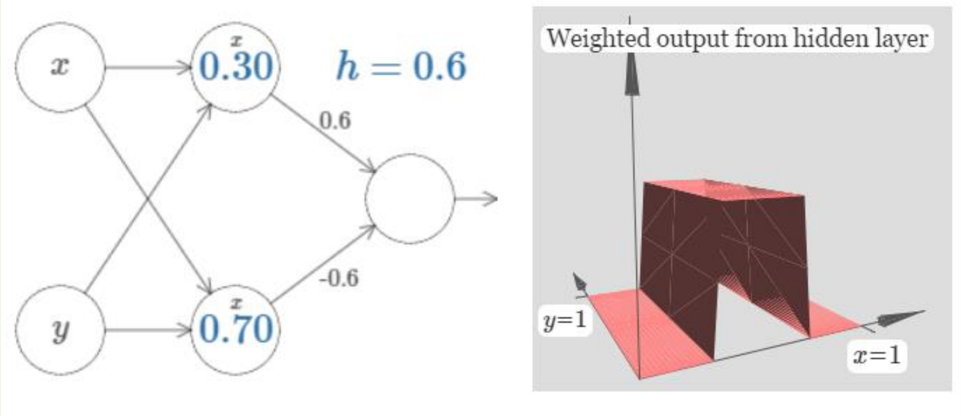


For very large w_2 and zero w_1

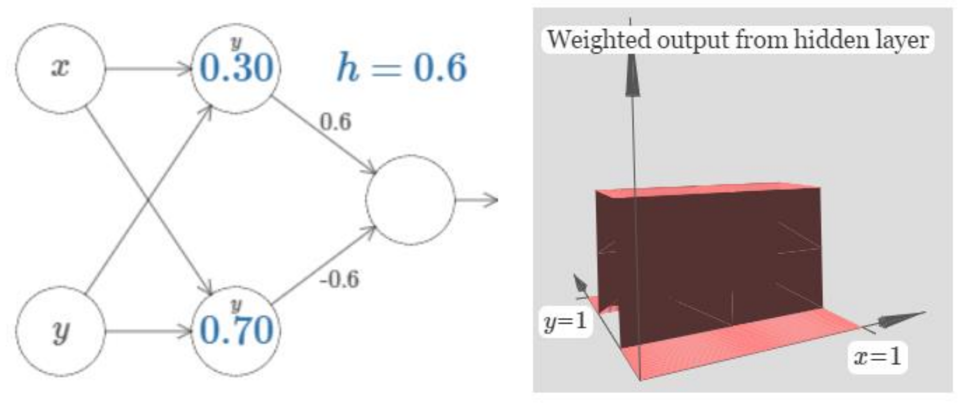


Many input variables

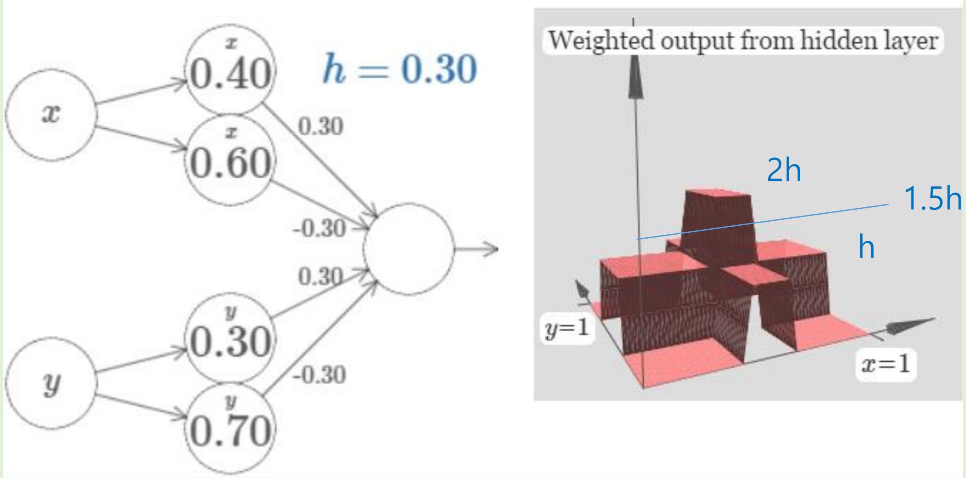
Two input variables



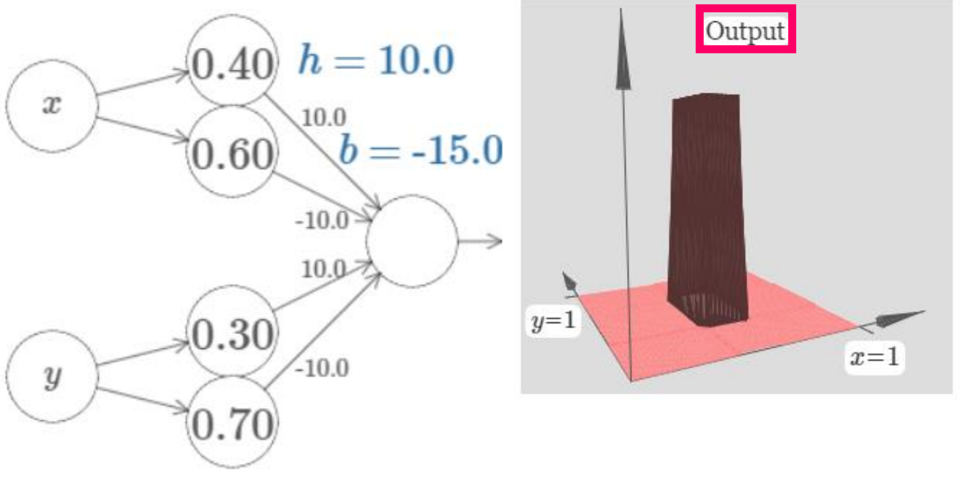
x-bump function



y-bump function



Adding x/y-bump functions



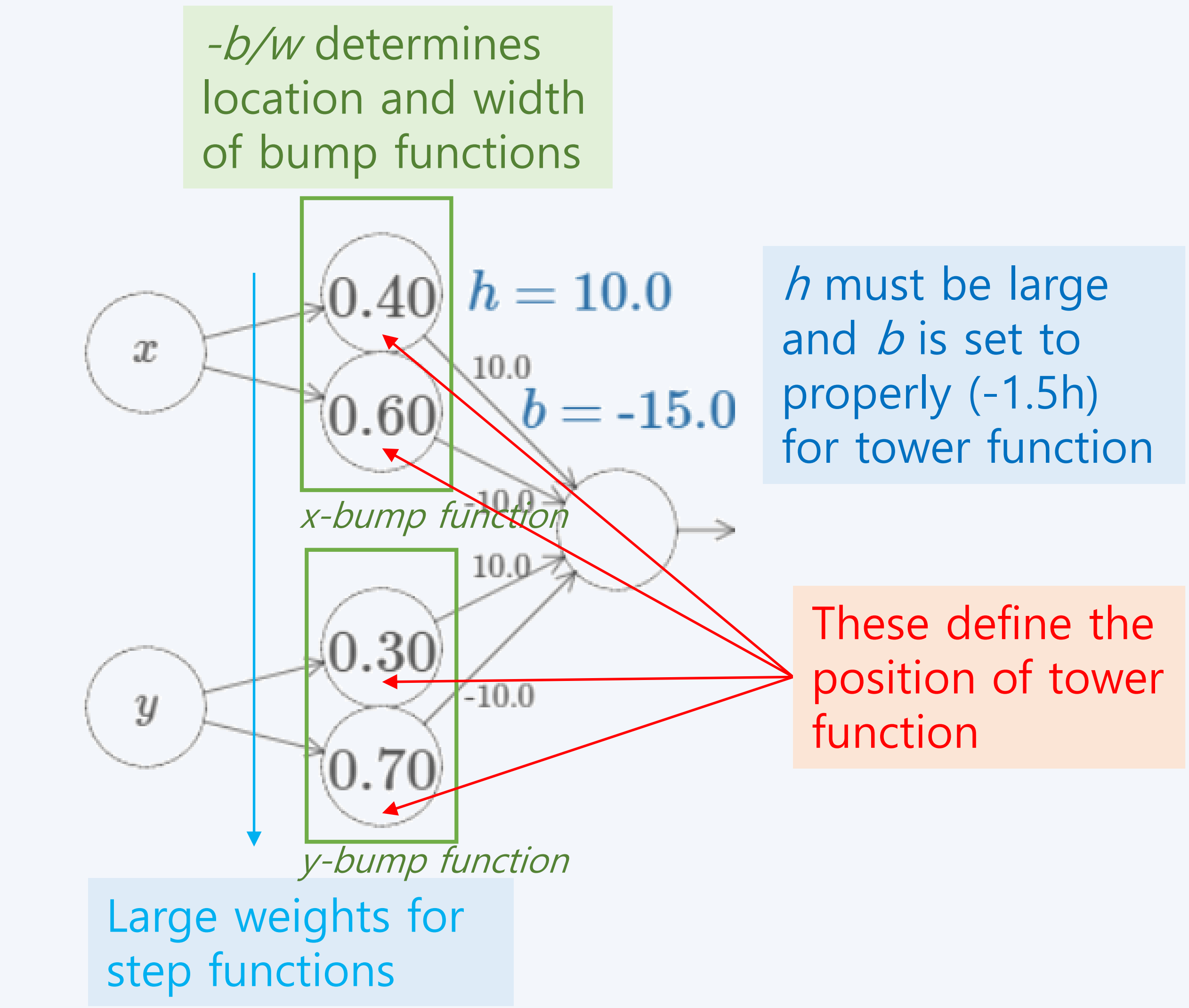
Tower function

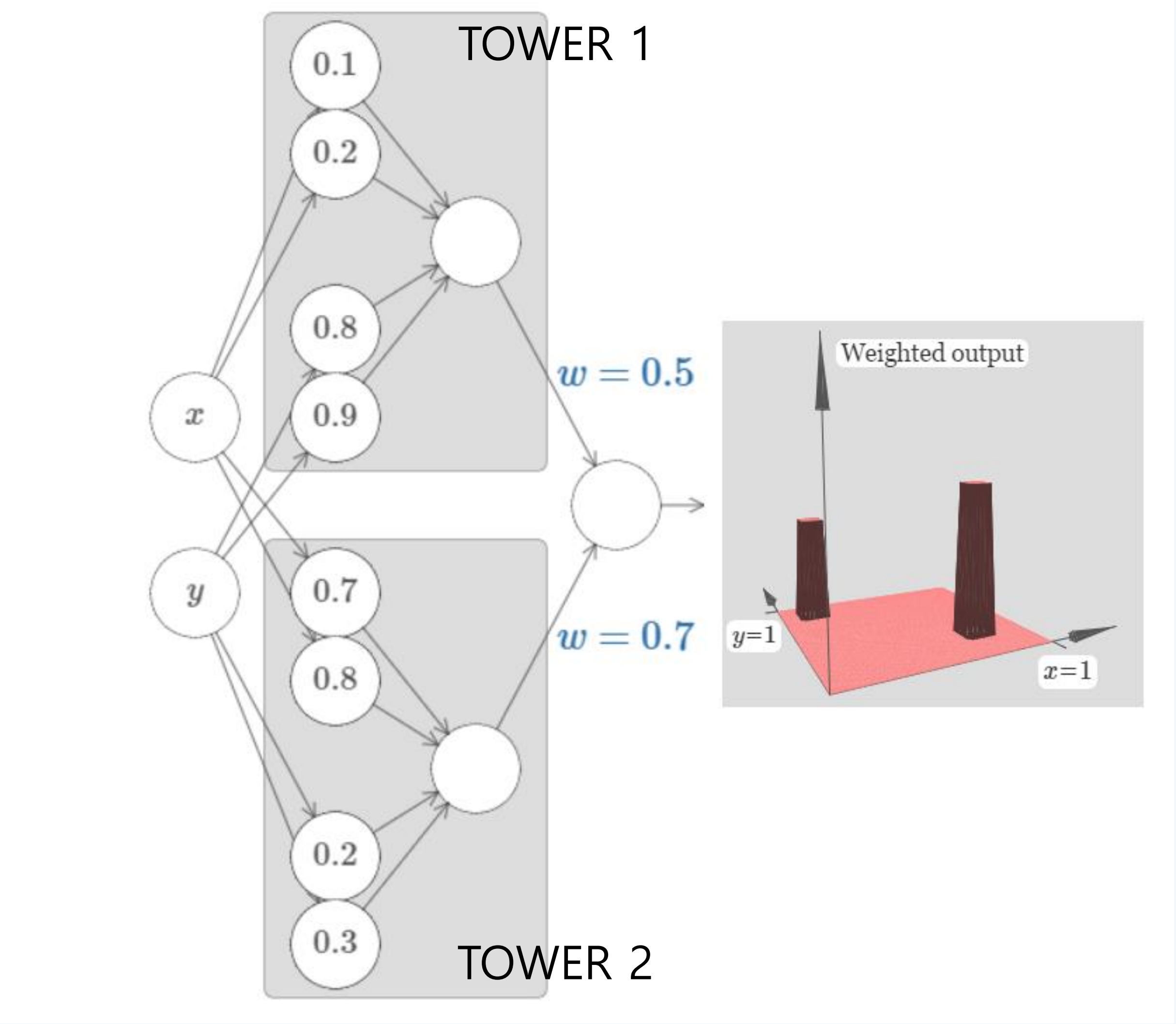
Output = σ(weighted output from hidden layer + b)
Tower function can be obtained from using threshold value of 1.5h. It means b should be -1.5h

(1) To get the output neuron to show the right kind of if-then-else behaviour, we need the input weights (all h or $-h$) to be large
(2) the value of b determines the scale of the if-then-else threshold.

Many input variables

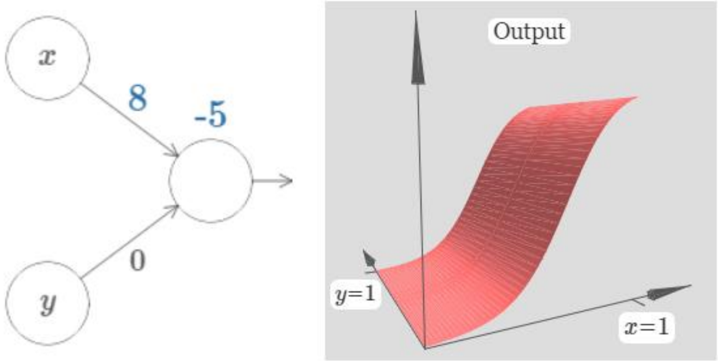
Two input variables



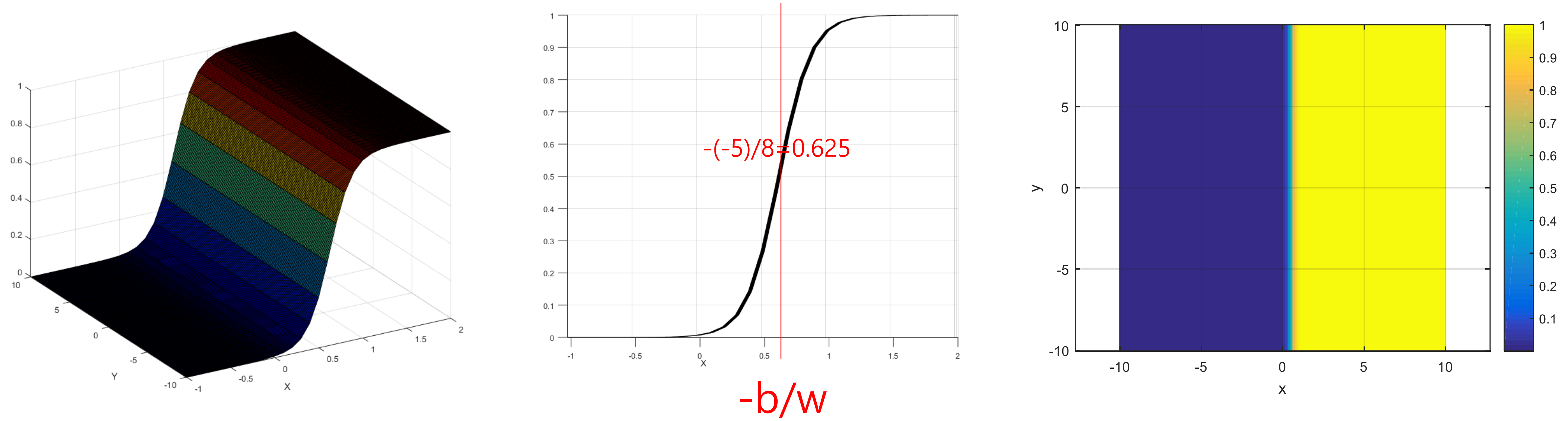


Many input variables

Two input variables



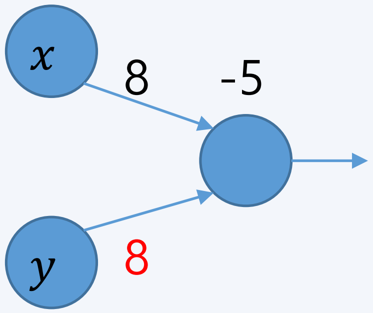
We've dealt with one weight zero cases.



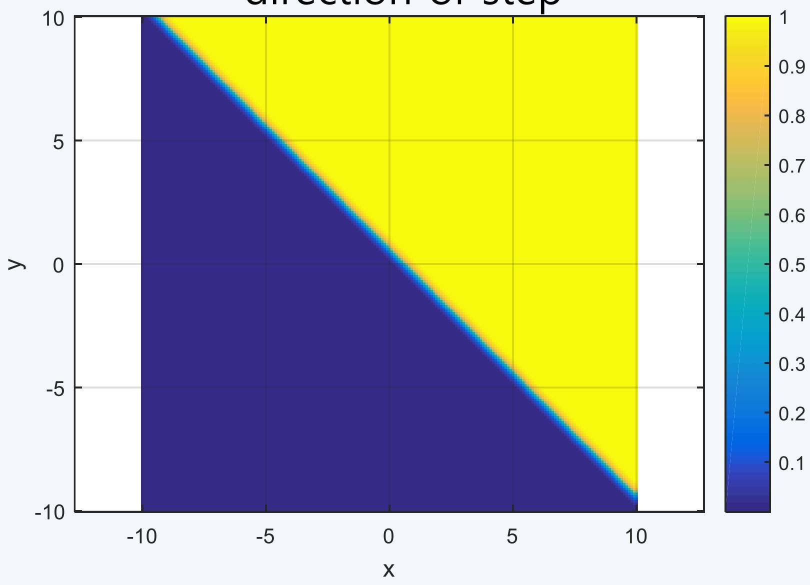
Many input variables

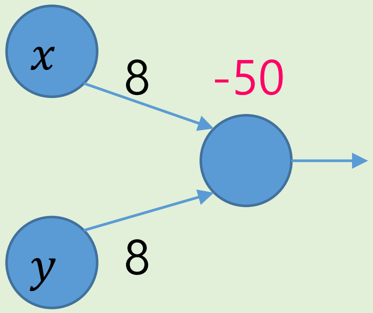
Two input variables

If neither weights are zero, it gives rotation effect to step function

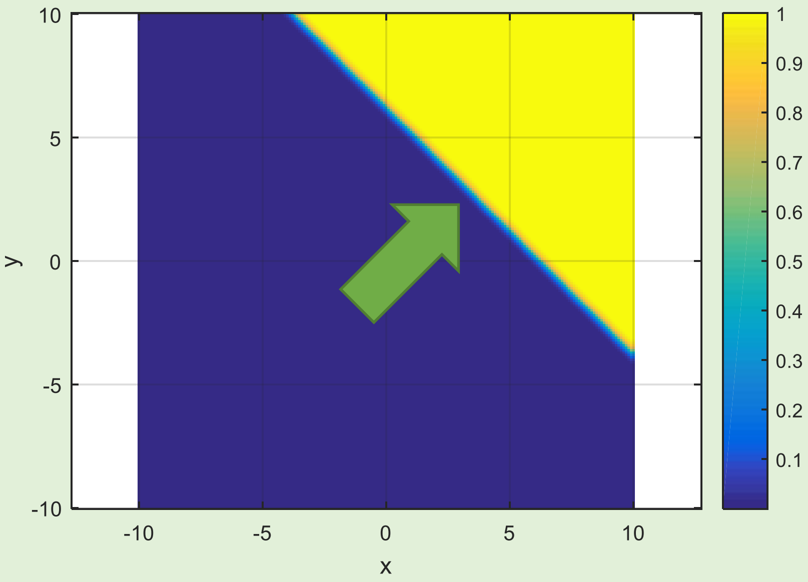


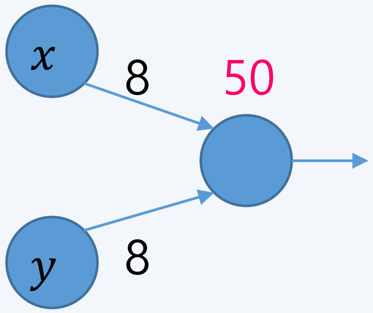
Rotation!! (ratio of weights defines the direction of step



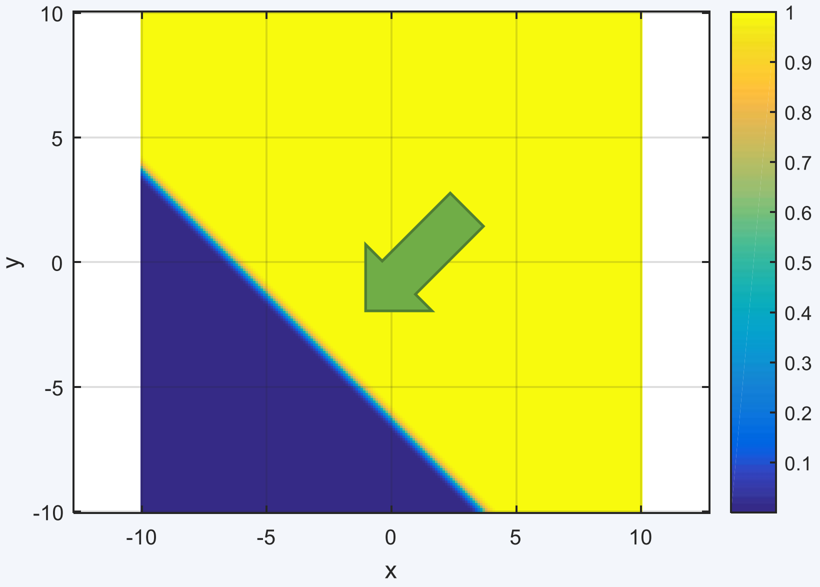


Shift up along the direction



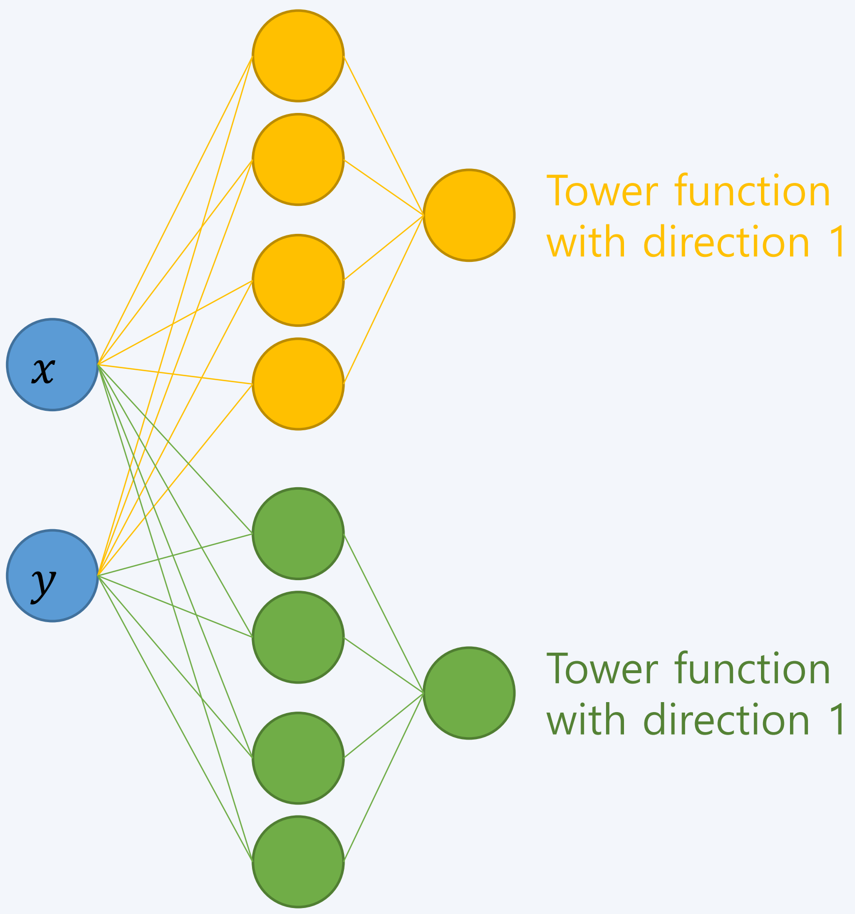


Shift down along the direction

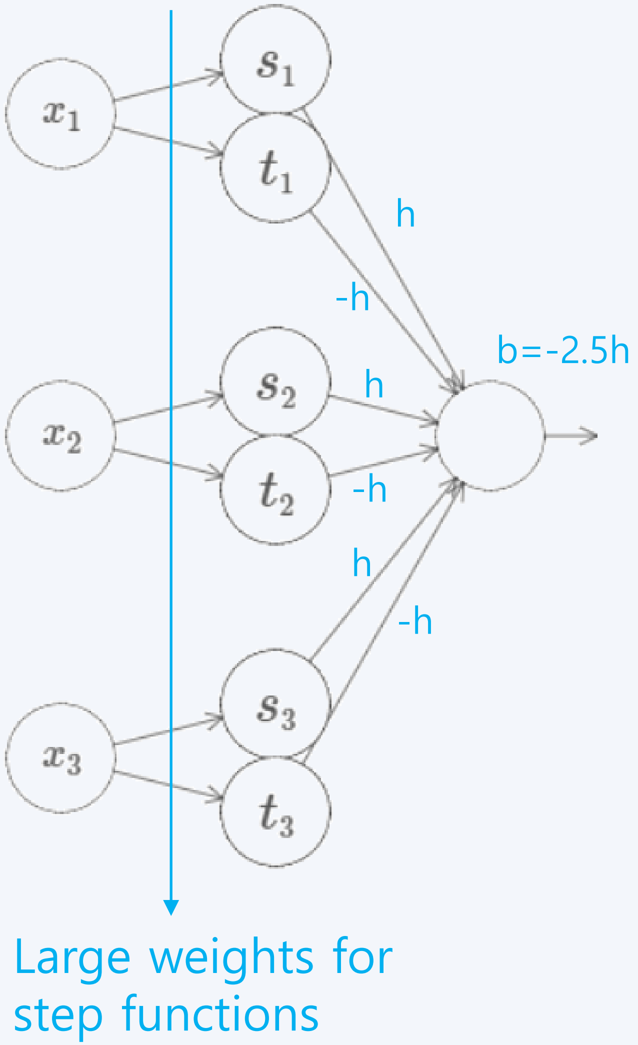


Many input variables

Two input variables



Three input variables



Extension beyond sigmoid neurons

Are rectified linear units universal for computation? YES!!