1DT109 - Accelerating systems with FPGAs

Expressive power of neural networks

Riccardo De Masellis

Uppsala University

December 6, 2021

On computing functions

A neural network with one hidden layer can approximate (to any desired precision) any continuous function with any number of inputs and outputs (Cybenko '89).

On computing functions

A neural network with one hidden layer can approximate (to any desired precision) any continuous function with any number of inputs and outputs (Cybenko '89).

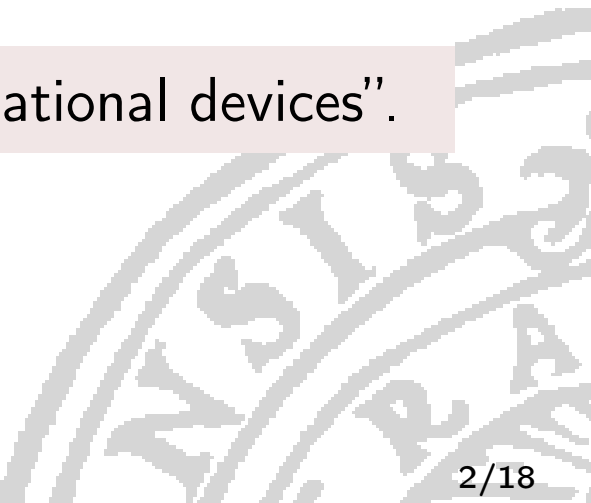
the more hidden neurons, the better the approximation (fix $\varepsilon > 0$, there exists a number of hidden neurons such that the error of the net is below ε for every input).

On computing functions

A neural network with one hidden layer can approximate (to any desired precision) any continuous function with any number of inputs and outputs (Cybenko '89).

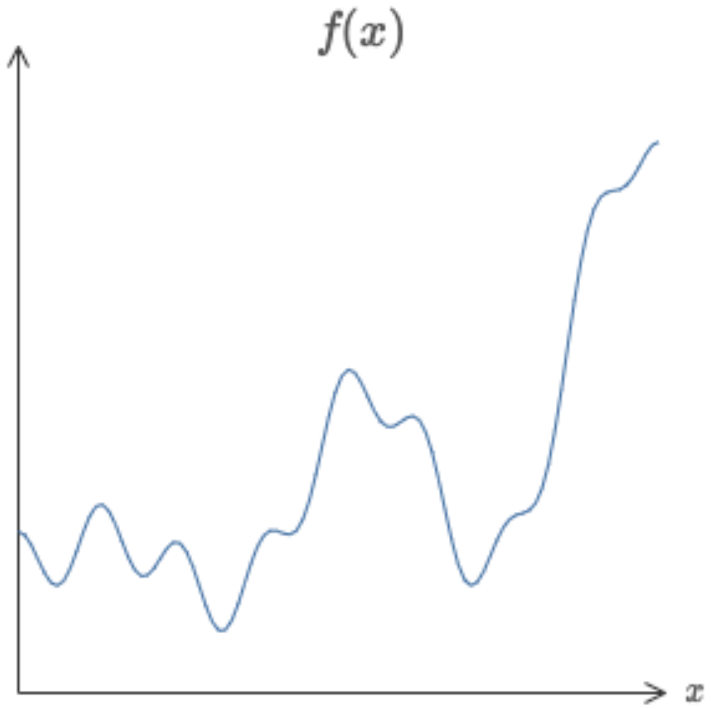
the more hidden neurons, the better the approximation (fix $\varepsilon > 0$, there exists a number of hidden neurons such that the error of the net is below ε for every input).

Therefore, neural networks are "universal computational devices".

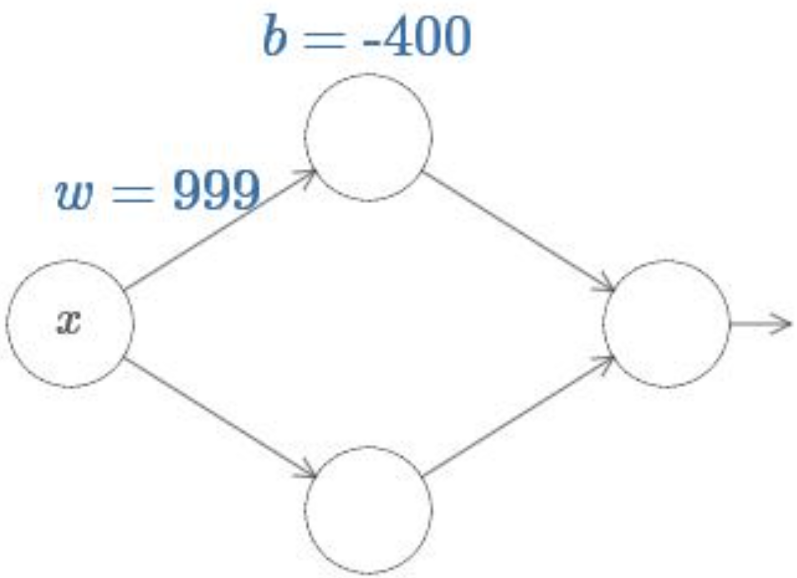


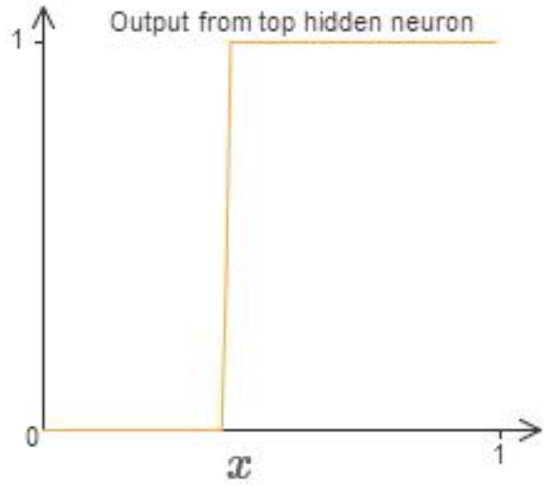
Simple case

Let us consider a single input-single output function:



Step 1: what a single hidden neuron can compute





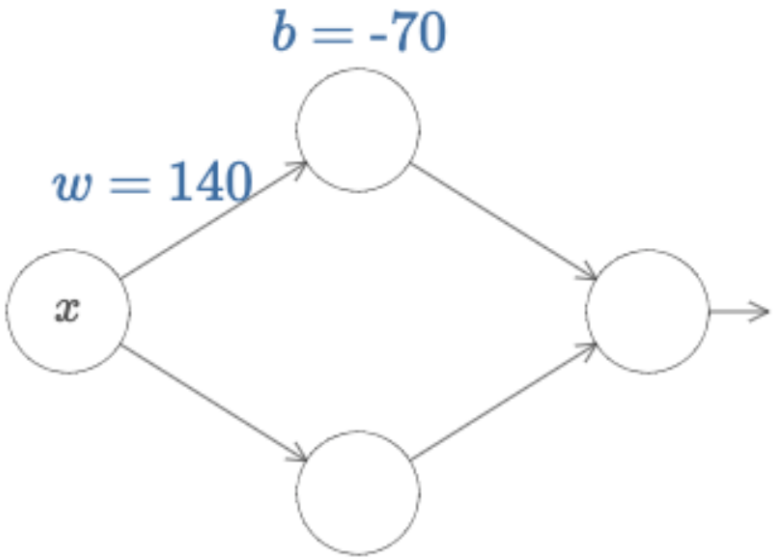
1. fix the weight w to a large value;
2. set the position by modifying the bias b.

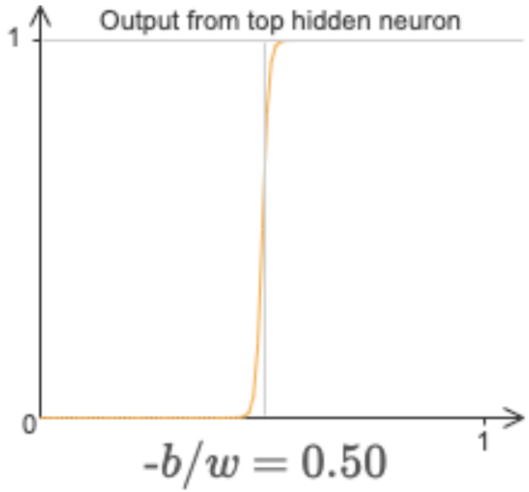
(We use step functions as it is easier to understand their contribution to the output layer)

Position of the step function

With large values of w, the position of the step is:

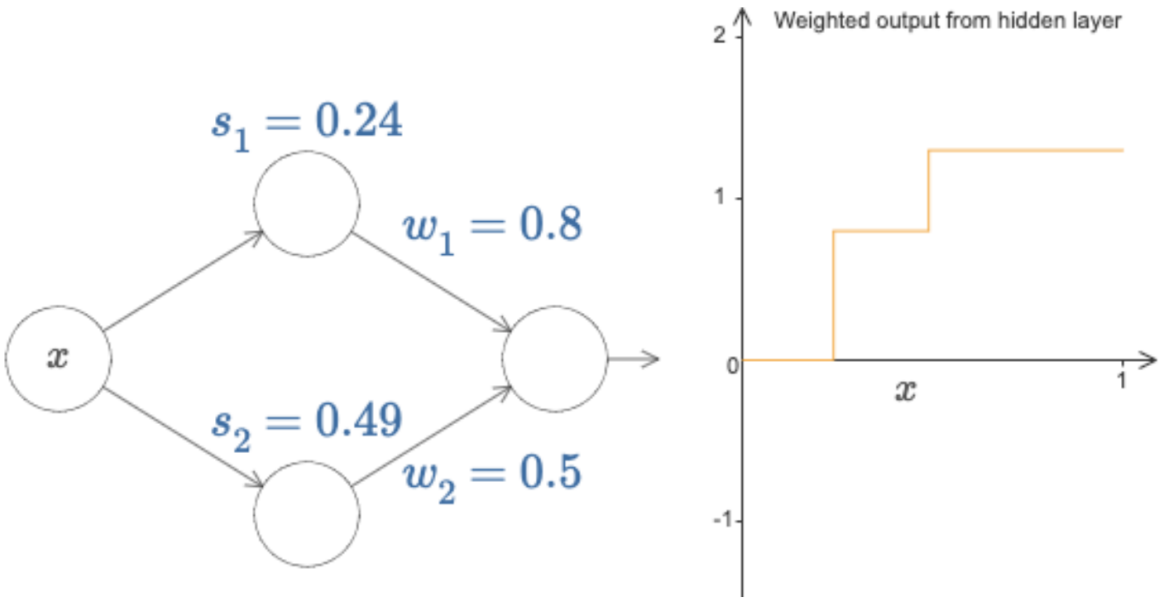
$$s = -\frac{b}{w}$$





Step 2: adding a neuron

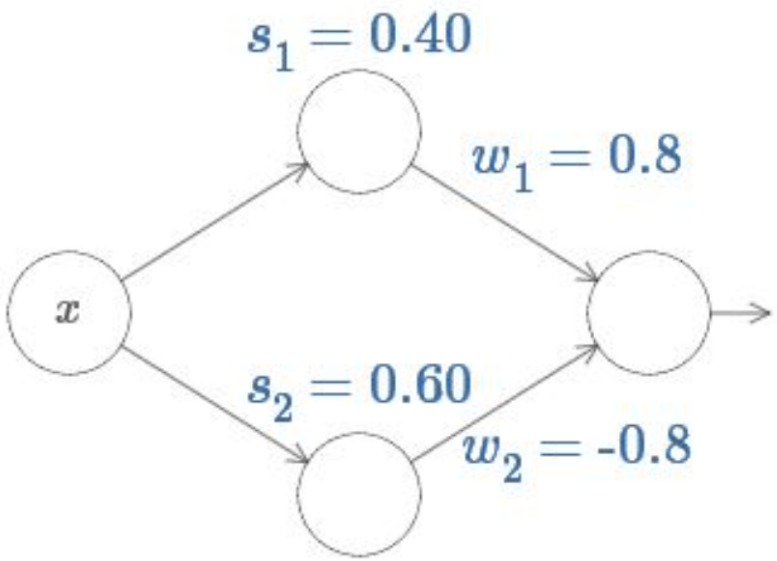
Let us consider the contributions of the bottom hidden neuron.

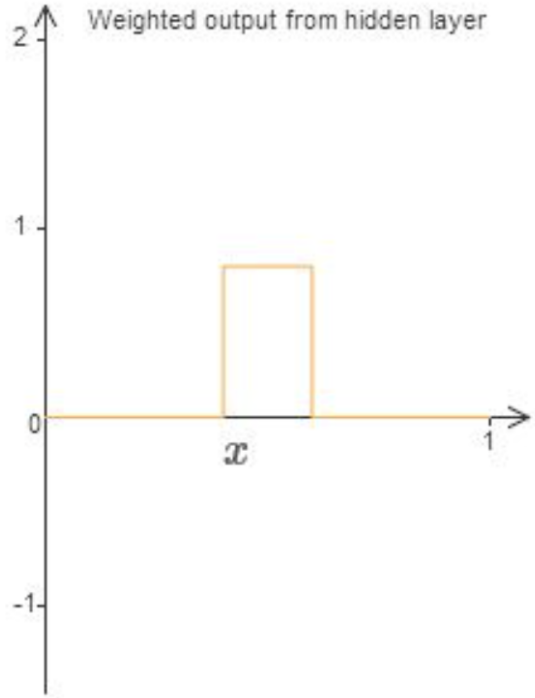


The plot is for $w_1 a_1 + w_2 a_2$ where a_1 and a_2 are the *activations* of the top and bottom hidden neurons. This is different from the output of the net, which computes $\sigma(w_1 a_1 + w_2 a_2 + b)$ where b is the bias of the output neuron.

Step 3: the "bump"

By adjusting the weights of the output layer, we can obtain a "bump" of any height, positioned depending on s_1 and s_2.

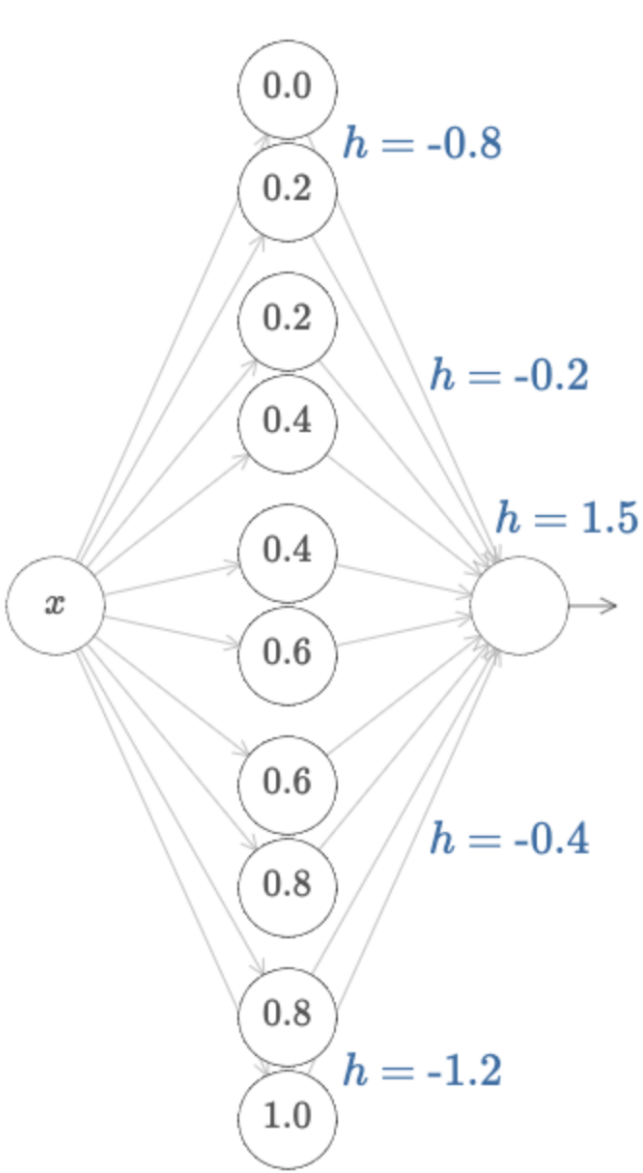


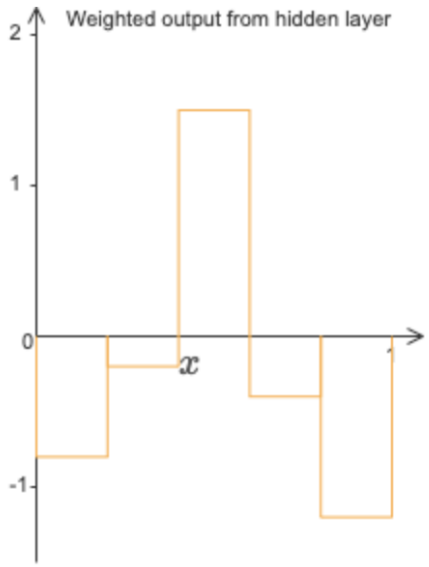


To simplify notation, we can introduce parameter h for the height (instead than w_1 and $w_2 = -w_1$).

Step 4: add pairs of hidden neurons

By adding pairs of hidden neurons, each pair "generating" a bump...



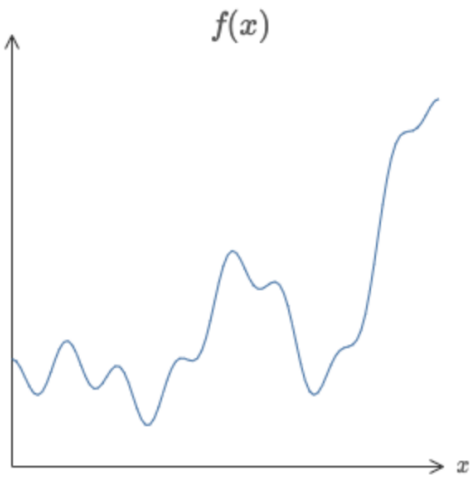


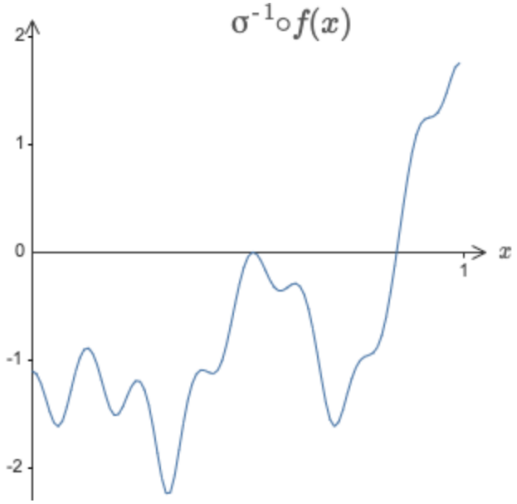
The missing step

The plots so far show the weighted sum of the activation of the hidden layer, not the output of the network...

The missing step

The plots so far show the weighted sum of the activation of the hidden layer, not the output of the network...





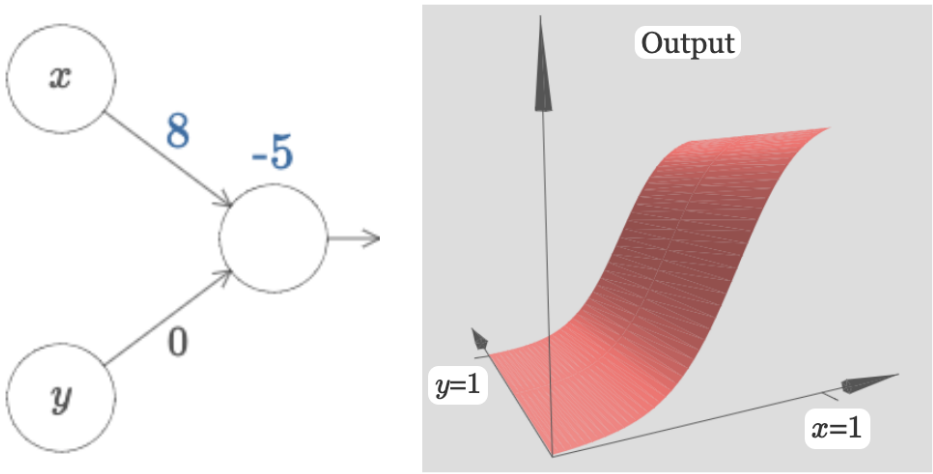
If we try to approximate $\hat{f} = \sigma^{-1}(f(x))$ with $w_1 a_1 + w_2 a_2 + \ldots w_n a_n$ then the output of the NN is:

$$\sigma(\hat{f}) = w_1 a_1 + w_2 a_2 + \ldots w_n a_n$$

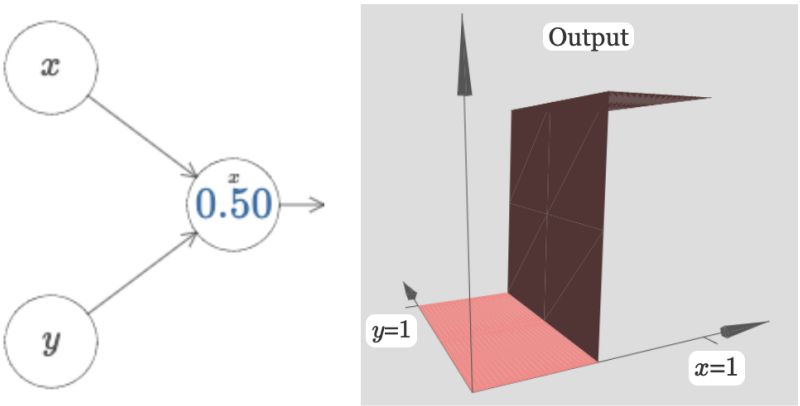
where σ^-1 is the inverse of σ and we set the bias of the last layer to 0.

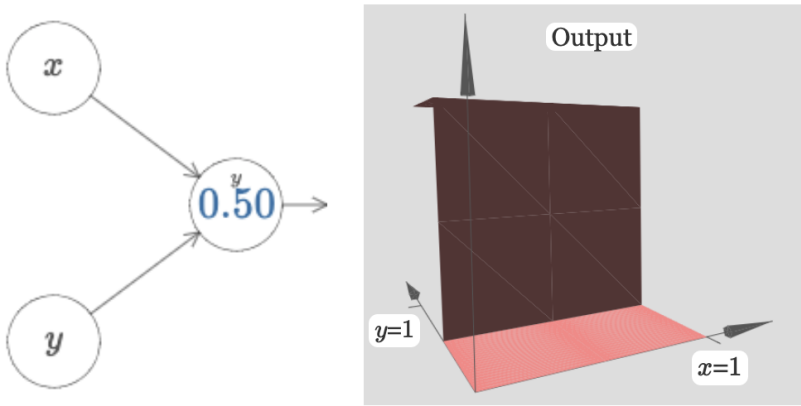
Many input variables, single hidden neuron

Same idea but on a n-dimensional space.



Step function in x and y directions

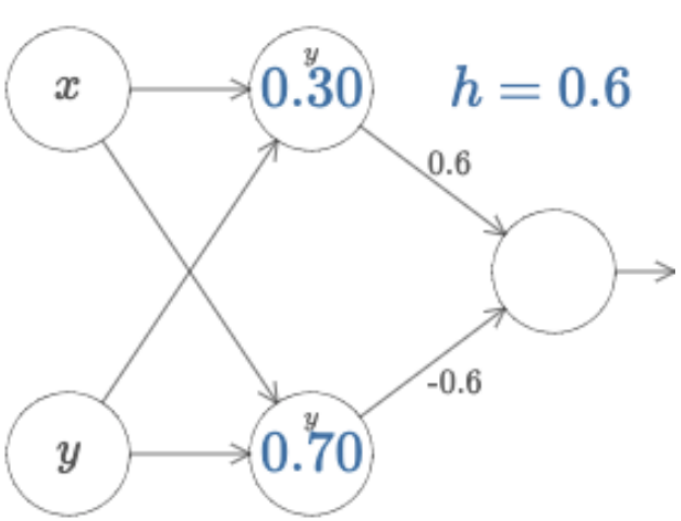


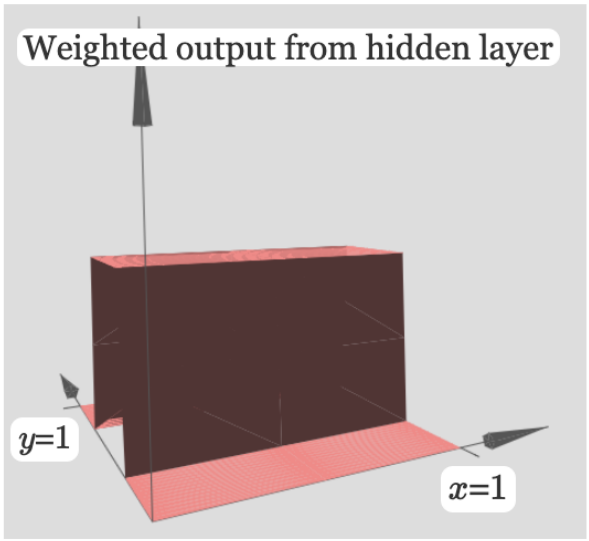


$$S_x = -\frac{b}{w_1} \text{ with } w_2 = 0$$

$$S_y = -\frac{b}{w_2} \text{ with } w_1 = 0$$

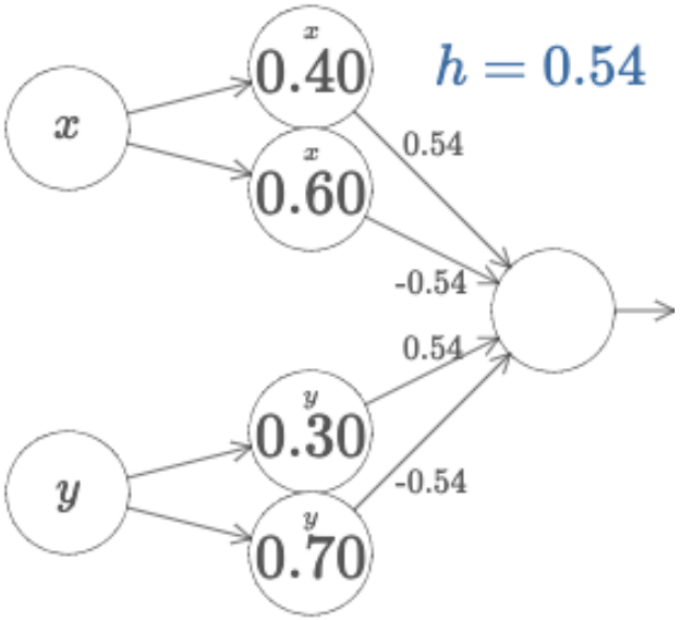
Two hidden units: multi-dimensional bump

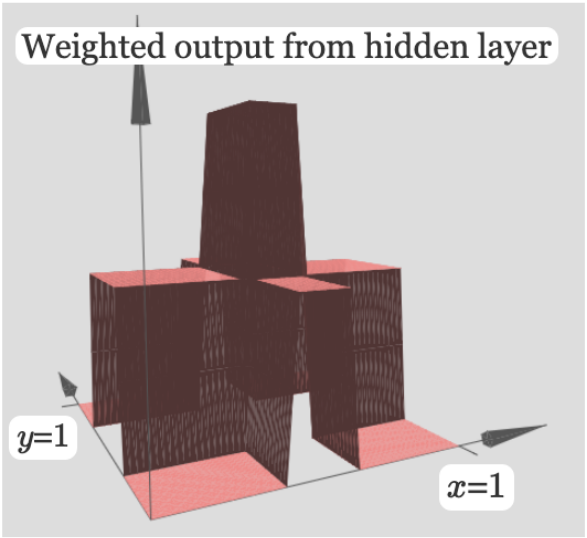




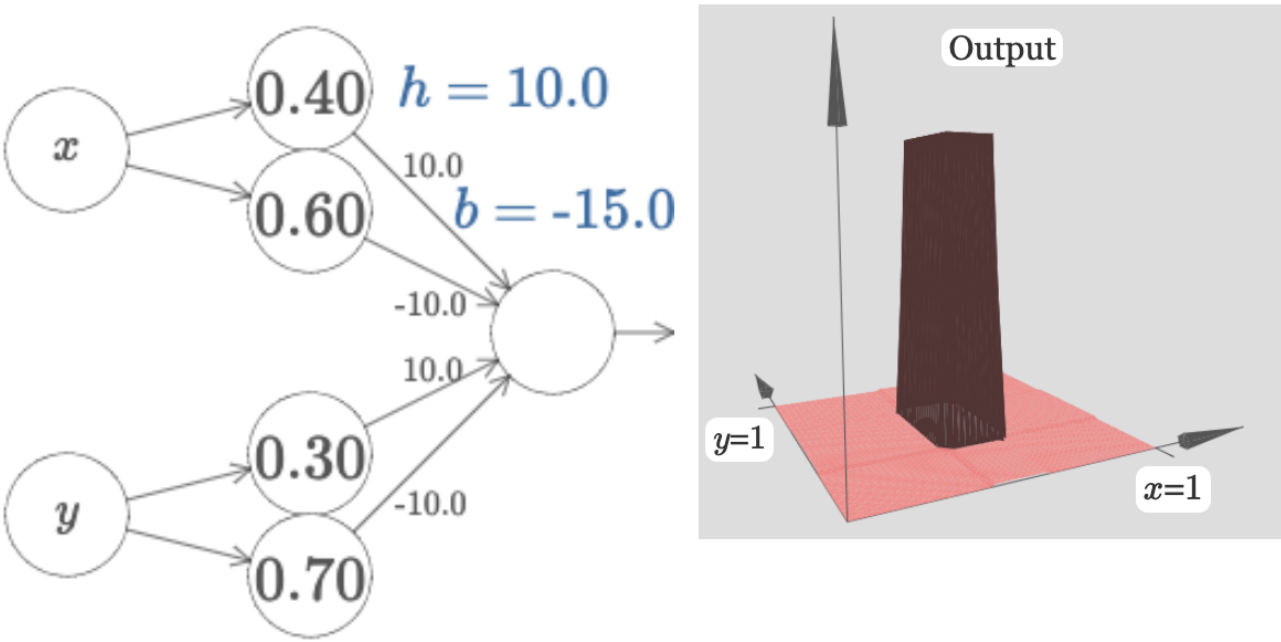
where parameter h is as before, and controls the height of the bump.

More hidden neurons...



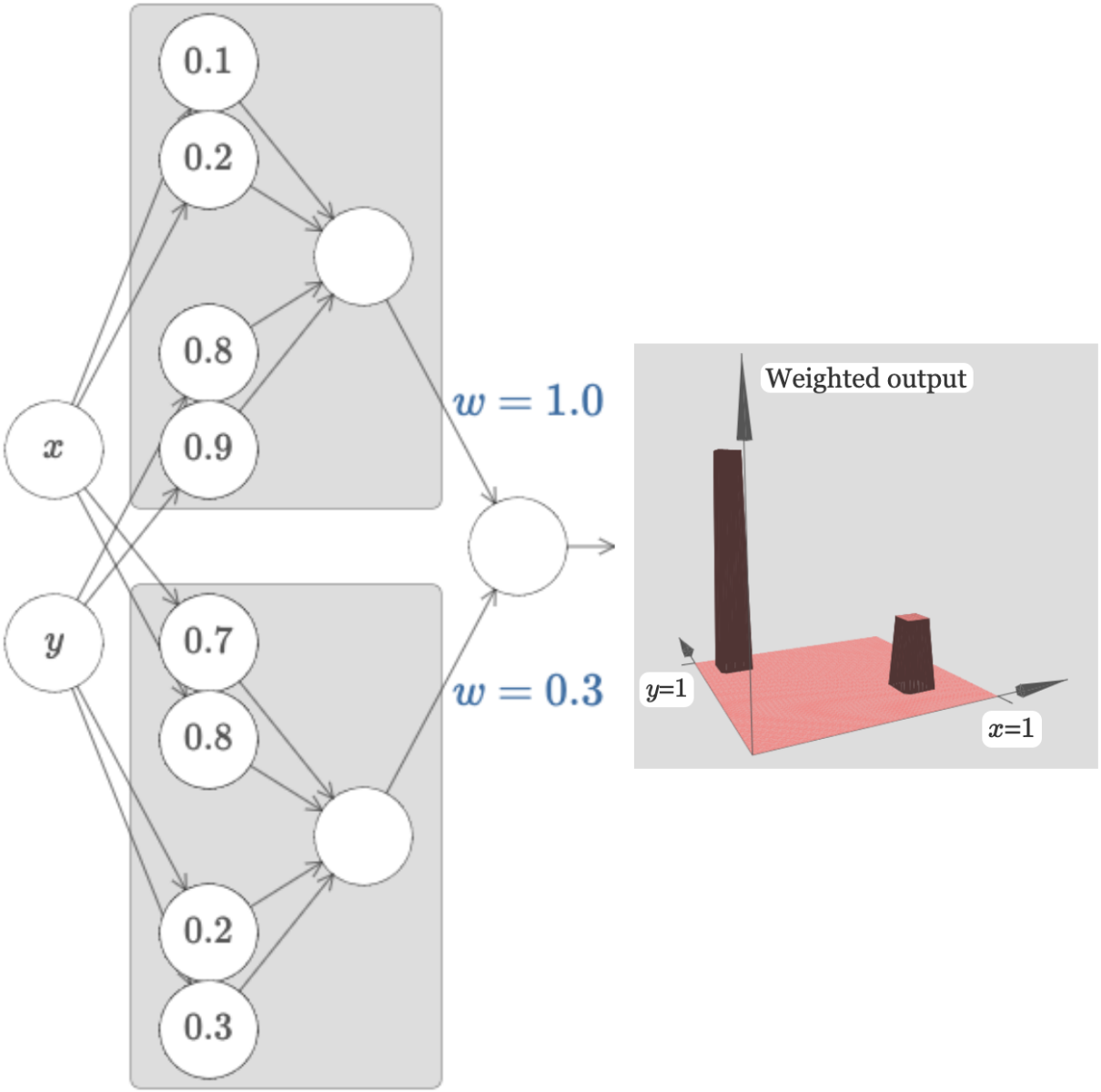


Now we do need the bias on the output layer

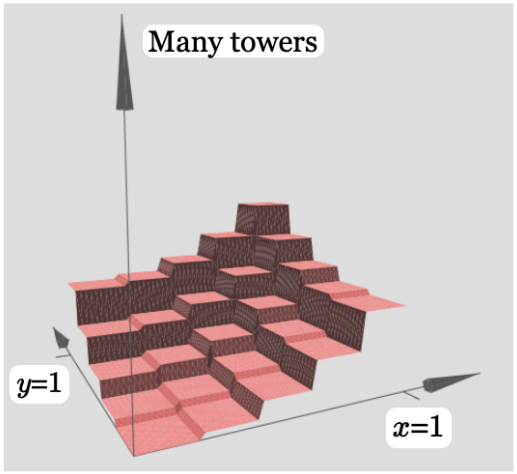


Generally, $b \approx -\frac{3h}{2}$

More towers...



Many towers



Once again, the shape of of the above should approximate $\sigma^{-1}(f)$, as all the plots we have seen before represent the weighted output in the last layer (not the net output).

Even more input variables

Same reasoning, we try to compute multi-dimensional towers.
The bias of the output neuron is then set to:

$$(-m + \frac{1}{2})h \quad \text{where } m \text{ is the number of inputs/dimensions}$$

Final remarks

- This just give an intuition of the universality of NN, does not say that it is the right way of learning functions;
- we have showed universality with more than one hidden layer, while the theorem says one hidden layer...
- that does not mean it is in general a good idea to use only one layer (think of convolutional networks).