Deep Learning Theory and Applications

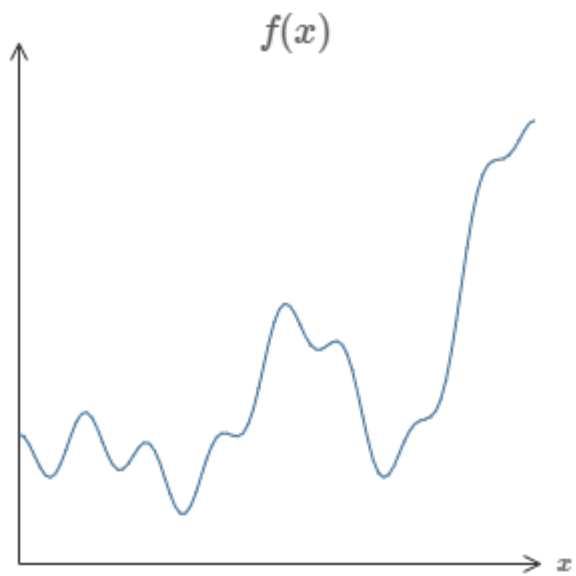
Universality of Neural Networks

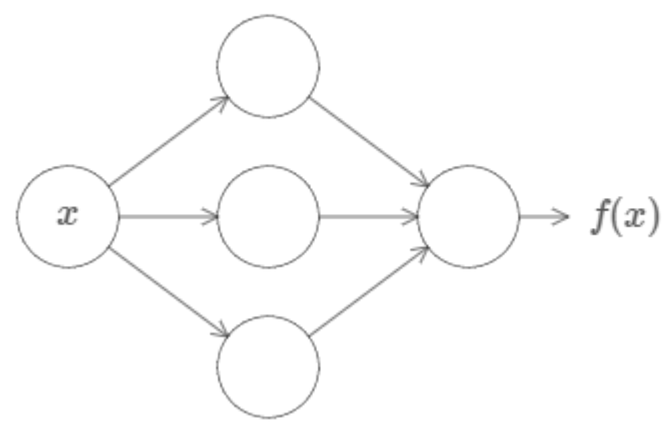
Outline

1. Introduction
2. Universality with one input and one output
3. Many inputs
4. Wrap-up
   - Beyond sigmoid functions
   - Fixing step functions

Universality of neural networks

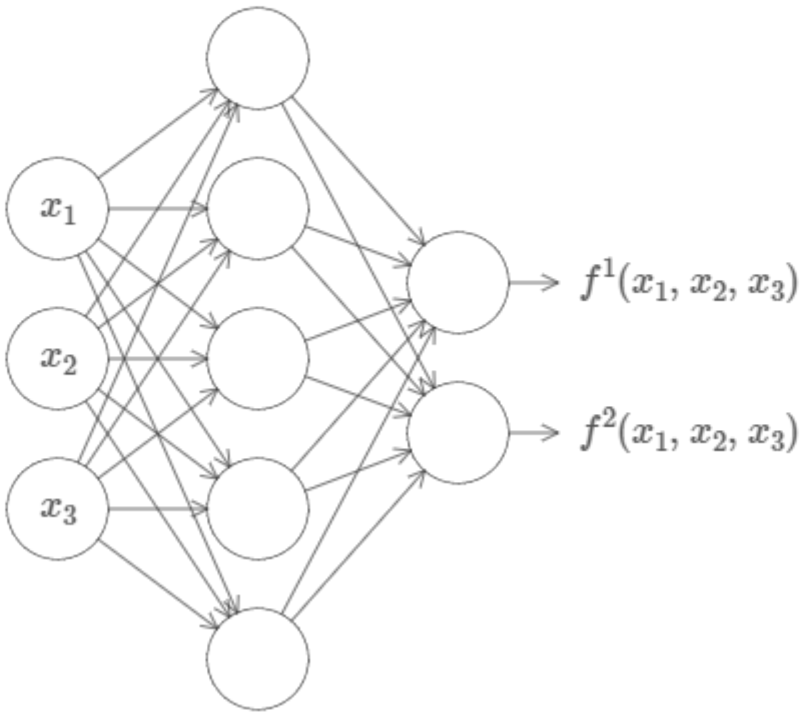
- Neural networks can compute (almost) any function
- No matter the function, there is guaranteed to be a neural network that for every possible input x, the network closely approximates $f(x)$





Universality of neural networks

- Also applies to functions with many inputs and many outputs



Universality of neural networks

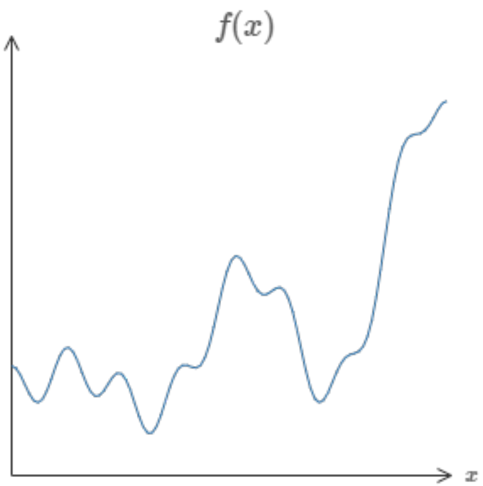
- In other words, neural networks have a ***universality***
  - No matter what function we want to compute, there is a neural network that can do it
- This is true for networks with only a single hidden layer
- Most proofs are quite technical
- Today, we'll give a simple and visual explanation of the universality theorem

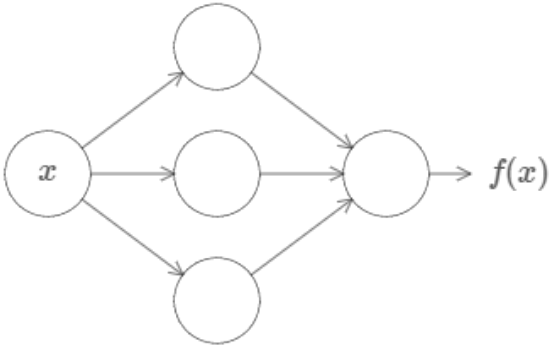
A note on universality

- Universality theorems are quite astonishing
  - The ability to compute an arbitrary function is remarkable
- Almost any process can be thought of as function computation
- Examples:
  - Name a piece of music based on a short sample
  - Translate Chinese text to English
    - There may be many possible functions
  - Generate a plot description from a movie file
- Universality means that neural networks can do all of these things and more
  - However, this doesn’t mean that we have good techniques for constructing or even recognizing such a network

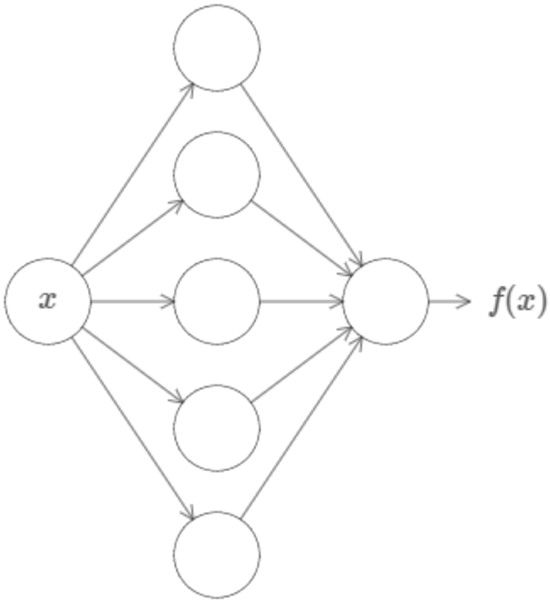
Some caveats

- Neural networks can't *exactly* compute any function
  - Rather we can get an *approximation* that is as good as we want
  - Increasing the number of hidden neurons can improve the approximation





Poor approximation



Better approximation

- Can do better with even more hidden neurons

Some caveats

- Let's make this more precise
- Suppose we're trying to approximate $f(x)$ within some accuracy $\epsilon > 0$
- The guarantee is that with enough hidden neurons, there exists a neural network whose output $g(x)$ satisfies $\forall x$

$$|g(x) - f(x)| < \epsilon$$

Some caveats

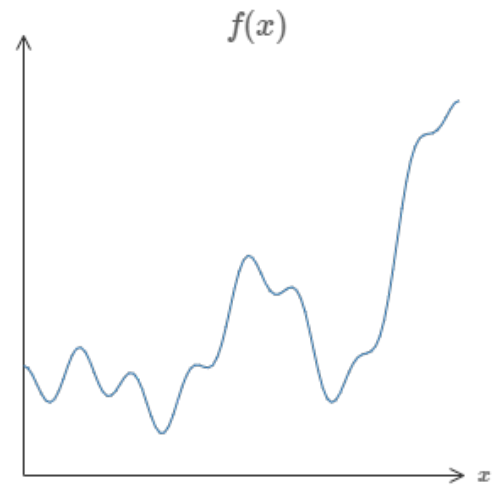
- Second, we can only guarantee this accuracy for *continuous* functions
  - If a function is discontinuous, then it won't be generally possible to approximate it at each point since the neural network output is continuous
- However, often a continuous approximation of a discontinuous function is good enough

Some caveats

- So in practice, continuity isn't a major limitation

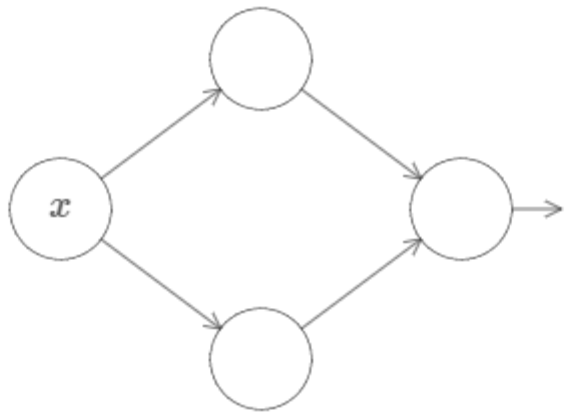
- Summary: neural networks with a single hidden layer can be used to approximate any continuous function to any desired precision
  - For simplicity, we'll focus on the case with 2 hidden layers

Universality with one input and one output

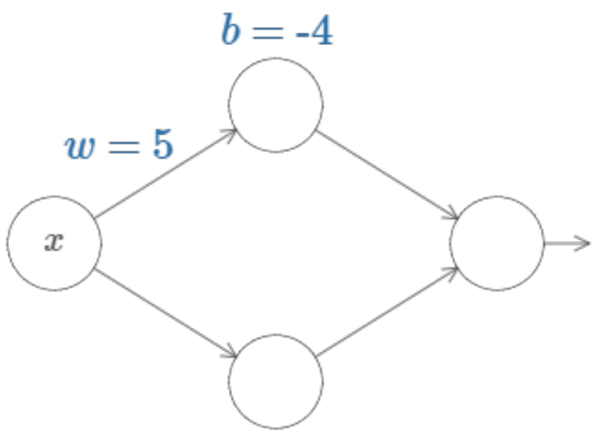


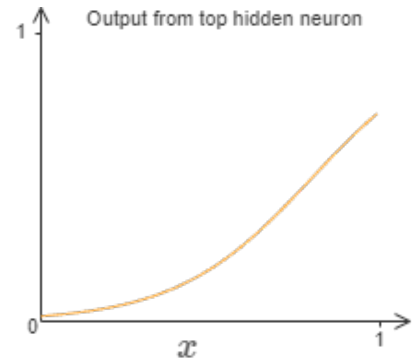
One input, one output

- Start simple

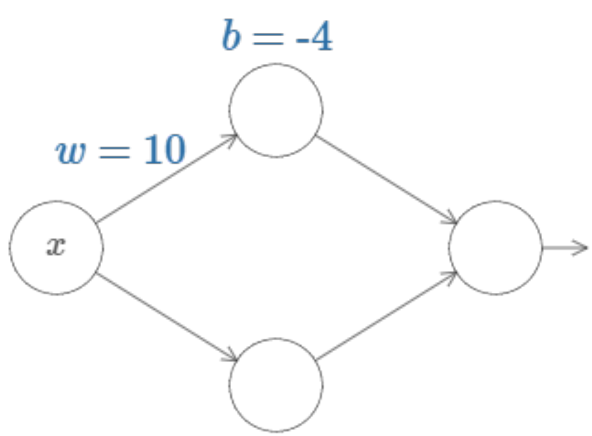


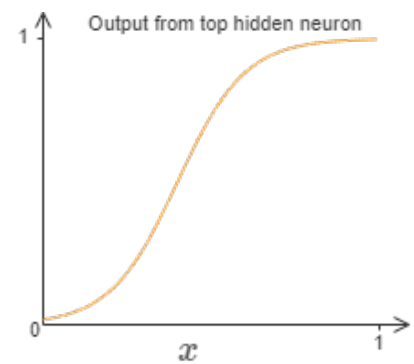
One input, one output



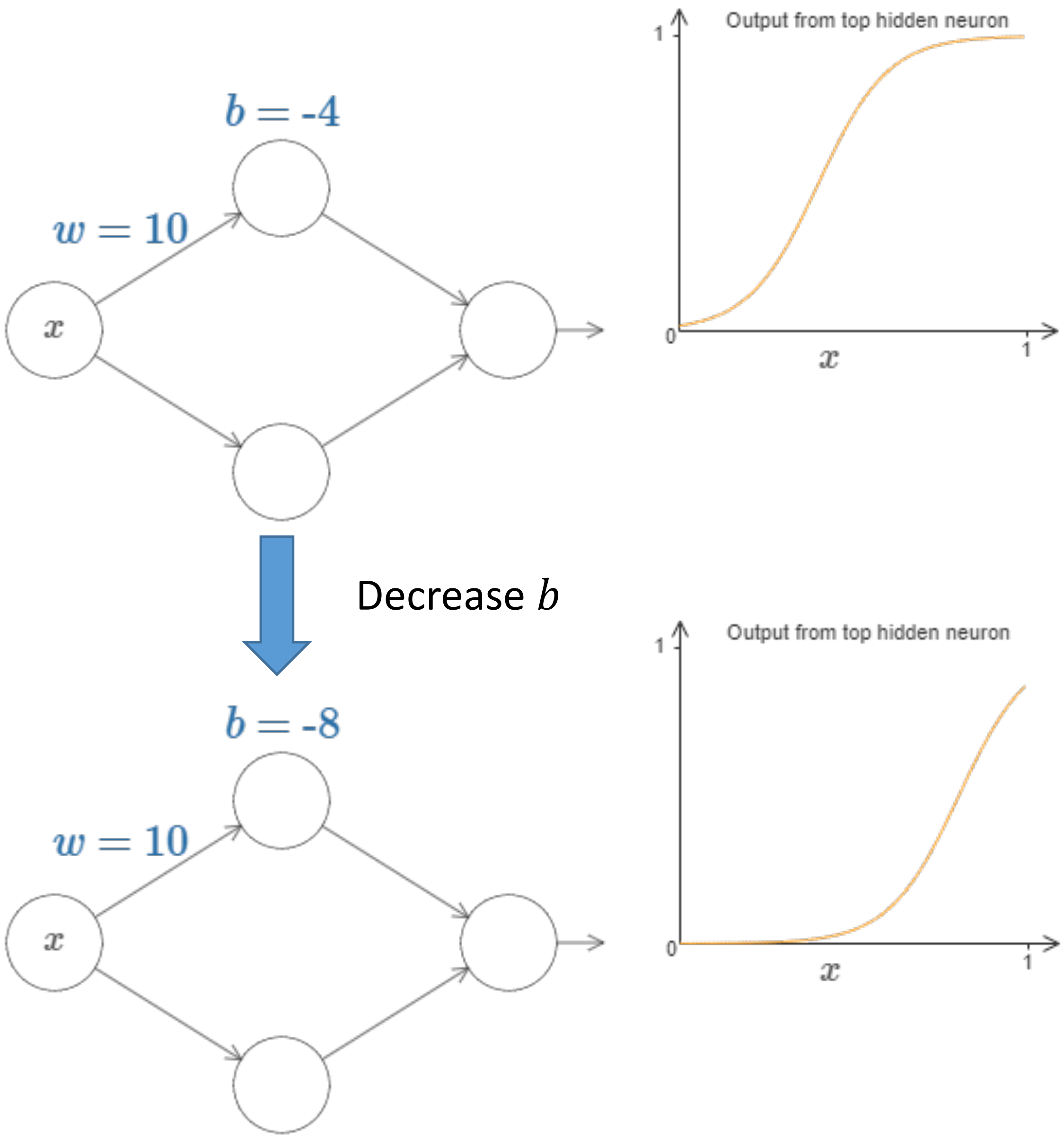


Increase w

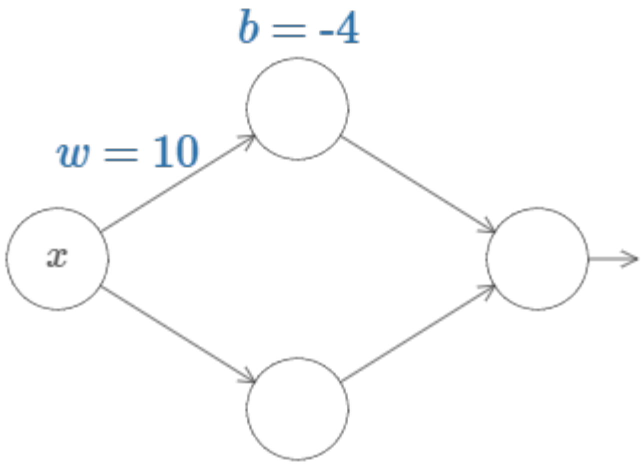


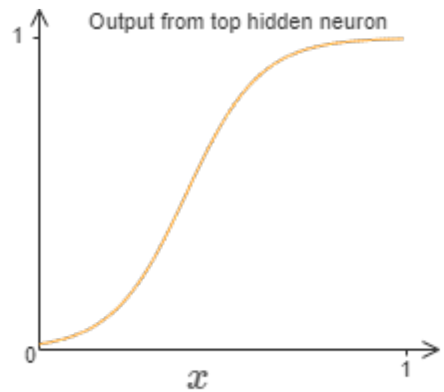


One input, one output

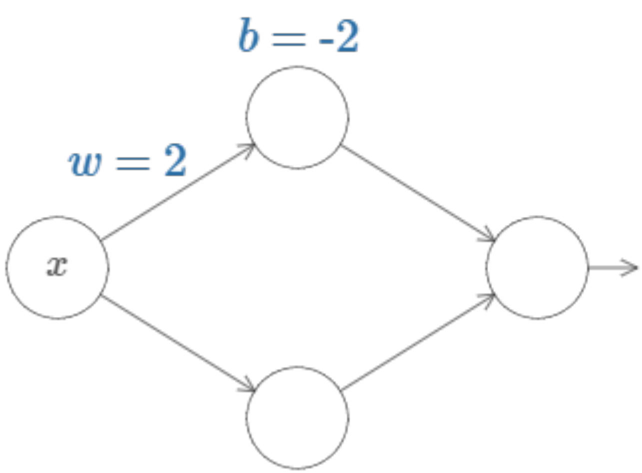


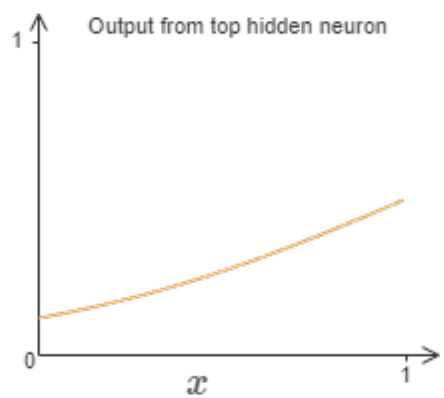
One input, one output



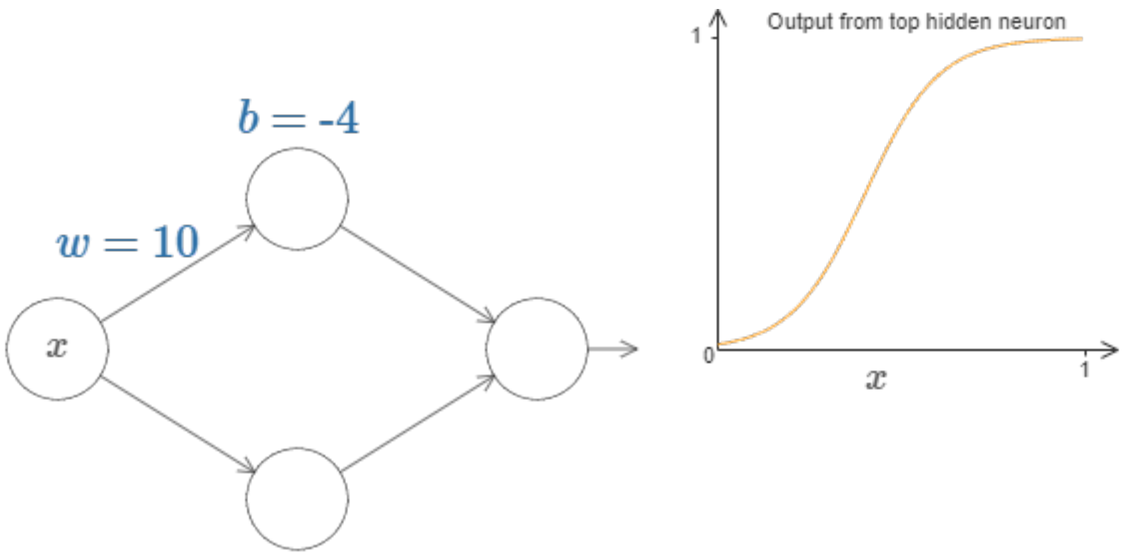


Decrease w

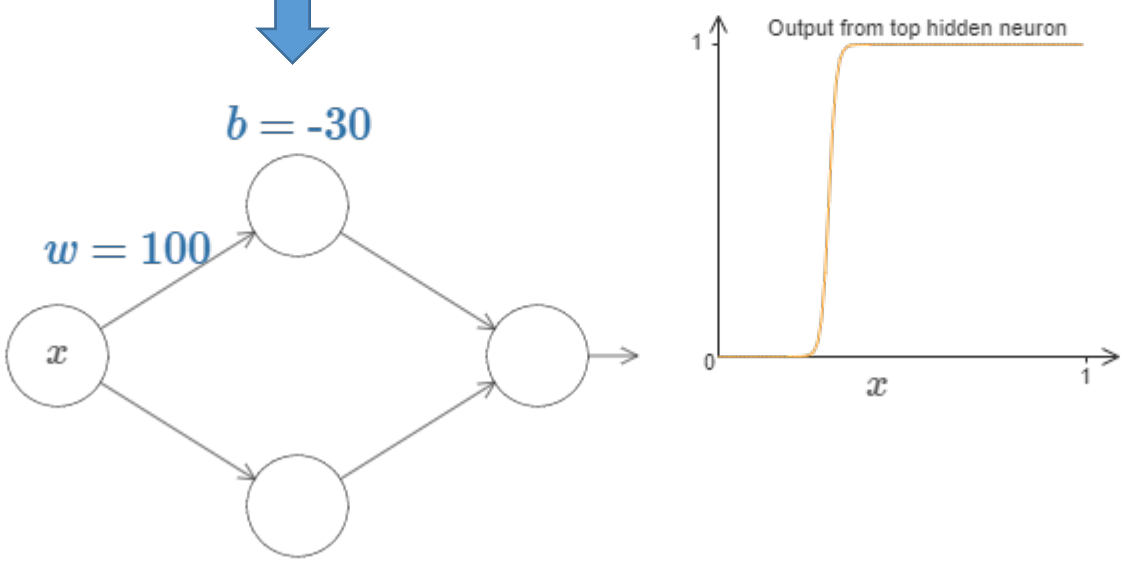




One input, one output

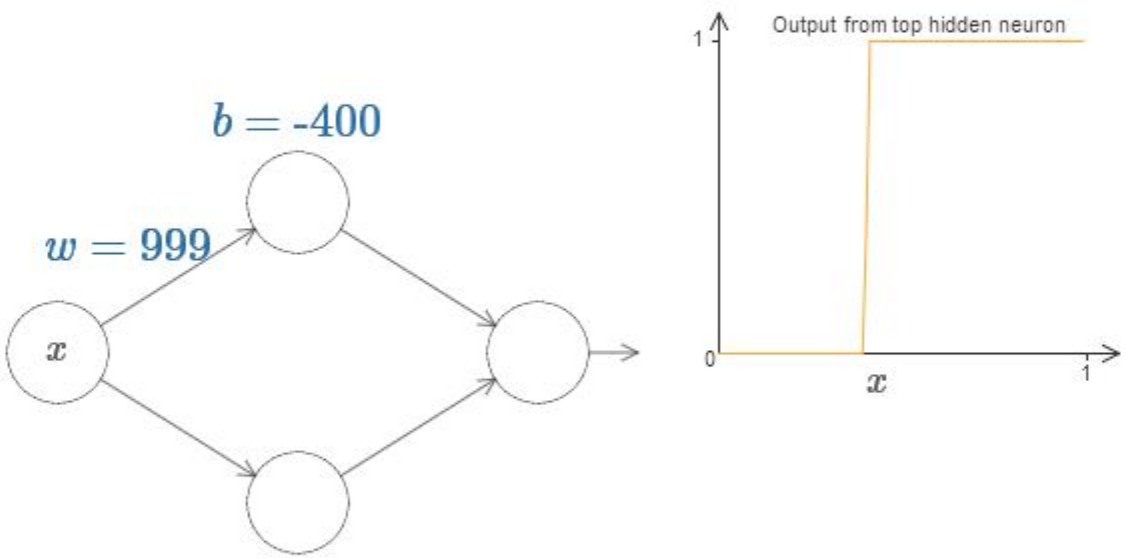


Increase w



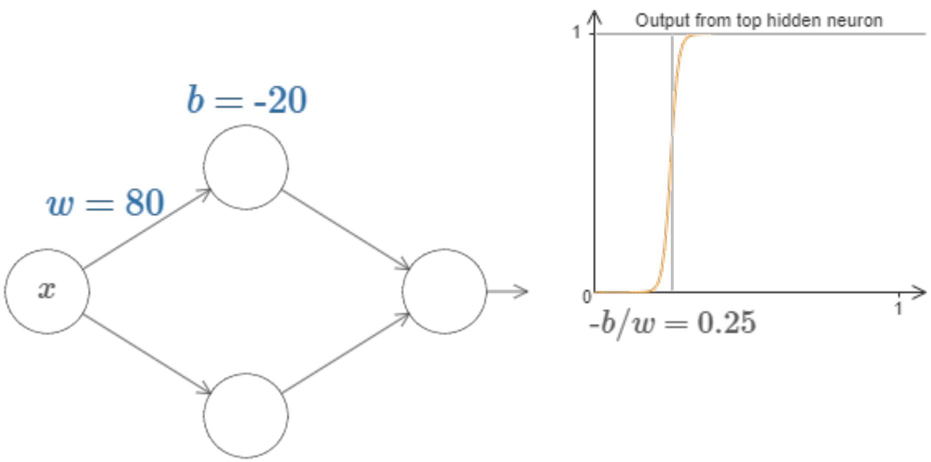
One input, one output

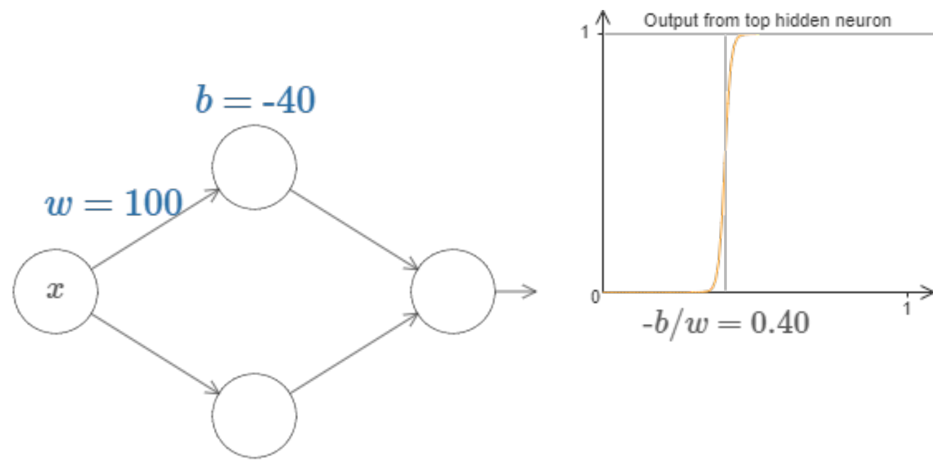
- We can simplify our analysis a lot by using a step function
  - The output layer is a sum of contributions from all hidden neurons
  - Easier to analyze the sum of step functions
  - Approximate a step function by setting *w* to be very large, and modifying the bias appropriately
  - Later, we'll cover the effect of this approximation



One input, one output

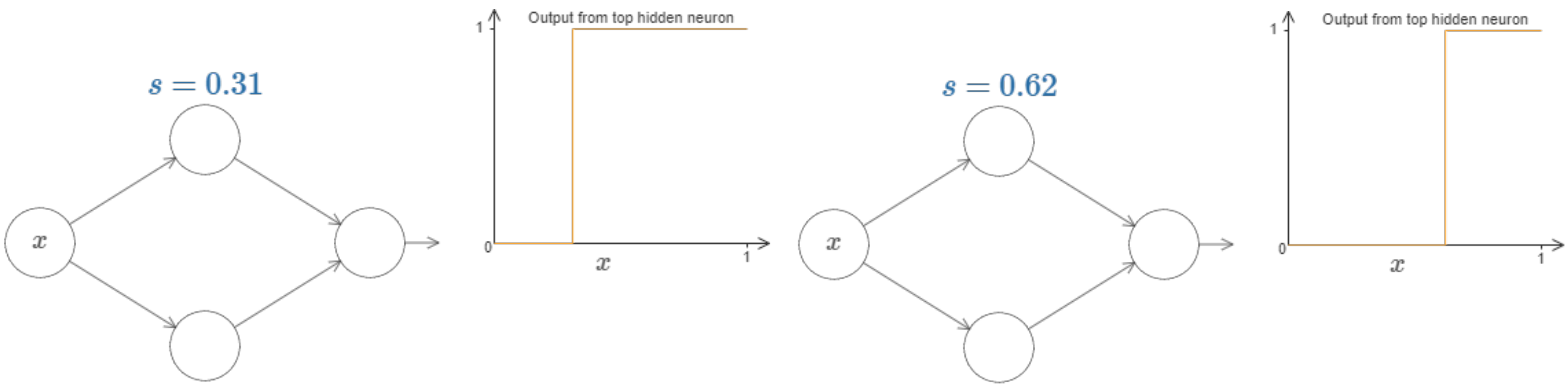
- Where does the step occur?
- The position of the step is proportional to b and inversely proportional to w
  - The step is at position $s = -\frac{b}{w}$





One input, one output

- We can simplify things by using a step function with parameter s
  - I.e., we set w to be some very large value and then adjust b
  - Recover $b = -ws$

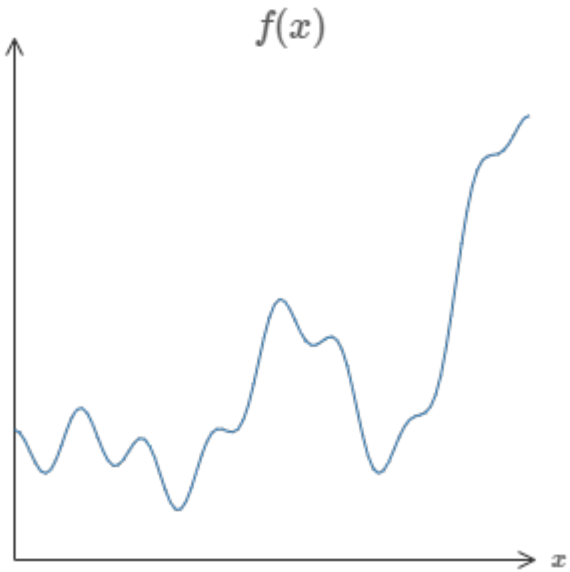


One input, one output

- Let's add the bottom node now
- http://neuralnetworksanddeeplearning.com/chap4.html#universality_with_one_input_and_one_output

One input, one output

- Challenge: approximate this function

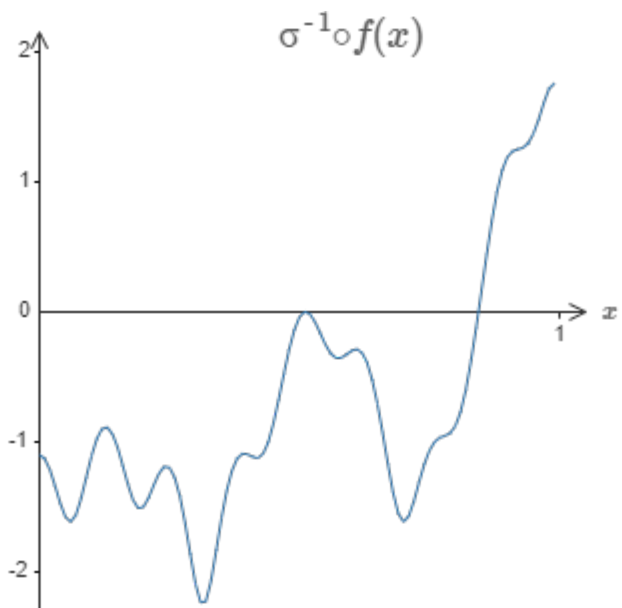


$$f(x) = 0.2 + 0.4x^2 + 0.3x\sin(15x) + 0.05\cos(50x)$$

- Range and domain are $[0,1]$

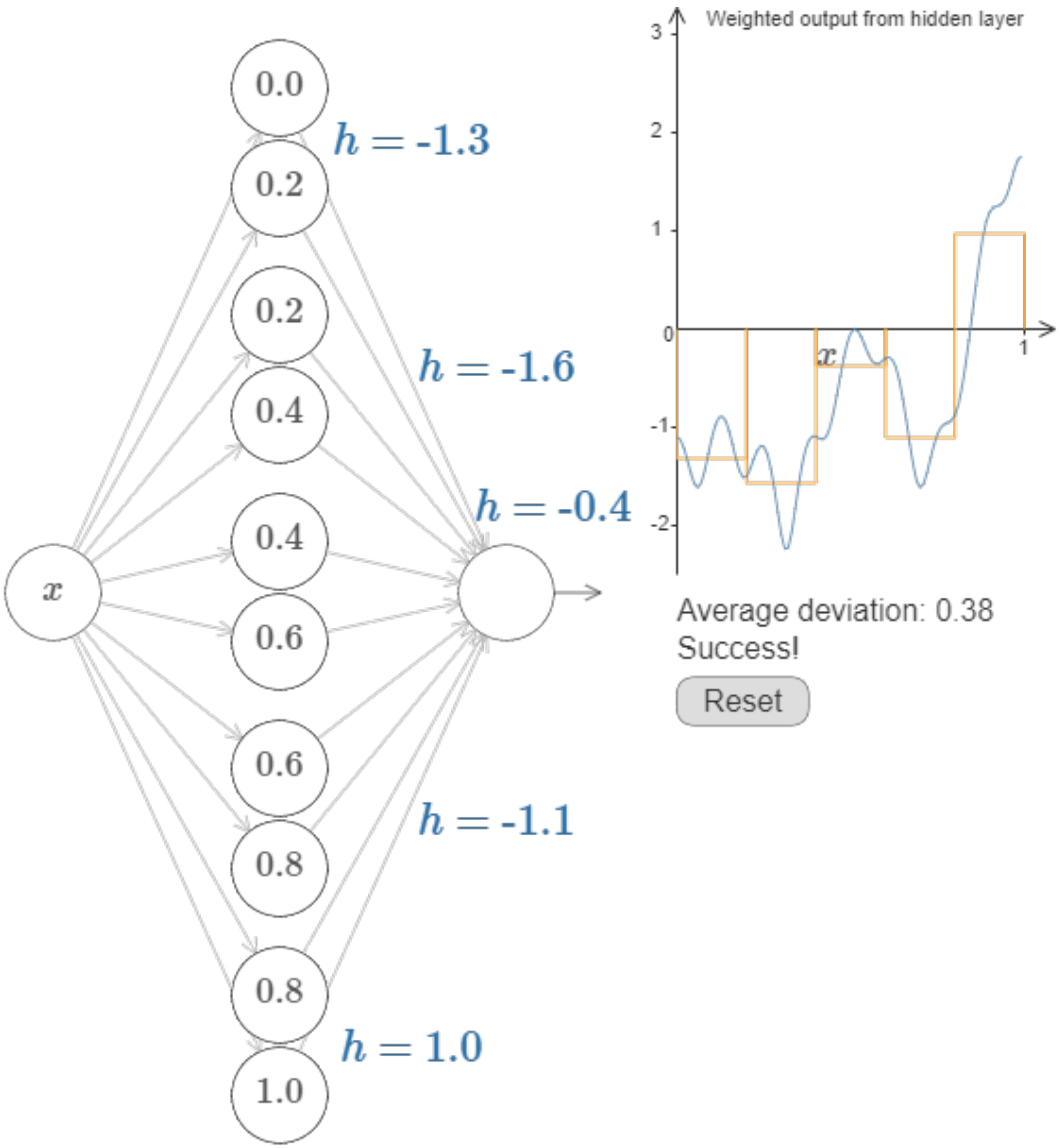
One input, one output

- We've been looking at the weighted combination from the hidden neurons $\sum_j w_j a_j$
- The actual output is $\sigma\left(\sum_j w_j a_j + b\right)$
- Take the inverse of the sigmoid function: $\sigma^{-1} \circ f(x)$



- http://neuralnetworksanddeeplearning.com/chap4.html#universality_with_one_input_and_one_output

One input, one output



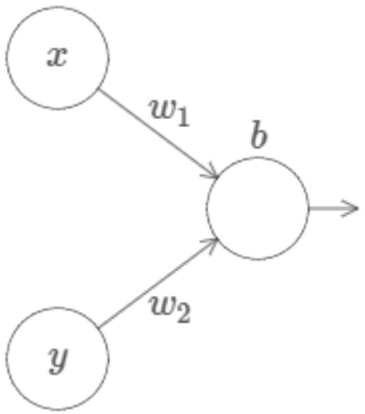
One input, one output

- How do we convert back to standard parameterization?

1. Set $w = 1000$ for first layer of weights
2. Biases on hidden neurons are $b = -ws$
3. Final layer of weights come from the $\pm h$ values
4. Bias on the output neuron is 0

Many inputs

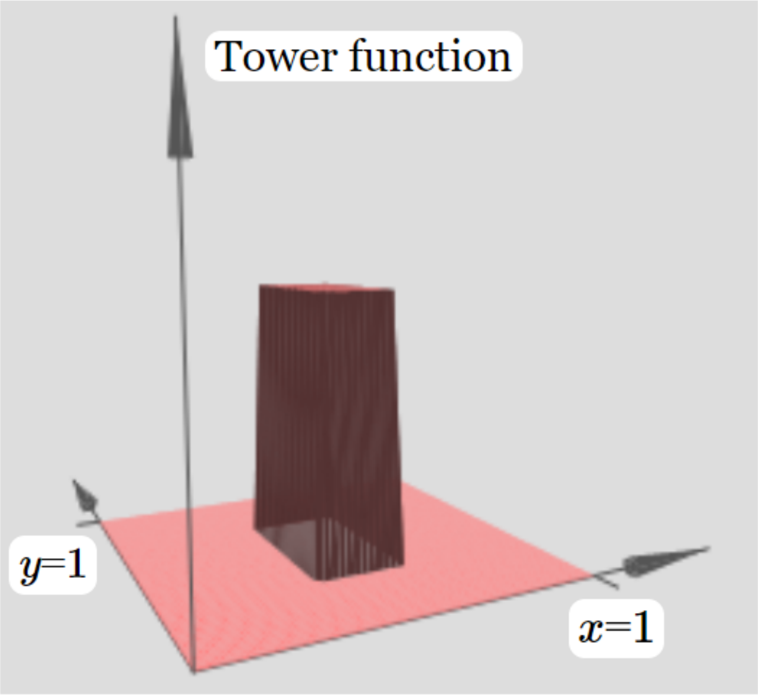
Two inputs



- http://neuralnetworksanddeeplearning.com/chap4.html#many_input_variables

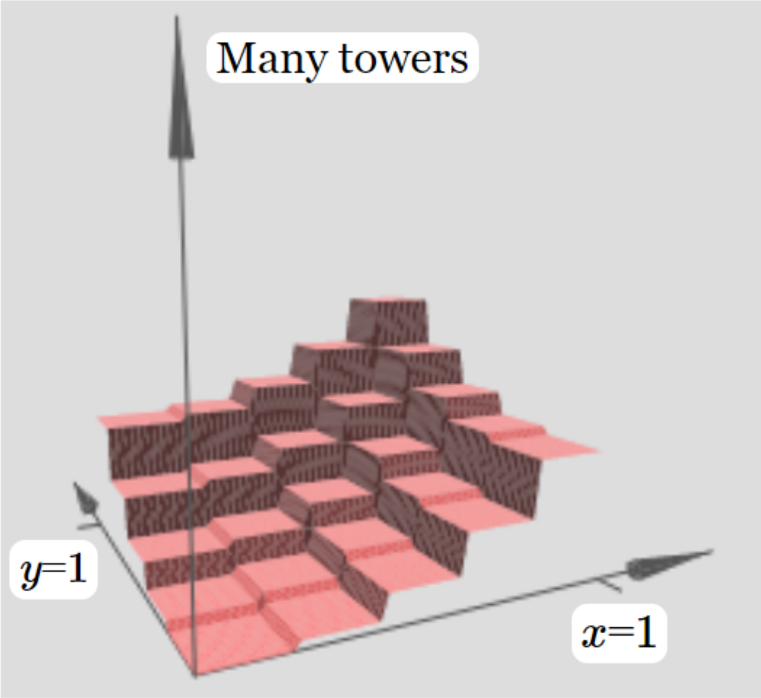
Two inputs

- We've built something that looks a little like a tower function



Two inputs

- We can approximate arbitrary functions by adding towers of different heights in different locations



Two inputs

- With step functions, we've been implementing an if-then-else statement with neurons:

```
if input >= threshold:
    output 1
else:
    output 0
```

- We can generalize this for multiple inputs:

```
if combined output from hidden neurons >= threshold:
    output 1
else:
    output 0
```

- If we choose an appropriate threshold, we can squash the plateau down and leave only the tower

Two inputs

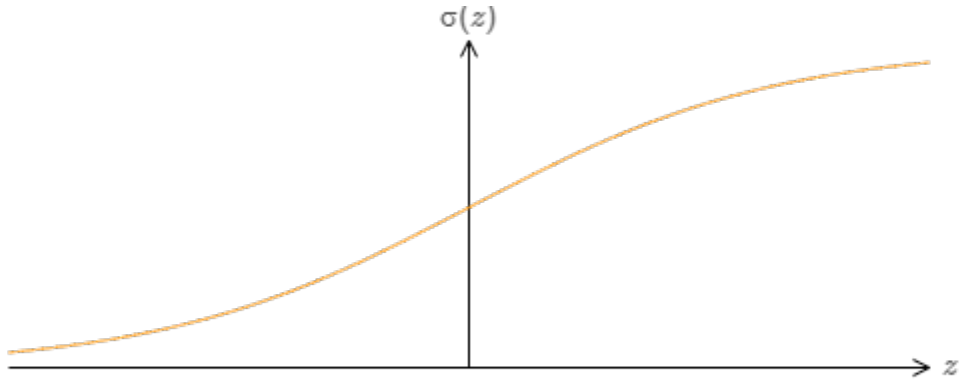
- Let’s make a tower
- http://neuralnetworksanddeeplearning.com/chap4.html#many_input_variables

Multiple outputs

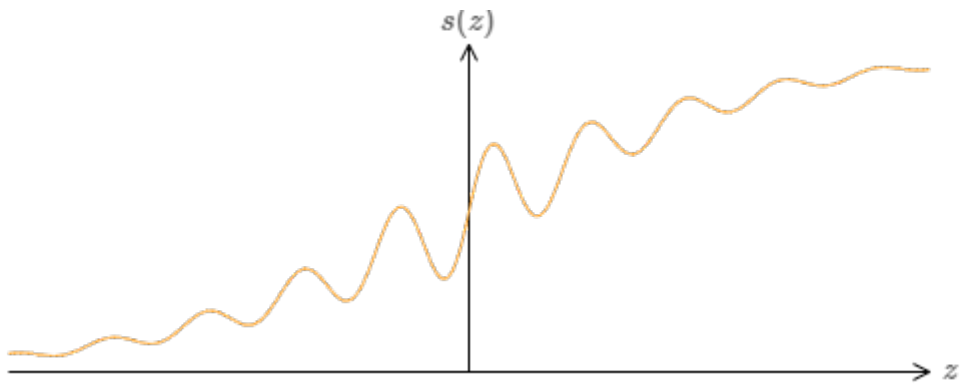
- What about multiple outputs?

- A vector-valued function can be viewed as d real-valued functions
- We can simply construct a network approximating each component

Beyond sigmoid neurons



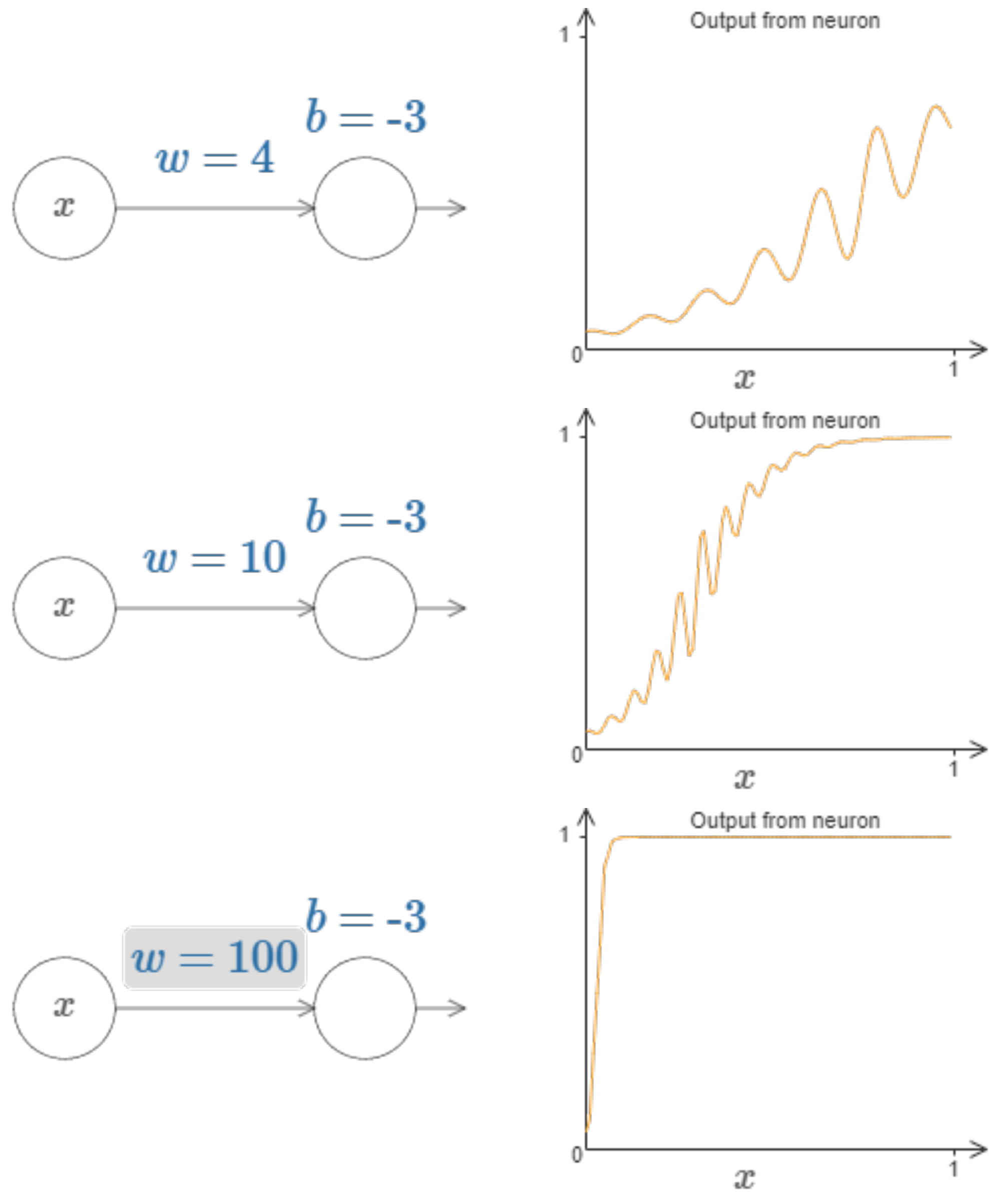
- What if we used this instead:



Beyond sigmoid neurons

- Increasing the weight gives an approximation of a step function

- Changing the bias changes the position of the step



Beyond sigmoid neurons

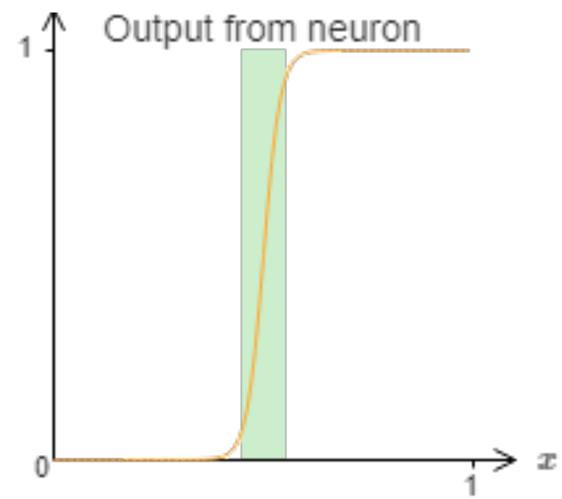
- What properties do we need for this approach?

1. Need $s(z)$ to be well-defined as $z \to -\infty$ and $z \to \infty$
   - These are the values taken by the step function
2. The limits must be different from each other
   - Otherwise we get a constant function

- These conditions are sufficient but not necessary for universality
  - The ReLU activation function also gives universality

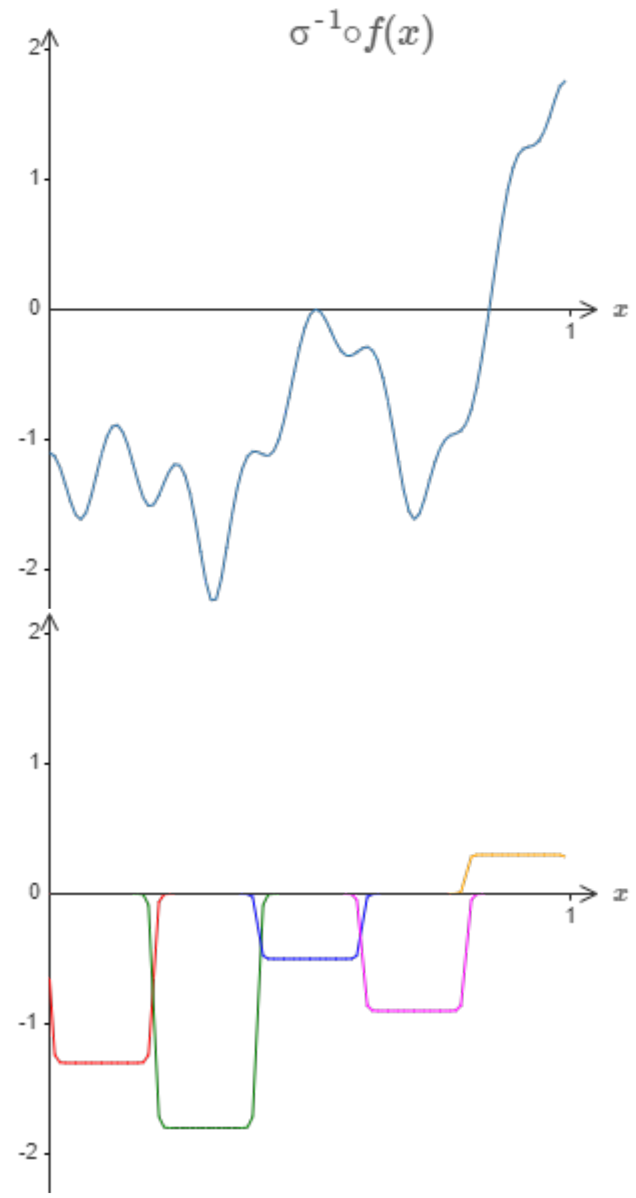
Fixing the step functions

- We've been assuming our neurons produce exact step functions
- We actually only get an approximation
  - There's a narrow window of failure
  - We can increase the weights to make the window small
  - But is there a better way?



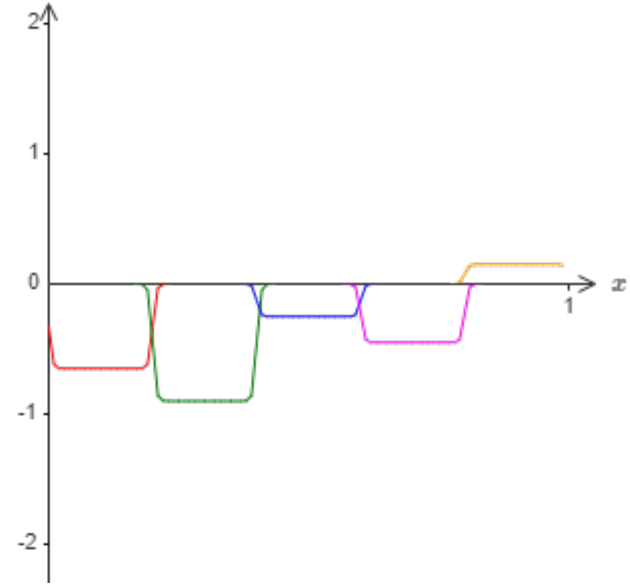
Fixing the step functions

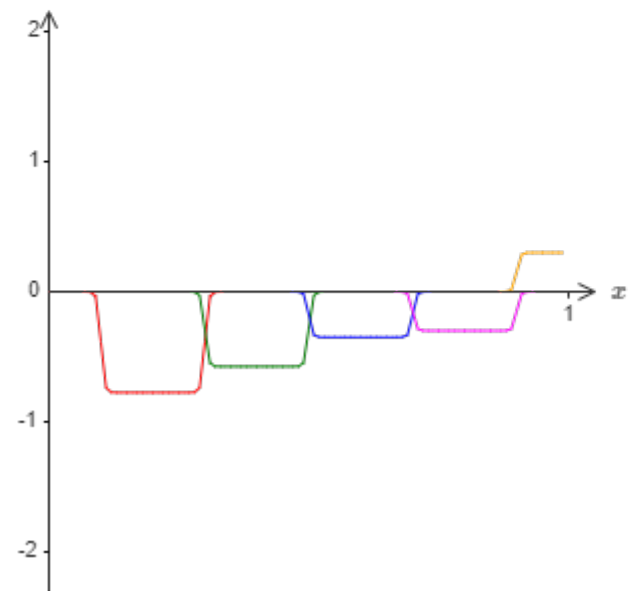
- Consider the function from before
- We can approximate it with a sequence of bump functions
  - Windows of failure have been exaggerated for demonstration
  - We get a reasonable approximation except within the windows of failure



Fixing the step functions

- Let's approximate half the original function: $\sigma^{-1} \circ f(x)/2$
- Now approximate $\sigma^{-1} \circ f(x)/2$ shifted by half a bump
- Adding these together gives an overall approximation of $\sigma^{-1} \circ f(x)$
- The approximation is roughly a factor of 2 better in the windows of failure
- Could get further improvement by approximating $\sigma^{-1} \circ f(x)/M$ with M overlapping approximations





Wrap-up

- This explanation does not give a good prescription for designing neural networks!
  - Thus the result isn't directly useful for constructing networks
- However, universality answers the question of whether any particular function is computable with a neural network
- This changes the question to whether there is a good way to learn the function

Wrap-up

- If single layer network is universal, why use deep networks?
  - Note that our universality explanation required many hidden neurons
  - Earlier in the class, we argued that the hierarchical structure of deep networks is also helpful

- Summary:
  - Universality tells us neural networks can compute any function
  - Empirical evidence suggests deep networks are best adapted to learn those functions in practice

Further reading

- Nielsen book, chapter 4
- Roman Vershynin, High-Dimensional Probability, 2018