PadhAI Week 4: Representation Power of Functions

by Manick Vennimalai

4.1: Representation Power of Functions

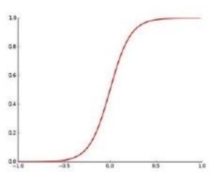
4.1.1: Why do we need complex functions

The need for complex functions

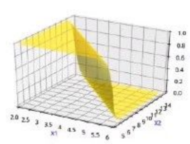
1. Here's a quick recap on what we've covered so far

	Data	Task	Model	Loss	Learning	Evaluation
MP Neuron	{0,1}	Binary Classification	$g(x) = \Sigma^{n}_{i=1} x_i$ y = 1 if g(x) >= b y = 0 otherwise	Loss = $\Sigma_i (y_i \mathrel{!=} \hat{y}_i)$	Brute Force Search	Accuracy
Perceptron	Real Inputs	Binary Classification	y = 1 if $\Sigma^{n}_{i=1} w_i x_i$ >= b y = 0 otherwise	Loss = $\Sigma_i (y_i - \hat{y}_i)^2$	Perceptron Learning Algorithm	Accuracy
Sigmoid	Real Inputs	Classification /Regression	$y = \frac{1}{1+e^{-(w^T x + b)}}$	Loss = $\Sigma_i (y_i - \hat{y}_i)^2$ Or Loss = $-[(1-y)log(1-\hat{y}) + ylog(\hat{y})]$	Gradient Descent	Accuracy/RMSE

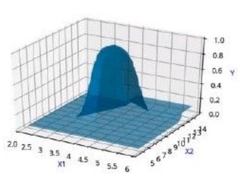
2. We must remember that none of the above 3 models can handle non-linearly separable data
3. Here's another recap on Continuous Functions



$$\hat{y} = \frac{1}{1+e^{-(2*x_1+5)}}$$



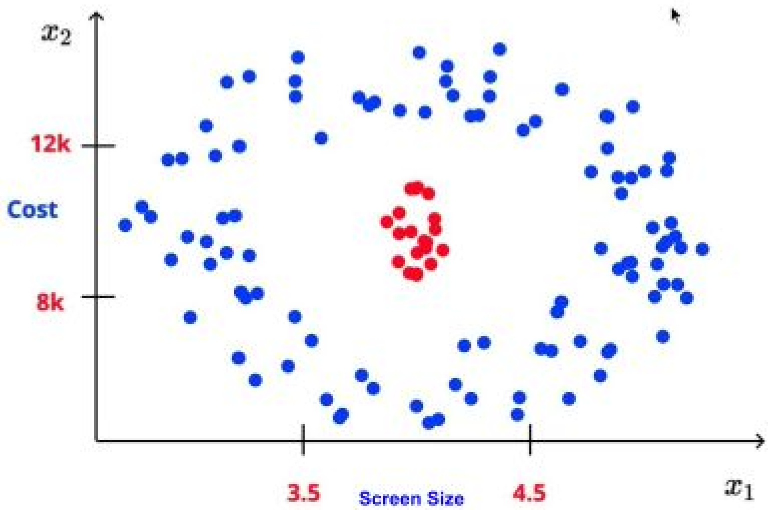
$$\hat{y} = \frac{1}{1+e^{-(-2*x_1+2*x_2+20)}}$$



$$\hat{y} = sig_1(sig_2(x_1, x_2), sig_3(x_1, x_2), sig_4(x_1, x_2))$$

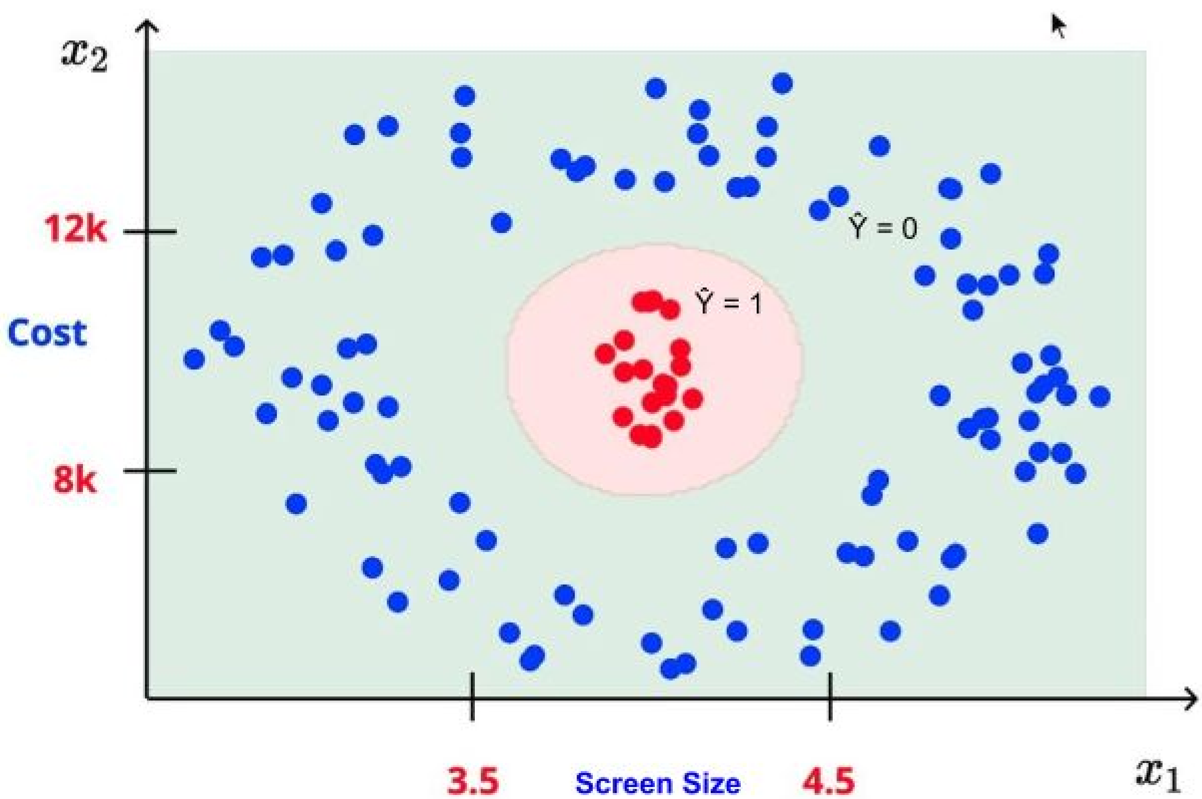
4. We care about continuous functions because our learning algorithm (Gradient Descent) requires that the input functions be differentiable (i.e. Continuous)
5. Let's take a look at a real world example of how complex functions are relevant to our situation

6. Consider the following example of where we're trying to predict like/dislike for a non-linearly separable dataset of mobile phones.



7.

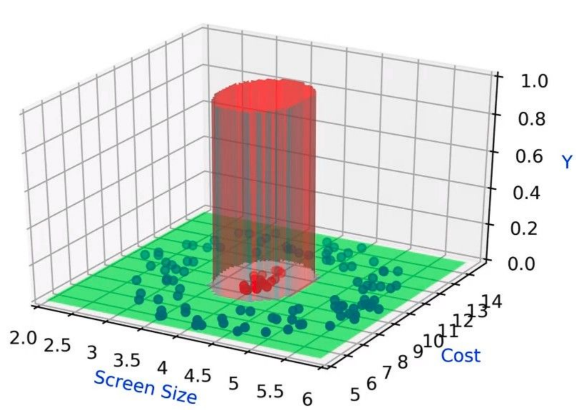
8. Here, our desirable set of phones lies in the centre of a circle of non-desirable phones, based on the values of the variable Cost and Screen Size.
9. Ideally, we would need a decision boundary like so

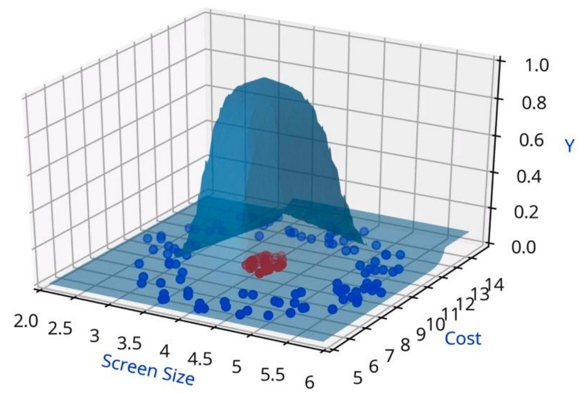


10. However, none of the functions we have seen so far will be able to plot such a decision boundary (ie boundary that separates the two classes = 0 and ŷ = 1)

11. Let's take a 3D plot of the two variables with the output values mapped along the z-axis

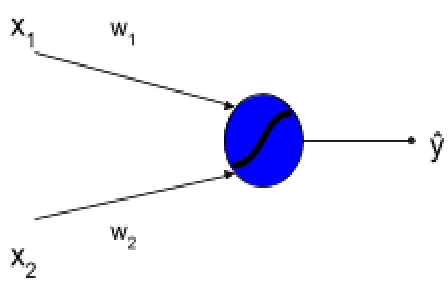
Discrete (abrupt) **Continuous (smooth)**

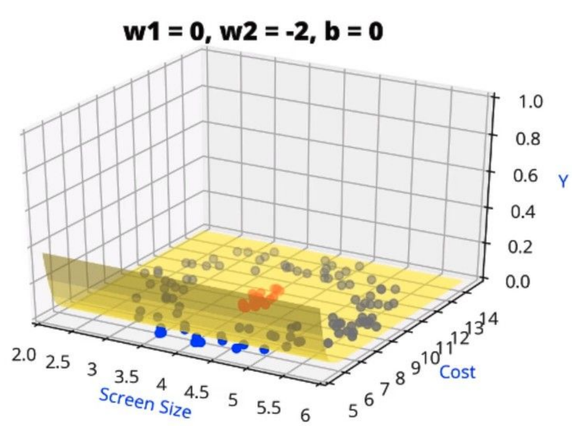


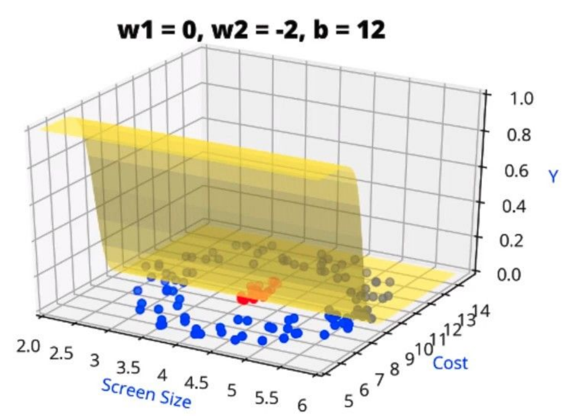


12. Here, the Continuous function has a smooth distribution, and the Y value gradually increases as we converge to the centre, becoming 1 at the region around the red dots
13. However, such an output is not possible with the sigmoid functions, regardless of how we manipulate the values of w and b

Sigmoid decision boundary, can range from s-shape to flat, based on w and b values







14. We can see that the sigmoid function is unsuitable for modelling complex decision boundaries.
15. Such complex relations are actually seen quite frequently in real world examples

4.1.2: Complex functions in the real world

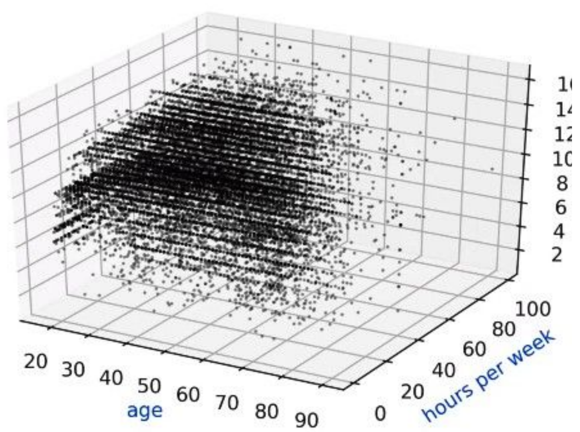
Are such complex functions seen in most real world examples?

1. Consider predicting whether the Annual Income of person $\geq 50k$ *or* $< 50k$

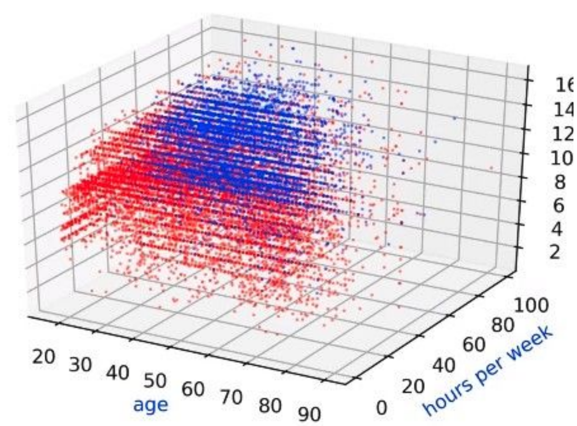
Age	hour/week	Education year	…… More features	Income
90	40	9		0
54	40	4		0
74	20	16		1
45	35	16		1

2. Plotting the data would give us a plot like so

Non Separated



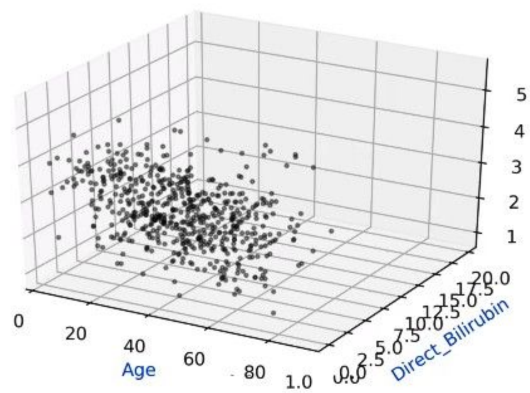
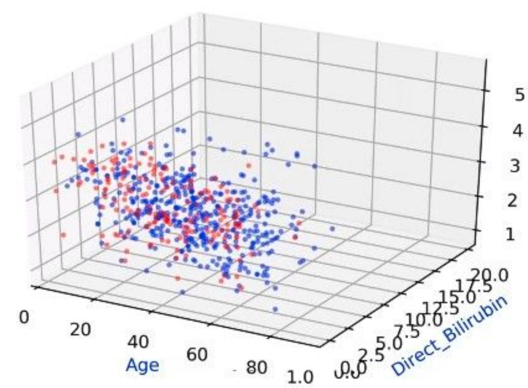
Coloured for +ve and -ve



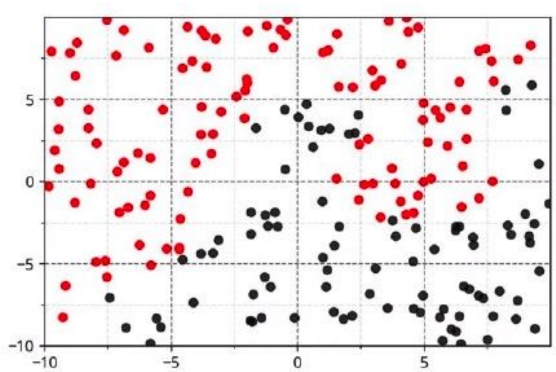
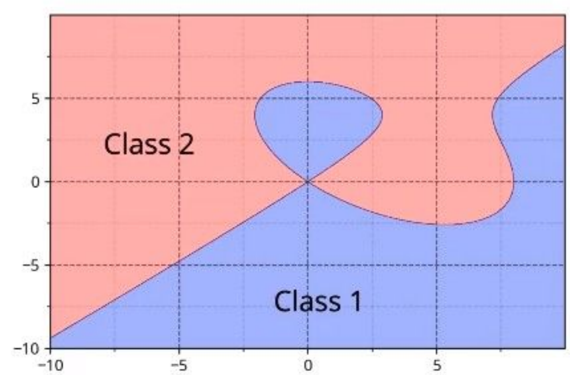
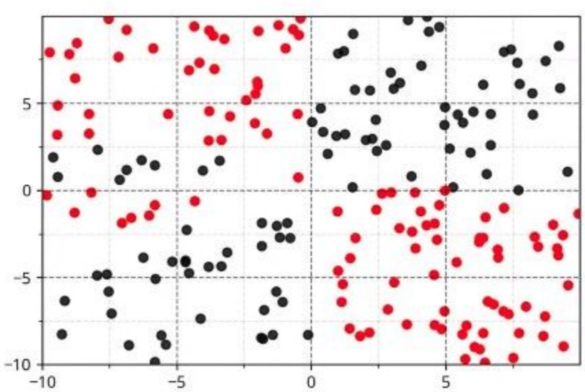
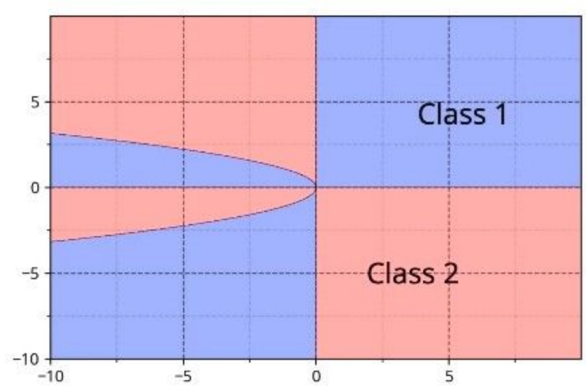
3. $\hat{y} = \hat{f}(x_1, x_2, x_3, \dots x_n)$ or $\hat{income} = \hat{f}(age,\ hour,\ \dots\ education)$
4. Consider predicting whether the person need to be diagnosed with a liver ailment or not

Age	Albumin	T_Bilirubin	…… More features	Diagnosis
65	3.3	0.7		0
62	3.2	10.9		0
20	4	1.1		1
84	3.2	0.7		1

5. Plotting the data gives us

Non Separated	**Coloured for +ve and -ve**
	

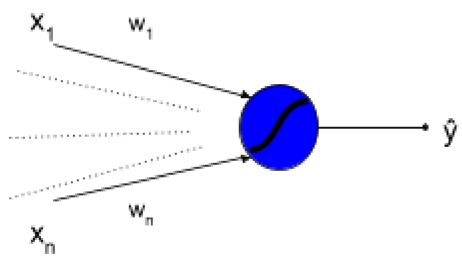
6. $\hat{y} = \hat{f}(x_1, x_2, x_3, \ldots x_n)$ or $\hat{disease} = \hat{f}(age,\ albumin,\ \ldots\ direct\_bilirubin)$
7. Here are a few more examples of complex decision boundaries

Non Separated	**Separated**
	
	

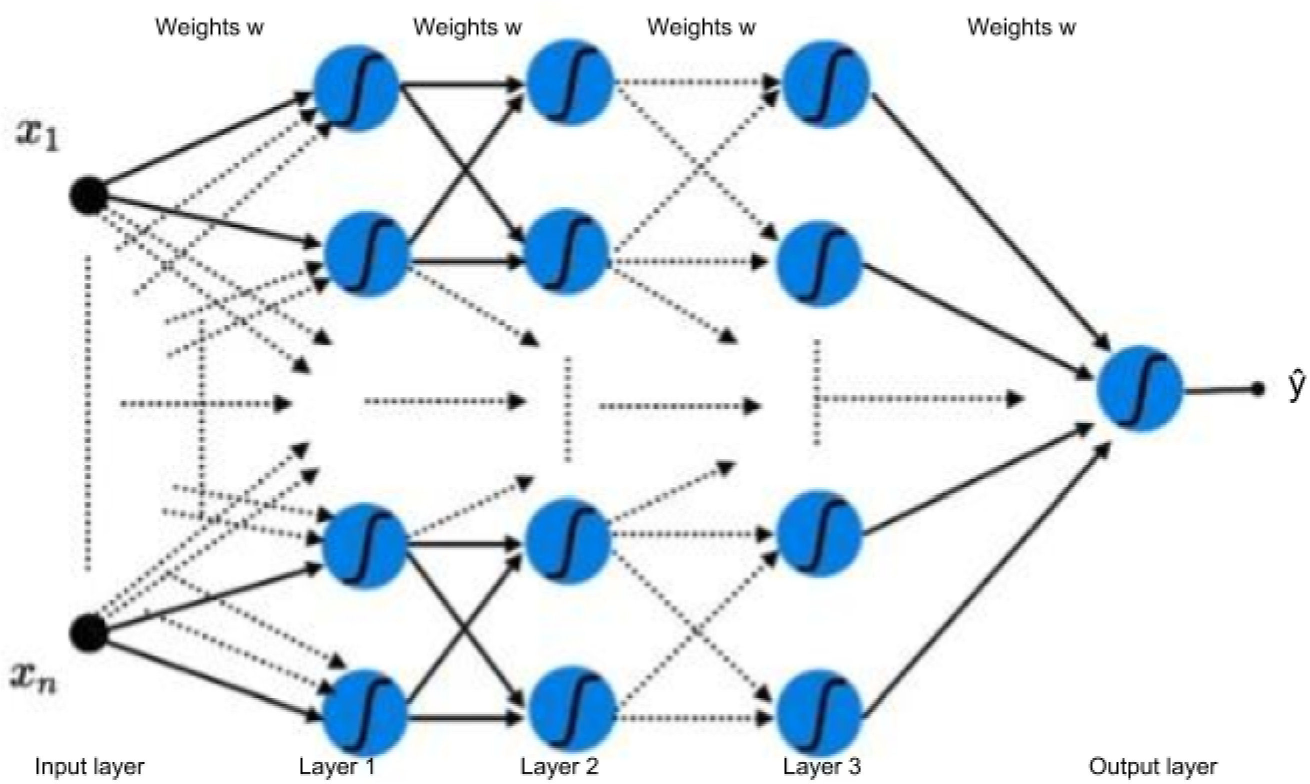
4.1.3: Building complex functions (A simple recipe)

How do we even come up with such functions?

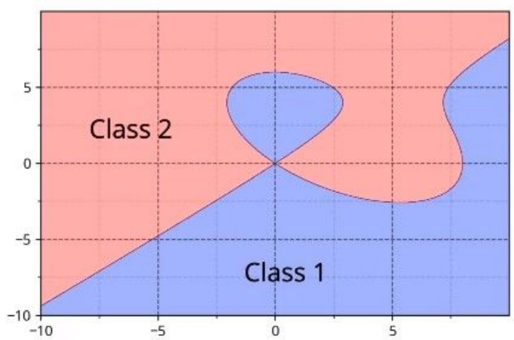
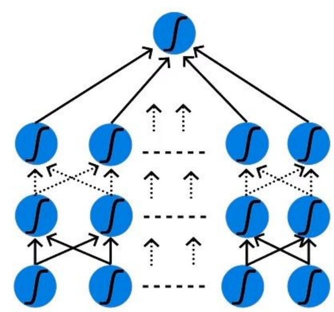
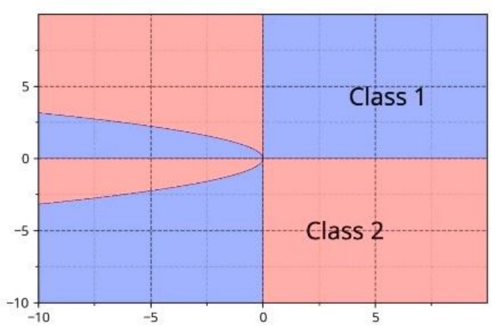
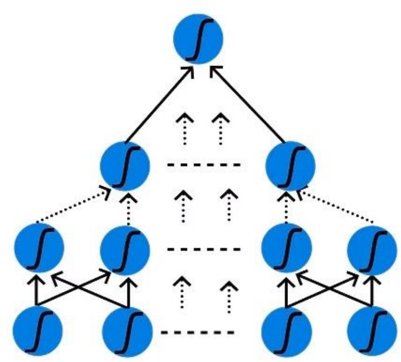
1. $\hat{y} = (\sin wx + \cos wx^3 + e^x + x^{10}) * \frac{1}{\log x}$ is an example of a function that could create a complex decision boundary
2. Clearly, we can see that it's hard to come up with such functions, thus we need a simpler approach
3. Consider the following analogy, to build a house/building, we don't simply conjure the building out of thin air, instead we consider the most basic unit of the building: the brick.
4. The bricks are combined one after the other, in different ways, that ultimately amount to a very complicated structure.
5. In our context, the building would be the complex function and the brick would be a single sigmoid neuron
6. So, here's the brick



7. And here's the building



8. The building that we have constructed with the sigmoid neurons is nothing but a **deep neural network**.
9. Consider a complex function **y’** (read y-prime)
10. The output function of the deep neural network $\hat{y} = f(x_1, x_2, ...x_n)$
11. Regardless of what **y’** we consider, we will be able to approximate it with ŷ by using different configurations of layers and sigmoid neurons.
12. To state this more formally: A deep neural network with a certain number of hidden layers would be able to approximate any function between the input and output
13. This is called the Universal Approximation Theorem **(ŷ≈y’)**
14. Consider the examples from the previous section

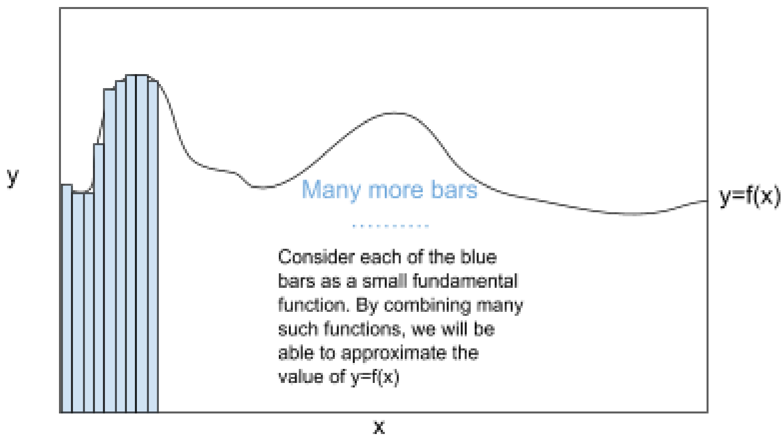
Complex function	Deep Neural Network Representation
	
	

15. With regards to figuring out the DNN configuration for each function, we have to try out different combinations to see what fits best
16. For eg:
    a. Select between 1 - 7 hidden layers
    b. Each hidden layer can have 50, 100 or 150 neurons
    c. Construct several neural networks and select the combination that yields the minimum loss
    d. Thanks to the democratization of models, we have a fairly good idea of the combinations to select based on the task at hand, ie we don’t need to try all possible configurations.

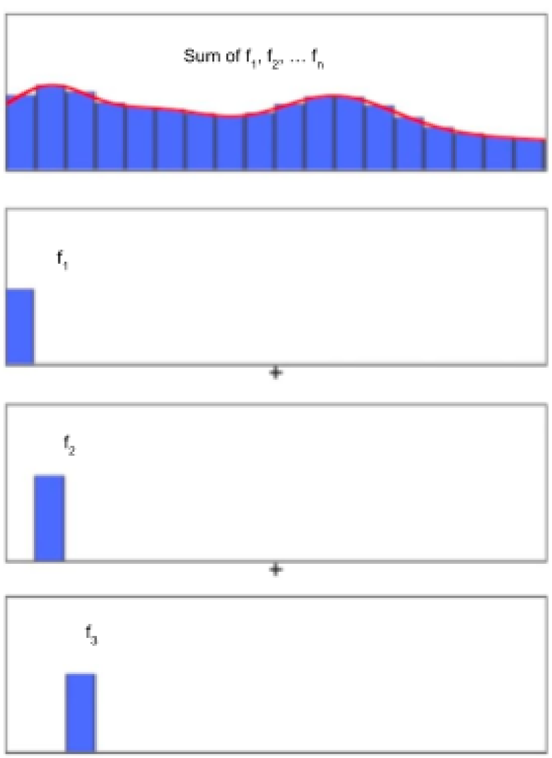
4.1.4: Illustrative proof of Universal Approximation Theorem

The representation power of deep neural networks

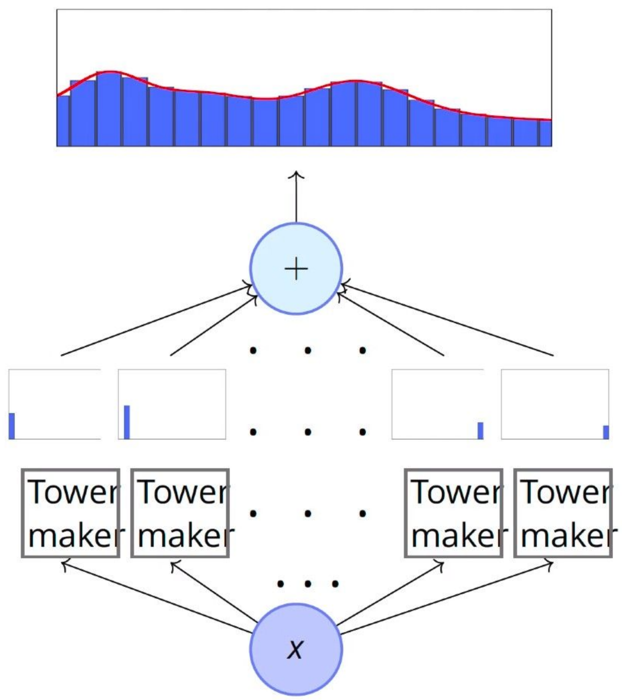
1. Consider the function $y = f(x)$, we want to obtain $\hat{f}(x)$ such that the two functions are are almost equal
2. However, creating a $\hat{f}(x)$ in one go is a daunting task
3. So, we can revisit our old analogy of building with bricks, where we represented a complex function as a combination of simple units
4. Consider the following illustration



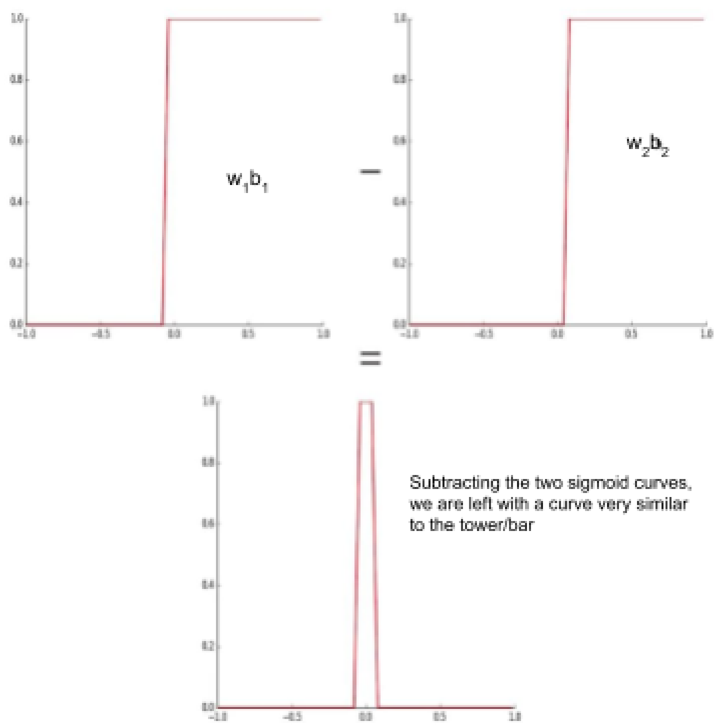
5. Here, the thinner the bar/tower, the better the approximation, because of less wasted space under/over the curve
6. Another illustration



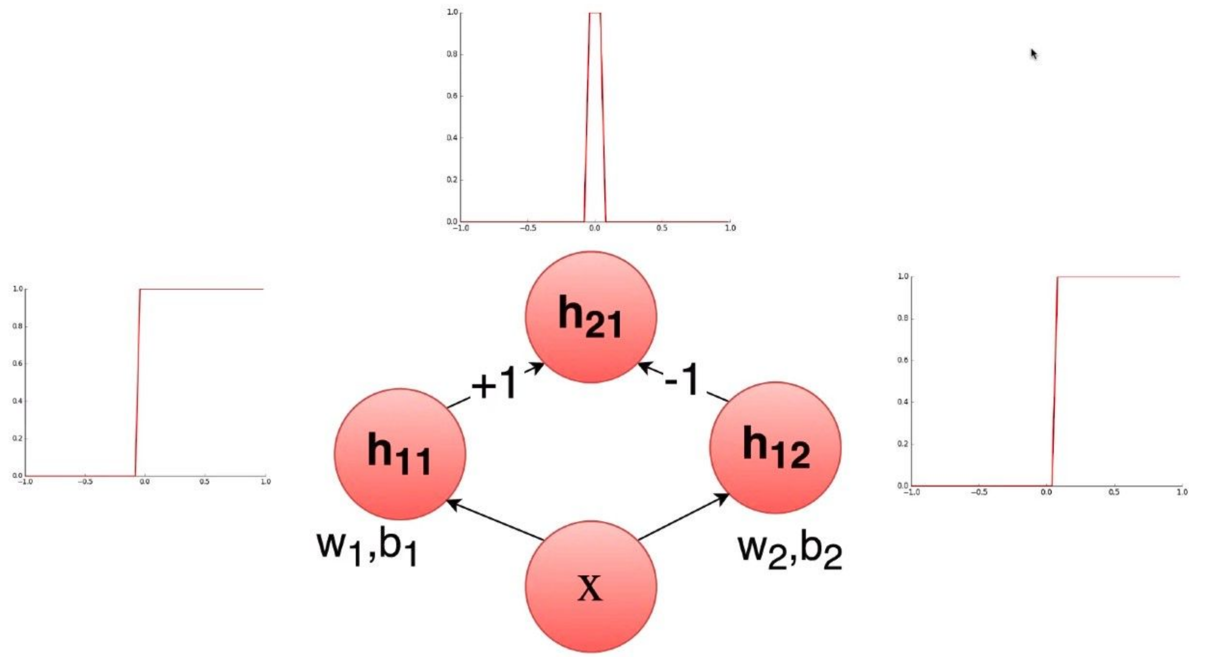
7. How does this tie back the the Sigmoid function
8. Consider the functions required to create these individual towers/bars



9. Let's see how the tower maker function is connected to the sigmoid function
10. In the sigmoid function, w is directly proportional to the sharpness of the curve and b shifts the horizontal position of the threshold. Consider subtraction between two sigmoid functions



11. Neural network representation of sigmoid subtraction



12. With a network of many neurons, we will be able to create several towers/bars. These can then combine to approximate to any kind of function.

4.1.5: Summary

We need to start dealing with complex real-world data and functions

1. Here are some of the take-aways of this chapter
    a. Data: Real inputs
    b. Task: non-linear Classification
    c. Model: Deep Neural Network
    d. Loss: $\Sigma_i(y_i - \hat{y}_i)^2$
    e. Learning: $w = w + \eta\frac{\partial L}{\partial w}$ and $b = b + \eta\frac{\partial L}{\partial b}$
    f. Evaluation: Accuracy = $\frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ predictions}$