PadhAI: Representation Power of Functions

One Fourth Labs

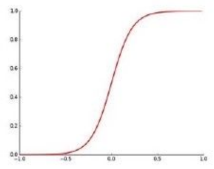
Why do we need complex functions

The need for complex functions

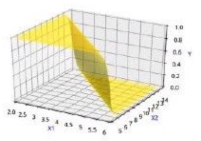
1. Here's a quick recap on what we've covered so far

	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 = $\Sigma_i (y_i != \hat{y}_i)$	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

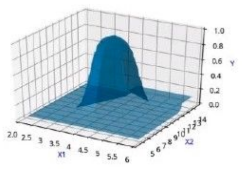
2. We must remember that none of the above 3 models can handle non-linearly separable data
3. Here's another recap on Continuous Functions



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

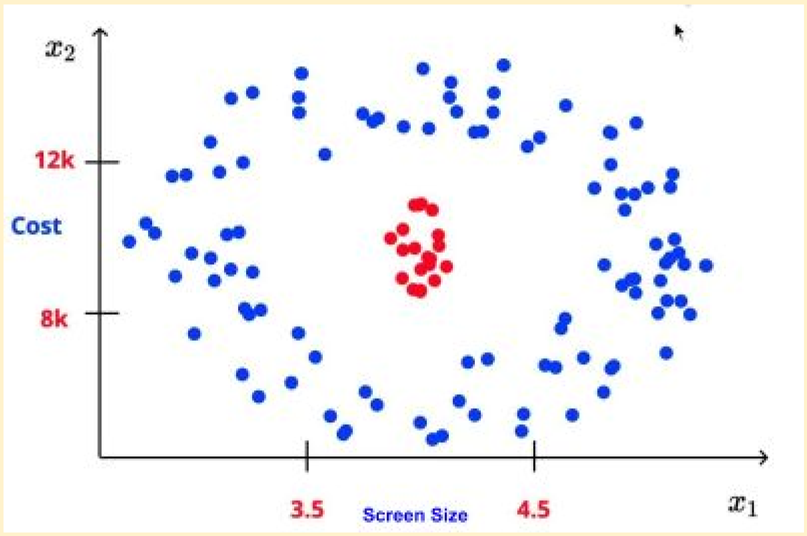


$$\hat{y} = \frac{1}{1+e^{-(-2*x_1+2*x_2+20)}}$$

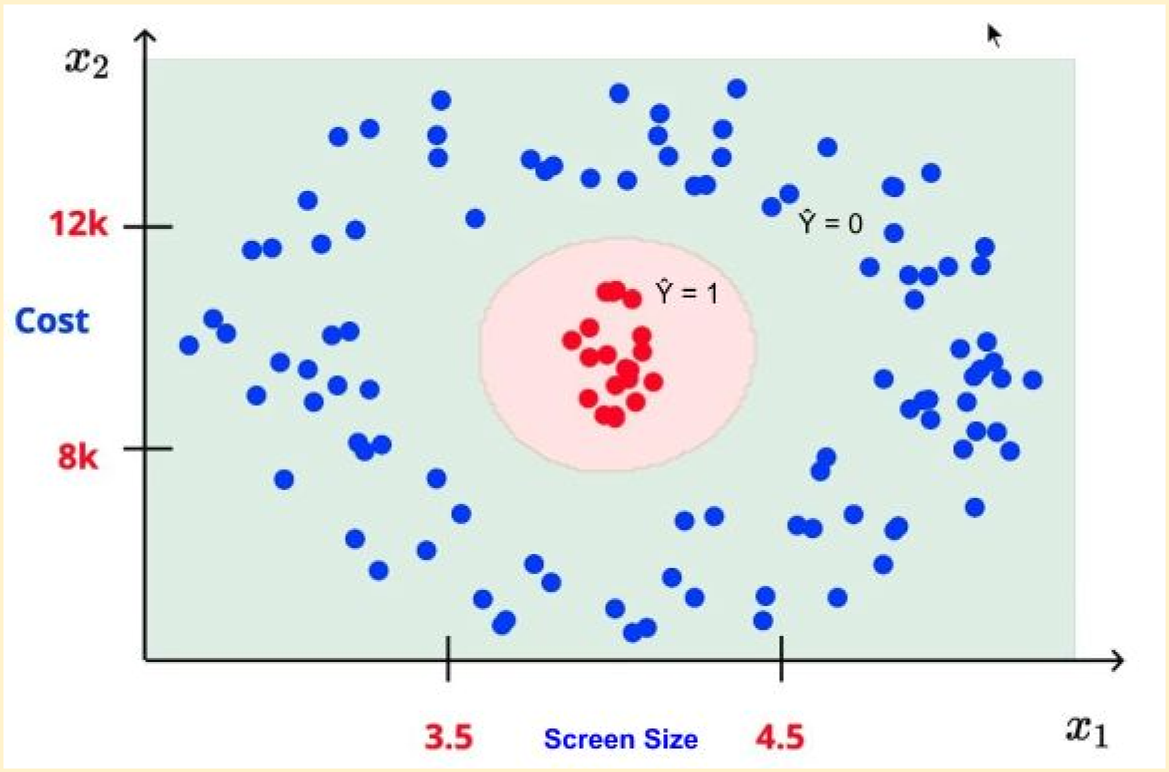


$$\hat{y} = sig_1(sig_2(x_1, x_2), sig_3(x_1, x_2), sig_4(x_1, x_2))$$

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. 

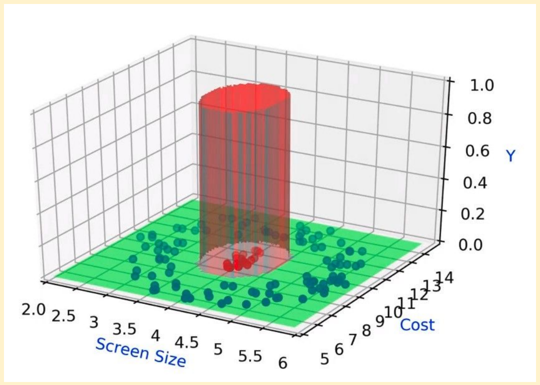
8. Here, our desirable set of phones lies in the centre of a circle of non-desirable phones, based on the values of the variable Cost and Screen Size.
9. Ideally, we would need a decision boundary like so

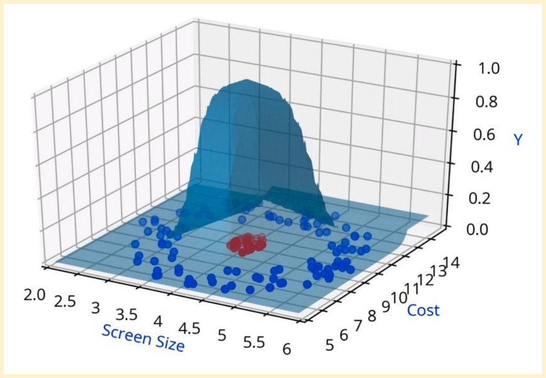


10. However, none of the functions we have seen so far will be able to plot such a decision boundary (ie boundary that separates the two classes = 0 and ŷ = 1)

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

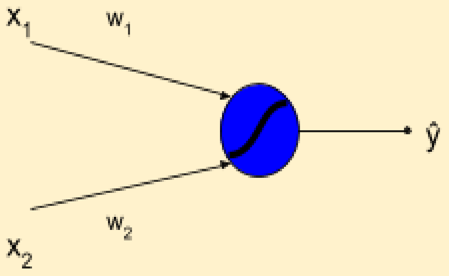
Discrete (abrupt) **Continuous (smooth)**

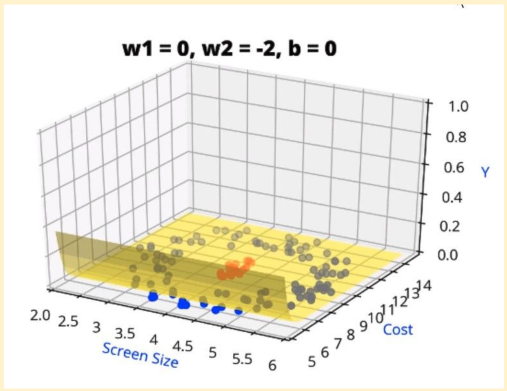


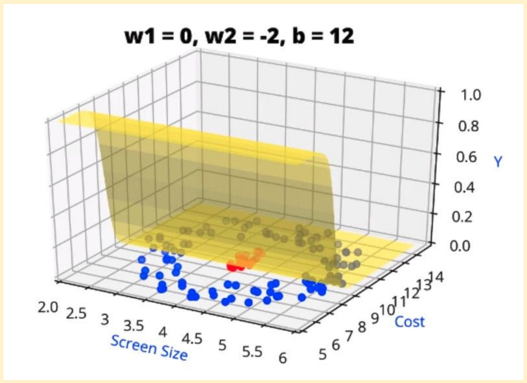


12. Here, the Continuous function has a smooth distribution, and the Y value gradually increases as we converge to the centre, becoming 1 at the region around the red dots
13. However, such an output is not possible with the sigmoid functions, regardless of how we manipulate the values of w and b

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

Complex functions in the real world

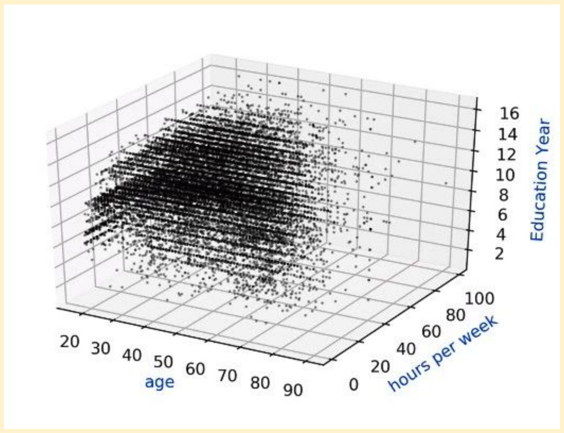
Are such complex functions seen in most real world examples?

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

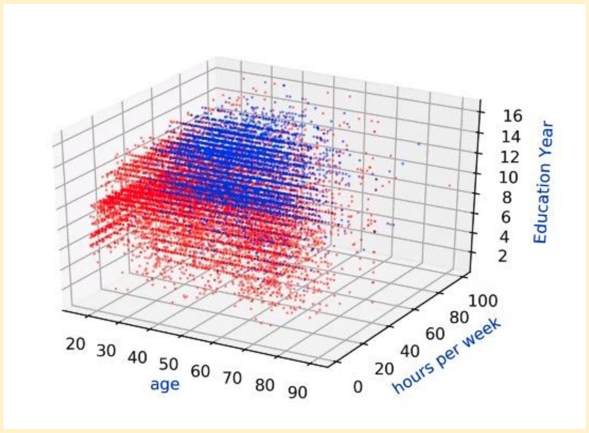
Age	hour/week	Education year	…… More features	Income
90	40	9		0
54	40	4		0
74	20	16		1
45	35	16		1

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

Non Separated



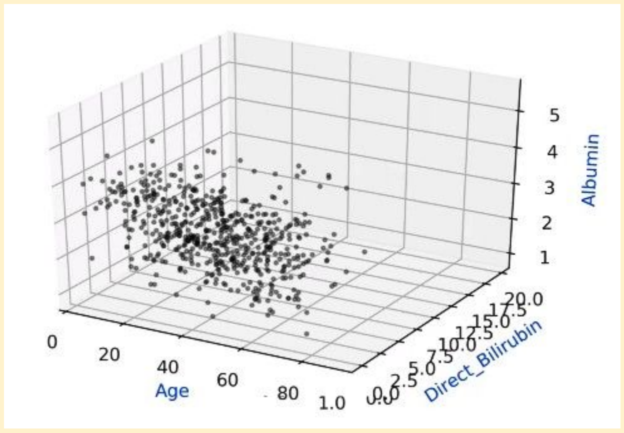
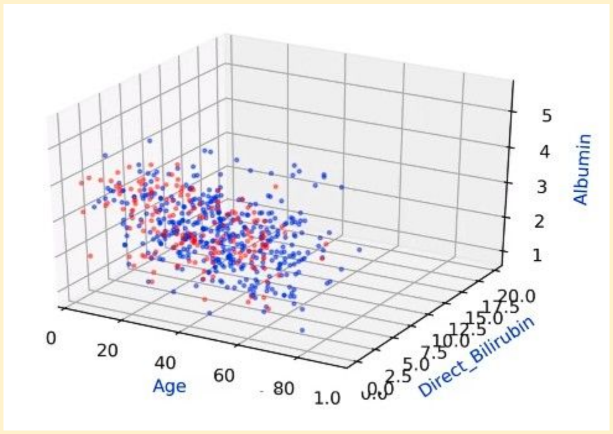
Coloured for +ve and -ve



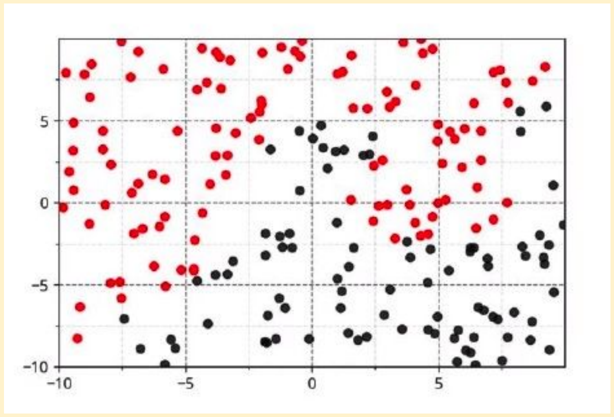
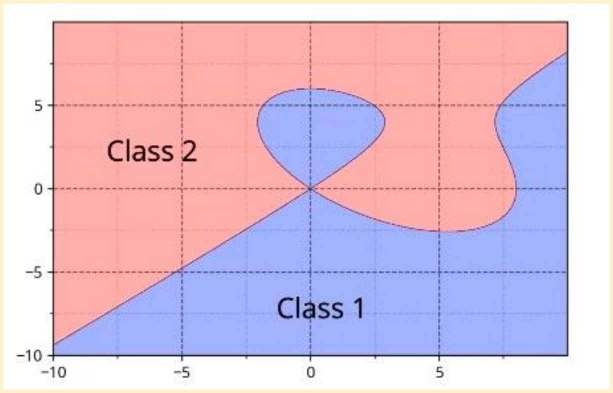
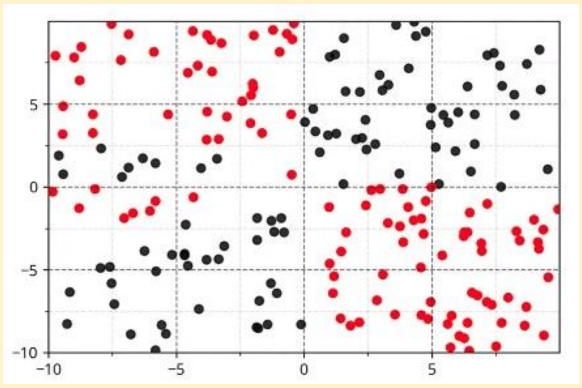
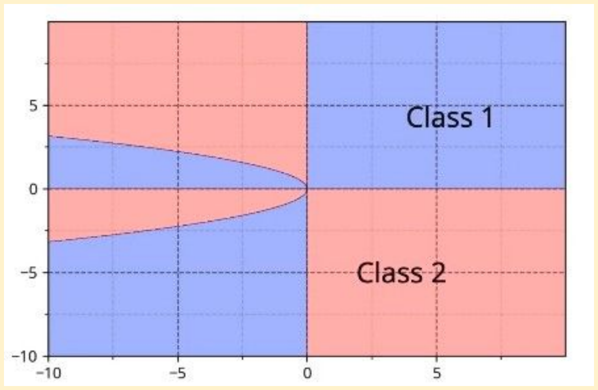
3. $\hat{y} = \hat{f}(x_1, x_2, x_3, \ldots x_n)$ or $\hat{income} = \hat{f}(age,\ hour,\ \ldots\ education)$
4. Consider predicting whether the person need to be diagnosed with a liver ailment or not

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**
	

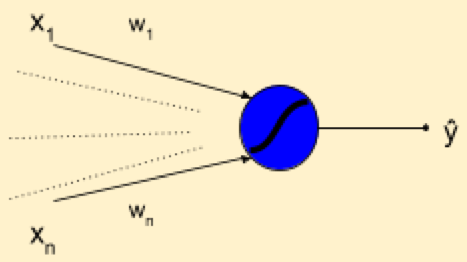
6. $\hat{y} = \hat{f}(x_1, x_2, x_3, ...x_n)$ or $\hat{disease} = \hat{f}(age,\ albumin,\ ...\ direct\_bilirubin)$
7. Here are a few more examples of complex decision boundaries

Non Separated	**Separated**
	
	

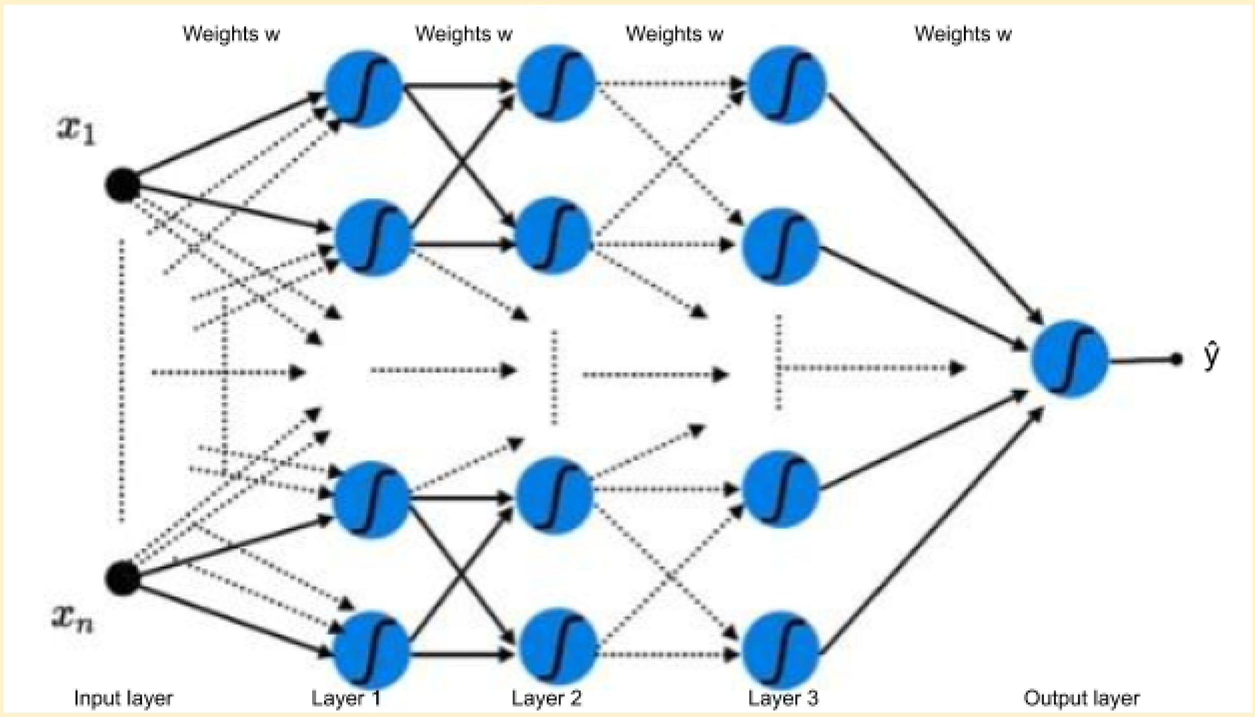
Building complex functions (A simple recipe)

How do we even come up with such functions?

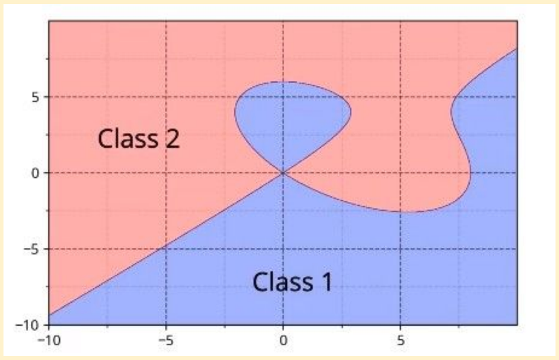
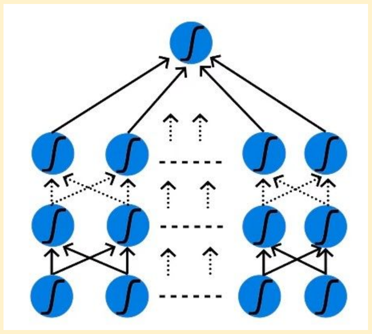
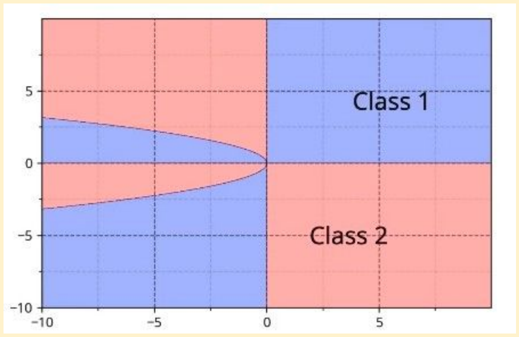
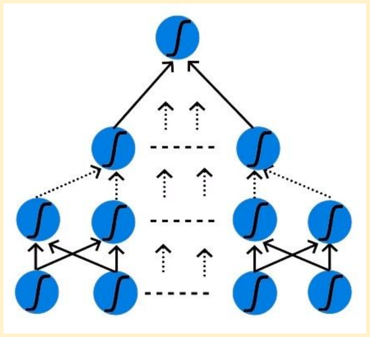
1. $\hat{y} = (\sin wx + \cos wx^3 + e^x + x^{10}) * \frac{1}{\log x}$ is an example of a function that could create a complex decision boundary
2. Clearly, we can see that it's hard to come up with such functions, thus we need a simpler approach
3. Consider the following analogy, to build a house/building, 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

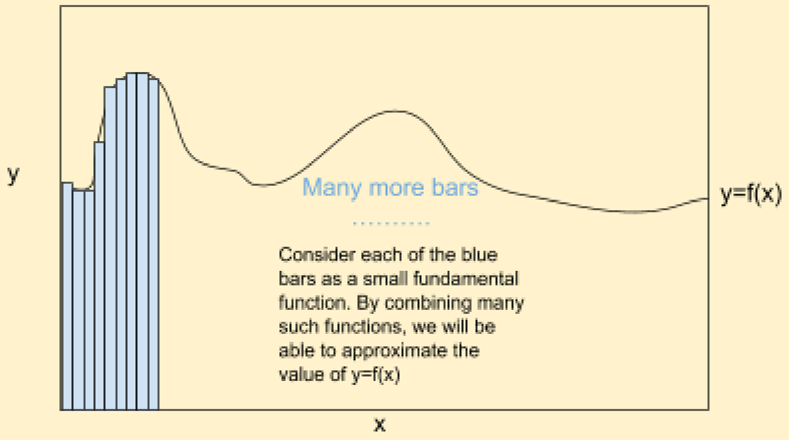
Complex function	**Deep Neural Network Representation**
	
	

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.

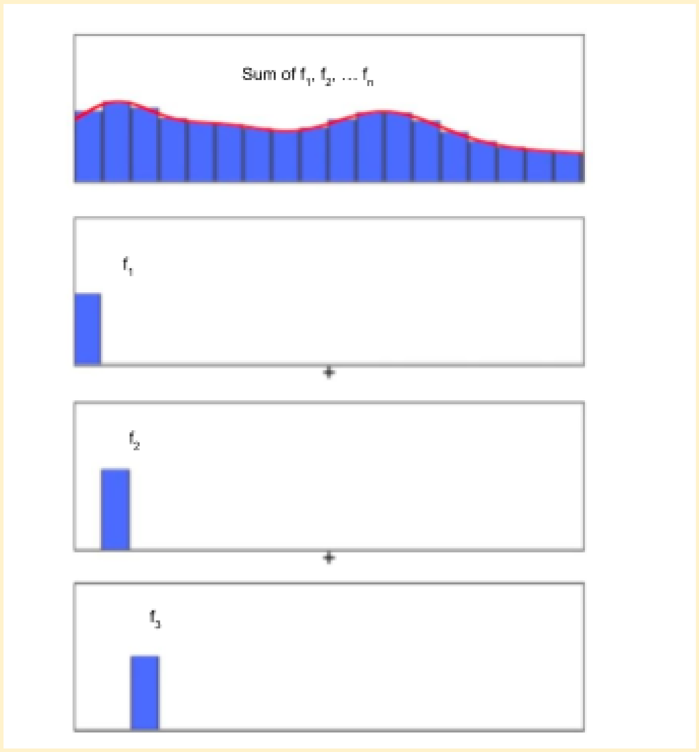
Illustrative proof of Universal Approximation Theorem

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

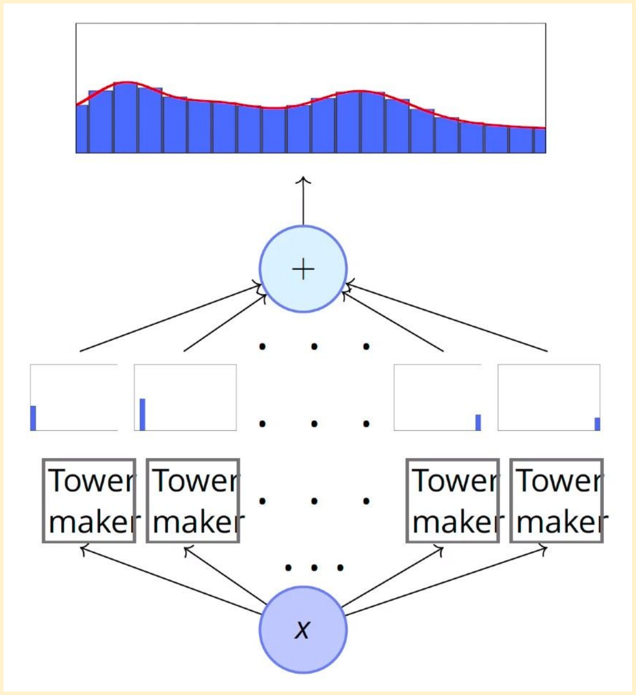

y
y=f(x)
x
Many more bars
Consider each of the blue bars as a small fundamental function. By combining many such functions, we will be able to approximate the value of y=f(x)

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

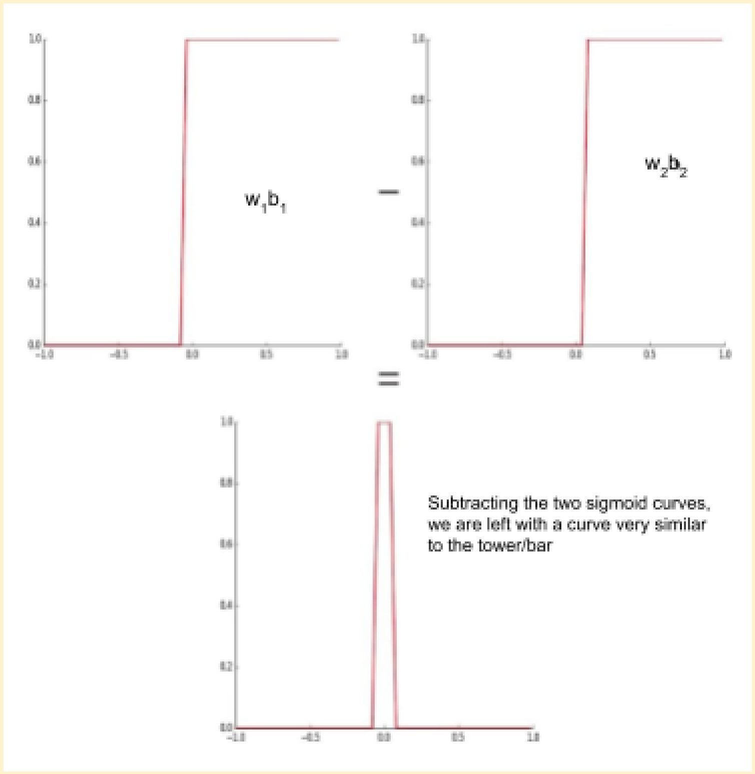


7. How does this tie back to the Sigmoid function

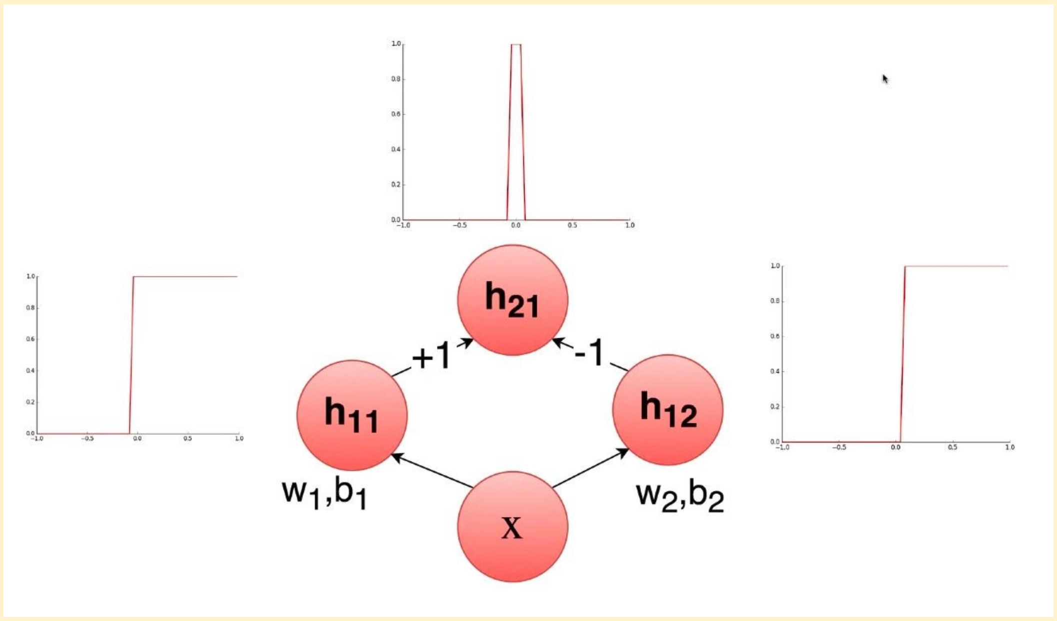
8. Consider the functions required to create these individual towers/bars



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.

Summary

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

1. Here are some of the takeaways 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 = $\frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ predictions}$

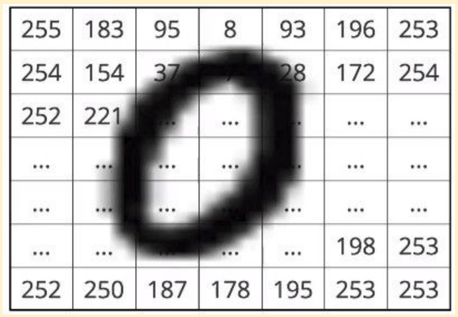
Deep Neural Networks

Data and Tasks

What kind of data and tasks have DNNs been used for?

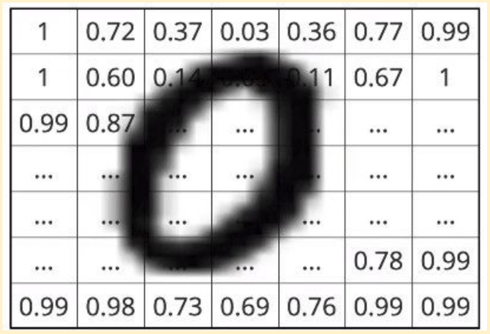
1. One common example is digit classification using the MNIST dataset

   a. MNIST images are vectorized using the pixel values of each cell
   b. Matrix having pixel values will be of size 28x28 (as MNIST images are of the size 28x28)



255	183	95	8	93	196	253
254	154	37	[illegible]	28	172	254
252	221	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	198	253
252	250	187	178	195	253	253

   c. Each pixel can range from 0 to 255. Standardise pixel values by dividing with 255

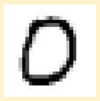
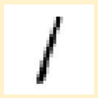
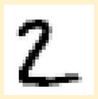
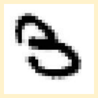
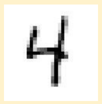
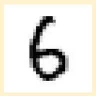
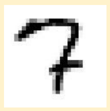
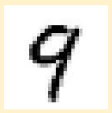


1	0.72	0.37	0.03	0.36	0.77	0.99
1	0.60	0.14	[illegible]	0.11	0.67	1
0.99	0.87	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	0.78	0.99
0.99	0.98	0.73	0.69	0.76	0.99	0.99

   d. Now, flatten the matrix to convert into a vector of 784 (28x28)

2. Convert all images to vectors of order $\mathbb{R}^{784}$

3. Let's look at the data along with the labels (Multi-Class Classification)

28x28 images	Vectorized form	Class Label	Class Labels - One Hot Representation
	[1.00, 0.72, ... 0.99]	0	[1,0,0,0,0,0,0,0,0,0]
	[1.00, 0.85, ... 1.00]	1	[0,1,0,0,0,0,0,0,0,0]
	[1.00, 0.76, ... 1.00]	2	[0,0,1,0,0,0,0,0,0,0]
	[0.99, 0.82, ... 1.00]	3	[0,0,0,1,0,0,0,0,0,0]
	[0.73, 0.81, ... 0.67]	4	[0,0,0,0,1,0,0,0,0,0]
	[1.00, 1.00, ... 0.99]	5	[0,0,0,0,0,1,0,0,0,0]
	[0.84, 0.72, ... 0.99]	6	[0,0,0,0,0,0,1,0,0,0]
	[0.33, 0.52, ... 1.00]	7	[0,0,0,0,0,0,0,1,0,0]
	[0.85, 0.72, ... 0.99]	8	[0,0,0,0,0,0,0,0,1,0]
	[0.84, 0.92, ... 0.99]	9	[0,0,0,0,0,0,0,0,0,1]

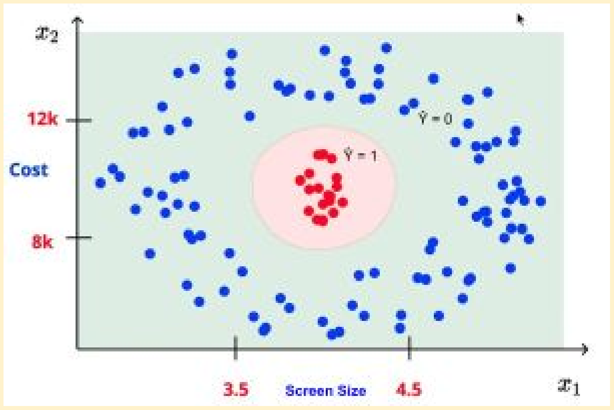
4. Another example would be the Indian Liver Patient classification problem. There are only two possible outcomes, hence it is a Binary-Class classification task
5. An example for regression would be Housing Price Prediction, where instead of predicting a discrete output, the prediction is a real-number or continuous value (decimals, fractions etc)

Model

A Simple Deep Neural Network

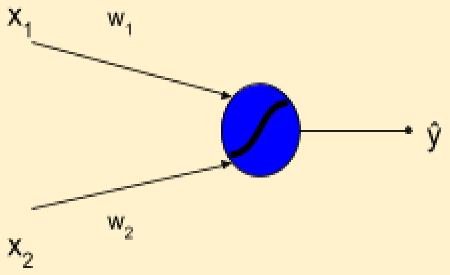
How to build complex functions using Deep Neural Networks

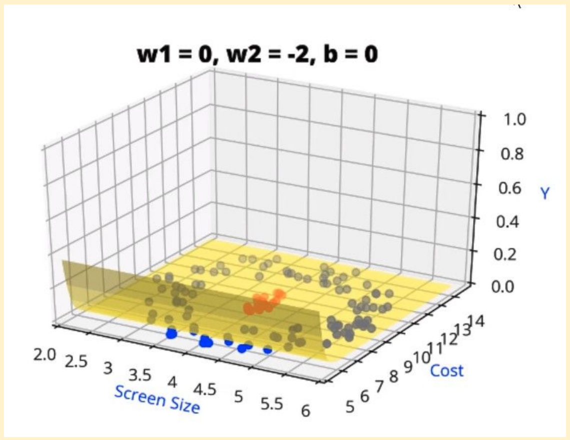
1. Consider the previously used example of mobile phone like/dislike predictor with the variables Screen-size and Cost. It has a complex decision boundary as shown here

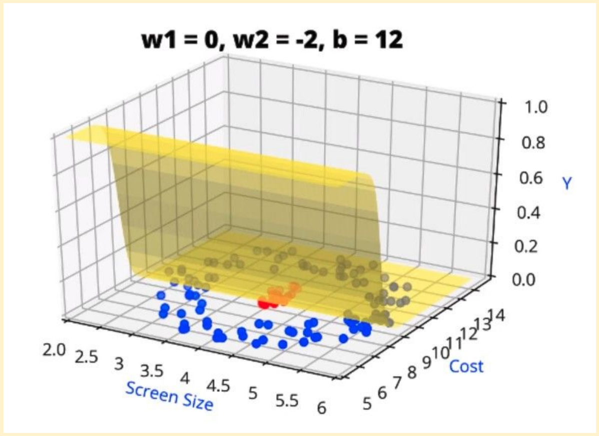


2. With a single sigmoid neuron, it is impossible to obtain this shape, regardless of how we vary the parameters w & b, as the sigmoid neuron can only produce a shape ranging from s-shaped to flat. The formula is $\hat{y} = f(x_1, x_2)$ or $\hat{y} = \frac{1}{1 + e^{-(w_1 * x_1 + w_2 * x_2 + b)}}$

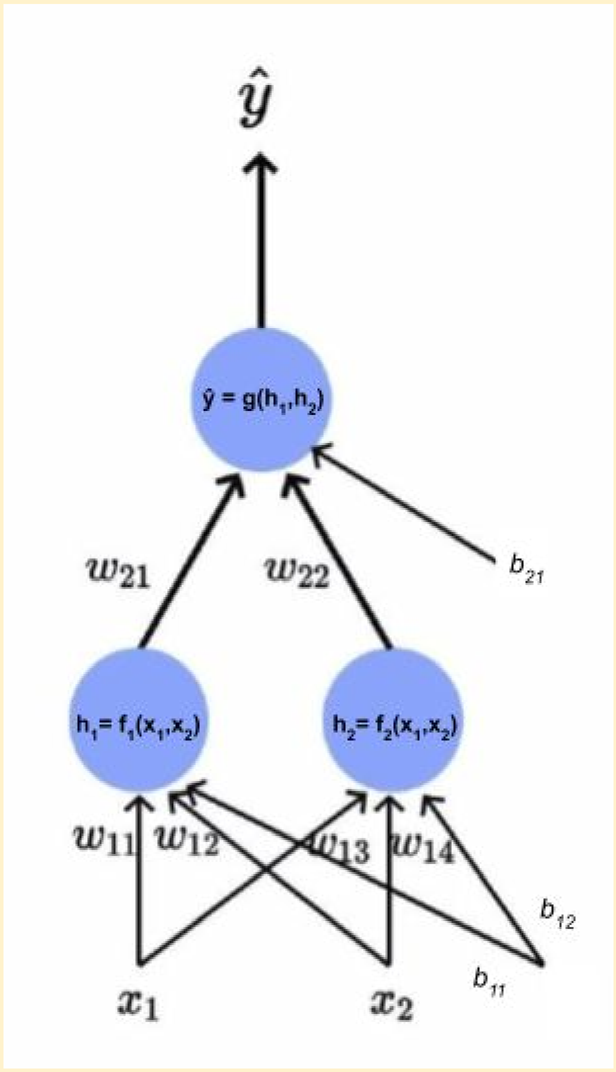
Sigmoid decision boundary, can range from s-shaped to flat, based on w and b values







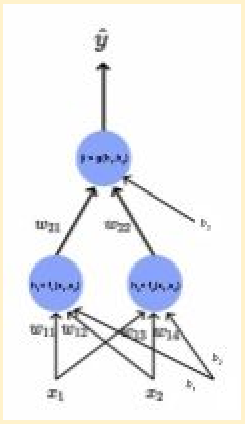
3. Now, let us consider a Deep Neural Network for the same mobile phone like/dislike predictor

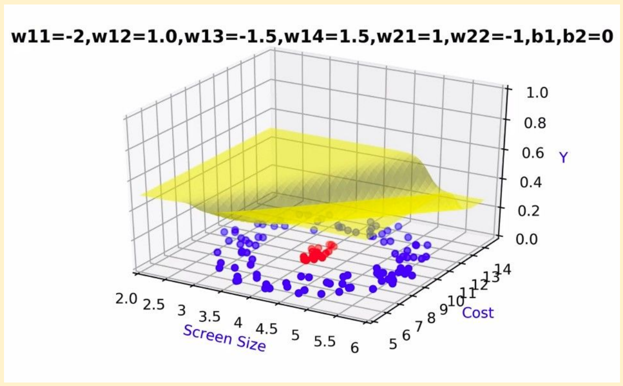


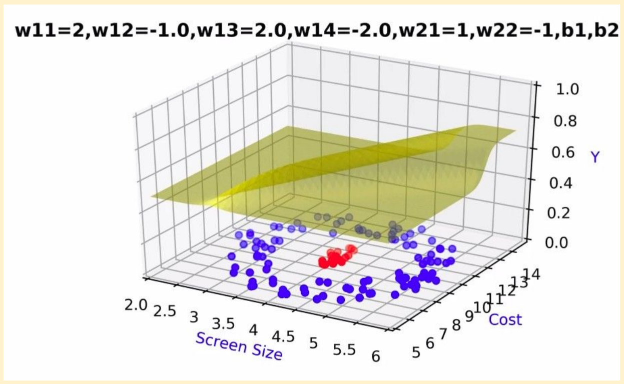
4. Breaking down the model:
   a. x_1 = Screen-Size, x_2 = Cost
   b. First Neuron $h_1 = f_1(x_1,x_2)$ or $h_1 = \frac{1}{1 + e^{-(w_{11}*x_1 + w_{12}*x_2 + b_1)}}$
      i. Here, w_{11} and w_{12} are the weights of x_1 and x_2 corresponding to the first neuron h_1
      ii. b_{11} is the corresponding bias
   c. Second Neuron $h_2 = f_2(x_1,x_2)$ or $h_1 = \frac{1}{1 + e^{-(w_{13}*x_1 + w_{14}*x_2 + b_2)}}$
      i. Here, w_{13} and w_{14} are the weights of x_1 and x_2 corresponding to the second neuron h_2
      ii. b_{12} is the corresponding bias
   d. Output Neuron $\hat{y} = g(h_1,h_2)$ or $\hat{y} = \frac{1}{1+e^{-(w_{21}*(\frac{1}{1 + e^{-(w_{11}*x_1 + w_{12}*x_2 + b)}}) + w_{22}*(\frac{1}{1 + e^{-(w_{13}*x_1 + w_{14}*x_2 + b)}}) + b_3)}}$
      i. Here, w_{21} and w_{22} are the weights of h_1 and h_2 corresponding to the output neuron $\hat{y}$
      ii. b_{21} is the corresponding bias
   e. From this configuration, we have 9 parameters (w_{11}, w_{12}, w_{13}, w_{14}, w_{21}, w_{22}, b_1, b_2, b_3), which allow for a much more complex decision boundary than a single sigmoid neuron with 3 parameters

5. The output would look something like this

Deep Neural Network Decision Boundary, more complex than a single sigmoid neuron.

6. 





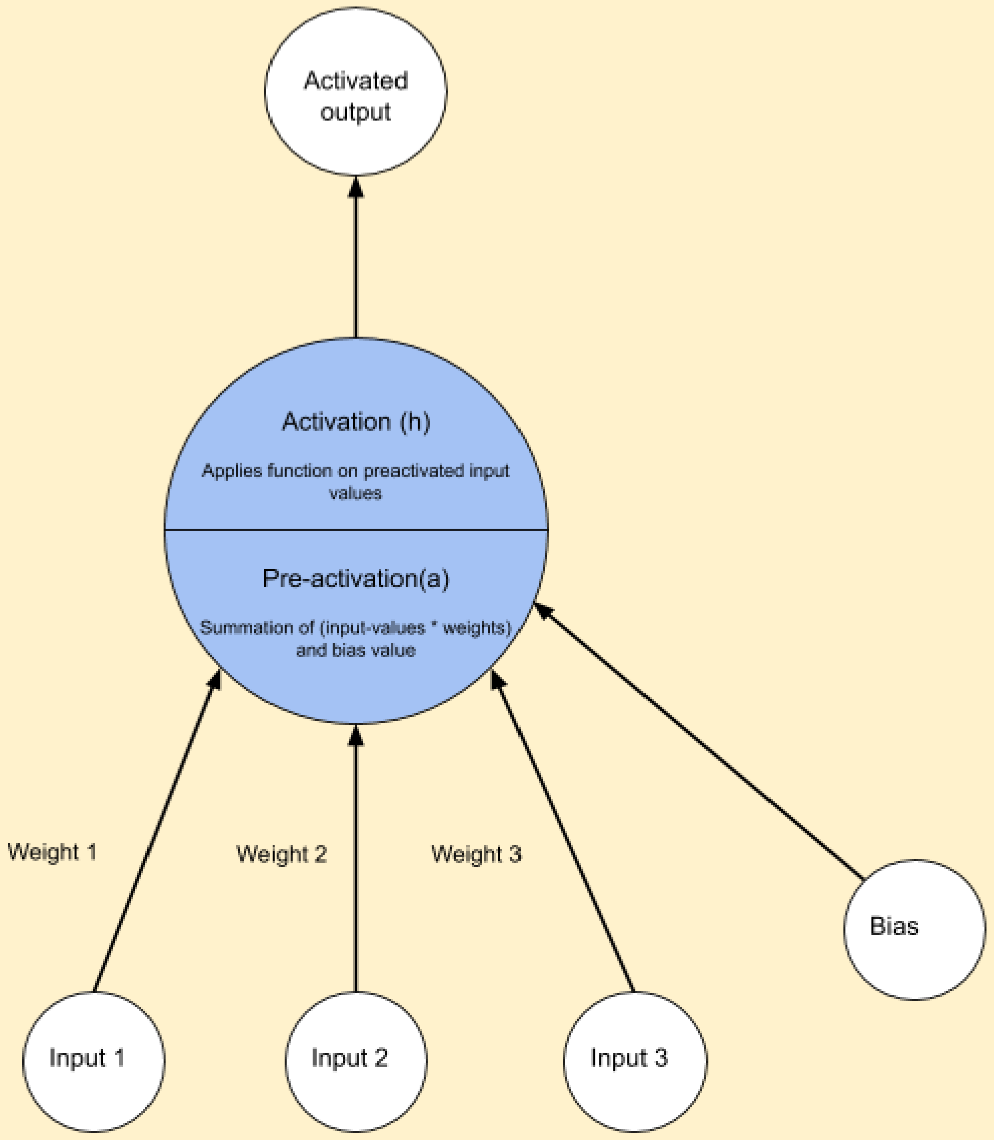
* This simple neural network already allows for a much better decision boundary than with a single sigmoid neuron

7. The next step would be figuring out how to choose the best configuration of the DNN for our task, this is called **Hyperparameter Tuning.**
8. For now, we can rest easy knowing that by the **Universal Approximation Theorem** we will be able to approximate any kind of function with our Neural Network

A Generic Deep Neural Network

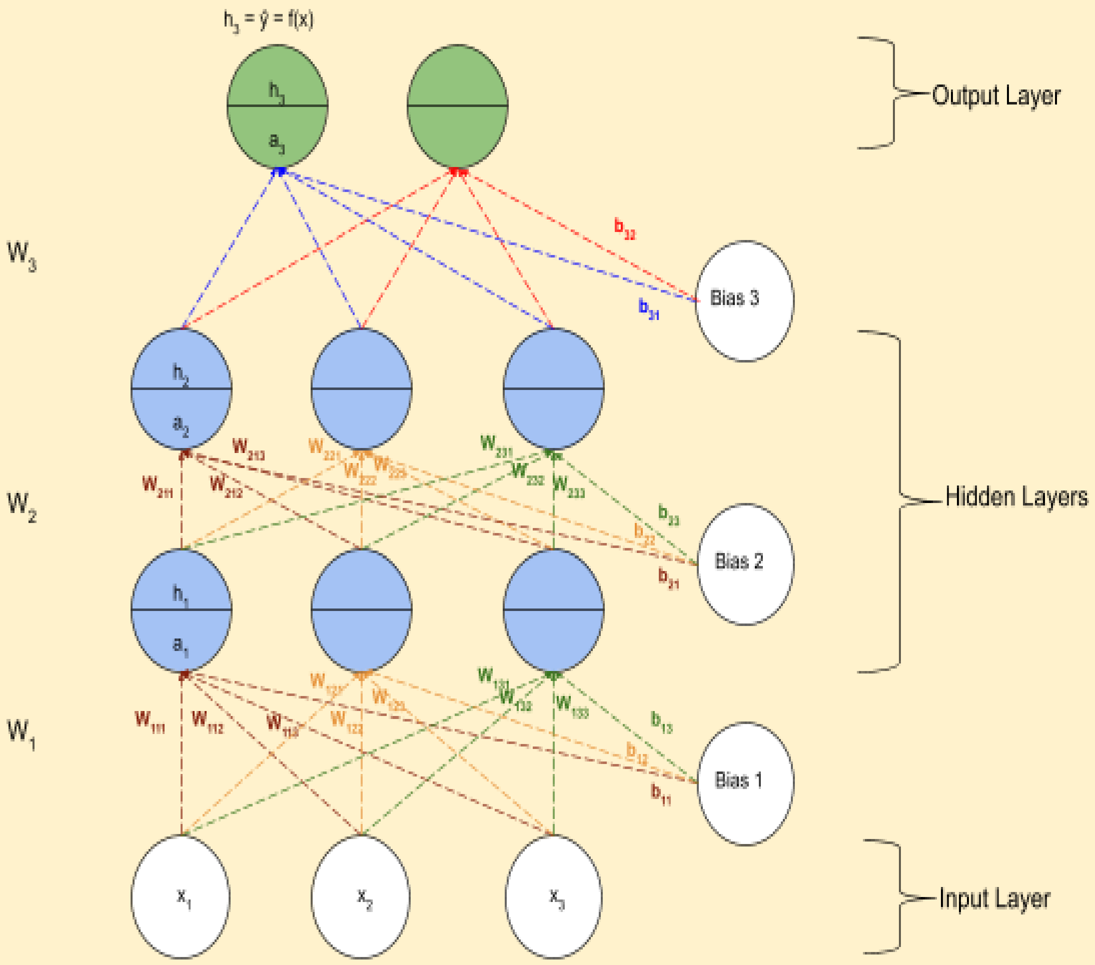
Can we clarify the terminology used?

1. Let us revisit the structure of a neuron


Activated output
Activation (h)
Applies function on preactivated input values
Pre-activation(a)
Summation of (input-values * weights) and bias value
Weight 1
Weight 2
Weight 3
Bias
Input 1
Input 2
Input 3

2. Let us break down the terms
   a. Let i refer to the layer being referenced
   b. **Pre-activation function** $a_i = \Sigma(input * weights) \ + bias$
   c. **Activation function** $h_i = \frac{1}{1+e^{-(a_i)}}$ a
   d. Here, the activation function is the sigmoid function.
   e. The construction of a Neural network is a simple stacking of these neurons in layers, one on top of the other
   f. The outputs of one layer of neurons become the inputs for the next layer.
   g. The cycle of pre-activation and activation repeats itself from the input layer, till we reach the output layer and obtain the desired function

3. Let us break down the structure of a Neural Network

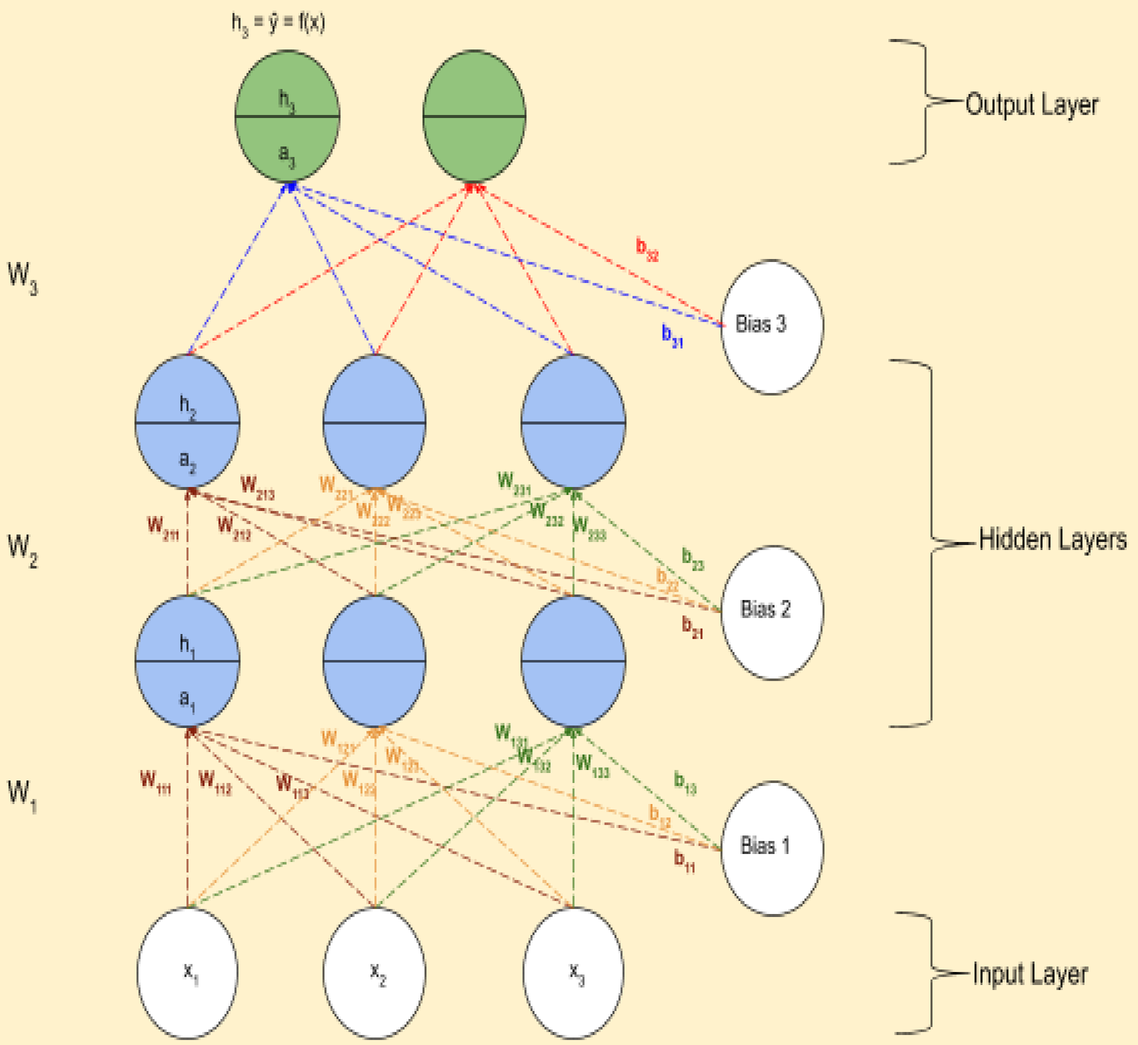


4. Let's break down some of the terms used:
   a. The format of w is $W_{\text{(Layer number)(Next layer Neuron)(Current Layer Input/neuron)}}$
      i. So W_{213} refers to the weight corresponding to the 3rd input on 1st neuron of the 2nd hidden layer
   b. For each layer i where 0 <= i <= L
      i. Pre-activation $a_i(x) = W_i h_{i-1}(x) + b_i$
      ii. Activation $h_i(x) = g(a_i(x))$ where 'g' is called the activation function
      iii. Activation at output layer L is $f(x) = h_L = O(a_L)$ where 'O' is called the output activation function

Understanding the Computations in a Deep Neural Network

Let's look at the computations inside a DNN

1. Consider the same DNN drawn in the previous section



2. The preactivation outputs for the first layer a_{11}, a_{12} a_{13}, are calculated using simple Matrix-vector multiplication

$$W_1 = \begin{bmatrix} w_{111} & w_{112} & w_{113} \\ w_{121} & w_{122} & w_{123} \\ w_{131} & w_{132} & w_{133} \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

3. Here, the preactivation values are as follows
   a. $a_{11} = w_{111} * x_1 + w_{112} * x_2 + w_{113} * x_3 + b_{11}$
   b. $a_{12} = w_{121} * x_1 + w_{122} * x_2 + w_{123} * x_3 + b_{12}$
   c. $a_{13} = w_{131} * x_1 + w_{132} * x_2 + w_{133} * x_3 + b_{13}$
   d. These values are just the individual rows of the dot-product between W_1 and X plus the bias vector
   e. Thus $W_1X = a_1$ is given by

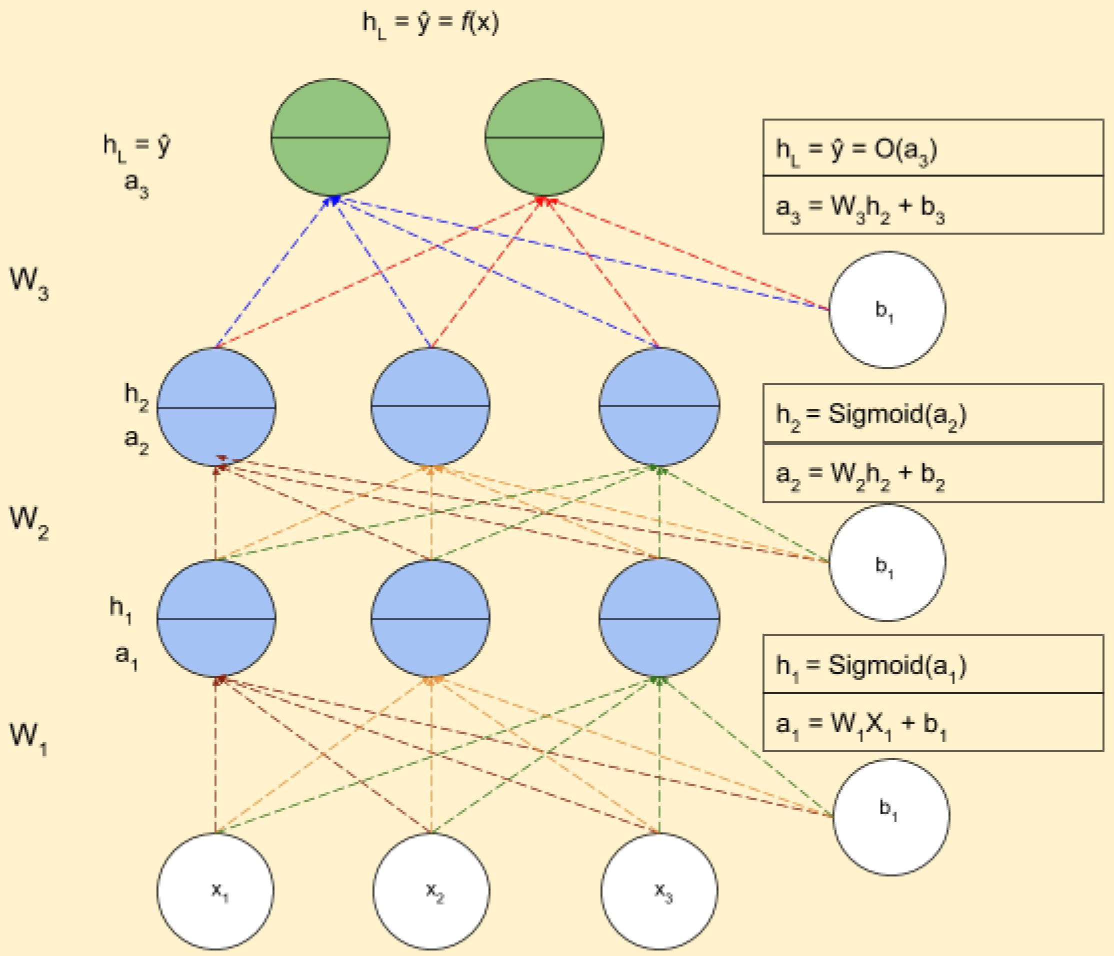
$$a_1 = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix}$$

   f. Here, $W_1 \in \mathbb{R}^{3x3}$, $X \in \mathbb{R}^{3x1}$, and $W_1X \in \mathbb{R}^{3x1}$
   g. $a_i = W_i$
4. The activation values are as follows
   a. $h_i = g(a_i)$
   b. They are simply the result on applying the activation function (in this case: sigmoid) on the preactivated values
   c. $h_{11} = \frac{1}{1+e^{-(a_{11})}}$
   d. $h_{12} = \frac{1}{1+e^{-(a_{12})}}$
   e. $h_{13} = \frac{1}{1+e^{-(a_{13})}}$

The Output Layer of a Deep Neural Network

How do we decide the output layer?

1. Here are the dimensions of our values
   a. m : *number of neurons in a layer*
   b. n : *number of inputs to a layer*
   c. $a_i = W_i x_{i-1} + b_i$
      i. $a_i \in \mathbb{R}^m$
      ii. $W_i \in \mathbb{R}^{m \times n}$
      iii. $x_i \in \mathbb{R}^{n \times 1}$
      iv. $b_i \in \mathbb{R}^m$
   d. $h_i = \sigma(a_i)$
      i. $h_i \in \mathbb{R}^m$
2. Here is a quick sample

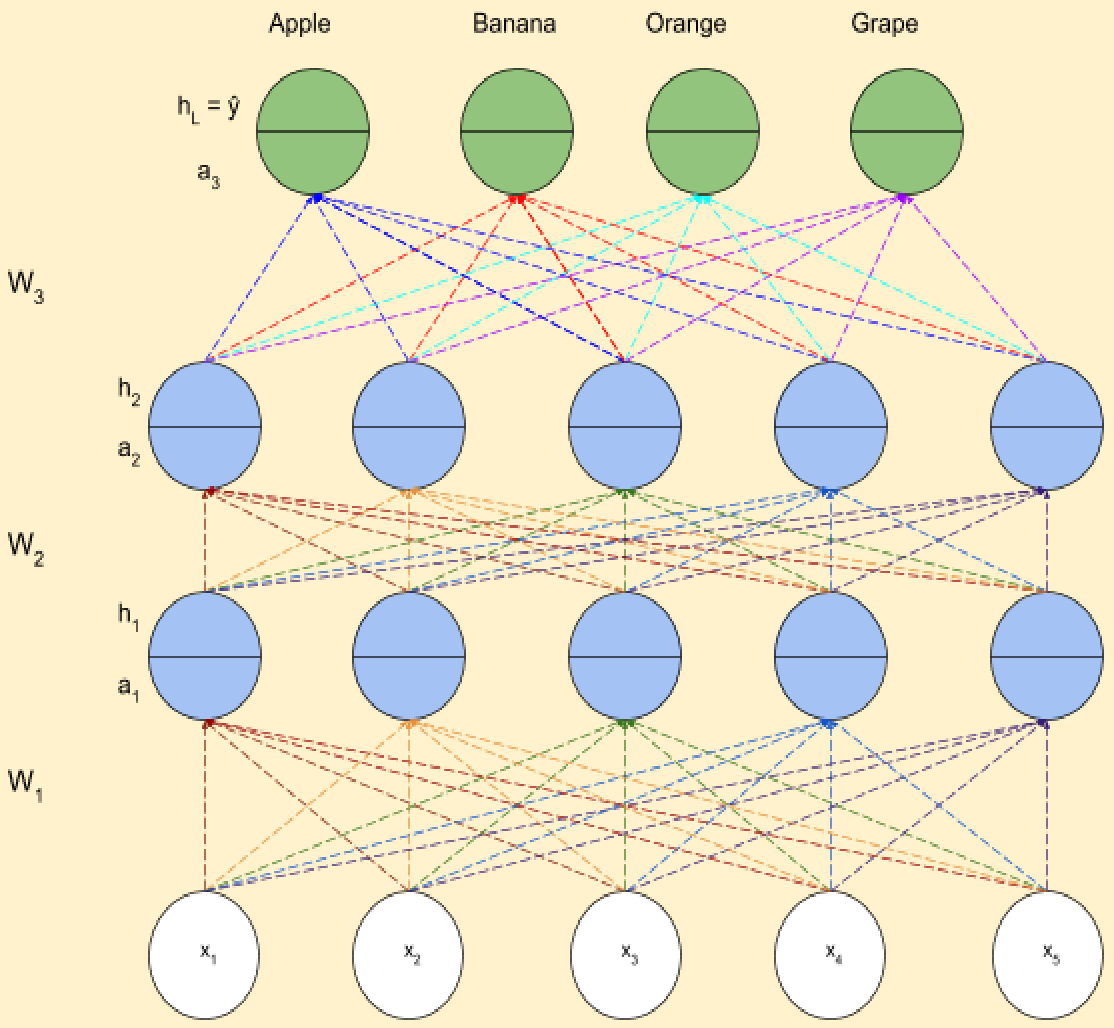


3. Here, it is possible to write the output function ŷ completely in terms of x
4. $\hat{y} = f(x) = O(W_3 g(W_2 g(W_1 x + b_1) + b_2) + b_3)$
5. Here $\hat{y} \in \mathbb{R}^K$ where K is the number of output units

Output Layer of a Multi-Class Classification Problem

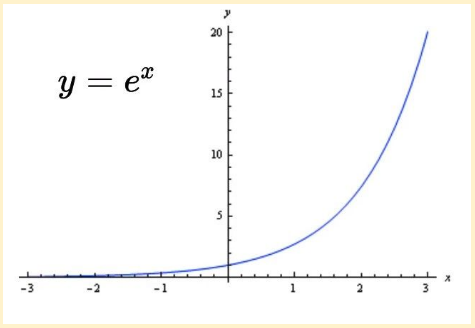
Deciding the output layer

1. The Output Activation function is chosen depending on the task at hand (can be a softmax, linear etc)
2. Consider the following multi-class classification problem



3. At the last layer, we compute $a_3 = W_3h_2 + b_3$
4. We need to apply a function to $\hat{y}_i = O(a_{3i})$ such that the 4 outputs form a probability distribution.
5. Output activation function has to be chosen such that the output is probability.
6. Let's assume $a_3 = [3\ \ 4\ \ 10\ \ 3]$
   a. Take each entry and divide by the sum of all entries to get a probability distribution
   b. $\hat{y_1} = \frac{3}{(3+4+10+3)} = 0.15$
   c. $\hat{y_2} = \frac{4}{(3+4+10+3)} = 0.20$
   d. $\hat{y_3} = \frac{10}{(3+4+10+3)} = 0.50$
   e. $\hat{y_4} = \frac{3}{(3+4+10+3)} = 0.15$
   f. However, **this will not work** if any of the entries are negative

7. So we consider the softmax function
8. $softmax(z_i) = \frac{e^{z_i}}{\Sigma_{j=1}^{k} e^{z_j}}$ for i = 1...k
9. Note: the output of e^x is always positive, irrespective of the input



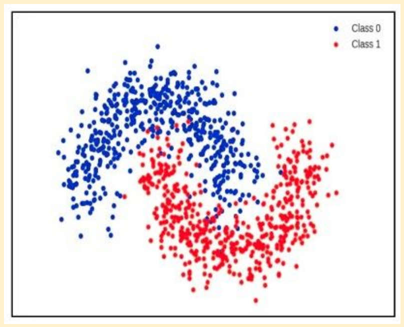
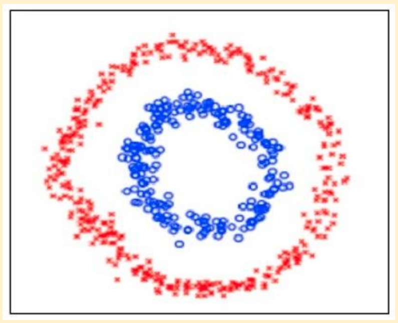
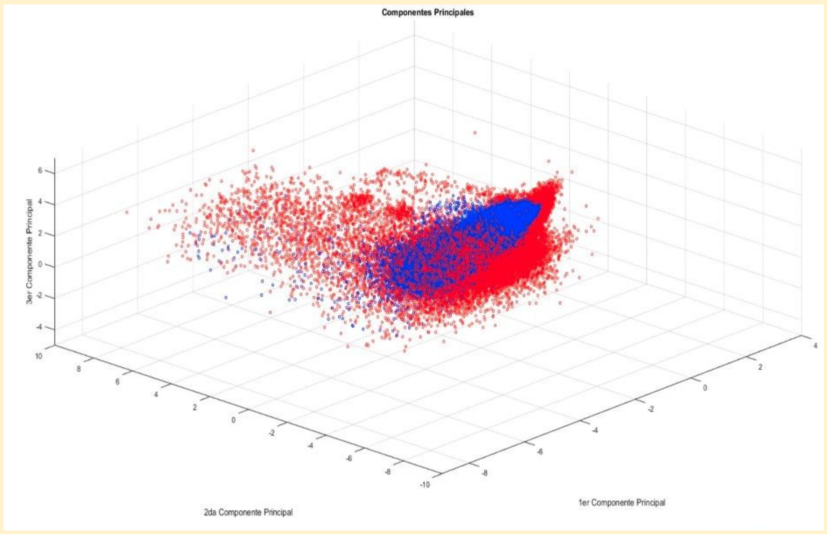
10. This property is important to counter the negative-value shortcoming that we observed in the previous example
11. Now, let us illustrate the softmax function at the last layer of our Neural Network
12. $a = [a_1\ a_2\ a_3\ a_4]$
13. $softmax(a) = [\frac{e^{a_1}}{\Sigma_{j=1}^{k} e^{a_j}}\ \frac{e^{a_2}}{\Sigma_{j=1}^{k} e^{a_j}}\ \frac{e^{a_3}}{\Sigma_{j=1}^{k} e^{a_j}}\ \frac{e^{a_4}}{\Sigma_{j=1}^{k} e^{a_j}}]$
14. Raising the numerators to e^x ensures that they are all positive
15. The denominator is just the sum of all the values raised to e^x
16. $softmax(a_i)$ *is the* i^{th} *element of the softmax output*
17. So for our multi-class fruit classifier, the equations are as follows

Layer	**Pre-activation**	**Activation/Output**
Hidden Layer 1	$a_1 = W_1 * x + b_1$	$h_1 = g(a_1)$
Hidden Layer 2	$a_2 = W_2 * h_1 + b_2$	$h_2 = g(a_2)$
Output Layer	$a_3 = W_3 * h_2 + b_3$	$\hat{y} = softmax(a_3)$

How do you choose the right Network Configuration

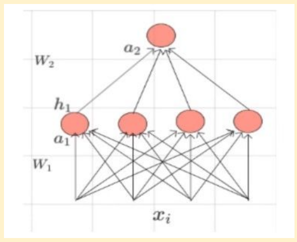
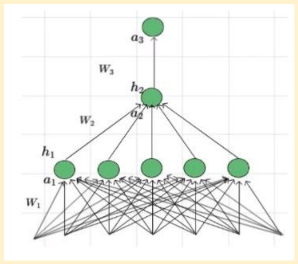
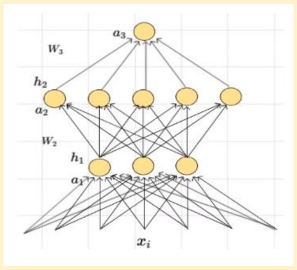
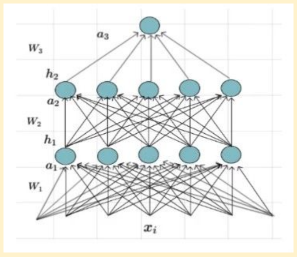
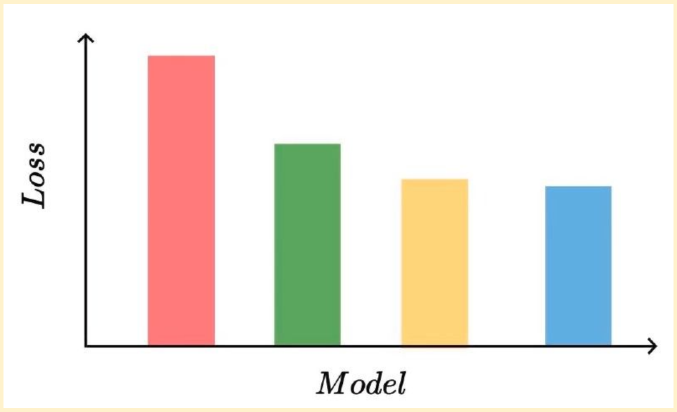
In practice how would you deal with extreme non-linearity

1. Consider the following datasets, 2D and 3D

Dimension	Data	
2D		
3D		

2. Up to 3D, we can visualise our data to check if it is linearly-separable or not.
3. However, in real-life scenarios, datasets often approach 1000-10000 dimensions, so there is no way to visualise the data to ascertain non-linearity.
4. In order to choose the best configuration for our neural network, we need to try out different combinations and select the one with the lowest loss

5. Here are some sample configurations and their corresponding losses

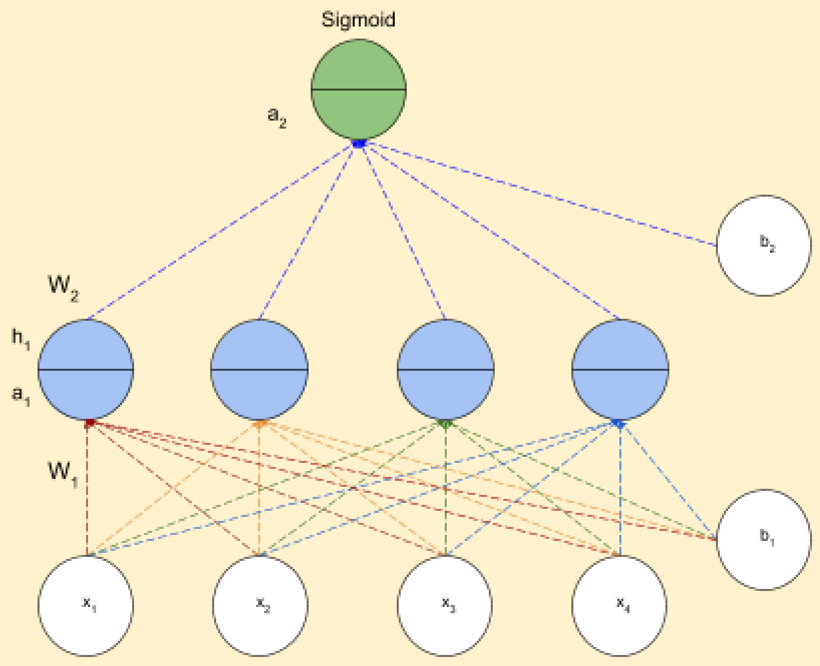
6. From the above figures, it is evident that the fourth yields the lowest loss.
7. This process of tuning the DNN i.e. The No. of layers, No. of neurons in each layer, learning rate, batch size etc are together called **Hyperparameter Tuning**

Loss function

Loss function for Binary Classification

What is the loss function that you can use for a binary classification problem

1. In normal cases, the number of neurons in the output layer would be equal to the number of classes
2. However a shortcut in the case of binary classification would be to use only one output neuron that uses a sigmoid function. Here is a diagrammatic representation of that configuration



3. Here, $\hat{y} = P(y = 1)$
4. Therefore, we can obtain $P(y = 0) = 1 - P(y = 1)$
5. Consider the following values for the variables
   a. $b = [\, 0.5 \;\; 0.3 \,]$
   b. y = 1

$$W_1 = \begin{bmatrix} 0.9 & 0.2 & 0.4 & 0.3 \\ -0.5 & 0.4 & 0.3 & 0.3 \\ 0.1 & 0.1 & -0.1 & 0.2 \\ -0.2 & 0.5 & 0.5 & / \end{bmatrix}$$

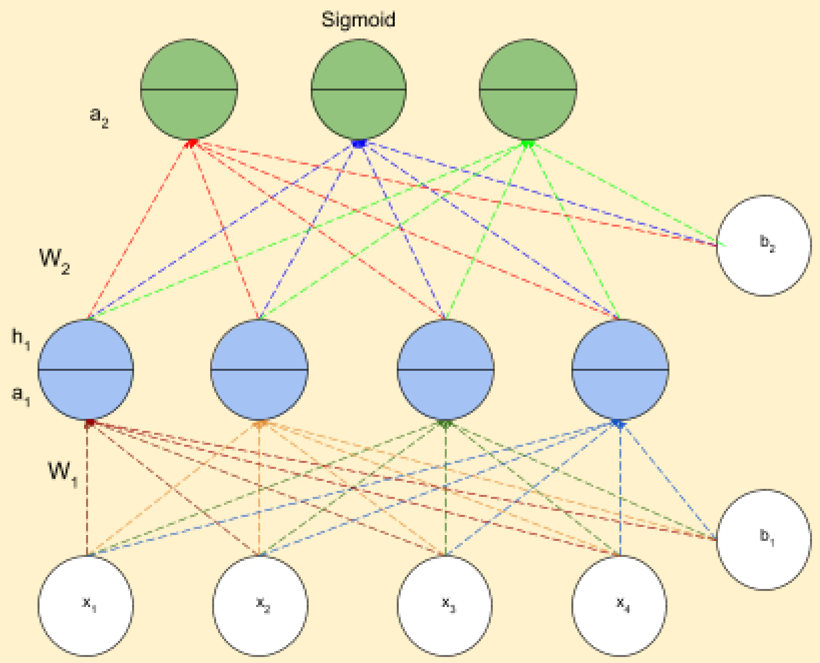
   c. $W_2 = [0.5 \;\; 0.8 \;\; -0.6 \;\; 0.3]$
   d. $x = [0.3 \;\; 0.5 \;\; -0.4 \;\; 0.3]$

6. The output values are as follows
   a. $a_1 = W_1 * x + b_1 \quad = [0.8 \ \ 0.52 \ \ 0.68 \ \ 0.67]$
   b. $h_1 = sigmoid(a_1) \quad = [0.69 \ \ 0.63 \ \ 0.66 \ \ 0.67]$
   c. $a_2 = W_2 * h_1 + b_2 \quad = 0.948$
   d. $\hat{y} = sigmoid(a_2) = 0.7207$
   e. In this case $y = 1 \quad True\ distribution\ [0 \ \ 1]$
   f. $Predicted\ distribution\ \hat{y} \quad [0.2793 \quad 0.7207]$
   g. Cross Entropy Loss:
      i. $L(\Theta) = (y)(-\log(\hat{y})) + (1-y)(-\log(\hat{y}))$
      ii. In this case, since y = 1
      iii. $L(\Theta) = -1 * log(0.7207)$
      iv. $L(\Theta) = 0.327$
7. Consider another case where x = [-0.6 -0.6 0.2 0.3] and true class y = 1
8. The output values are as follows
   a. $a_1 = W_1 * x + b_1 \quad = [0.01 \ \ 0.71 \ \ 0.42 \ \ 0.63]$
   b. $h_1 = sigmoid(a_1) \quad = [0.50 \ \ 0.67 \ \ 0.60 \ \ 0.65]$
   c. $a_2 = W_2 * h_1 + b_2 \quad = 0.921$
   d. $\hat{y} = sigmoid(a_2) = 0.7152$
   e. In this case $y = 0 \quad True\ distribution\ [1 \ \ 0]$
   f. $Predicted\ distribution\ \hat{y} \quad [0.2848 \quad 0.7152]$
   g. Cross Entropy Loss:
      i. $L(\Theta) = (y)(-\log(\hat{y})) + (1-y)(-\log(\hat{y}))$
      ii. In this case, since y = 0
      iii. $L(\Theta) = -1 * log(1 - 0.7152)$
      iv. $L(\Theta) = 1.2560$
      v. Here, even though the true value was 0, our neuron was outputting a very large value(0.7152) which was already indicative of a large loss value.

Loss function for Multi-Class Classification

What is the loss function that you can use for a multi-class classification problem

1. Here is an illustration of a sample multi-class classification Neural Network



2. Consider the following values for the parameters
   a. b = [0 0]
   b.

$$W_1 = \begin{bmatrix} 0.1 & 0.3 & 0.8 & -0.4 \\ --0.3 & -0.2 & 0.5 & 0.5 \\ -0.3 & 0.1 & 0.5 & 0.4 \\ 0.2 & 0.5 & -0.9 & 0.7 \end{bmatrix}$$

   c.

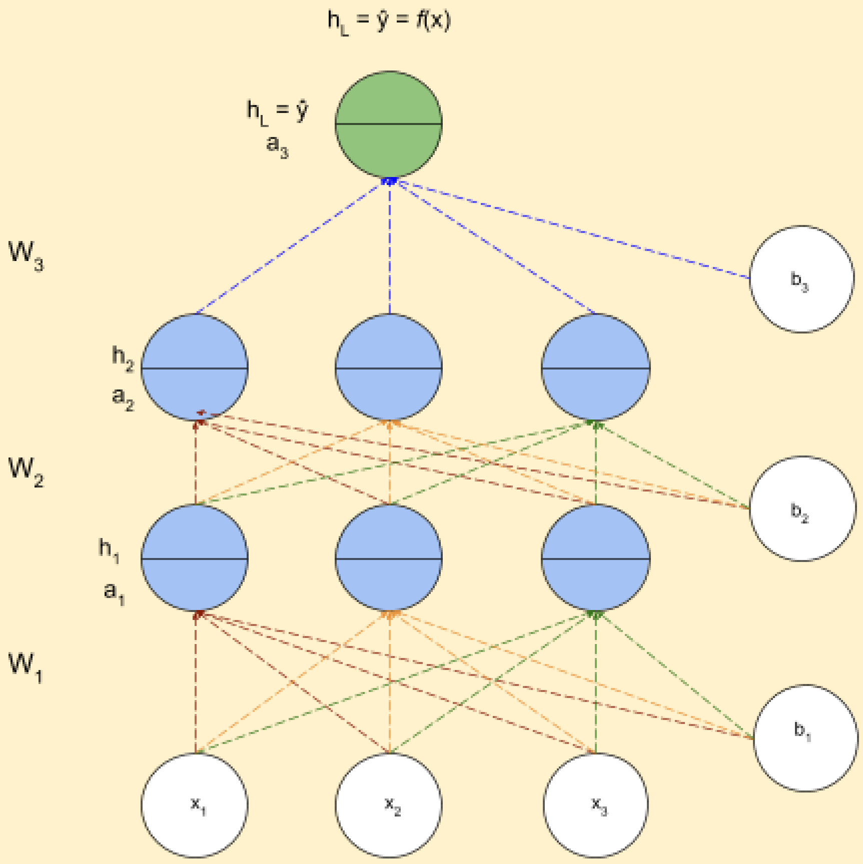
$$W_2 = \begin{bmatrix} 0.3 & 0.8 & -0.2 & -0.4 \\ 0.5 & -0.2 & -0.3 & 0.5 \\ 0.3 & 0.1 & 0.6 & 0.6 \end{bmatrix}$$

3. Consider a case where x = [-0.6 -0.6 0.2 0.3] and true class y =[0 1 0]
4. The output values are as follows
   a. $a_1 = W_1 * x + b_1 = [-0.19 \ \ -0.16 \ \ -0.09 \ \ 0.77]$
   b. $h_1 = sigmoid(a_1) = [0.45 \ \ 0.46 \ \ 0.49 \ \ 0.68]$
   c. $a_2 = W_2 * h_1 + b_2 = [0.13 \ \ 0.33 \ \ 0.89]$
   d. $\hat{y} = softmax(a_2) = [0.23 \ \ 0.28 \ \ 0.49]$
   e. Cross Entropy Loss
      i. $L(\Theta) = -\Sigma^{k}_{i=1} y_i log(\hat{y}_i)$
      ii. $L(\Theta) = -1 * \log(0.28)$
      iii. $L(\Theta) = 1.2729$
5. Consider another case where x = [0.6 0.4 0.6 0.1] and true class y =[0 0 1]
6. The output values are as follows
   a. $a_1 = W_1 * x + b_1 = [0.62 \ \ 0.09 \ \ 0.2 \ \ -0.15]$
   b. $h_1 = sigmoid(a_1) = [0.65 \ \ 0.52 \ \ 0.55 \ \ 0.46]$
   c. $a_2 = W_2 * h_1 + b_2 = [0.32 \ \ 0.29 \ \ 0.85]$
   d. $\hat{y} = softmax(a_2) = [0.2718 \ \ 0.2634 \ \ 0.4648]$
   e. Cross Entropy Loss
      i. $L(\Theta) = -\Sigma^{k}_{i=1} y_i log(\hat{y}_i)$
      ii. $L(\Theta) = -1 * \log(0.4648)$
      iii. $L(\Theta) = 0.7661$
7. A quick summary of what we've learned so far
   a. Given weights, we know how to compute the model's output for a given input
   b. This is called Forward-propagation.
   c. Given weights, we know how to compute the model's loss for a given input
   d. But who will give us the weights?
8. The weights can be obtained from the learning algorithm

Learning Algorithm (Non-Math version)

Can we use the same Gradient Descent algorithm as before

1. We will be looking at the non-math version of the learning algorithm
2. Consider the following Neural Network



3. The algorithm
   a. **Initialise:** w_{111}, w_{112}, ... w_{313}, b_1, b_2, b_3 randomly
   b. **Iterate over data**
      i. Compute $\hat{y}$
      ii. Compute L(w,b) Cross-entropy loss function
      iii. $w_{111} = w_{111} - \eta \Delta w_{111}$
      iv. $w_{112} = w_{112} - \eta \Delta w_{112}$
      ...
      v. $w_{313} = w_{111} - \eta \Delta w_{313}$
      vi. $b_i = b_i + \eta \Delta b_i$
      vii. Pytorch/Tensorflow have functions to compute $\frac{\delta l}{\delta w}$ *and* $\frac{\delta l}{\delta b}$
   c. **Till satisfied**
      i. Number of epochs is reached (ie 1000 passes/epochs)
      ii. Continue till Loss < ε (some defined value)

Evaluation

How do you check the performance of a deep neural network

1. Consider the Indian Liver Patient Diagnosis task

Age	Albumin	T_Bilirubin
65	3.3	0.7
62	3.2	10.9
20	4	1.1
84	3.2	0.7

...... More features

y	ŷ
0	0
0	1
1	1
1	0

2. $Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ \ number\ of\ predictions}$
3. $Accuracy = \frac{2}{4} = 500\%$
4. The question here is, how do we resolve a probability distribution like [0.45 0.55] to a binary output
5. It is done by picking the class that corresponds to the highest probability, in the above case it is 1.
6. For multiclass classification, the concept remains the same. The label is selected based on the highest value in the probability distribution.
7. The predicted label corresponds to the index of the highest value in the probability distribution (argmax)

Test Data	y	Predicted
	0	0
	1	7
	3	8
	5	5
	1	1

8. In addition to accuracy, we can also calculate the per-class accuracy. In this case, the accuracy of the class **'1'** is 50% and of class **'5'** is 100%
9. This allows us to analyse where the model is performing poorly, and enables us to take steps to improve the accuracy for the lagging classes, such as adding more images etc.

Summary

What are the new things that we learned in this module?

1. Here are some of the takeaways from this chapter
    a. Data: Real inputs $x_i \in \mathbb{R}$
    b. Task:
        i. Binary classification
        ii. Multi-class classification
        iii. Regression
    c. Model: Deep Neural Network to deal with complex decision boundaries
    d. Loss:
        i. Cross entropy loss: $L(\Theta) = -\frac{1}{N}\Sigma^{N}_{i=1}\Sigma^{d}_{i=1} y_{ij} \log(\hat{y}_{ij})$
        ii. Square Error Loss: $L(\Theta) = -\frac{1}{N}\Sigma^{N}_{i=1}\Sigma^{d}_{i=1}(y_{ij} - \hat{y}_{ij})^2$
    e. Learning: Gradient Descent with backpropagation
    f. Evaluation:
        i. Accuracy = $\frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ predictions}$
        ii. Per-class Accuracy = $\frac{Number\ of\ correct\ predictions\ of\ a\ class}{Total\ Number\ of\ true\ values\ of\ that\ class}$
2. Note: the text highlighted in red indicates concepts that will be covered in the upcoming chapters.