

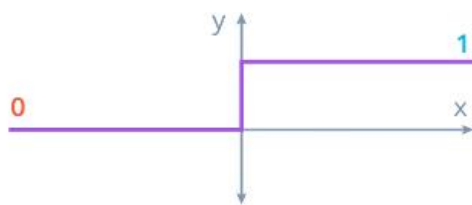
Error Function:

The error function should be continuous and differentiable.

Discrete vs Continuous Predictions

The **discrete** answer will be of the form (yes, no), while a **continuous** will be a number.

Activation Functions



DISCRETE:
Step Function

$$y = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$



CONTINUOUS:
Sigmoid Function

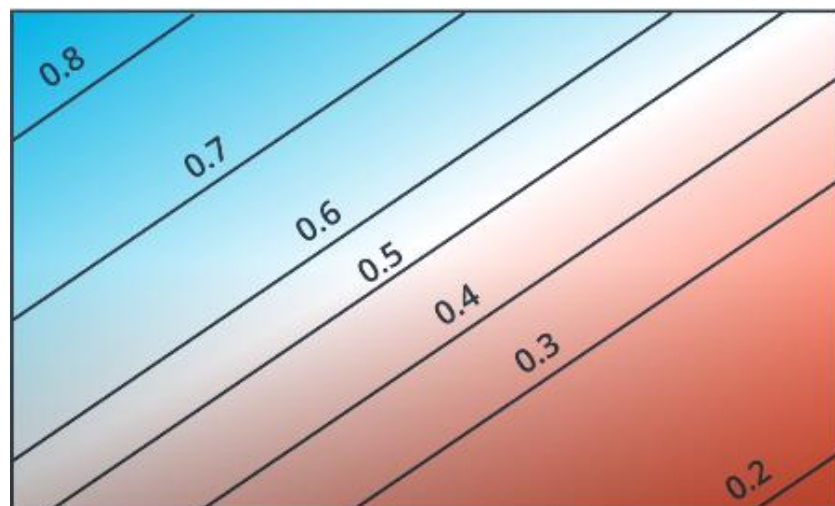
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

1- discrete Activation Function:

It will give us value 1 if x is greater than 0, and 0 when x is less than 0.

2- Continuous Activation Function:

the large positive numbers will give us value that is close to “1”,
and large positive numbers will give us value that is close to “0”,
and for numbers that is close to zero, it will give us a value close to “0.5”.



QUIZ QUESTION

The sigmoid function is defined as $\text{sigmoid}(x) = 1/(1+e^{-x})$. If the score is defined by $4x_1 + 5x_2 - 9 = \text{score}$, then which of the following points has exactly a 50% probability of being blue or red? (Choose all that are correct.)

☒ (1, 1)

☐ (2, 4)

☐ (5, -5)

☒ (-4, 5)

The Softmax Function:

LINEAR FUNCTION

SCORES:

Z_1, \dots, Z_n




$$P(\text{class } i) = \frac{e^{Z_i}}{e^{Z_1} + \dots + e^{Z_n}}$$

If we want to get the probability of event than have more than two outcomes, then **Activation** will not work well, here we need **Softmax Function**.

Example:

Let us assume we have 3 animals, we saw duck 2 times, beaver 1 time, and walrus 0 times. To get the probability of each we put it as power of “e” then divide it by the sum of all,

The reason why we use “e” instead of putting the number directly is because this way we will not have a problem even if we had negative numbers.

	2	$\frac{e^2}{e^2 + e^1 + e^0}$	=	0.67	P(duck)
	1	$\frac{e^1}{e^2 + e^1 + e^0}$	=	0.24	P(beaver)
	0	$\frac{e^0}{e^2 + e^1 + e^0}$	=	0.09	P(walrus)

Quiz: Coding Softmax

And now, your time to shine! Let's code the formula for the Softmax function in Python.

My Solution:

```
softmax.py  solution.py
1  import numpy as np
2
3  # Write a function that takes as input a list of numbers, and returns
4  # the list of values given by the softmax function.
5  def softmax(L):
6      #e = 2.71828
7      val = []
8      sum = 0
9      for i in L:
10         sum = sum + (2.71828 ** i)
11
12     for i in L:
13         val.append((2.71828 ** i)/sum)
14     return val
15     pass
```

Test Run:

Trying for L=[5,6,7].

The correct answer is

```
[0.09003057317038046, 0.24472847105479764, 0.6652409557748219]
```

And your code returned

```
[0.09003066856407584, 0.24472856574435609, 0.6652407656915682]
```











Correct!

Udacity Solution:

```
softmax.py  solution.py
1  import numpy as np
2
3  def softmax(L):
4      expl = np.exp(L)
5      sumExpl = sum(expl)
6      result = []
7      for i in expl:
8         result.append(i*1.0/sumExpl)
9      return result
10
11     # Note: The function np.divide can also be used here, as follows:
12     # def softmax(L):
13     #     expl = np.exp(L)
14     #     return np.divide (expl, expl.sum())
```

One Hot encoding:

One-Hot Encoding

2 outcomes		>=3 outcomes			
GIFT?	VARIABLE	ANIMAL	DUCK?	BEAVER?	WALRUS?
	1		1	0	0
	1		0	1	0
	0		1	0	0
	1		0	0	1
	0		0	1	0

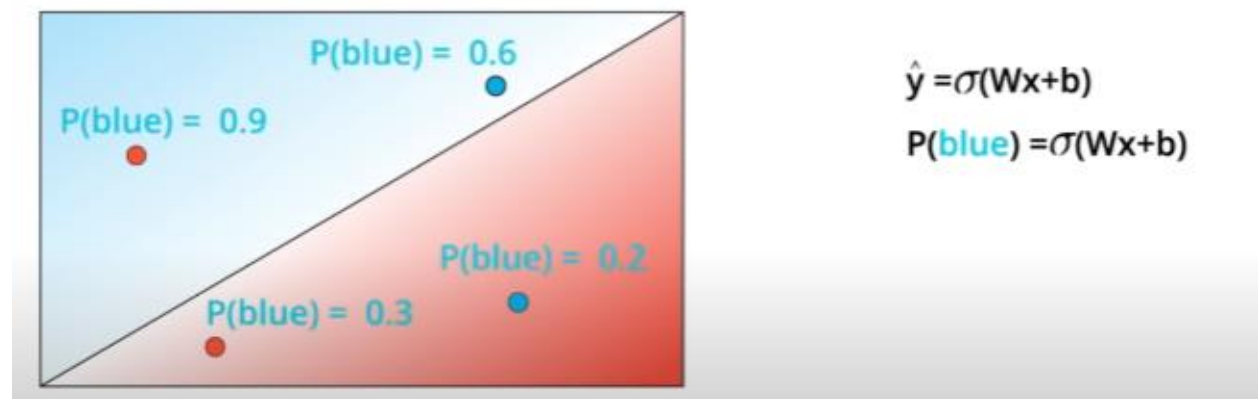
When we have **2 outcomes**, for example getting a gift or not, we can say the input variable is 1, and when you do not it is 0.

But again, what if we have **more than 2 outcomes**? Like the previous example, then we use One Hot encoding, and how it works is:

We give a variable for each one of the outcomes, and when the outcome happens its column will be 1 and the others will be 0, like in the example above, when the duck accord it's column is 1 and other are 0.

This is **One-Hot Encoding**, and it will be used a lot for processing data.

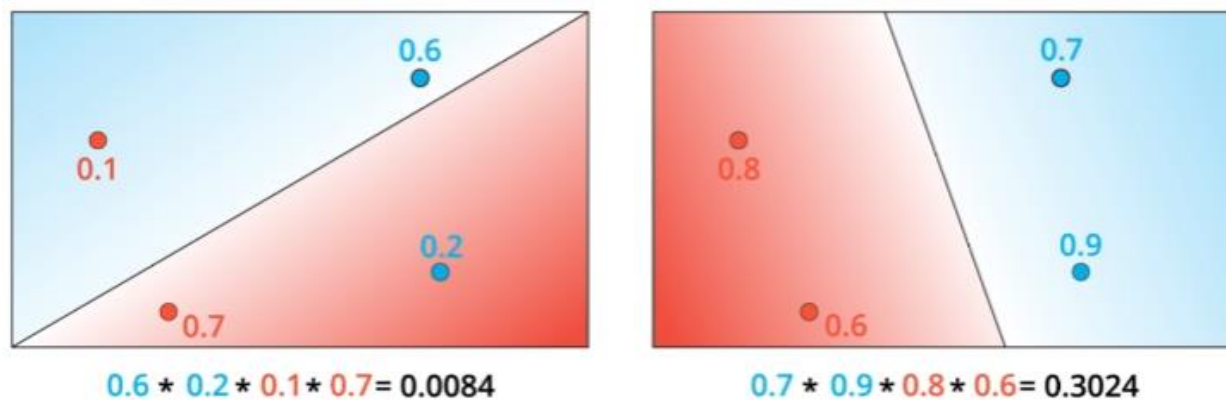
Maximum Likelihood



For this model we use the upper method to calculate the percentage of a point being blue, to get the percentage of being red we take the inverse,

Now what we want to do is to calculate the probability of the four points are of the correct color.

We do that by getting the product of the probability of all points,



The final answer is called “P(all)”

Here we calculate the probability of two models, here the model on the right is better because it makes the arrangement of the points much more likely to have those colors.

This method is called Maximum Likelihood.

Maximizing Probabilities

For the previous, we need to maximize the probability, and we do that by turning products into sums, we can do that by using $\log()$: since $\log(ab) = \log(a) + \log(b)$,

we will be using the “**natural logarithm**”, also called “**log base e**” or “**ln**”.

Notice that nothing difference happens with **log base 10**, everything is the same as everything gets scaled by the same factor but using “**ln**” is just more for convention.

$$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$$

$$\ln(0.6) + \ln(0.2) + \ln(0.1) + \ln(0.7) \\ -0.51 \quad -1.61 \quad -2.3 \quad -0.36$$

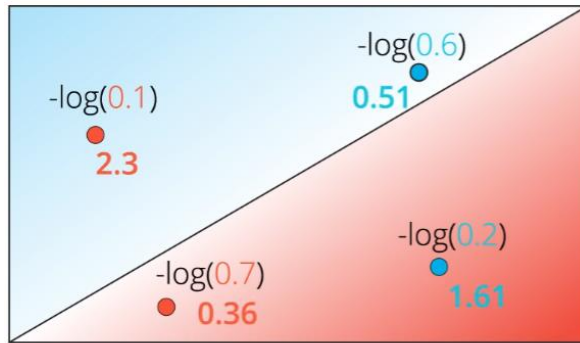
$$-\ln(0.6) - \ln(0.2) - \ln(0.1) - \ln(0.7) = 4.8 \\ 0.51 \quad 1.61 \quad 2.3 \quad 0.36$$

$$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$$

$$\ln(0.7) + \ln(0.9) + \ln(0.8) + \ln(0.6) \\ -0.36 \quad -0.1 \quad -0.22 \quad -0.51$$

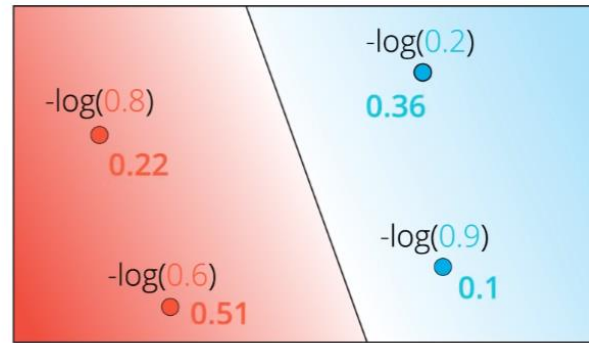
$$-\ln(0.7) - \ln(0.9) - \ln(0.8) - \ln(0.6) = 1.2 \\ 0.36 \quad 0.1 \quad 0.22 \quad 0.51$$

For above, we take the negative of each $\ln(x)$, because the log of a number between 1-0 is always negative, the result is called “**cross entropy**”, a good model will give us a low “**cross entropy**”, and a bad model will give us high one.



$$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$$

$$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7) = 4.8$$



$$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$$

$$-\log(0.7) - \log(0.9) - \log(0.8) - \log(0.6) = 1.2$$

If we calculate the value “ $-\log(x)$ ” for each point, we will see that the points which are **mis-classified** has **higher values**, while the points that are **correctly classified** have **smaller values**.










The reason for this of this is that a correctly classified point will have a probability closer to 1, which give us small value when taking the negative of “ $\log(x)$ ”.

Now the goal is to minimize the “**cross entropy**” as much as we can.

Example:

Here we have doors that could have present or not, with each percentage, and we get the Cross-Entropy like the photo underneath.

In this example we want to get "Cross-Entropy" for having present in the 1st and 2nd door but not in the 3rd.

 $p_1 = 0.8$				Cross-Entropy $-\ln(0.8) - \ln(0.7) - \ln(0.9)$
 $p_2 = 0.7$	 0.8	 0.7	 0.9	
 $p_3 = 0.1$	p_1	p_2	$1 - p_3$	
$y_i = 1$ if present on box i	$y_1 = 1$	$y_2 = 1$	$y_3 = 0$	

cross entropy formula:

$$\text{Cross-Entropy} = - \sum_{i=1}^m y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

P = the probability of event being positive.

Y = 1 if even is positive, else = 0

Multi-Class Cross-Entropy

ANIMAL	DOOR 1	DOOR 2	DOOR 3
	p_{11}	p_{12}	p_{13}
	p_{21}	p_{22}	p_{23}
	p_{31}	p_{32}	p_{33}

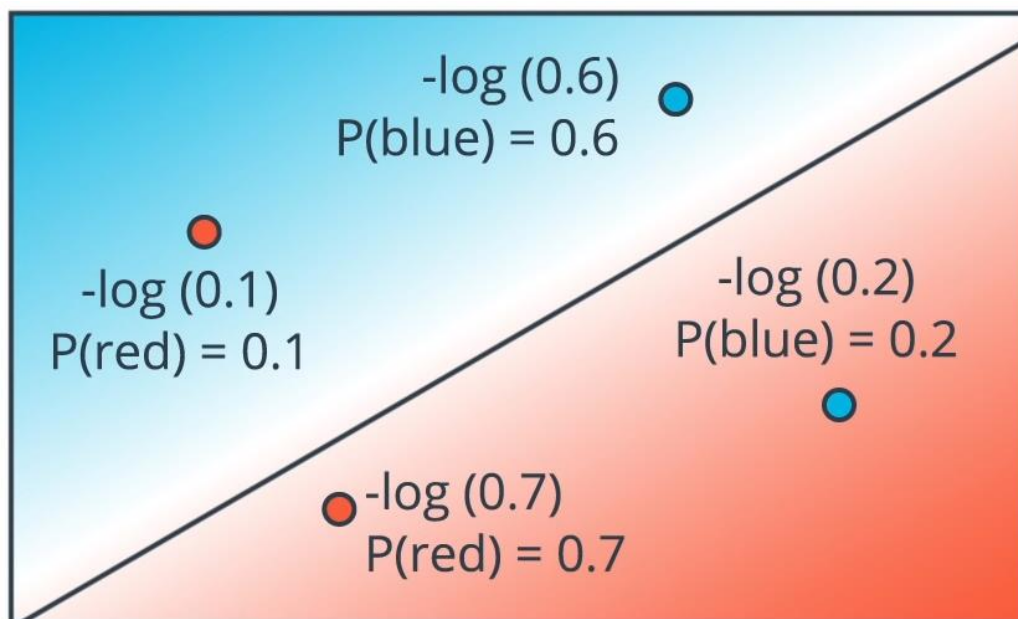
$y_{1j} = 1$ if  behind 

$y_{2j} = 1$ if  behind 

$y_{3j} = 1$ if  behind 

$$\text{Cross-Entropy} = - \sum_{i=1}^n \sum_{j=1}^m y_{ij} \ln(p_{ij})$$

Example:



$$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7) = 4.8$$

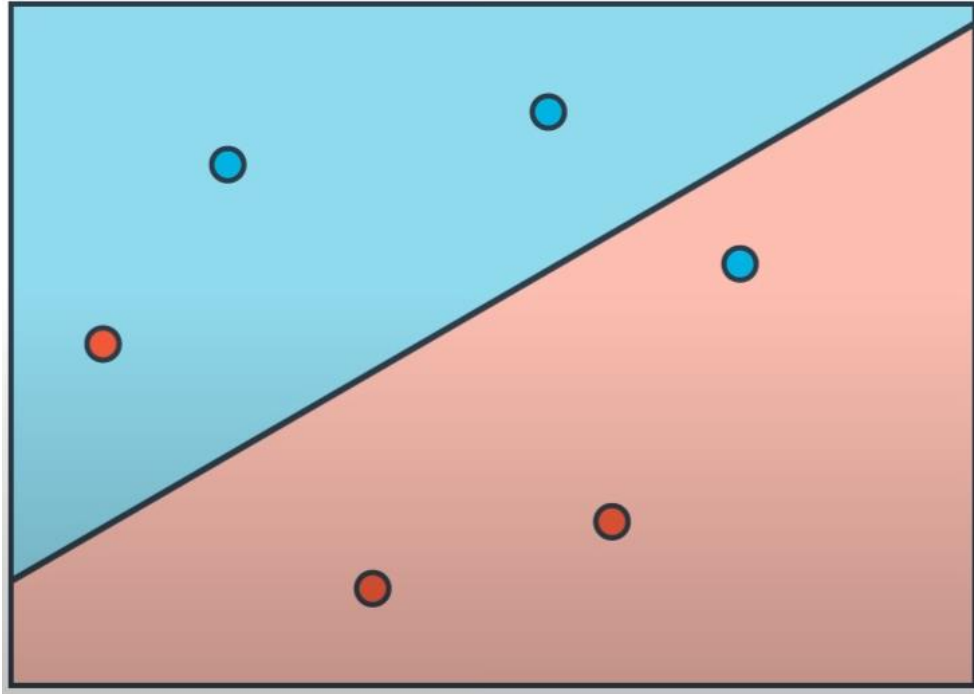
0.51

1.61

2.3

0.36

Minimizing the error function



Example graph, we will be started with random weights which will give us the predictions, $\sigma(Wx + b)$.

$$E(W,b) = -\frac{1}{m} \sum_{i=1}^m (1-y_i) \ln(1-\sigma(Wx^{(i)}+b)) + y_i \ln(\sigma(Wx^{(i)}+b))$$

We will use gradient decent in order to get a smaller error function, which will give us $\sigma(W'x + b')$ which will give us much better predictions.

Gradient Descent Algorithm (Pseudo code):

1. Start with random weights:

$$w_1, \dots, w_n, b$$

"this will give us a random line $\sigma(Wx + b)$, Now we calculate the error for every point, the misclassified points have high error."2

2. For every point (x_1, \dots, x_n) :

2.1. For $i = 1 \dots n$

2.1.1. Update $w'_i \leftarrow w_i - \alpha (\hat{y} - y)x_i$

2.1.2. Update $b' \leftarrow b - \alpha (\hat{y} - y)$

A

3. Repeat until error is small

When we have small error, that means a line that separates the points better.

https://www.youtube.com/watch?v=snxmBgi_GeU&t=81s&ab_channel=Udacity

Perceptron vs Gradient Descent

GRADIENT DESCENT ALGORITHM:

Change
 w_i to $w_i + \alpha(y - \hat{y})x_i$

PERCEPTRON ALGORITHM:

If x is misclassified:

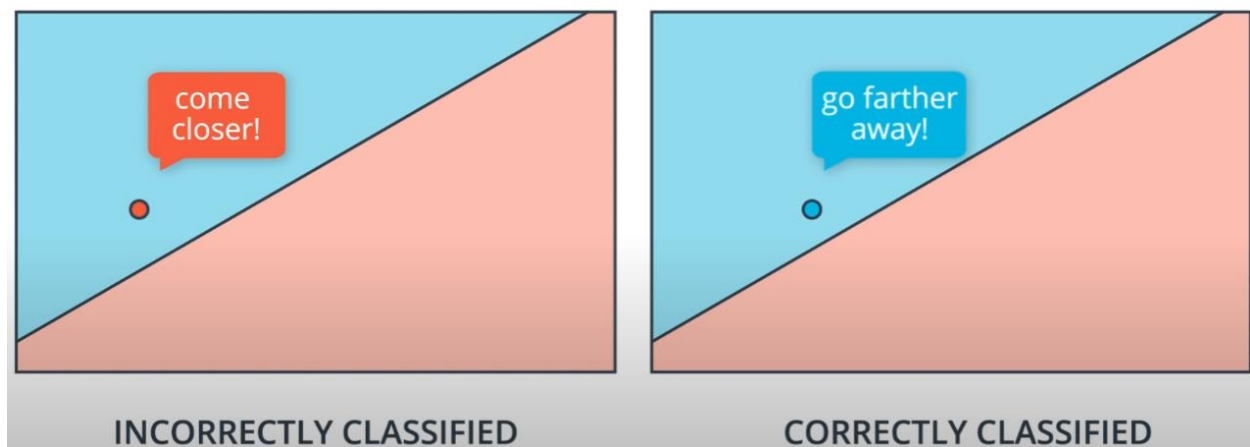
Change w_i to $\begin{cases} w_i + \alpha x_i & \text{if positive} \\ w_i - \alpha x_i & \text{if negative} \end{cases}$

If correctly classified: $y - \hat{y} = 0$

If misclassified: $\begin{cases} y - \hat{y} = 1 & \text{if positive} \\ y - \hat{y} = -1 & \text{if negative} \end{cases}$

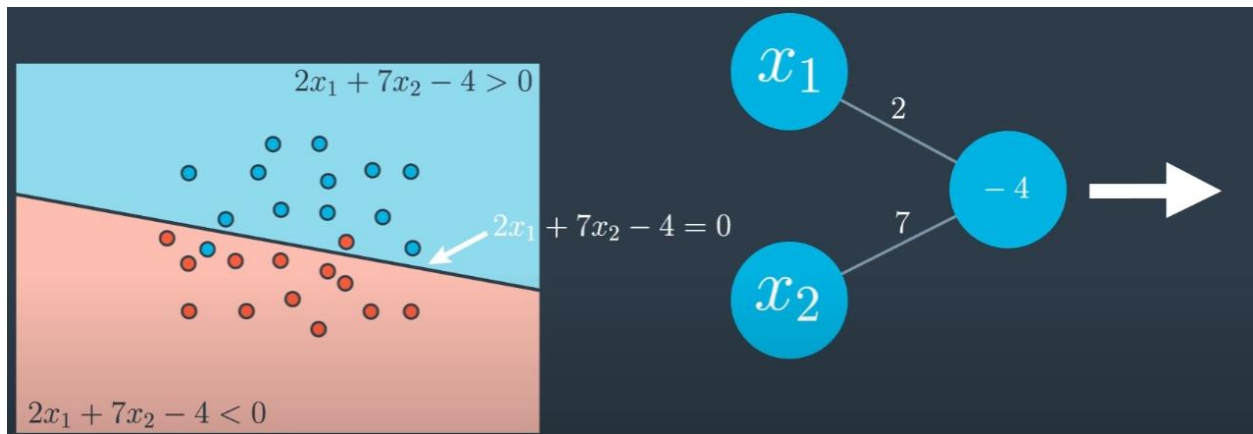
If we look on them, we can see that both are the same thing, the only difference is that "Gradient Descent" can take any value between 0 and 1, while "Perceptron" can only take exactly 0 or 1.

What Gradient Descent does:



When the point is misclassified, it tells the line to come closer, if it is correctly classified it tells the line to go away, this way by listening to all points it will eventually get a pretty good solution.

Continuous Perceptrons



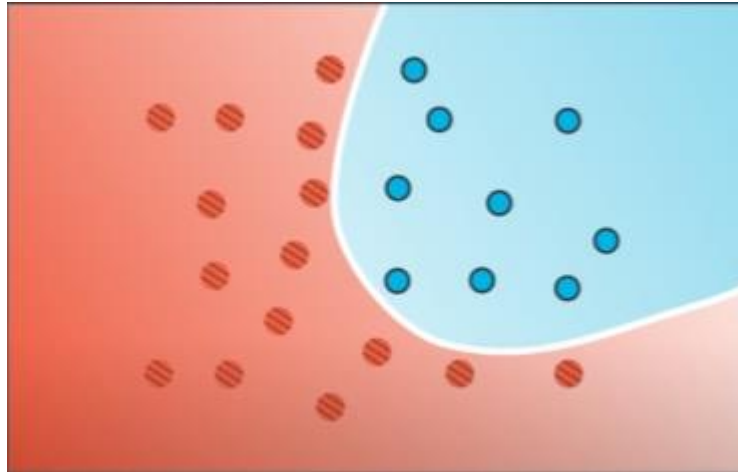
What the perceptron does here is take the points x_1 , x_2 and then return the probability of the point being blue, in this case it could be 0.9

And this mimics the neurons in the brain as they receive nerve impulses, do something inside then return nerve impulses.

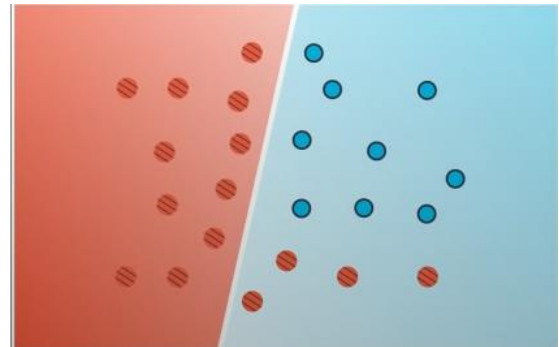
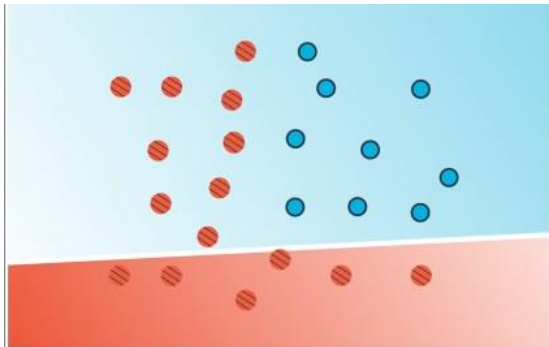


Non-Linear Data and its models

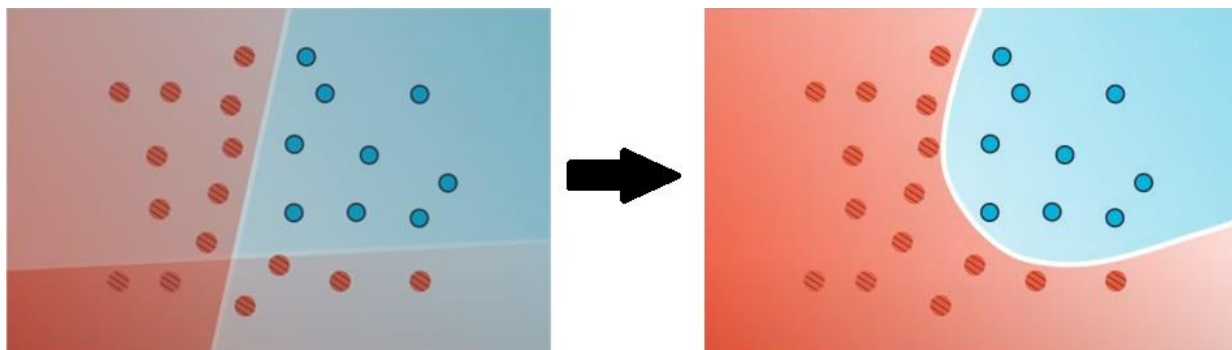
As we know the real world is way more complex than we saw and there are a lot of data that cannot be separated by a line, here we need the Non-Linear models.



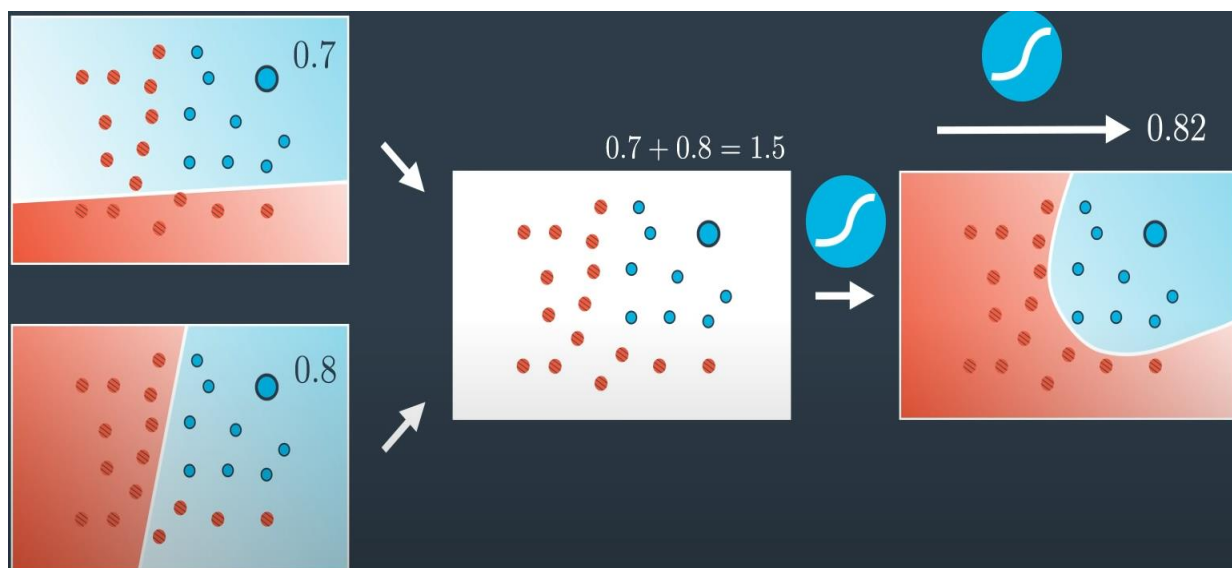
To make this non-linear model, we are going to do a very simple trick, we will combine these two linear models.



It is something like this:



Now every point has different probability of being blue, and the same point in the two linear models have different probability since the line is different, so to get the probability of the point in the new model we use this simple method.



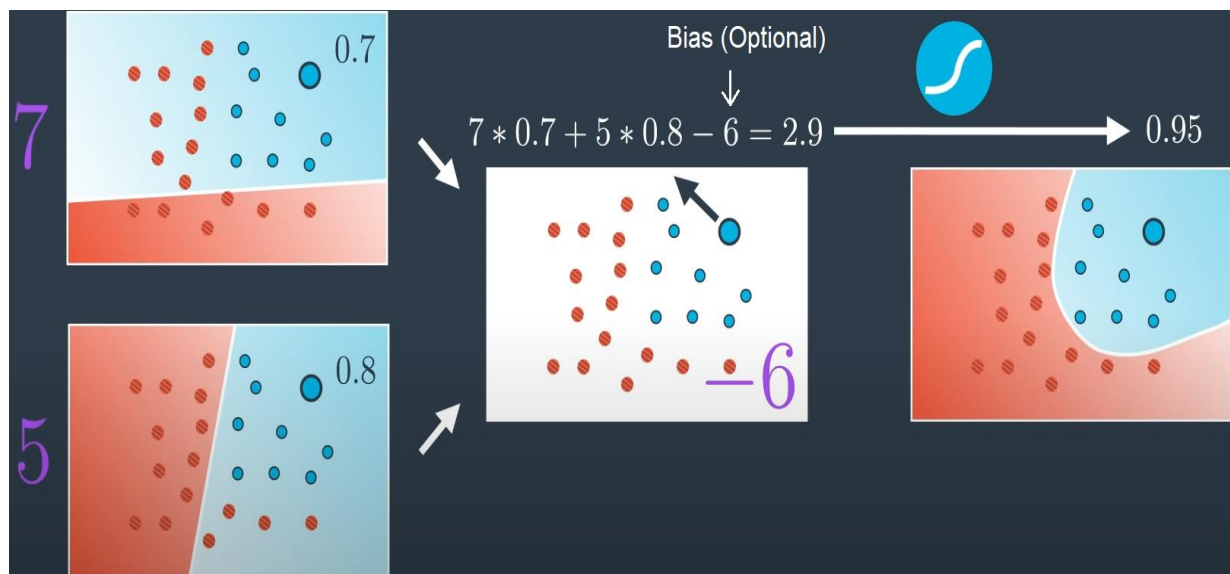
We add the two, then get the sigmoid of the sum, and the result is the probability of the point in the new model.

Now what if we want to give a weight for each model, like we want the first model to have more impact on the combined model than the second one.

Example: let's say we want to combine the two and take 7 times the first one and 5 time the second one:

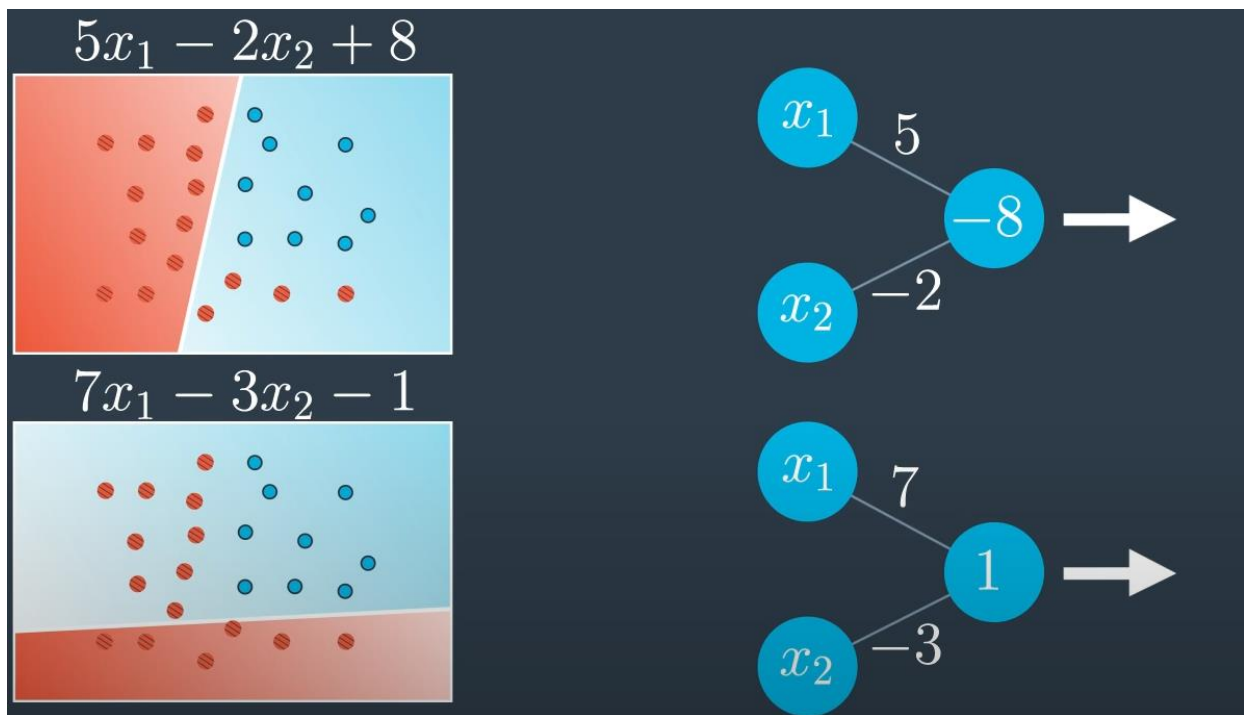
We can do it by taking the (probability of the first * 7) plus (the probability of the second * 5) minus (Bias) if we have one.

Then take the sigmoid of the result, and we get our new probability.

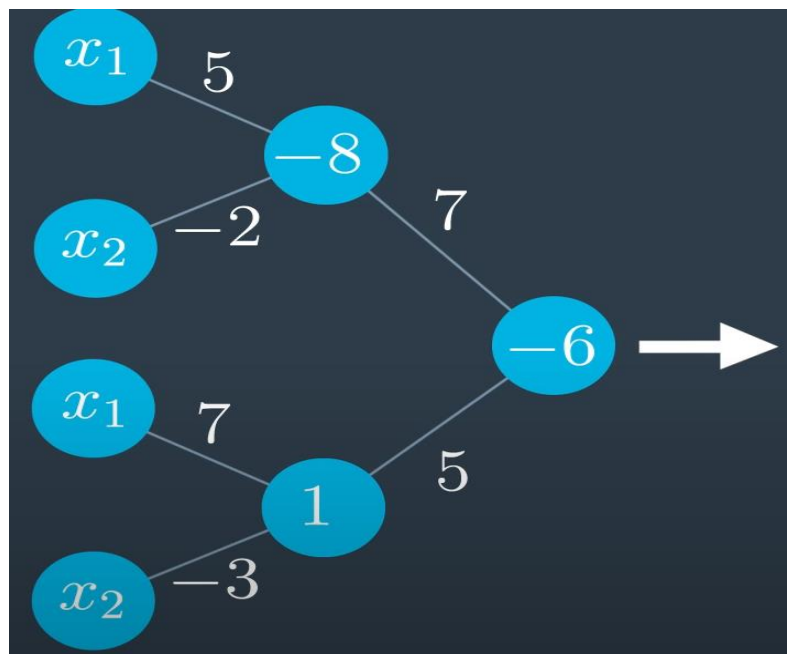


Example:

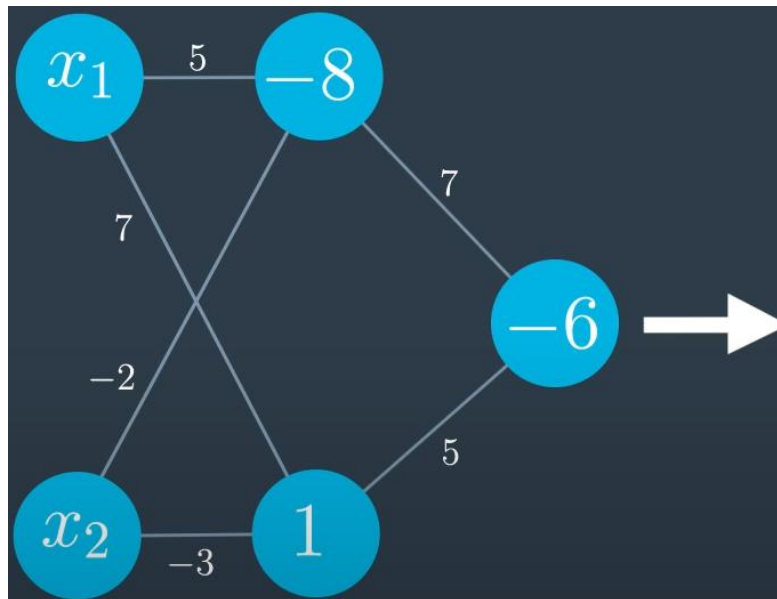
Let us take the same two equations which is represented by these perceptions.



Now let us combine these two models using the linear equations, and we get this.

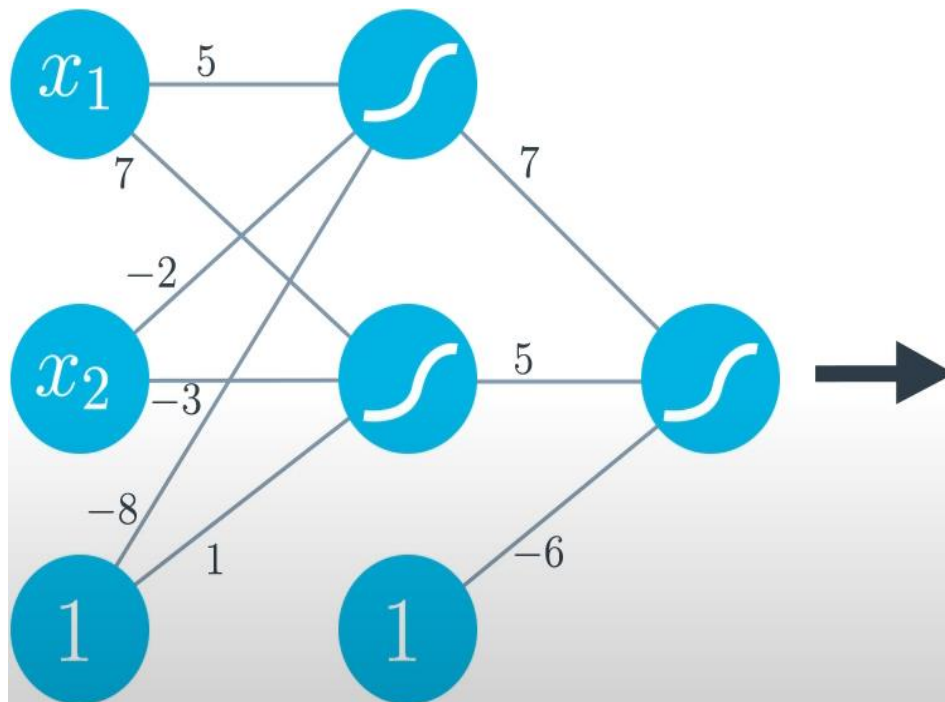


After cleaning it a bit it looks like this:



Notice this was draw using the bias that puts a bias inside the node,

It can also be drawn using the notation that keep the bias as a separate node.



Here, in every layer we have a bias unit coming from a node containing (1).

QUESTION 1 OF 2

Based on the above video, let's define the combination of two new perceptrons as $w_1 * 0.4 + w_2 * 0.6 + b$. Which of the following values for the weights and the bias would result in the final probability of the point to be 0.88?

☐ $w_1: 2, w_2: 6, b: -2$

☒ $w_1: 3, w_2: 5, b: -2.2$

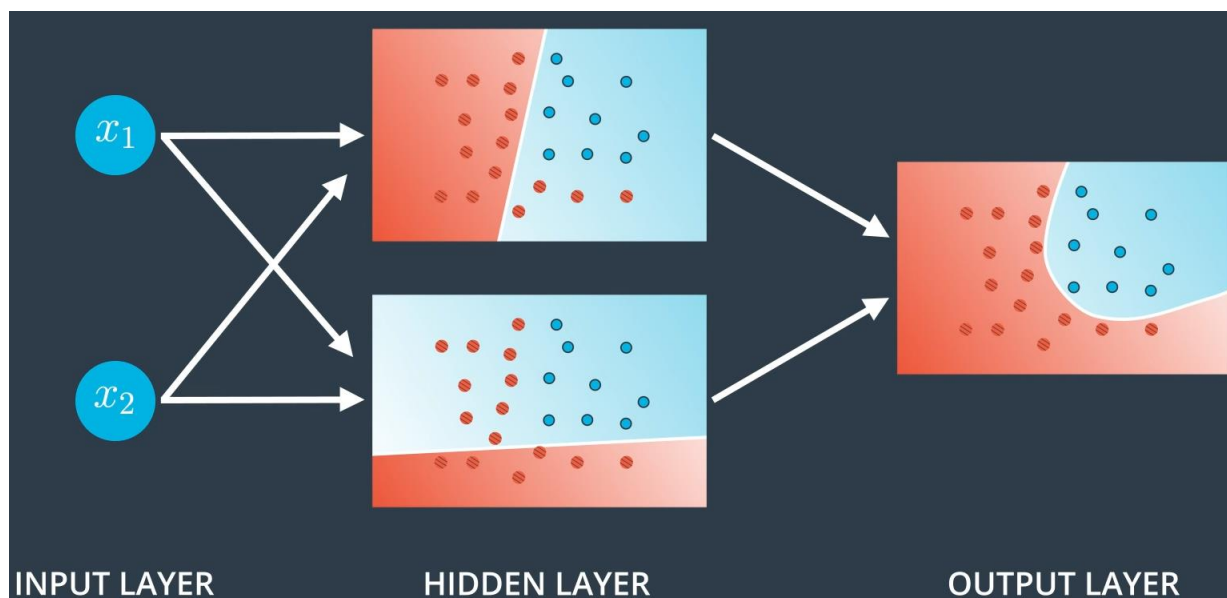
☐ $w_1: 5, w_2: 4, b: -3$

$$(3 * 0.4) + (5 * 0.4) + (-2.2) = 2$$

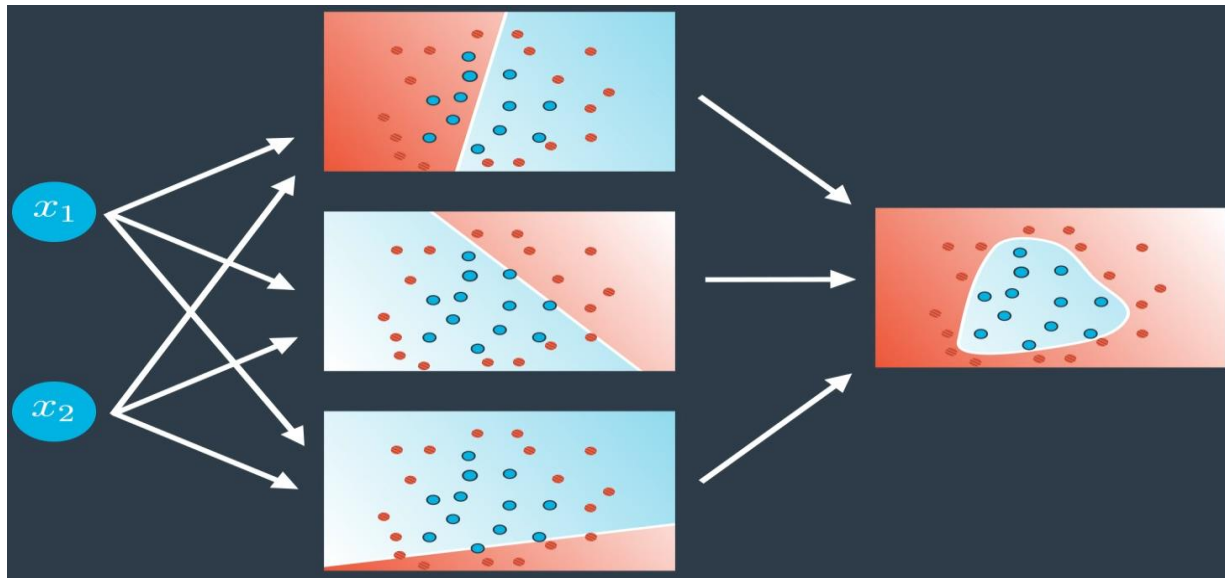
Sigmoid (2) = $\frac{1}{1 + e^{-x}}$ — $>$ $\frac{1}{1 + e^{-2}}$

Multiple layers:

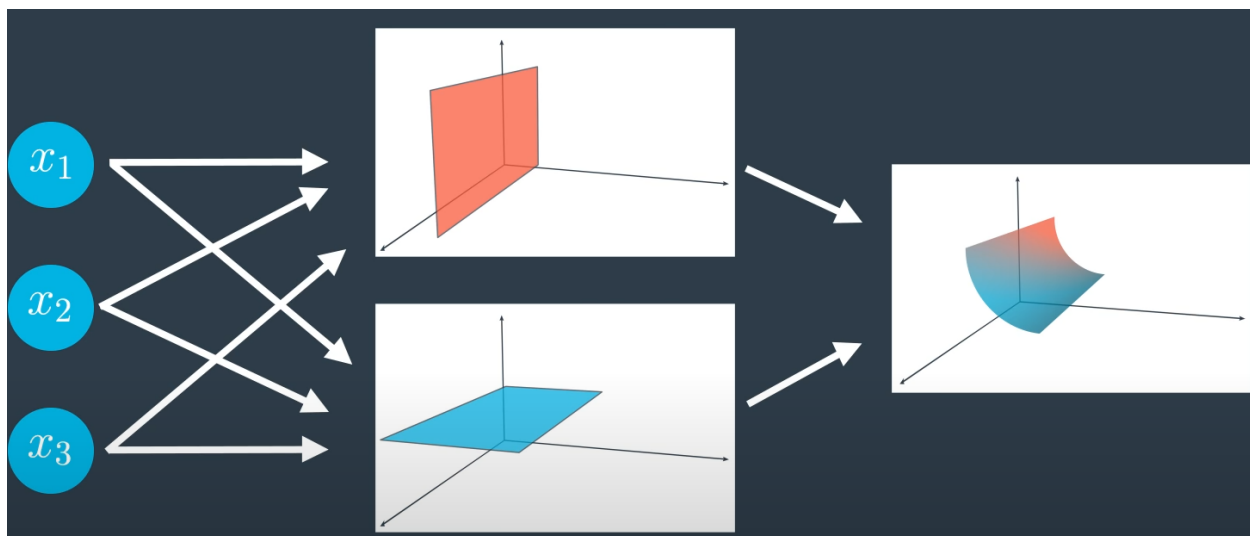
The we have several layers:



We can have Larger Hidden Layer:



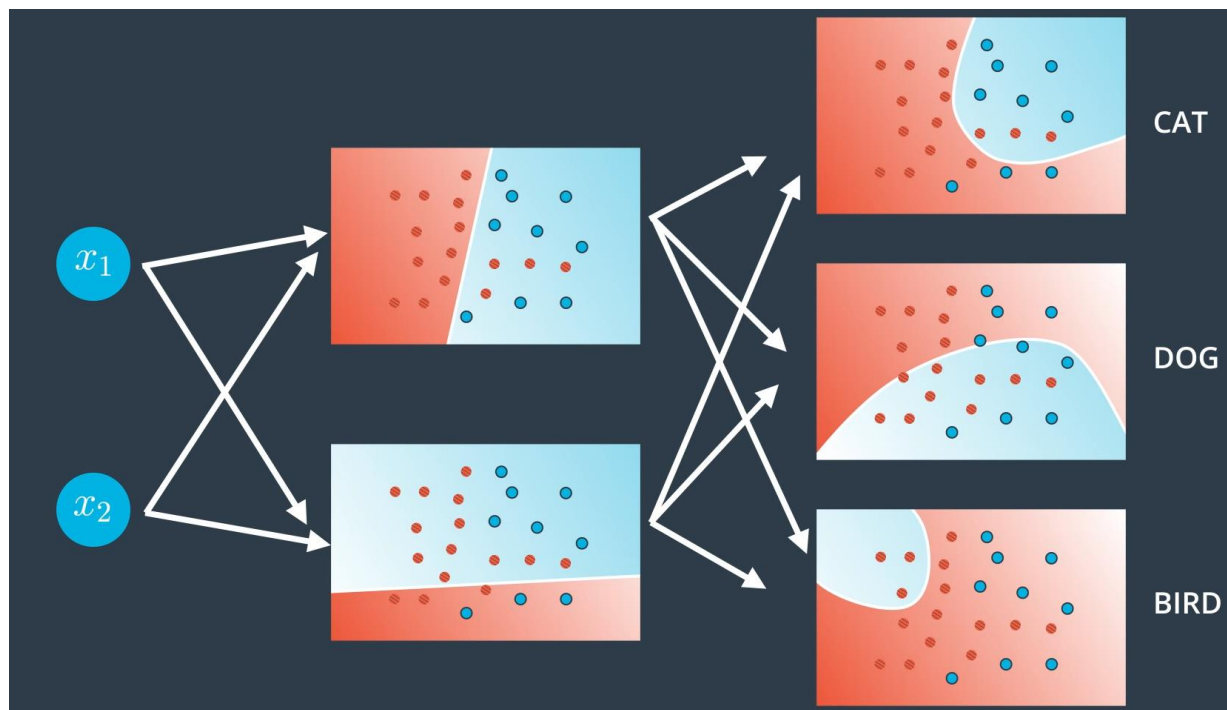
Or we can have more Nodes (Larger Input Layers).



This means in this example that we are in 3D dimension, not 2D anymore.

In general, having n number of nodes means we are in “N Dimensional space”.

what if we have more output layers?



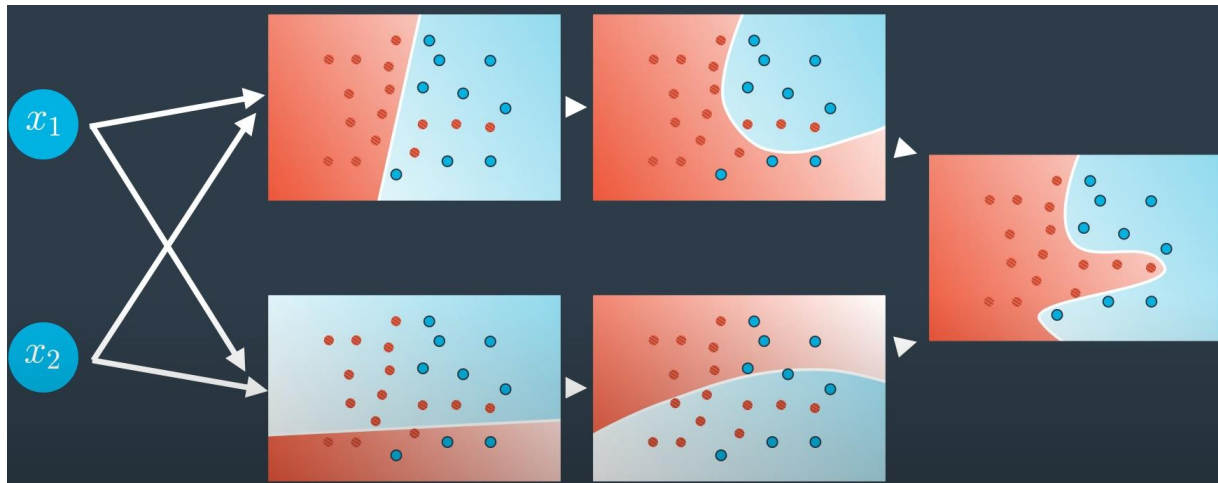
This means we have Multi-Class Classification model, for example a model that classify the image to be a cat, dog, or a bird.

Quiz:

How many nodes in the output layer would you require if you were trying to classify all the letters in the English alphabet?

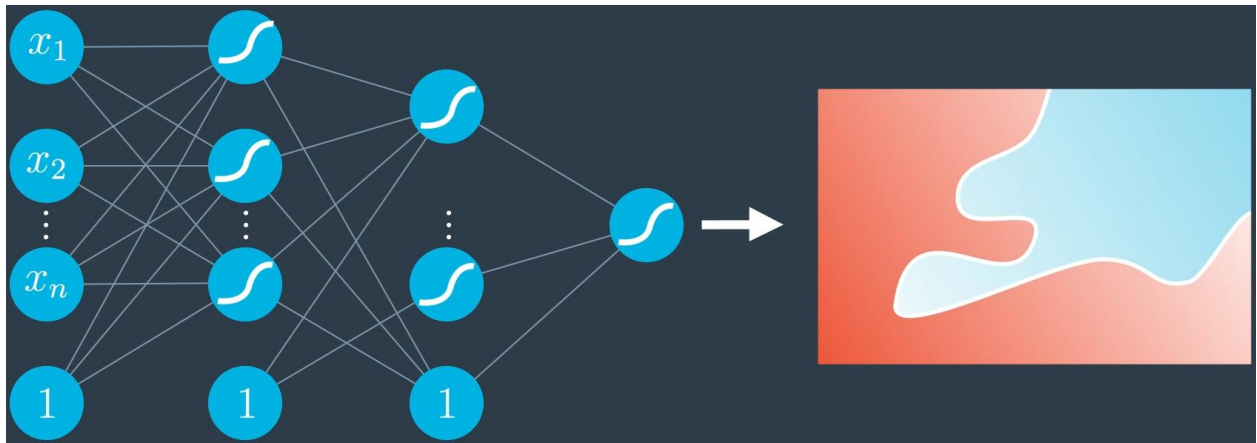
Answer: 26, (or 52 if we consider Uppercase)

What if we have more layers?



Here we have what is called “Deep Neural Network”, what happens here is our linear models combine to create a non-linear models, and then these combine to create even more non-linear models.

In general, we can do this as many times as we want and obtain highly complex models with lots of hidden layers.



Many of the models in real life such as self-driving cars, or gameplaying agents have so many hidden layers.

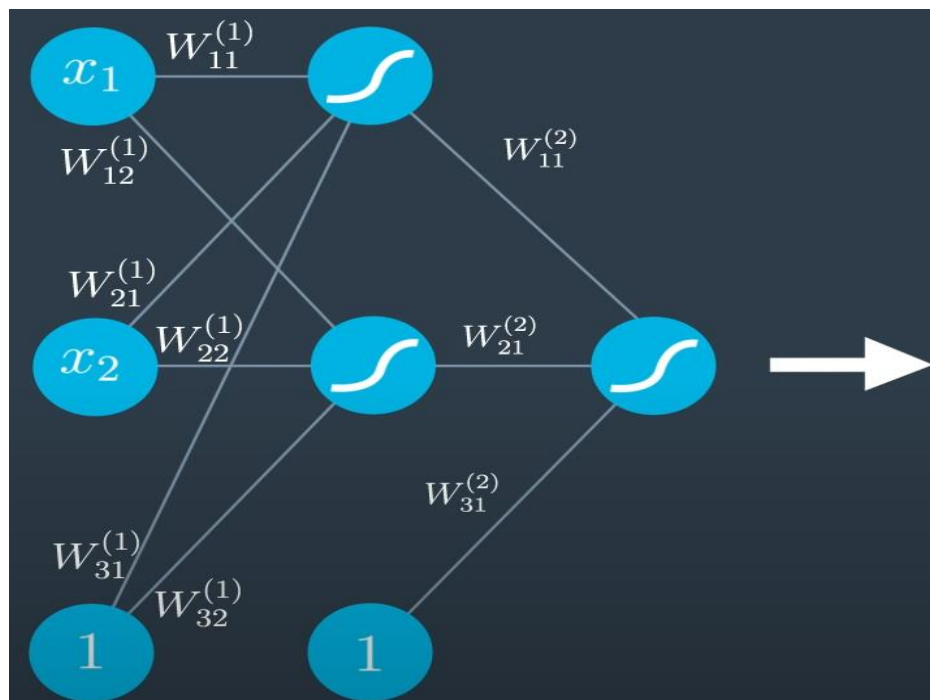
Feedforward

To get the probability of a Point to be blue (Positive) in a multi-layers model, we use this method.

$$\hat{y} = \sigma \begin{pmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{pmatrix} \sigma \begin{pmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

$$\hat{y} = \sigma \circ W^{(2)} \circ \sigma \circ W^{(1)}(x)$$

Perception in this example:



Error Function

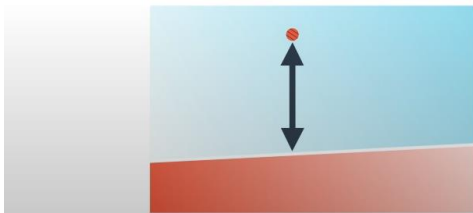
The error function is still the same form the Linear Model, except now the \hat{y} is more complicated.

Linear Prediction

$$\hat{y} = \sigma(Wx + b)$$

ERROR FUNCTION

$$E(W) = -\frac{1}{m} \sum_{i=1}^m y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)$$

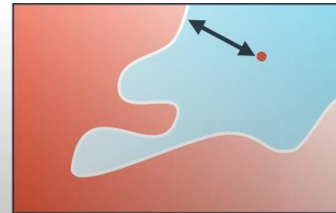


non-Linear Prediction

$$\hat{y} = \sigma \circ W^{(3)} \circ \sigma \circ W^{(2)} \circ \sigma \circ W^{(1)}$$

ERROR FUNCTION

$$E(W) = -\frac{1}{m} \sum_{i=1}^m y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)$$



Backpropagation

In a nutshell, backpropagation will consist of:

- **Doing a feedforward operation.**
- **Comparing the output of the model with the desired output.**
- **Calculating the error.**
- **Running the feedforward operation backwards (backpropagation) to spread the error to each of the weights.**
- **Use this to update the weights and get a better model.**
- **Continue this until we have a model that is good.**

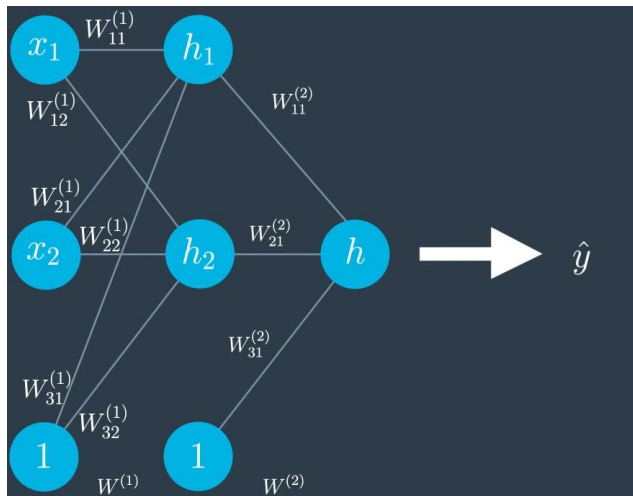
Calculation of the derivative of the sigmoid function

Recall that the sigmoid function has a beautiful derivative, which we can see in the following calculation. This will make our backpropagation step much cleaner.

$$\begin{aligned}\sigma'(x) &= \frac{\partial}{\partial x} \frac{1}{1+e^{-x}} \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\ &= \sigma(x)(1 - \sigma(x))\end{aligned}$$

Example :

Feedforward



$$h_1 = W_{11}^{(1)} x_1 + W_{21}^{(1)} x_2 + W_{31}^{(1)}$$

$$h_2 = W_{12}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{32}^{(1)}$$

$$h = W_{11}^{(2)} \sigma(h_1) + W_{21}^{(2)} \sigma(h_2) + W_{31}^{(2)}$$

$$\hat{y} = \sigma(h)$$

$$\hat{y} = \sigma \circ W^{(2)} \circ \sigma \circ W^{(1)}(x)$$