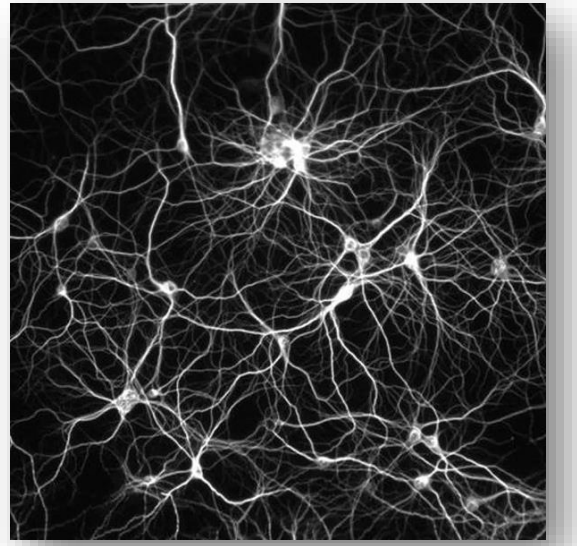
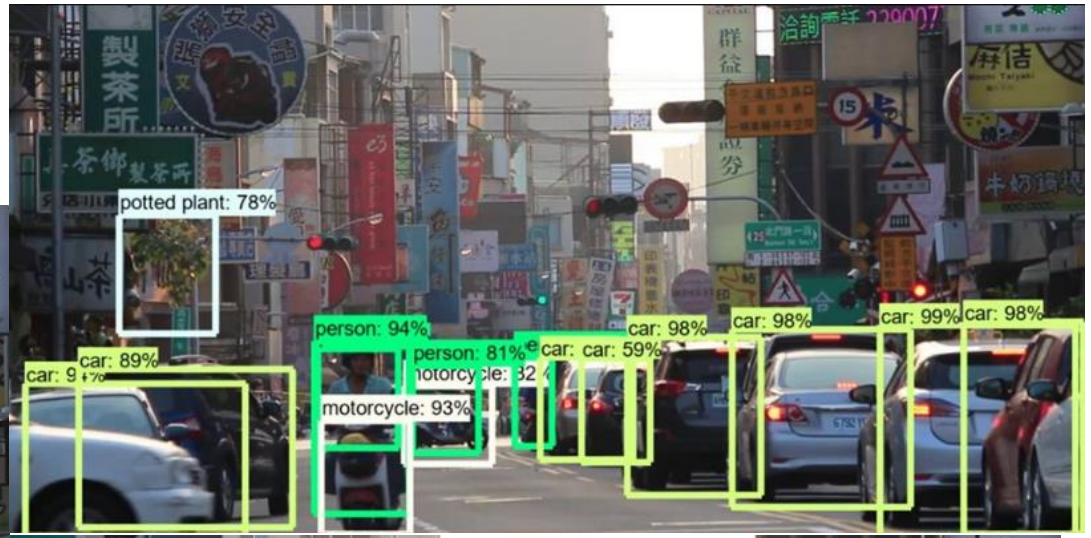
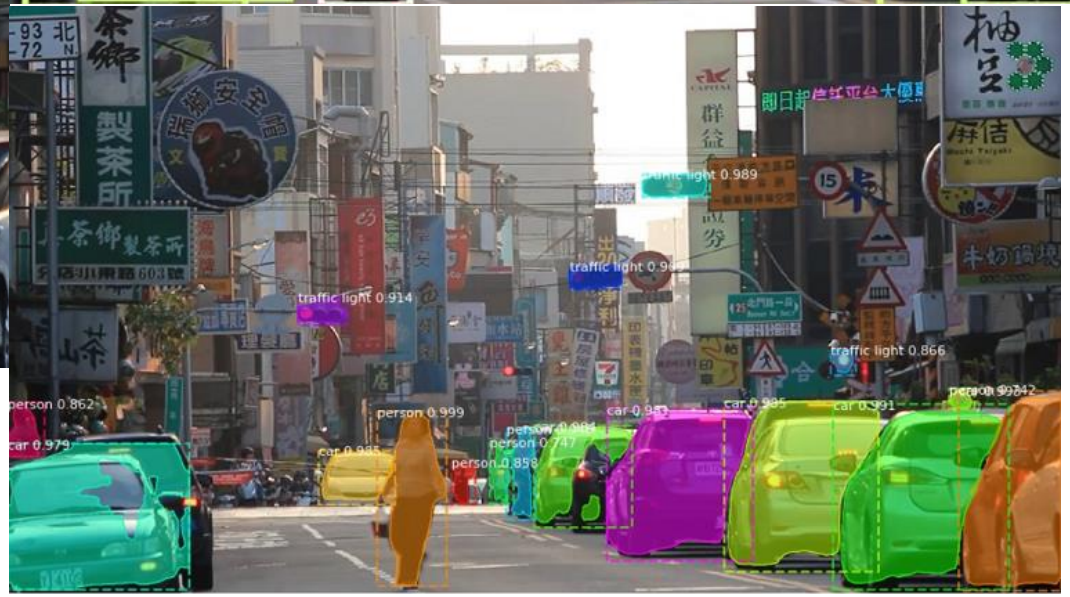
Introduction to Neural Networks 1

20210323



Module Introduction





Deep learning is just another term for using deep neural networks to solve problems and it's become really important for self-driving cars.

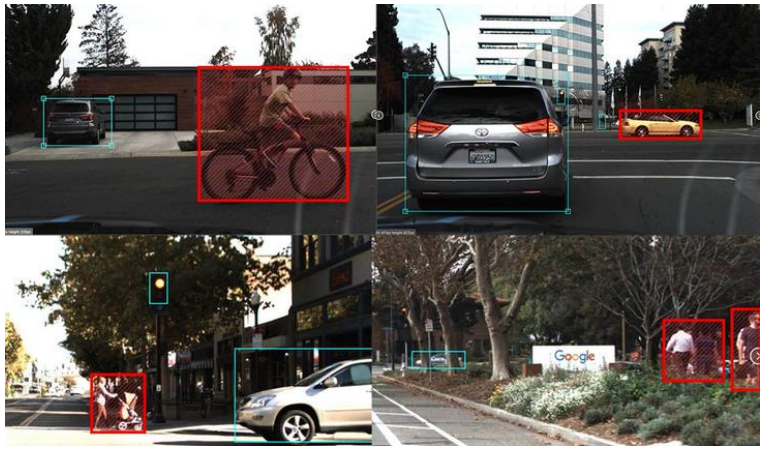
The Value of Machine Learning

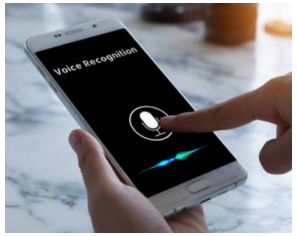
Data Analysis

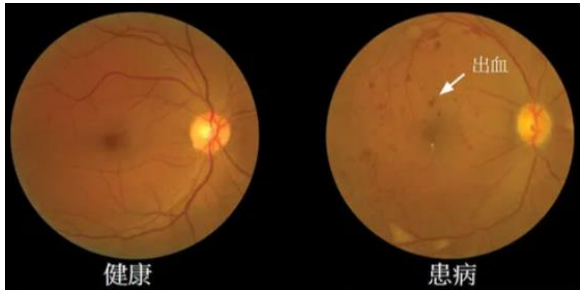
Beyond human comprehension

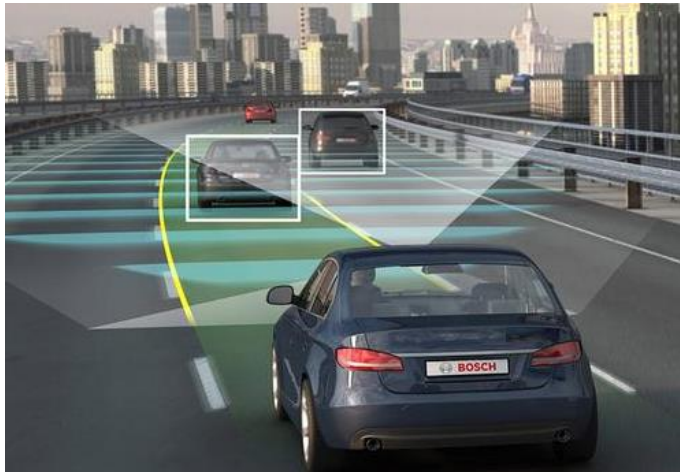
Towards Automation

Real-World Machine Learning









Teachable Machine

有戴口罩

20 Image Samples

Webcam | Upload

沒戴口罩

20 Image Samples

Webcam | Upload

Add a class

Training

Model Trained

Advanced

Epochs: 50

Batch Size: 16

Learning Rate: 0.001

Reset Defaults

Under the hood

Preview

Export Model

Input ON Webcam

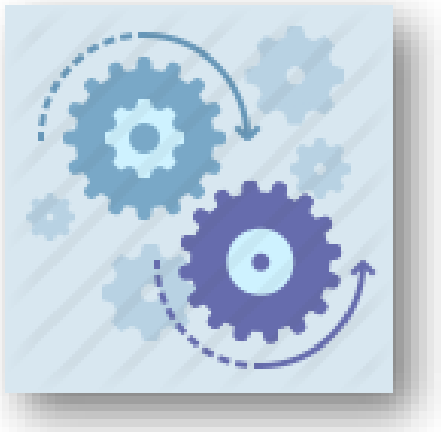
Switch Webcam

Output

有戴口罩 100%

沒戴口罩

Teachable Machine



How Does App Work?

How does image recognition work?

How to program the App?

Rule-based

Designers and engineers develop flow and logic

```
if object.facemask is null:
  do x
if object.mouth is not null:
  do y
if object.facemask.width > 2:
  do z
...
```

Frame the goal of the product

Refine until goals are met

Rule-based

Designers and engineers develop flow and logic

```
if object.facemask is null:
  do x
if object.mouth is not null:
  do y
if object.facemask.width > 2:
  do z
...
```

Frame the goal of the product

Refine until goals are met

Input (know)

Process Function (know)

Output

Rule-based

Designers and engineers develop flow and logic

```
if object.facemask is null:
  do x
if object.mouth is not null:
  do y
if object.facemask.width > 2:
  do z
...
```

Frame the goal of the product

Refine until goals are met

Input (know)

Process Function (know)

Output

Rules

Data

Traditional Programming

Anwsers

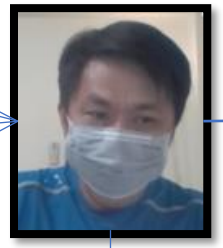
Machine Learning

Learning a model from data

Class : wear mask

Train a model using examples

MASK RECOGNITION



Refine until goals are met

Frame the goal of the product

Machine Learning

Learning a model from data

Class : wear mask

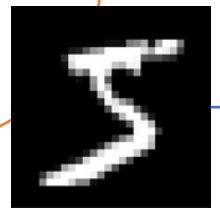
Train a model using examples

MASK RECOGNITION

Frame the goal of the product

Refine until goals are met

Input (know)



Process Function (unknow)

Output (know)

5

Machine Learning

Class : wear mask

Learning a model from data

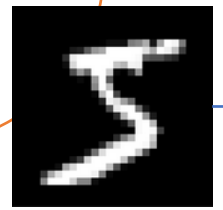
Train a model using examples

MASK RECOGNITION

Frame the goal of the product

Refine until goals are met

Input (know)



Process Function (unknow)

Output (know)

5

Anwsers

Data

Machine Learning

Rules

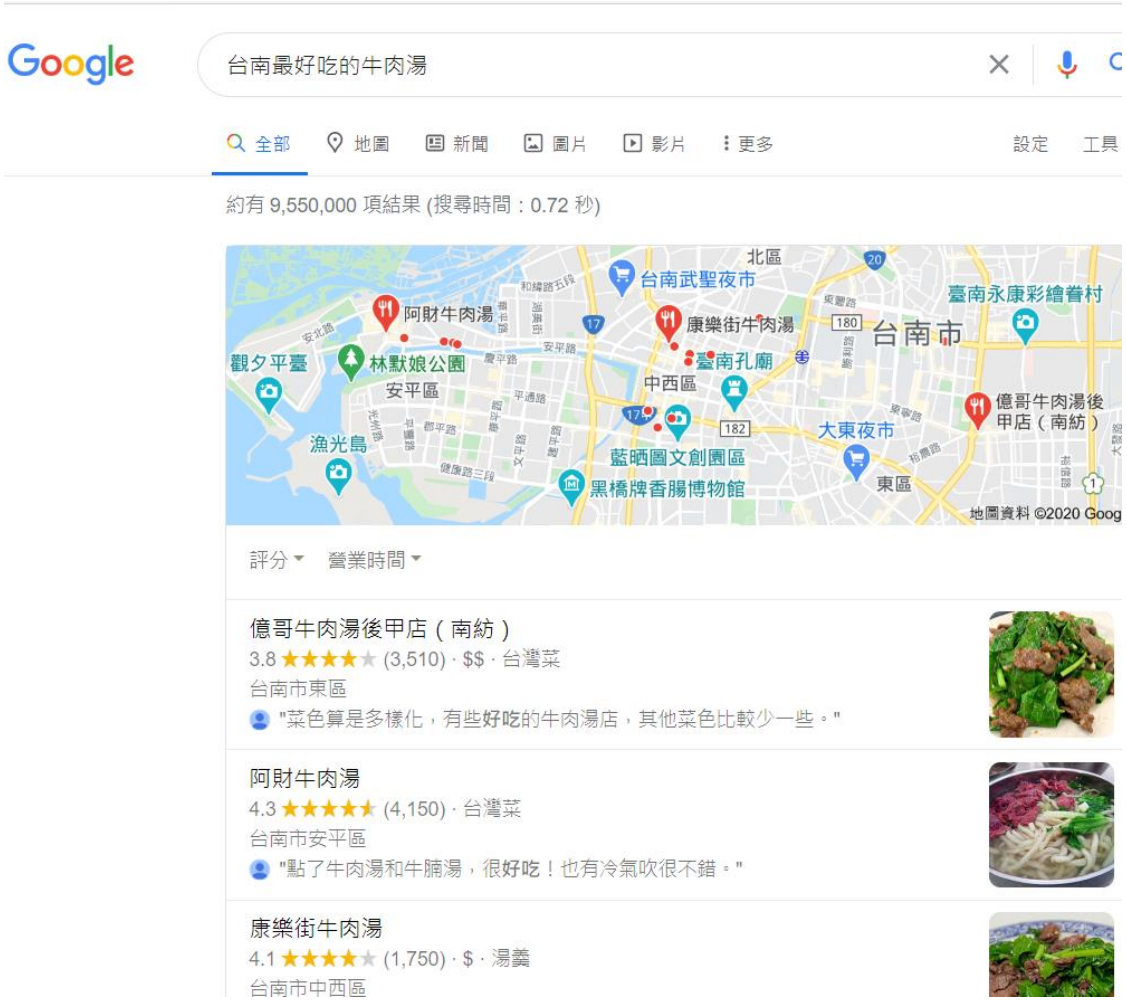
Which Approach to Use?

facial recognition unlocking system

Which Approach to Use?

Sorting for Apple Music playlists.

Which Approach to Use?



Web Searching

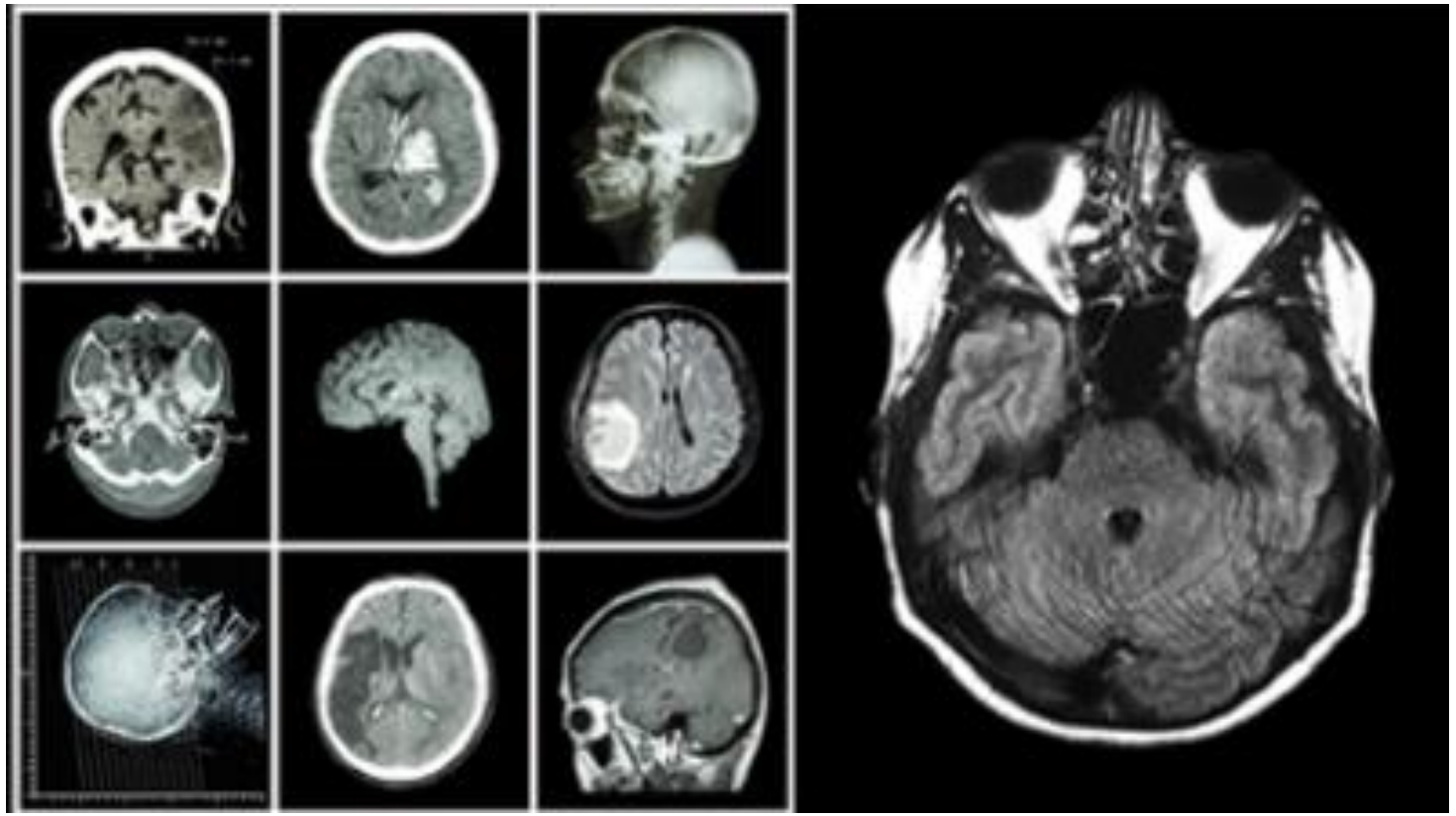
Which Approach to Use?

Detect Potential Repayment Capacity Of User

Which Approach to Use?

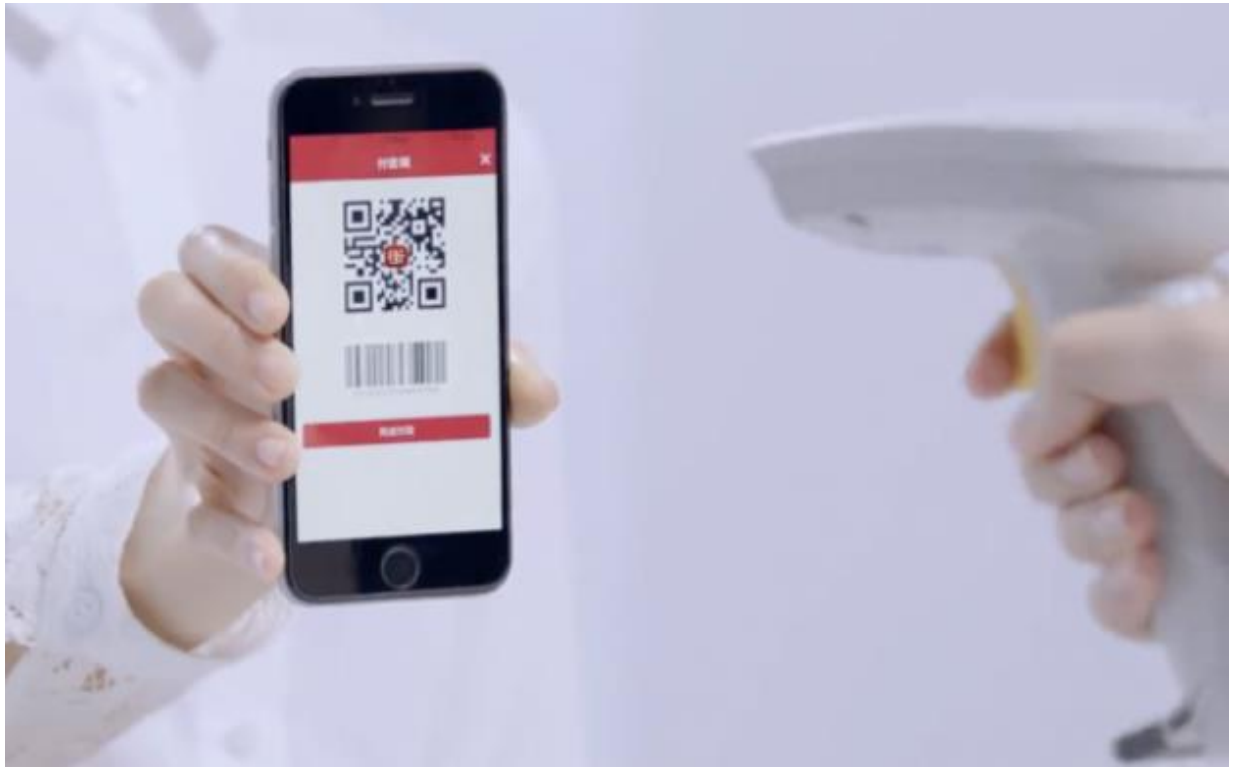
House Price Forecast

Which Approach to Use?



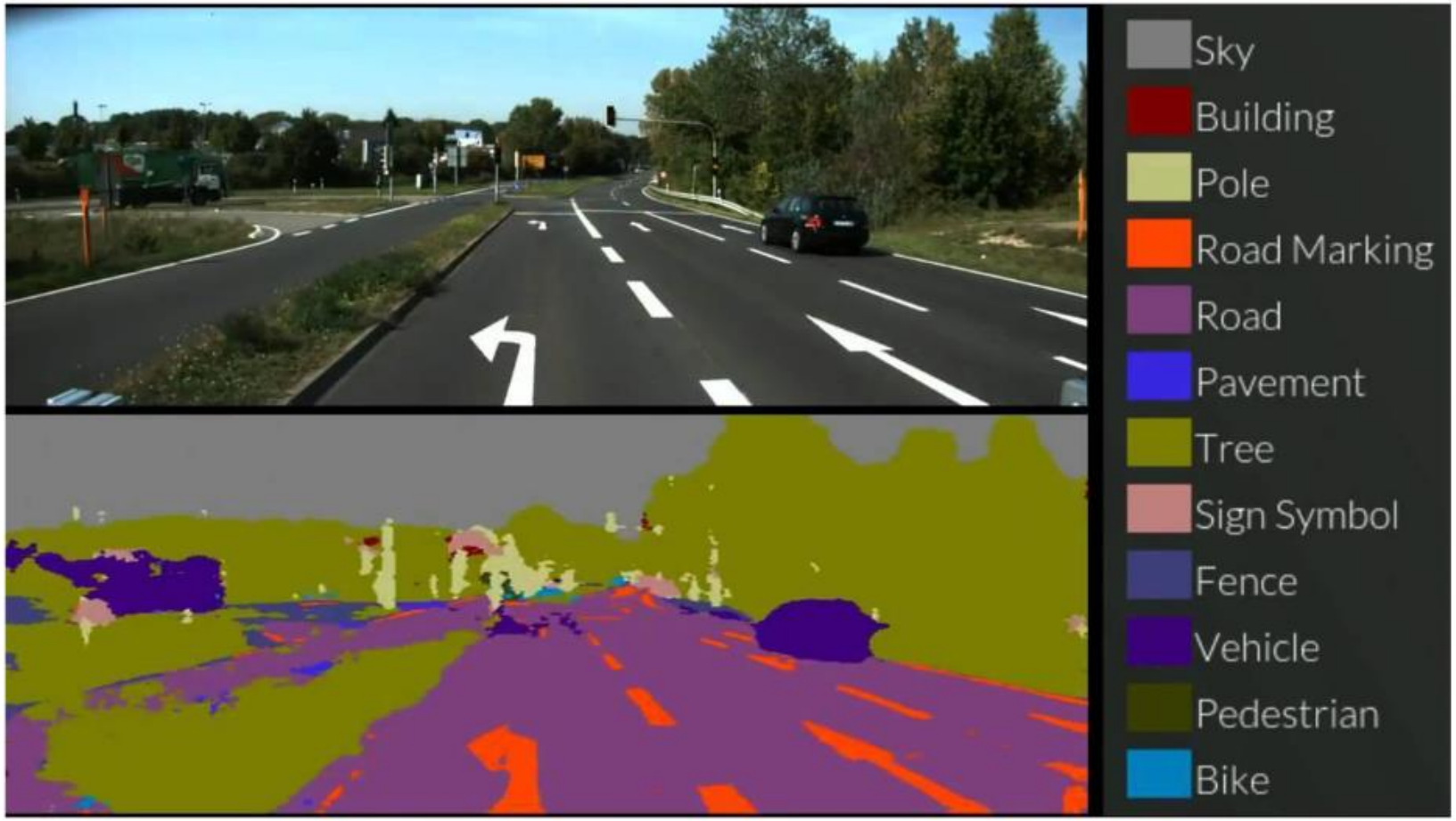
Medical Diagnostic Imaging Analysis

Which Approach to Use?



Online Payment Transaction

Which Approach to Use?



TYPES OF MACHINE LEARNING

Supervised learning

Supervised learning, in which the machine is presented with input data and a desired output, and the goal is to learn from those training examples in such a way that meaningful predictions can be made for data that the machine has never observed before.

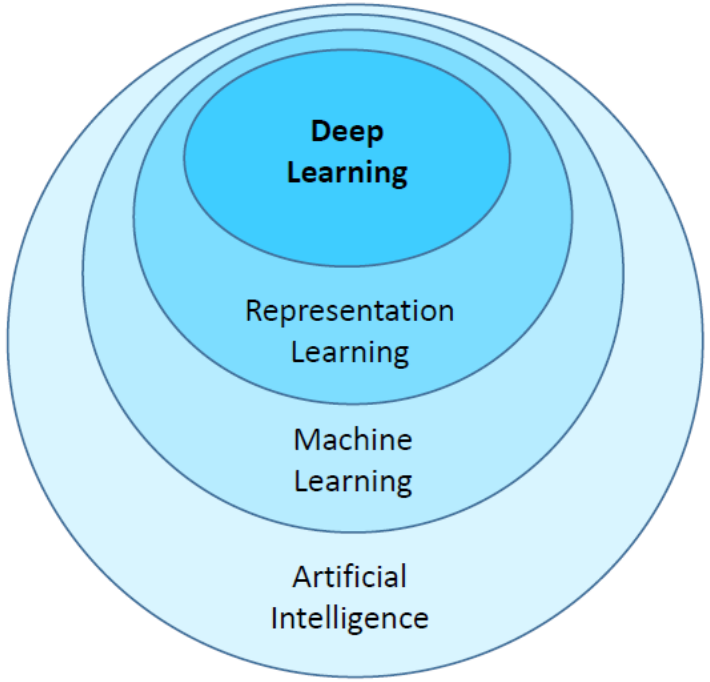
Unsupervised learning

Unsupervised learning, in which the machine is presented with input data only, and the machine has to subsequently find some meaningful structure
by itself, with no external supervision or input.

Reinforcement learning

Reinforcement learning, in which the machine acts as an agent, interacting with the environment. The machine is provided with "rewards" for behaving in a desired manner, and "penalties" for behaving in an undesired manner. The machine attempts to maximize rewards by learning to develop its behavior accordingly.

Introduction to Deep Learning & Self-Driving Car



Deep Learning is Representation Learning
(aka Feature Learning)

Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs.

A special case of machine learning is deep learning, where the algorithms used mimic the human brain, by using so-called artificial neural networks (ANN). To quote the authors in:

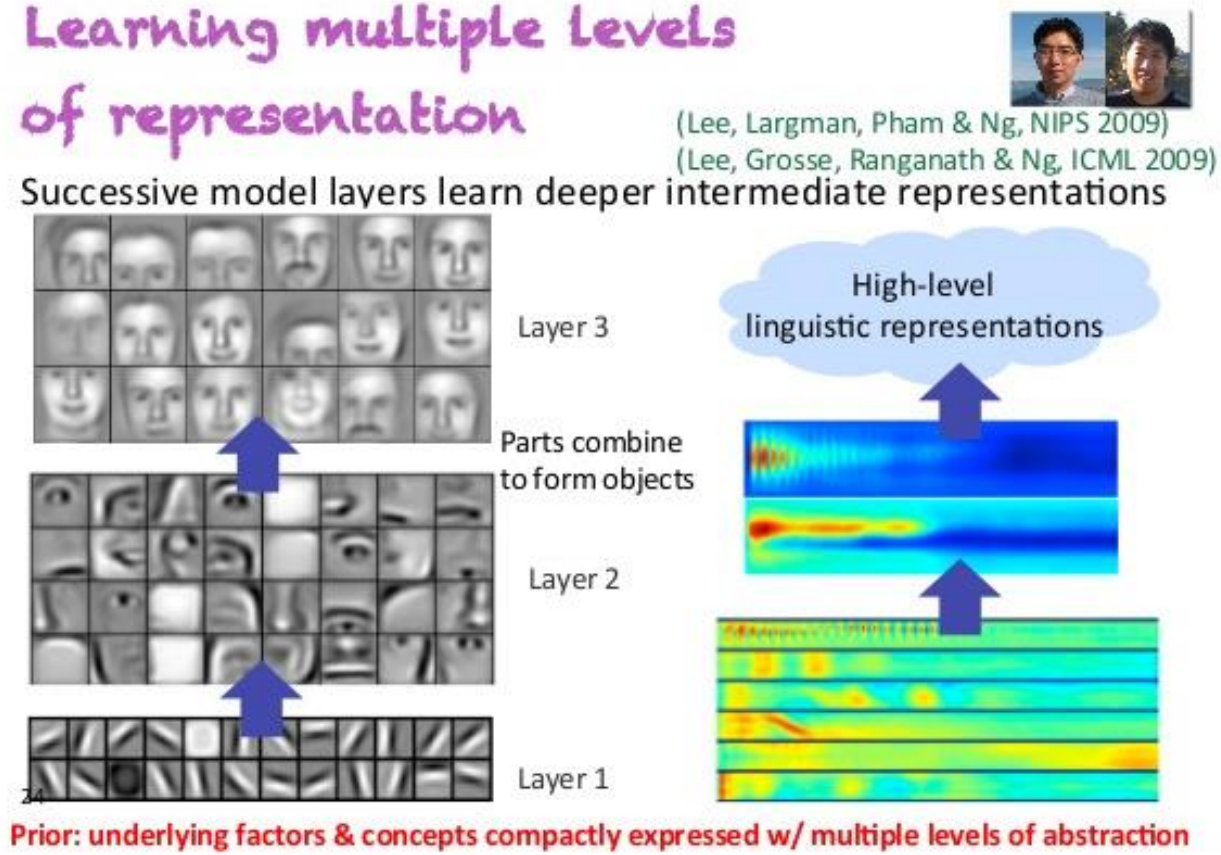
"*From a layman's standpoint, deep learning is a high performance, dynamic way of computerized decision-making that can learn features, objects, and patterns automatically and more accurately with the more (and better quality) data you give it. A deep learning system identifies and classifies patterns utilizing a set of analytical layers. The first layer does a relatively primitive task, such as identifying the edge of an image. It then sends the output to the next layer, which does a slightly more complex task, such as identifying the corner of the image. This process continues through each successive layer until every feature is identified. In the final, deepest layer, the system should reliably and quickly recognize the pattern.*"

Representation learning

In machine learning, **feature learning** or **representation learning** is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task.

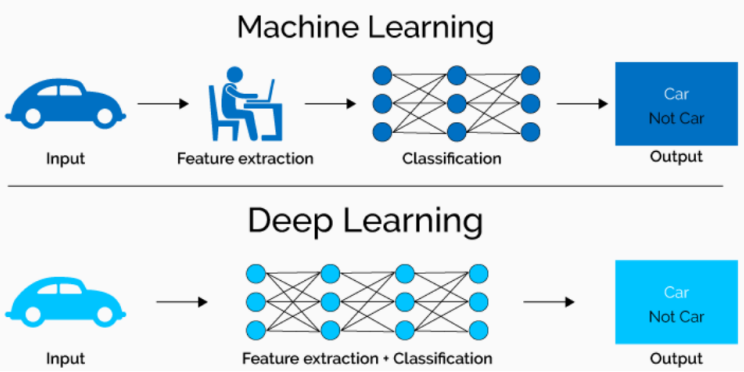
Intelligence: Ability to accomplish complex goals.

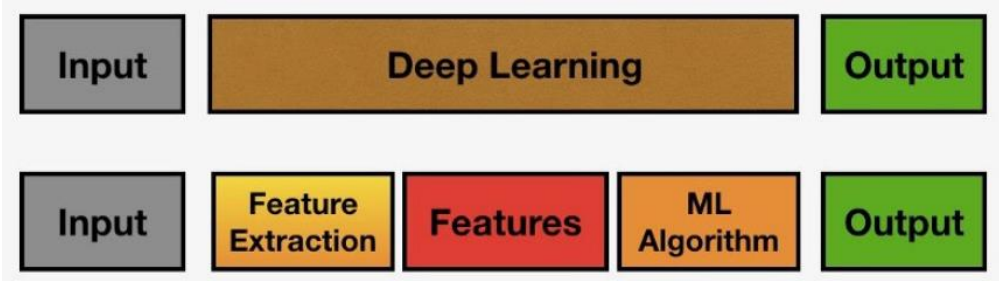
Understanding: Ability to turn complex information to into simple, useful information.

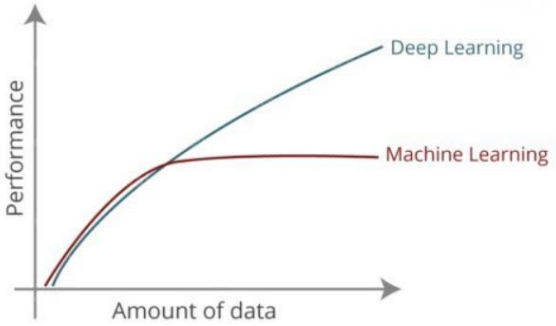


Machine learning VS. Deep learning

The main difference between machine learning and deep learning is the depth to which the system can autonomously teach itself. Where as machine learning uses features from input (from training data) and makes predictions based on a single or a few layers of nodes, a deep neural network contains many hidden layers that adds new features and exceeds human coding capacity. This makes deep learning more powerful for complex computing tasks such as object recognition.

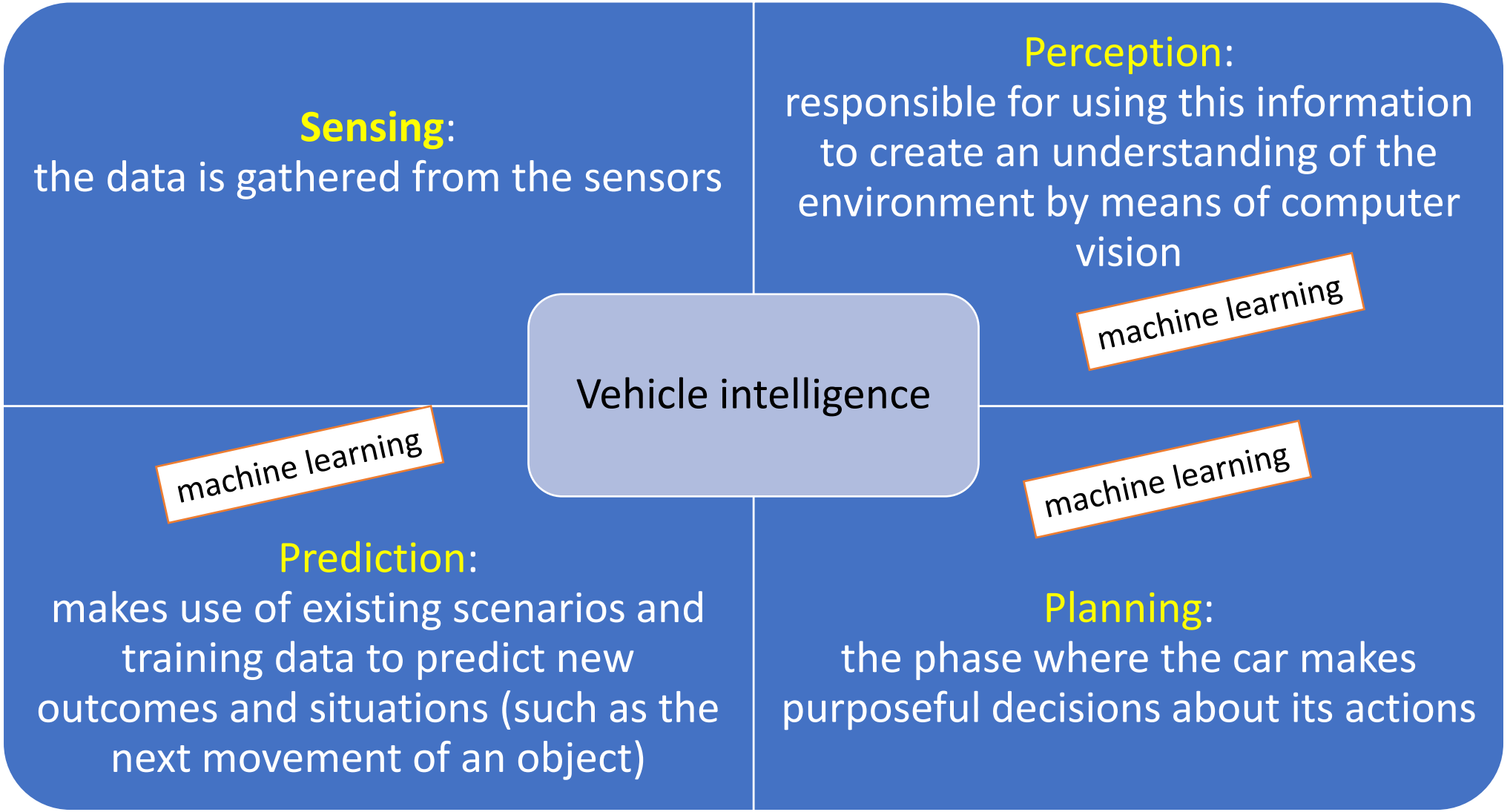






Block Diagram Comparing Deep Learning Model vs Machine Learning Model

Introduction to Deep Learning & Self-Driving Car



REFERENCE : IS DEEP LEARNING REALLY THE SOLUTION FOR EVERYTHING IN SELF-DRIVING CARS?(HTTPS://WWW.AUTOMOTIVE-IQ.COM/)

Introduction to Deep Learning & Self-Driving Car

Is deep learning really the solution for everything?

Why Deep Learning?

Deep learning Perform really well with huge data.

Deep learning: Set of techniques that have worked well for AI techniques in recent years due to advancement in research and GPU capabilities. Self Driving car are Systems that can utilize these.

How deep learning is used and what are its advantages

The main application of deep learning within the automotive domain is that of advanced computer vision and perception. Visual tasks, including, but not limited to lane detection, pedestrian detection, road signs recognition and blind-spot monitoring are handled more effectively with deep learning.

Shortcomings and challenges of deep learning

- Processing power: Since deep learning requires such a high level of computing power, a very powerful “brain” is needed to handle the big data capabilities and processing requirements.
- Available training data: As noted before, an end-to-end learning system especially, requires to be fed a huge amount of training data, in order to predict as many driving scenarios as possible and to fulfil a minimum safety requirement.
- Safety: One of the main challenges with safety of deep neural networks is the fact that they are unstable under so-called adversarial perturbations. For example, minimal modifications in camera images, such are resizing, cropping and the change of lighting conditions might cause the system to misclassify the image. Additionally, in general, safety assurance and verification methods for machine learning are poorly studied. The prevailing automotive safety standard of ISO26262, does not have a way to define safety for self-learning algorithms such as deep learning.

Other AI techniques in self-driving cars

There are a number of other machine learning techniques that are currently being used in ADAS applications.

A Survey of Deep Learning Techniques for Autonomous Driving

- https://arxiv.org/pdf/1910.07738.pdf

Deep Learning for Driving Scene Perception and Localization

Deep Learning for Path Planning and Behavior Arbitration

Motion Controllers for AI-based Self-Driving Cars

Safety of Deep Learning in Autonomous Driving

Data Sources for Training Autonomous Driving Systems

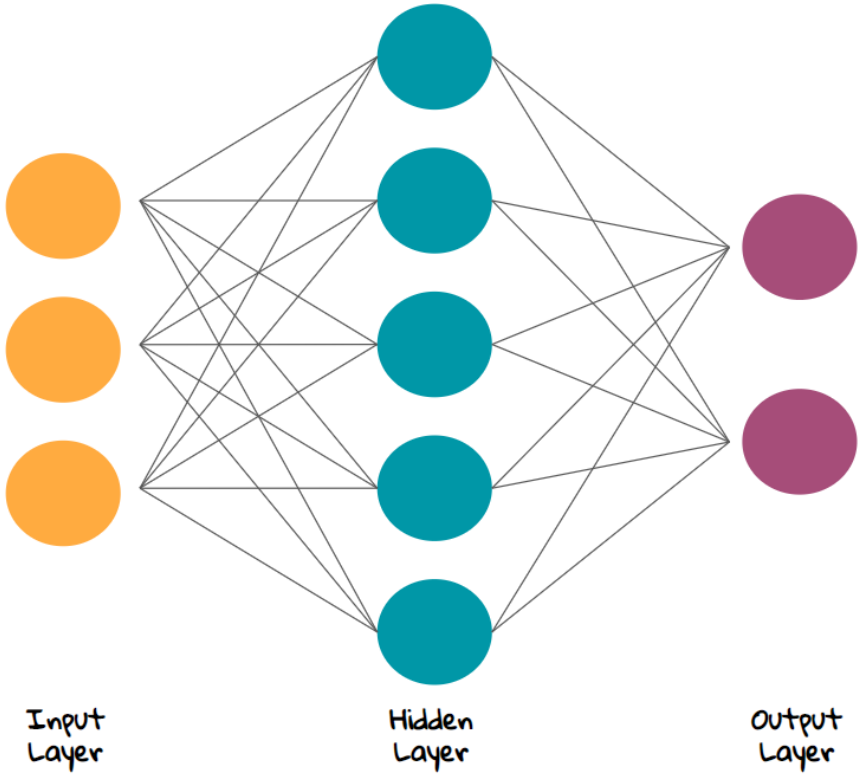
Computational Hardware and Deployment

Starting Machine Learning

- Introduction: The foundation concepts in the fields of machine learning and deep neural network.
- Afterward, we'll build on those concepts and apply them to self-driving car.
- ML is a field of AI that relies on computers to learn about the environment from data, instead of relying on the rule set by computer programmers.
- DP is an approach to ML, that uses deep neural networks. DP uses this one tool to accomplish an amazing array of objectives, from speech recognition, to driving a car.

What is a neural network?

The basic intuitions

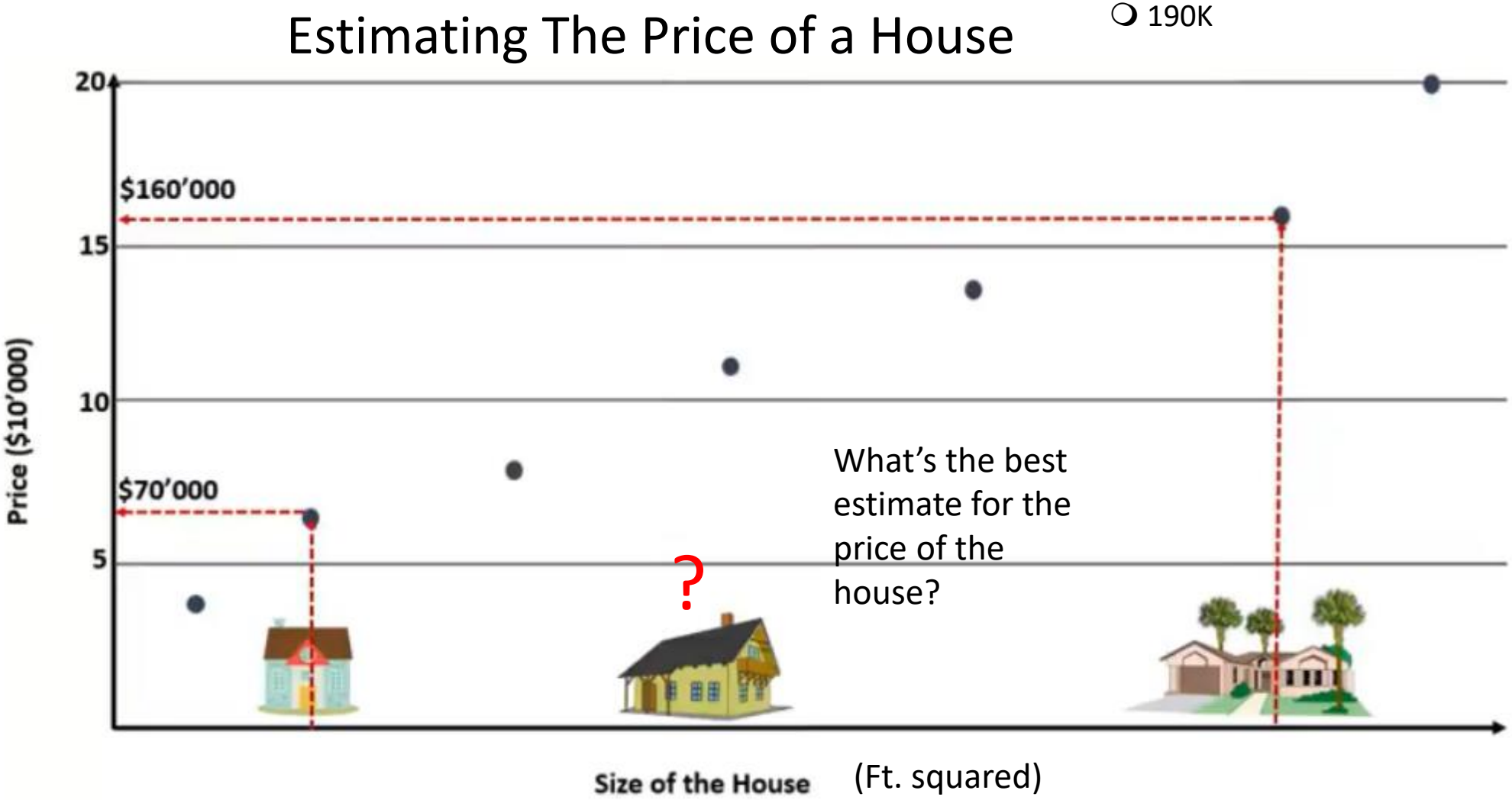


Linear Regression Quiz

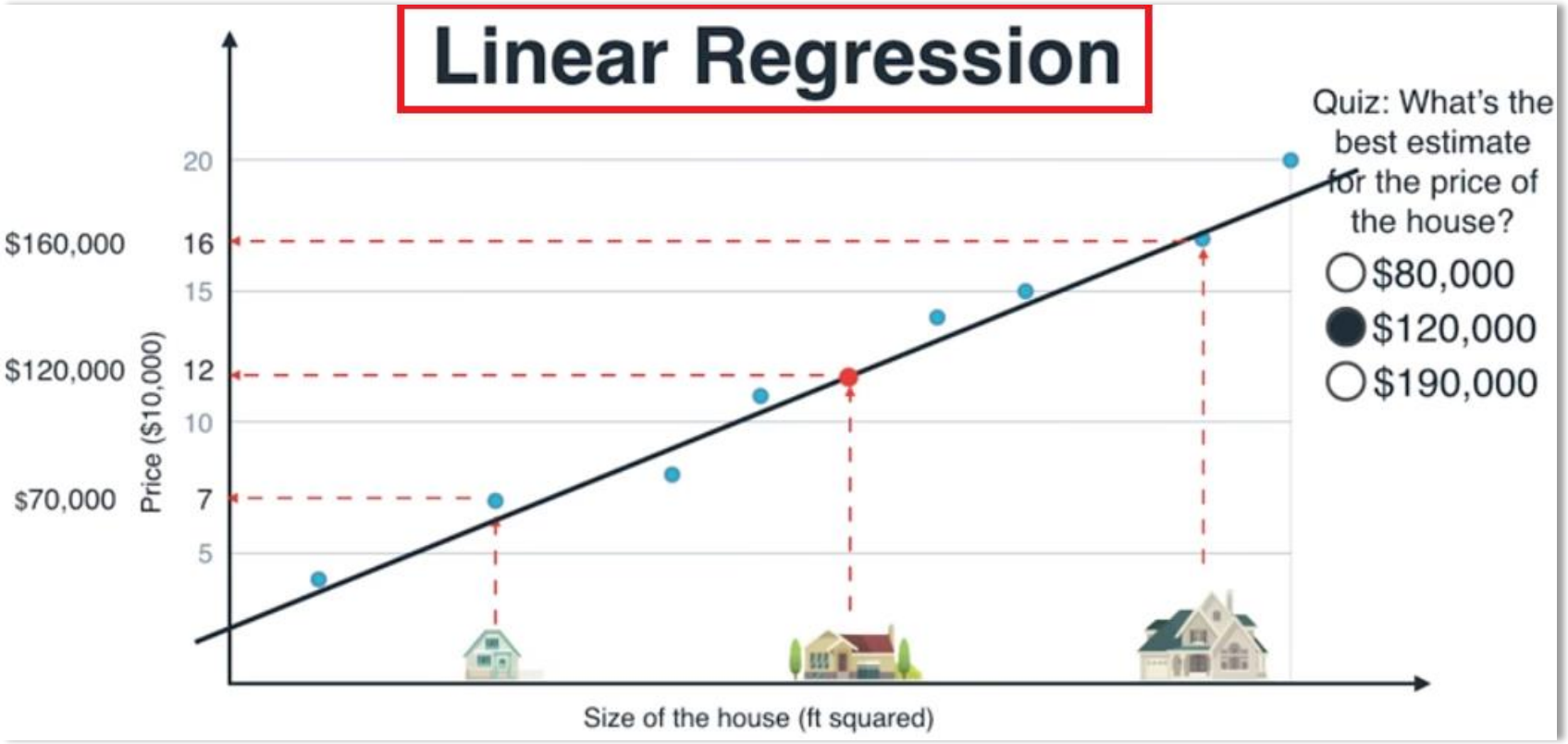
Quiz Question

What's the best estimate for the price of a house?

- ❍ 80K
- ❍ 120K
- ❍ 190K



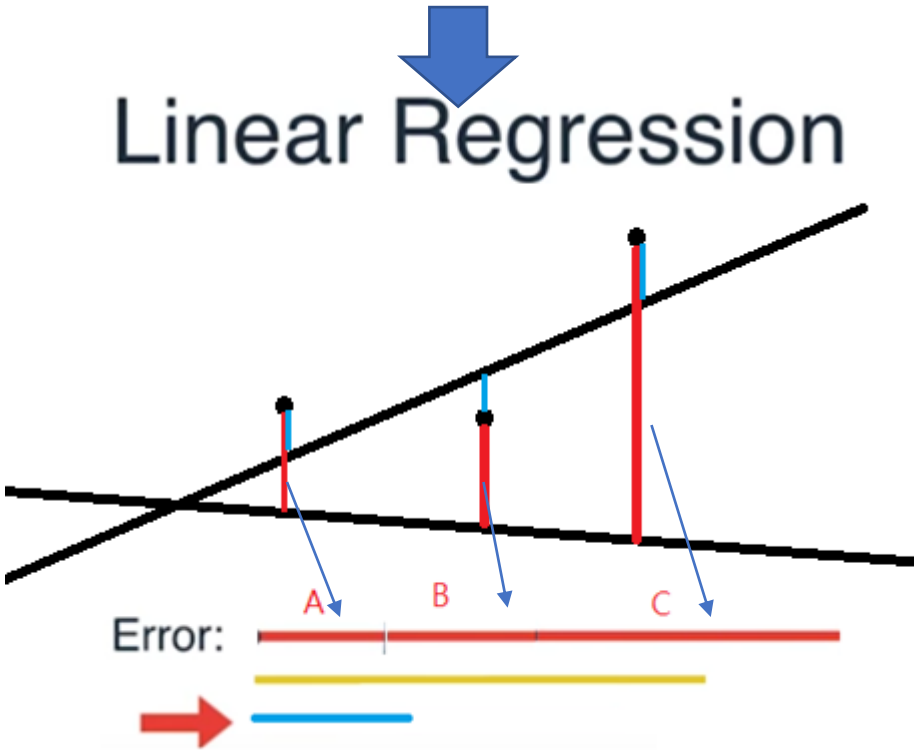
Linear Regression Answer



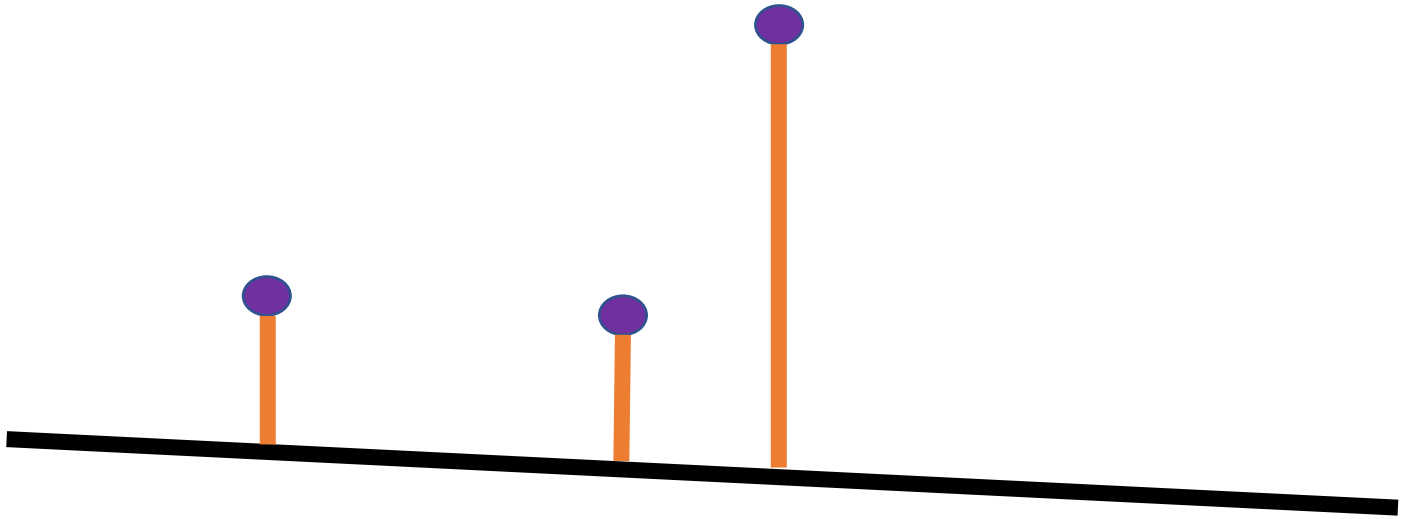
Linear Regression

Linear Regression

How do we find this line?

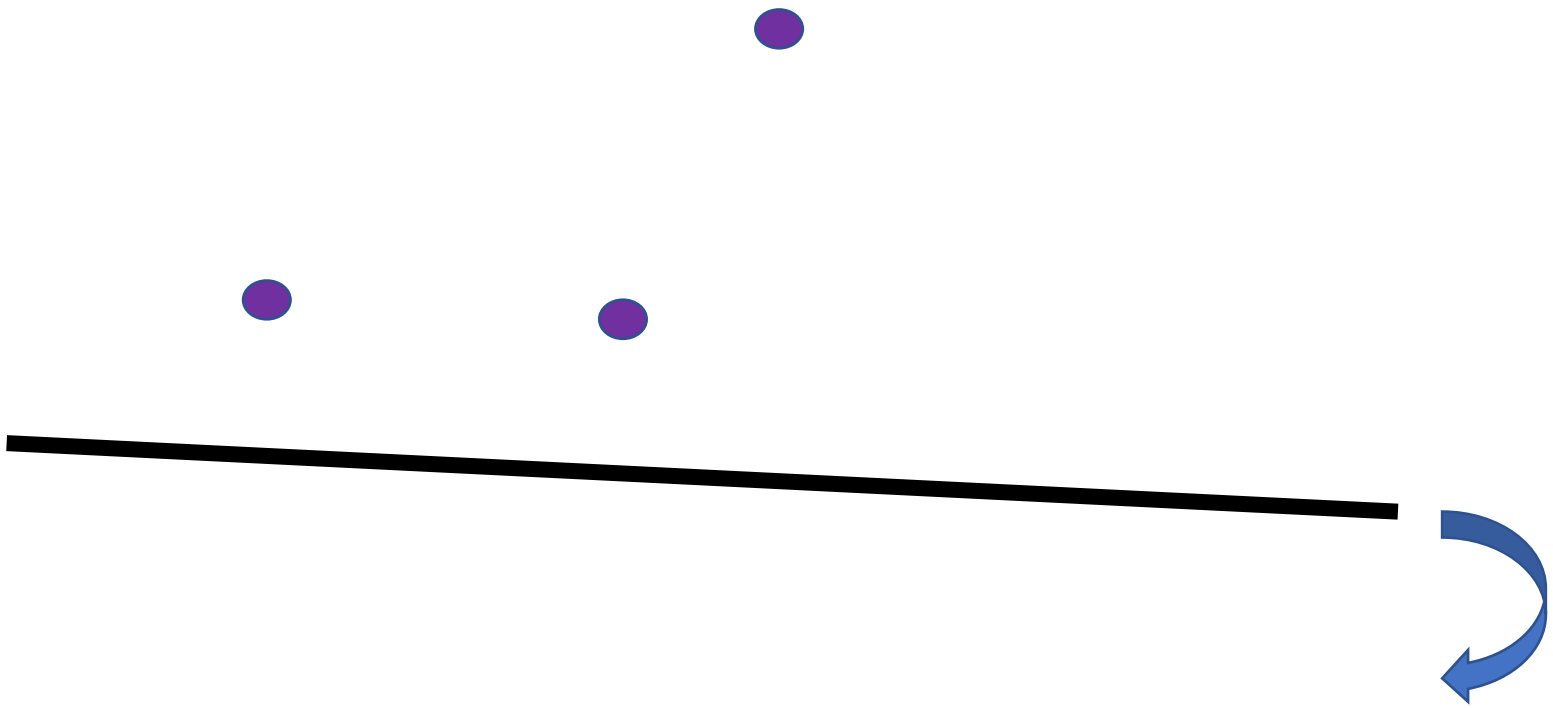


Linear Regression



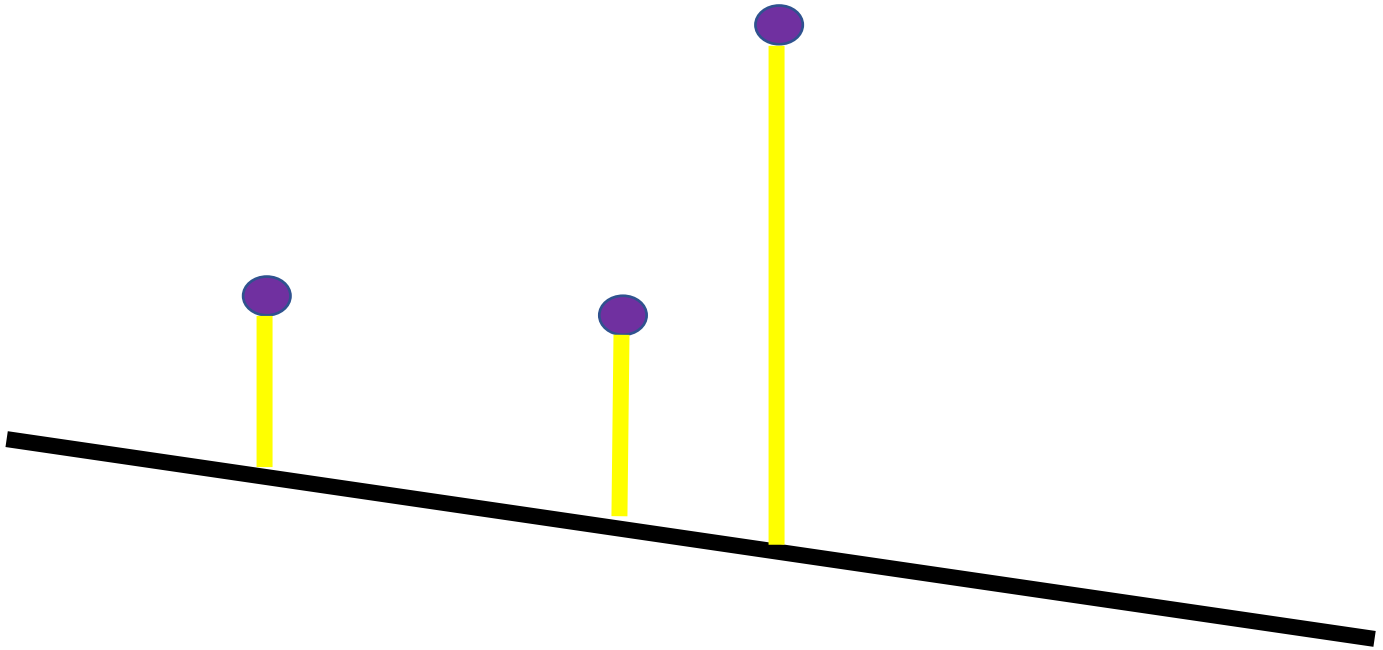
Error:

Linear Regression



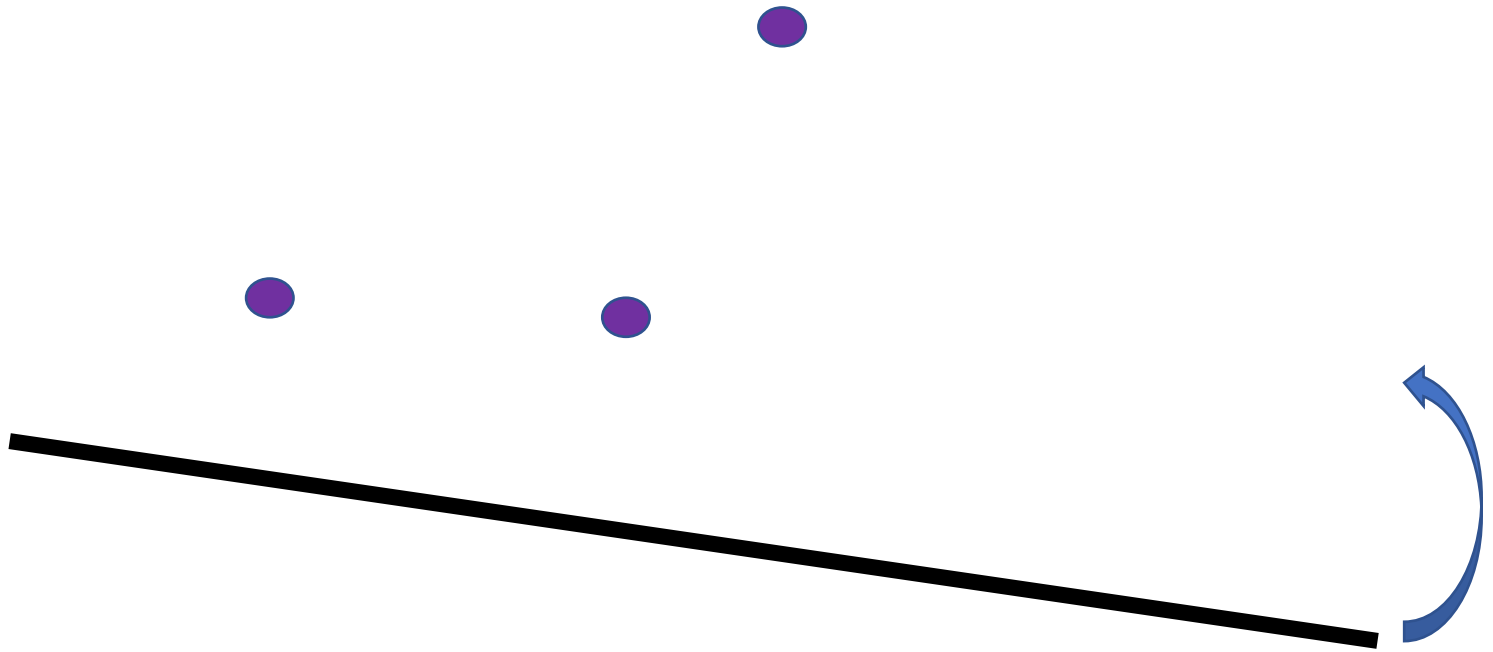
Error:

Linear Regression



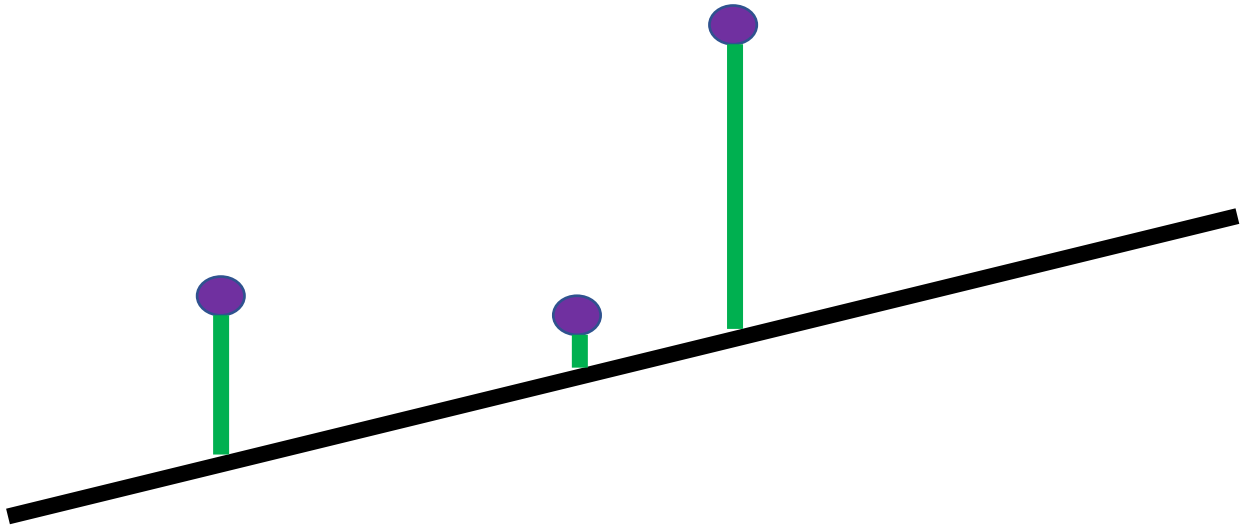
Error:

Linear Regression



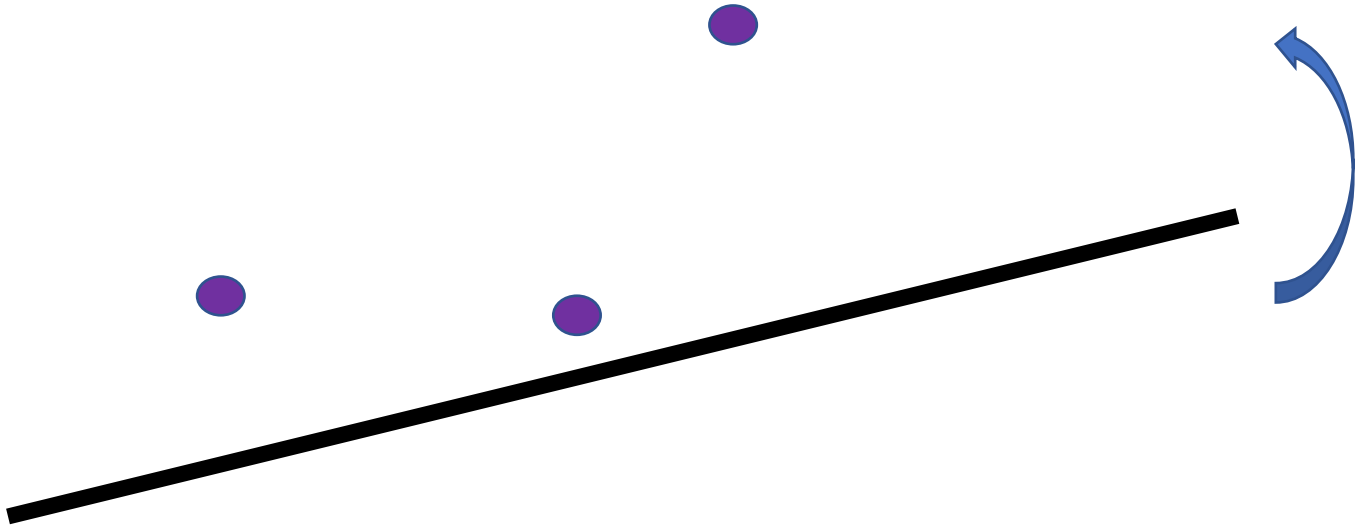
Error:

Linear Regression



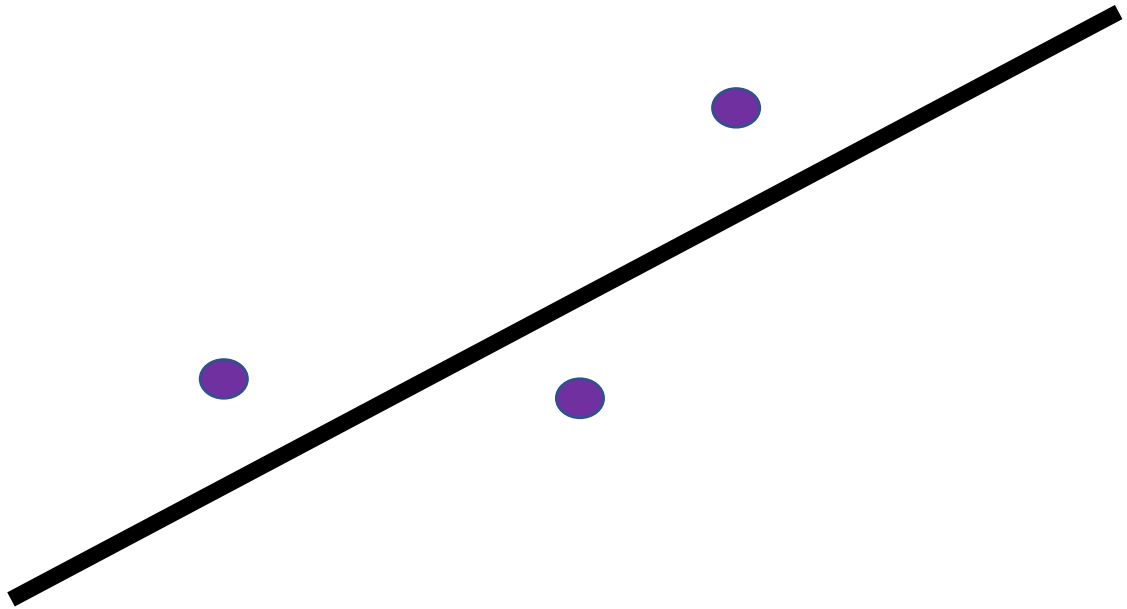
Error:

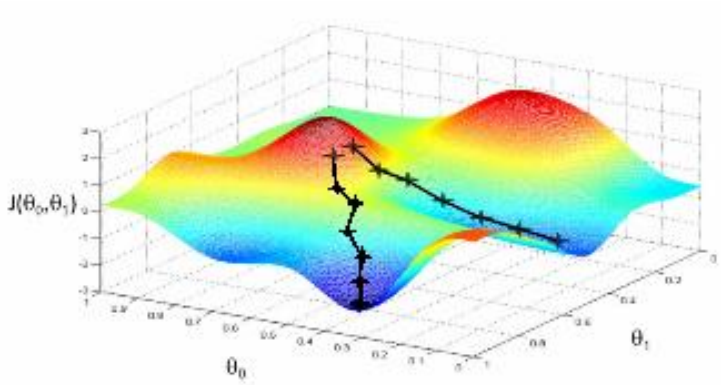
Linear Regression



Error:

Linear Regression



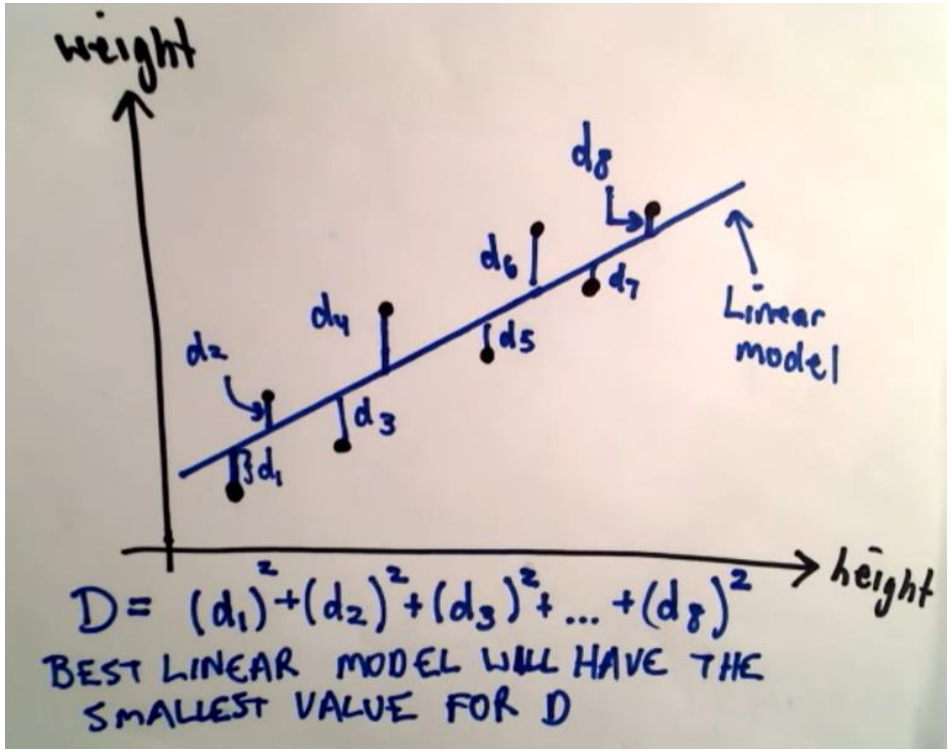


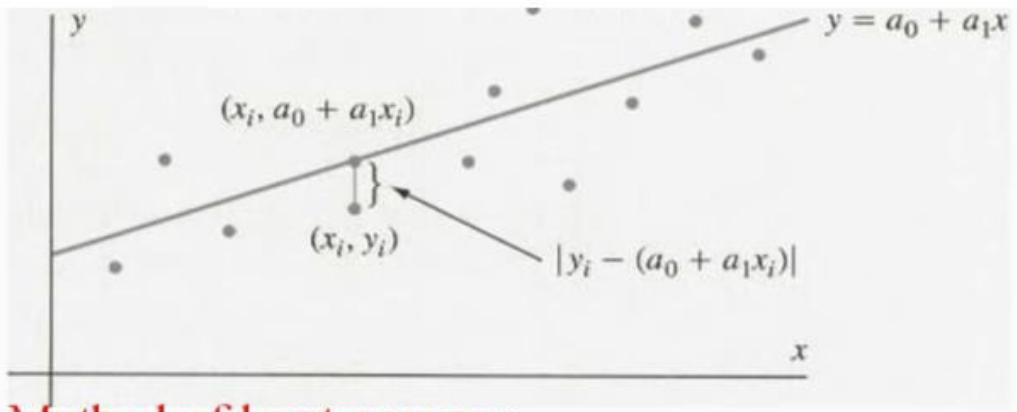
Gradient descent

(Least squares) Minimize the Error

Error:

Supplementary





data pairs:

$x_1 \rightarrow y_1$

$x_2 \rightarrow y_2$

$\vdots$

$x_i \rightarrow y_i$

$\vdots$

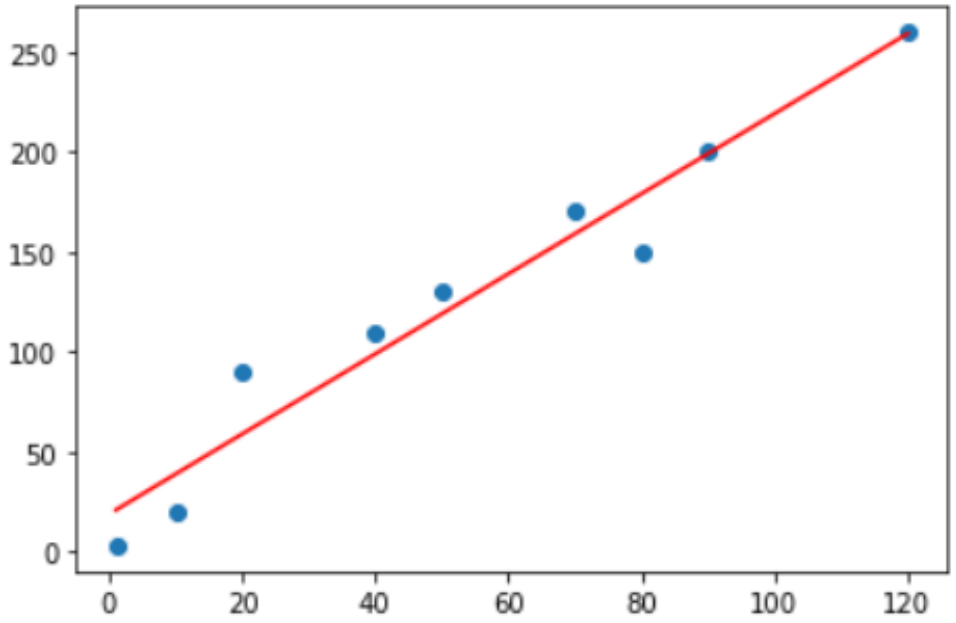
Method of least squares:

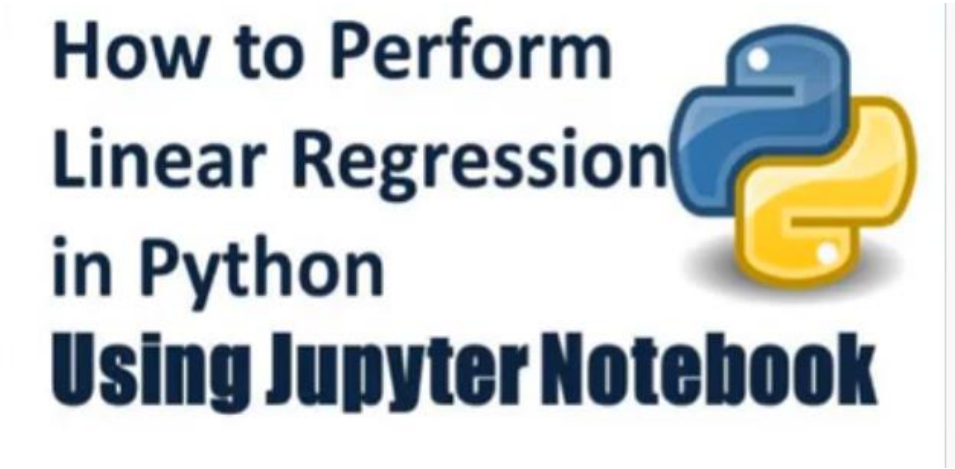
to find the "least-square line" $y = a_0 + a_1x$ to best fit the data in the sense that the error sum of squares

$$E = [y_1 - (a_0 + a_1x_1)]^2 + [y_2 - (a_0 + a_1x_2)]^2 + \cdots + [y_n - (a_0 + a_1x_n)]^2$$

is minimized.

Supplementary





https://www.youtube.com/watch?v=_kb2X85DkKk

Linear to Logistic Regression

Linear Regression helps predict values on a continuous spectrum, like predicting what the price of a house will be.
>How about classifying data among discrete classes?

Logistic Regression : classifying data among discrete classes.
Examples of classification tasks:

>Determining whether a patient has cancer
>Identifying the species of a fish
>Figuring out who's talking on a conference call

Classification problems are important for self-driving cars.

Self-driving cars might need to classify whether an object crossing the road is a car, pedestrian, and a bicycle. Or they might need to identify which type of traffic sign is coming up, or what a stop light is indicating.

Linear regression → logistic regression → neural networks

Linear regression leads to logistic regression and ultimately neural networks, a more advanced classification tool.

Acceptance a University

Test

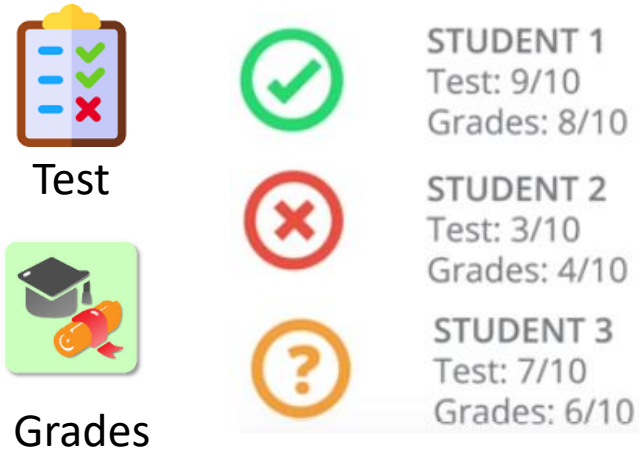
Grades

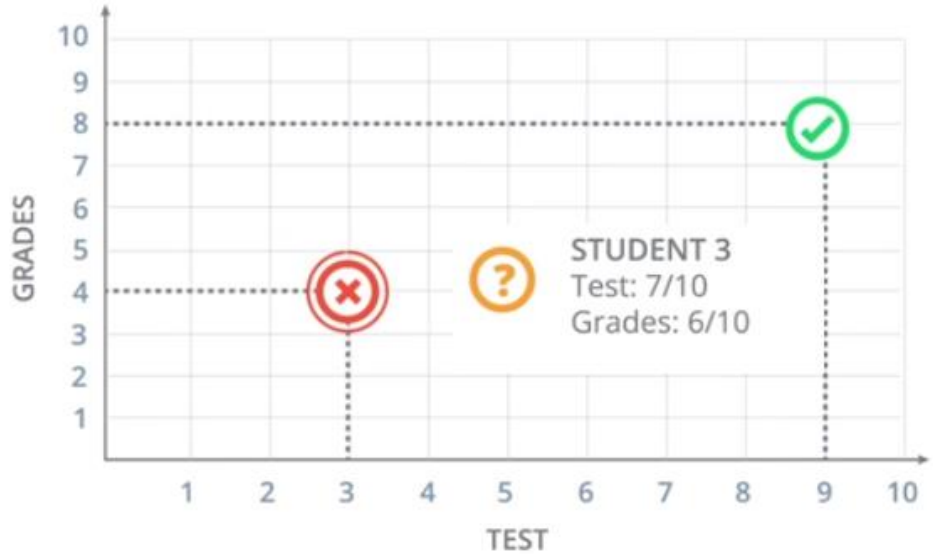
Test

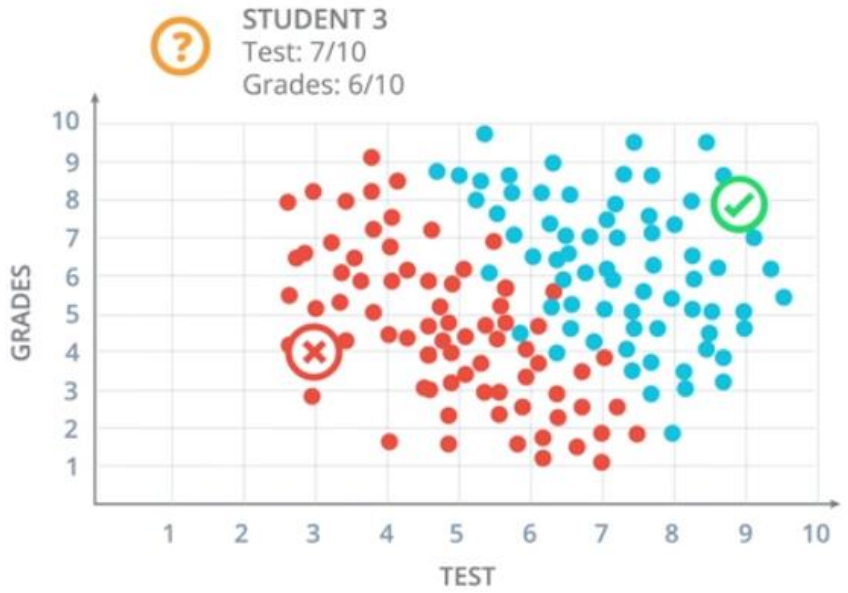
Grades

Student 1	Student 2	Student 3
Test:9/10 Grades:8/10	Test:3/10 Grade:4/10	Test:7/10 Grade:6/10

Classification Problems 1







Quiz Question

Does the student get Accepted?

- ❍ Yes
- ❍ No

Linear Boundaries

BOUNDARY:
A LINE

$2x_1 + x_2 - 18 = 0$

Score =
2*Test + Grades - 18

PREDICTION:
Score > 0: Accept
Score < 0: Reject

Linear Boundaries

BOUNDARY:

A LINE

$w_1x_1 + w_2x_2 + b = 0$

$Wx + b = 0$

$W = (w_1, w_2)$

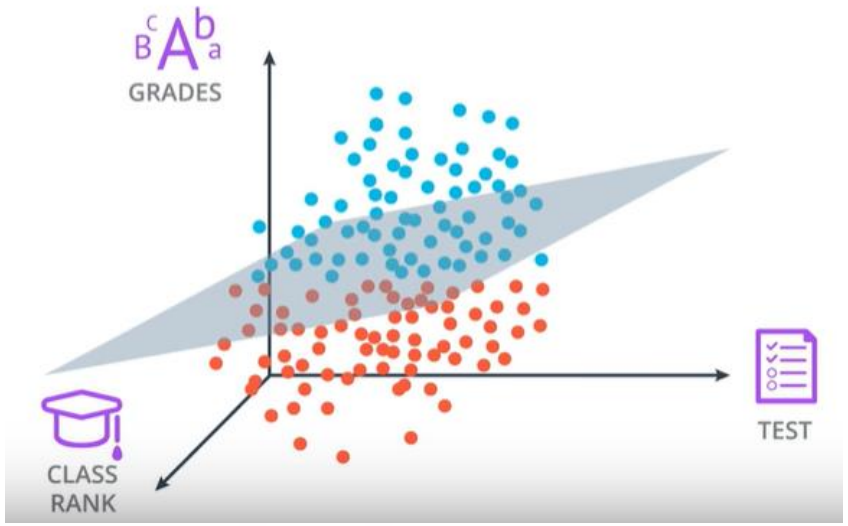
$x = (x_1, x_2)$

y = label: 0 or 1

PREDICTION:

$$\hat{y} = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases}$$

Higher Dimensions



BOUNDARY:
A PLANE

$w_1x_1 + w_2x_2 + w_3x_3 + b = 0$

$Wx + b = 0$

PREDICTION:

$$\hat{y} = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases}$$

	x_1	x_2	x_3		x_n	y
	EXAM 1	EXAM 2	GRADES	...	ESSAY	PASS?
STUDENT 1	9	6	5	...	6	1(yes)
STUDENT 2	8	4	8	...	3	0(no)
...	...	...	...	...	...	
STUDENT n	6	7	2	...	8	1(yes)

n columns

n-dimensional space

$x_1, x_2, \ldots, x_n$

BOUNDARY:

n-1 dimensional hyperplane

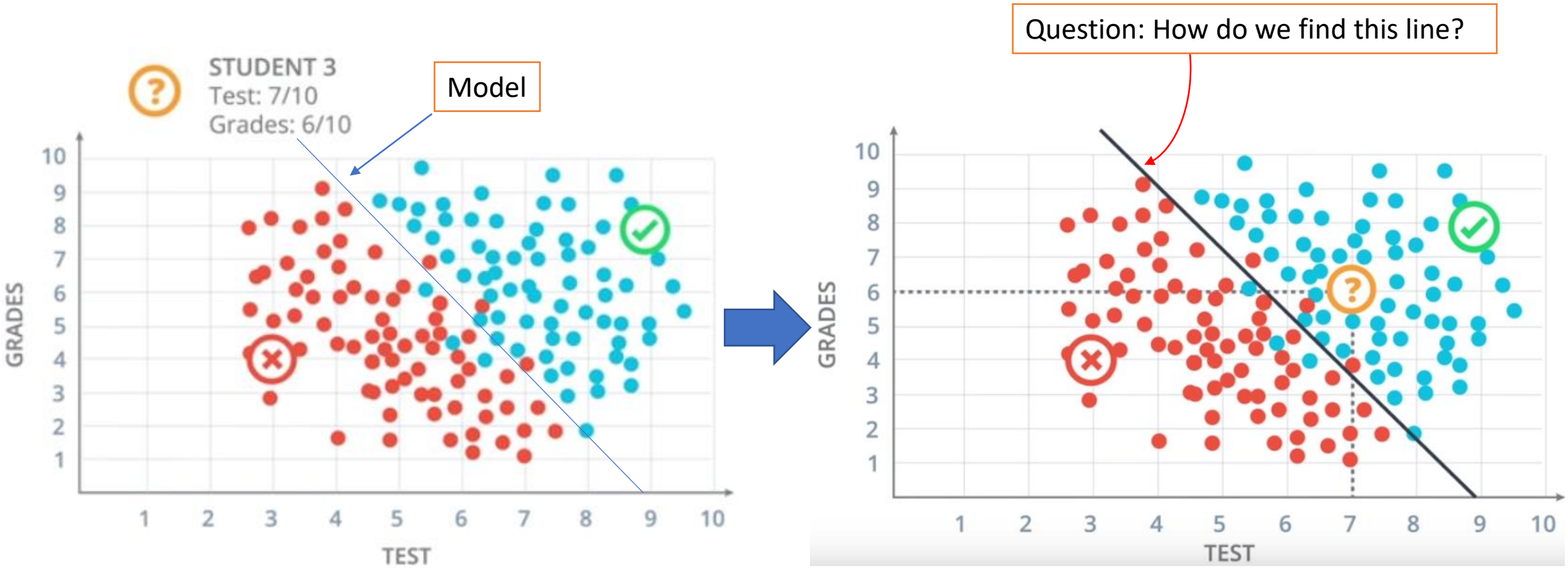
$w_1x_1 + w_2x_2 + w_nx_n + b = 0$

$Wx + b = 0$

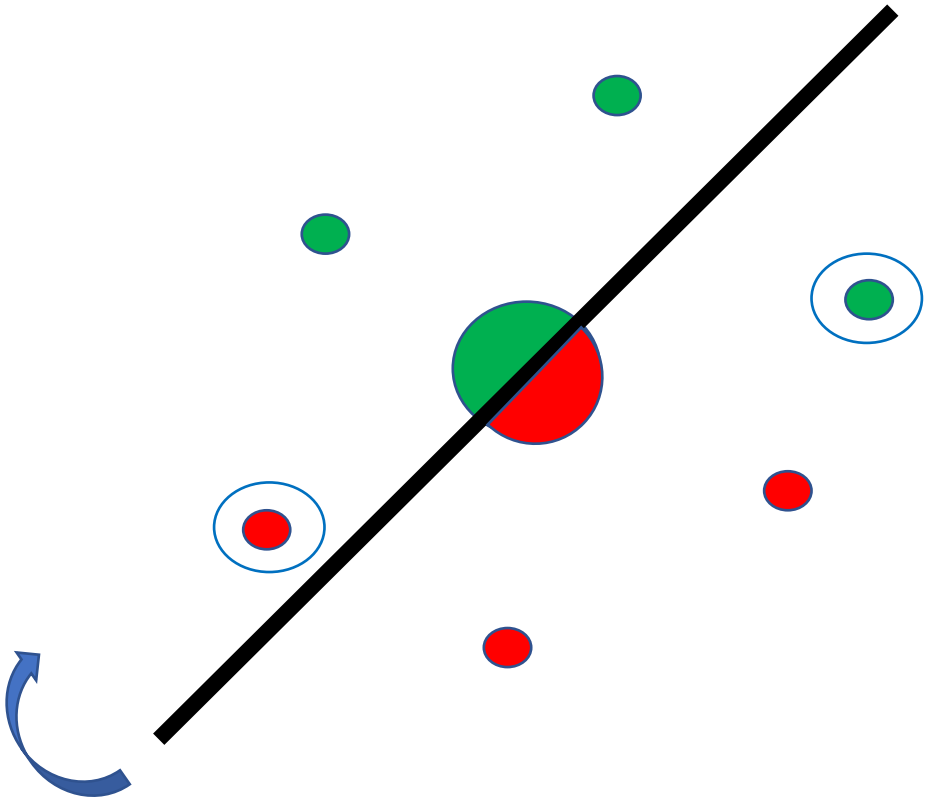
PREDICTION:

$$\hat{y} = \begin{cases} 1 \text{ if } Wx + b \geq 0 \\ 0 \text{ if } Wx + b < 0 \end{cases}$$

Classification Problems 2

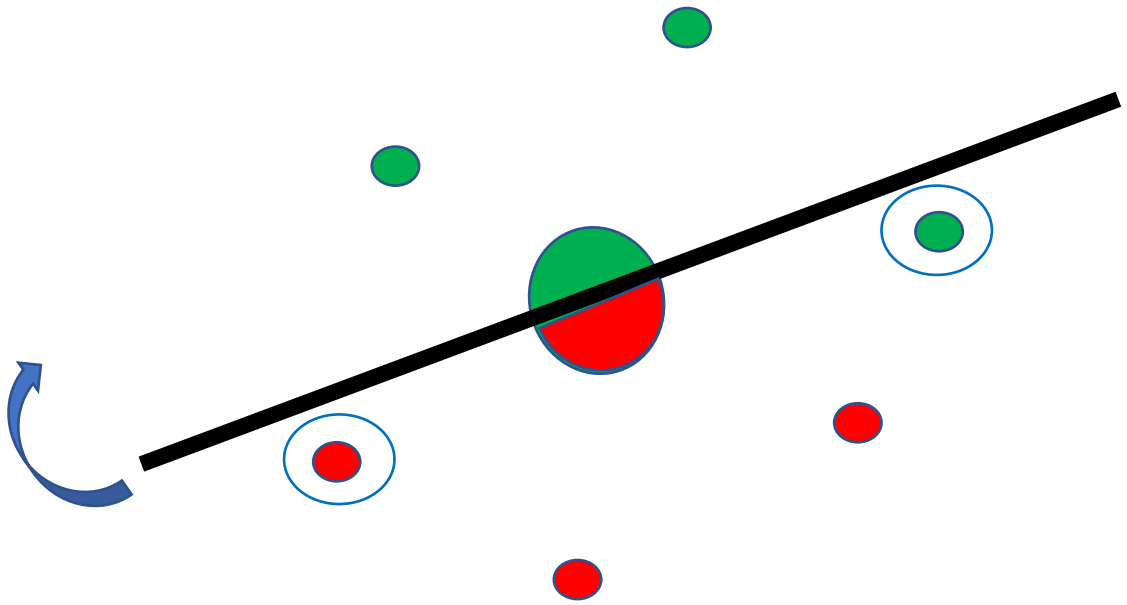


Logistic Regression



$$\frac{\text{Error}}{2}$$

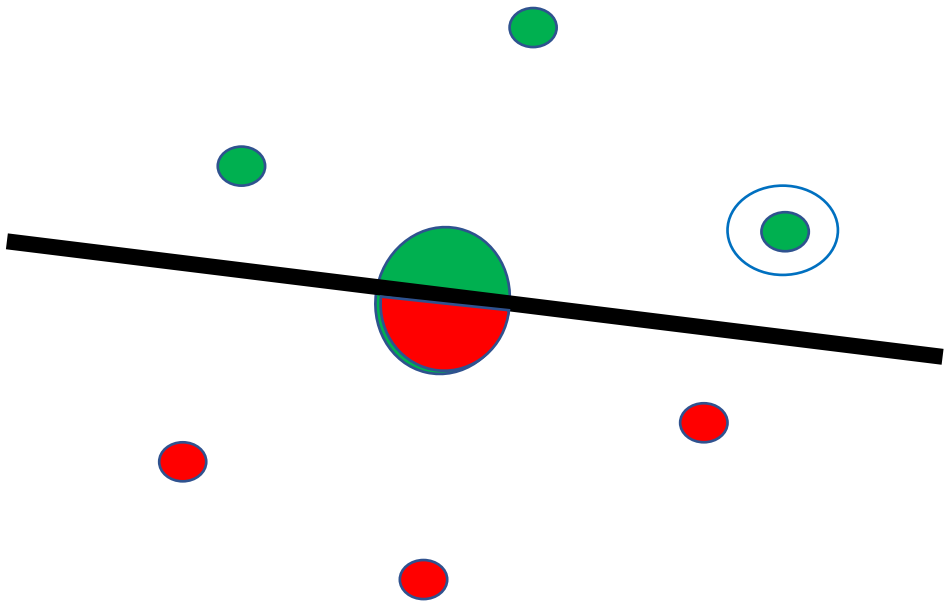
Logistic Regression



Error

1

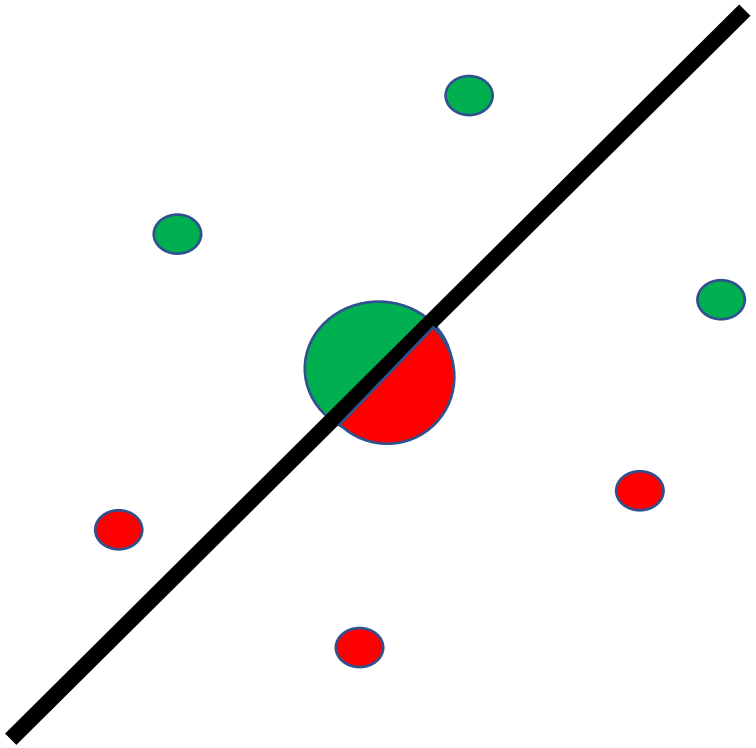
Logistic Regression



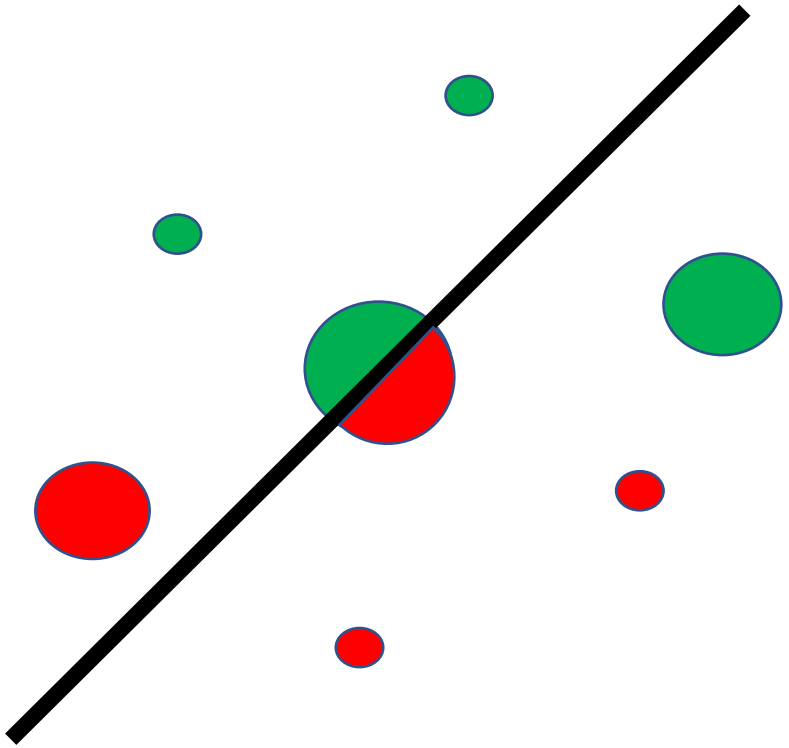
Error

θ

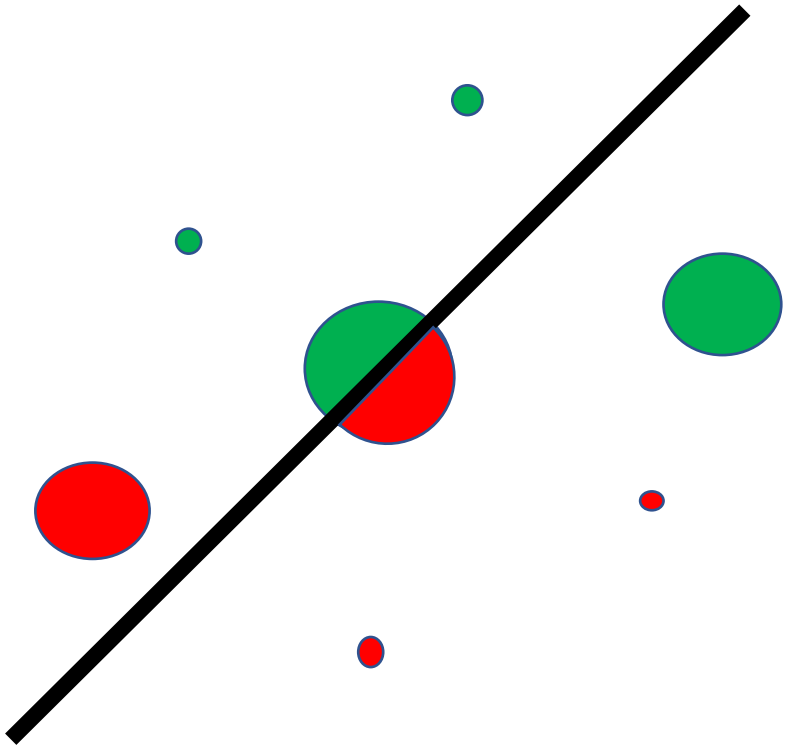
Logistic Regression



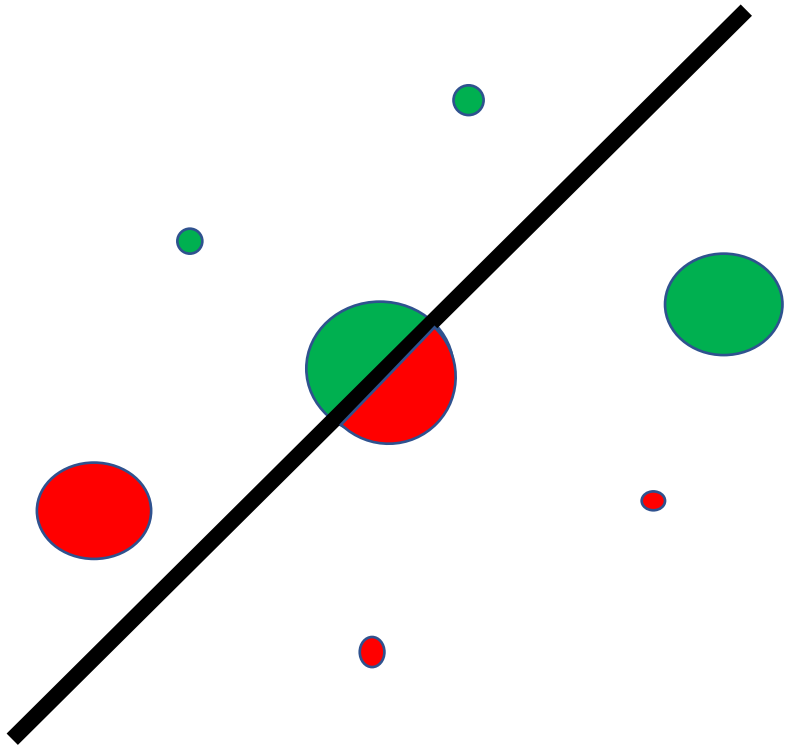
Logistic Regression

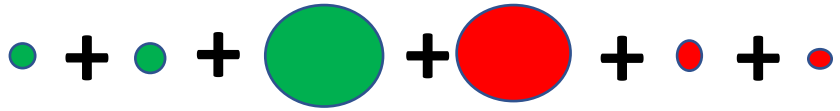


Logistic Regression

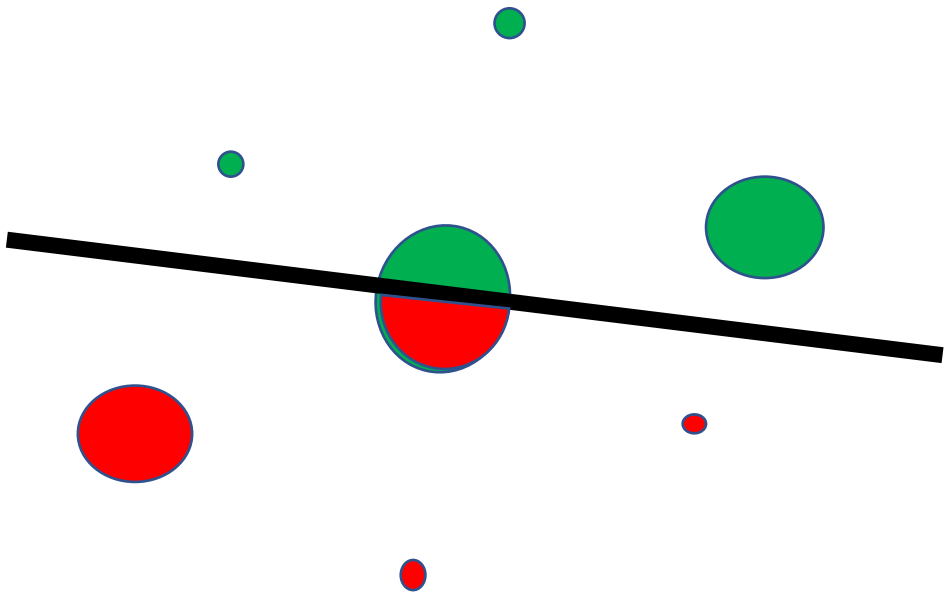


Logistic Regression



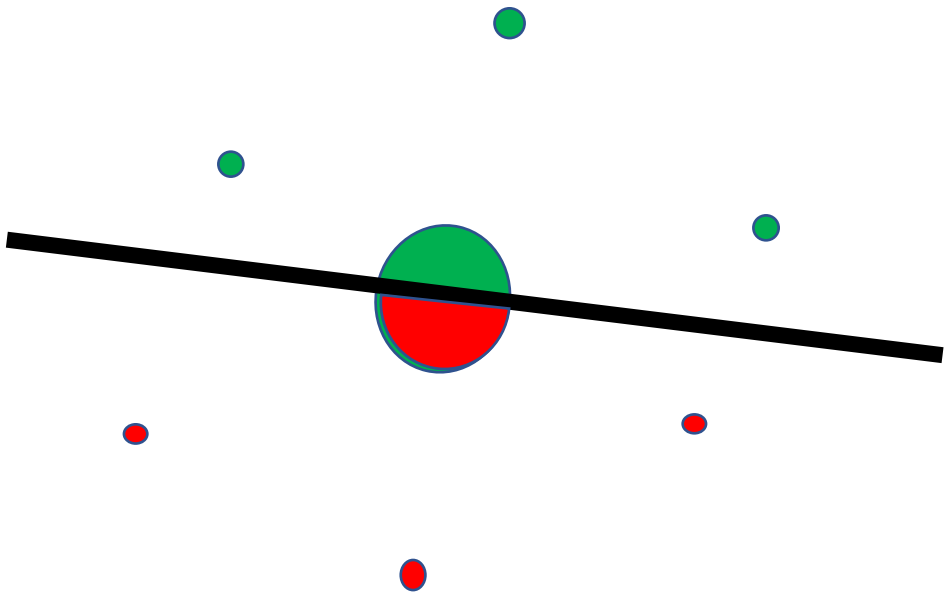
Error= 

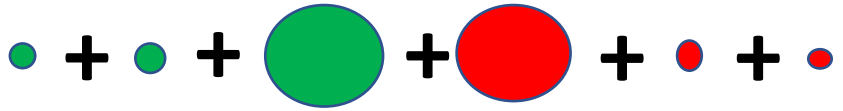
Logistic Regression



Error= + + + + +

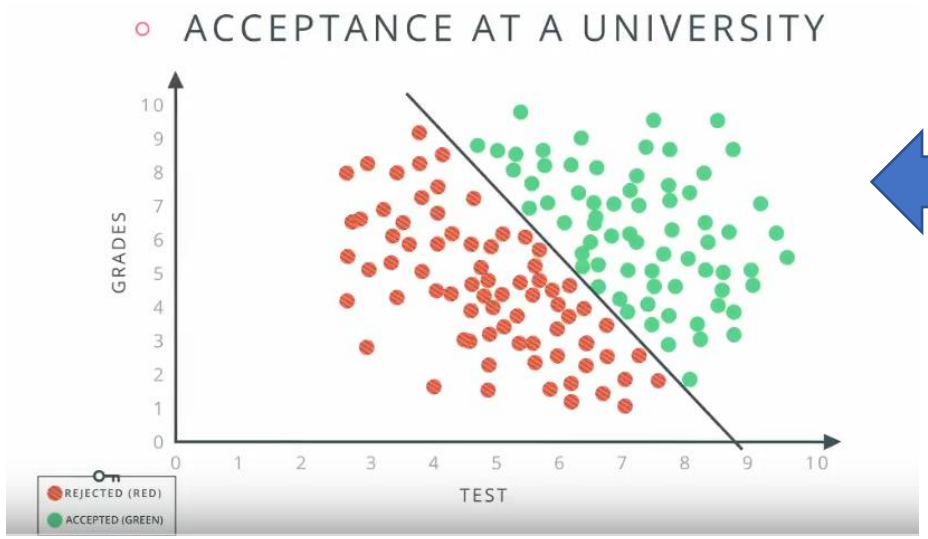
Logistic Regression



Error= 

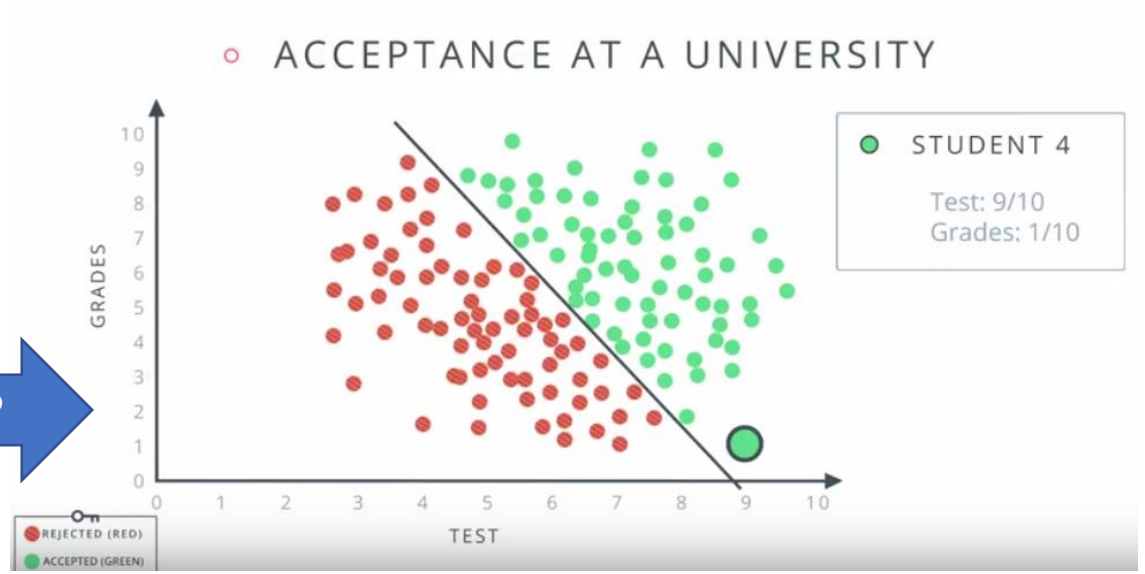
Error= 

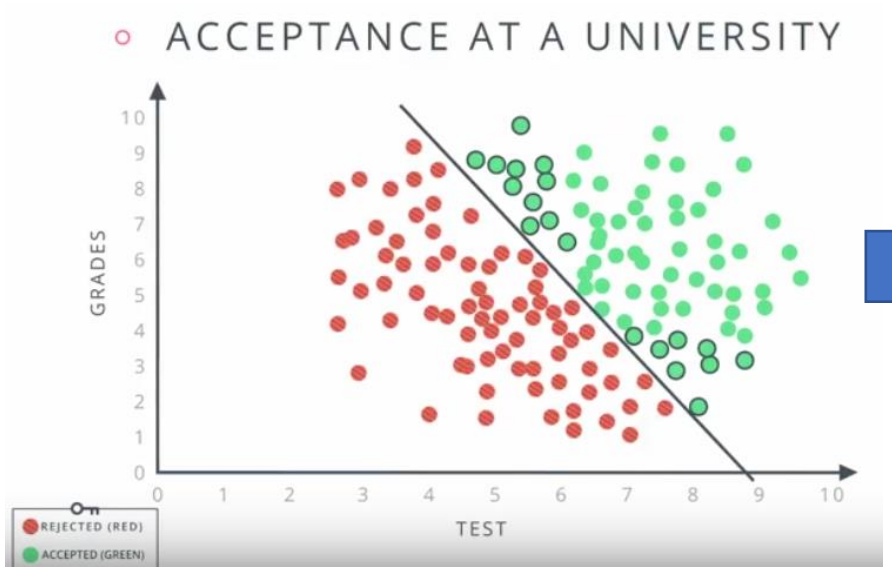
MINIMIZE ERROR



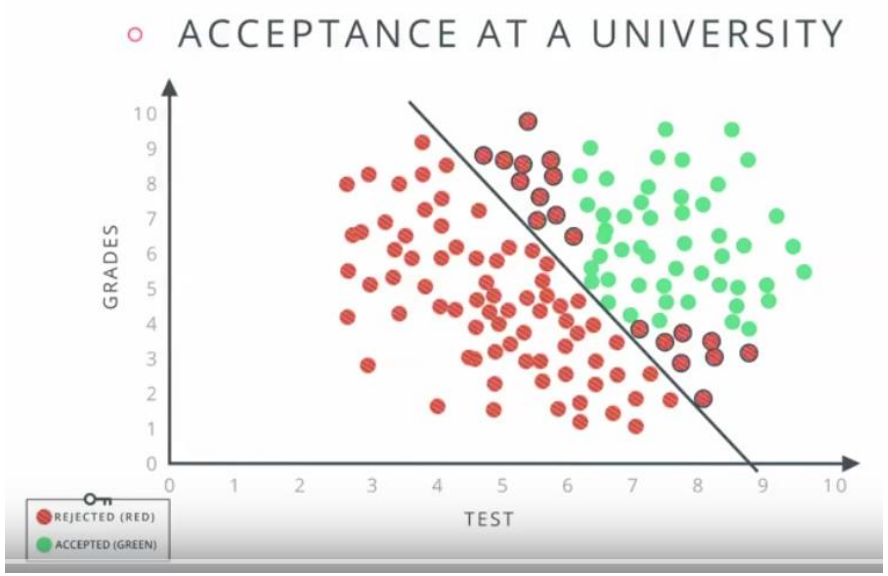
Check

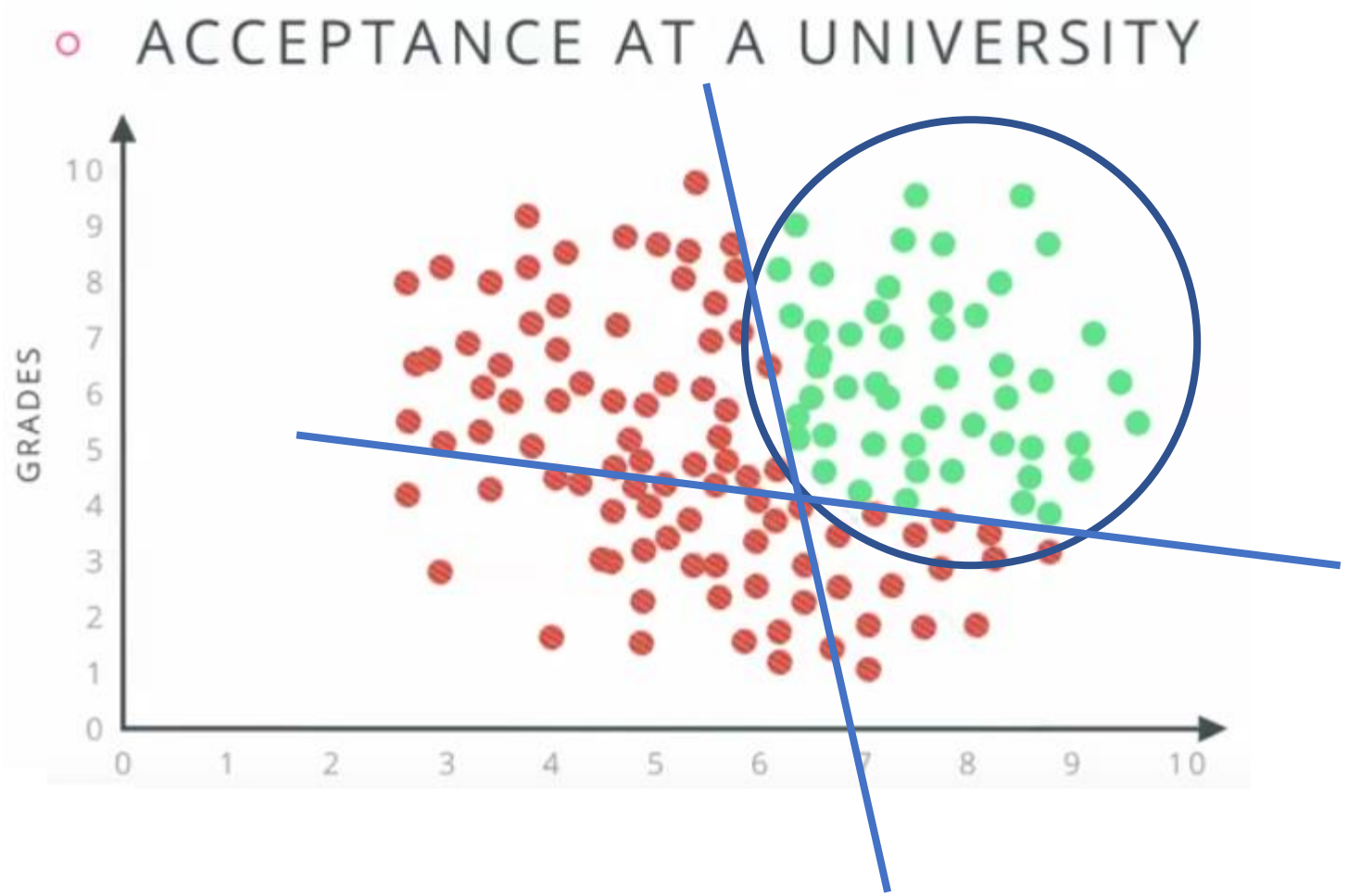
?





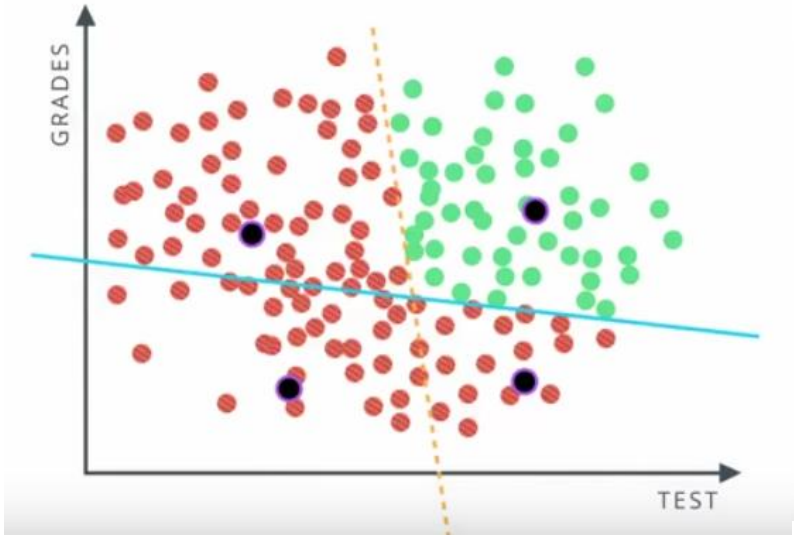
Real data





GRADIENT DESCENT

NEURAL NETWORK

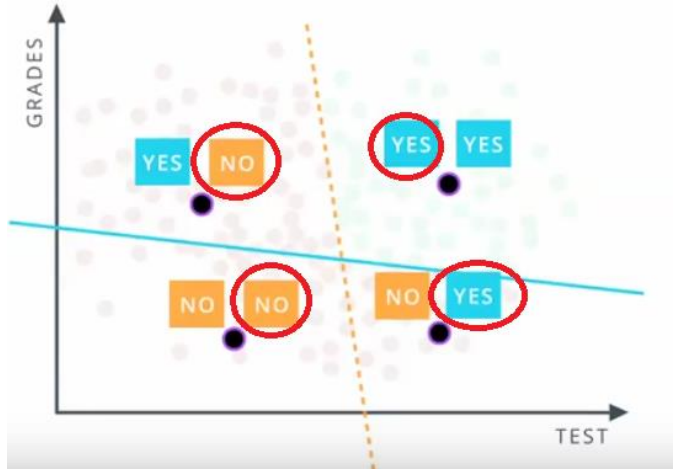


QUESTION 1

Is this point over the BLUE line?

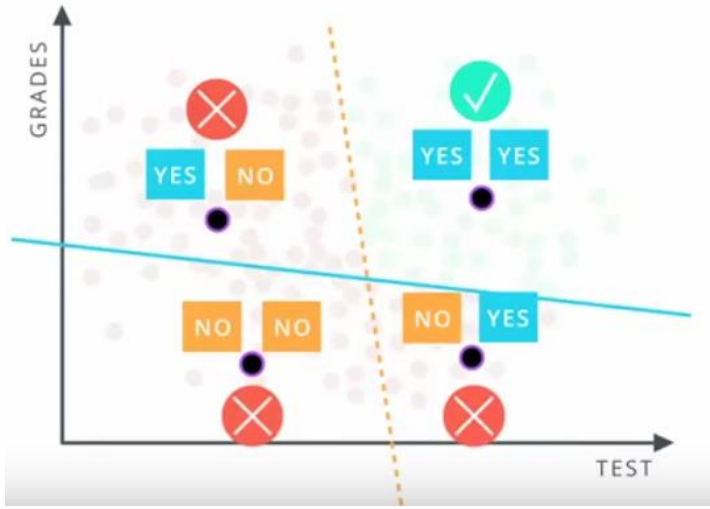
QUESTION 2

Is this point over the ORANGE line?



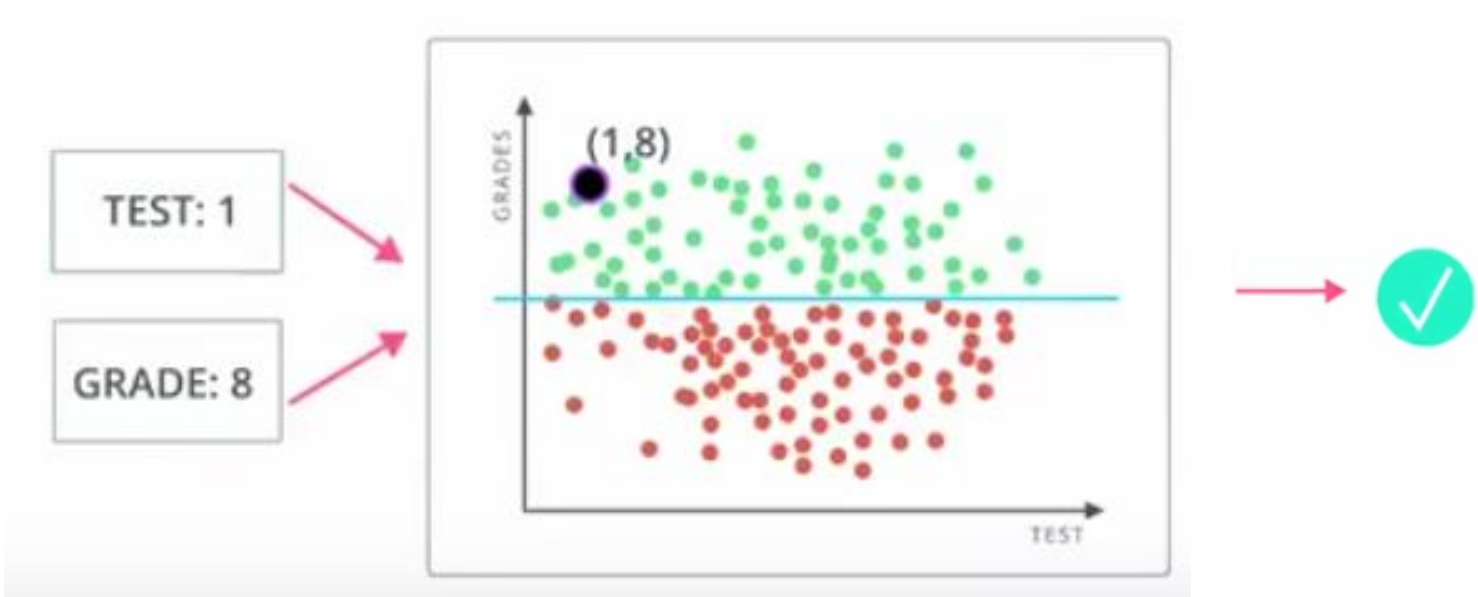
QUESTION 3

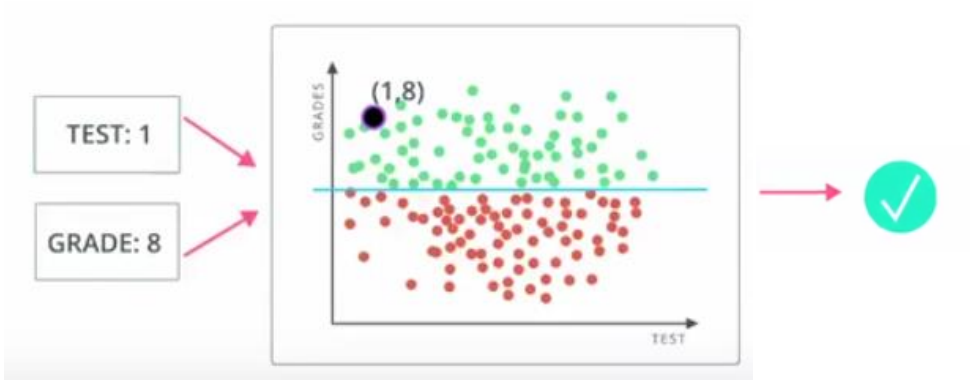
Are the answers to Questions 1 and 2 both **yes**?



QUESTION 1

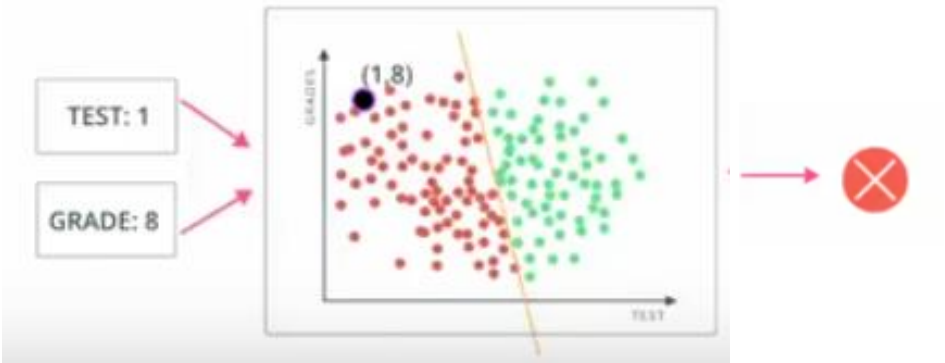
Is this point over the BLUE line?





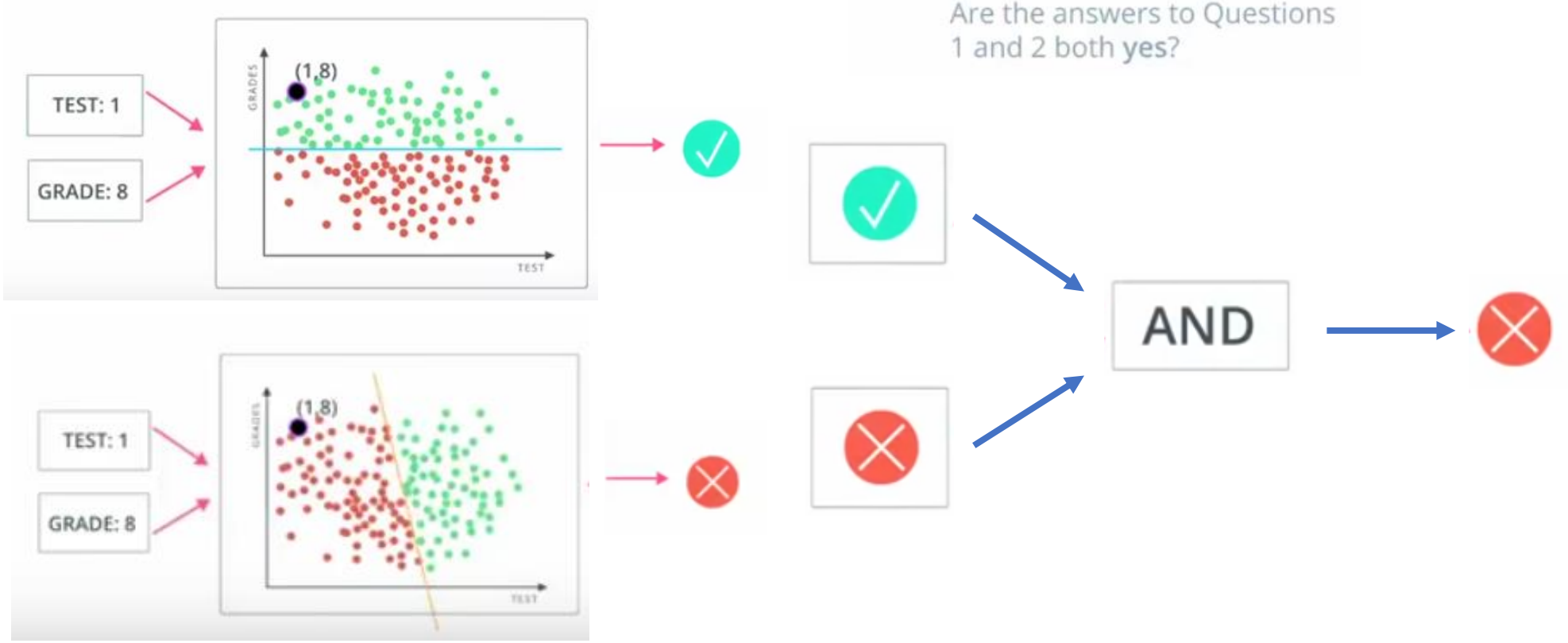
QUESTION 2

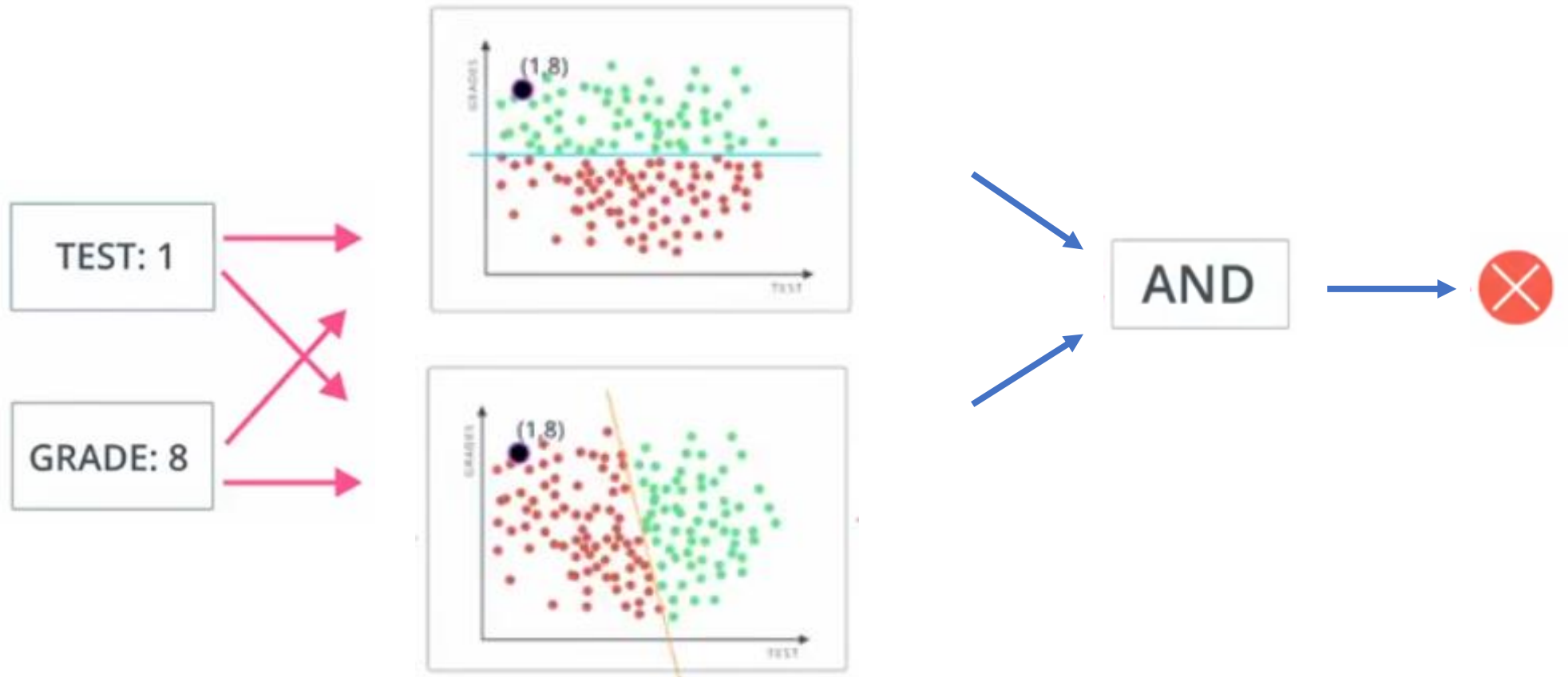
Is this point over the ORANGE line?

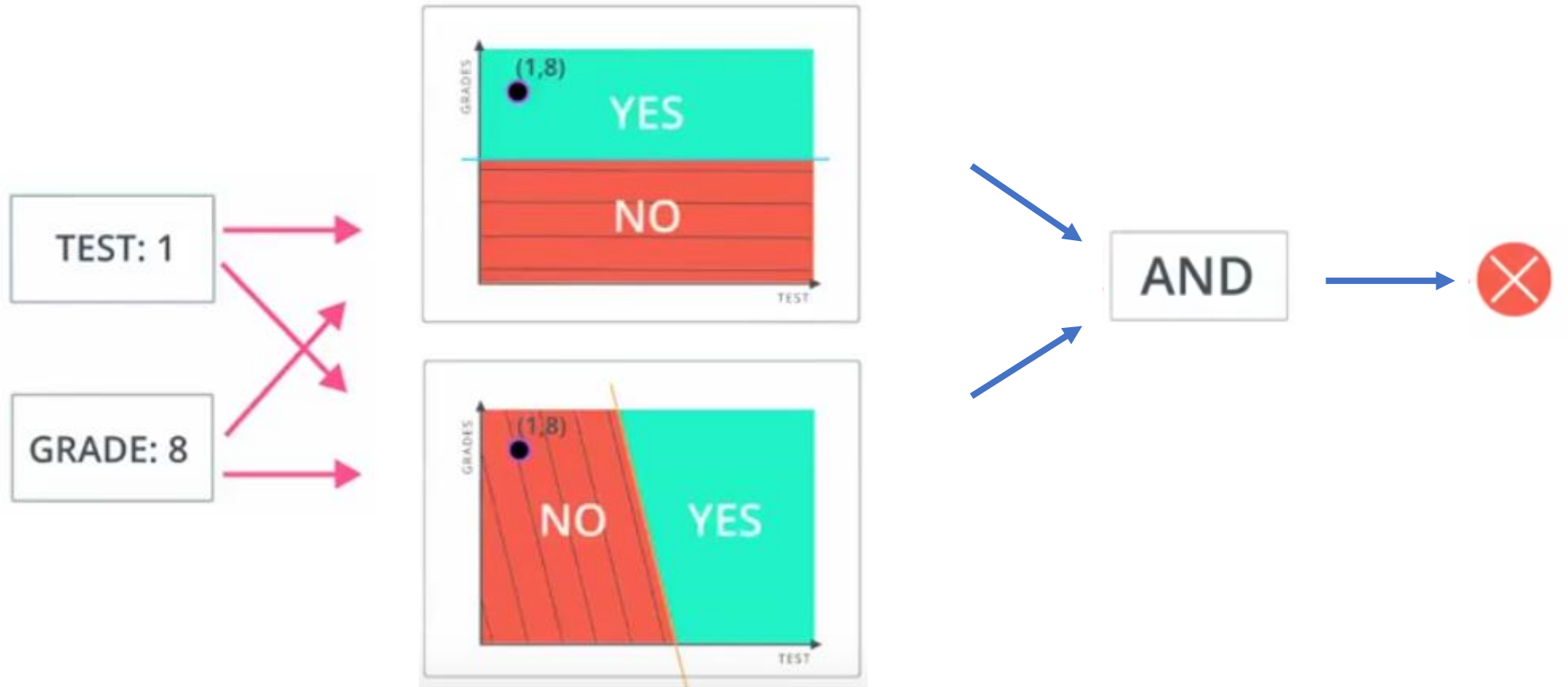


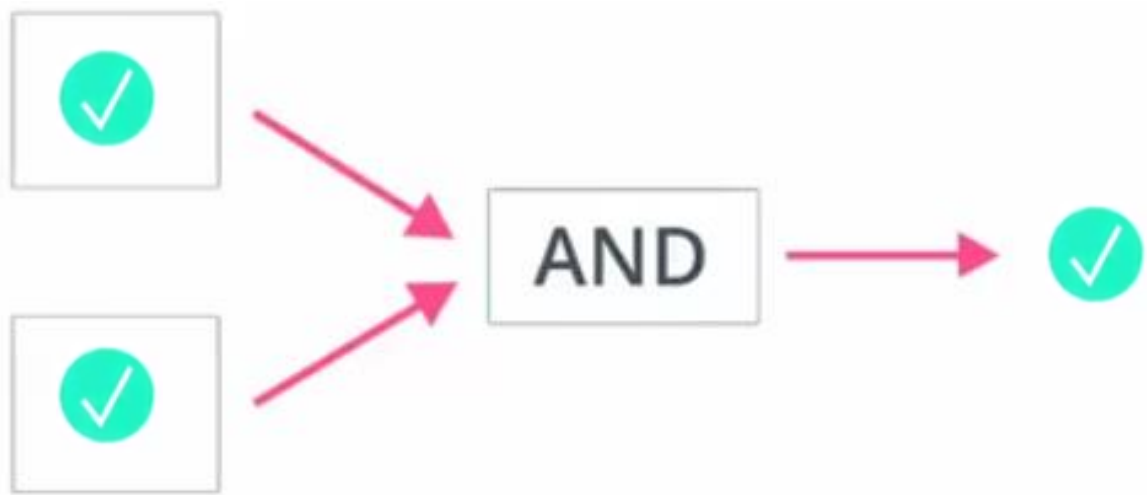
QUESTION 3

Are the answers to Questions 1 and 2 both yes?

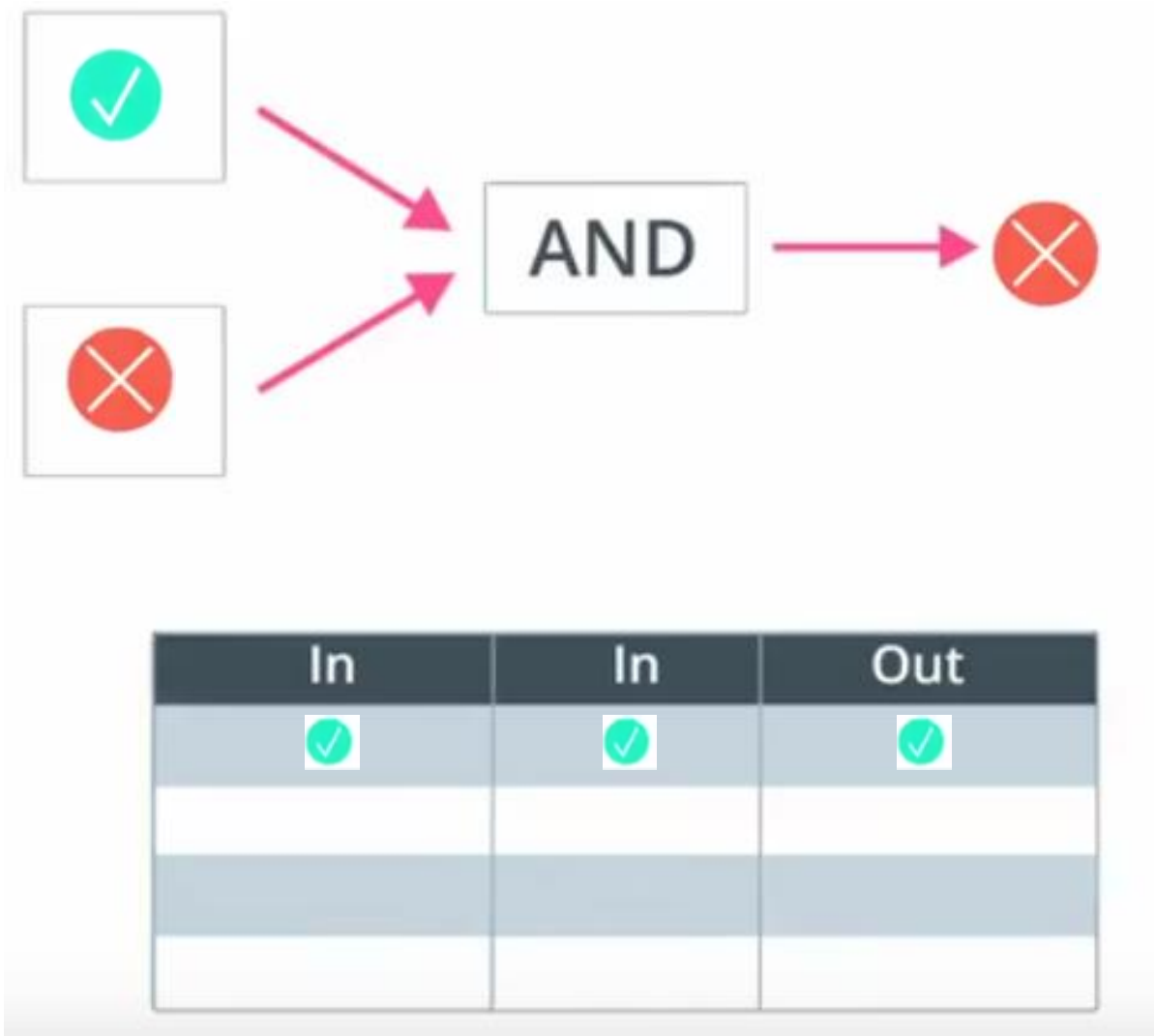


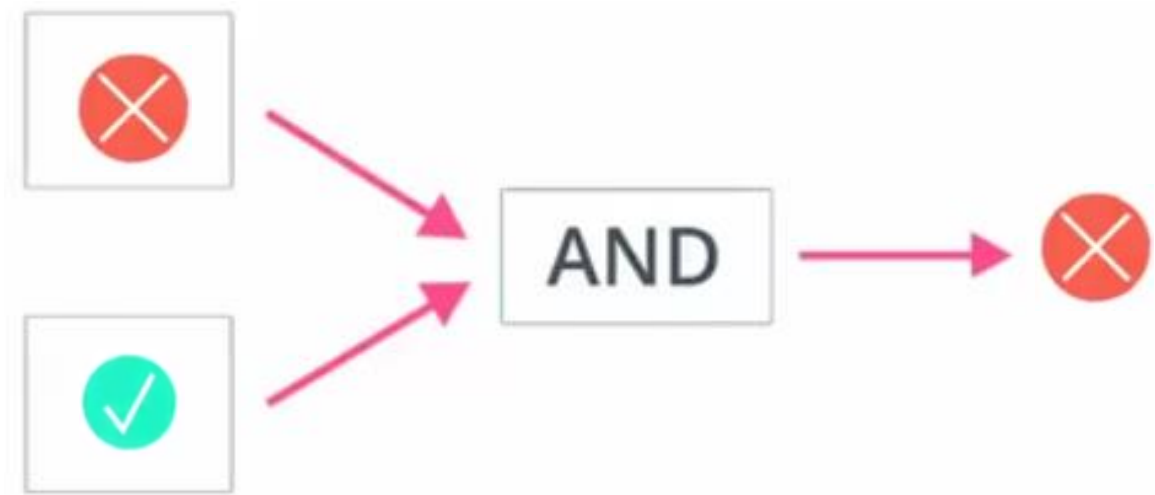

TEST: 1
GRADE: 8
(1,8)
(1,8)
AND


TEST: 1
GRADE: 8
(1,8)
YES
NO
GRADES
TEST
(1,8)
NO
YES
GRADES
TEST
AND

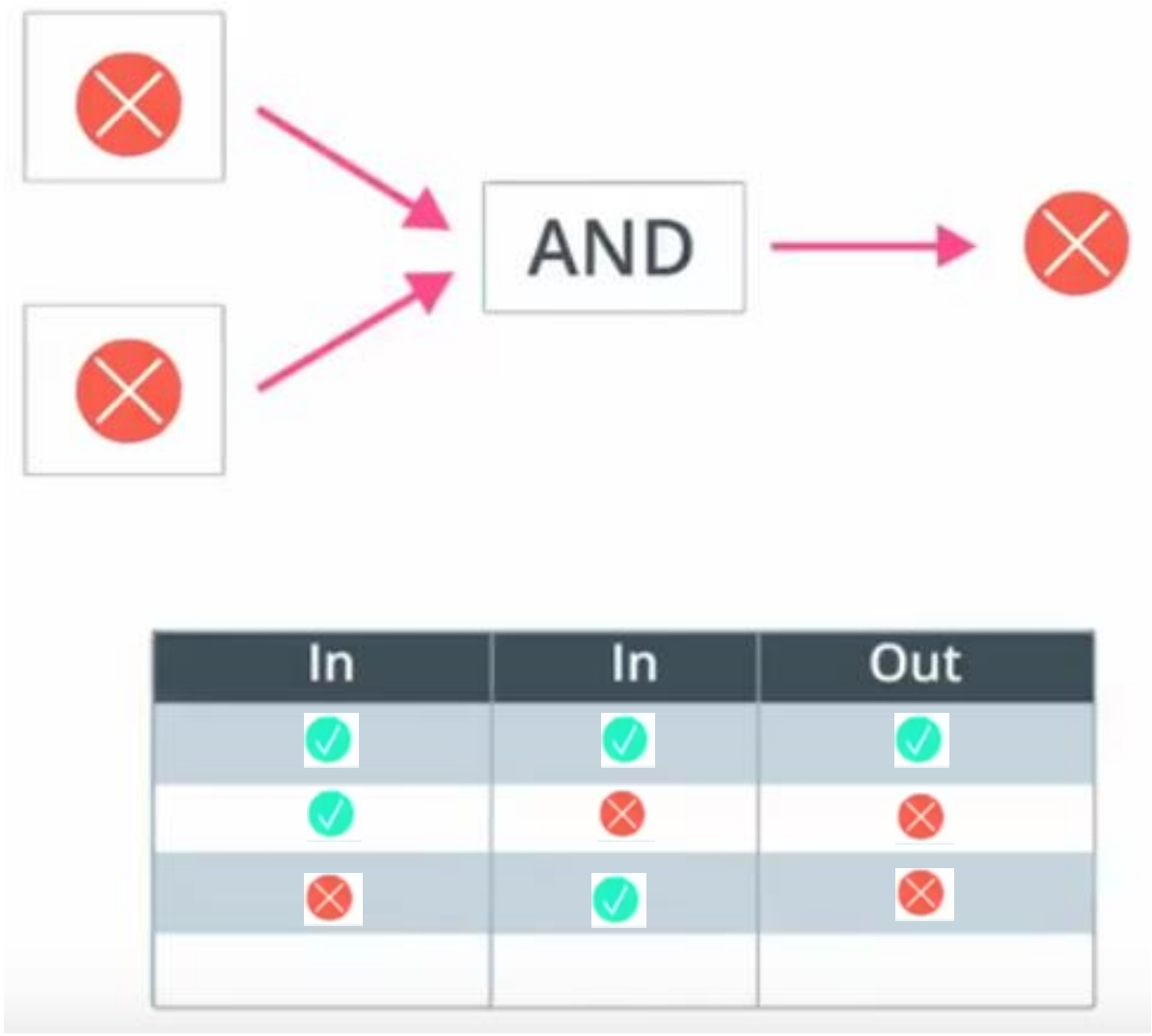


In	In	Out

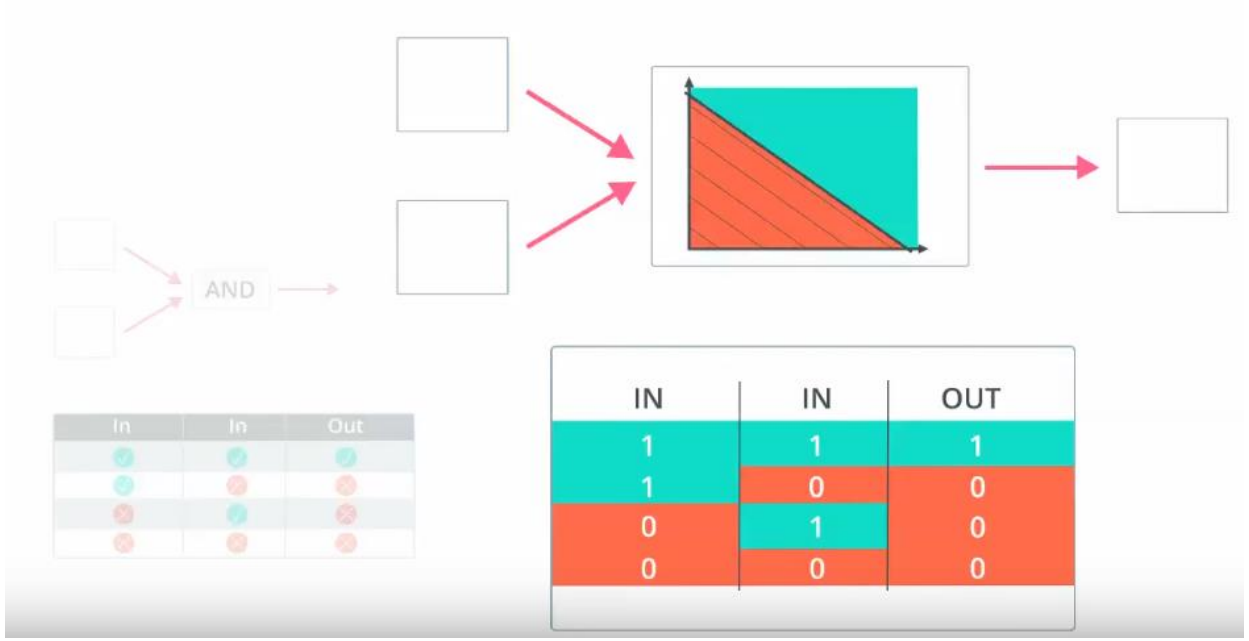

AND
In
In
Out



In	In	Out
✓	✓	✓
✓	✗	✗

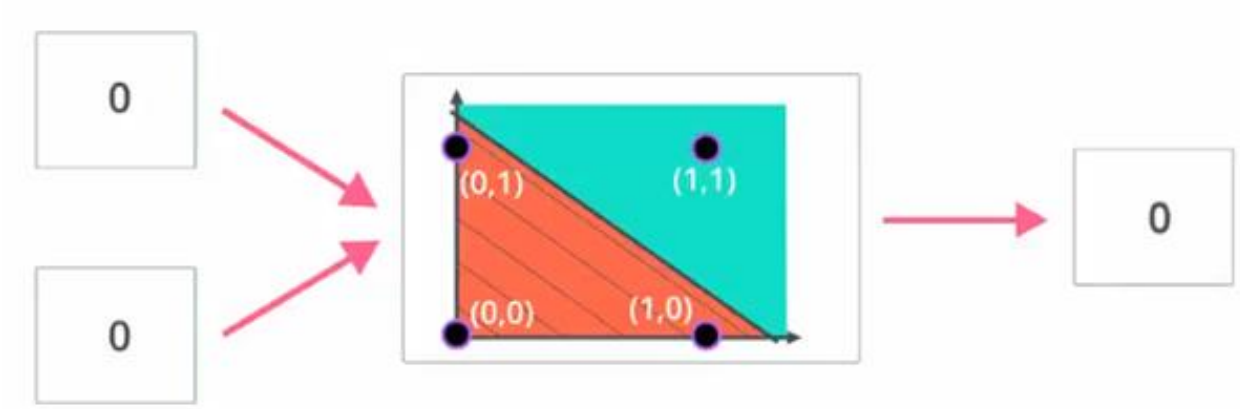

AND
In
In
Out

In	In	Out
✓	✓	✓
✓	✗	✗
✗	✓	✗
✗	✗	✗

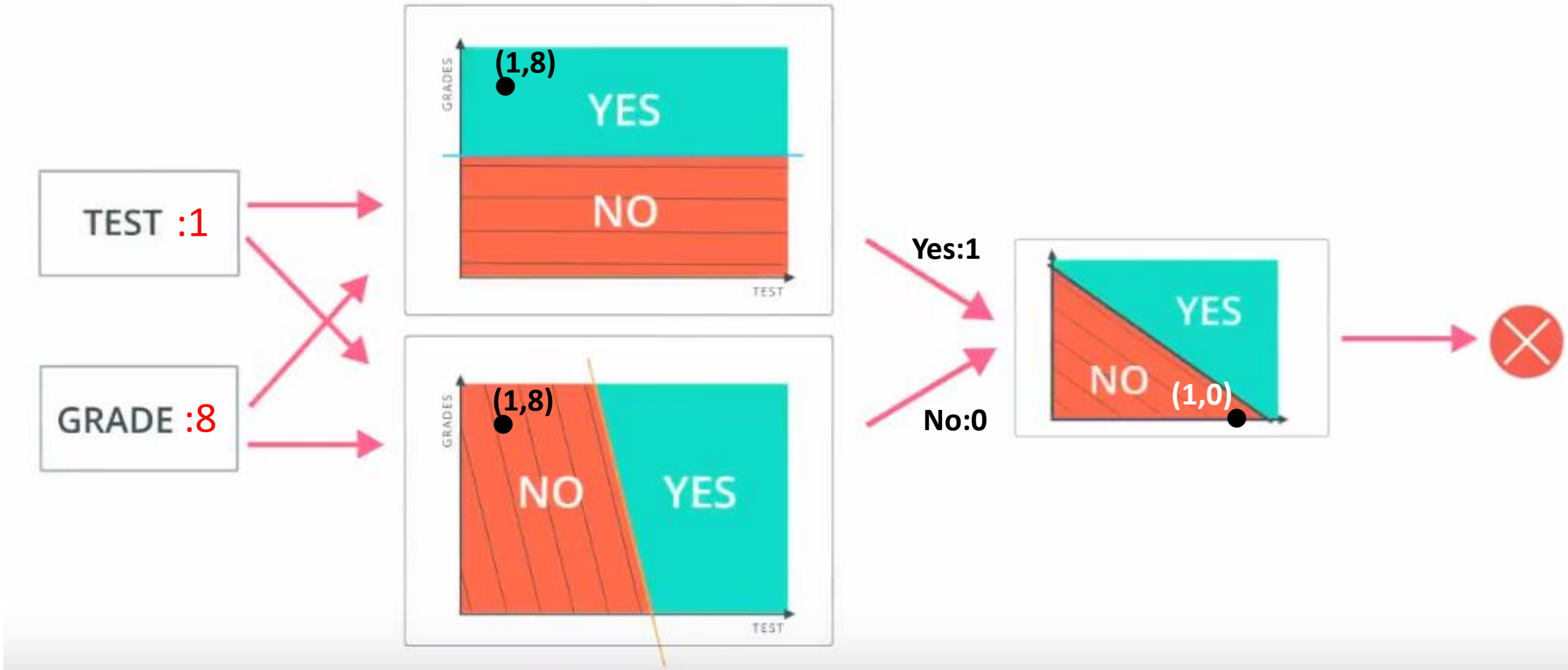


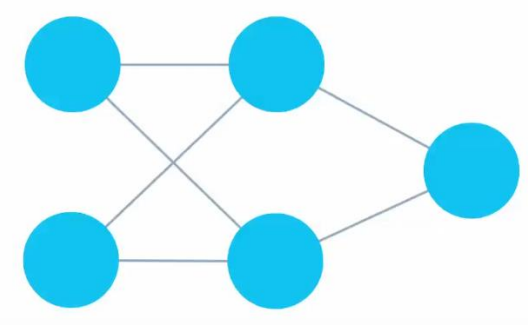
In	In	Out

IN	IN	OUT
1	1	1
1	0	0
0	1	0
0	0	0

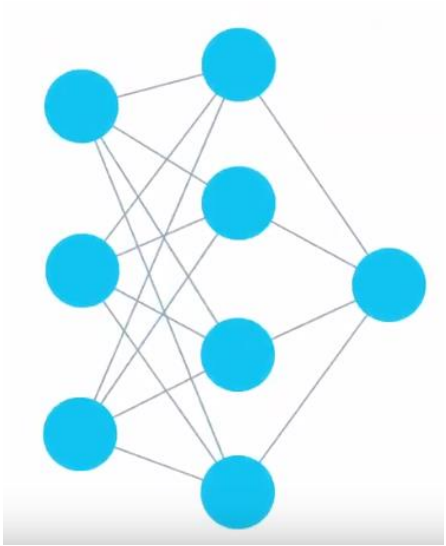


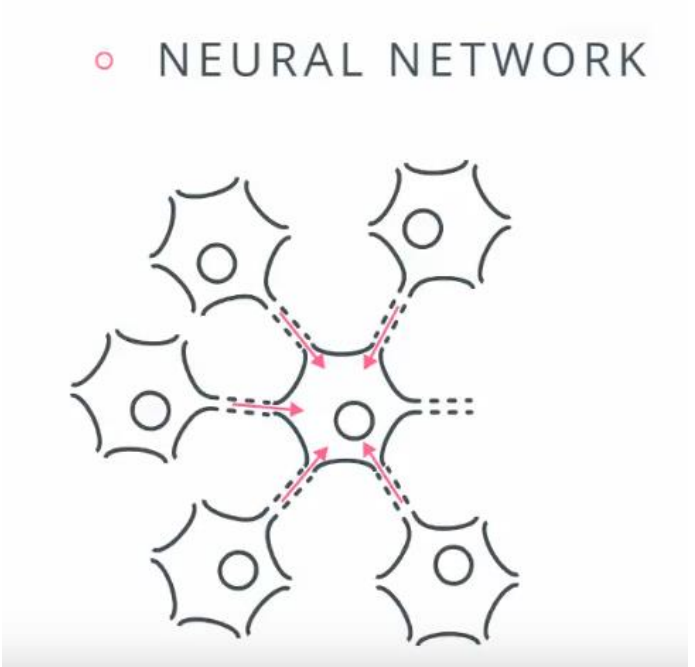
IN	IN	OUT
1	1	1
1	0	0
0	1	0
0	0	0


TEST :1
GRADE :8
GRADES
(1,8)
YES
NO
TEST
GRADES
(1,8)
NO
YES
TEST
Yes:1
No:0
YES
NO
(1,0)



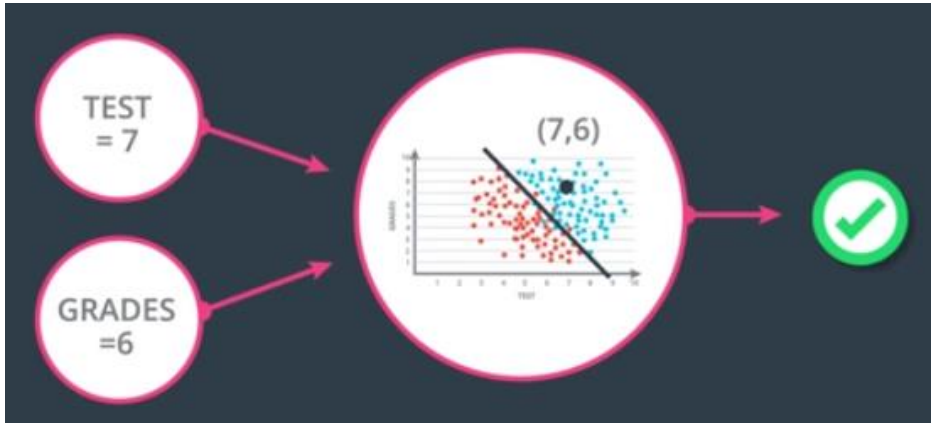

Self
Driving




NEURAL NETWORK

Perceptrons

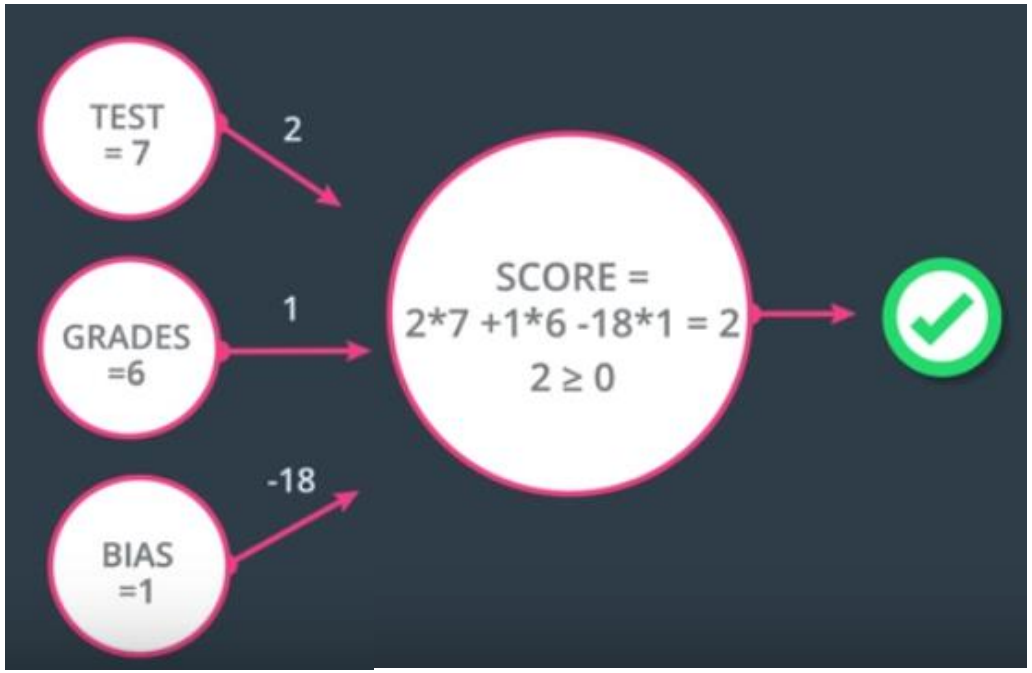
The building block of neural networks. In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers. The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

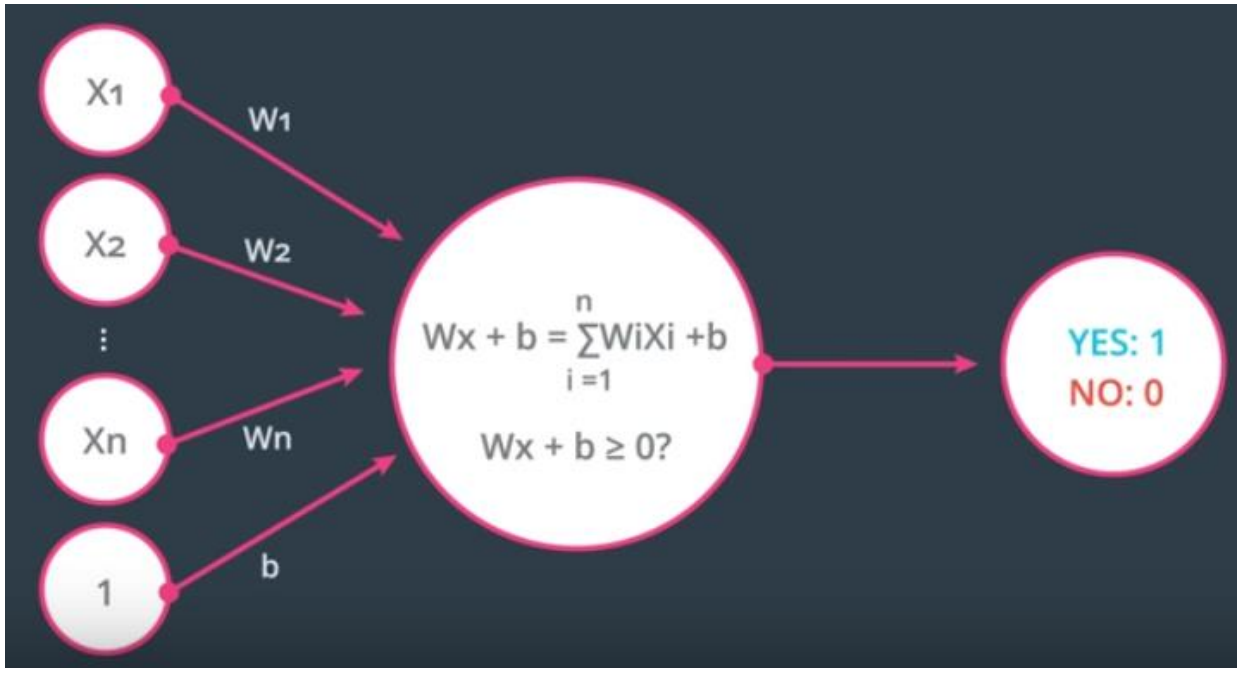


Score=
2*Test + 1*Grades -18

PREDICTION:
Score ≥ 0 Accept
Score < 0 Reject

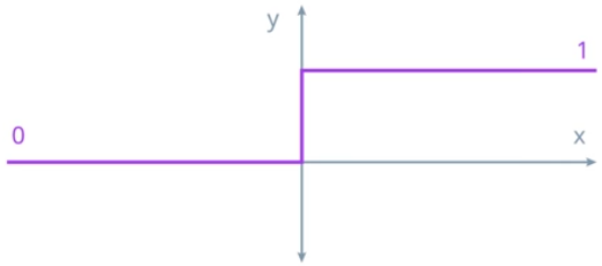
Perceptrons



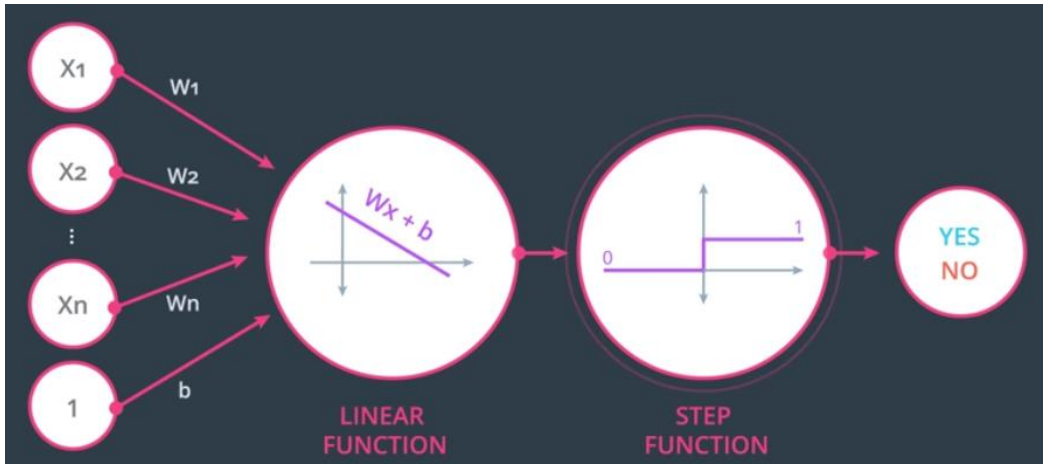


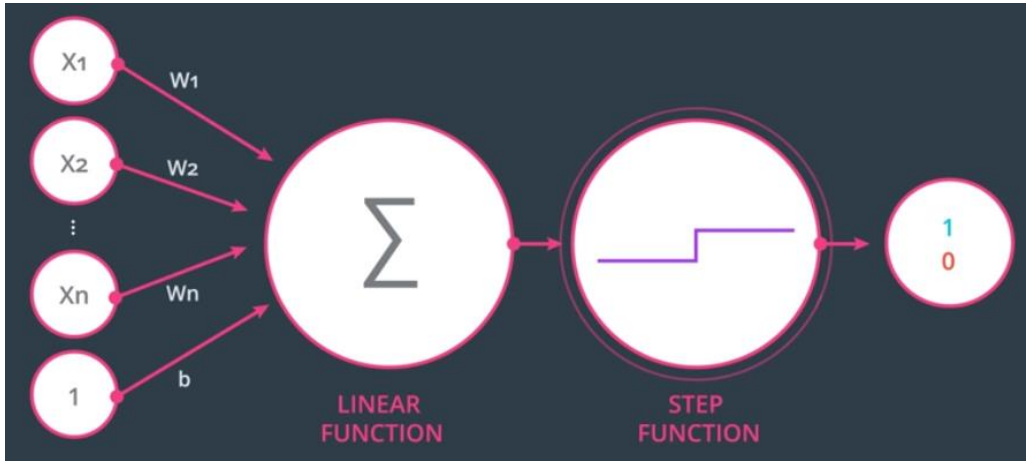
Perceptrons

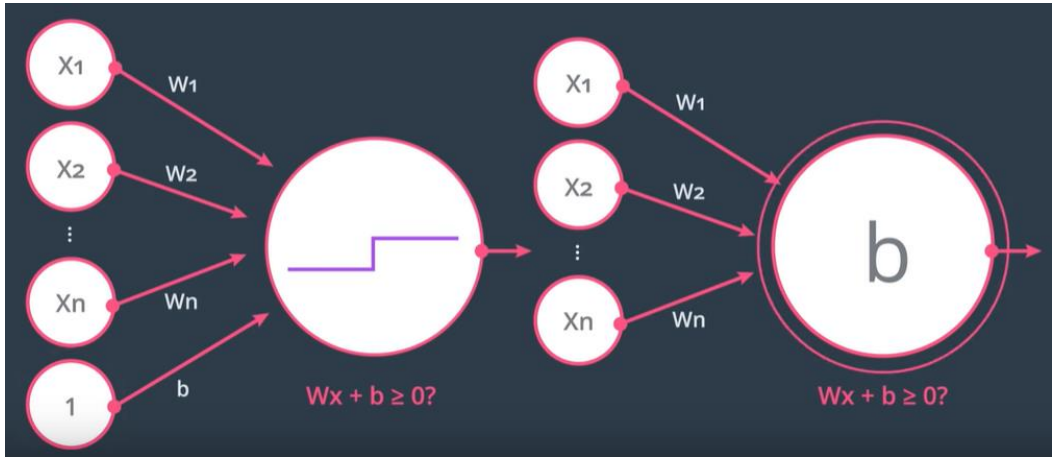
Step Function



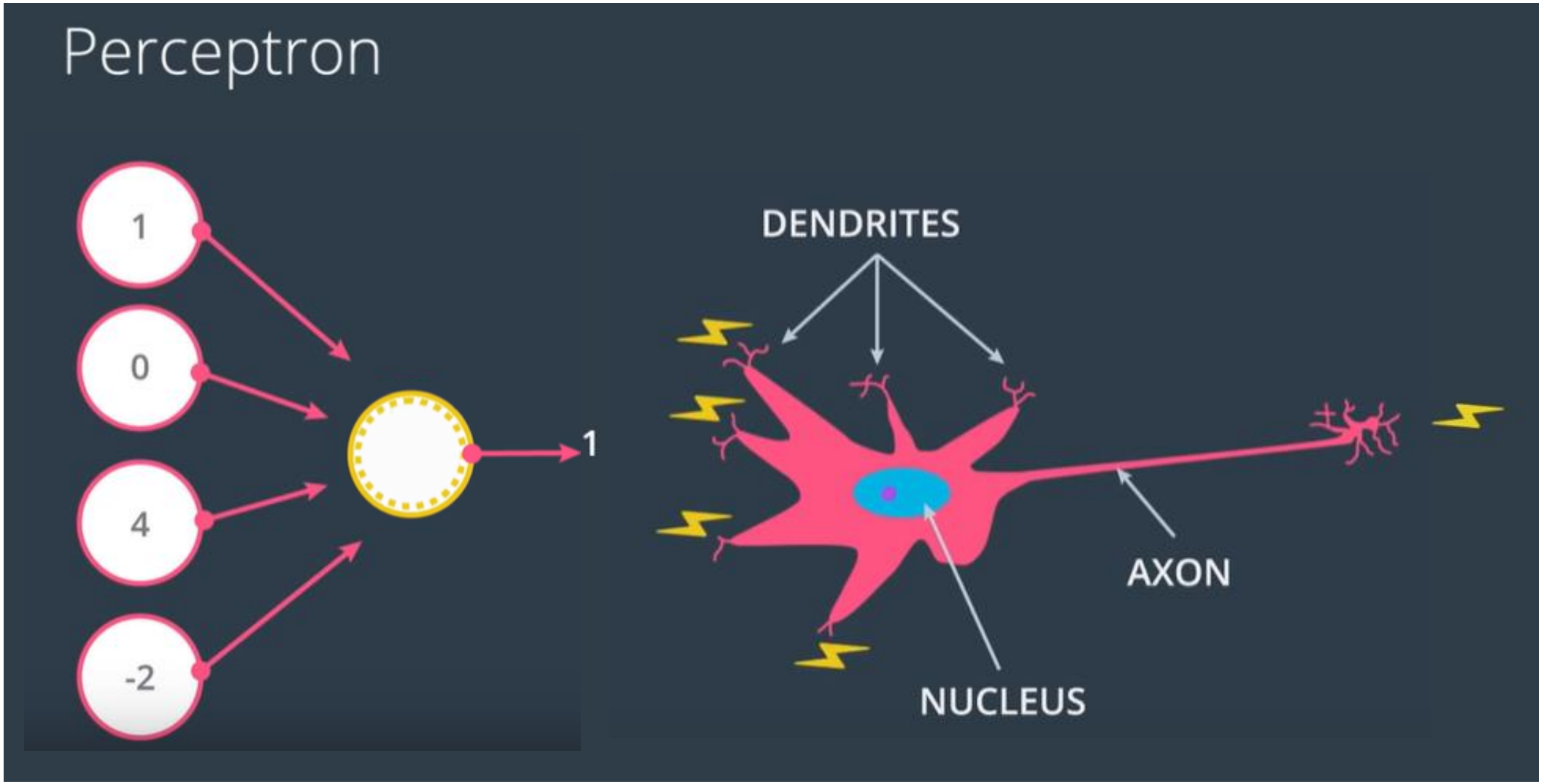
$$y = \begin{cases} 1 \text{ if } x \geq 0 \\ 1 \text{ if } x < 0 \end{cases}$$



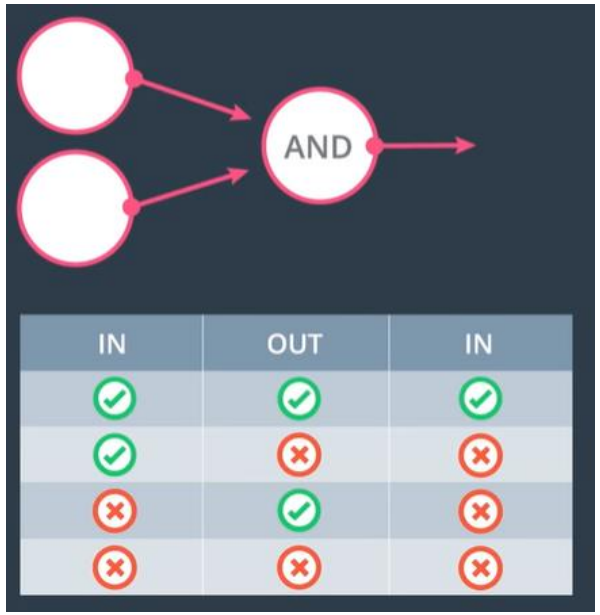


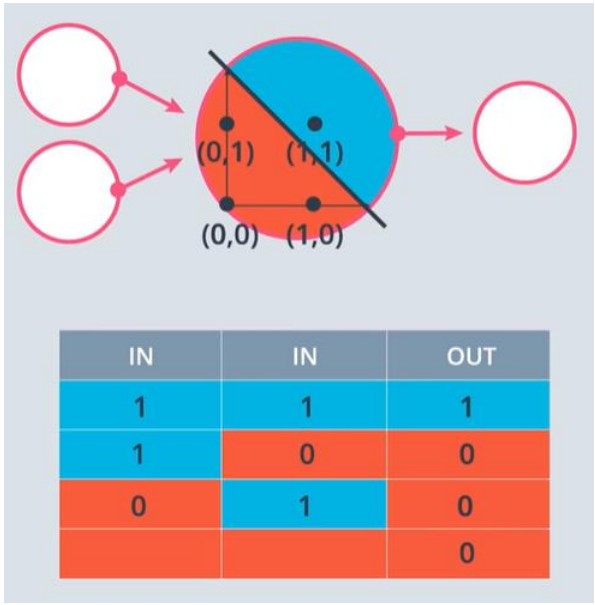


Why "Neural Networks"?

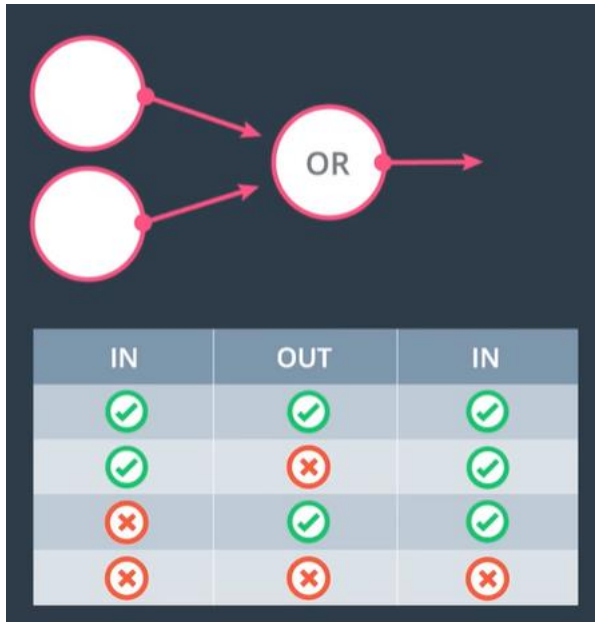


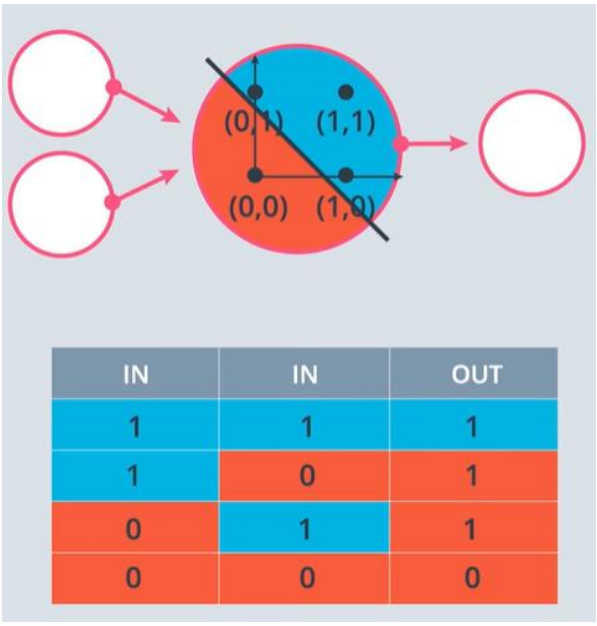
AND Perceptrons



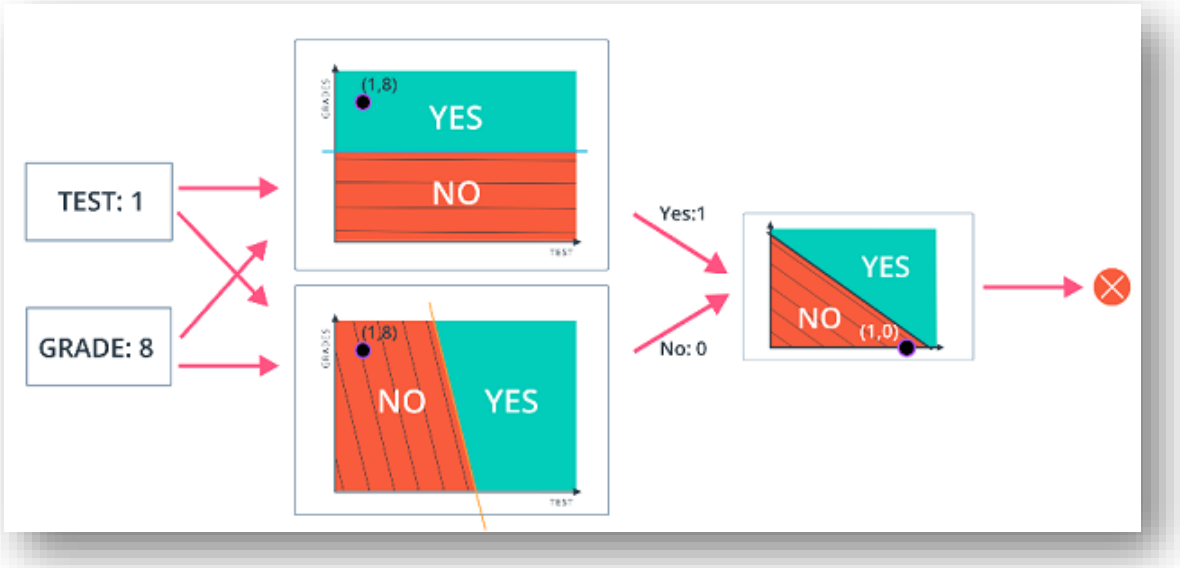


OR Perceptrons





Perceptron

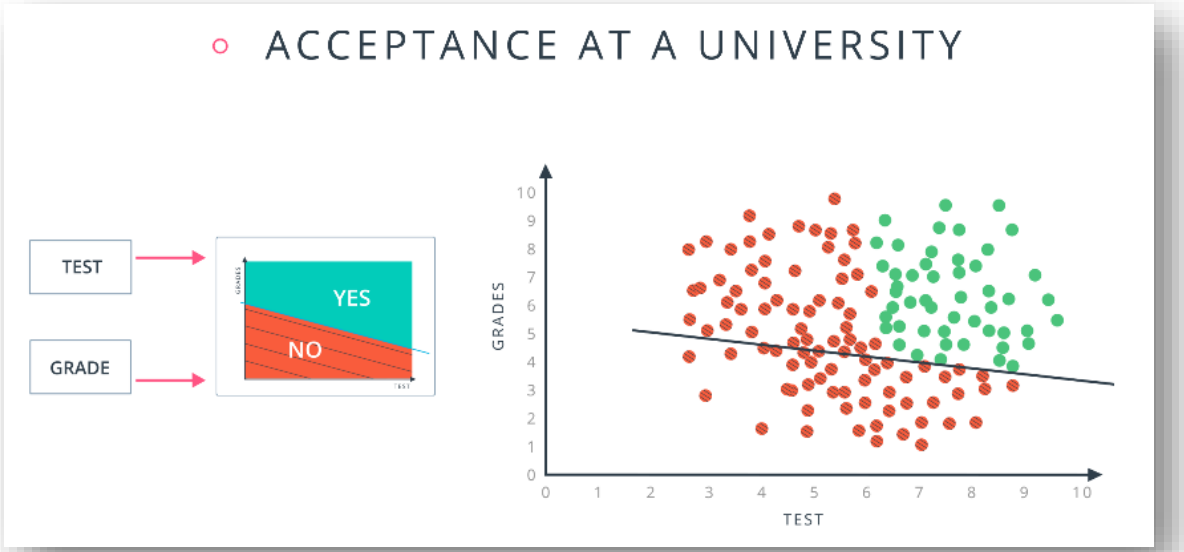


- **How a simple neural network makes decisions:** by taking in input data, processing that information, and finally, producing an output in the form of a decision!

  Let's take a deeper dive into the university admission example to learn more about processing the input data.

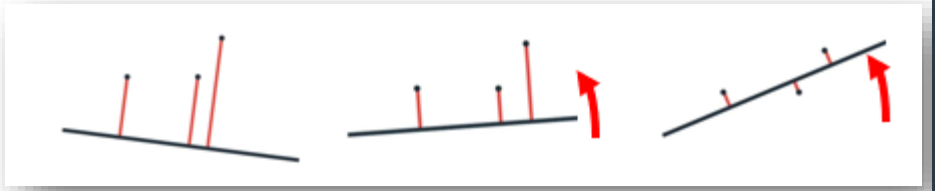
- Data, like test scores and grades, are fed into a network of interconnected nodes. These individual nodes are called perceptrons, or artificial neurons, and they are the basic unit of a neural network. In the example above, the input either passes a threshold for grades and test scores or doesn't, and so the two categories are: yes (passed the threshold) and no (didn't pass the threshold). These categories then combine to form a decision -- for example, if both nodes produce a "yes" output, then this student gains admission into the university.

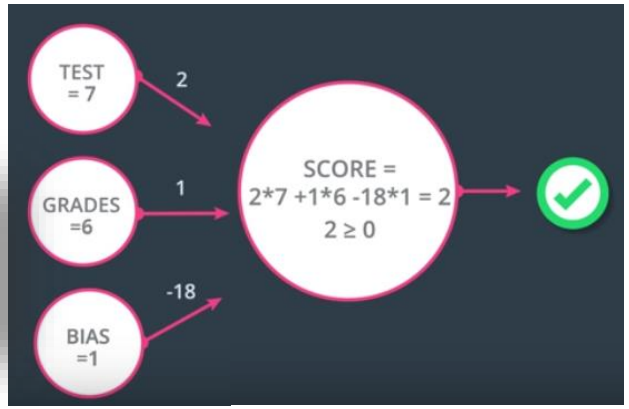
Perceptron



- Let's zoom in even further and look at how a single perceptron processes input data.
- The perceptron above is one of the two perceptrons from the image(Page 67) that help determine whether or not a student is accepted to a university. It decides whether a student's grades are high enough to be accepted to the university. You might be wondering: "How does it know whether grades or test scores are more important in making this acceptance decision?" Well, when we initialize a neural network, we don't know what information will be most important in making a decision. It's up to the neural network to learn for itself which data is most important and adjust how it considers that data.
- It does this with something called weights.

Perceptron





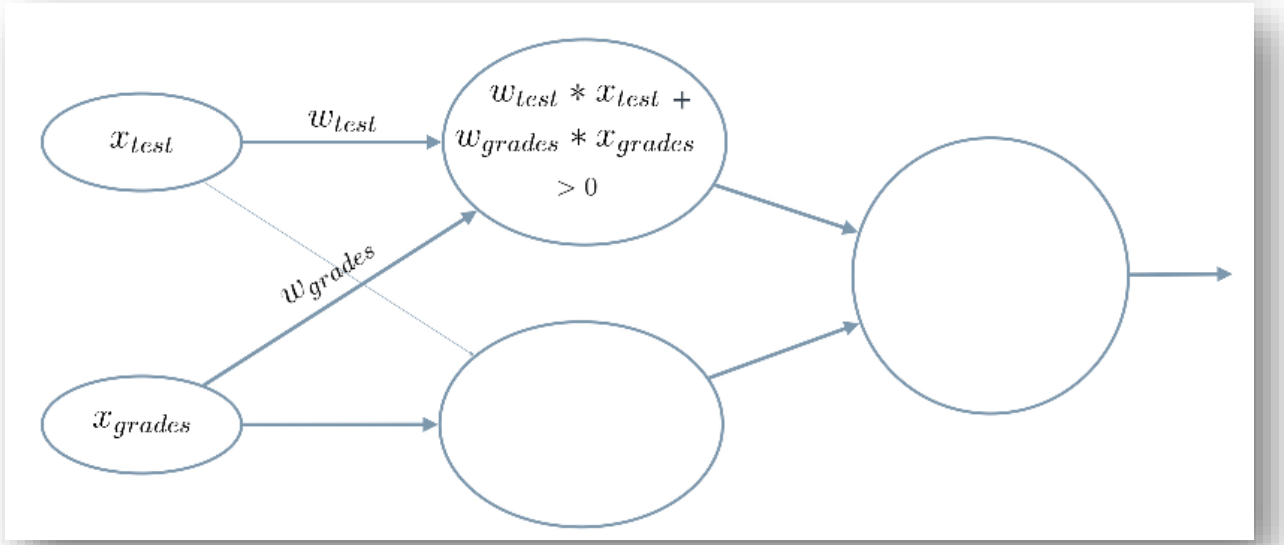
Weights

- When input comes into a perceptron, it gets multiplied by a weight value that is assigned to this particular input. For example, the perceptron above has two inputs, tests for test scores and grades, so it has two associated weights that can be adjusted individually. These weights start out as random values, and as the neural network learns more about what kind of input data leads to a student being accepted into a university, the network adjusts the weights based on any errors in categorization that results from the previous weights. This is called training the neural network.
- A higher weight means the neural network considers that input more important than other inputs, and lower weight means that the data is considered less important. An extreme example would be if test scores had no affect at all on university acceptance; then the weight of the test score input would be zero and it would have no affect on the output of the perceptron.

Perceptron

Summing the Input Data

- Each input to a perceptron has an associated weight that represents its importance. These weights are determined during the learning process of a neural network, called training. In the next step, the weighted input data are summed to produce a single value, that will help determine the final output - whether a student is accepted to a university or not. Let's see a concrete example of this.



We weight x_test by w_test and add it to x_grades weighted by w_grades.

Perceptron

- When writing equations related to neural networks, the weights will always be represented by some type of the letter ***w***. It will usually look like a *W* when it represents a **matrix** of weights or a ***w*** when it represents an **individual** weight, and it may include some additional information in the form of a subscript to specify *which* weights (you'll see more on that next). But remember, when you see the letter ***w***, think **weights**.
- In this example, we'll use w_{grades} for the weight of grades and w_{test} for the weight of test. For the image above, let's say that the weights are: w_{grades} =−1, w_{test} =−0.2. You don't have to be concerned with the actual values, but their relative values are important. **W***grades* is 5 times larger than **W***test*, which means the neural network considers grades input 5 times more important than test in determining whether a student will be accepted into a university.

Perceptron

- The perceptron applies these weights to the inputs and sums them in a process known as linear combination. In our case, this looks like $w_{grades} \cdot x_{grades} + w_{test} \cdot x_{test} = -1 \cdot x_{grades} - 0.2 \cdot x_{test}$
- Now, to make our equation less wordy, let's replace the explicit names with numbers. Let's use 1 for grades and 2 for tests. So now our equation becomes

  $w_1 \cdot x_1 + w_2 \cdot x_2$
- In this example, we just have 2 simple inputs: ***grades*** and ***tests***. Let's imagine we instead had m different inputs and we labeled them $x_1, x_2, \ldots, x_m$. Let's also say that the weight corresponding to x_1 is w_1 and so on. In that case, we would express the linear combination succinctly as:

$$\sum_{i=1}^{m} W_i \cdot X_i$$

Perceptron

Here, the Greek letter Sigma $\sum$ is used to represent **summation**. It simply means to evaluate the equation to the right multiple times and add up the results. In this case, the equation it will sum is $W_i \cdot X_i$

But where do we get w_i and x_i?

$\sum_{i=1}^{m}$ means to iterate over all *i* values, from 1 to m.

So to put it all together, $\sum_{i=1}^{m} W_i \cdot X_i$ means the following:

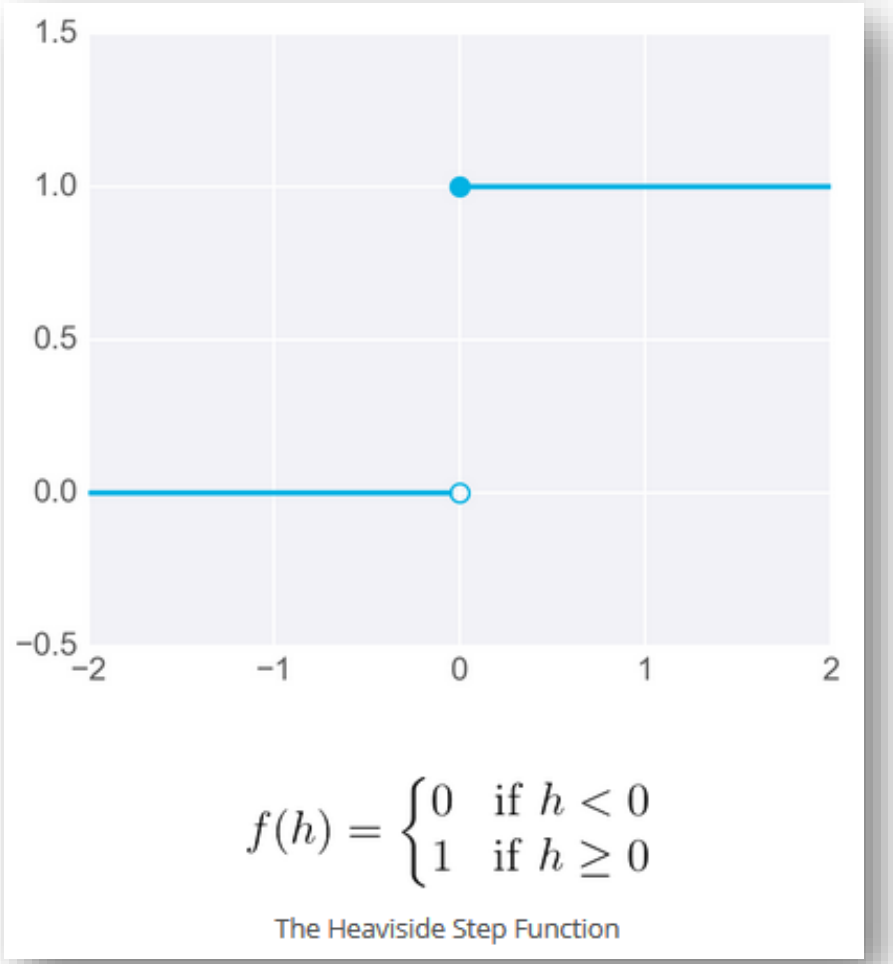
- Start at *i*=1
- Evaluate $W_i \cdot X_i$ and remember the results
- Move to *i*=2
- Evaluate $w_2 \cdot x_2$ and add these results to $W_i \cdot X_i$
- Continue repeating that process until *i*=m, where m is the number of inputs.

One last thing: you'll see equations written many different ways, both here and when reading on your own. For example, you will often just see $\sum_i$ instead of $\sum_{i=1}^{m}$. The first is simply a shorter way of writing the second. That is, if you see a summation without a starting number or a defined end value, it just means perform the sum for all of the them. And *sometimes*, if the value to iterate over can be inferred, you'll see it as just $\sum$. Just remember they're all the same thing: $\sum_{i=1}^{m} w_i \cdot x_i = \sum_i w_i \cdot x_i = \sum w_i \cdot x_i$.

Perceptron

Calculating the Output with an Activation Function

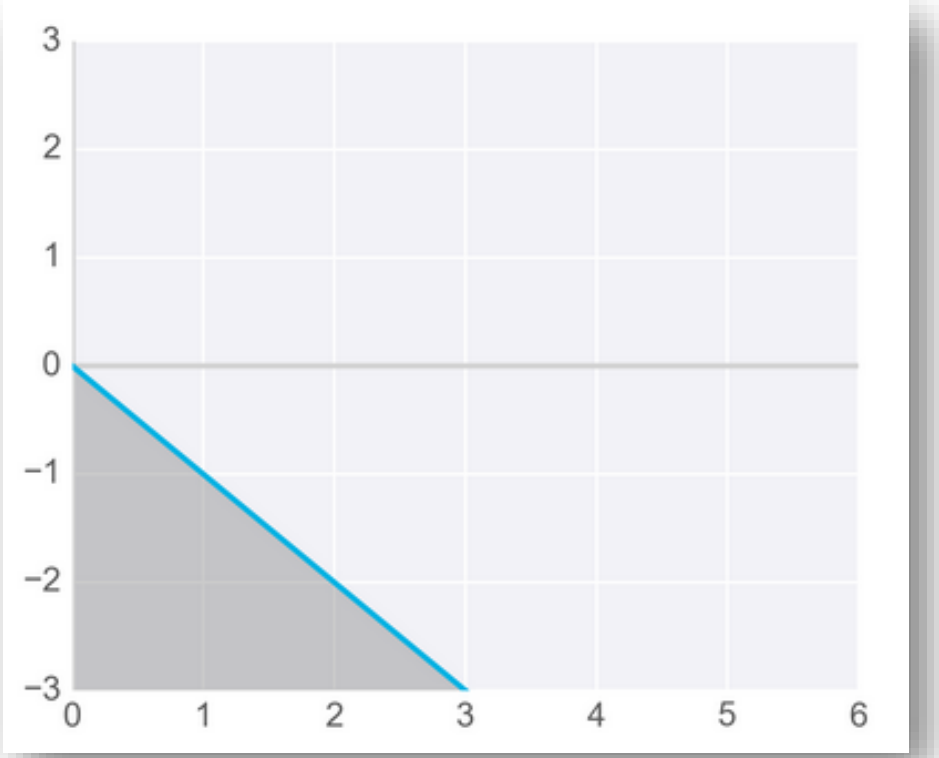
- Finally, the result of the perceptron's summation is turned into an output signal! This is done by feeding the linear combination into an **activation function**.
- Activation functions are functions that decide, given the inputs into the node, what should be the node's output? Because it's the activation function that decides the actual output, we often refer to the outputs of a layer as its "activations".
- One of the simplest activation functions is the **Heaviside step function**. This function returns a **0** if the linear combination is less than 0. It returns a **1** if the linear combination is positive or equal to zero. The Heaviside step function is shown below, where h is the calculated linear combination:



The Heaviside Step Function

Perceptron

- In the university acceptance example above, we used the weights w_{grades} =−1, w_{test} =−0.2. Since w_{grades} and w_{test} are negative values, the activation function will only return a 1 if grades and test are 0! This is because the range of values from the linear combination using these weights and inputs are (−∞,0] (i.e. negative infinity to 0, including 0 itself).
- It's easiest to see this with an example in two dimensions. In the following graph, imagine any points along the line or in the shaded area represent all the possible inputs to our node. Also imagine that the value along the y-axis is the result of performing the linear combination on these inputs and the appropriate weights. It's this result that gets passed to the activation function.
- Now remember that the step activation function returns 1 for any inputs greater than or equal to zero. As you can see in the image, only one point has a y-value greater than or equal to zero – the point right at the origin, (0,0):



Perceptron

- Now, we certainly want more than one possible grade/test combination to result in acceptance, so we need to adjust the results passed to our activation function so it activates – that is, returns 1 – for more inputs. Specifically, we need to find a way so all the scores we'd like to consider acceptable for admissions produce values greater than or equal to zero when linearly combined with the weights into our node.
- One way to get our function to return 1 for more inputs is to add a value to the results of our linear combination, called a **bias**.
- A bias, represented in equations as b, lets us move values in one direction or another.
- For example, the following diagram shows the previous hypothetical function with an added bias of +3. The blue shaded area shows all the values that now activate the function. But notice that these are produced with the same inputs as the values shown shaded in grey – just adjusted higher by adding the bias term:


```
import numpy as np
import matplotlib.pyplot as plt
def graph(formula, x_range):
    x = np.array(x_range)
    y = formula(x)
    plt.plot(x, y)
    plt.show()

def my_formula(x):
    return x*(-1)+x*(-0.2)+3

graph(my_formula, range(0, 6))
```

Perceptron

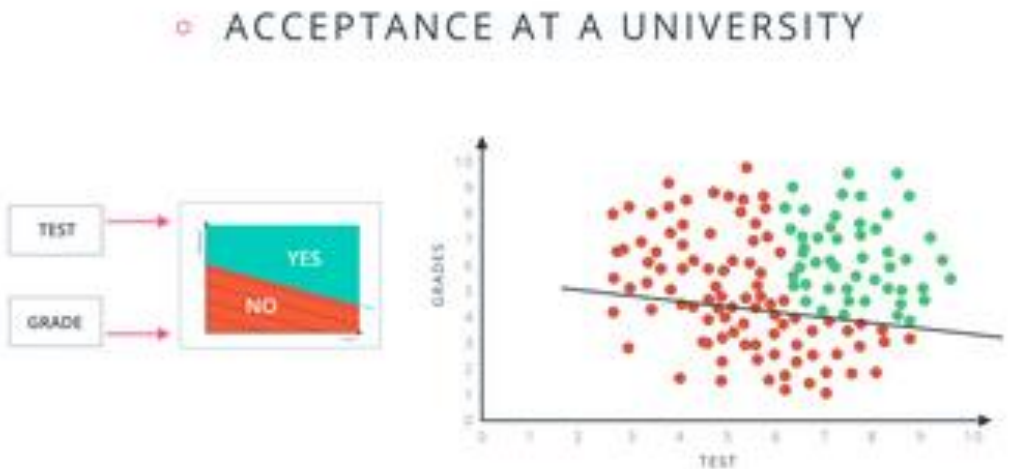
- Of course, with neural networks we won't know in advance what values to pick for biases. That's ok, because just like the weights, the bias can also be updated and changed by the neural network during training. So after adding a bias, we now have a complete perceptron formula:

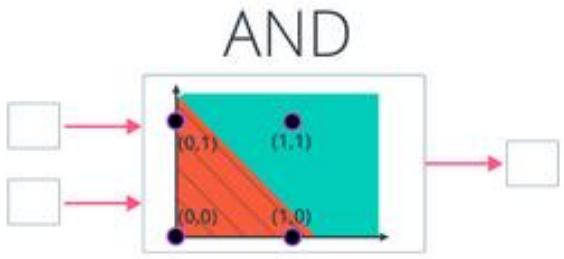
$$f(x_1, x_2, \ldots, x_m) = \begin{cases} 0 & \text{if } b + \sum w_i \cdot x_i < 0 \\ 1 & \text{if } b + \sum w_i \cdot x_i \geq 0 \end{cases}$$

Perceptron Formula

- This formula returns 1 if the input (x_1,x_2,...,x_m) belongs to the accepted-to-university category or returns 0 if it doesn't. The input is made up of one or more real numbers, each one represented by xi, where m is the number of inputs.
- Then the neural network starts to learn! Initially, the weights (wi) and bias (b) are assigned a random value, and then they are updated using a learning algorithm like gradient descent. The weights and biases change so that the next training example is more accurately categorized, and patterns in data are "learned" by the neural network.
- Now that you have a good understanding of perceptions, let's put that knowledge to use. In the next section, you'll create the AND perceptron from the *Neural Networks* video by setting the values for weights and bias.

AND Perceptron Quiz





IN	IN	OUT
1	1	1
1	0	0
0	1	0
0	0	0

What are the weights and bias for the AND perceptron?

Set the weights (weight1, weight2) and bias bias to the correct values that calculate AND operation as shown left.

In this case, there are two inputs as seen in the table above (let's call the first column input1 and the second column input2), and based on the perceptron formula, we can calculate the output.

First, the linear combination will be the sum of the weighted inputs: linear_combination = weight1*input1 + weight2*input2 then we can put this value into the biased Heaviside step function, which will give us our output (0 or 1):

$$f(x_1, x_2, \ldots, x_m) = \begin{cases} 0 & \text{if } b + \sum w_i * x_i < 0 \\ 1 & \text{otherwise} \end{cases}$$

Perceptron Formula

AND Perceptron Quiz

Code: 01AND_Perceptron_Quiz.txt

ANSWER: 01AND_Perceptron_Quiz_ANSWER.txt

```
(ncku) C:\Users\FUDE>conda install pandas
Fetching package metadata ...........
Solving package specifications: .........

Package plan for installation in environment C:\Users\FUDEV\Anaconda3\envs\ncku:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    pandas-0.20.3              |           py35_0         8.1 MB

The following NEW packages will be INSTALLED:

    pandas: 0.20.3-py35_0

Proceed ([y]/n)? y

Fetching packages ...
pandas-0.20.3- 100% |############################| Time: 0:00:00  10.76 MB/s
Extracting packages ...
```

zip- Iterate over two lists in parallel

```
alist = ['a1', 'a2', 'a3']
blist = ['b1', 'b2', 'b3']
for a, b in zip(alist, blist):
    print a, b
```

Results:

```
a1 b1
a2 b2
a3 b3
```

If you still need a hint, think of a concrete example like so:

Consider input1 and input2 both = 1, for an AND perceptron, we want the output to also equal 1! The output is determined by the weights and Heaviside step function such that

output = 1, if weight1*input1 + weight2*input2 + bias >= 0

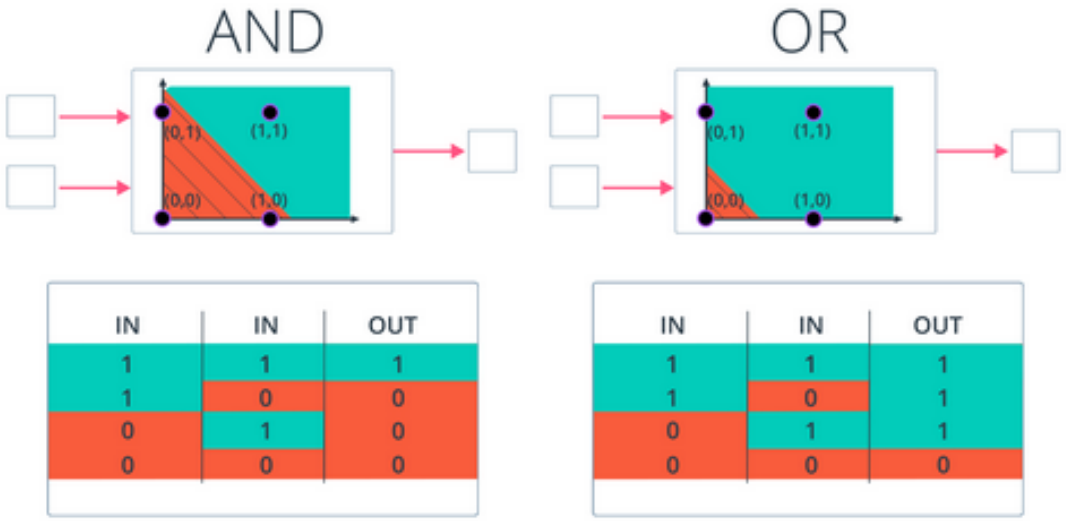
or

output = 0, if weight1*input1 + weight2*input2 + bias < 0

So, how can you choose the values for weights and bias so that if both inputs = 1, the output = 1?

OR & NOT Perceptron Quiz
-OR Perceptron

The OR perceptron is very similar to an AND perceptron. In the image below, the OR perceptron has the same line as the AND perceptron, except the line is shifted down. What can you do to the weights and/or bias to achieve this? Use the following AND perceptron to create an OR Perceptron.



Question 1 of 2

What are two ways to go from an AND perceptron to an OR perceptron?

- ☐ Increase the weights
- ☐ Decrease the weights
- ☐ Increase a single weights
- ☐ Decrease a single weights
- ☐ Increase the magnitude of the bias
- ☐ Decrease the magnitude of the bias

OR & NOT Perceptron Quiz
-NOT Perceptron

Unlike the other perceptrons we looked at, the NOT operations only cares about one input. The operation returns a 0 if the input is 1 and a 1 if it's a 0. The other inputs to the perceptron are ignored.

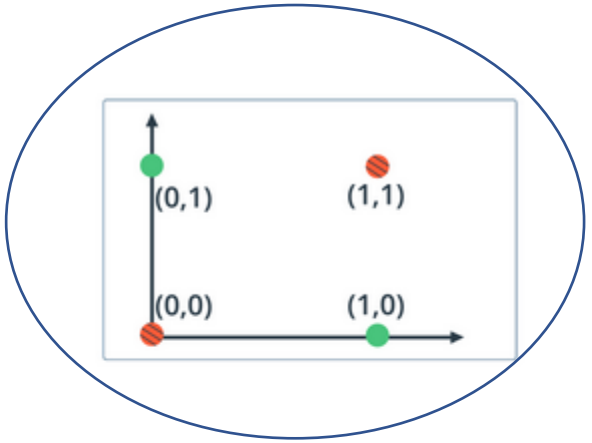
In this quiz, you'll set the weights (weight1, weight2) and bias bias to the values that calculate the NOT operation on the second input and ignores the first input.

Code: 02OR_&_NOT_Perceptron_Quiz.txt

ANSWER: 02OR_&_NOT_Perceptron_Quiz_ANSWER.txt

We have a perceptron that can do AND, OR, or NOT operations. Let's do one more, XOR. In the next section, you'll learn how a neural network solves more complicated problems like XOR.

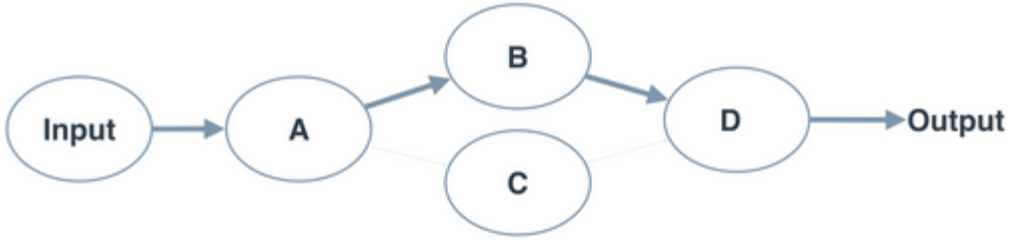
XOR Perceptron Quiz



IN	IN	OUT
1	1	0
1	0	1
0	1	1
0	0	0

An XOR perceptron is a logic gate that outputs 0 if the inputs are the same and 1 if the inputs are different. Unlike previous perceptrons, this graph isn't linearly separable. To handle more complex problems like this, we can chain perceptrons together.

Let's build a neural network from the AND, NOT, and OR perceptrons to create XOR logic. Let's first go over what a neural network looks like.



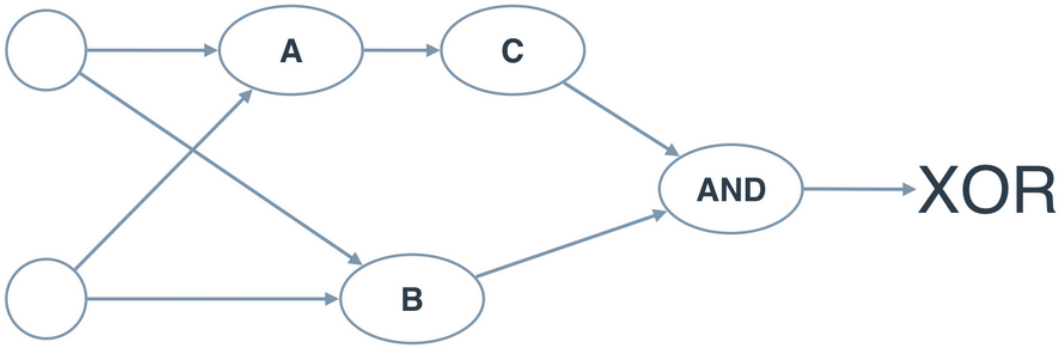
The above neural network contains 4 perceptrons, A, B, C, and D. The input to the neural network is from the first node. The output comes out of the last node. The weights are based on the line thickness between the perceptrons. Any link between perceptrons with a low weight, like A to C, you can ignore. For perceptron C, you can ignore all input to and from it. For simplicity we wont be showing bias, but it's still in the neural network.

Quiz: Build an XOR Multi-Layer Perceptron

Now, let's build a multi-layer perceptron from the AND, NOT, and OR perceptrons to create XOR logic!

The neural network below contains 3 perceptrons, A, B, and C. The last one (AND) has been given for you. The input to the neural network is from the first node. The output comes out of the last node.

The multi-layer perceptron above calculates XOR. Each perceptron is a logic operation of AND, OR, and NOT. However, the perceptrons A, B, and C don't indicate their operation. In the following quiz, set the correct operations for the four perceptrons to calculate XOR.



Quiz Question

Set the operations for the perceptrons in the XOR neural network?

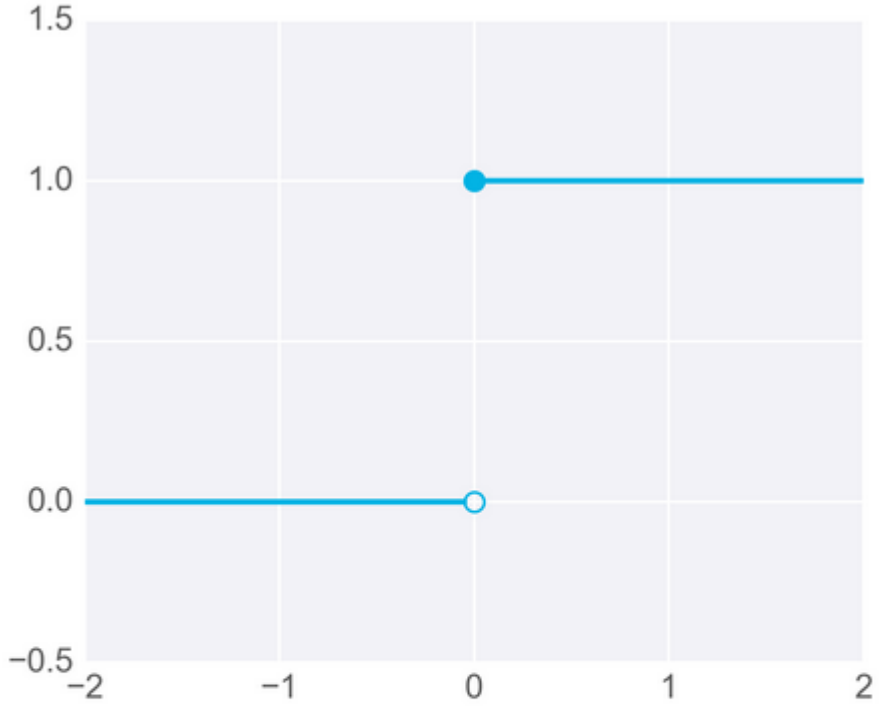
NOT AND OR

PERCEPTRON	OPERATIONS
A	
B	
C	

The Simplest Neural Network

The simplest neural network

So far you've been working with perceptrons where the output is always one or zero. The input to the output unit is passed through an activation function, *f(h)*, in this case, the step function.



$$f(h) = \begin{cases} 0 & \text{if } h < 0 \\ 1 & \text{if } h \geq 0 \end{cases}$$

The step activation function.

The Simplest Neural Network

The output unit returns the result of f(h), where h is the input to the output unit:

$$h = \sum_i w_i x_i + b$$

The diagram below shows a simple network. The linear combination of the weights, inputs, and bias form the input h, which passes through the activation function f(h), giving the final output of the perceptron, labeled y.

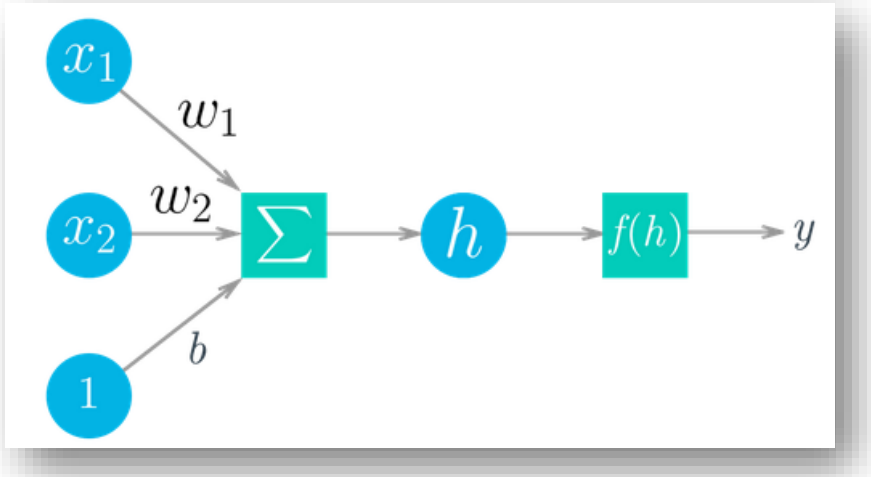


Diagram of a simple neural network. Circles are units, boxes are operations.

The Simplest Neural Network

The cool part about this architecture, and what makes neural networks possible, is that the activation function, f(h) can be *any function*, not just the step function shown earlier.

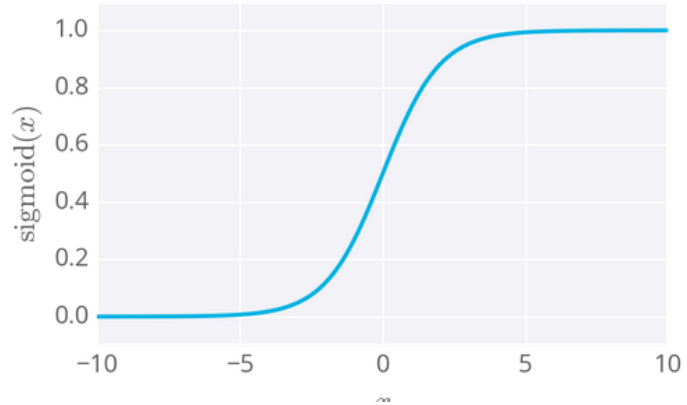
For example, if you let f(h)=h, the output will be the same as the input. Now the output of the network is

$$y = \sum_i w_i x_i + b$$

This equation should be familiar to you, it's the same as the linear regression model!

Other activation functions you'll see are the logistic (often called the sigmoid), tanh, and softmax functions. We'll mostly be using the sigmoid function for the rest of this lesson:

$$sigmoid(x) = 1/(1 + e^{-x})$$



The sigmoid function

The Simplest Neural Network

Simple network exercise

Below you'll use NumPy to calculate the output of a simple network with two input nodes and one output node with a sigmoid activation function. Things you'll need to do:

- Implement the sigmoid function.
- Calculate the output of the network.

The sigmoid function is
$sigmoid(x) = 1/(1 + e^{-x})$

For the exponential, you can use Numpy's exponential function, np.exp.

And the output of the network is

$$y = f(x) = sigmoid(\sum_i w_i x_i + b)$$

For the weights sum, you can do a simple element-wise multiplication and sum, or use NumPy's dot product function.

Code: 03The_Simplest_Neural_Network.txt

ANSWER: 03The_Simplest_Neural_Network_ANSWER.txt

Gradient Descent

Learning weights

You've seen how you can use perceptrons for AND and XOR operations, but there we set the weights by hand. What if you want to perform an operation, such as predicting college admission, but don't know the correct weights? You'll need to learn the weights from example data, then use those weights to make the predictions.

To figure out how we're going to find these weights, start by thinking about the goal. We want the network to make predictions as close as possible to the real values. To measure this, we need a metric of how wrong the predictions are, the **error**. A common metric is the sum of the squared errors (SSE):

$$E = \frac{1}{2}\sum_u \sum_j [y_j^u - \hat{y}_j^u]^2$$

Gradient Descent

$$E = \frac{1}{2}\sum_u \sum_j [y_j^u - \hat{y}_j^u]^2$$

where $\hat{y}$ is the prediction and y is the true value, and you take the sum over all output units j and another sum over all data points μ. This might seem like a really complicated equation at first, but it's fairly simple once you understand the symbols and can say what's going on in words.

First, the inside sum over j. This variable j represents the output units of the network. So this inside sum is saying for each output unit, find the difference between the true value y and the predicted value from the network $\hat{y}$, then square the difference, then sum up all those squares.

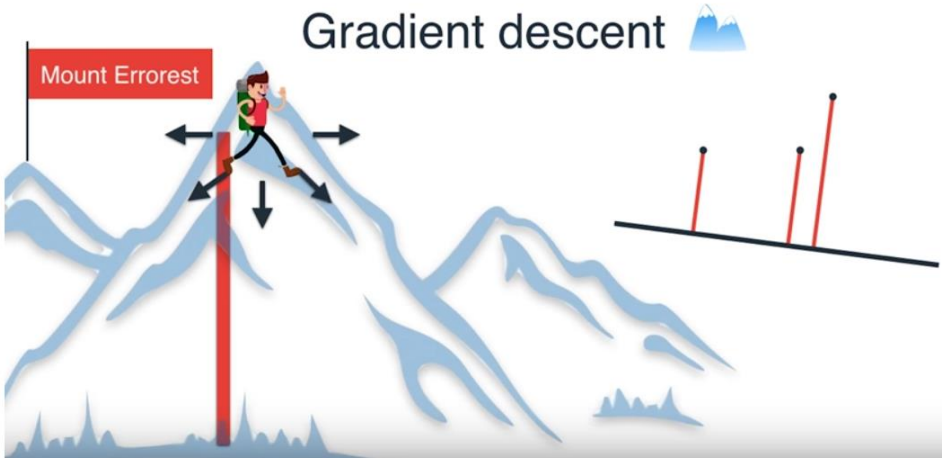
Then the other sum over μ is a sum over all the data points. So, for each data point you calculate the inner sum of the squared differences for each output unit. Then you sum up those squared differences for each data point. That gives you the overall error for all the output predictions for all the data points.

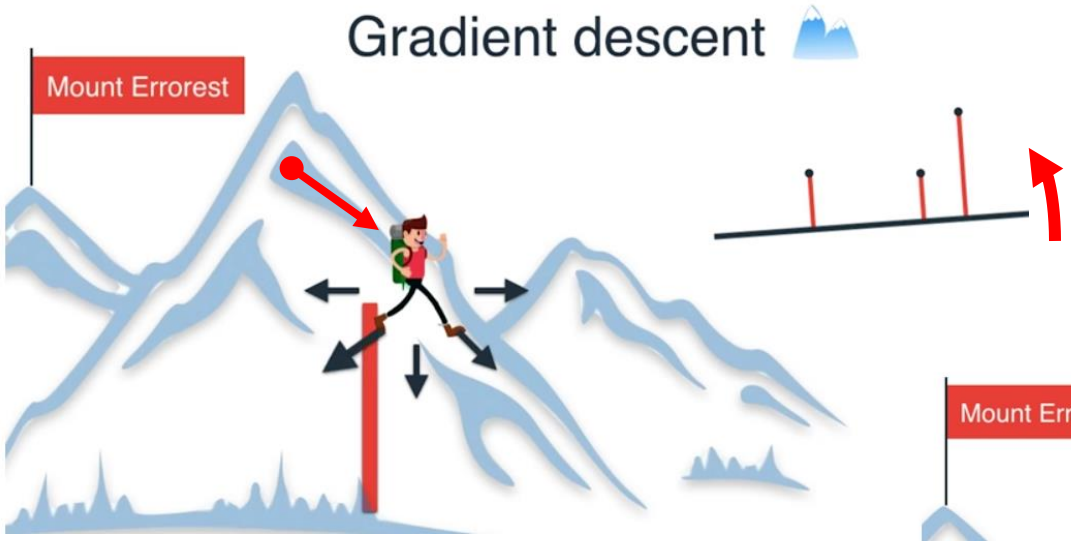
Gradient Descent

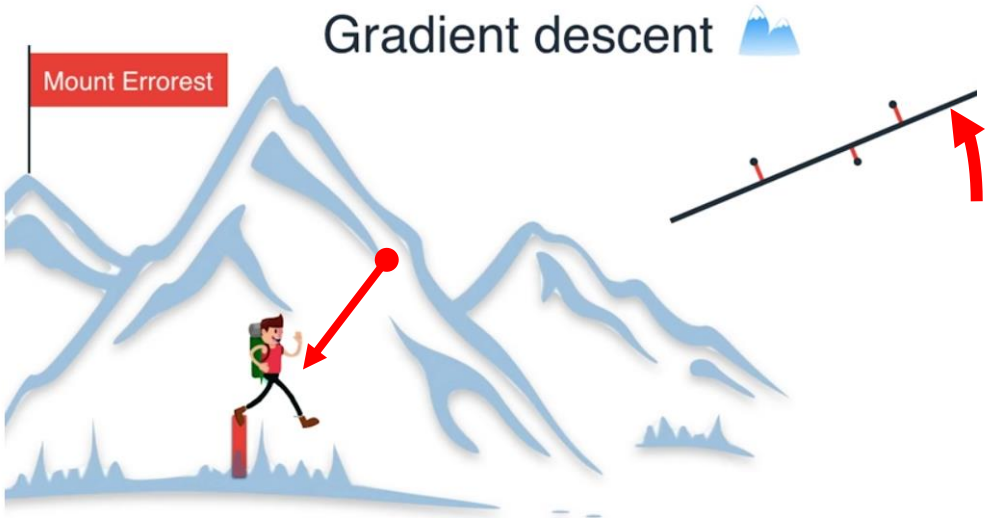
The SSE is a good choice for a few reasons. The square ensures the error is always positive and larger errors are penalized more than smaller errors. Also, it makes the math nice, always a plus.

Remember that the output of a neural network, the prediction, depends on the weights $\hat{y}_j^u = f(\sum_i w_{ij} x_i^u)$ and accordingly the error depends on the weights $E = \frac{1}{2}\sum_u \sum_j [\, y_j^u - f(\sum_i w_{ij} x_i^u)]^2$

We want the network's prediction error to be as small as possible and the weights are the knobs we can use to make that happen. Our goal is to find weights wij that minimize the squared error E. To do this with a neural network, typically you'd use **gradient descent**.


Gradient descent
Mount Errorest

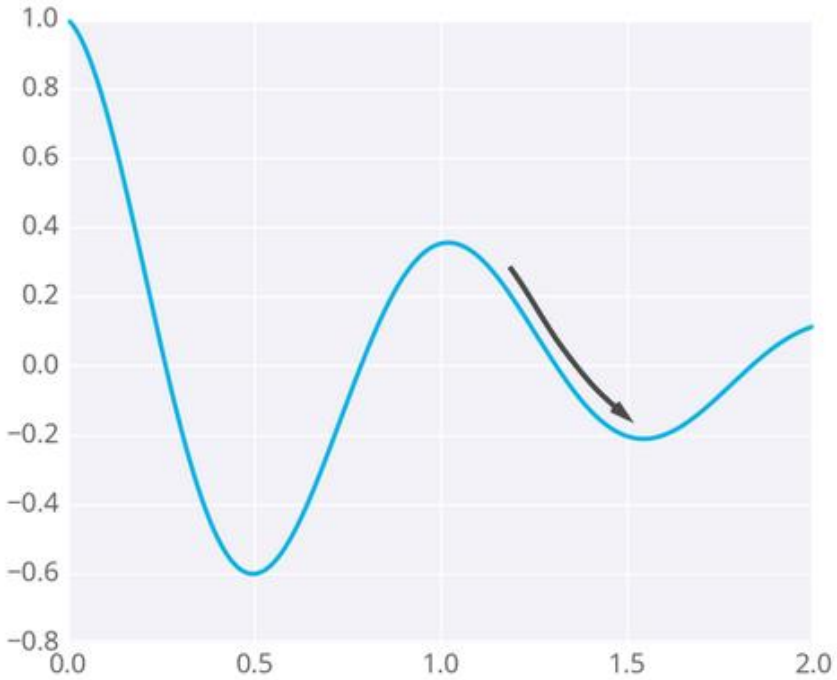

Gradient descent
Mount Errorest


Gradient descent
Mount Errorest

Gradient Descent

Caveats

Since the weights will just go where ever the gradient takes them, they can end up where the error is low, but not the lowest. These spots are called local minima. If the weights are initialized with the wrong values, gradient descent could lead the weights into a local minimum, illustrated below.



Gradient descent leading into a local minimum

There are methods to avoid this, such as using momentum

Next Step:

Gradient Descent: The Math、The Code、Implementing

Multilayer Perceptrons

Backpropagation