Decision Tree – Regression

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed

The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

Predictors

Target

Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	46
Sunny	Mild	High	False	45
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	46
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

Decision Tree Algorithm

The core algorithm for building decision trees called **ID3** by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with *Standard Deviation Reduction*.

Standard Deviation

$$\textbf{Standard Deviation} = S = \sqrt{\frac{\sum(x - \bar{x})^2}{n}}$$

$$\textbf{Coeffeicient of Variation} = CV = \frac{S}{\bar{x}} * 100\%$$

a) Standard deviation for **one** attribute:

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$\boldsymbol{Count} = n = 14$$

$$\boldsymbol{Average} = \bar{x} = \frac{\sum x}{n} = 39.8$$

$$\boldsymbol{Standard\ Deviation} = S = \sqrt{\frac{\sum(x - \bar{x})^2}{n}} = 9.32$$

$$\boldsymbol{Coeffeicient\ of\ Variation} = CV = \frac{S}{\bar{x}} * 100\% = 23\%$$

- Standard Deviation (**S**) is for tree building (branching).
- Coefficient of Deviation (**CV**) is used to decide when to stop branching. We can use Count (**n**) as well.
- Average (**Avg**) is the value in the leaf nodes.

b) Standard deviation for **two** attributes (target and predictor):

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)*__S__(Sunny) + **P**(Overcast)*__S__(Overcast) + **P**(Rainy)*__S__(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

Standard Deviation Reduction

The standard deviation reduction is based on the decrease in standard deviation after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches).

Step 1: The standard deviation of the target is calculated.

Standard deviation (Hours Played) = 9.32

Step 2:
The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)*__S__(Sunny) + **P**(Overcast)*__S__(Overcast) + **P**(Rainy)*__S__(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR= 0.48		

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

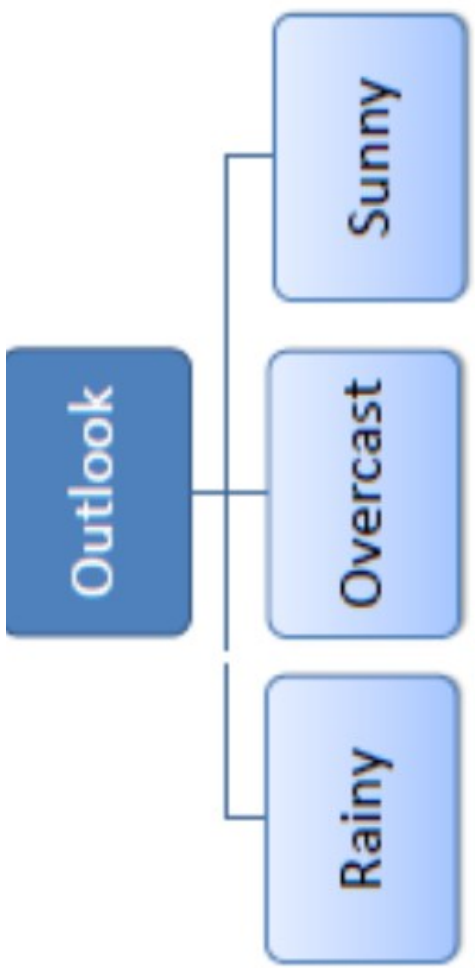
		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

★		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

Step 4a:

The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.



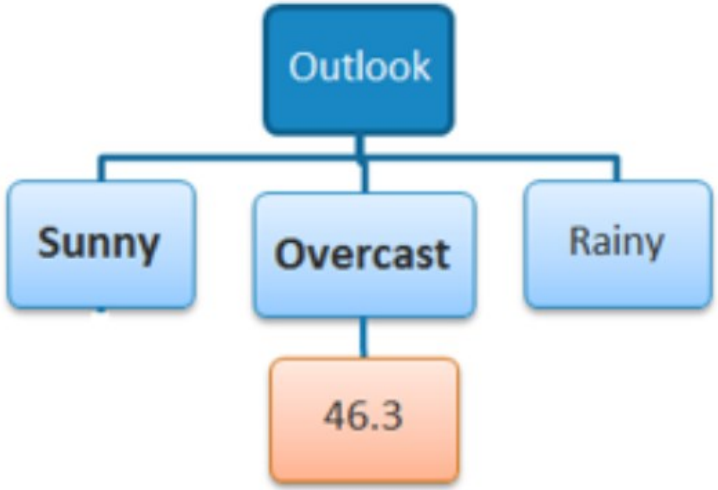
Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48

In practice, we need some termination criteria. For example, when coefficient of deviation (CV) for a branch becomes smaller than a certain threshold (e.g., 10%) and/or when too few instances (n) remain in the branch (e.g., 3).

Step 4b: "Overcast" subset does not need any further splitting because its CV (8%) is less than the threshold (10%). The related leaf node gets the average of the "Overcast" subset.

Outlook - Overcast

		Hours Played (StDev)	Hours Played (AVG)	Hours Played (CV)	Count
Outlook	Overcast	3.49	46.3	8%	4
	Rainy	7.78	35.2	22%	5
	Sunny	10.87	39.2	28%	5



Step 4c: However, the "Sunny" branch has an CV (28%) more than the threshold (10%) which needs further splitting. We select "Temp" as the best best node after "Outlook" because it has the largest SDR.

Outlook - Sunny

Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	**45**
Cool	Normal	FALSE	**52**
Cool	Normal	TRUE	**23**
Mild	Normal	FALSE	**46**
Mild	High	TRUE	**30**
			S = 10.87
			AVG = 39.2
			CV = 28%

		Hours Played (StDev)	Count
Temp	Cool	14.50	2
	Mild	7.32	3

SDR = 10.87-((2/5)*14.5 + (3/5)*7.32) = **0.678**

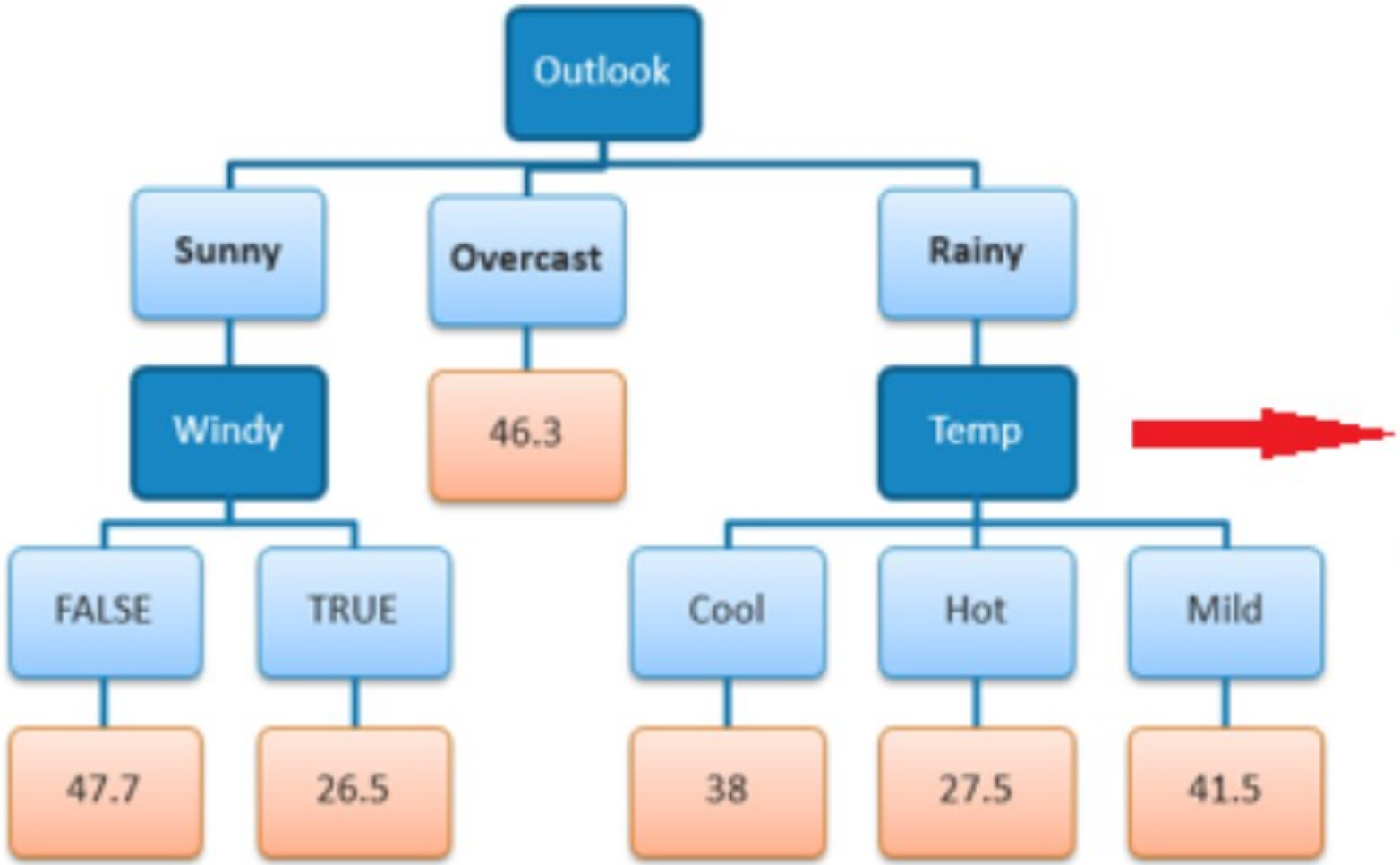
		Hours Played (StDev)	Count
Humidity	High	7.50	2
	Normal	12.50	3

SDR = 10.87-((2/5)*7.5 + (3/5)*12.5) = **0.370**

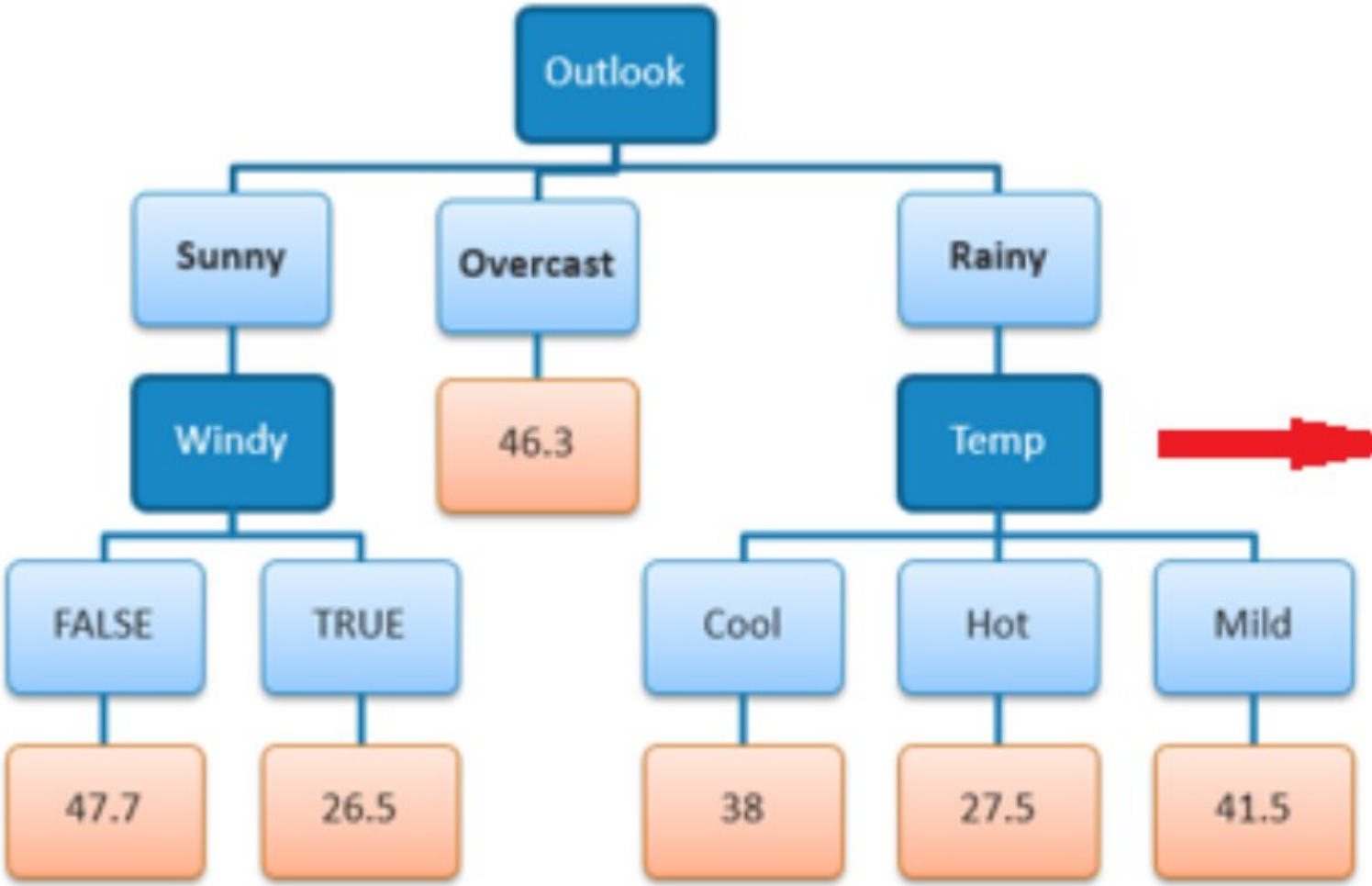
		Hours Played (StDev)	Count
Windy	False	3.09	3
	True	3.50	2

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5) = **7.62**

Because the number of data points for all three branches (Cool, Hot and Mild) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.



Temp	Hours Played
Cool	38
Hot	25
Hot	30
Mild	35
Mild	48



Temp	Hours Played
Cool	38
Hot	25
Hot	30
Mild	35
Mild	48

When the number of instances is more than one at a *leaf node* we calculate the *average* as the final value for the target.

In machine learning, the terms training data, testing data, and validation data refer to different subsets of a dataset used for various stages in the development and evaluation of a model. These subsets play crucial roles in training and assessing the performance of machine learning models. Here's a brief explanation of each:

Training Data:

- **Purpose:** This is the subset of data used to train the machine learning model.
- **Usage:** During the training phase, the model learns patterns and relationships within the training data.
- **Size:** The training dataset is typically the largest portion of the overall dataset.

Testing Data (or Test Data):

- **Purpose:** This is a separate subset of data used to evaluate the performance of the model after it has been trained.
- **Usage:** The model has never seen the testing data during training, so its ability to generalize to new, unseen examples is assessed using the testing data.
- **Size:** The testing dataset is held out until the model has been trained to avoid biasing the evaluation.

Validation Data:

•**Purpose:** This is another subset of data used to fine-tune the model during the training phase.

•**Usage:** It helps to assess the model's performance on data it has not seen during training and provides a basis for adjusting hyperparameters to improve generalization.

•**Size:** The validation dataset is used to make decisions about the model's architecture or hyperparameters, and it is not used in the final evaluation of the model's performance.

The typical split among these subsets can vary, but a common practice is to use a large portion (e.g., 70-80%) for training, a smaller portion (e.g., 10-15%) for testing, and the rest for validation. The exact split depends on factors such as the size of the dataset and the specific requirements of the machine learning task. The key idea is to ensure that the model is trained on diverse data, tested on unseen data, and validated to improve its generalization capabilities.

Validation data plays a crucial role in the training process of a machine learning model, particularly in the context of fine-tuning and improving its performance. Here are more details about validation data:

Purpose of Validation Data:

•**Fine-Tuning Hyperparameters:** During the training phase, a machine learning model has various hyperparameters (e.g., learning rate, regularization strength) that need to be set. The goal is to find the combination of hyperparameter values that results in the best model performance. The validation dataset is used to evaluate the model's performance for different hyperparameter settings, helping to choose the best configuration.

•**Preventing Overfitting:** Overfitting occurs when a model performs well on the training data but fails to generalize to new, unseen data. The validation data helps monitor the model's performance on examples it hasn't seen during training. If the model performs well on the training data but poorly on the validation data, it may be overfitting. Adjustments can then be made to prevent overfitting, such as reducing model complexity or applying regularization techniques.

Size of Validation Data:

The size of the validation dataset is crucial. It should be large enough to provide a representative sample of the data but not so large that it significantly reduces the amount of data available for training. Common splits include 80% for training, 10% for validation, and 10% for testing.

validation data is used to fine-tune a model during training, making adjustments to hyperparameters and preventing overfitting. It helps ensure that the model generalizes well to new, unseen data by providing an unbiased evaluation throughout the training process.

Cross Validation

- in a real-life scenario, the model will be tested for its efficiency and accuracy with an altogether different and unique data set.
- Under those circumstances, you'd want your model to be efficient enough or at least to be at par with the same efficiency that it shows for the training set.
- Basically this testing is known as cross-validation in Machine Learning so that it is fit to work with any model in the future.

Cross Validation

- Cross validation is a technique for assessing how the statistical analysis generalizes to an independent data set.
- It is a technique for evaluating machine learning models by training several models on subsets of the available input data and evaluating them on the complementary subset of the data.
- Using cross-validation, there are high chances that **we can detect over-fitting with ease.**

Types Of Cross-Validation

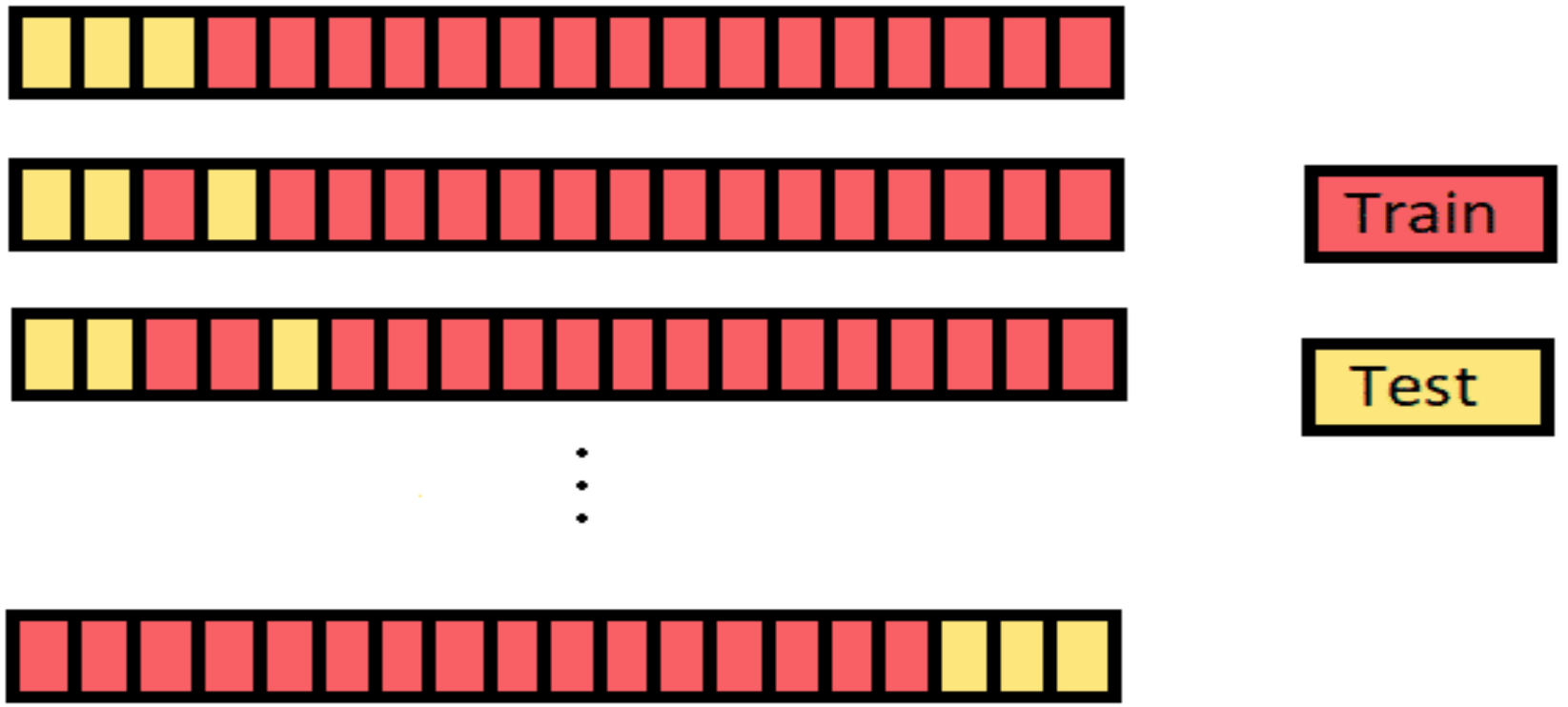
There are two types of cross-validation techniques in Machine Learning.

- **Exhaustive Cross-Validation** – This method basically involves testing the model in all possible ways, it is done by dividing the original data set into training and validation sets. Example: Leave-p-out Cross-Validation, Leave-one-out Cross-validation.
- **Non-Exhaustive Cross-Validation** – In this method, the original data set is not separated into all the possible permutations and combinations. Example: K-fold Cross-Validation, Holdout Method.

Various Types of cross validation

- There are several **cross validation techniques** such as :-
- 1. Leave One-out Cross Validation
  2. Leave P-out Cross Validation
  3. K-Fold Cross Validation
  4. Stratified K-Fold Cross Validation
  5. Holdout Method

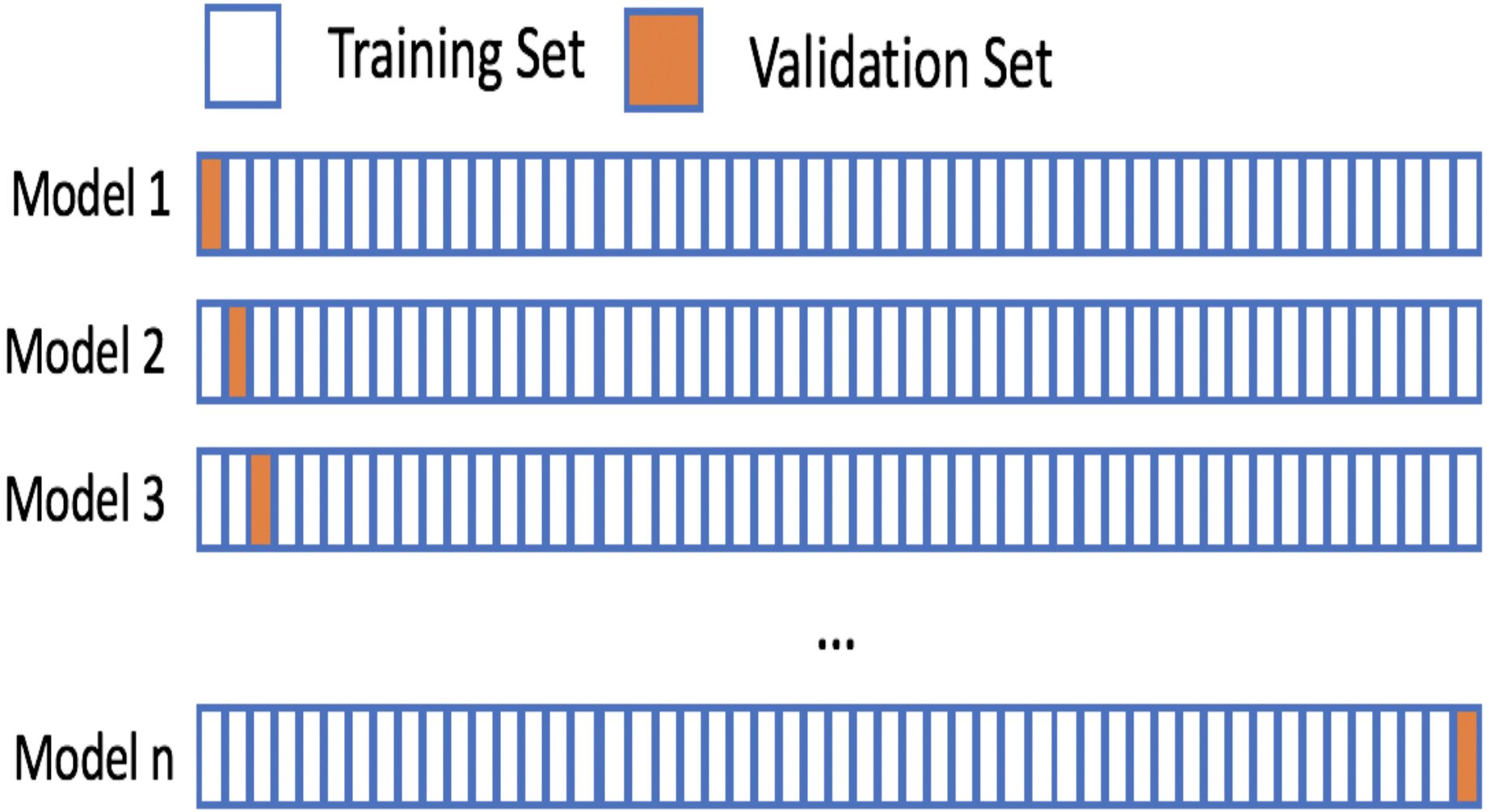
Leave P Out



Leave P out

- In this approach, **p** data points are left out of the training data. Let's say there are **m** data points in the data set, then **m-p** data points are used for the training phase. And the **p** data points are kept as the validation set.
- This technique is rather exhaustive because the above process is repeated for all the possible combinations in the original data set. To check the overall effectiveness of the model, the error is averaged for all the trials.
- It becomes computationally infeasible since the model needs to train and validate for all possible combinations and for a considerably large **p**.

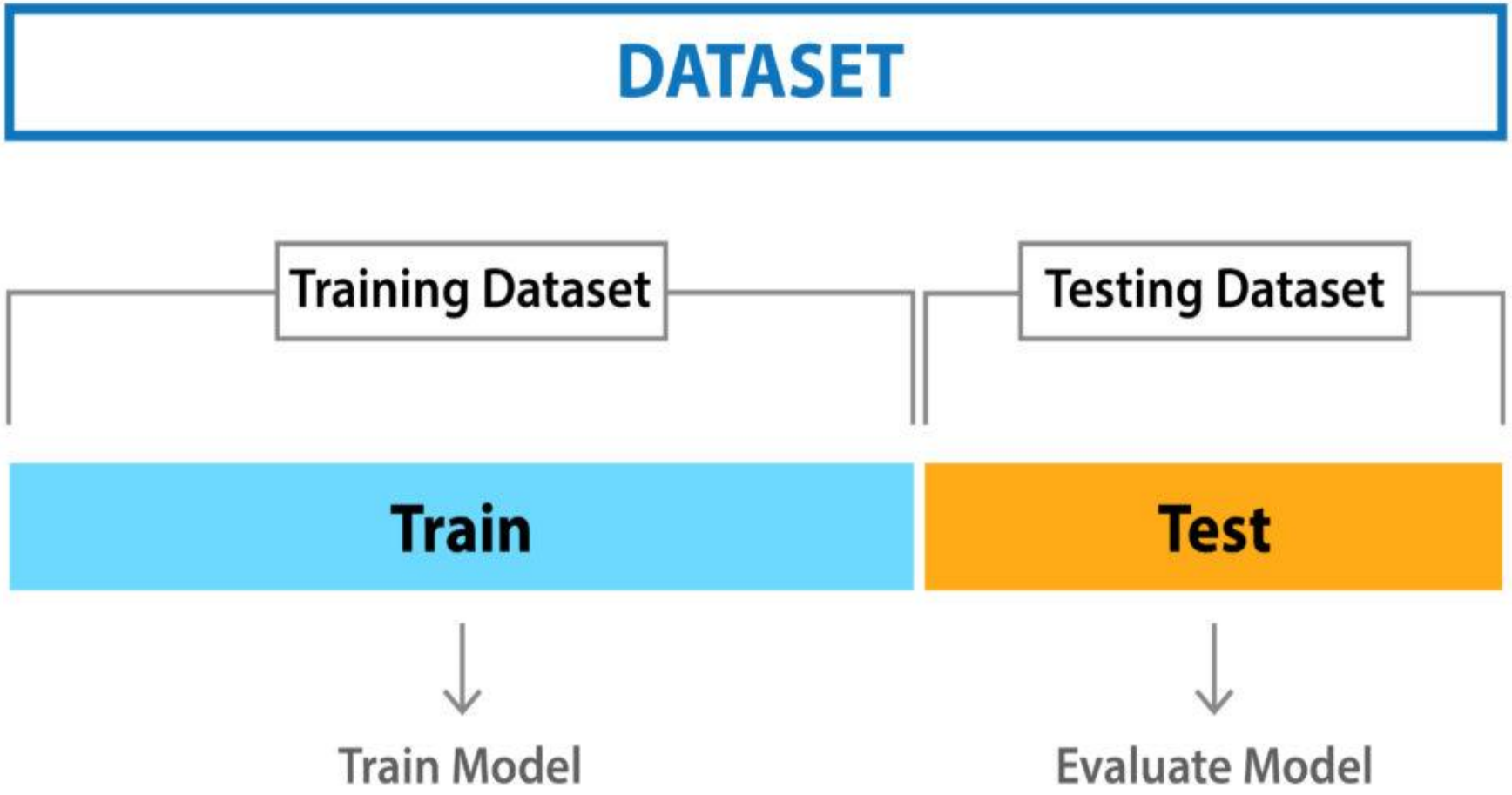
Leave 1 out



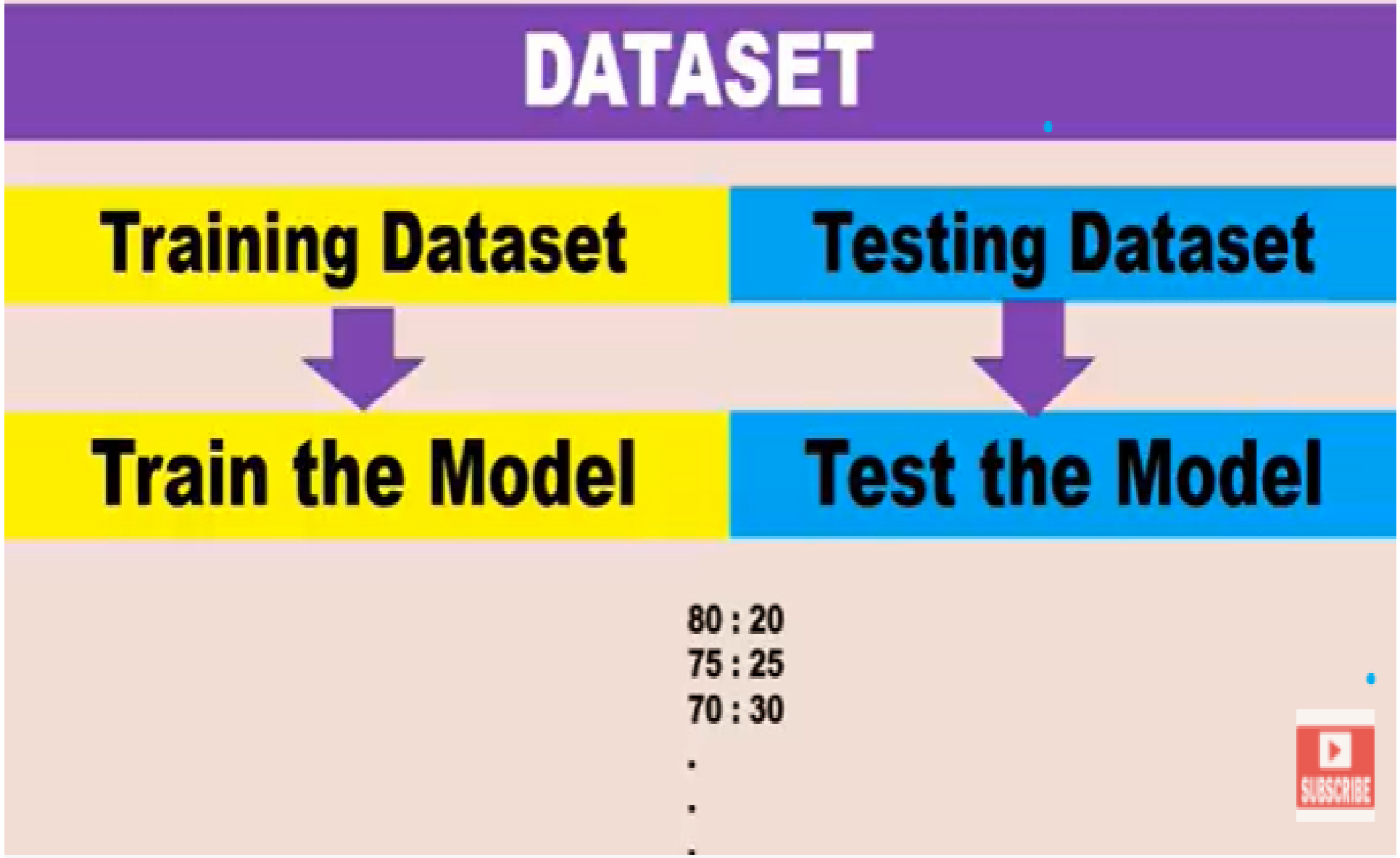
Leave 1 Out

- This method of Cross-validation is similar to Leave-p-out Cross-validation but the only difference is that in this case **p = 1**. It actually saves a lot of time which is a big advantage.
- Although If the sample data is too large, it can still take a lot of time. But it would still be quicker than the Leave-p-out cross-validation method.

Hold Out Method



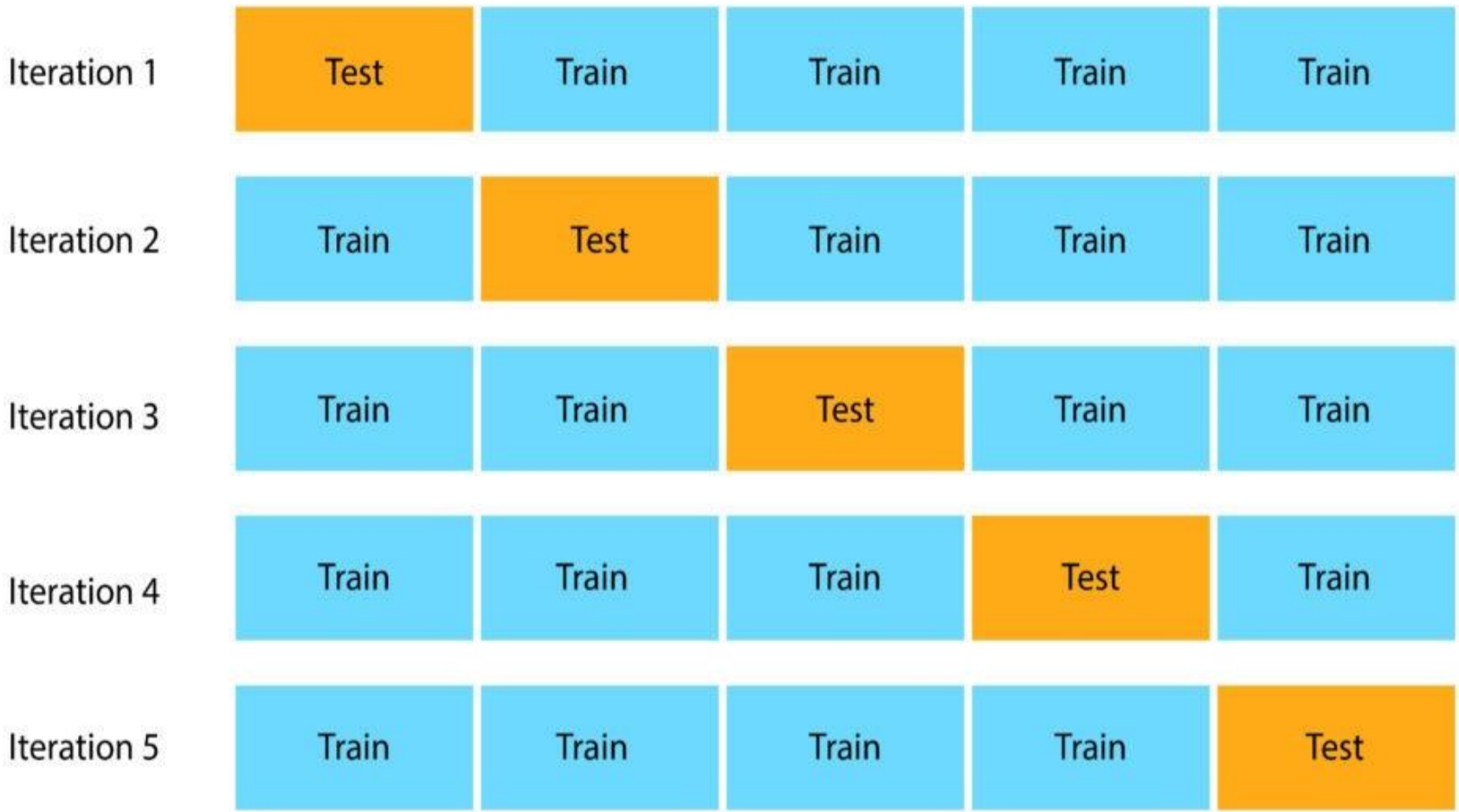
HOLD-OUT METHOD



Hold Out Method

- This is a quite basic and simple approach in which we divide our entire dataset into two parts viz- training data and testing data.
- As the name, we train the model on training data and then evaluate on the testing set.
- Usually, the size of training data is set more than twice that of testing data, so the data is split in the ratio of 70:30 or 80:20.
- In this approach, the data is first shuffled randomly before splitting.
- As the model is trained on a different combination of data points, the model can give different results every time we train it, and this can be a cause of instability.
- Also, we can never assure that the train set we picked is representative of the whole dataset.

K fold cross validation



- K-fold cross validation is one way to improve the holdout method. This method guarantees that the score of our model does not depend on the way we picked the train and test set.
- The data set is divided into k number of subsets and the holdout method is repeated k number of times. Let us go through this in steps:
- Randomly split your entire dataset into k number of folds (subsets)
- For each fold in your dataset, build your model on k – 1 folds of the dataset. Then, test the model to check the effectiveness for kth fold
- Repeat this until each of the k-folds has served as the test set
- The average of your k recorded accuracy is called the cross-validation accuracy and will serve as your performance metric for the model.

Limitations Of Cross-Validation

The following are a few limitations faced by Cross-Validation:

- In an ideal situation, Cross-Validation will produce optimum results. But in case of **inconsistent data**, the results may vary drastically. It is quite uncertain what kind of data will be encountered by the model.

- Predictive modeling often requires an **evolution in terms of data**, this can pretty much change the training and the validation sets drastically.

- The results may **vary depending upon the features of the data set**. Let us say we make a predictive model to detect an ailment in a person and we train it with a specific set of population. It may vary with the general population causing inconsistency and reduced efficiency.
-

Cross-Validation Applications

- With the overpowering applications to prevent a Machine Learning model from Overfitting and Underfitting, there are several other applications of Cross-Validation listed below:
- We can use it to compare the performances of a set of predictive modeling procedures.
- Cross-Validation excels in the field of medical research.
- It can be used in the meta-analysis since a lot of data analysts are already using cross-validation.

Consider the dataset, S given below:

Elevation	Road Type	Speed Limit	Speed
steep	Uneven	Yes	Slow
steep	Smooth	Yes	Slow
flat	Uneven	No	Fast
steep	Smooth	No	Fast

Elevation, Road Type and speed Limit are the features and Speed is the target label that we want to predict.

Find the entropy of the dataset, S as given above:

a. 0.5
b. 0
c. 1
d. 0.7

Consider the dataset, S given below:

Elevation	Road Type	Speed Limit	Speed
steep	Uneven	Yes	Slow
steep	Smooth	Yes	Slow
flat	Uneven	No	Fast
steep	Smooth	No	Fast

Find the information Gain if the dataset is split at the feature "Elevation":

a. 1
b. 0
c. 0.675
d. 0.325

Consider the dataset, S given below:

Elevation	Road Type	Speed Limit	Speed
steep	Uneven	Yes	Slow
steep	Smooth	Yes	Slow
flat	Uneven	No	Fast
steep	Smooth	No	Fast

Find the feature on which the parent node must be chosen to split the dataset, S based on information gain:

a. Speed Limit
b. Road Type
c. Elevation

Pruning is a technique that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. This is done in order to avoid:

a. overfitting
b. underfitting

You have been given the following 2 statements. Find out which of these options is/are true in case of k-NN?

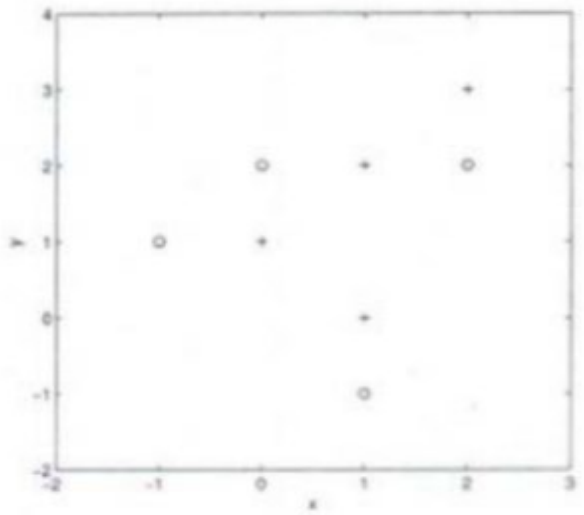
1. In case of very large value of k, we may include points from other classes into the neighborhood.
2. In case of too small value of k, the algorithm is very sensitive to noise.

a. 1 is True and 2 is False
b. 1 is False and 2 is True
c. Both are True
d. Both are False

Suppose, you have given the following data where x and y are the 2 input variables and Class is the dependent variable.

x	y	Class
-1	1	$-$
0	1	$+$
0	2	$-$
1	-1	$-$
1	0	$+$
1	2	$+$
2	2	$-$
2	3	$+$

Below is a scatter plot which shows the above data in 2D space.



Suppose, you want to predict the class of new data point x=1 and y=1 using Eucledian distance in 3-NN. In which class this data point belong to?

a. + Class
b. – Class
c. Can't Say
d. None of these

Which of the following necessitates feature reduction in machine learning?

a. Irrelevant and redundant features
b. Limited training data
c. Limited computational resources.
d. All of the above

Cross validation is a model evaluation method. **Leave-one-out cross validation**(LOOCV) is K-fold cross validation taken to its logical extreme, with K equal to N, the number of data points in the set. That means that N separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. Thus, it iterates over the other datapoints keeping the rest of the dataset fixed. What can be the major issues in LOOCV?

a. low variance
b. high variance
c. faster run time compared to K-fold cross validation
d. slower run time compared to K-fold cross validation

Consider the following dataset. a,b,c are the features and K is the class(1/0):

a	b	c	K
1	0	1	1
1	1	1	1
0	1	1	0
1	1	0	0
1	0	1	0
0	0	0	1

Classify the test instance given below into class 1/0 using a Naive Bayes Classifier.

a	b	c	K
0	0	1	?

a. 0
b. 1

Consider the following dataset. a,b,c are the features and K is the class(1/0):

a	b	c	K
1	0	1	1
1	1	1	1
0	1	1	0
1	1	0	0
1	0	1	0
0	0	0	1

Find P(K=0| a=1,b=1).

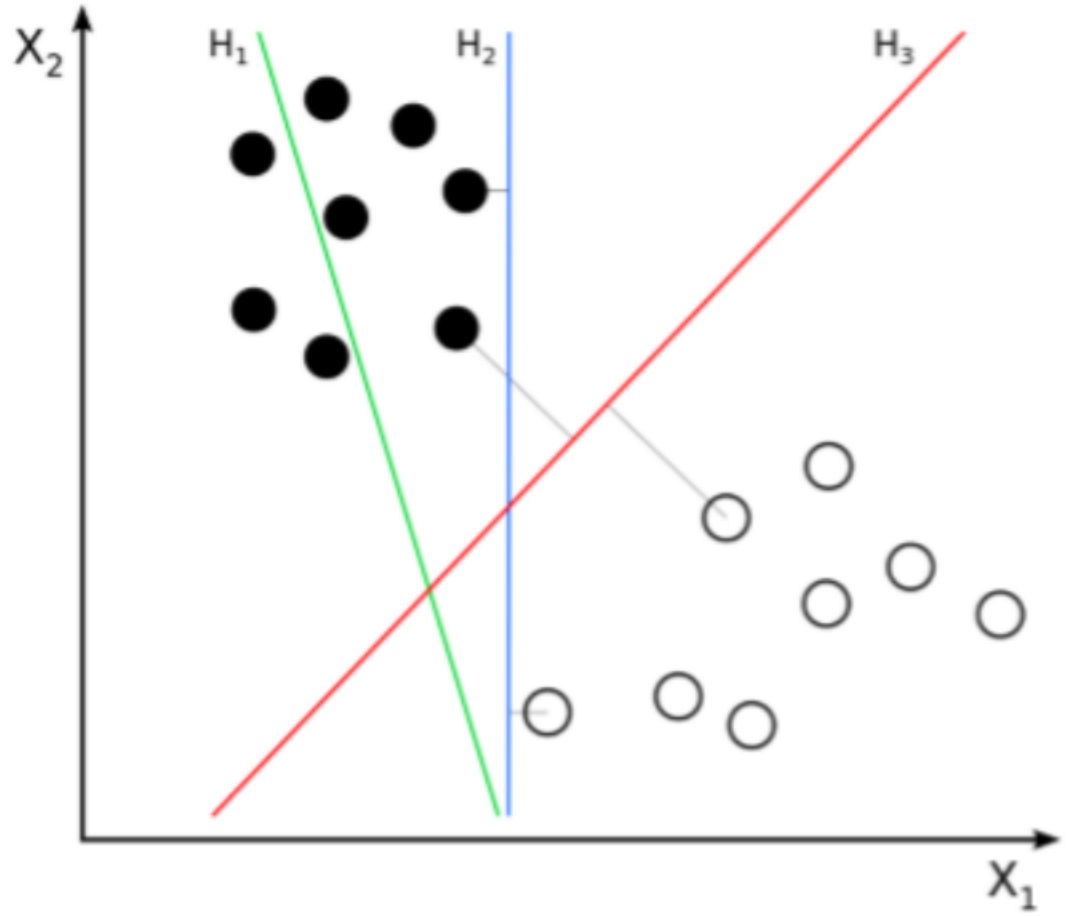
a. $\frac{1}{3}$

b. $\frac{2}{3}$

c. $\frac{1}{9}$

d. $\frac{8}{9}$

Consider the data-points in the figure below.



Let us assume that the black-colored circles represent positive class whereas the white-colored circles represent negative class. Which of the following among H1, H2 and H3 is the maximum-margin hyperplane?

(a) H1

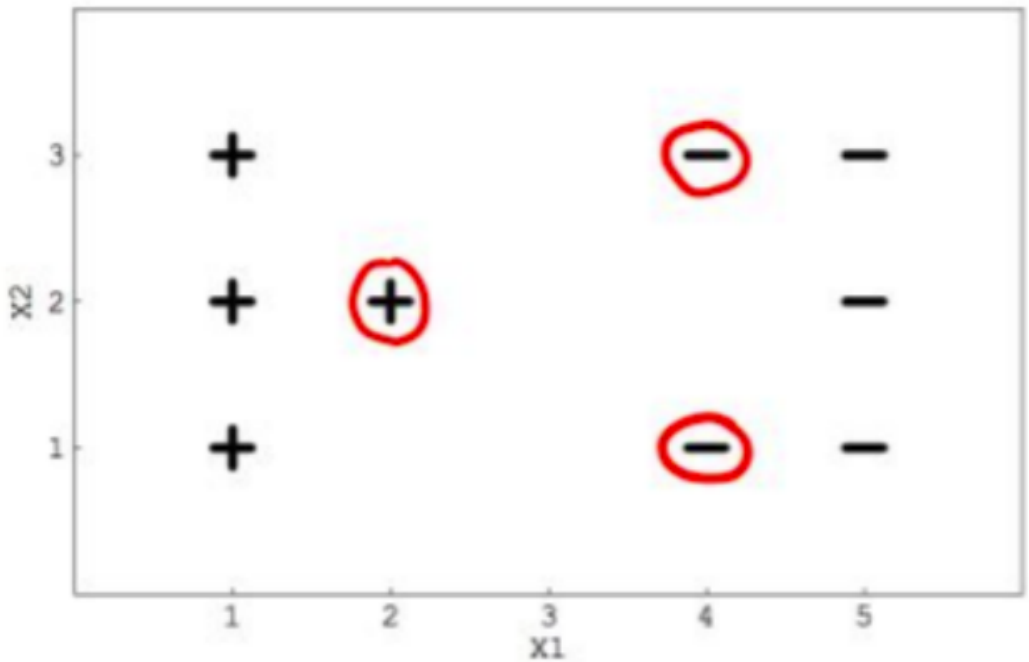
(b) H2

(c) H3

The soft margin SVM is more preferred than the hard-margin svm when:

1. The data is linearly separable
2. The data is noisy and contains overlapping point

Suppose you are using a Linear SVM classifier with 2 class classification problem. Consider the following data in which the points circled red represent support vectors.



Will the decision boundary change if any of the red points are removed?

a. Yes
b. No

Introduction

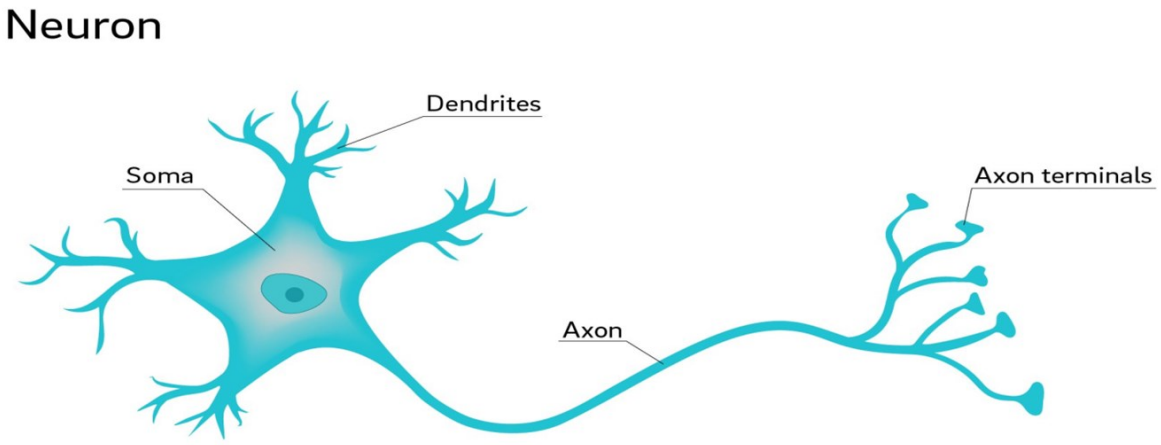
- Humans have made several attempts to mimic the biological systems, and one of them is artificial neural networks inspired by the biological neural networks in living organisms.
- However, they are very much different in several ways.
- For example, the birds had inspired humans to create airplanes, and the four-legged animals inspired us to develop cars.

Neural Network

- The brain is the control unit of the neural network,
- It has different subunits that take care of vision, senses, movement, and hearing.
- The brain is connected with a dense network of nerves to the rest of the body's sensors and actors.
- There are approximately 10^{11} neurons in the brain, and these are the building blocks of the complete central nervous system of the living body.

Perceptron

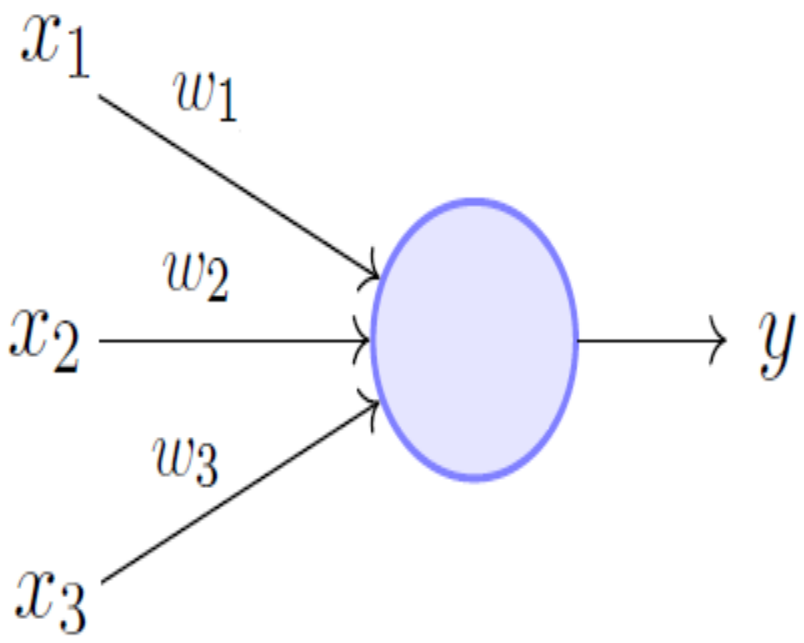
- Perceptrons, who are the predecessors of artificial neurons, were created to mimic certain parts of a biological neuron such as dendrite, axon, and cell body using mathematical models, electronics, and whatever limited information we have of biological neural networks.



Perceptron continued…

- A neuron comprises three major parts: the cell body (also called Soma), the dendrites, and the axon. The dendrites are like fibers branched in different directions and are connected to many cells
- Dendrites receive the signals from surrounding neurons, and the axon transmits the signal to the other neurons.
- At the ending terminal of the axon, the contact with the dendrite is made through a synapse
- Axon is a long fiber that transports the output signal as electric impulses along its length. Each neuron has one axon. Axons pass impulses from one neuron to another like a domino effect.

An artificial neuron



Perceptron Model (Minsky-Papert in 1969)

Biological Neural Networks vs Artificial Neural Networks

- The human brain consists of about 86 billion neurons and more than 100 trillion synapses. In artificial neural networks, the number of neurons is about 10 to 1000.
- **Biological neural network**s tolerate a great deal of ambiguity in data. However, artificial neural networks require somewhat precise, structured, and formatted data to tolerate ambiguity.
- Biological neural networks are fault-tolerant to a certain level, and the minor failures will not always result in memory loss.
- The brain can recover and heal to an extent. But the artificial neural networks are not designed for fault tolerance or self-regeneration.

Biological Neural Networks vs Artificial Neural Networks

- Talking about power consumption, the brain requires about 20% of all the human body's energy, equivalent to about 20 watts, which is exceptionally efficient. But computers need an enormous amount of computational power to solve the same problem, and they also generate a lot of heat during computation.

- Many attempts have been made to understand the complex mechanism of **biological neural network**s. Yet, they still hold many secrets to unlock and inspire the future of artificial intelligence.

Steps to implement Perceptron Model

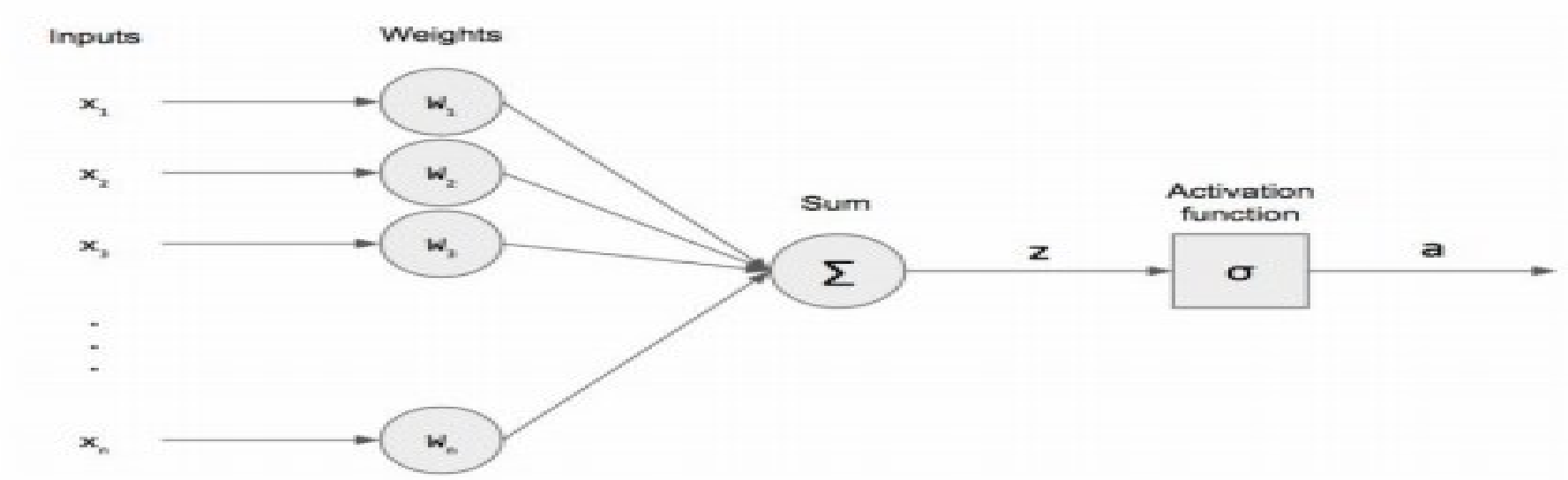
1. Apply some prediction function

1. Apply Activation function

1. Calculate cost or error

1. Update parameters to reduce the cost

Step 1 (Pre- activation)



In this model, we have n binary **inputs** (usually given as a vector) and exactly the same number of **weights** $W_1, \ldots, W_n$. We multiply these together and sum them up. We denote this as z and call it the **pre-activation**.

$$z = \sum_{i=1}^{n} W_i x_i = W^T x$$

Add Bias term

$$z = \sum_{i=1}^{n} W_i x_i + b = W^T x + b$$

$$z = \sum_{i=0}^{n} W_i x_i = W^T x$$

Step 2 (Apply activation Function)

After taking the weighted sum, we apply an activation function, σ, to this and produce an activation a. The activation function for perceptrons is sometimes called a **step function** because, if we were to plot it, it would look like a stair.

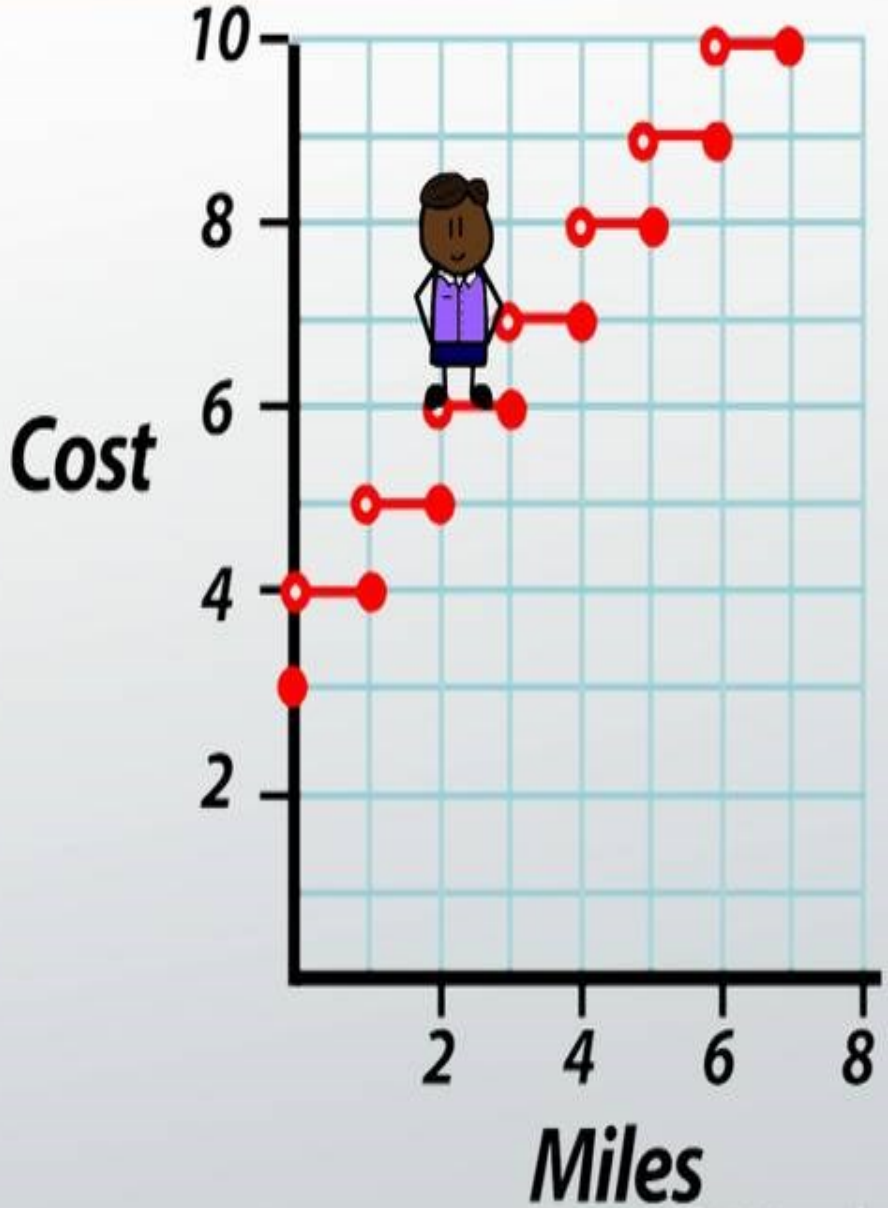
$$\sigma(q) = \begin{cases} 1 & q \geq 0 \\ 0 & q < 0 \end{cases}$$

In other words, if the input is greater than or equal to 0, then we produce an output of 1. Otherwise, we produce an output of 0.

$$a = \sigma(W^T x)$$

GRAPHING STEP FUNCTIONS

$$f(x) = \begin{cases} 4 \text{ if } 0 < x \leq 1 \\ 5 \text{ if } 1 < x \leq 2 \\ 6 \text{ if } 2 < x \leq 3 \\ 7 \text{ if } 3 < x \leq 4 \\ \vdots \end{cases}$$



Step 3(Calculate Error)

- e= predicted value – actual output

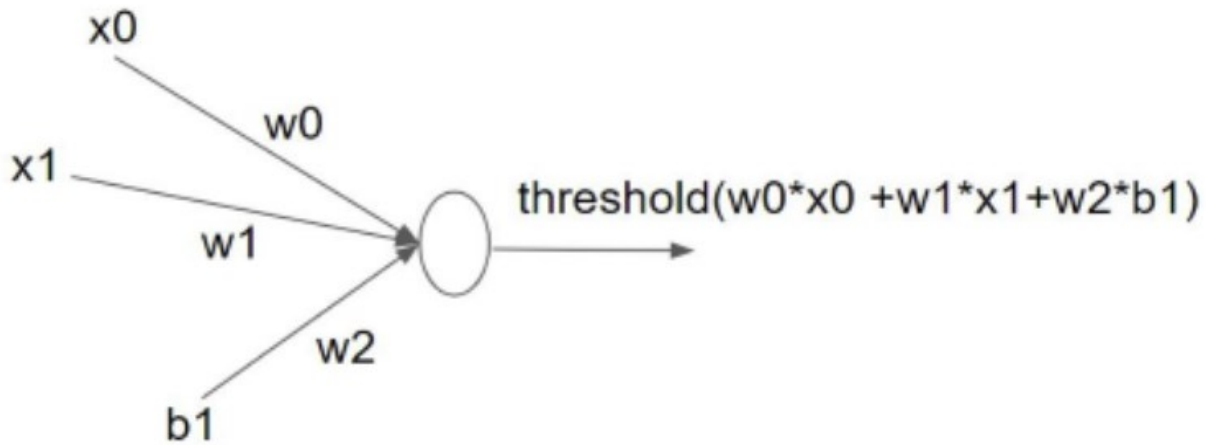
Step 4 (Update parameters to reduce error)

- $w_i = w_i + \Delta w_i$
- $w_i = w_i + \alpha \cdot e \cdot x_i$

1) Consider the neural network below. Find the appropriate weights for w0, w1 and w2 to represent the AND function.

Threshold function={1, if output >0; 0 otherwise}.

x0 and x1 are the inputs and b1=1 is the bias.

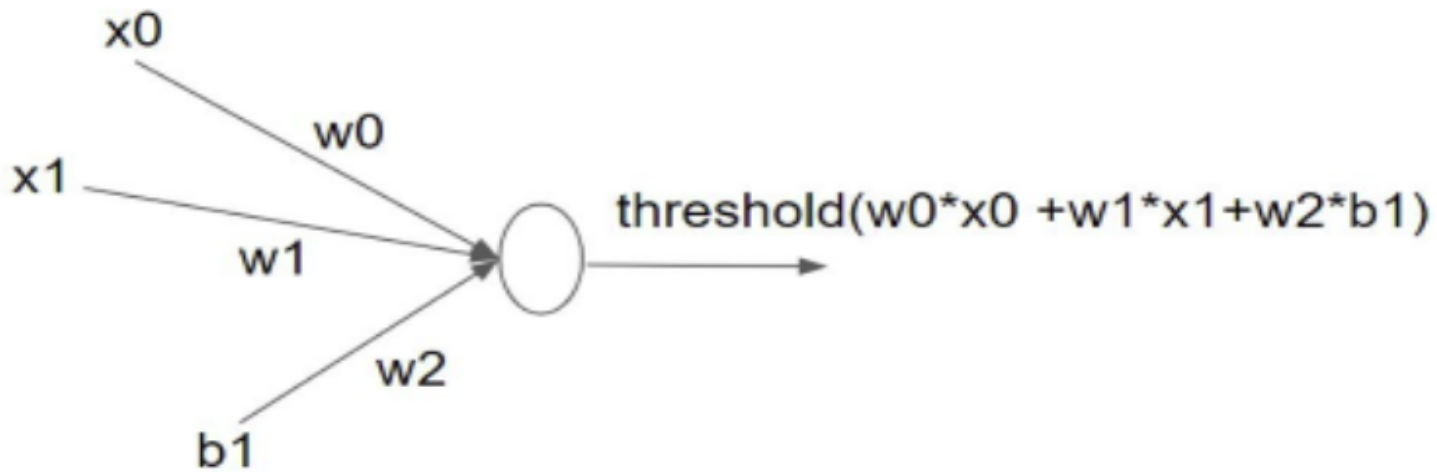


a. w0=1, w1=1, w2=1
b. w0=1, w1=1, w2=-1
c. w0=-1, w1=-1, w2=-1
d. w0=2, w1=-2, w3=-1

Consider the neural network below. Find the appropriate weights for w0, w1 and w2 to represent the AND function.

Threshold function={1, if output >0; 0 otherwise}.

x0 and x1 are the inputs and b1=1 is the bias.

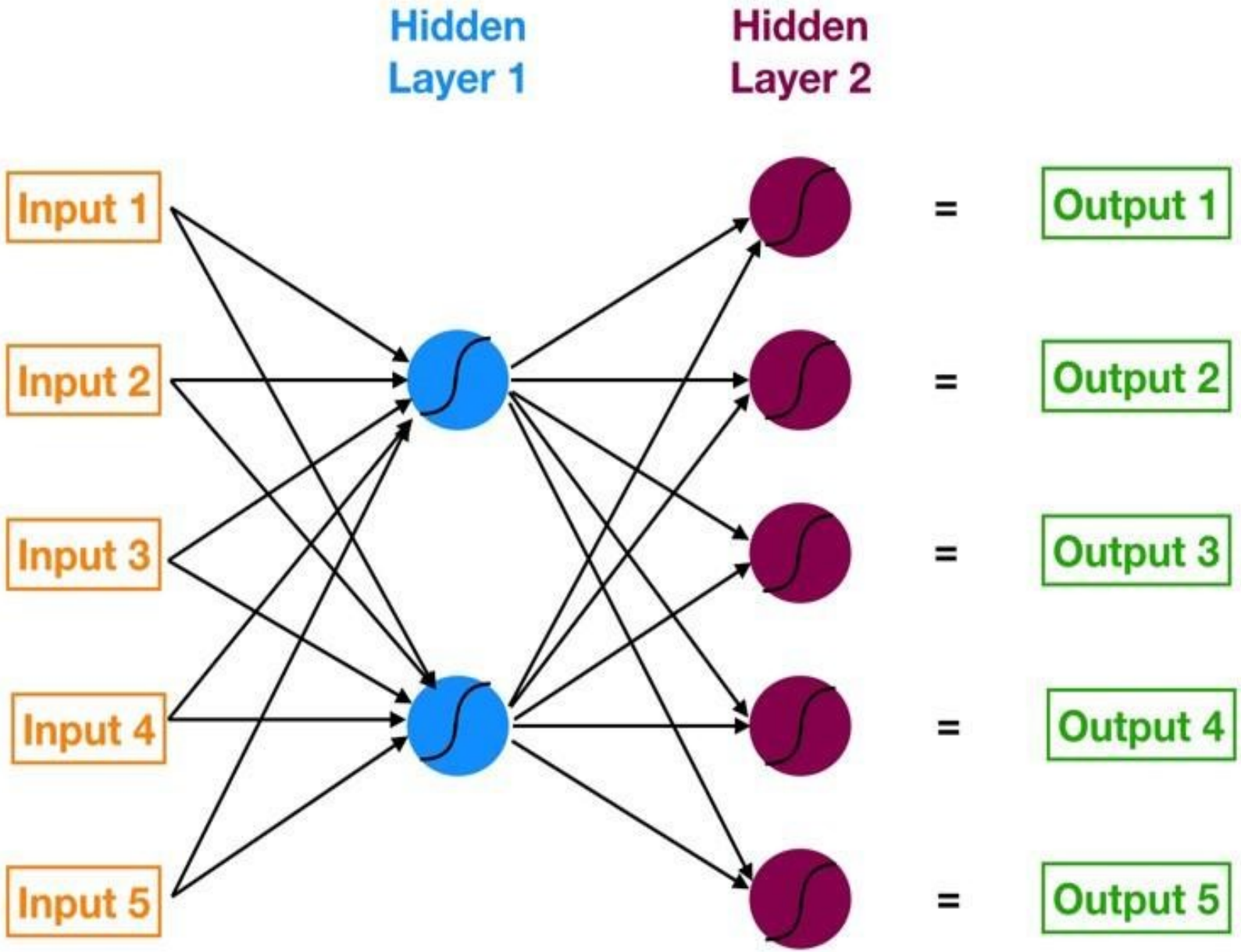


Which of the combination of weights make the network represent OR function:

a. w0=1,w1=1,w2=0
b. w0=1, w2=1, w3=1
c. w0=1, w1=1, w2=-1
d. w0=-1, w1=-1, w2=-1

Neural networks

- **Neural networks are multi-layer networks of neurons (the blue and magenta nodes in the chart below) that we use to classify things, make predictions, etc**.
- Below is the diagram of a simple neural network with five inputs, 5 outputs, and two hidden layers of neurons.


Hidden Layer 1
Hidden Layer 2
Input 1
Input 2
Input 3
Input 4
Input 5
=
Output 1
=
Output 2
=
Output 3
=
Output 4
=
Output 5

Forget for a second the more complicated looking picture of the neural network I drew above and focus on this simpler one below.

Logistic regression (with only one feature) implemented via a neural network

$$Sigmoid\,(B_1 * X + B_0) = Predicted\ Probability$$

Logistic regression equation

Let’s Add a Bit of Complexity Now

- Now that we have our basic framework, let’s go back to our slightly more complicated neural network and see how it goes from input to output. Here it is again for reference:

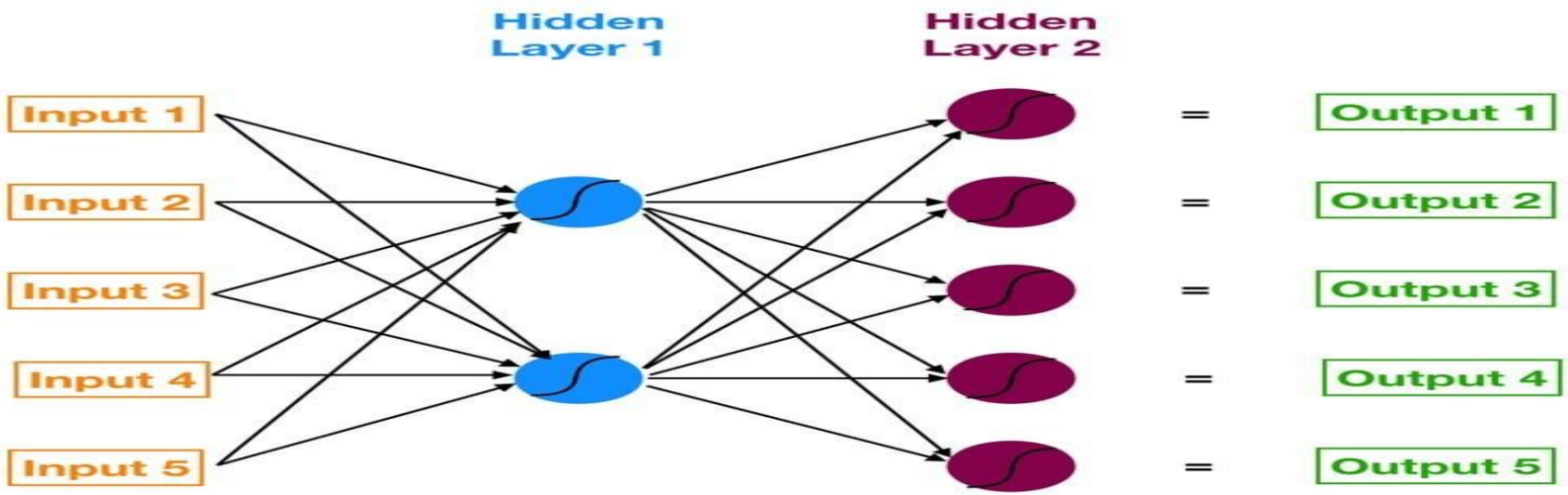
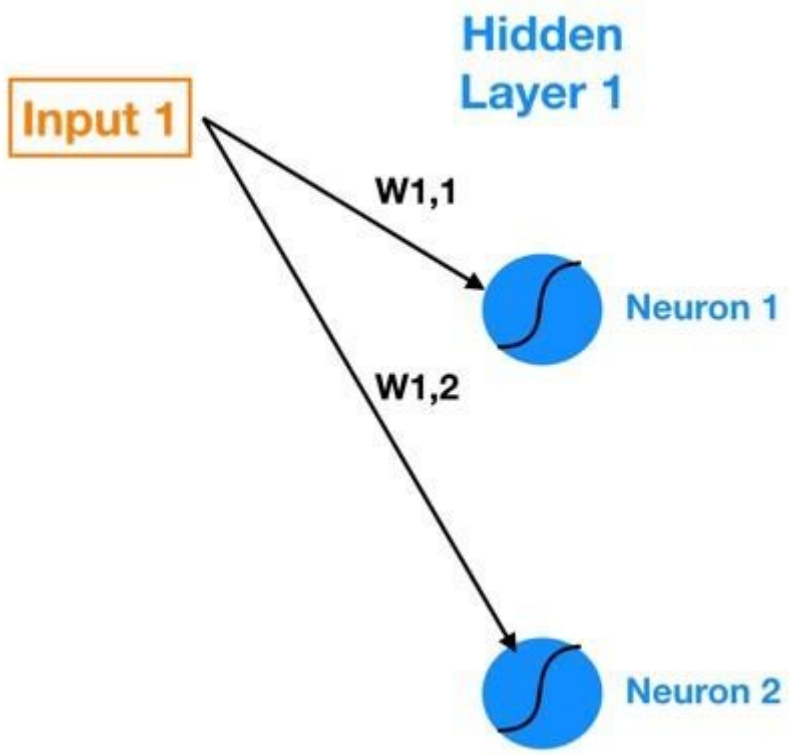
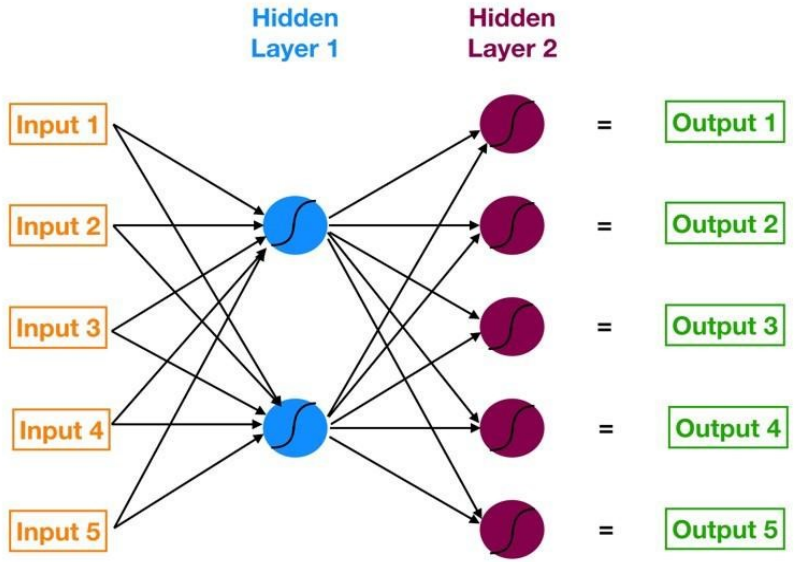


image (below) shows just the connections between Input 1 and Hidden Layer 1.



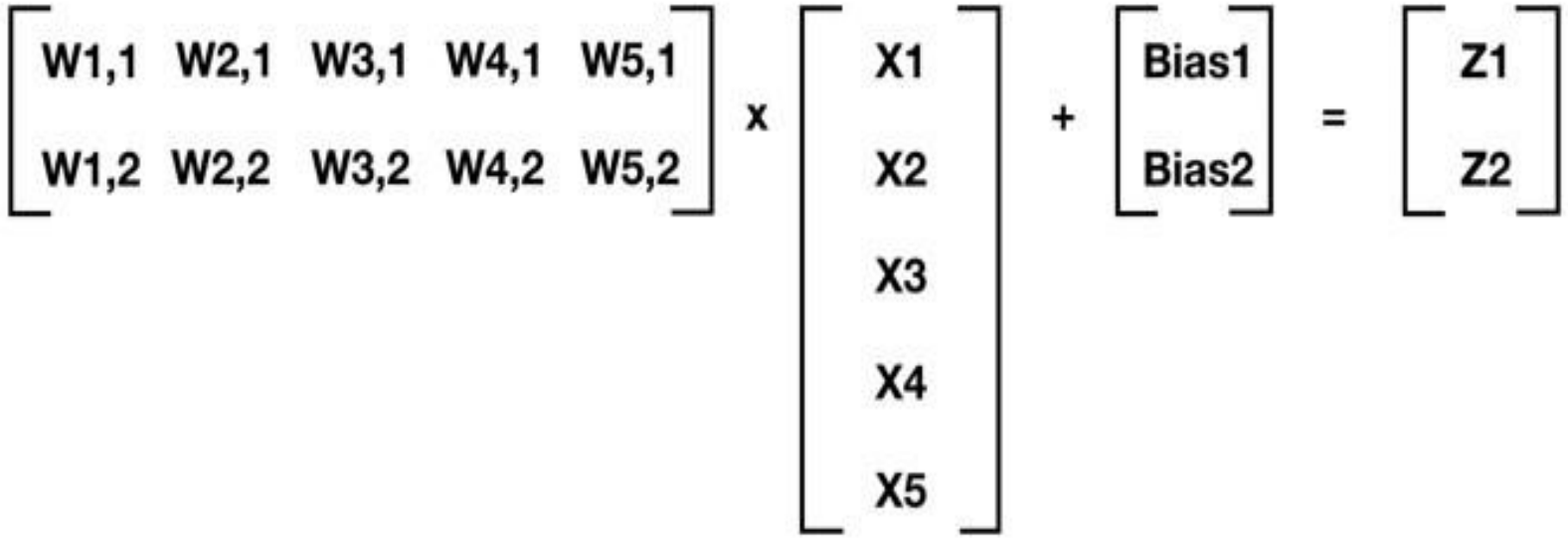
W1,1 denotes the weight that lives in the connection between Input 1 and Neuron 1 and
W1,2 denotes the weight in the connection between Input 1 and Neuron 2.

So the general notation that I will follow is W**a,b** denotes the weight on the connection between Input **a** (or Neuron **a**) and Neuron **b**.



- Now let's calculate the outputs of each neuron in Hidden Layer 1 (known as the activations). We do so using the following formulas (**W** denotes weight, **In** denotes input).
- *Z1 = W1*In1 + W2*In2 + W3*In3 + W4*In4 + W5*In5 + Bias_Neuron1*
- *Neuron 1 Activation = Sigmoid(Z1)*

We can use matrix math to summarize this calculation (remember our notation rules

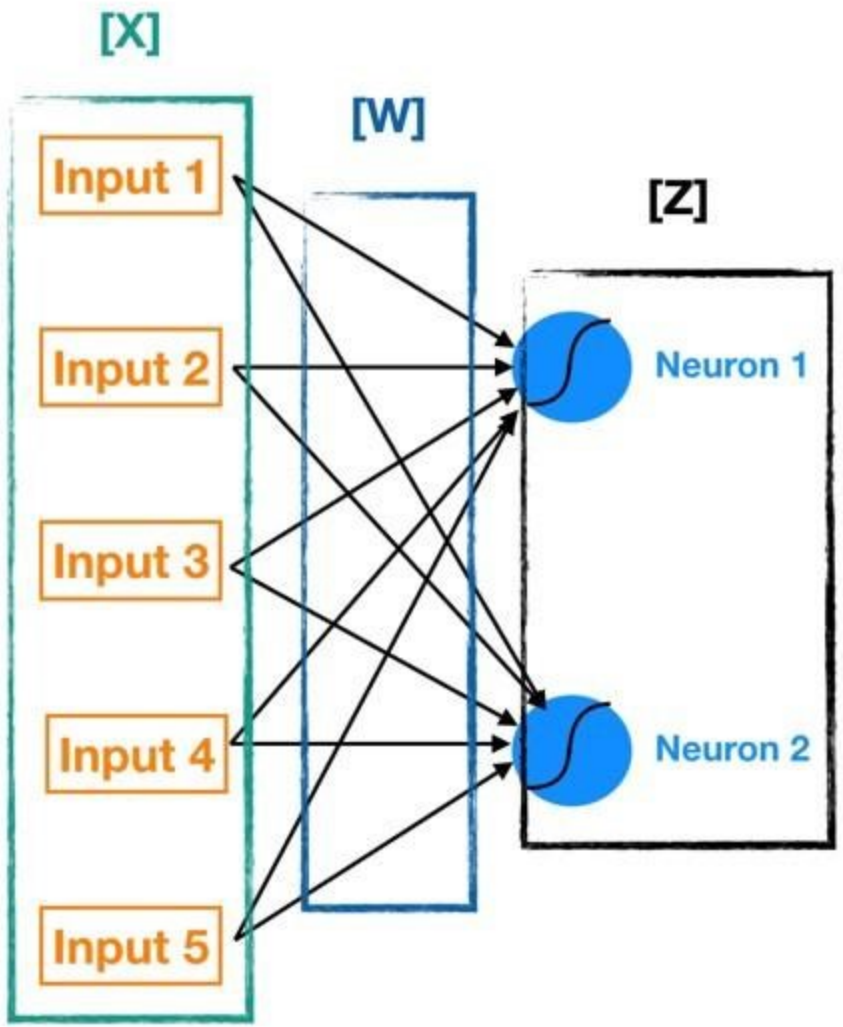


this generalizes to:

[W] @ [X] + [Bias] = [Z]

Once we have [Z], we can apply the activation function (sigmoid in our case) to each element of [Z] and that gives us our neuron outputs (activations) for the current layer.

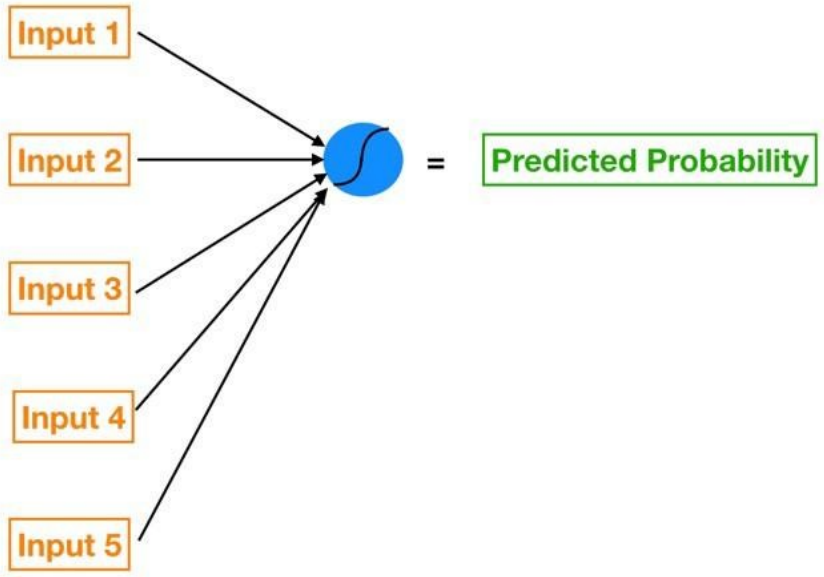
Visualizing [W], [X], and [Z]



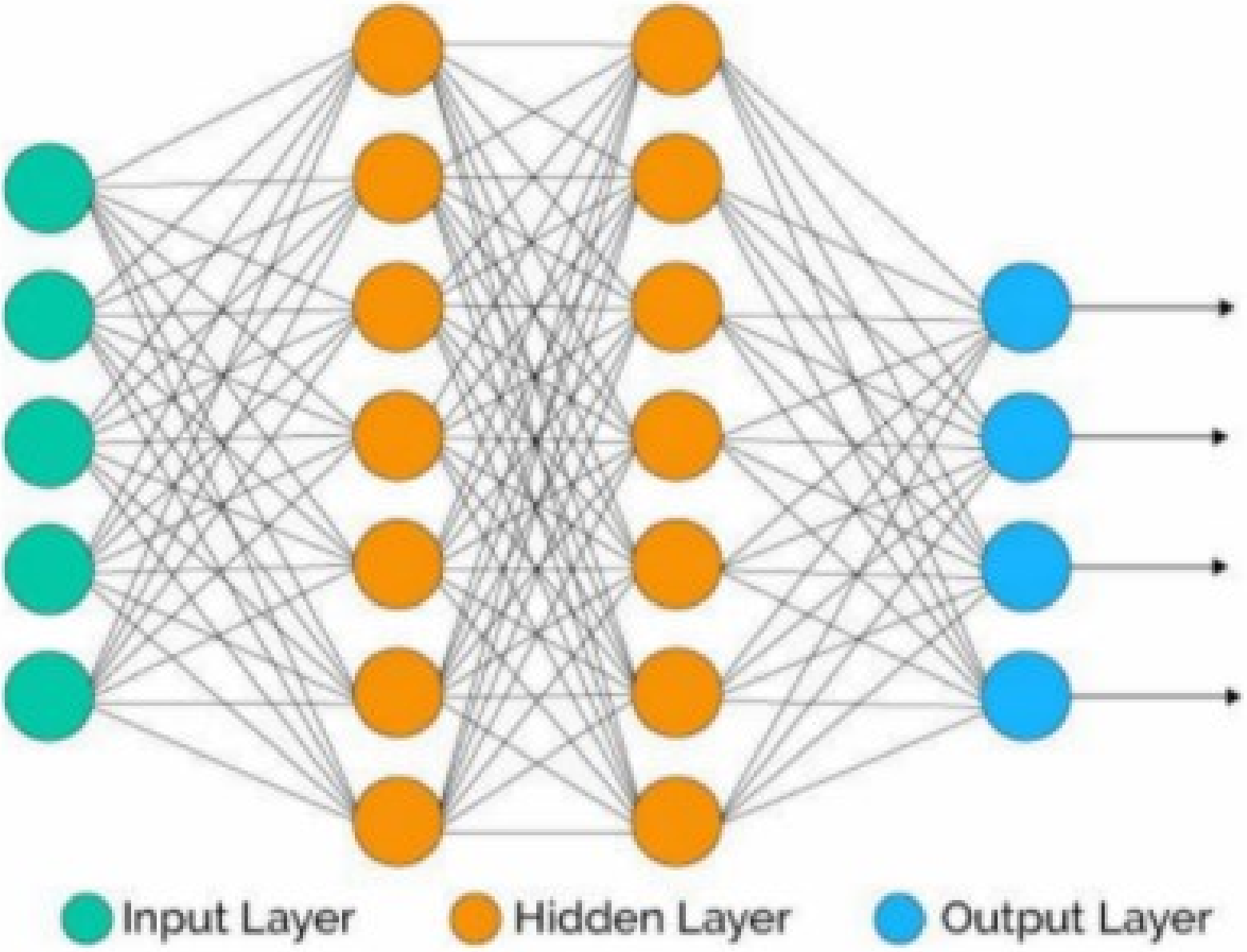
How Neural Network Learn?

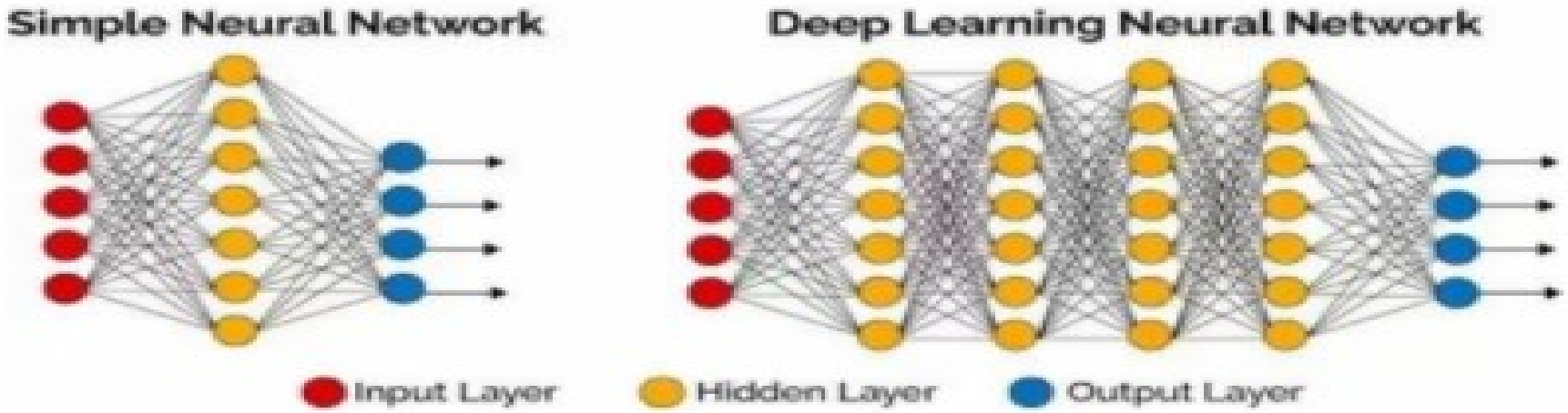
- *The training process of a neural network, at a high level, is like that of many other data science models* — ***define a cost function and use gradient descent optimization to minimize it****.*
- In traditional regression, we can change any particular beta in isolation without impacting the other beta coefficients.
- So by applying small isolated shocks to each beta coefficient and measuring its impact on the cost function, it is relatively straightforward to figure out in which direction we need to move to reduce and eventually minimize the cost function.

- **In a neural network, changing the weight of any one connection (or the bias of a neuron) has a reverberating effect across all the other neurons and their activations in the subsequent layers**.
- **That's because each neuron in a neural network is like its own little model**.
- For example, if we wanted a five feature logistic regression, we could express it through a neural network, like the one on the left, using just a singular neuron!



- *So each hidden layer of a neural network is basically **a stack of models** (each individual neuron in the layer acts like its own model) **whose outputs feed into even more models further downstream** (each successive hidden layer of the neural network holds yet more neurons).*


Input Layer
Hidden Layer
Output Layer



Perceptrons: The First Neural Networks