



# Supervised learning- Artificial neural network



Parcours Progis

Etudes, Medias, communication, Marketing

Bahareh Afshinpour

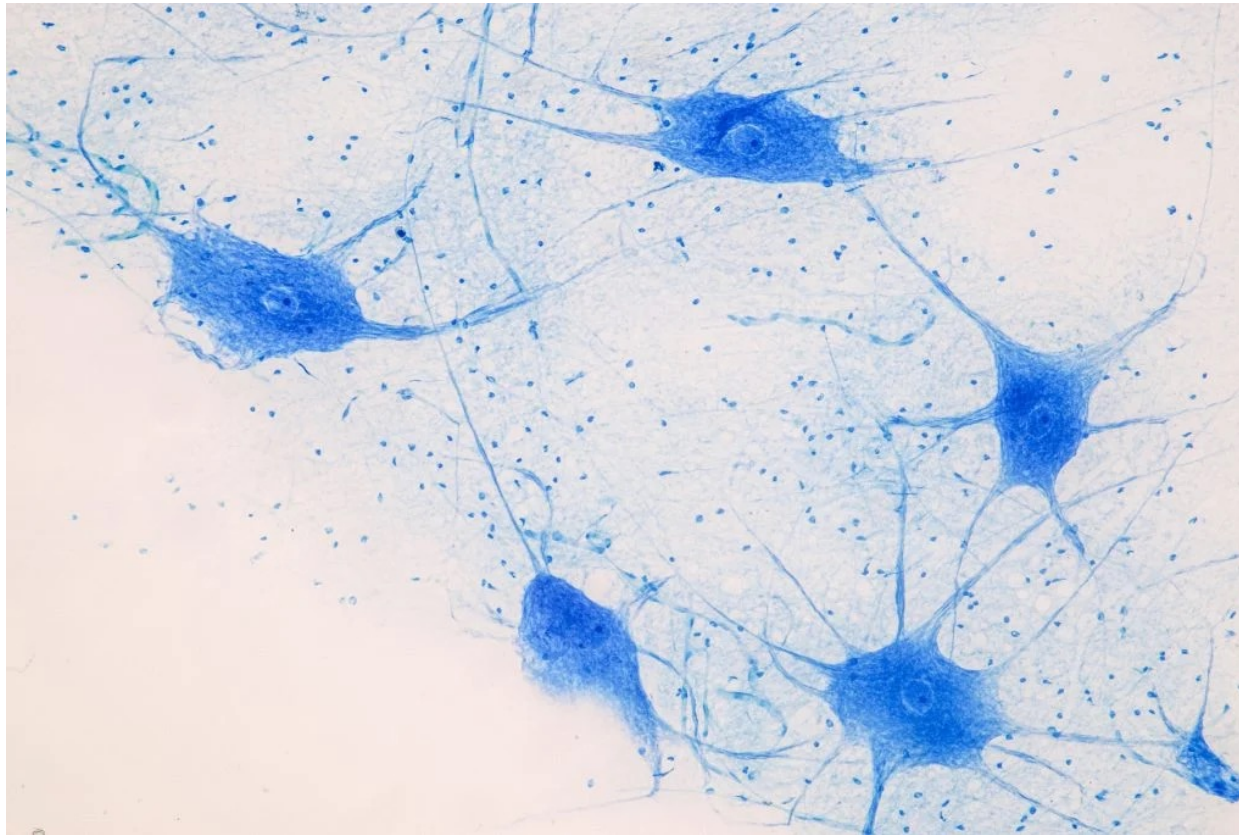
03.03.2025

## Suggested Reading

Perceptrons and MLPs and Back-Propagation training algorithm

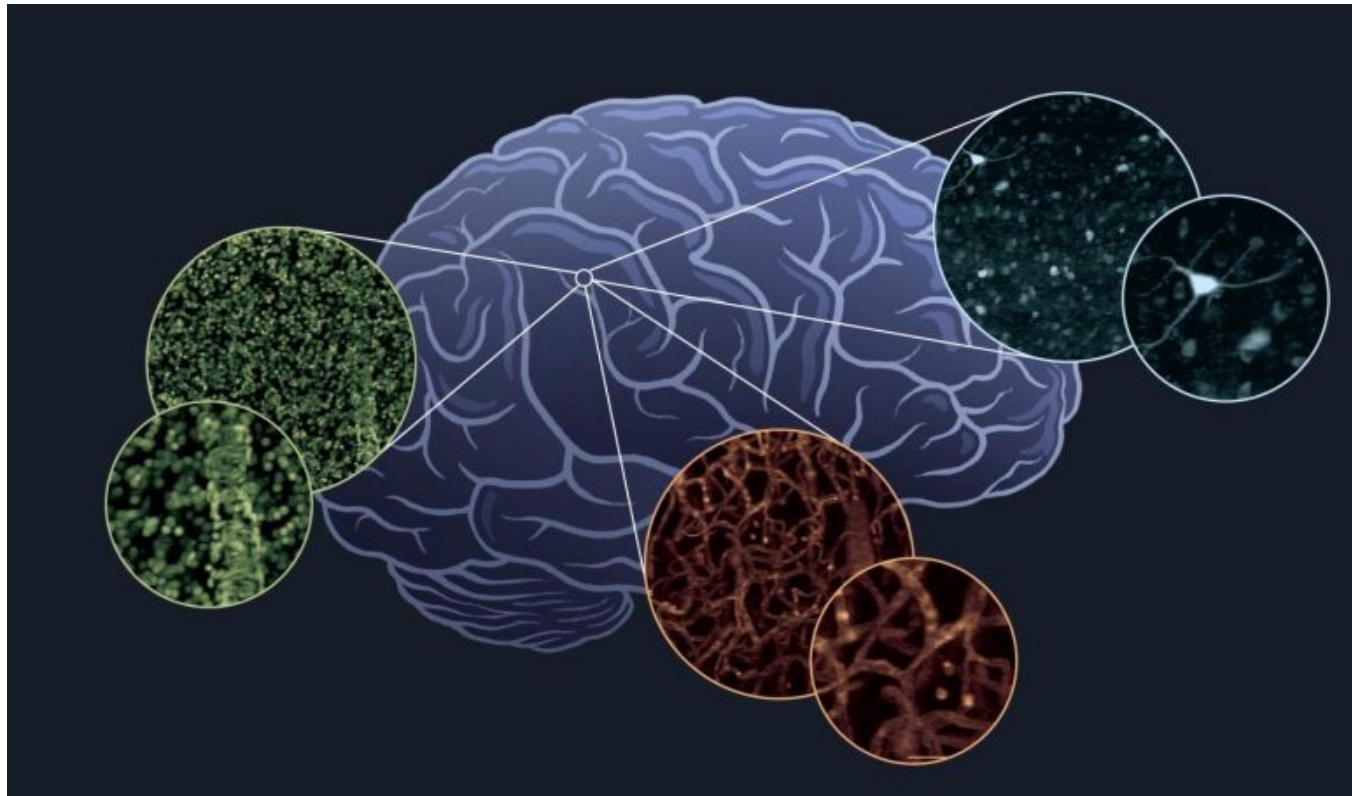
- Haykin: Sections 3.5 - 3.9 , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6
- Bishop(NNs for PR): Sections 4.1, 4.8
- Hwang, Yoon. 2019. "Conducting Customer Analytics with Python." In Hands-On Data Science for Marketing. Chapter 11, 580-581
- [https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1\\_67000Dx\\_ZCJB-3pi&index=1](https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=1)
- <https://xnought.github.io/backprop-explainer/>

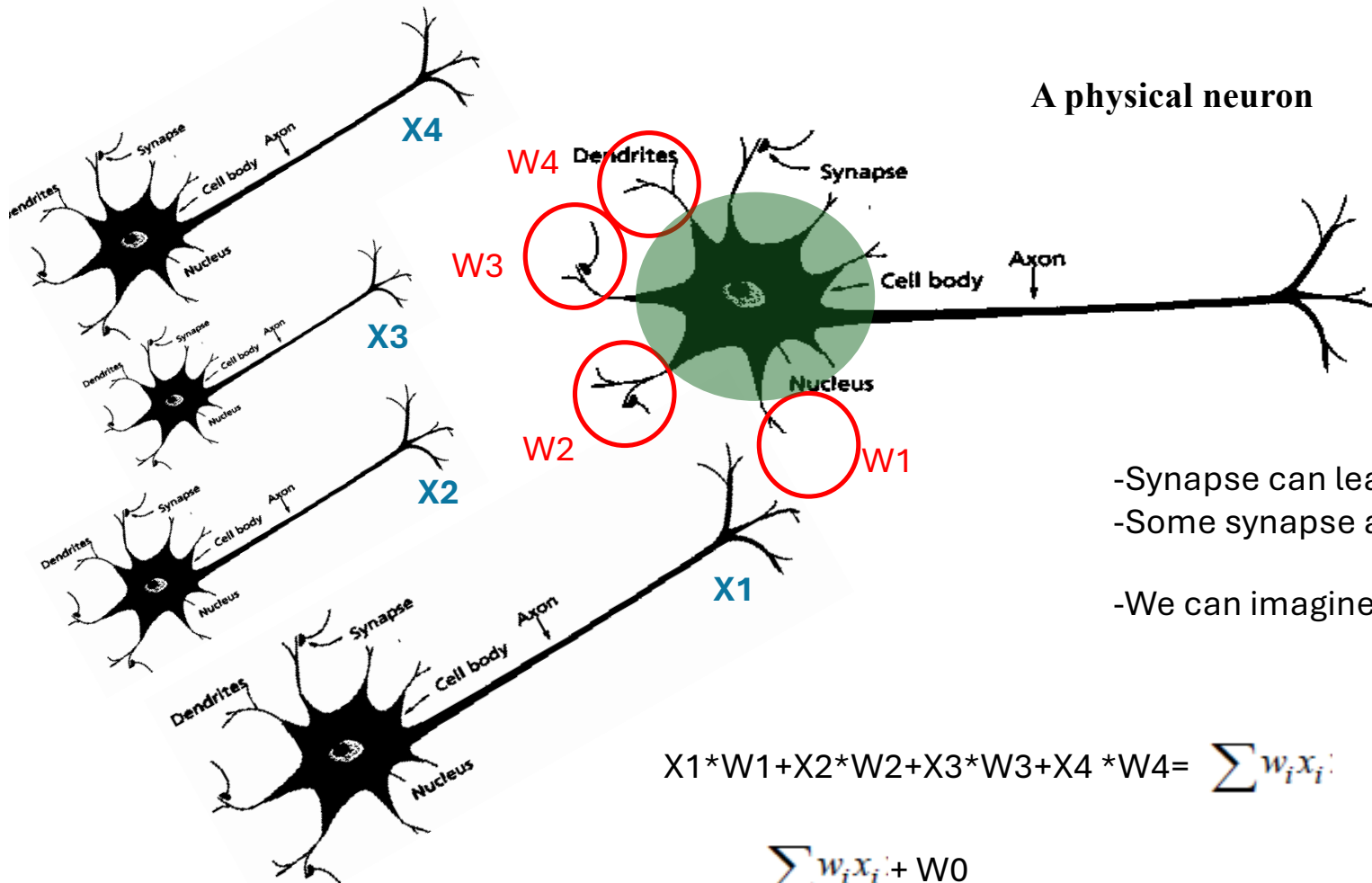
# Neuron



# Neuron

Artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.





-Synapse can learn during the time

-Some synapse are weak

-We can imagine that we have **W<sub>i</sub>** for each synapse

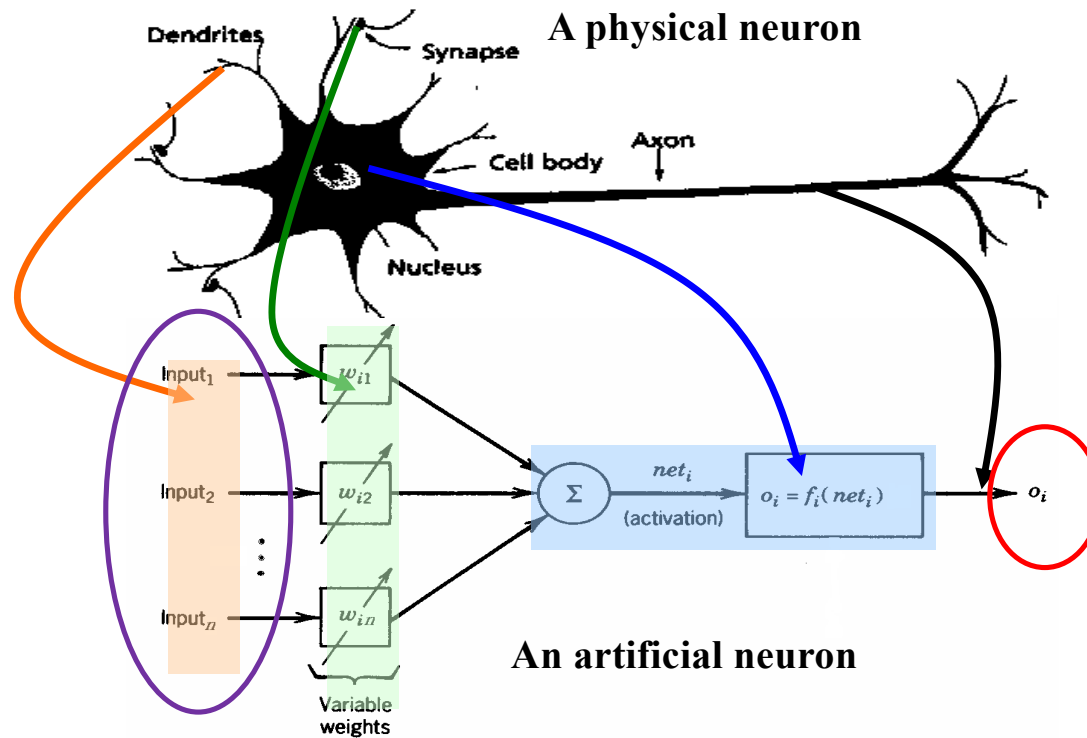
$$X1*W1+X2*W2+X3*W3+X4*W4 = \sum w_i x_i$$

$$\sum w_i x_i + W0$$

# Neuron and Perceptron

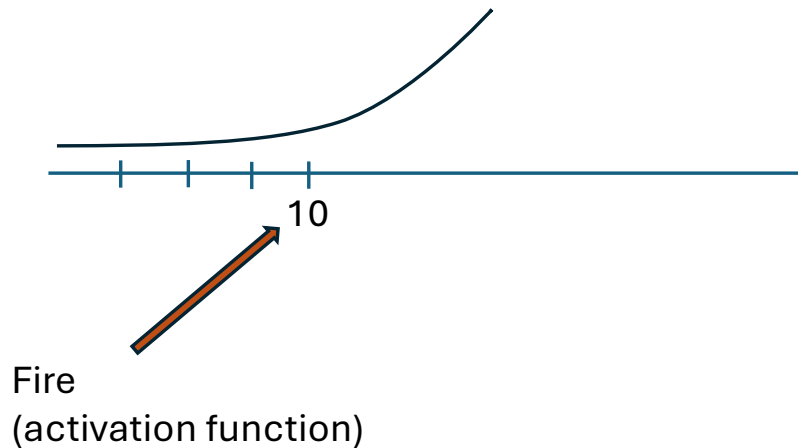
- Inspiration from the brain
- we're going to model the brain
- Cognitive scientists and neuroscientists whose aim is to understand the functioning of the brain
- Models of the natural neural networks in the brain and make simulation studies.

How to recreate  
a neuron?



# Activation

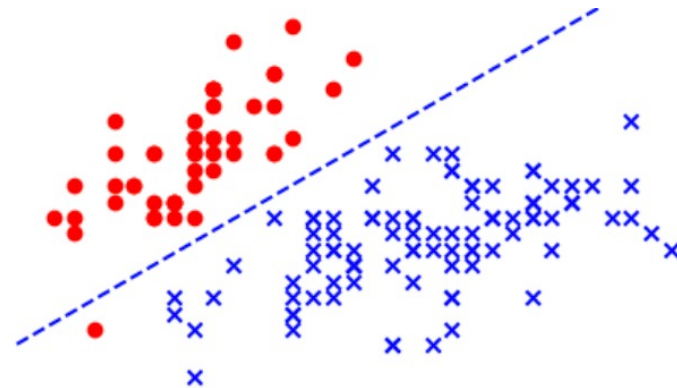
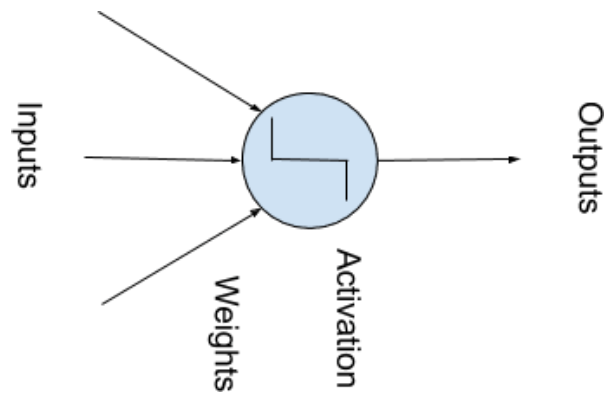
- For one specific moment, neuron starts to fire. (It start to create an electric pulse in real life.)
- If sum of the signals that go to the center, is less than for example 10, the neuron is off. After 10, it start to fire and have an output.





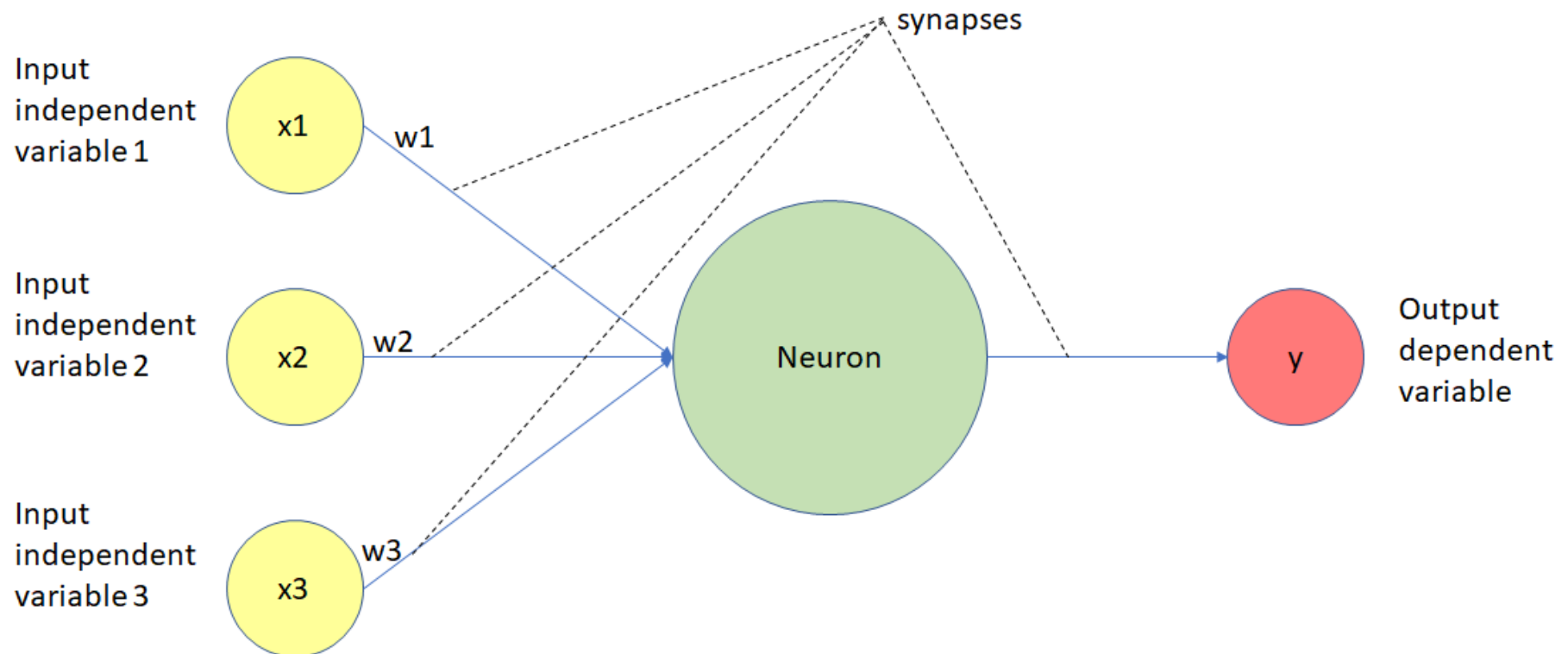
# Perceptron

- The building blocks for neural networks are artificial neurons.
- These are simple computational units that have weighted input signals and produce an output signal using an activation function.
- A simple perceptron is used for the linear classification.





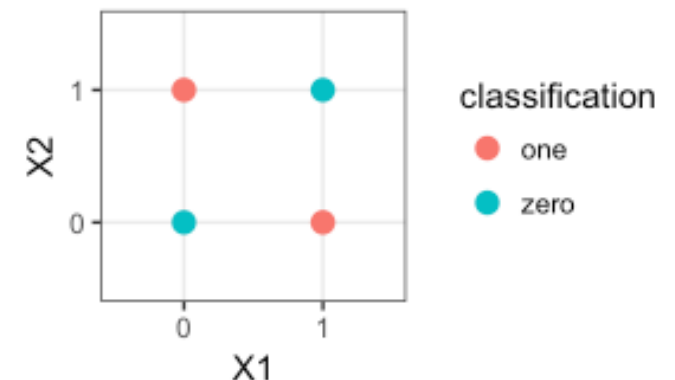
# Perceptron



# Historical Note and problem

There was great interest in Perceptrons in the '50s and '60s

- EXOR problem  
regions must be linearly separable.  
you **can not** draw **a straight line** to separate the red point from the others.



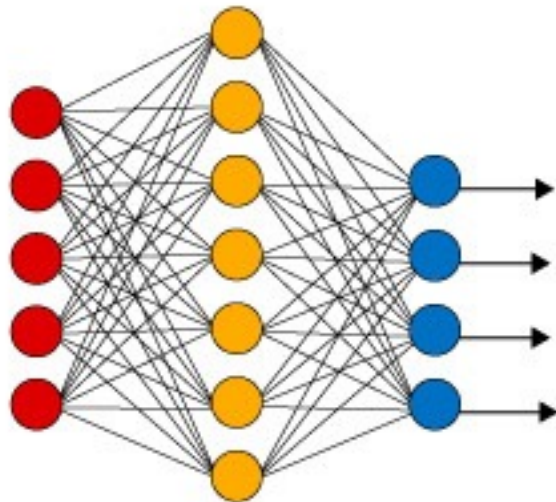
## Historical Note

- In 2006, Geoff Hinton published a series of articles showing how they could efficiently train a neural network with **multiple layers**.

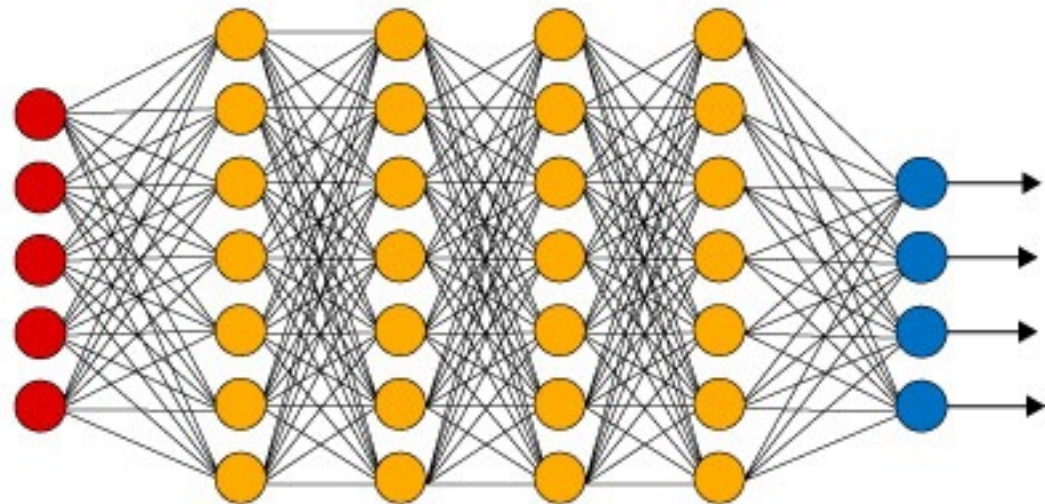


# Multi Layer Neural Network (MLP)

## Simple Neural Network



## Deep Learning Neural Network



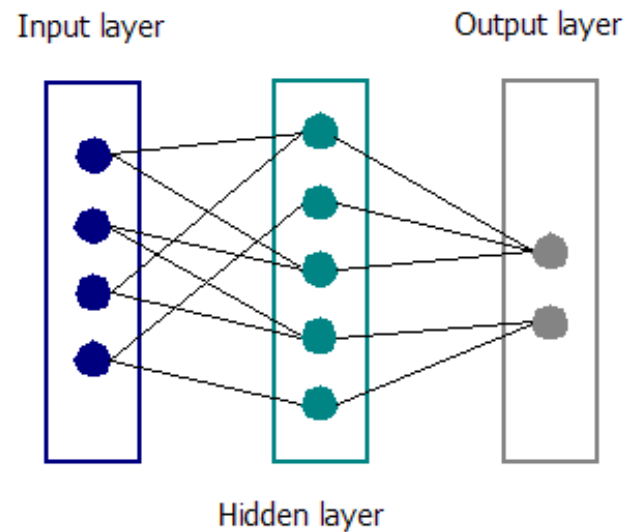
● Input Layer

● Hidden Layer

● Output Layer

<https://thedata scientist.com/what-deep-learning-is-and-isnt/>

# Hidden layers



Here, the graphic shows a tri-layered neural network with a four-dimensional input layer and a two-dimensional output layer. The hidden layer is five-dimensional.

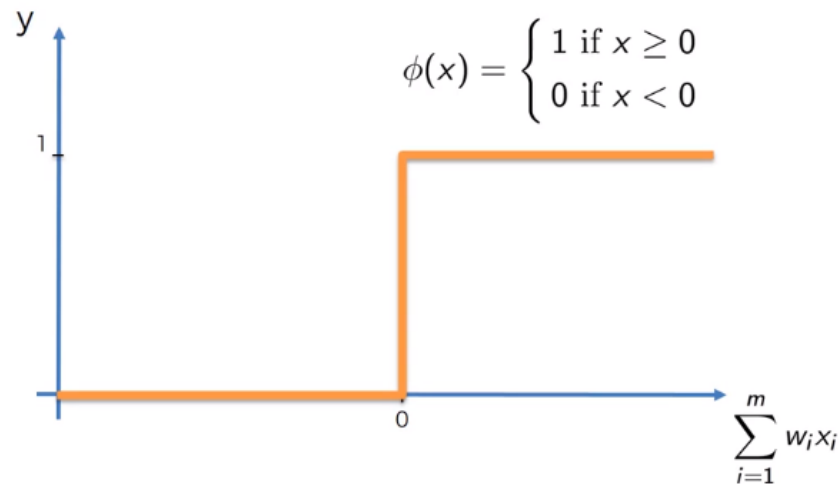
- ANNs has evolved into a broad family of techniques that have advanced the state of the art across multiple domains.
- The simplest types have one or more static components, including number of units, number of layers, unit weights, and topology.
- **MLP model:**
  - as a neural network model that has at least **one or more** hidden layers of nodes, including one layer for the input and another layer for the output.
- **Deep learning:**
  - This is a type of ANN model where the number of layers between the input and the output layers **is large**.

# Activation Function

- An activation function is an internal state of a neuron that converts an input signal to an output signal.
- There are more different types of activation functions.

## 1- Threshold function:

- if the value is less than zero then the function is zero, but if the value is more than zero the function is one. It is basically a yes/no type of function.



<https://www.andreaperlato.com/aipost/the-activation-function/>

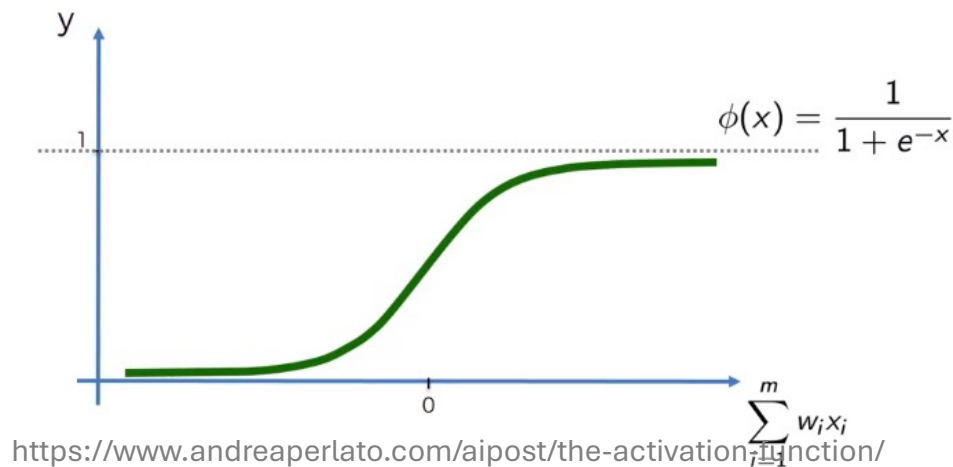
Without the activation function, the output values from the neurons can range between (- infinity) to (+infinity).



# Activation Function

## 2- Sigmoid Function

- The value  $x$  in its formula is the value of the sum of the weighed.
- It is very similar to the logistic regression.
- It has a gradual progression and anything below zero is just zero, and above zero it approximates to one.
- In fact, outputs of sigmoid functions are interpretable as **probabilities**. (digit between 0 and 1)
- It is especially useful in the output layer, especially when we are trying to predict probabilities.

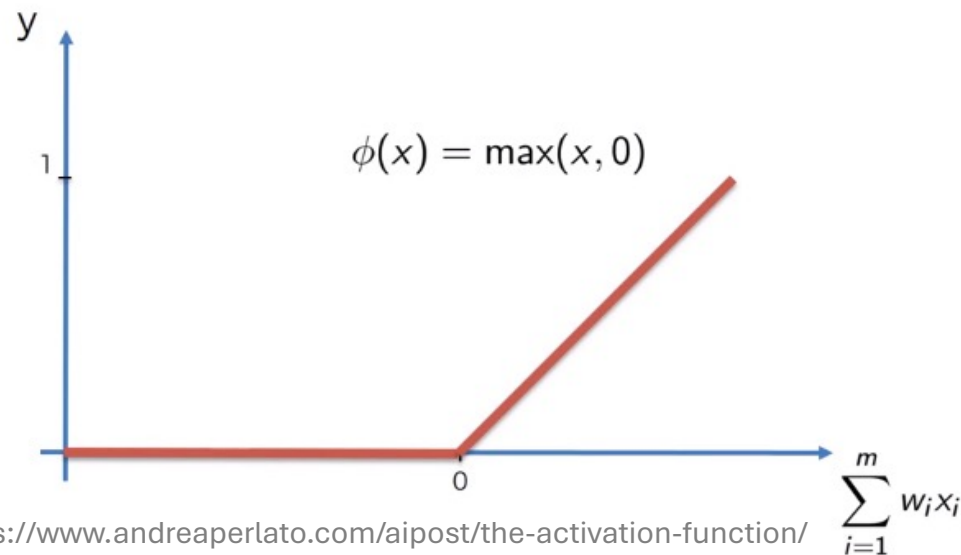


What is the probability that the input data, belongs to the for example the first class?

# Activation Function

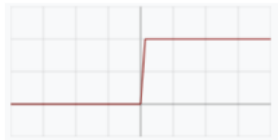

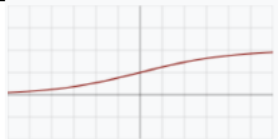
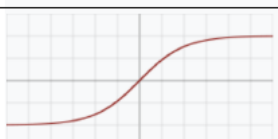

## • 3-Rectifier Function

- is one of the most popular functions for ANN, so when it goes all the way to zero it is zero, and from there it is **gradually progresses** as the input value increase as well.
- It is used in almost all the convolutional neural networks(CNN) or deep learning.

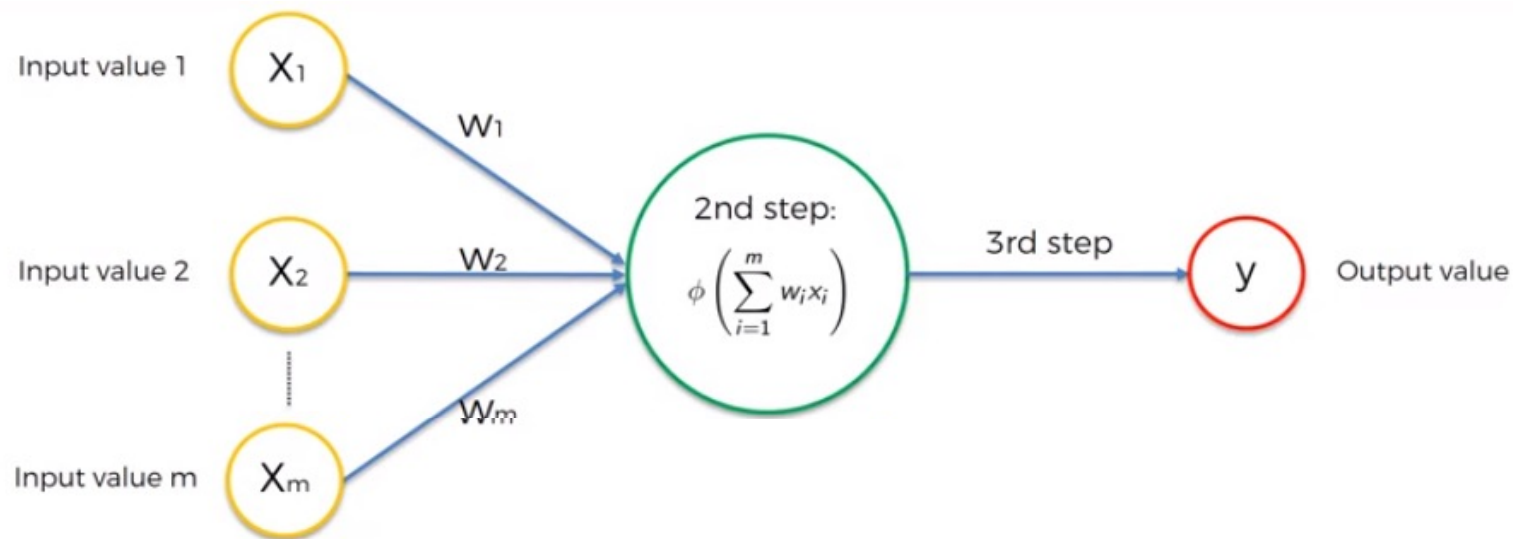


<https://www.andreaperlato.com/aipost/the-activation-function/>

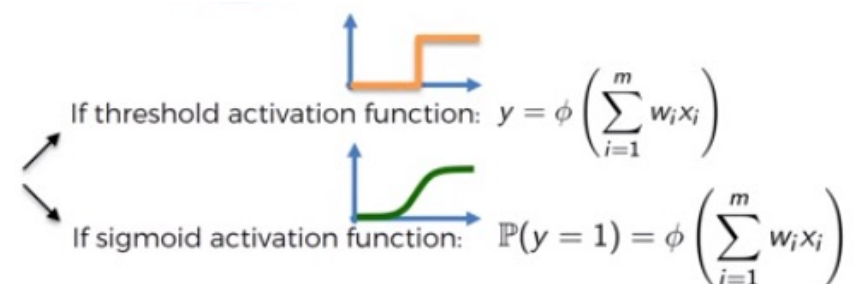
# Activation Function (in French)

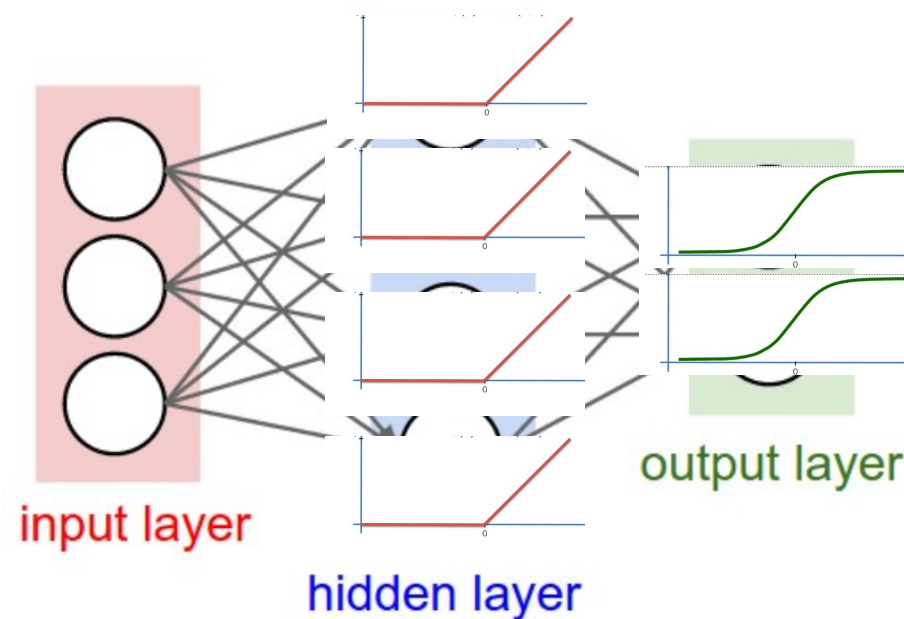
Nom	Définition	Graphe
Heaviside	$\bar{H}(z) = \begin{cases} 1, & \text{si } z \geq 0 \\ 0, & \text{sinon.} \end{cases}$	
Linéaire	$\bar{H}(z) = z$	
Sigmoïde	$\bar{H}(z) = \frac{1}{1+e^{-z}}$	
Tanh	$\bar{H}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	
Relu	$\bar{H}(z) = \begin{cases} z, & \text{si } z \geq 0 \\ 0, & \text{sinon.} \end{cases}$	

# Question



Assume we only need two labels at the end (0 and 1). Which activation function should we use?





What is commonly used is to apply the **rectifier** activation function for the hidden layer and then the signals are passed on to the output layer where the **sigmoid** activation function is used, and that will be the final output that predict the probability.

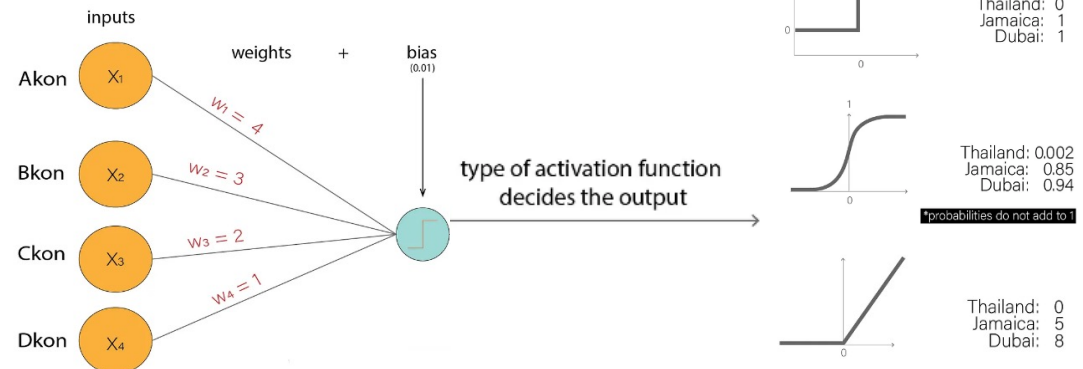
## Example to comprehend activation functions

- Four friends are choosing to take a vacation together.
- Their list of possibilities has been reduced to three places. Jamaica, Dubai, and Thailand.
- They individually choose to vote for the places on a scale of -10 to 10 in order to choose the final destination.

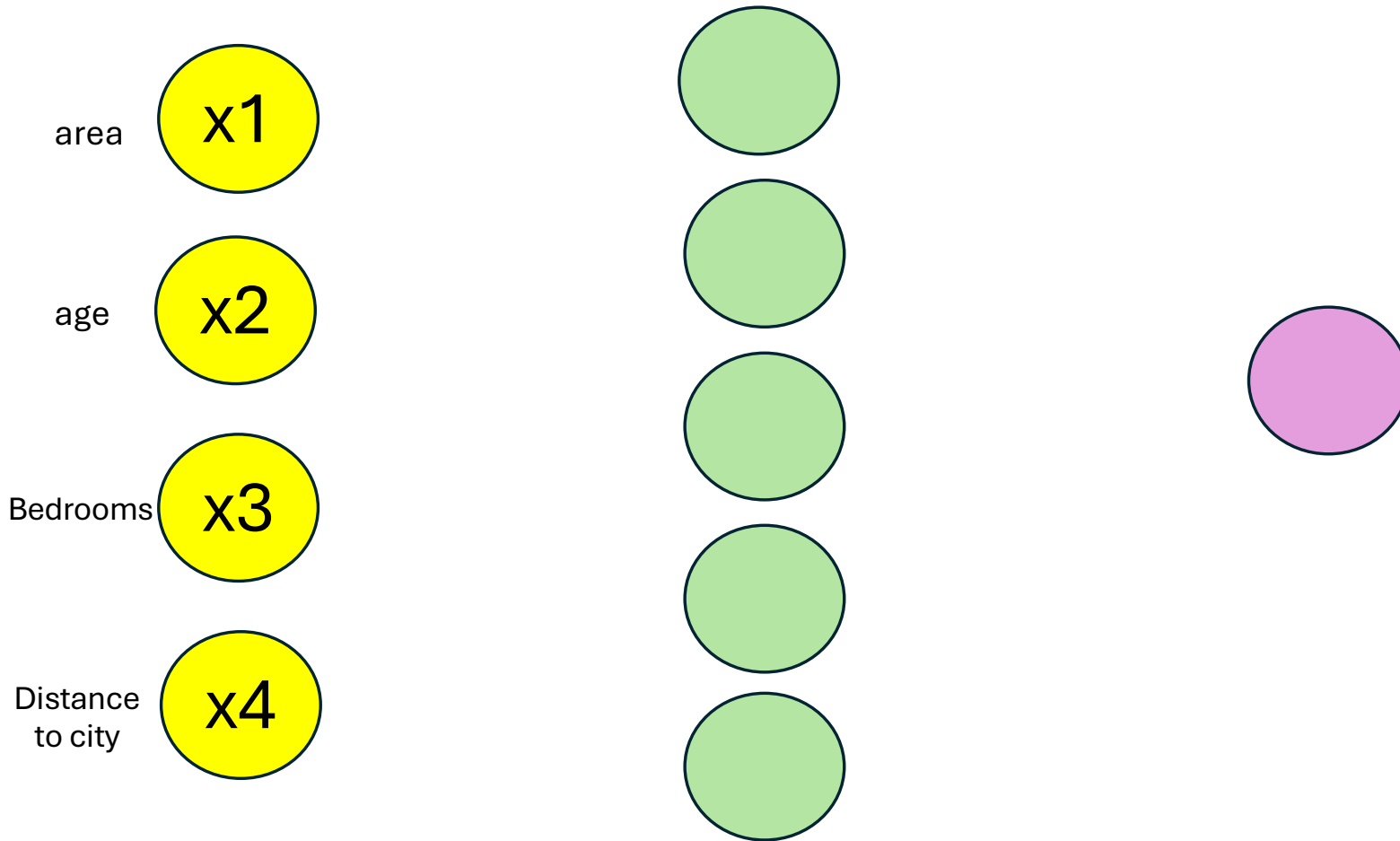
	Akon	Bkon	Ckon	Dkon
Thailand	-9	10	3	-2
Jamaica	-4	5	-1	8
Dubai	-3	6	-2	6

$Y = W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + C \text{ (bias)}$		
Thailand	$(4 \times -9) + (3 \times 10) + (2 \times 3) + (1 \times -2)$	= -2
Jamaica	$(4 \times -4) + (3 \times 5) + (2 \times -1) + (1 \times 8)$	= 5
Dubai	$(4 \times -3) + (3 \times 6) + (2 \times -2) + (1 \times 6)$	= 8

- assign initial random weights to each variable.
- The neuron calculates the weighted sum of its inputs and provides and output value.

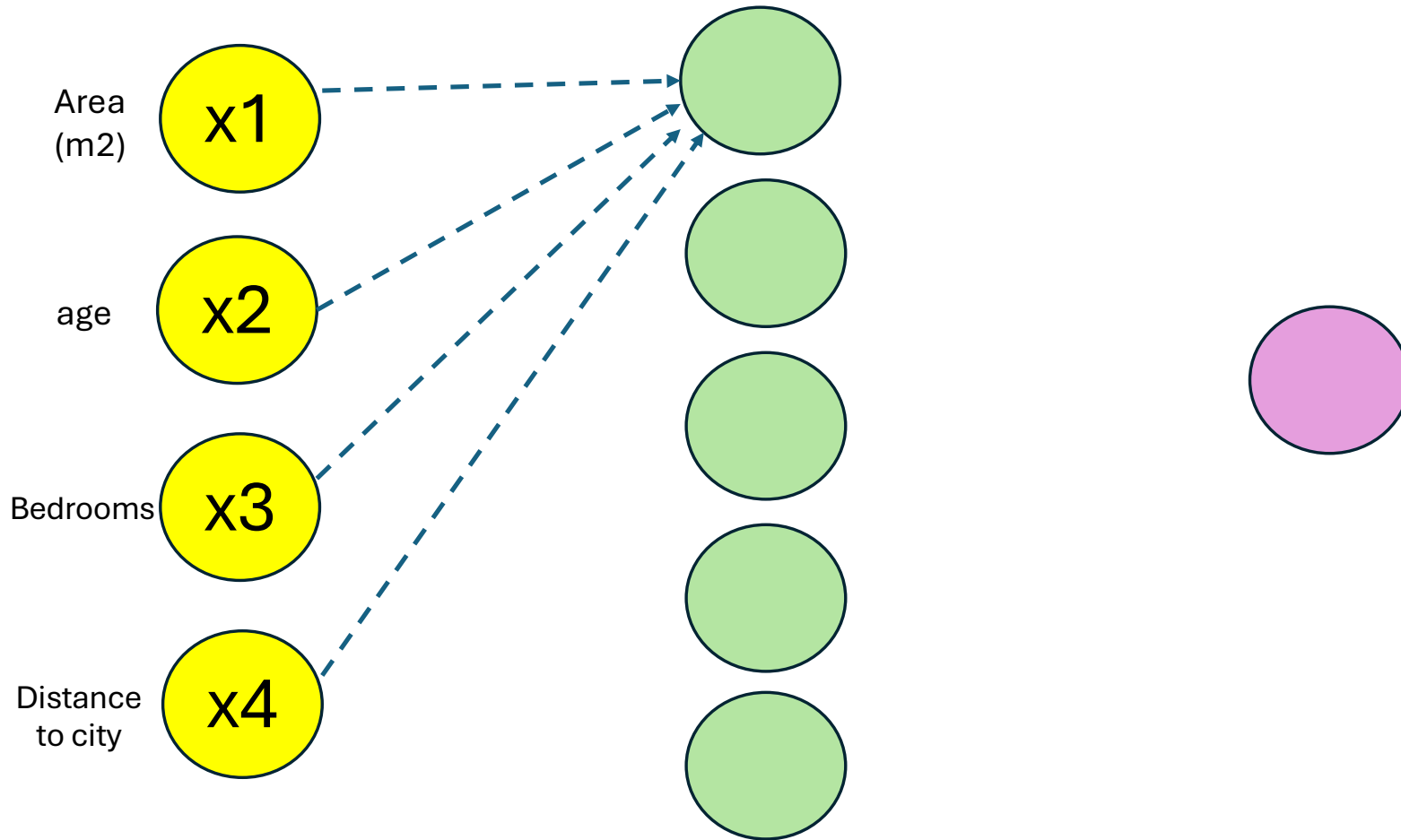


# How do NNs work?

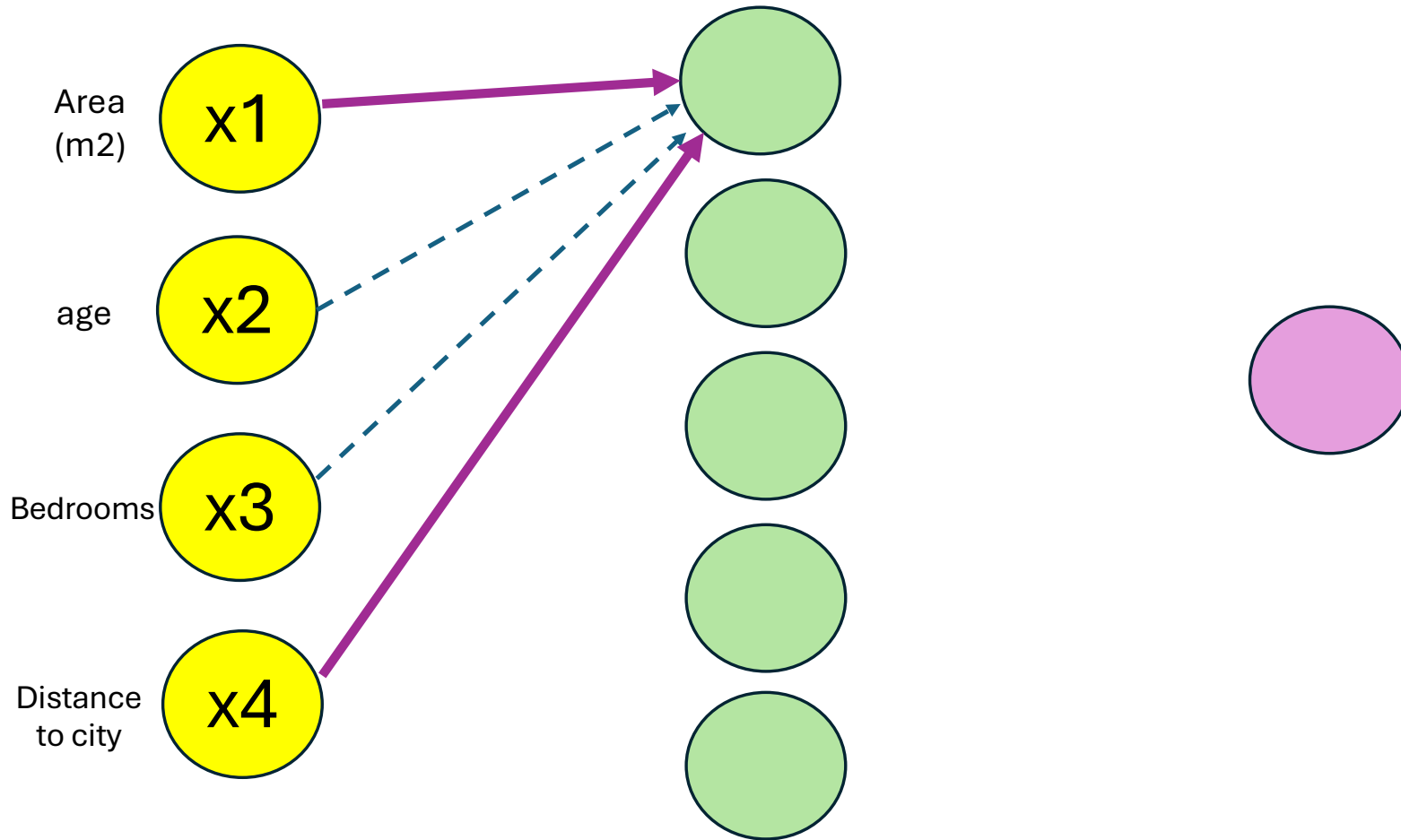




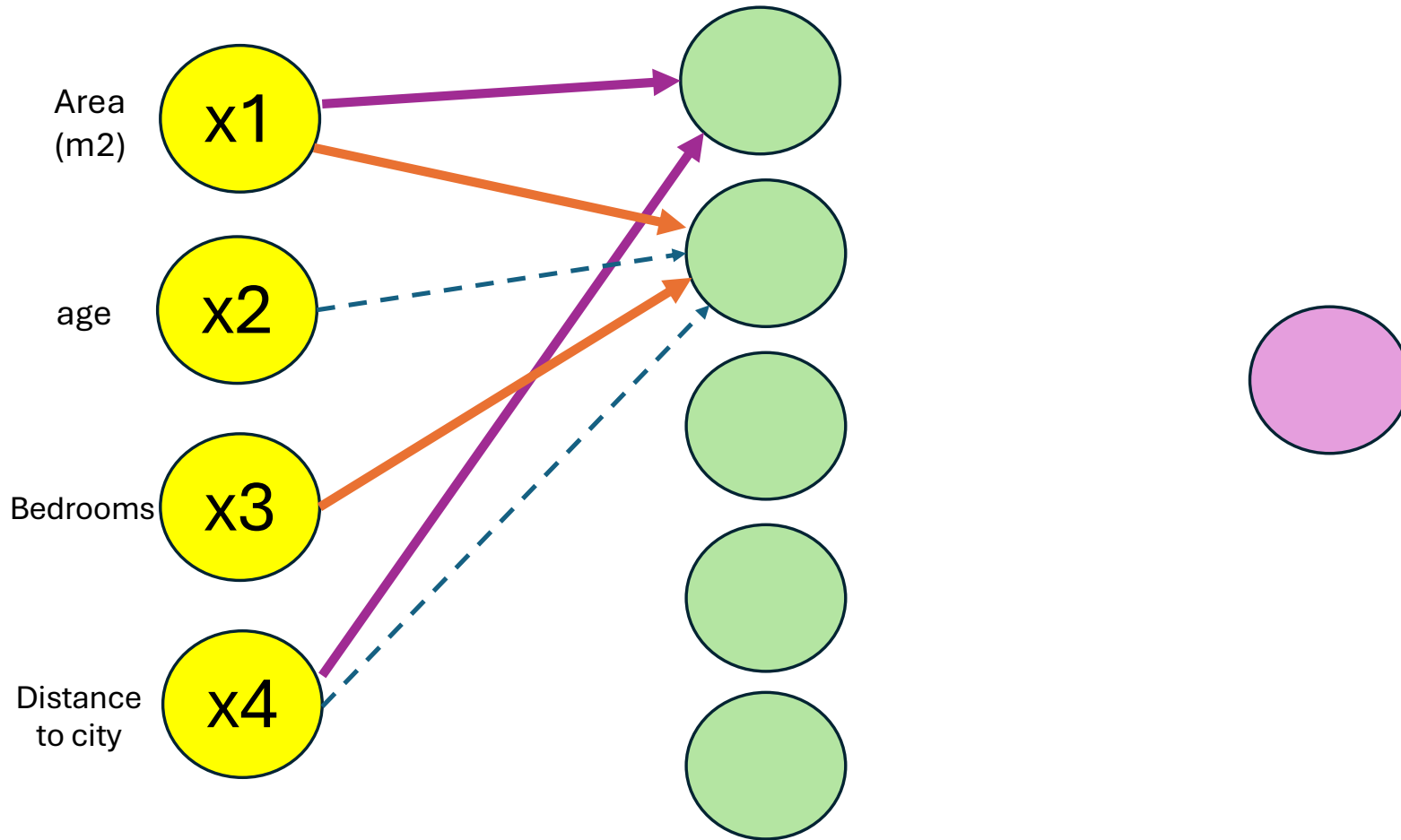
# How do NNs work?



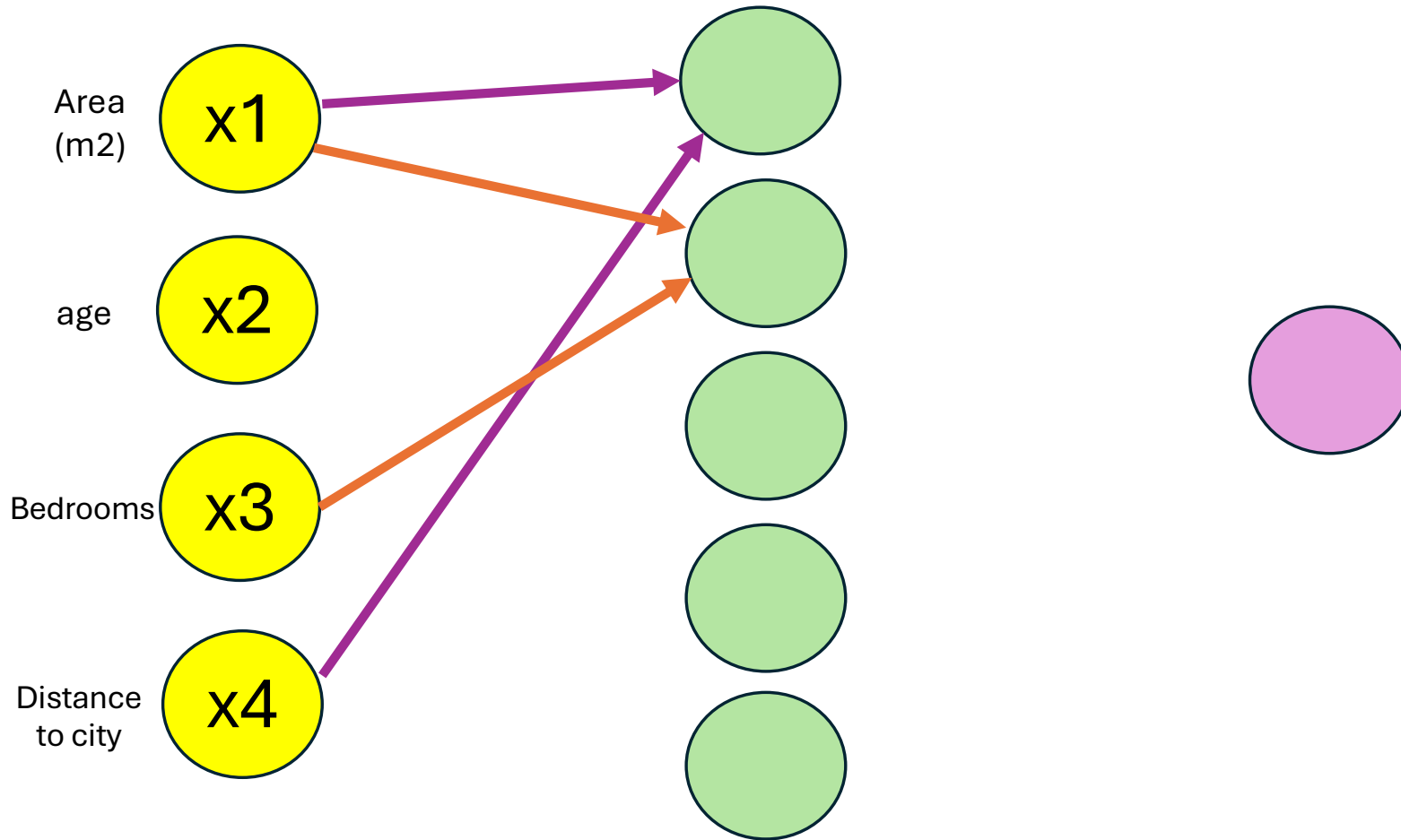
# How do NNs work?



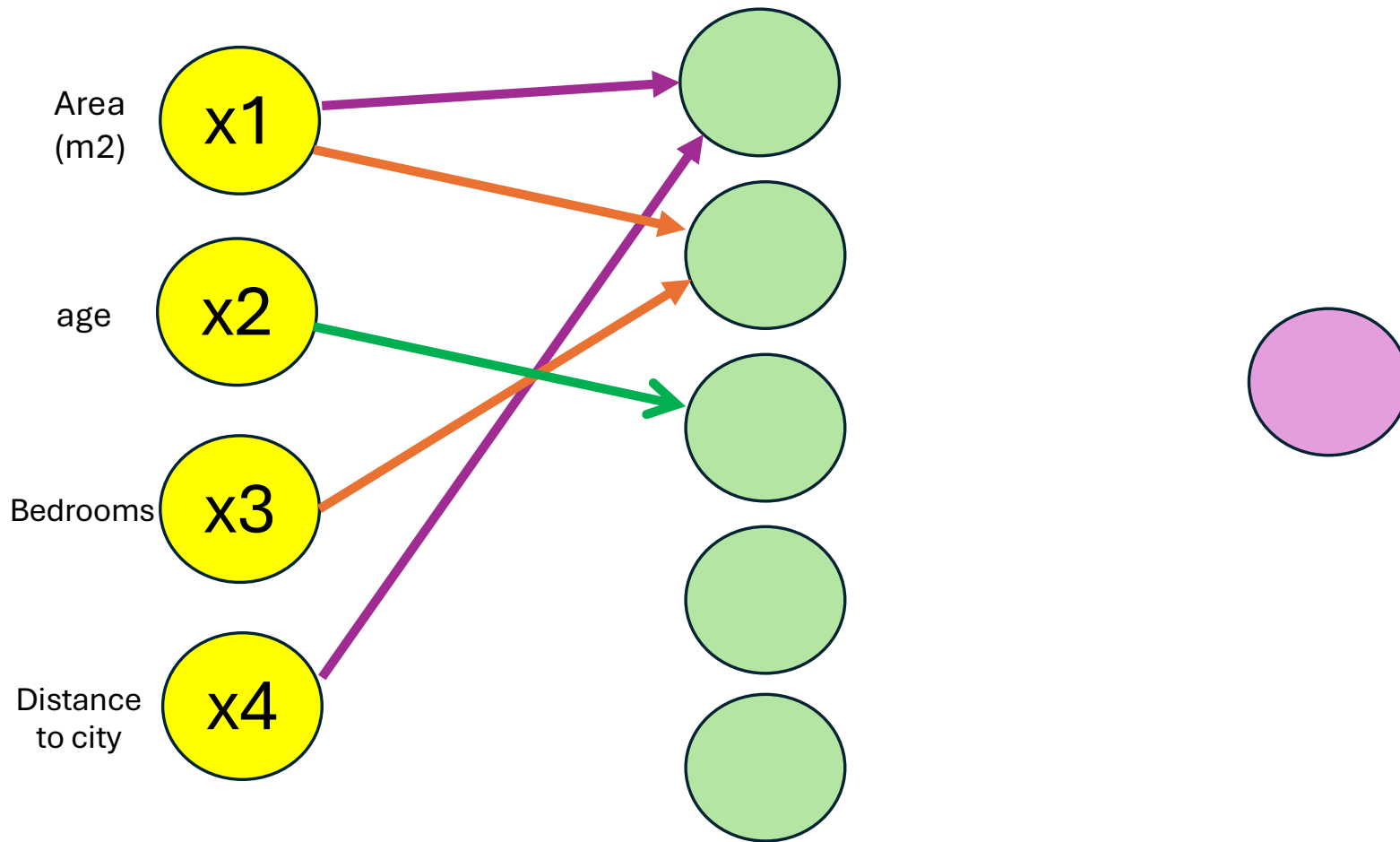
# How do NNs work?



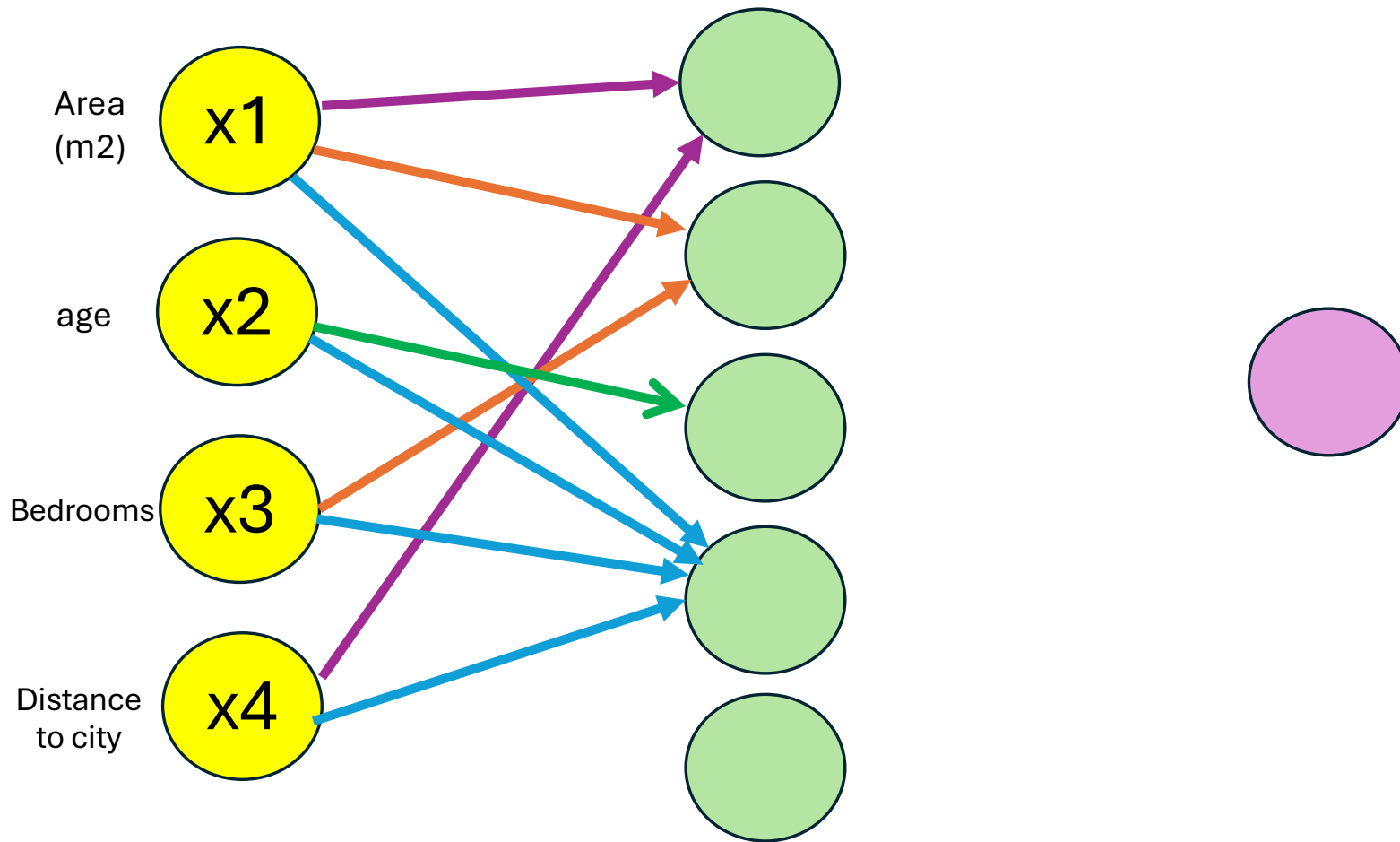
# How do NNs work?



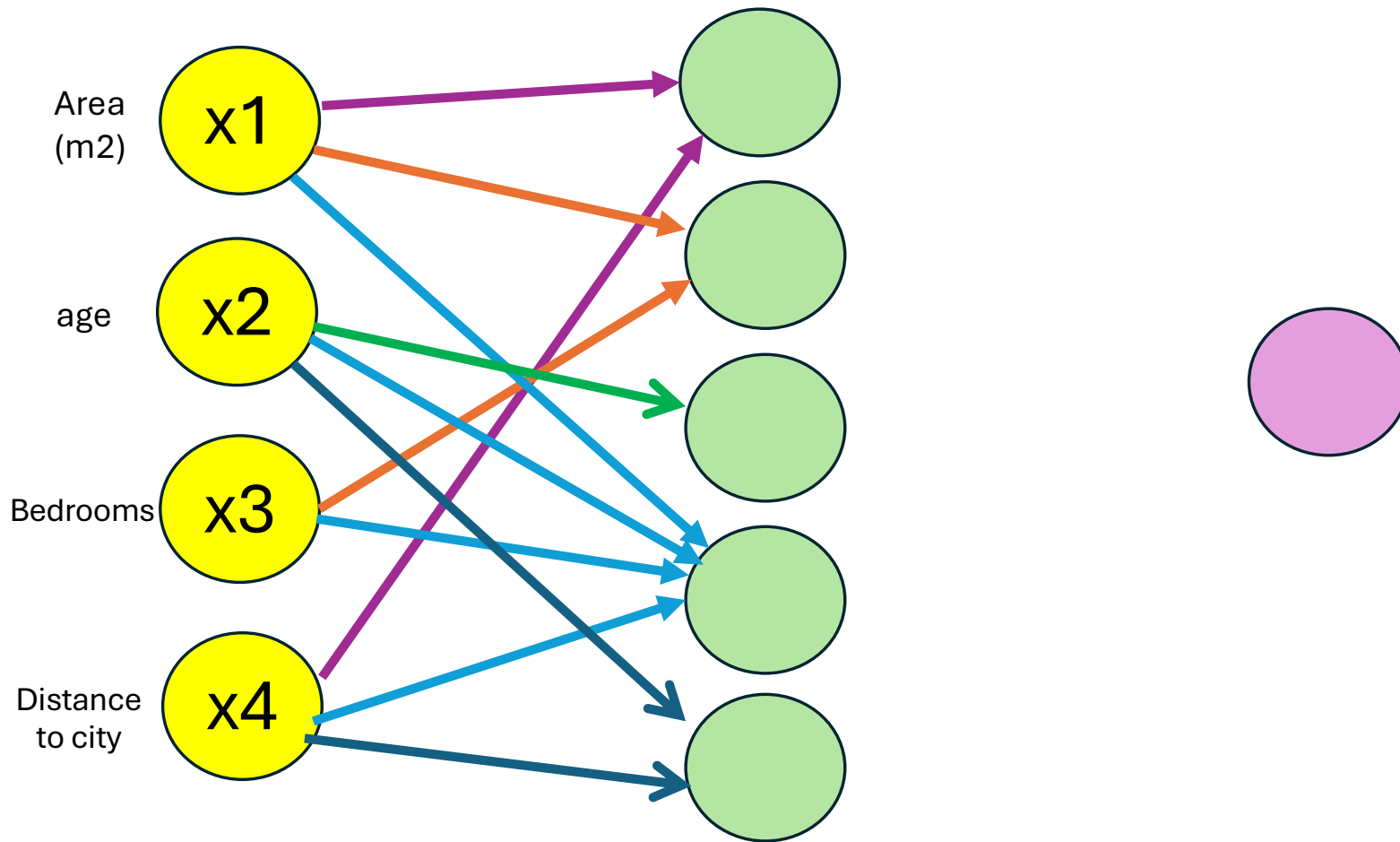
# How do NNs work?



# How do NNs work?

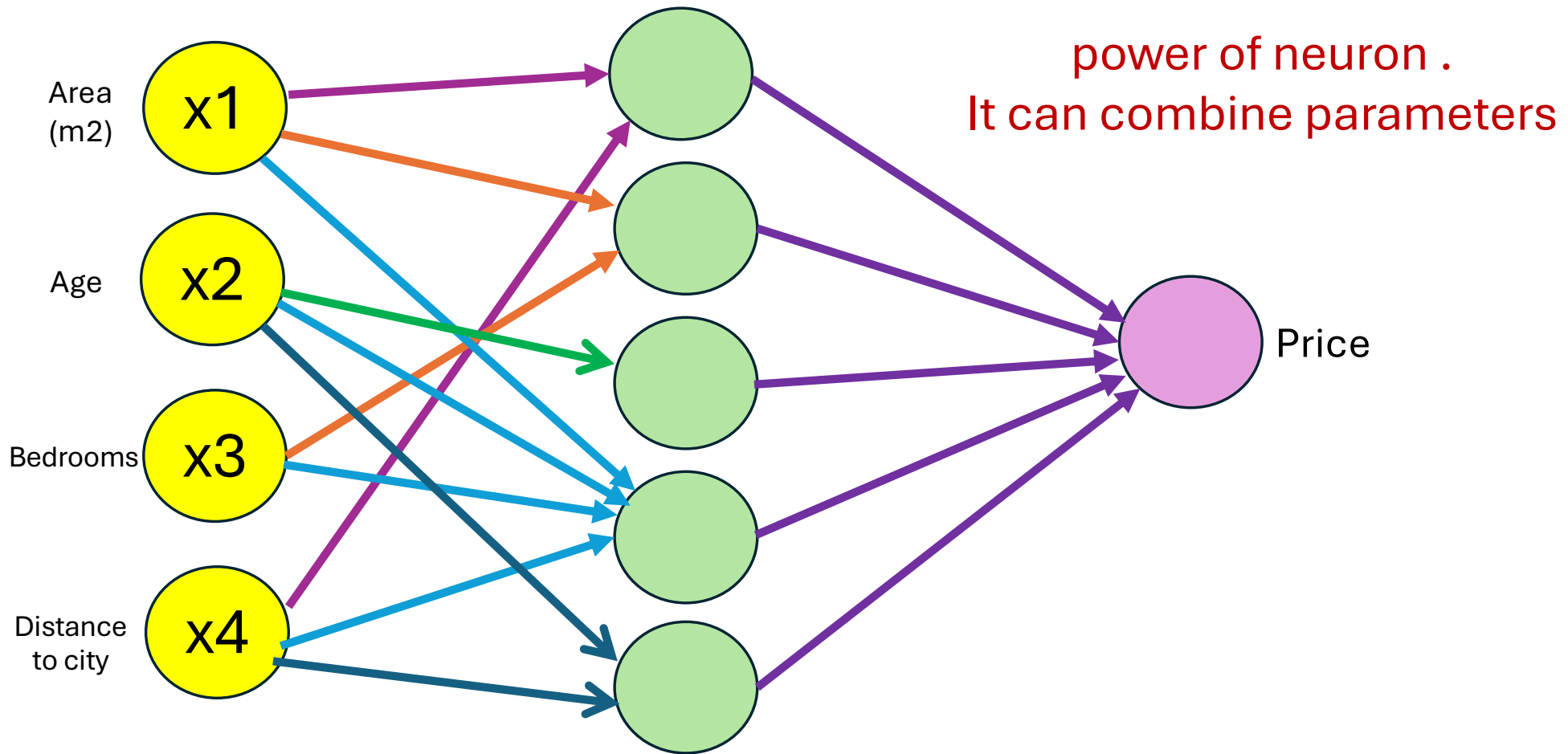


# How do NNs work?



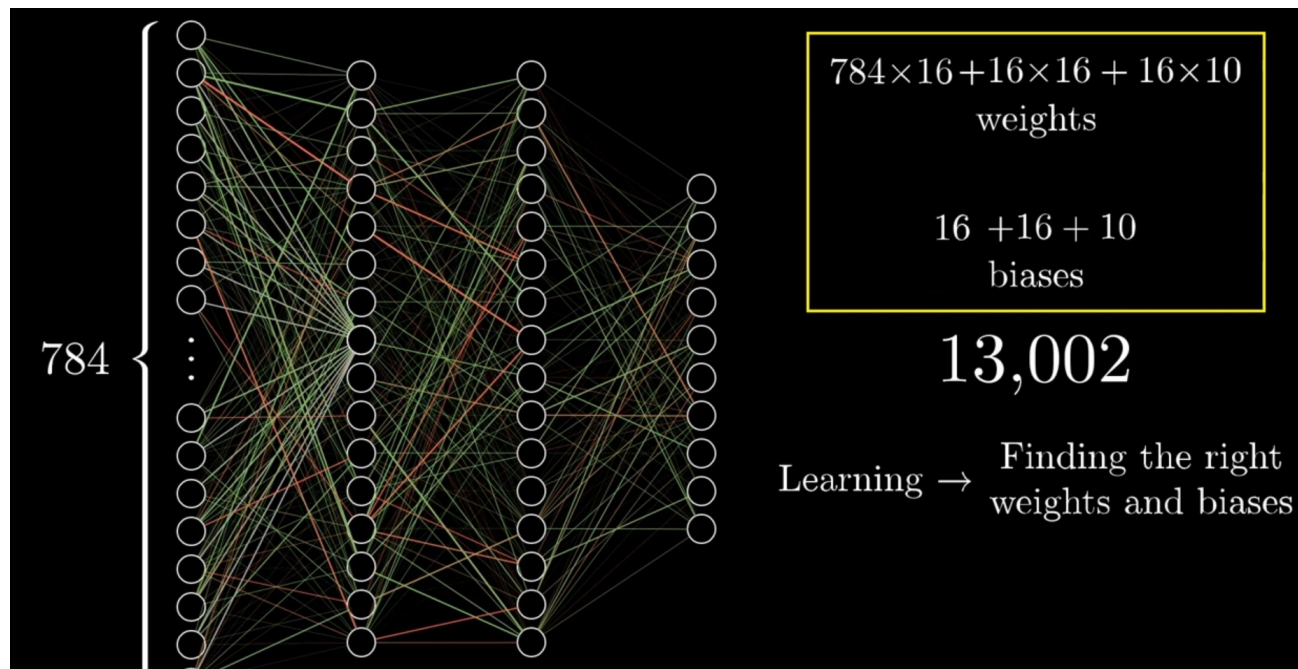


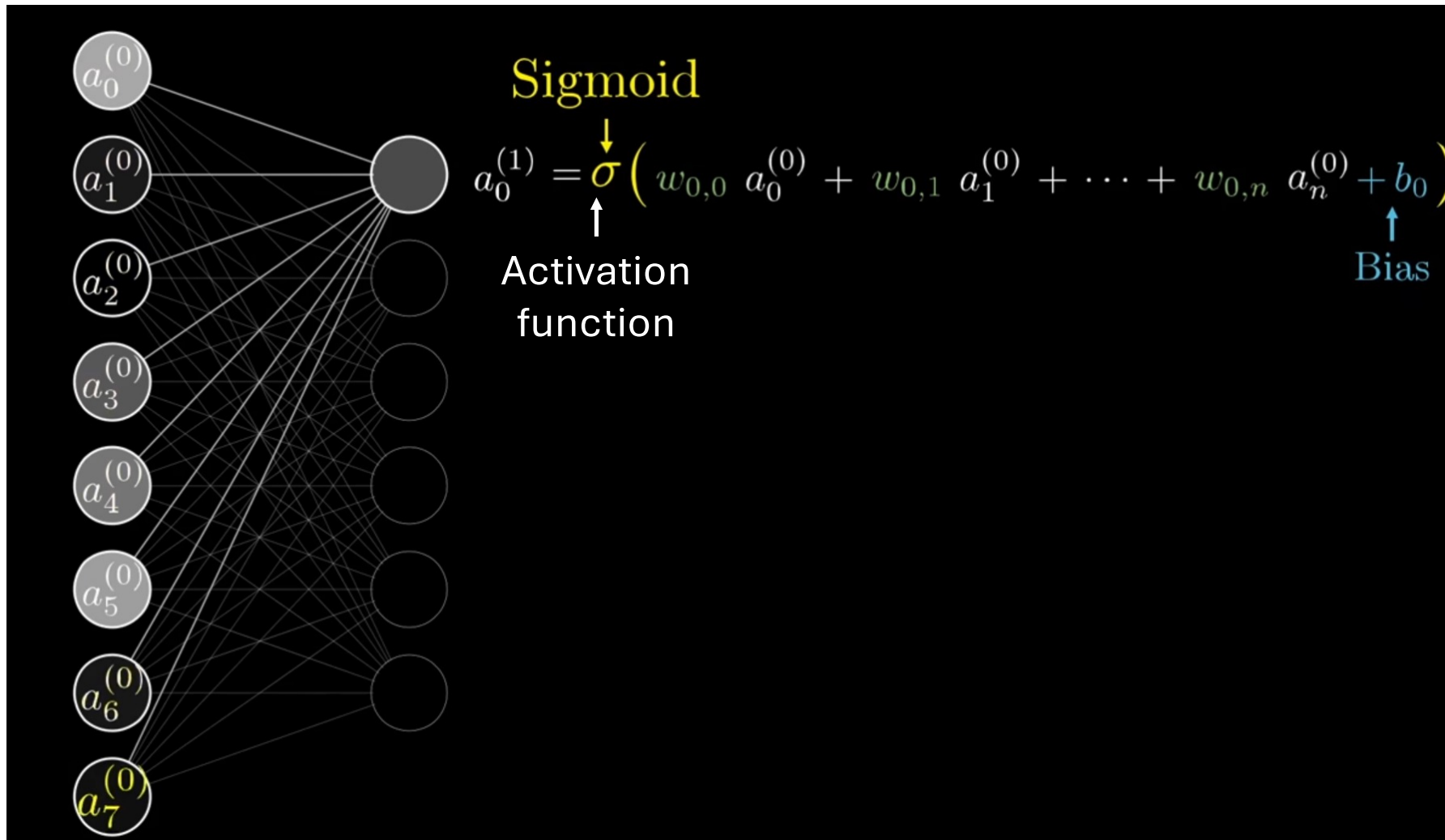
# How do NNs work?

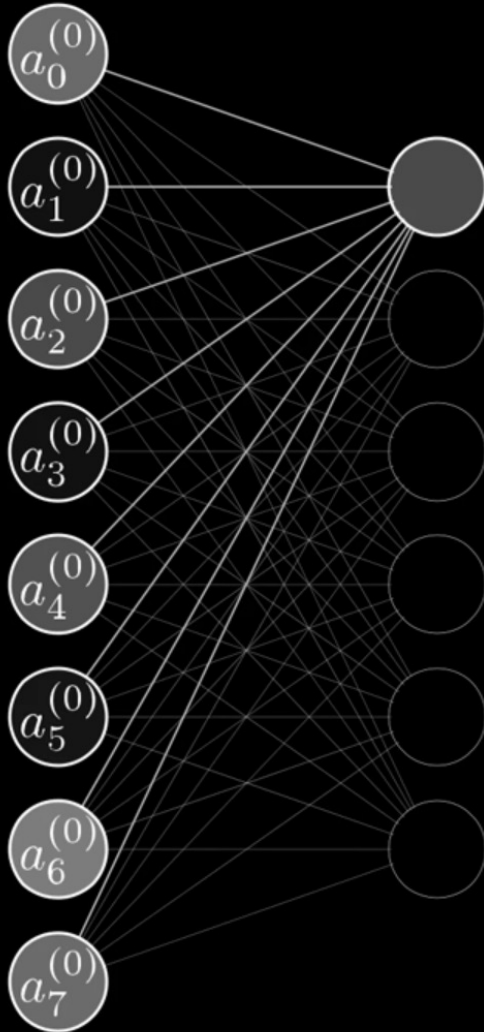


# Bias

- How high the weighted sum needs to be, before the neuron starts getting meaningfully active.
- Each neuron in hidden layer has one bias



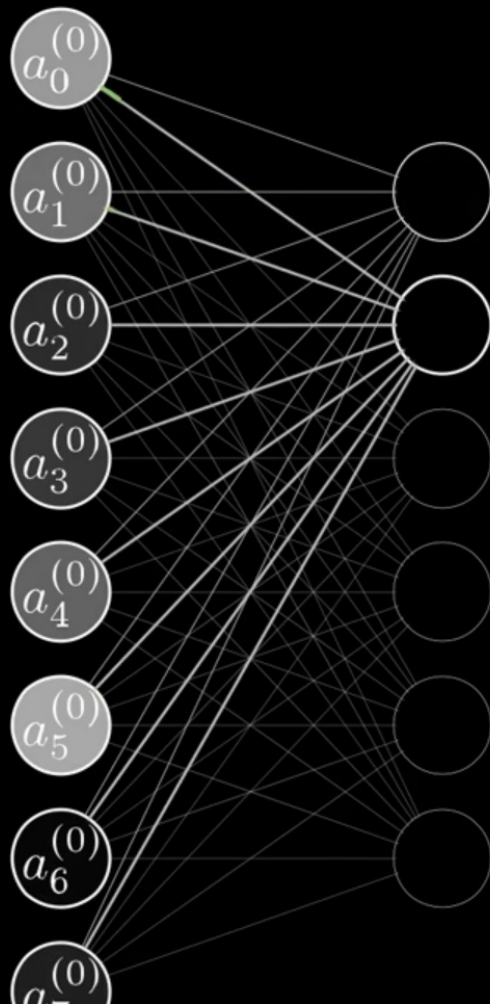




Sigmoid

$$a_0^{(1)} = \sigma \left( \underbrace{w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \dots + w_{0,n} a_n^{(0)}}_{\text{Bias}} + b_0 \right)$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \\ ? \end{bmatrix}$$

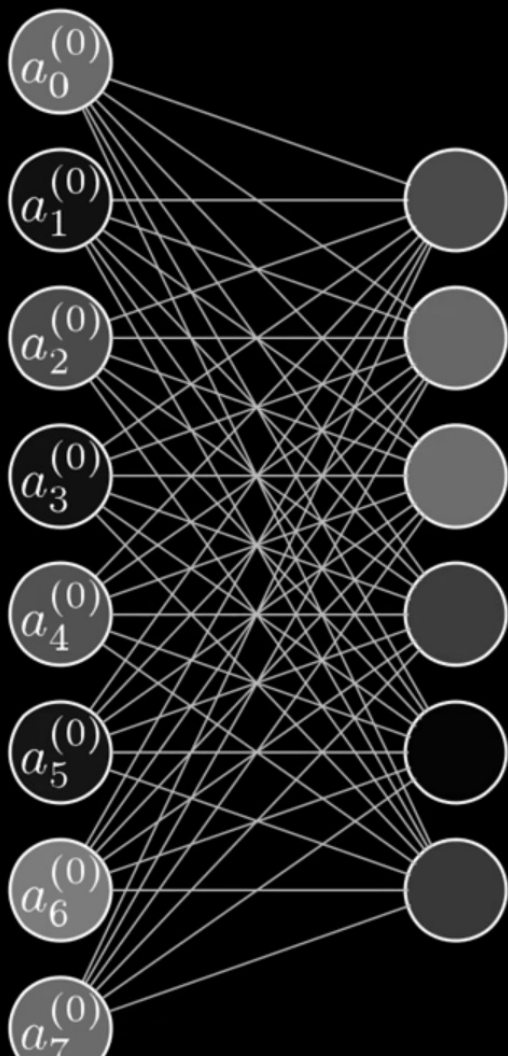


Sigmoid

$$a_0^{(1)} = \sigma \left( w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \dots + w_{0,n} a_n^{(0)} + b_0 \right)$$

↑  
Bias

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ \vdots \\ ? \end{bmatrix}$$

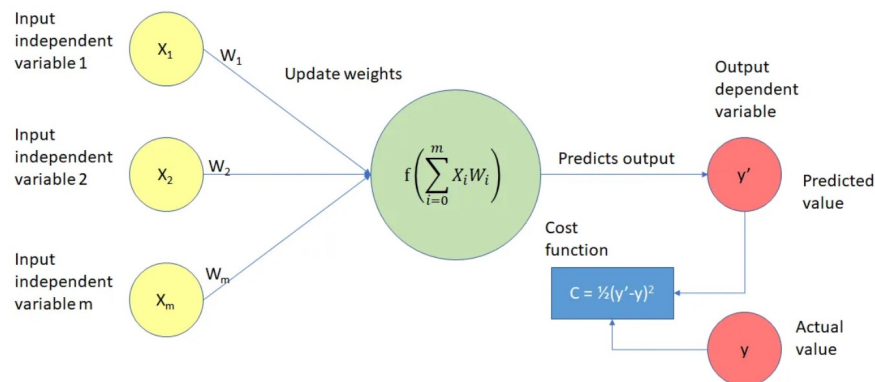


$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

$$\sigma \left( \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

# Cost function

- We initialize all of weights and biases totally randomly.
- So, the output layer is just looks like a mess.
- Now we need to define a **cost function**.
- We are trying to optimize the value of cost function.



<https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de>



# Cost function

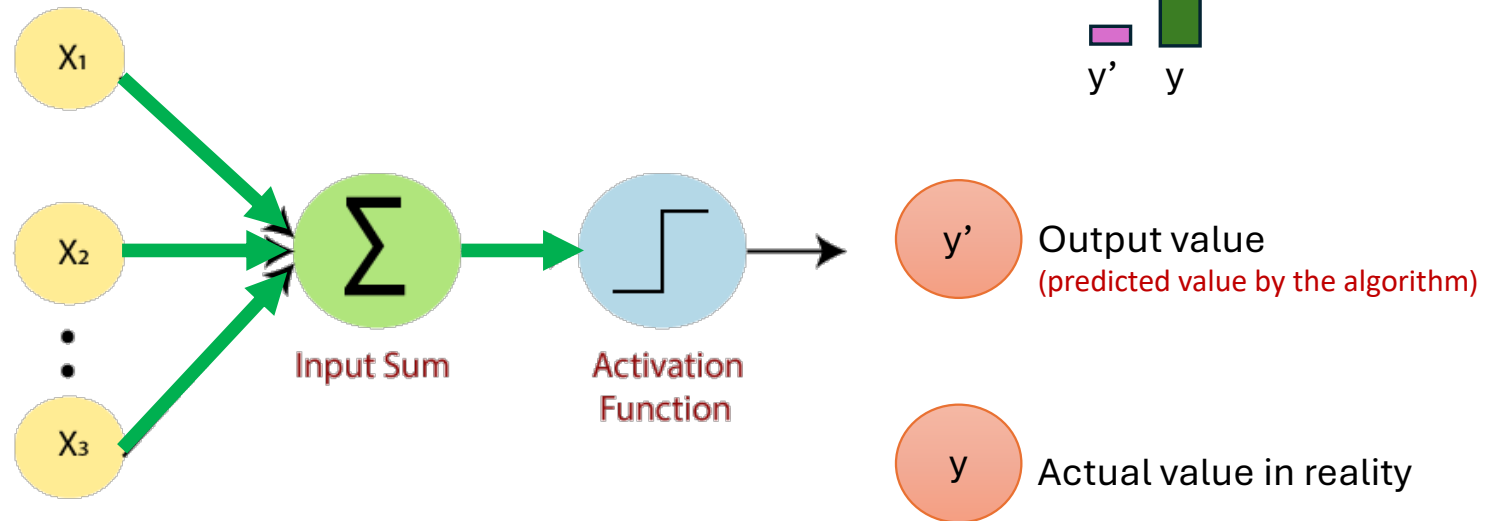
- To learn, we need to compare the **output value** to the **actual value** that we want the neural network to get.
- We are going to calculate a function called **cost function**,
  - It is calculated as one half of the squared difference between the actual value and the output value.
- There are many different functions (here we consider error)
- **Our goal is to minimize the cost function** (the lower the cost function the closer the  $\hat{y}$  is to  $y$ )

$$C = \frac{1}{2}(y' - y)^2$$

# Single layer feedforward neural network (a perceptron)

Cost function is a function that **measures the performance of a Machine Learning model** for given data.

Cost Function quantifies the error between predicted values and expected values and **presents it in the form of a single real number**.



Cost function

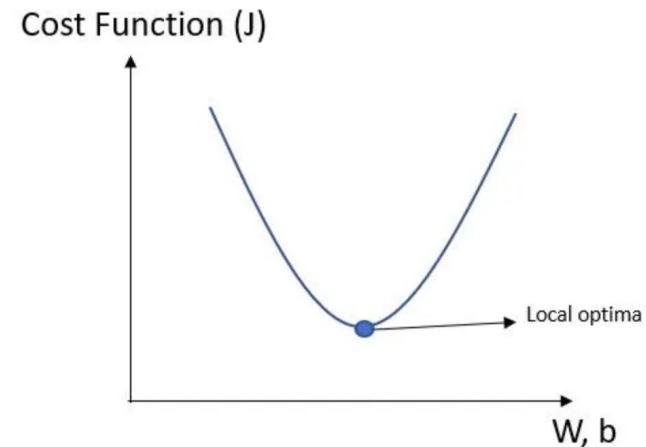
$$C = \frac{1}{2}(y' - y)^2$$

the lower the cost function,  
the closer the  $y'$  is to  $y$ .

As mentioned, the goal of an artificial neural network is to **minimize** the value of the cost function.

The cost function is minimized when your algorithm's predicted value is as close to the actual value as possible. Said differently, the goal of a neural network is to minimize the error it makes in its predictions!

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$



<https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de>

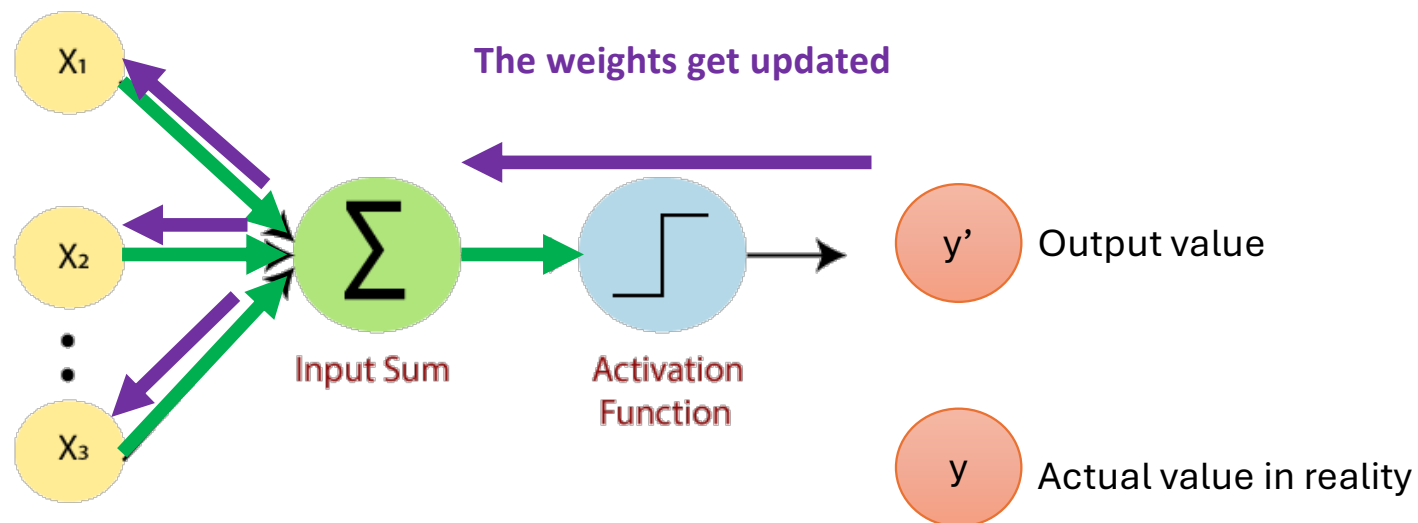
- What the cost function is telling is:

What is the error that you have in your prediction

- Just telling the computer what job it is doing and the result is not good is not very helpful.
- To make things better, we need to tell it how to change those weights and biases.
- The algorithm for computing gradient efficiently which is effectively the heart of how a neural network learns is called **back propagation**.

# Single layer feedforward neural network

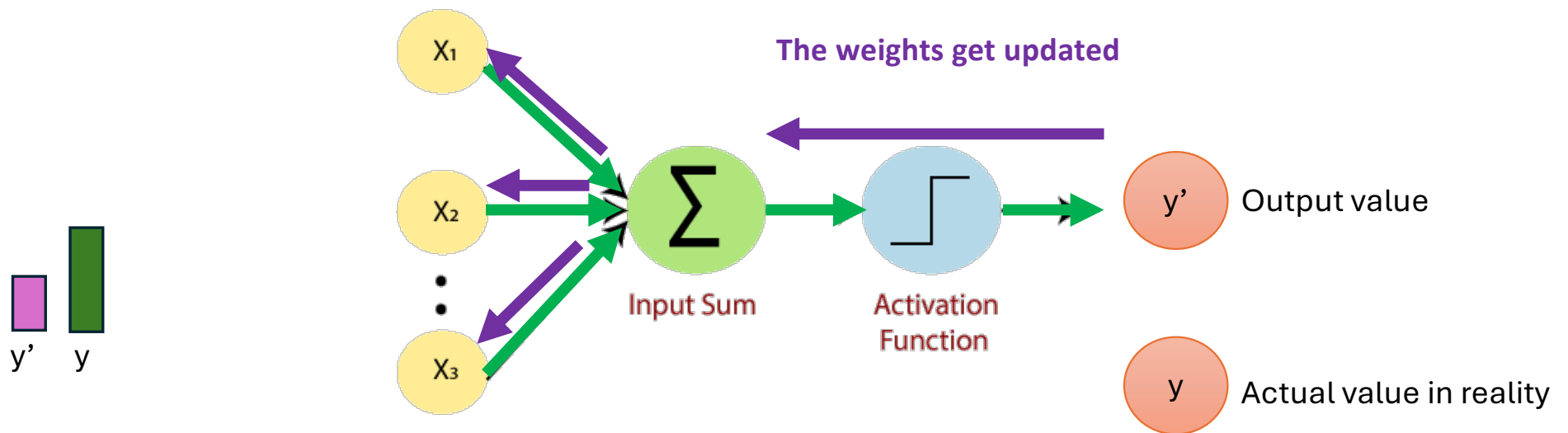
It is necessarily to know here, we're dealing with just the **one row**.



Cost function  $(Y_i - \hat{Y}_i)^2$

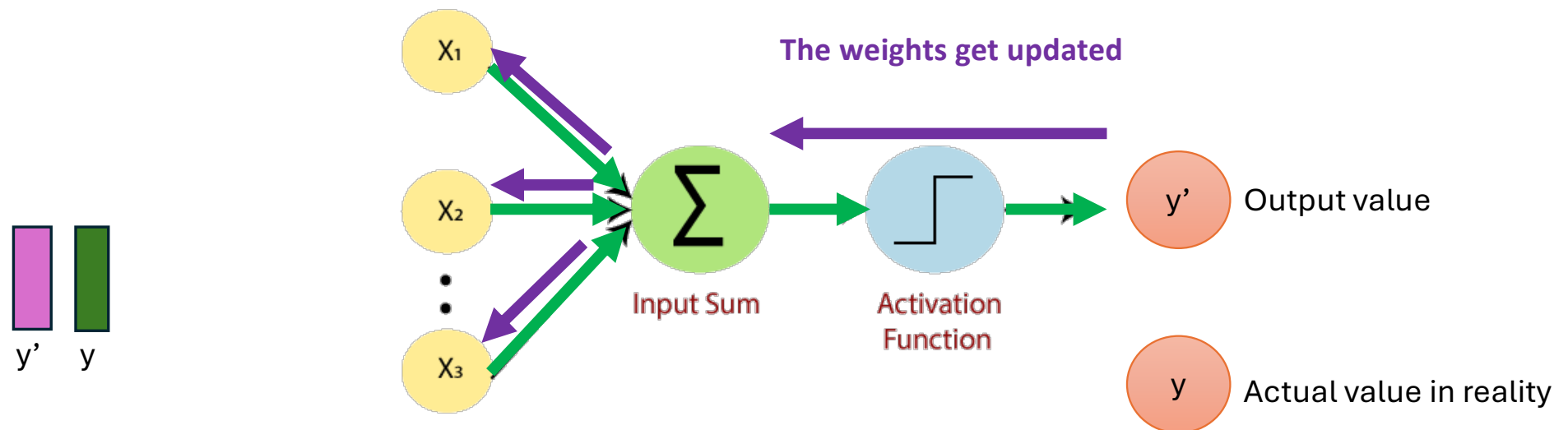
area	#bedroom	#bathroom	basement	price
7420	4	2	no	13300000

# Single layer feedforward neural network



Our output value, is changing, so our cost function is changing  $(Y_i - \hat{Y}_i)^2$   
 We back again, the weights get adjusted again

# Single layer feedforward neural network



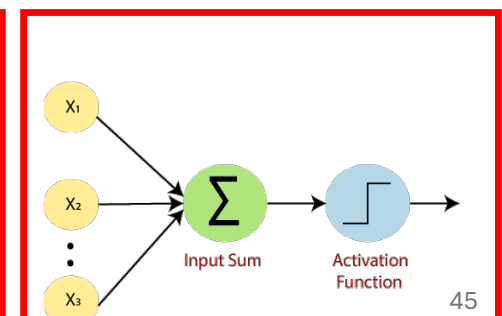
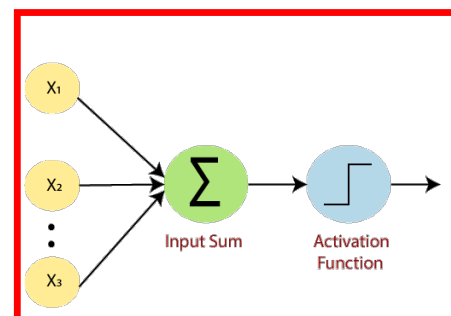
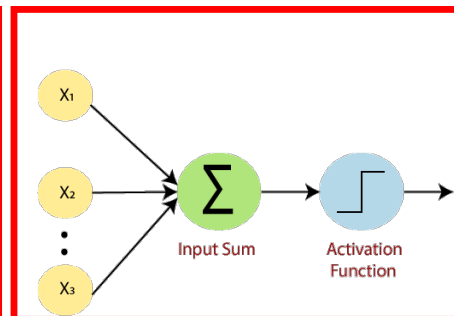
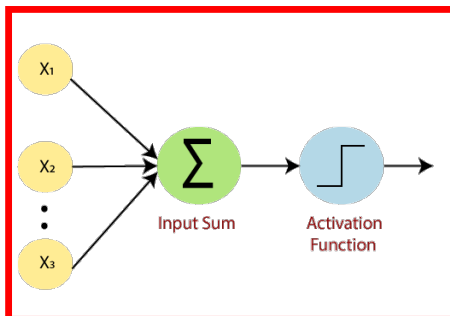
We feed the information back to the neural network again,  
the weights get adjusted again  
We try to minimize cost function

$$(Y_i - \hat{Y}_i)^2$$



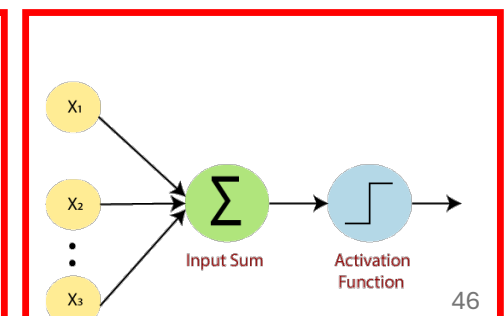
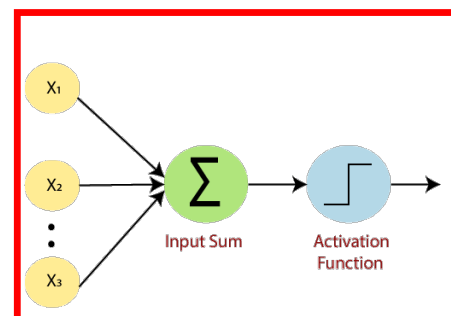
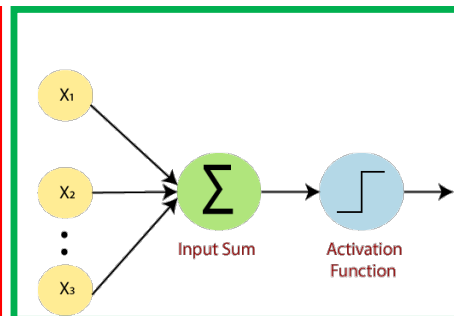
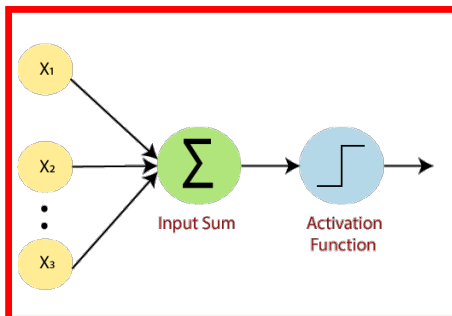
- One epoch: when we go through a whole data set and we train our neural network in all of these rows.

area	#bedroom	#bathroom	basement	price
7420	4	2	no	13300000
8960	4	2	no	12250000
9960	4	3	yes	12450000
7420	4	1	yes	11410000



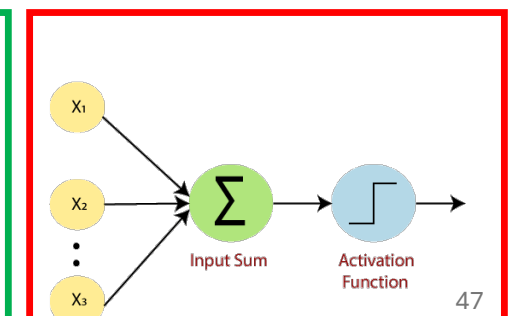
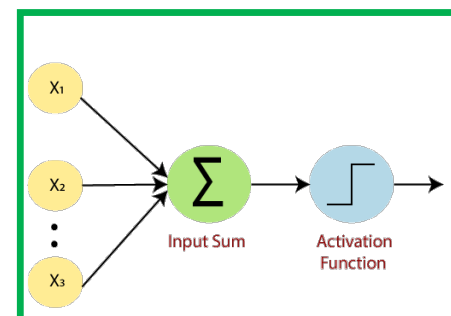
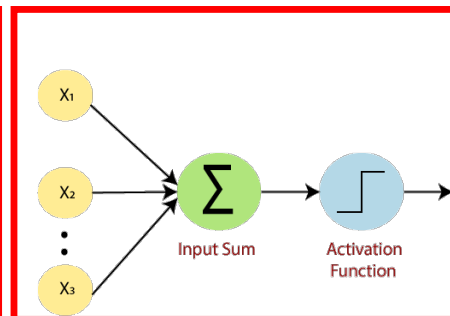
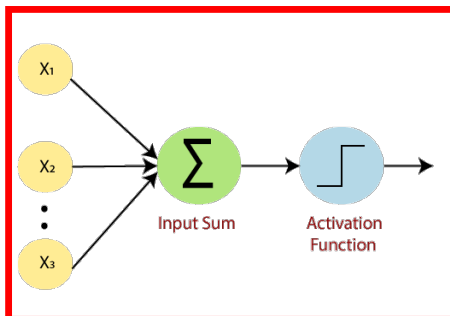
- One epoch: when we go through a whole data set and we train our neural network in all of these rows.

area	#bedroom	#bathroom	basement	price
7420	4	2	no	13300000
8960	4	2	no	12250000
9960	4	3	yes	12450000
7420	4	1	yes	11410000



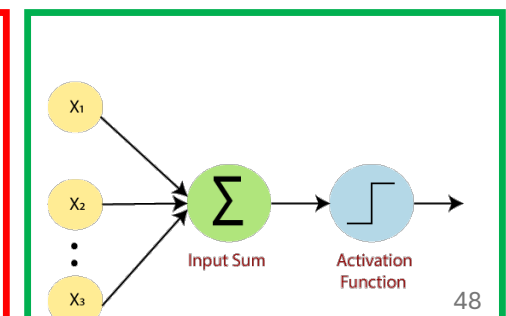
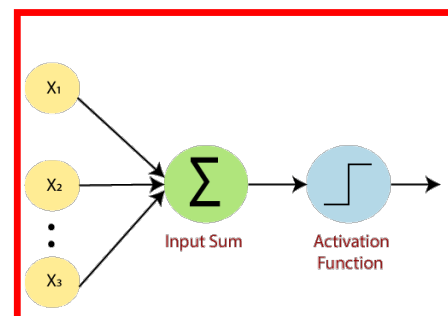
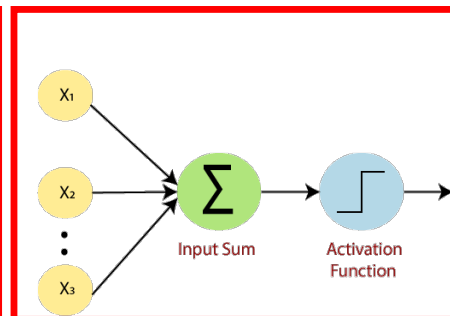
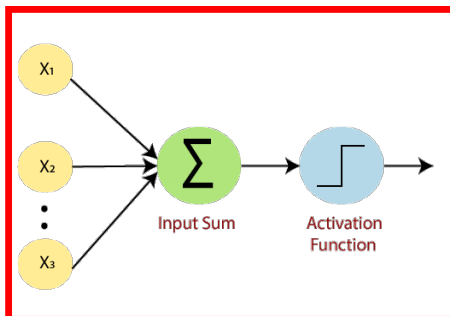
- One epoch: when we go through a whole data set and we train our neural network in all of these rows.

area	#bedroom	#bathroom	basement	price
7420	4	2	no	13300000
8960	4	2	no	12250000
9960	4	3	yes	12450000
7420	4	1	yes	11410000



- One epoch: when we go through a whole data set and we train our neural network in all of these rows.

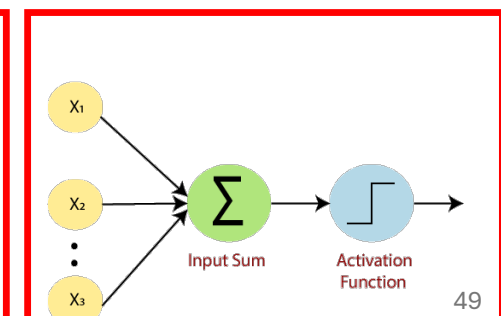
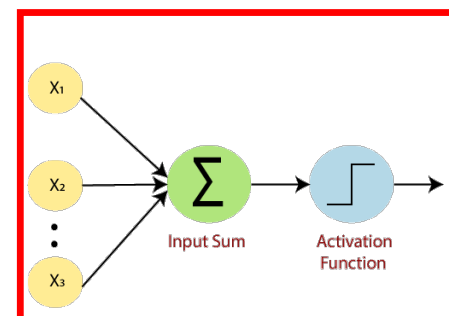
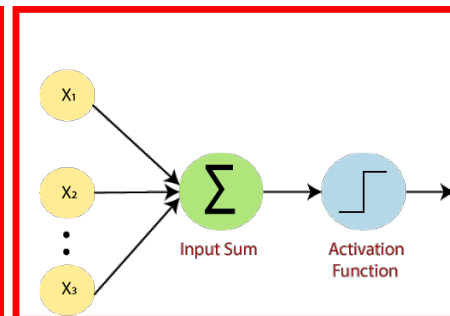
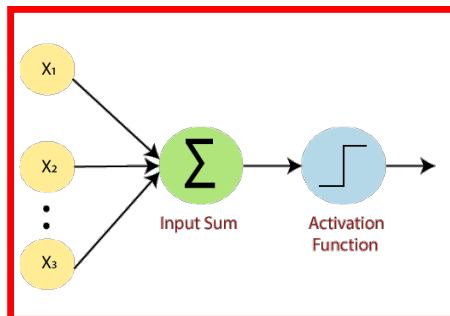
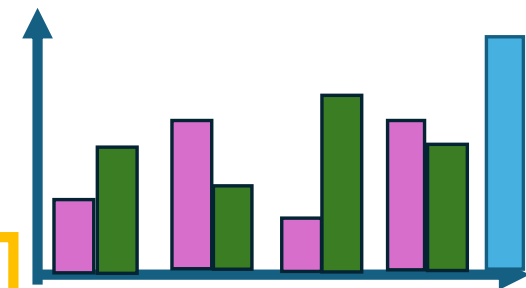
area	#bedroom	#bathroom	basement	price
7420	4	2	no	13300000
8960	4	2	no	12250000
9960	4	3	yes	12450000
7420	4	1	yes	11410000

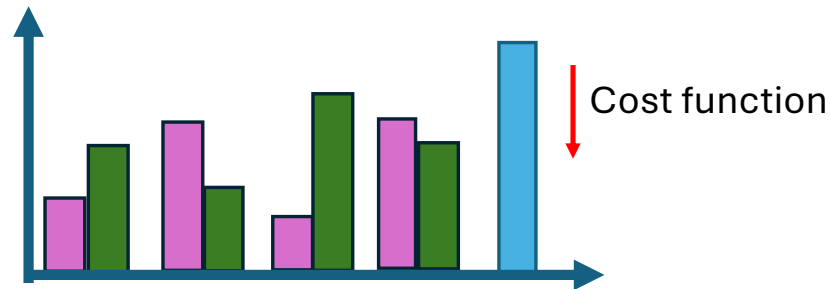


- One epoch: when we go through a whole data set and we train our neural network in all of these rows.

$$\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

area	#bedroom	#bathroom	basement	price
7420	4	2	no	13300000
8960	4	2	no	12250000
9960	4	3	yes	12450000
7420	4	1	yes	11410000





- The goal here that is minimize the cost function
- We go to the second epoch and adjust the weights
- This whole process is called **backpropagation**.
- Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.
- With propagation we are able to adjust all of the weights at the same time.

## Bank- churn customer

- For the bank, the more customers they have , the more they have money in the bank, and the more they make money from the diverse products they offer to their customers.
- Of course, they want to keep as many customers as possible, which is why they made this dataset to try to figure out why people leave the bank.
- Once they manage to build a predictive model that can predict if any new customer leaves the bank.
- To keep the money in the bank, they will be ready and may offer the customer a special deal.

# Bank- churn customer

- Customer churn means exactly the situation where some customers exit from the bank.



# print the version of TensorFlow

## Importing the libraries

```
!pip install tensorflow  
import numpy as np  
import pandas as pd  
import tensorflow as tf
```

```
tf.__version__
```

```
'2.18.0'
```

# Dataset

## Importing the dataset

```
dataset = pd.read_csv('./Churn_Modelling.csv')
dataset.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCreditCard
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	1
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	1
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1

-The customer ID is just a key identifier of each customer. So ,the customer ID has absolutely no impact on the dependent variable exited. We do not need to customer\_id. we an drop it. (same for the RowNumber column)

-Also, Surname, has no impact on the decision of a customer to stay in or leave the bank.

the decisions of the customers to stay or leave in the bank.

---

**Exited**

1

0

1

0

0

---

# Data Preprocessing

```
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

```
print(y)
```

```
[1 0 1 ... 1 1 0]
```

will also exclude these columns

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender
0	1	15634602	Hargrave	619	France	Female
1	2	15647311	Hill	608	Spain	Female
2	3	15619304	Onio	502	France	Female
3	4	15701354	Boni	699	France	Female
4	5	15737888	Mitchell	850	Spain	Female

0 1 2

other columns might  
help to predict if each  
customer

We will take all the columns starting from column 3 (the index of the column we want to start with) except the last one.

# Encoding categorical data

- we noticed that there are two categorical variables.

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	H
619	France	Female	42	2	0.00	1	
608	Spain	Female	41	1	83807.86	1	
502	France	Female	42	8	159660.80	3	
699	France	Female	39	1	0.00	2	
850	Spain	Female	43	2	125510.82	1	

Encoding :

- zero and one for the gender
- some one-hot encoding for this categorical variables in which indeed there is no relationship order between these values, between France, Spain and Germany.

# Encoding

- For gender:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

Gender  
column

Gender  
column

- For Geography:

there is no order relationship between France, Spain and Germany. So we couldn't, you know, encode France into zero, then Spain into one and Germany into three. We have to perform one hot encoding instead.

```
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

*#One Hot Encoding the "Geography" column*

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

Index of geography in dataset

## Splitting the dataset into the Training set and Test set

```
#Splitting the dataset into the Training set and Test set  
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

# Feature scaling

- Feature scaling is absolutely compulsory for deep learning.
- Apply feature scaling to all the features of both the training set and the test set.

```
#feature scaling  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X_train = sc.fit_transform(X_train)  
X_test = sc.transform(X_test)
```

in this slide, the data preprocessing phase will be over.



# Building ANN

```
import tensorflow as tf
ann = tf.keras.models.Sequential()
```

```
#Adding the input layer and the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

↑  
Add any new layer

↑  
The number of  
neurons in hidden  
layer

↑  
Rectifier  
Function

We call them **hyperparameters** in the sense that these are parameters that won't be trained during the training process.

Unfortunately, there is no rule to find the number of neurons. It is just based on experimentation .

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding second hidden layer

## Adding the output layer

- The output layer contains the dimensions of the output you want to predict.
- And here since we want to predict a binary variable. So, we only need **one** neuron for output layer.
- The activation function of the output layer, is a **sigmoid** activation function;
- it'll give you the probabilities that the binary outcome is 1

Exited

1

0

1

0

0

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Congratulations. We're done with the creation of this artificial neural network.

# Training the ANN

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

**Epoch: to improve the accuracy over time.**

Epoch 1/100

250/250 ————— 1s 452us/step - accuracy: 0.7925 - loss: 0.5764

Epoch 2/100

250/250 ————— 0s 357us/step - accuracy: 0.7922 - loss: 0.5015

Epoch 3/100

250/250 ————— 0s 355us/step - accuracy: 0.7921 - loss: 0.4747

Epoch 4/100

250/250 ————— 0s 360us/step - accuracy: 0.7931 - loss: 0.4587

Epoch 5/100

**Accuracy increase during the epochs**

Epoch 99/100

250/250 ————— 0s 362us/step - accuracy: 0.8673 - loss: 0.3218

Epoch 100/100

250/250 ————— 0s 359us/step - accuracy: 0.8629 - loss: 0.3357

# Prediction

- We are going to predict if the new customer with following information will leave the bank.
- We are going to get the probability that this customer leaves the bank.

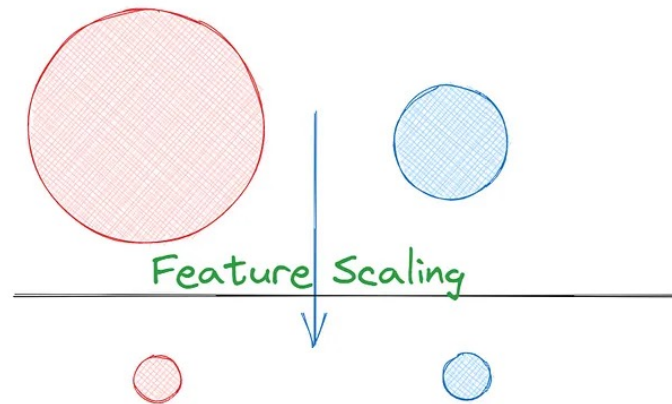
```
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 70, 3, 40000, 2, 1, 1, 10000]])))
```

```
1/1 ————— 0s 17ms/step  
[[0.01680748]]
```

This customer has a very low chance to leave the bank.

# Feature Scaling

- Imagine you're comparing apples and oranges; it's nonsensical.
- Features measured in vastly different units (like income and age) can skew machine learning algorithms if not addressed.
- Feature scaling addresses this by transforming data into a standard scale, enabling fair comparison between different features.



# Feature Scaling

- Feature scaling is a fundamental preprocessing step in machine learning aimed at ensuring that numerical features have a **similar scale**.
- Apply feature scaling before feeding the data into the machine learning model (K-NN), **except** for algorithms that are scale-invariant, such as **decision trees**.

# Common Techniques for Feature Scaling

- **Normalization :**

- This method scales each feature so that all values are within the range of 0 and 1.
- $x$  is the value you want to normalize
- $\min(X)$  is the minimum value in your data set
- $\max(X)$  is the maximum value in your data set

$$(x - \min(X)) / (\max(X) - \min(X))$$

- **Standardization (Z-score Scaling):**

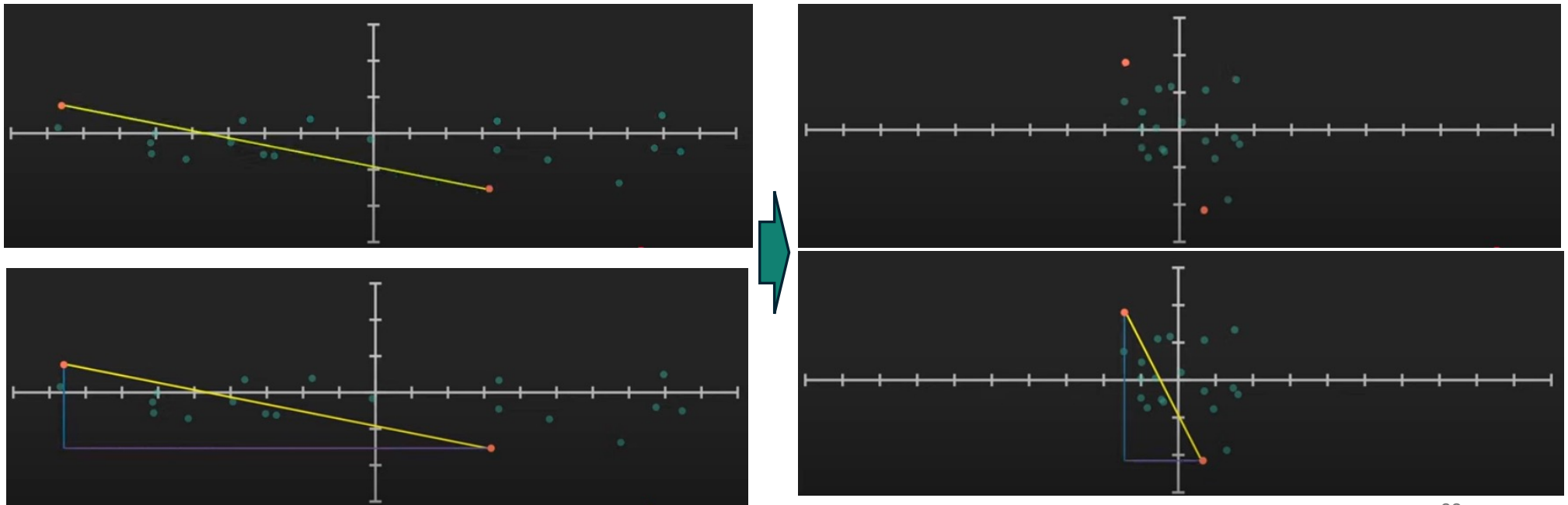
- Here, each feature is transformed to have a mean of 0 and a standard deviation of 1.
- This is achieved by subtracting the mean value and dividing by the standard deviation of the feature.
- $x$  is the original value you want to standardize,  $\mu$  (mu) is the mean of the data set,  $\sigma$  (sigma) is the standard deviation of the data set

$$Z = (x - \mu) / \sigma$$

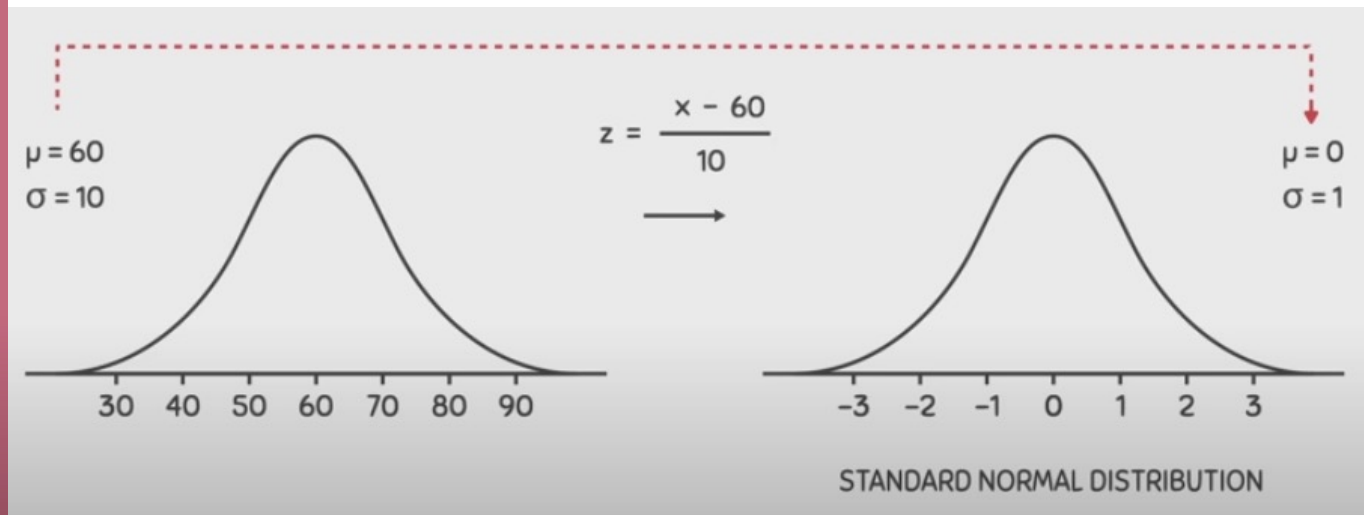
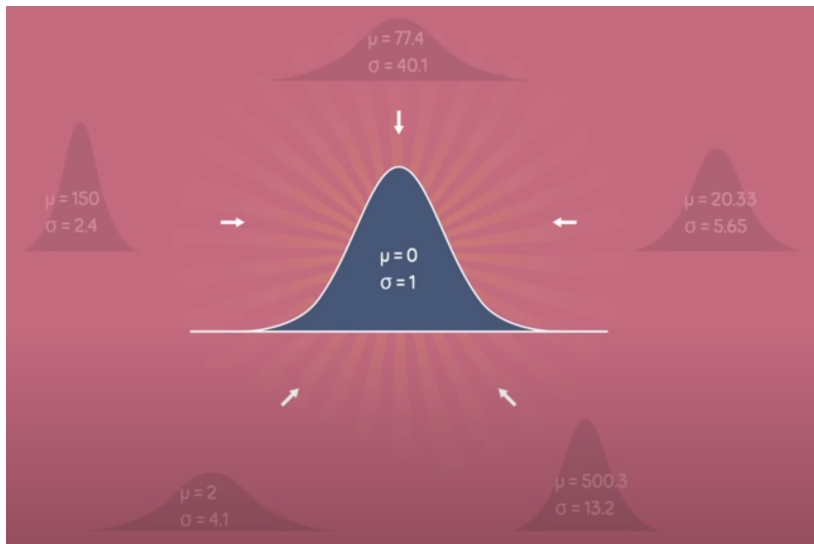


# Feature Scaling

- In the last figure, both feature contribute equally to the distance
- After that, your algorithm won't be affected by the feature with a higher scale.



```
from sklearn.preprocessing import StandardScaler
stdsc=StandardScaler()
X_std=pd.DataFrame(stdsc.fit_transform(X_all), index=X_all.index, columns=X_all.columns)
```



Any normal distribution with any value of mean and standard deviation(sigma) can be transformed into the standard normal distribution where we have a mean of zero and a standard deviation of one.

END