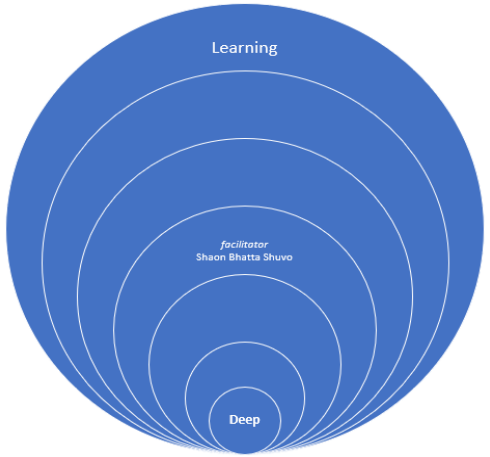


Workshop (Part1) on Deep Learning

facilitator

Shaon Bhatta Shuvo

PhD Student, School of Computer Science

University of Windsor, Ontario, Canada.

Email: shuvos@uwindsor.ca

University of Windsor

Key Topics

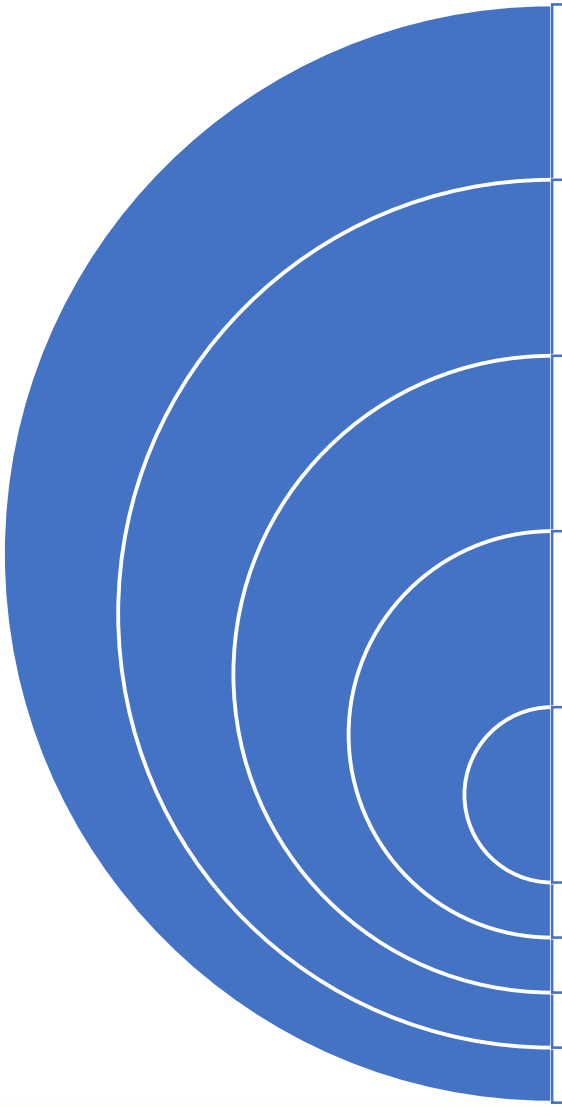
Why Deep Learning?
What is Deep Learning?
Machine Learning vs Deep Learning.
Why Deep Learning is getting popular now?
Concept Neuron, Artificial Neuron and Neural Networks.
Why do we call it Deep Learning?
How Neural Networks work?
Ingredients to train a Deep Neural Network.
Overview: How do Neural Networks Learn?
Parameters vs. Hyperparameters.
Deep Learning hands-on.

Why Deep Learning? (Applications)

Deep Learning shines when it comes to complex problems such as pattern recognition, image (/video) classification, natural language processing, and speech recognition.

Some major fields of Deep Learning applications include:	Computer Vision and Pattern recognition
	Autonomous vehicle
	Speech Recognition
	Information Retrieval
	Aerospace and Defense
	Marketing
	Medical Diagnosis and Drug Discovery
	Agriculture and Other Industries
	Natural Language Processing
	Social Network Analysis

Why Deep Learning? (Employment Opportunities)



Artificial Intelligence(AI) and Machine Learning job postings on Indeed rose **29.10%** between May 2018 and May 2019.

Machine Learning and Deep Learning Engineers are the **most popular** jobs posted on Indeed between 2018 and 2019.

Machine Learning Engineers are earning an average salary of **$142,858.57** in 2019 based on an analysis of all open positions on Indeed.

40 percent say they're adding jobs due to AI

133 million new jobs created by AI by 2022

Source: Forbes.com

Why Deep Learning? (Employment Opportunities)

#2 in-demand AI job: Deep learning engineer

Top 10 jobs involving AI skills

Top jobs seeking machine learning or artificial intelligence skills

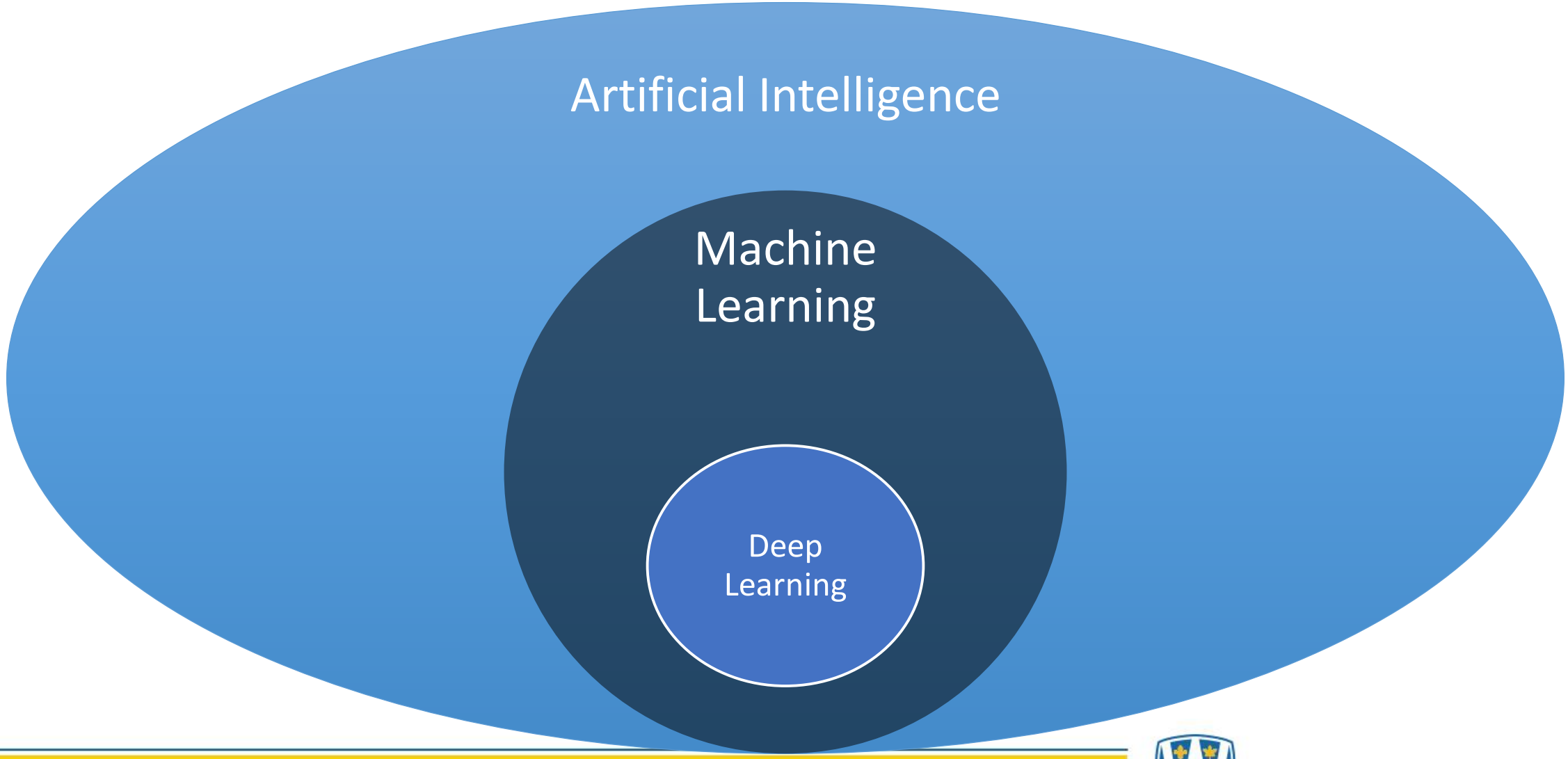
Rank	Job title	% of postings containing AI or machine learning	Rank	Job title	% of postings containing AI or machine learning
1.	Machine learning engineer	75.0%	6.	Algorithm developer	46.9%
2.	Deep learning engineer	60.9%	7.	Junior data scientist	45.7%
3.	Senior data scientist	58.1%	8.	Developer consultant	44.5%
4.	Computer vision engineer	55.2%	9.	Director of data science	41.5%
5.	Data scientist	52.1%	10.	Lead data scientist	32.7%

Source: Indeed

indeed

Deep learning didn't even appear on the same list previous year!

What is Deep Learning?



What is Deep Learning?

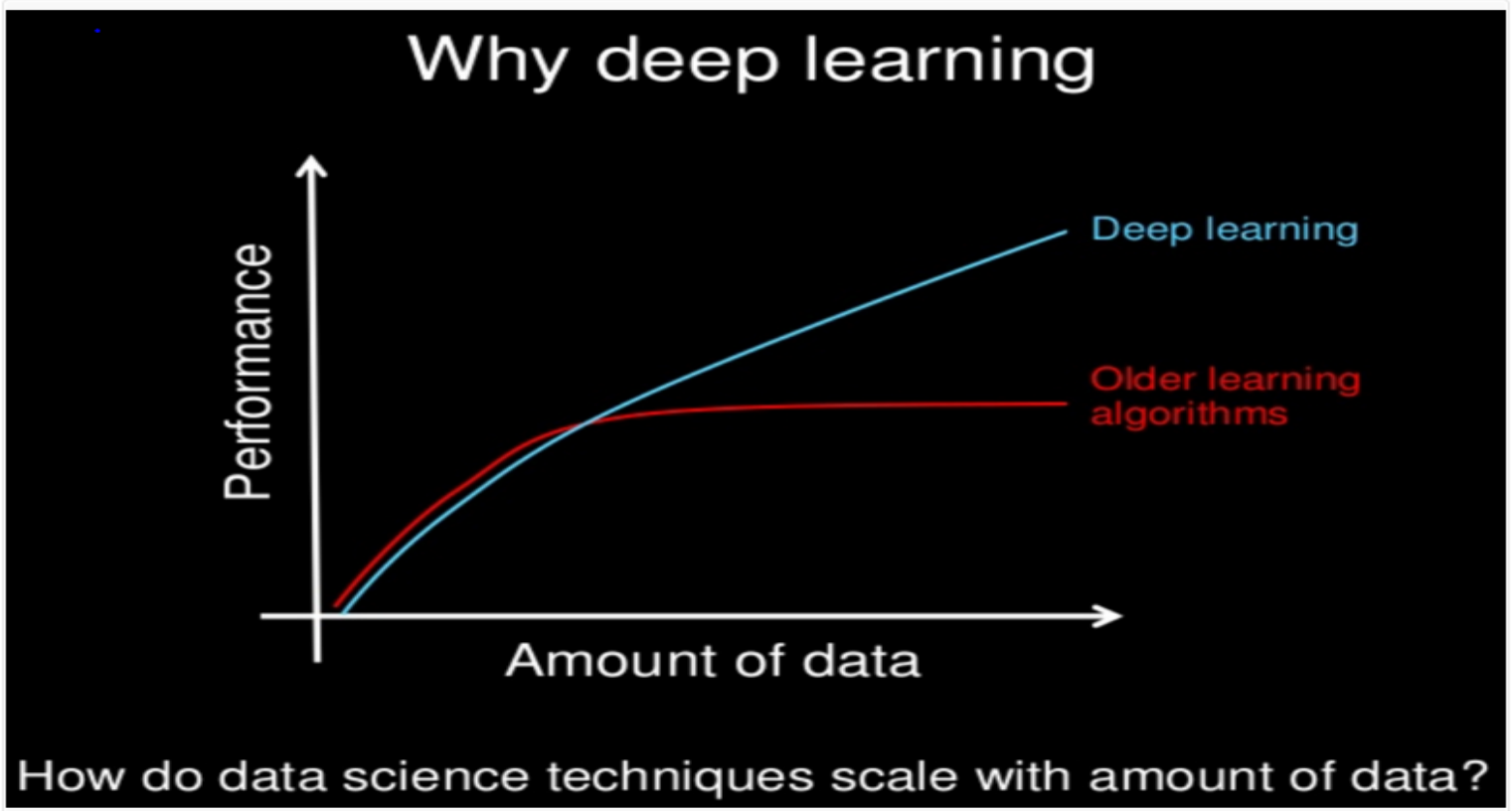
- Deep Learning is a subfield of Machine Learning** concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. [source: machinelearningmastery.com]

- Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. [source: mathworks.com]

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

Machine Learning vs. Deep Learning

- Data Dependency:



Why Deep Learning?
Slide by Andrew Ng, all rights reserved.

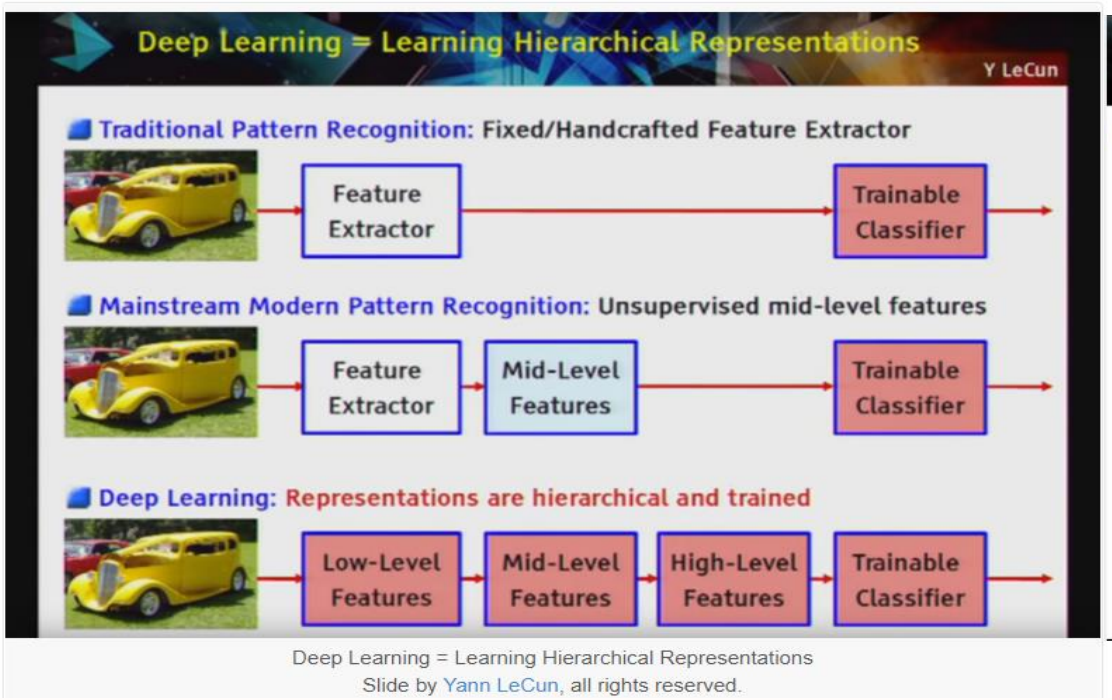
Machine Learning vs. Deep Learning

❑Features Engineering:

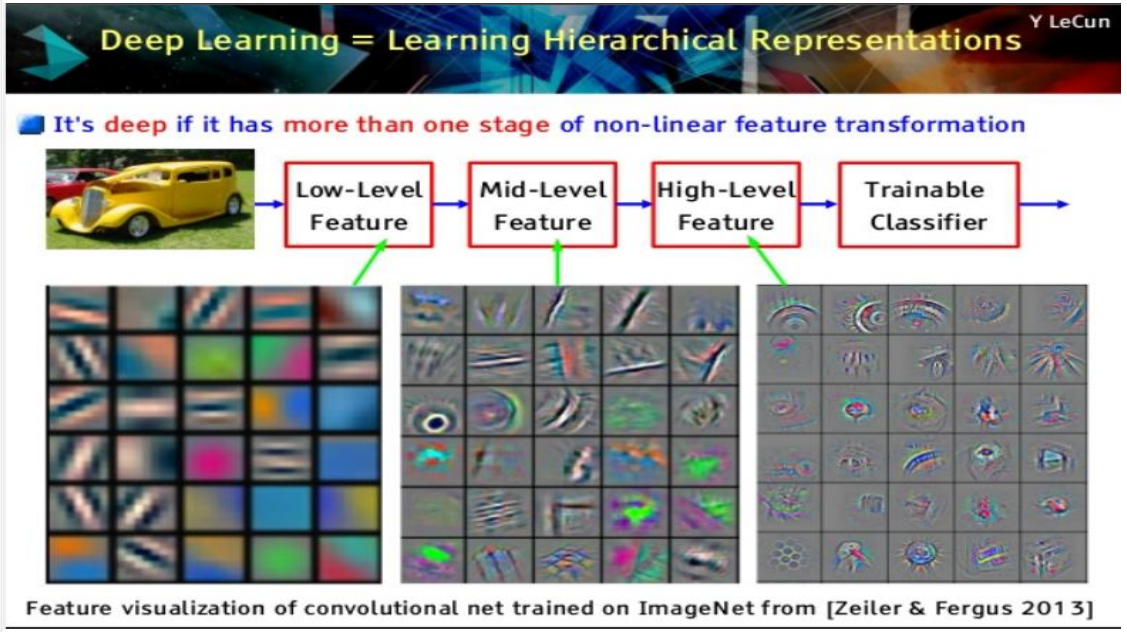
-In Machine learning, most of the applied features need to be identified by an expert.

-Deep learning algorithms try to learn high-level features from data.

(For example, in case of image classification features can be pixel values, shape, textures, position and orientation.)

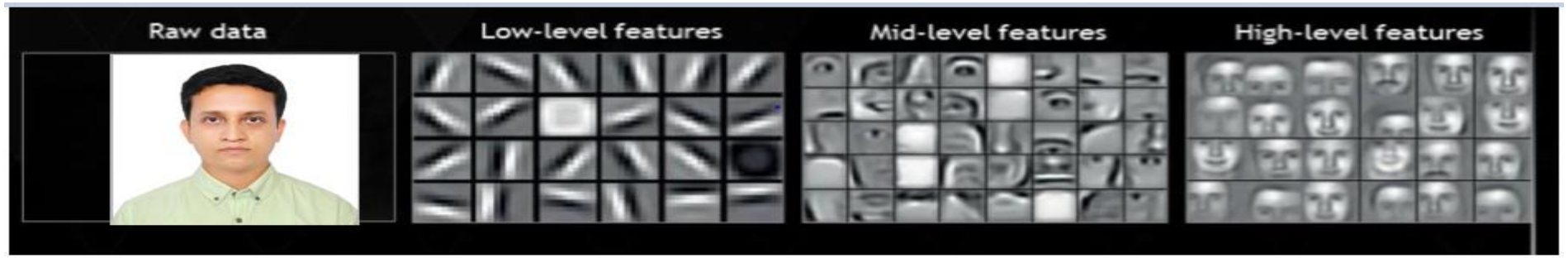


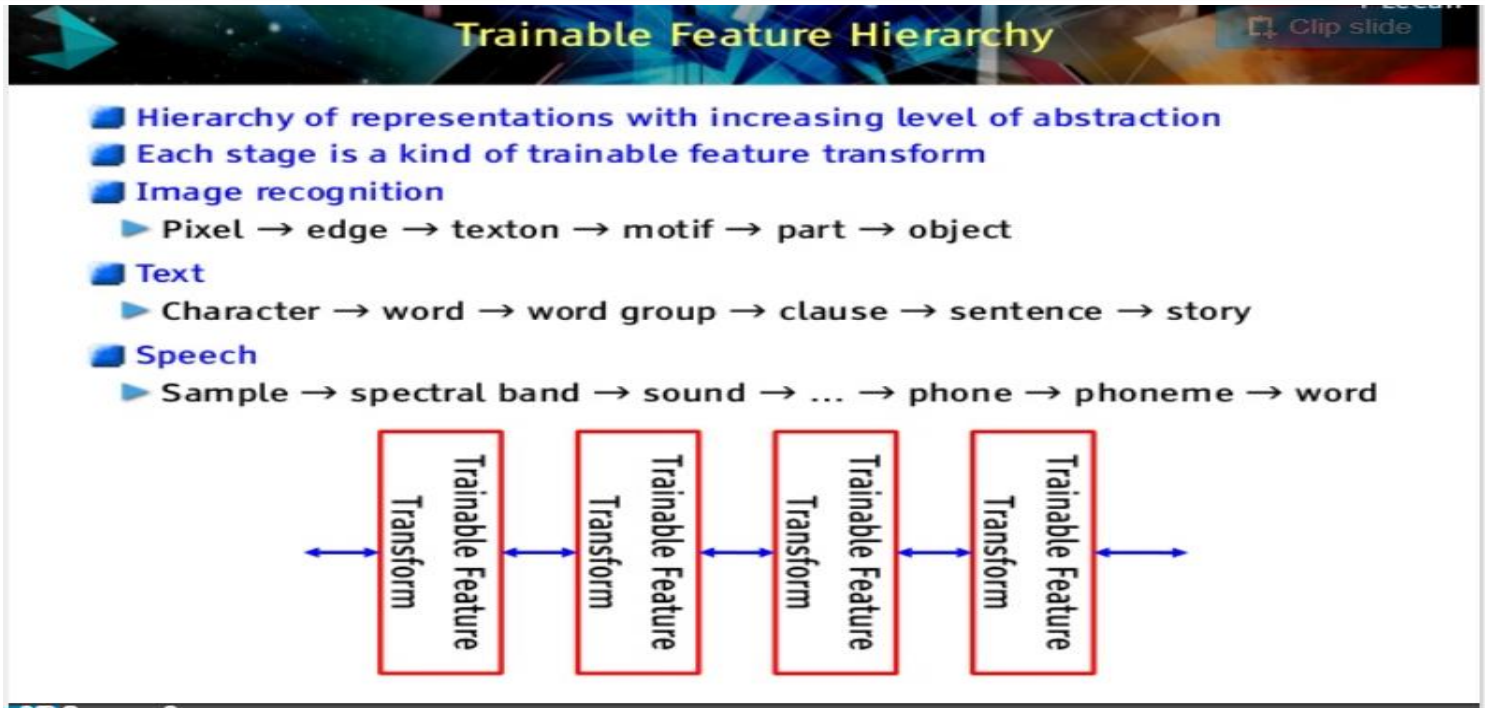
Deep Learning = Learning Hierarchical Representations
Slide by Yann LeCun, all rights reserved.



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Feature Engineering





Trainable Features Hierarchy
Slides by Yann LeCun, all rights reserved

Machine Learning vs. Deep Learning

- Execution Time:
  - Deep Leaning: Requires long time to train.
  - Machine Learning: Requires comparatively much less time to train.
- Hardware Dependencies:
  - Deep Learning: Requires high-end machines.
  - Machine Learning: Algorithms can work on low-end machines.
- Interpretability:
  - Deep Learning: It is often difficult to interpret the reason behind the outcome. (Attention Mechanism may help in that case)
  - Machine Learning: Comparatively easy to interpret the reasoning behind the result.

Machine Learning vs. Deep Learning

- Problem Solving Approach:

  -Machine Learning: It is generally recommend to divide the problem into smaller parts, then solve them individually and finally combine them to get the result.

  -Deep Learning: Advocates to solve the problem end-to-end.

  For example:

  Problem: Multiple object detection in an image.

  ML Approach: Typical ML approach would divide the problem into two steps, object detection and object recognition. In that case, we may need to combine tow different algorithms for two different task.

  Deep Learning: Would process end-to-end. We just pass an image, it would give us the outcome without the help of different algorithm.

God Fathers of Deep Learning

Geoffrey Hinton: Geoffrey Everest Hinton is an English Canadian cognitive psychologist, professor of computer science at the University of Toronto and a part of the Google Brain project. Hinton is considered by many to be the 'godfather of deep learning.

Some people also add two other scientists in the list:

Yoshua Bengio: A Canadian computer scientist, professor at the University of Montreal, Canada and started an AI company called Element AI.

Yaan LeCun: A French-American computer scientist, Facebook's chief AI scientist and a professor at New York University, USA. He is a student of Geoffrey Hinton.

From left to right: Yann LeCun | Photo: Facebook; Geoffrey Hinton | Photo: Google; Yoshua Bengio | Photo: Botler AI

The 2018 **Turing Award**, known as the "Nobel Prize of computing," has been given to the trio of researchers who laid the foundations for the current boom in artificial intelligence.

is Deep Learning a new concept?

- The history of Deep Learning can be traced back to 1943, when Walter Pitts and Warren McCulloch created a computer model based on the neural networks of the human brain.
- The earliest efforts in developing Deep Learning algorithms came from Alexey Grigoryevich Ivakhnenko (developed the *Group Method of Data Handling*) and Valentin Grigor'evich Lapa (author of *Cybernetics and Forecasting Techniques)* in 1965.

However, the overall progress of Artificial Intelligence got delay during two AI winters. For deep learning even it continued further.

(Details can be found at: https://www.dataversity.net/brief-history-deep-learning/)

➢Why people think it is new?

"There was a dark period between the mid-90s and early-to-mid-2000s when it was impossible to publish research on neural nets, because the community had lost interest in it,"-Yann LeCun.

Why Deep Learning is getting popular now?

- **Building traditional model is expensive:**
  - Designing feature extractor is long, painful and expensive.
  - Industry need more and more models.
  - The process must be automated.
- **Computational Power is Increasing:**

  -**Moore's Law(1970)**: The number of transistors in an integrated circuit would double every two years. In layman's terms, this effectively means that the processing power of computers **will double every two years**.
- **Data Size and Storage Capacity are Increasing:**

  -There are **2.5 quintillion bytes** of **data created** each day at our current pace, but that pace is only accelerating with the growth of the Internet of Things (IoT). Over the last two years alone 90 percent of the **data** in the world was generated.

  (Forbes.com, May21, 2018)

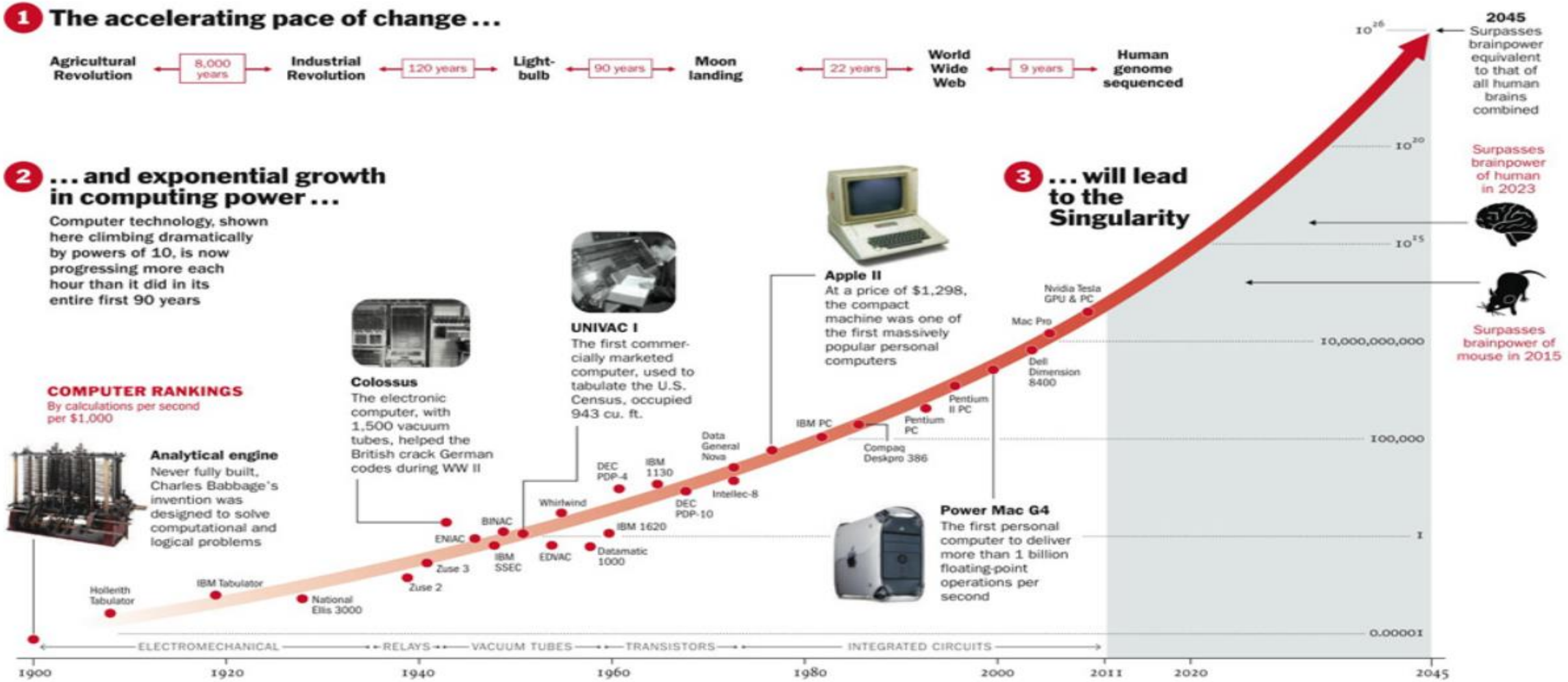
  -The storage cost is also reducing exponentially.

Growth of Computational Power

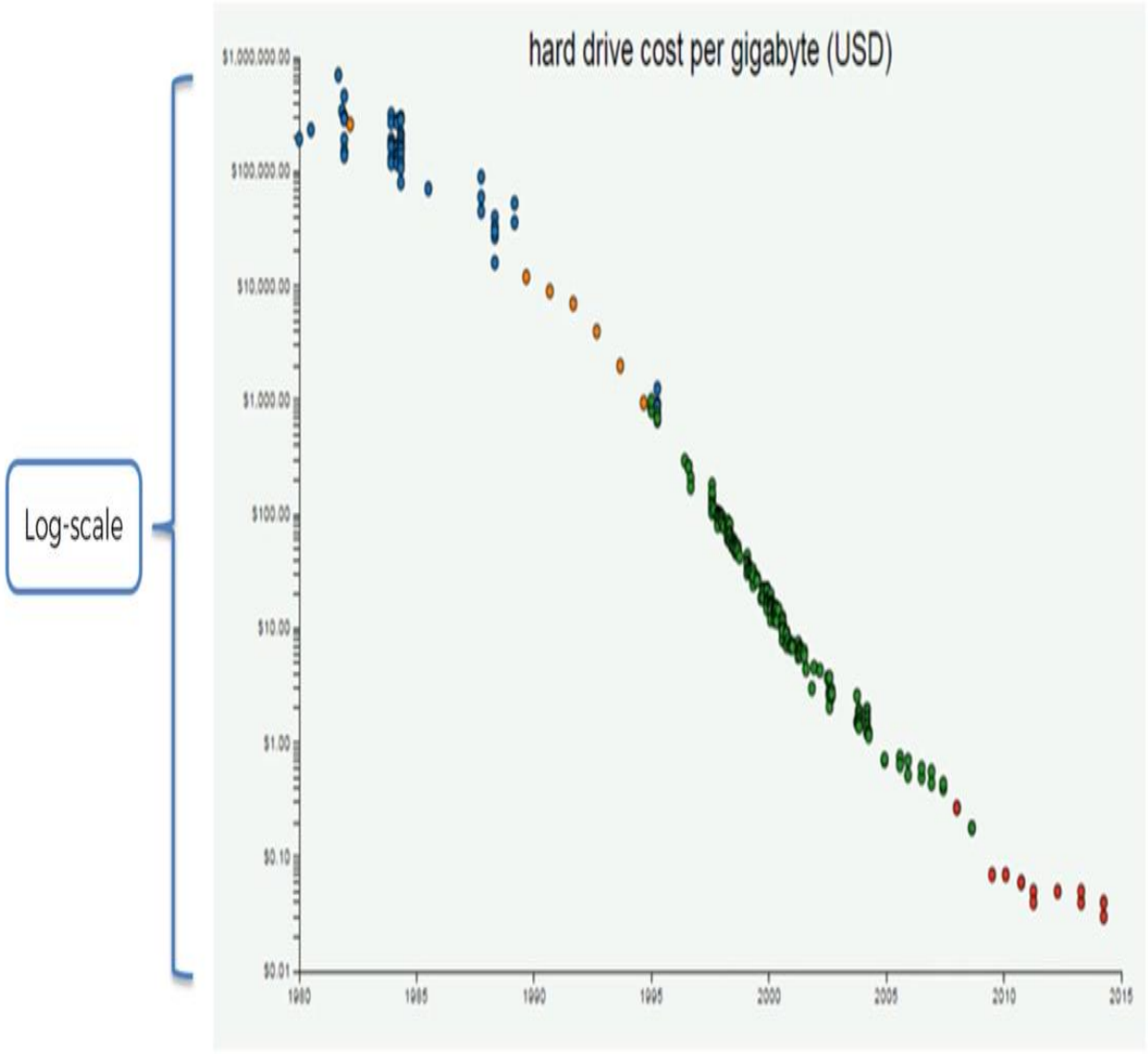
The accelerating pace of change and exponential growth in computing power will lead to the Singularity.

- 2045: The Year Man Becomes Immortal

Image Source: Time Magazine



Storage Capacity and Price



Source: mkomo.com

Image source: SuperDataScience

Neuron (Nervous System)

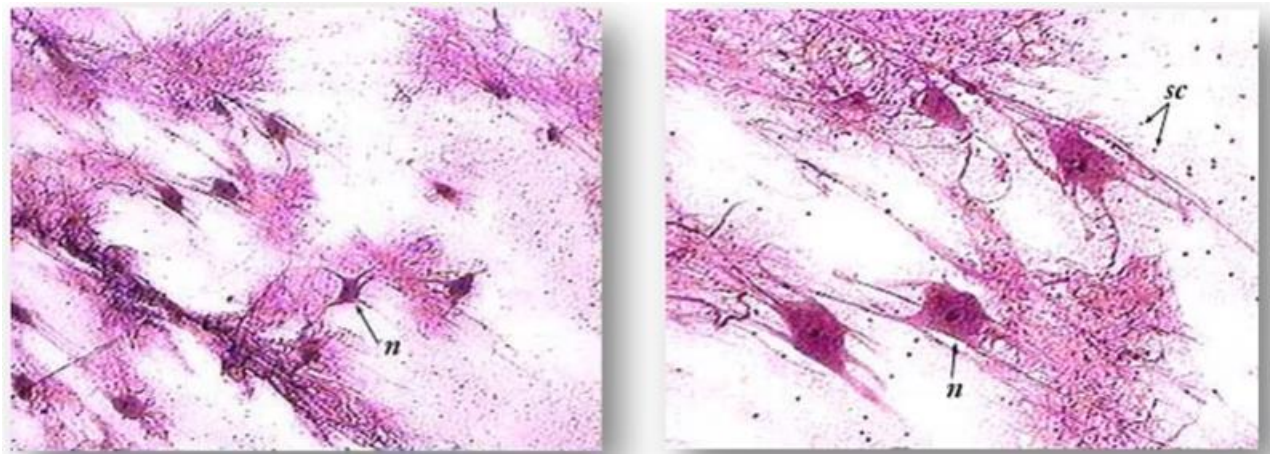
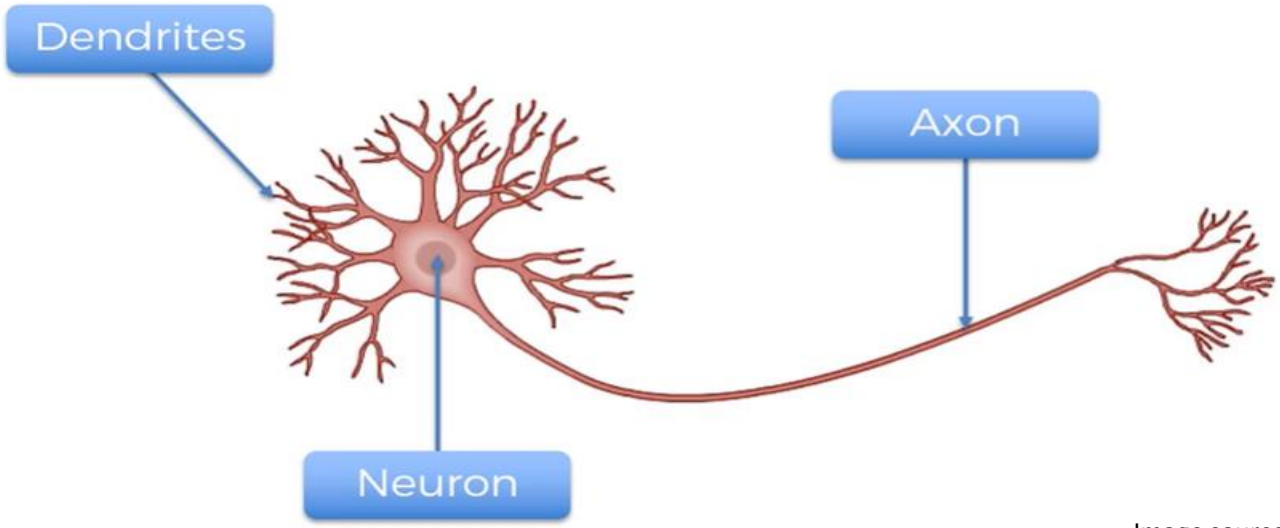


Image source: www.austicc.edu.



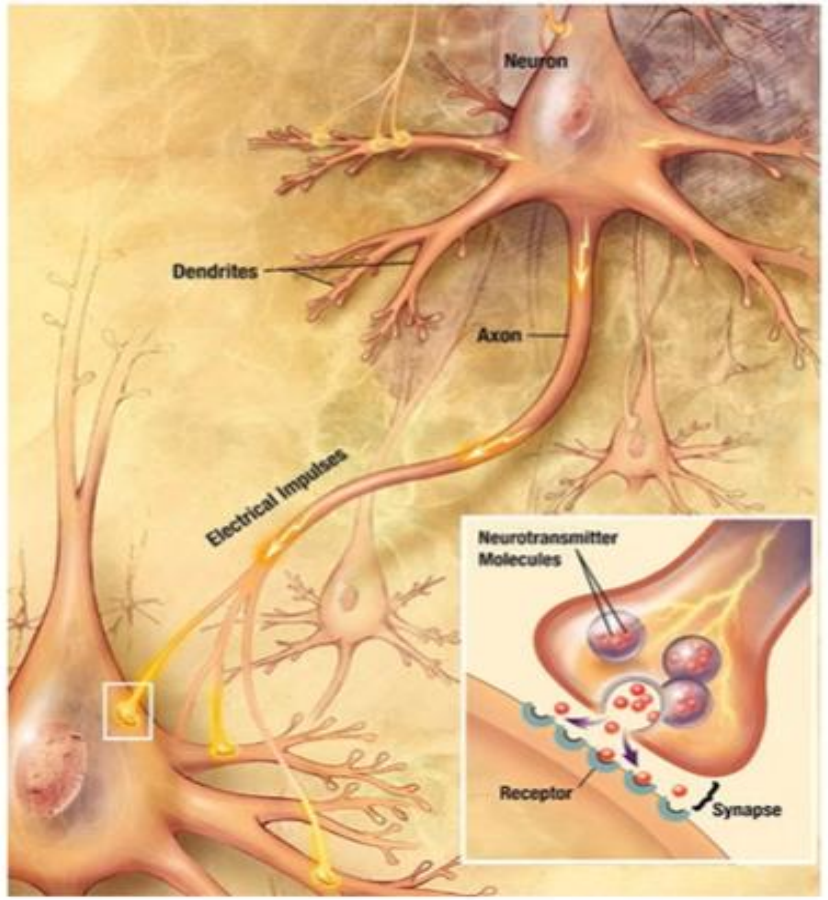
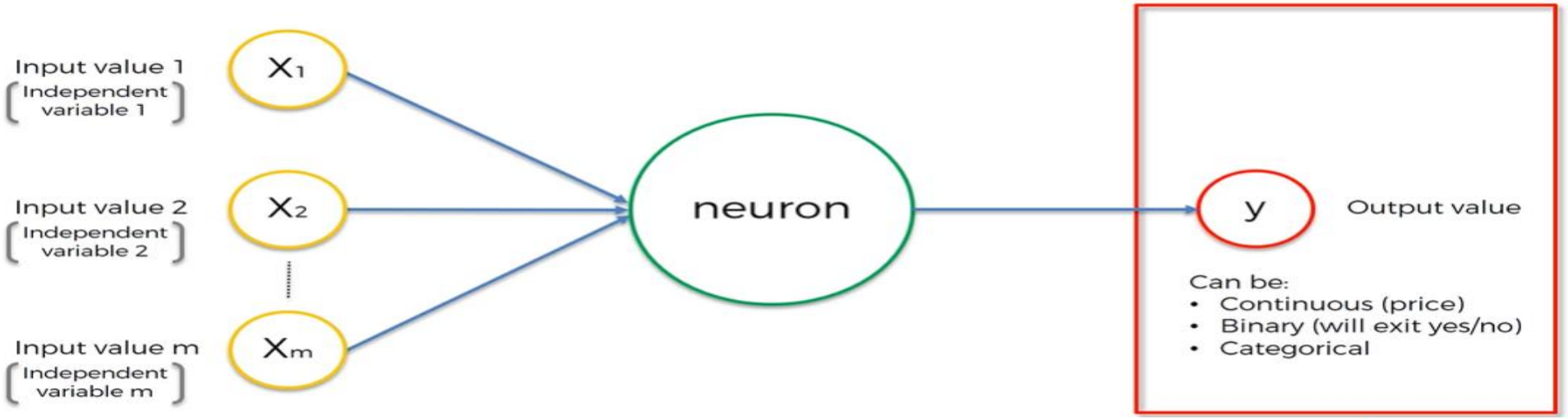


Image source: wikipedia

Neuron (Artificial): also known as Node/Unit in DL

There earliest form of neuron was called perceptron. Professor Frank Rosenblatt introduce the term in 1957. However, earlier perceptron can only handle binary outputs.



Output can have multiple values in case of categorial output (dummy variables) :

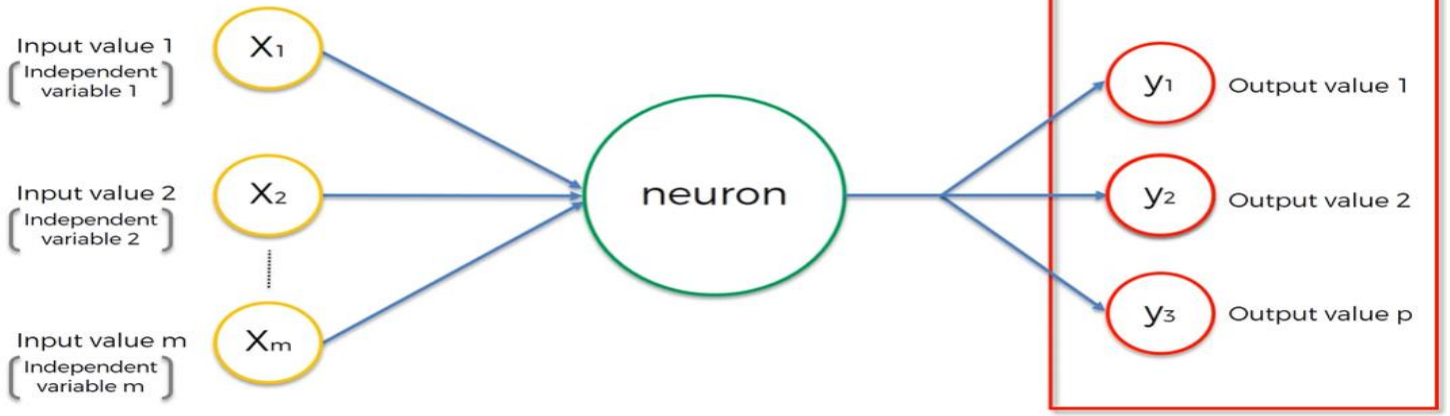
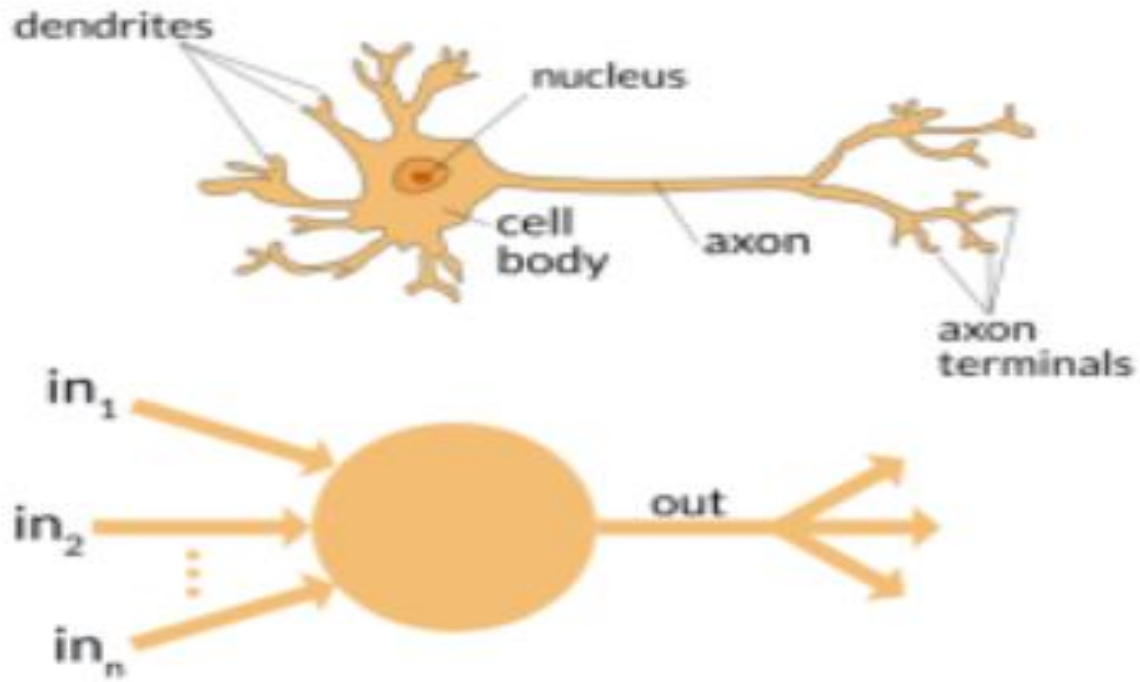


Image source: SuperDataScience

Similarities (Biological Neuron & Artificial Neuron)?



How Biological Neuron produces output?

-It's completely unclear today whether the human brain uses an algorithm.

How an Artificial Neuron produces output?

$$\text{Output} = \text{Activation}\left(\sum_1^n (weight * input)\right) + \text{bias}$$

Since we deal with matrix/vectors, therefore, Output = activation(dot(kernel, input) + bias)

Weight and Bias? (commonly referred to as w and b) are the learnable parameters of a **machine learning** model. (Bias is like the intercept in linear equation)

Table:1

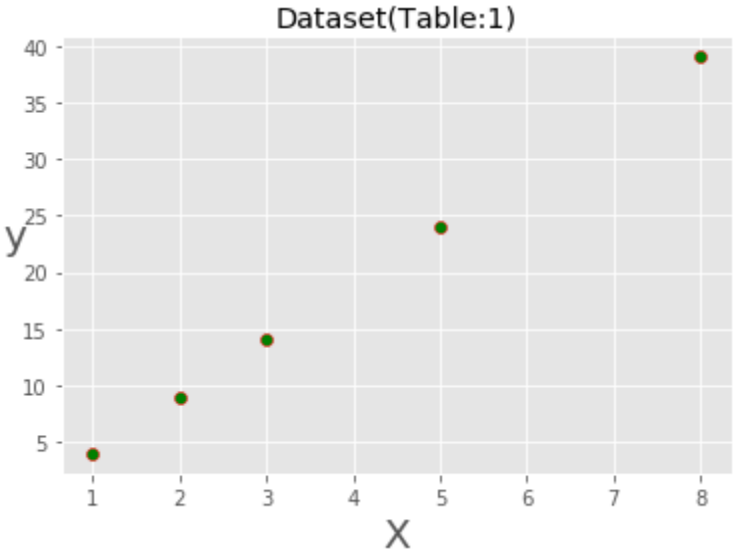
Input(X)	Output(Y)
1	4
2	9
5	24
8	39
3	14

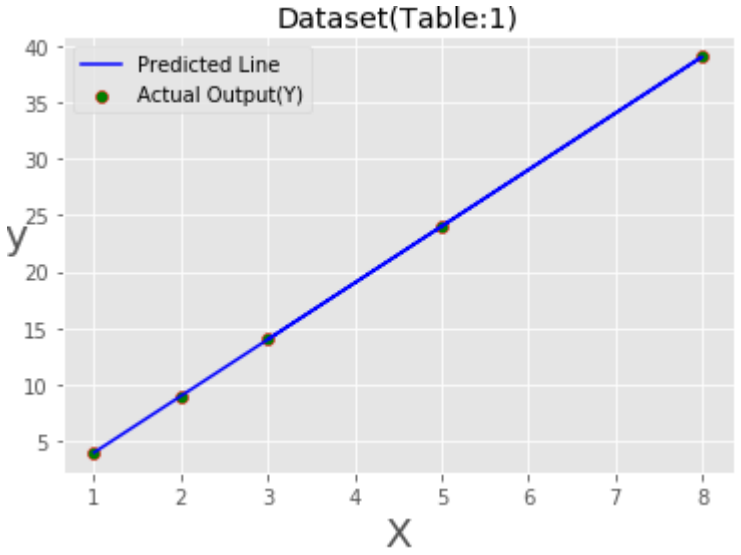
Table:2

Input(X)		Output(Y)
x1	**x2**	**Y**
2	1	4
3	4	8
1	5	7
3	2	6

Output = ~~Activation~~ ($\sum_1^n(weight * input)$ +bias)

Let's Calculate Weight and Bias for given tables?
For Table 1: Output (prediction), y = w*x+b
w = ?, b = ?





For Table 2: y = w1*x1+w2*x2 +b
w1 = ?, w2 = ?, b = ?

Let's look at another example:

X	Y
1	4
2	14
5	24
8	28
3	18

X	Y	$\hat{Y}$
1	4	10.7
2	14	13.5
5	24	21.7
8	28	29.9
3	18	16.2

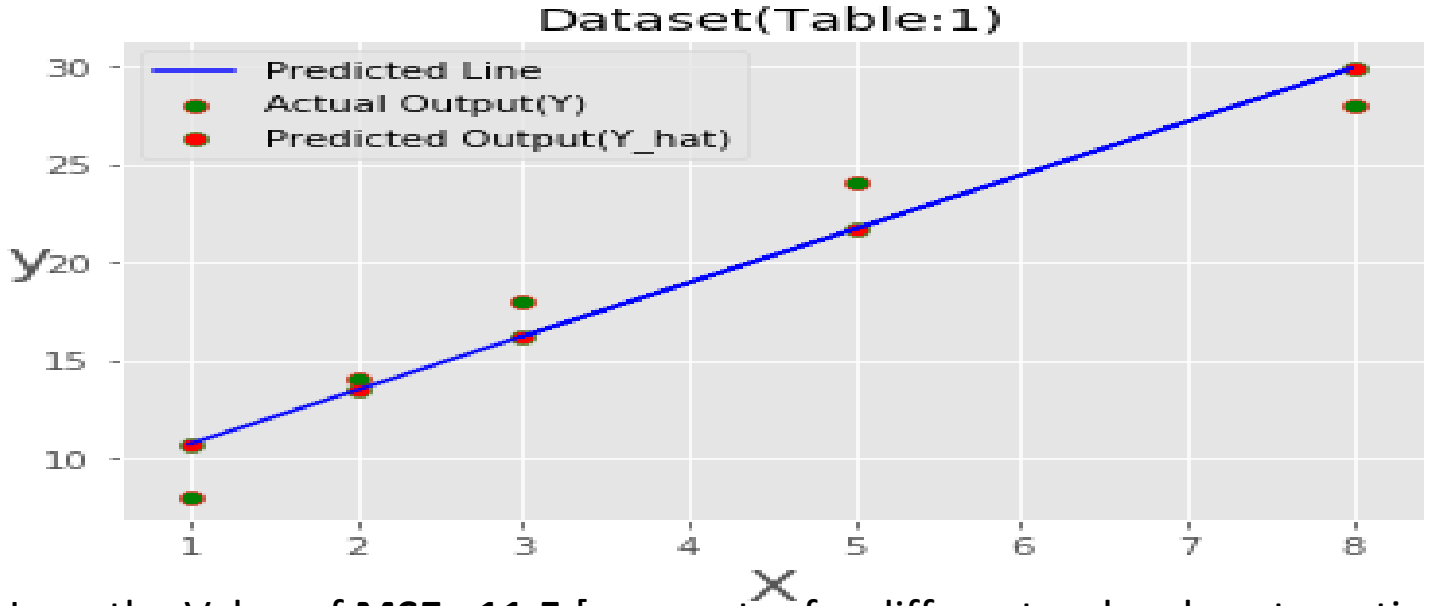
Output = ~~**Activation**~~ **(**$\sum_1^n(weight * input)$**+bias)**

For given data : **y_hat (prediction) = w*x+b**

w = ?, b = ?

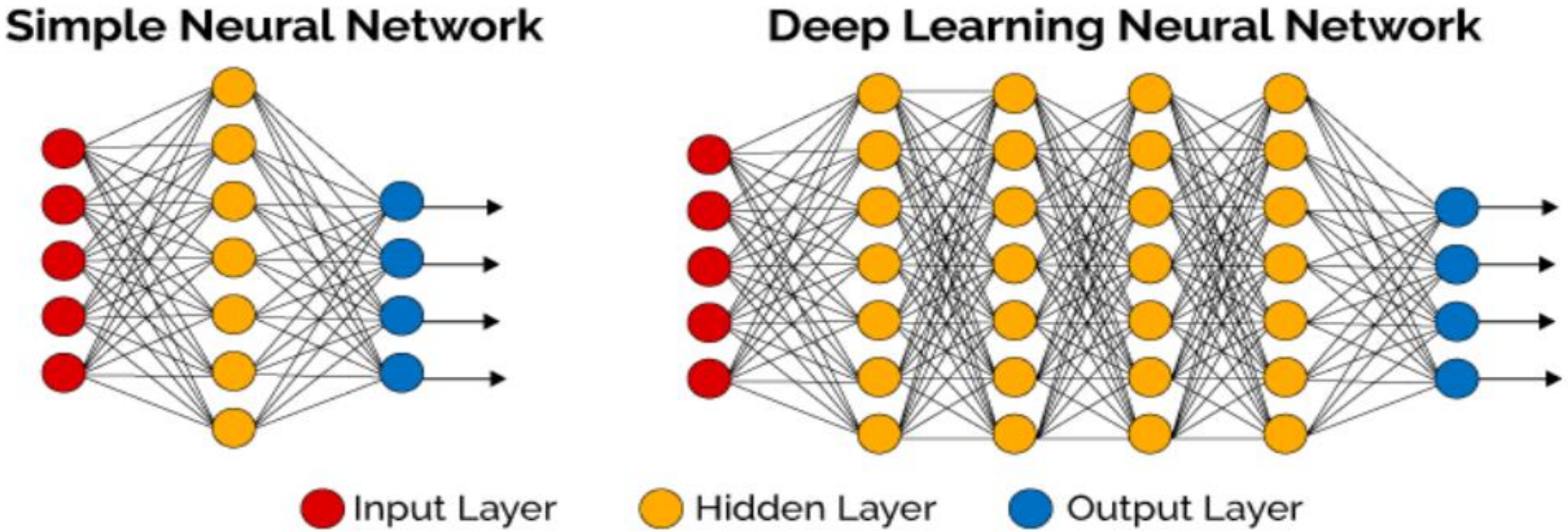
Not easy to predict the exact output?

Lets consider, **w = 2.74, b= 7.99**. However, we can minimize the error and adjust the value of w and b to get best fitting line using loss functions, a simple loss function $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$



Here, the Value of **MSE =11.5** [we can try for different w, b values to optimize]

Why do we call it Deep Learning?



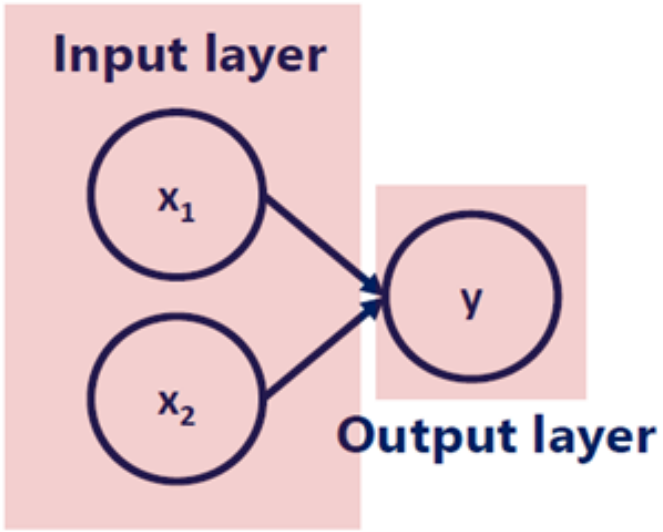
- Row = Depth of Network [The number of hidden layers in a network]
- Column = Width of the Layer. [The number of units(nodes) in a layer is the width of the layer]
- The **width** of the net is the number of units of the biggest layer.
- Neural Networks also known as multilayer perceptron.

Image source: www.kdnuggets.com.

Layers

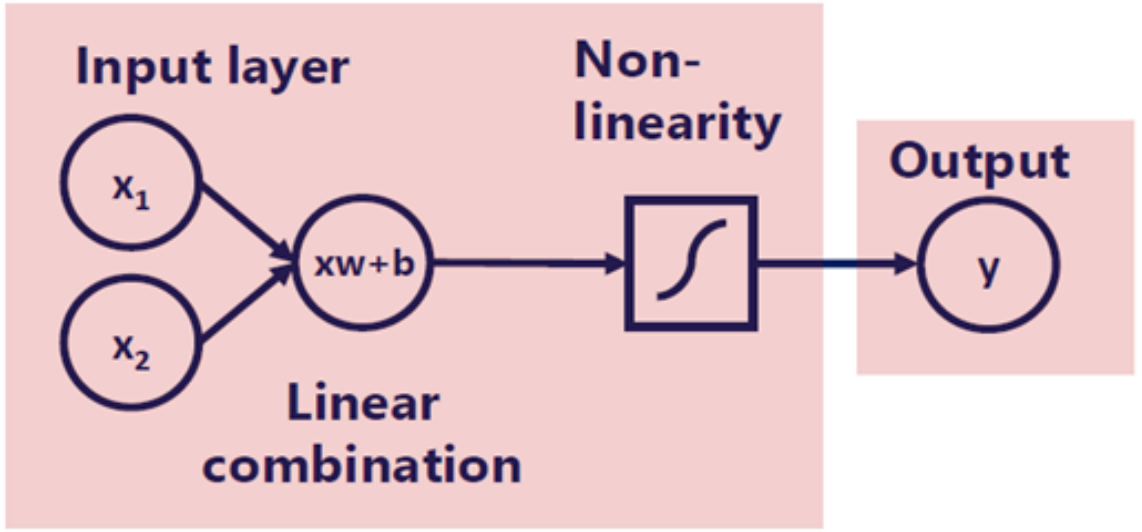
An initial linear combination and the added non-linearity form a **layer**. The layer is the building block of neural networks.

Minimal example (a simple neural network)



In the minimal example we trained a *neural network* which had no depth. There were solely an input layer and an output layer. Moreover, the output was simply **a linear combination** of the input.

Neural networks



Neural networks step on linear combinations, but add a non-linearity to each one of them. Mixing linear combinations and non-linearities allows us to model arbitrary functions.

Image source: 365DataScience

A Deep Neural Network.

This is a deep neural network (deep net) with 5 layers.

How to read this diagram:

A layer

A unit (a neuron)

Arrows represent mathematical transformations

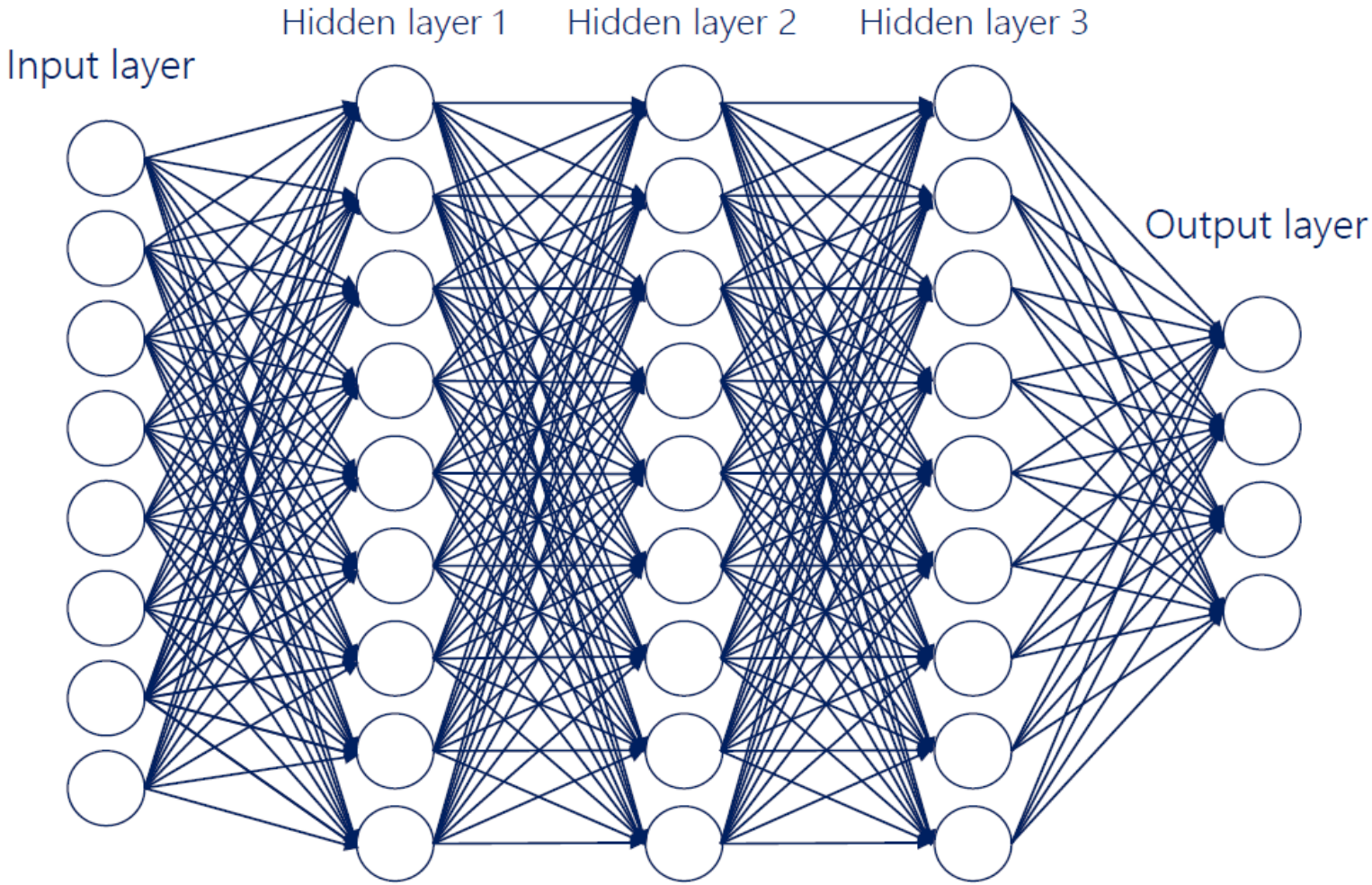


Image source: 365DataScience

How do Neural Networks work?

Example: Perceptron Learning (single input) uses threshold activation function.

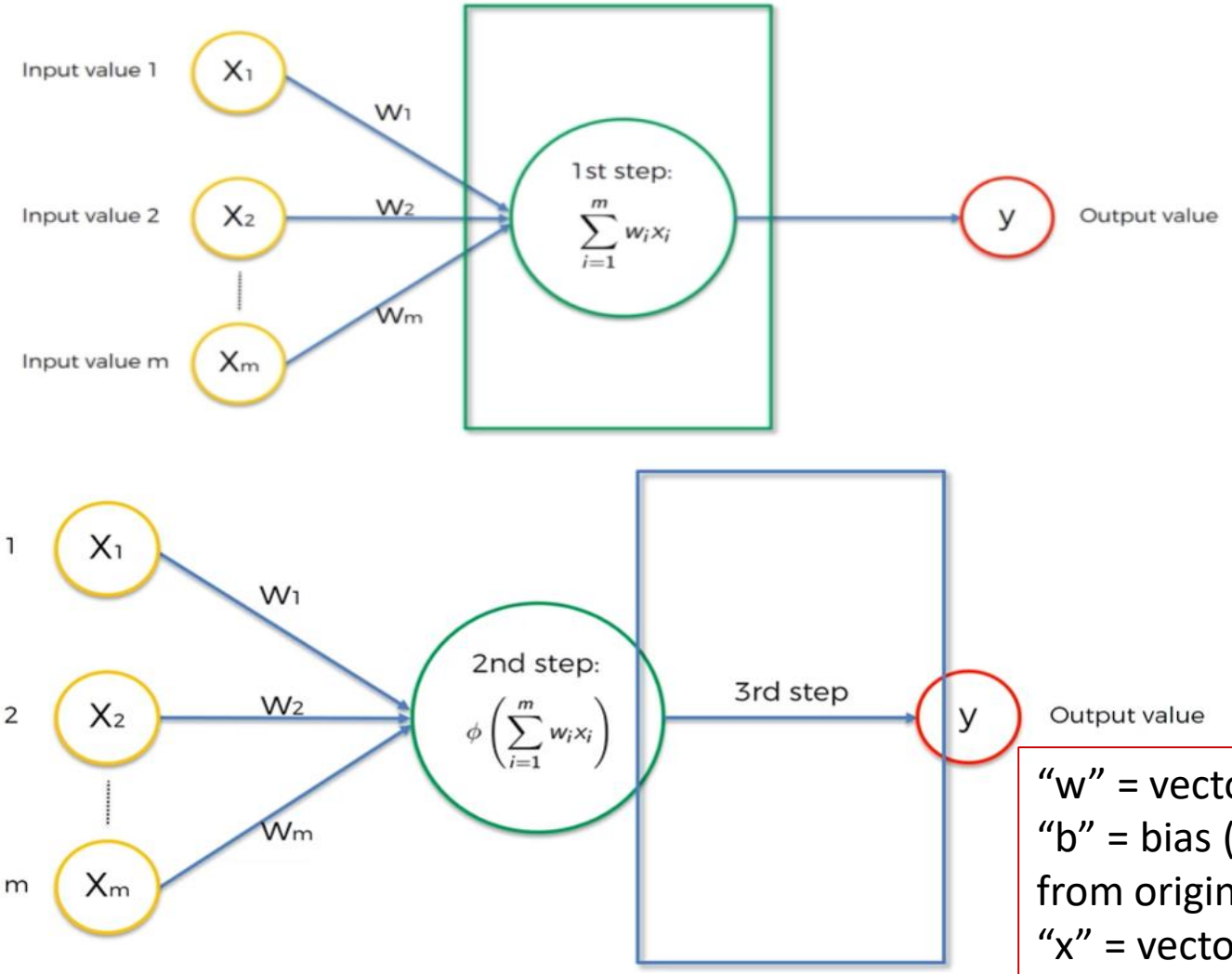
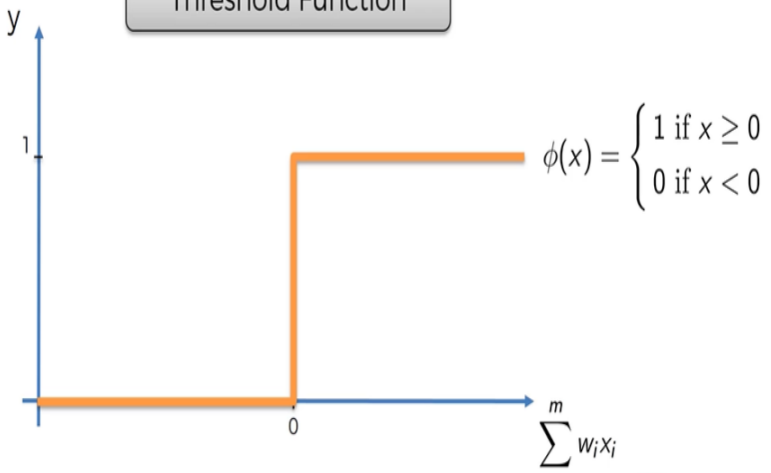


Image source: SuperDataScience

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Threshold Function



$$w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

"w" = vector of real-valued weights
"b" = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
"x" = vector of input x values
"m" = number of inputs to the Perceptron

How do Neural Networks work?

The basic process carried out by a neuron in a neural network is:

Initialization : Randomly initialized weights to small number close to 0 (but not 0). Then, Input first observation in your dataset in the input layer, each feature in one input node.

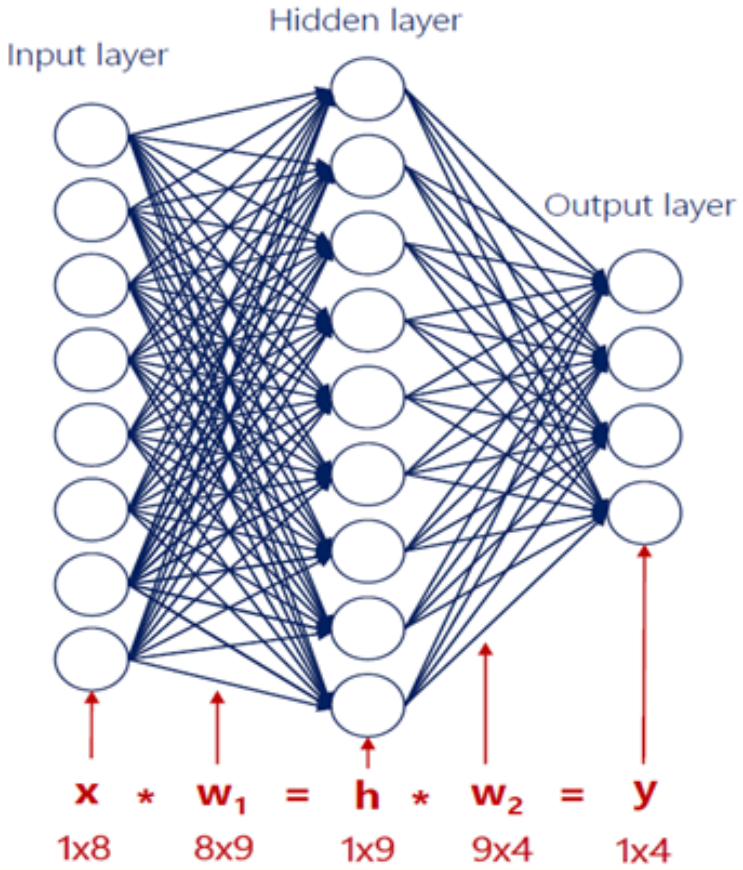
* This is just the number 1, making it possible to represent activation functions that do not cross the origin. Biases are also assigned a weight.

Image source: www.missinglink.ai.

Why Non-Linearity? To Stack Layers

Non-linearities | Stacking Layers | Depth | Deep Learning

You can see a net with no non-linearities: just linear combinations.



$$h = x * w_1$$

$$y = h * w_2$$

$$y = x * \underbrace{w_1 * w_2}_{8x9 \quad 9x4}$$

$$y = x * w^{\star}$$

8x4

Two consecutive linear transformations are equivalent to a single one.

Two consecutive linear transformations are equivalent to a single one.

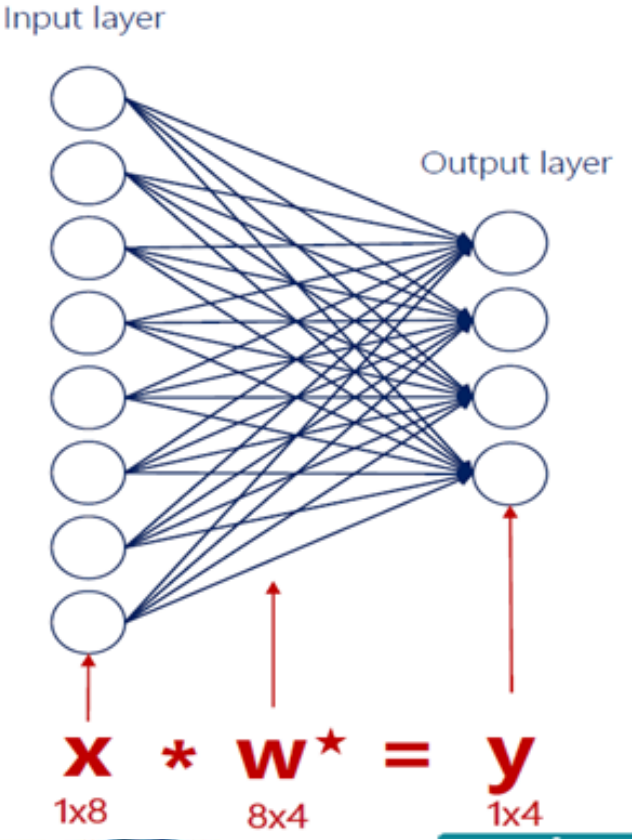
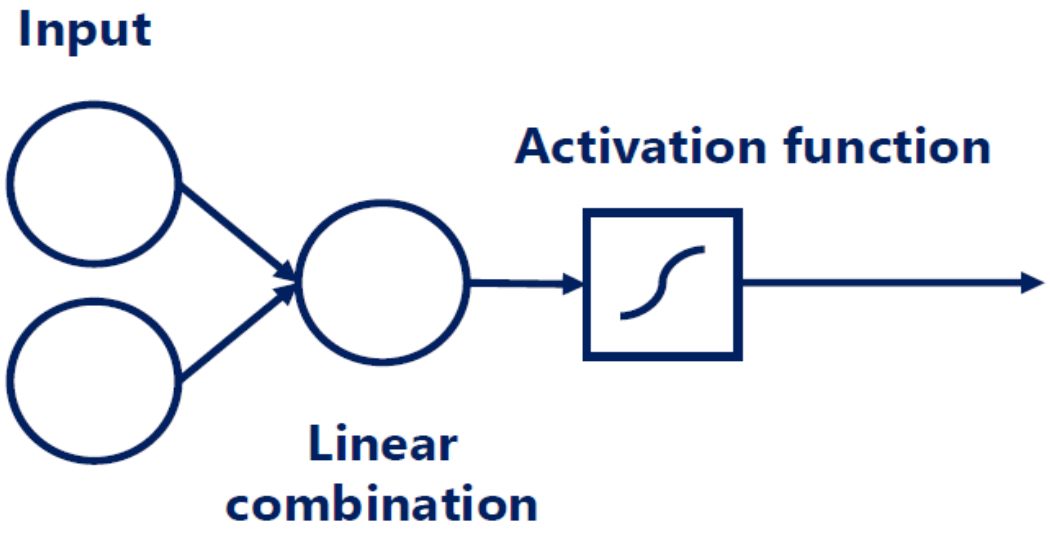


Image source: 365DataScience

Activation Functions (Non-linearities): also called Transfer Function



Activation functions (non-linearities) are needed so we can break the linearity and represent more complicated relationships.

Moreover, activation functions are required in order to **stack layers.**

Activation functions transform inputs into outputs of a different kind.

Non-Linearity does not change the shape of the expression, it only changes its linearity.

The temperature starts decreasing (which is a numerical change). Our brain is a kind of an 'activation function'. It tells us whether it is **cold enough** for us to put on a jacket.

Putting on a jacket is a binary action: 0 (no jacket) or 1 (jacket).

This is a very intuitive and visual (yet not so practical) example of how activation functions work.

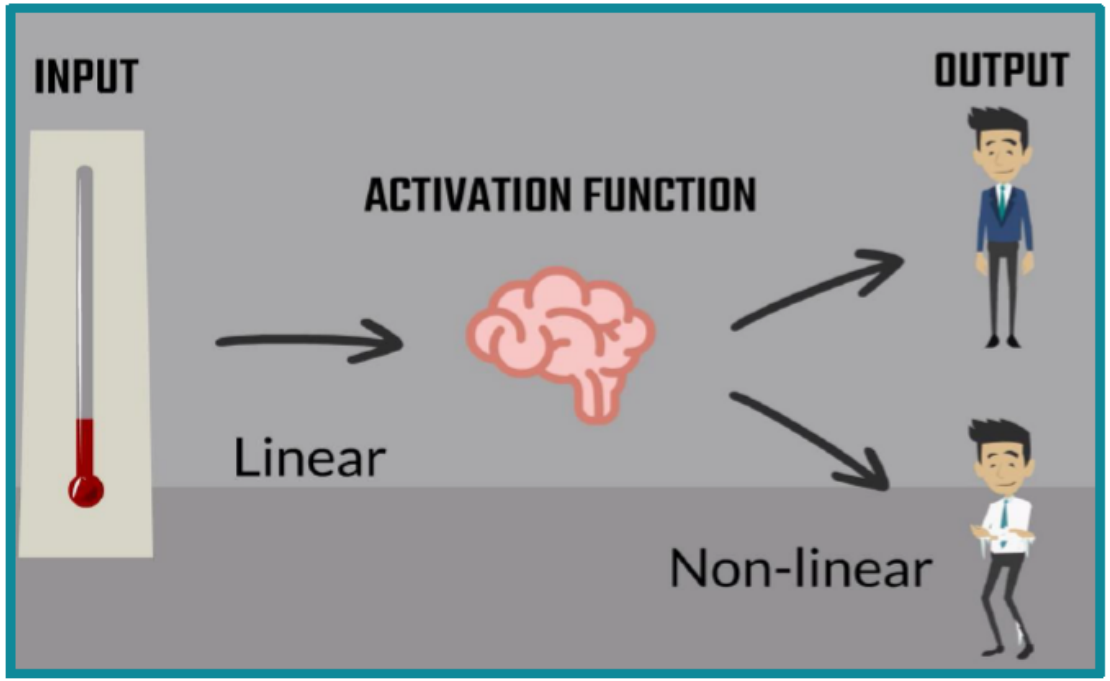
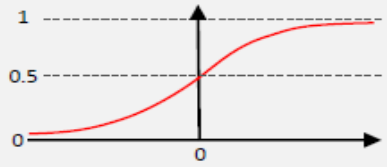


Image source: 365DataScience

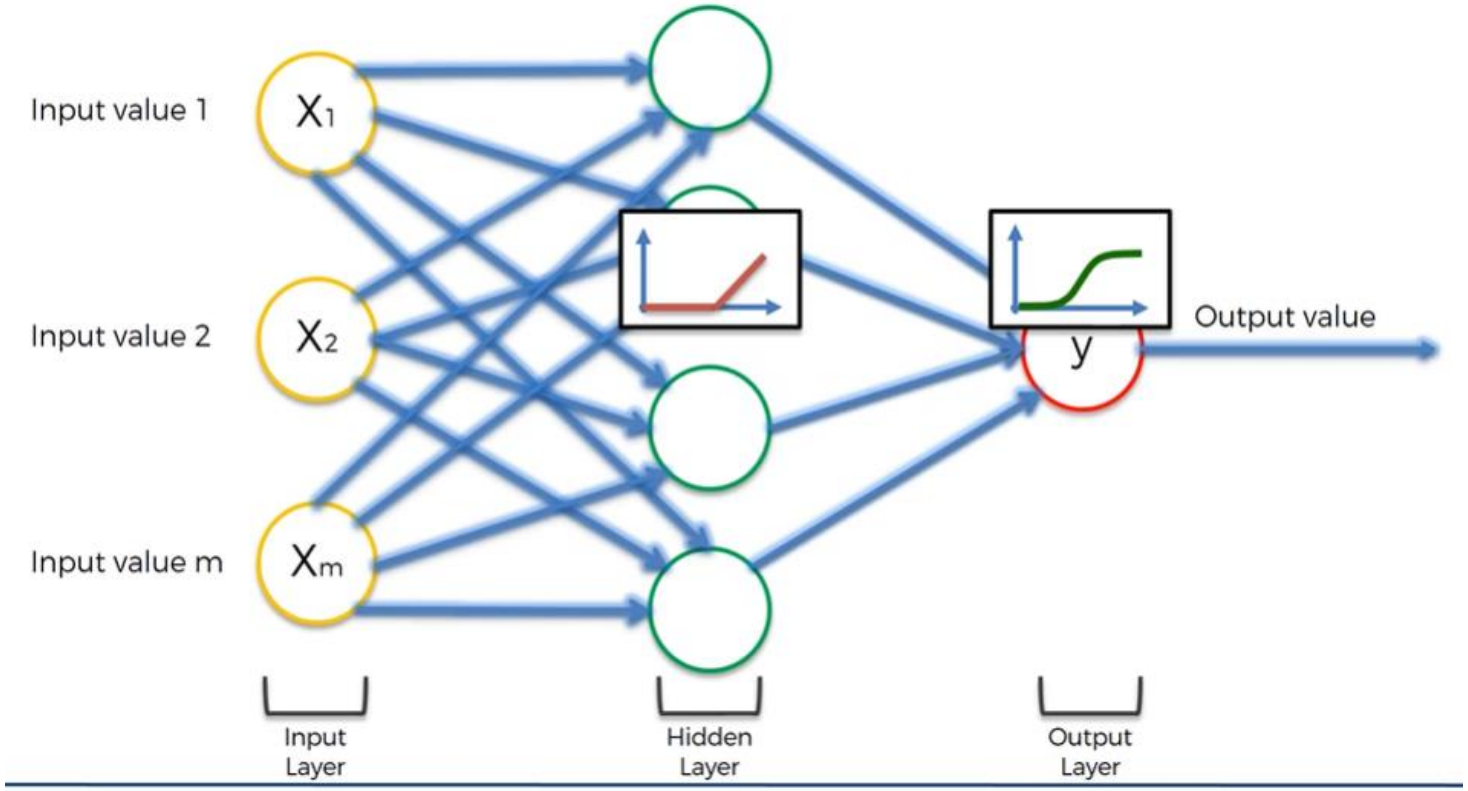
Some commonly used activation functions:

Name	Formula	Derivative	Graph	Range
sigmoid (logistic function)	$\sigma(a) = \frac{1}{1+e^{-a}}$	$\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$		(0,1)
TanH (hyperbolic tangent)	$\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$	$\frac{\partial \tanh(a)}{\partial a} = \frac{4}{(e^{a}+e^{-a})^2}$		(-1,1)
ReLu (rectified linear unit)	relu(a) = max(0,a)	$\frac{\partial \text{ relu}(a)}{\partial a} = \begin{cases} 0, if\ a \le 0 \\ 1, if\ a > 0 \end{cases}$		(0,∞)
softmax	$\sigma_i(\boldsymbol{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$	$\frac{\partial \sigma_i(\boldsymbol{a})}{\partial a_j} = \sigma_i(\boldsymbol{a})\left(\delta_{ij} - \sigma_j(\boldsymbol{a})\right)$ Where δ_{ij} is 1 if i=j, 0 otherwise	different every time	(0,1)

All common activation functions are: **monotonic, continuous,** and **differentiable**. These are important properties needed for the optimization.

Image source: 365DataScience

Common combination of activation functions:

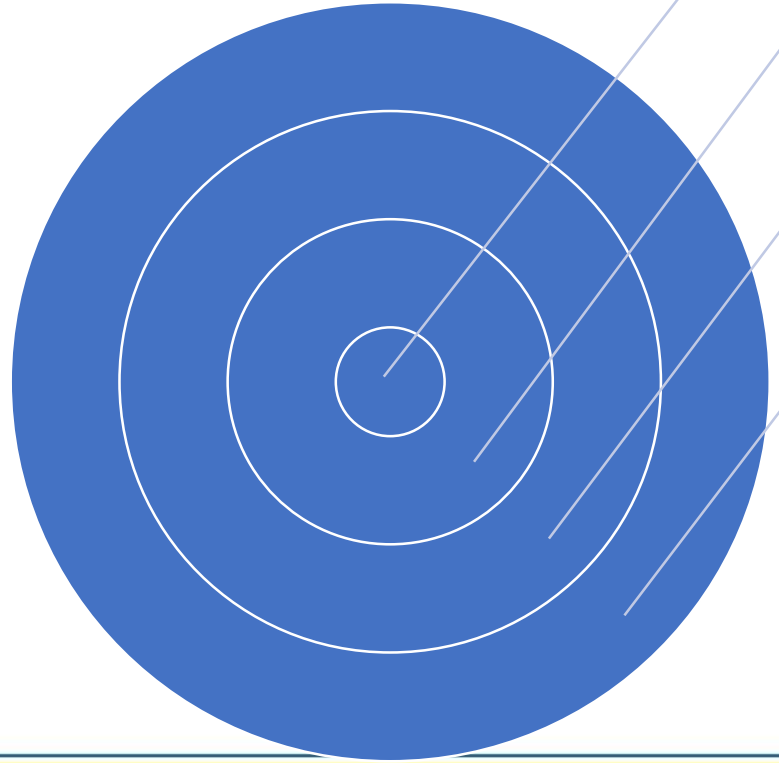


Summary:

- Activation functions are non-linear and differentiable.
- Differentiable for back propagation (We will discuss later.)
- Non-linear to approximate complex function. (We've already discussed.)

Image source: SuperDataScience

Deep Neural Network : Involves 4 Ingredients



Data

Model

Objective Function

Optimization Algorithm

Need to prepare a certain amount of data to train with.

DATA **MODEL** **OBJECTIVE FUNCTION** **OPTIMIZATION ALGORITHM**

Weather historical data

Location	Year	Month	Date	Temp	Dew point	Pressure	Visibility	Wind speed	Max wind speed	Max wind gust	Max temp	Min temp	Precipitation
723150	###		1	44.7	40.3	1022.00	6	3.2	15	20	64	30.9	0.00D
723150	###	1	2	55.5	50	1022.10	7.2	6.6	15.9	21	66.0*	48.2*	0.00D
723150	###	1	3	58.1	53.6	1019.70	9.5	7.2	15	20	66.9	48	0.00D
723150	###	1	4	57.7	50.8	1012.30	8.2	10.4	18.1	23.9	62.1*	46.4*	0.06G
723150	###	1	5	34.9	23.8	1021.60	10	10.2	17.1	27	46.0*	28.0*	0.14G
723150	###	1	6	31.6	22.1	1030.60	9.6	2.9	8.9	999.9	48	19	0.00G
723150	###	1	7	36.6	24.7	1028.50	9.9	8.2	17.1	23.9	50.0*	27.0*	0.00D
723150	###	1	8	35.7	23.5	1027.50	9.9	3.4	8.9	999.9	50	24.1	0.00C
723150	###	1	9	42.5	30.9	1018.90	6.6	4.2	8.9	999.9	50	24.1	0.44D
723150	###	1	10	52.8	34.1	1008.50	5.7	6.5	17.1	27	60.8*	46.4*	1.42G
723150	###	1	11	46.5	24.1	1013.20	10	6.5	25.1	33	64.9	28.9	0.03G
723150	###	1	12	44.7	24.9	1021.30	9.9	4.4	11.1	999.9	66	28	0.00G
723150	###	1	13	53.7	35	1017.60	9.9	10.6	24.1	34	66	28	0.08D
723150	###	1	14	31.5	12.7	1032.70	9.9	16.4	28.9	37.9	41.0*	24.1*	0.00H
723150	###	1	15	29.4	13.3	1037.00	9.9	3.3	8	999.9	50	15.1	0.00I
723150	###	1	16	38.6	23.9	1028.70	8.8	5.2	10.1	999.9	50	15.1	0.00H
723150	###	1	17	44.3	24.9	1026.30	9.6	6.4	8.9	15	52	27	0.03A
723150	###	1	18	35	25.8	1018.80	7.1	8.6	19	23.9	52	30	0.21G
723150	###	1	19	36.1	25.9	1015.60	9.9	12.2	20	27	45	30	0.01G
723150	###	1	20	35.4	27.7	1005.60	8	17.7	34	42.9	42.1*	26.6*	0.14G
723150	###	1	21	21.8	6.8	1016.30	9.9	16.2	25.1	32.1	39.9	14	0.01G
723150	###	1	22	21.6	11.5	1021.30	7.4	7.2	14	999.9	28.9	14	0.00G
723150	###	1	23	29.6	27.3	1016.30	5.9	4.3	8.9	999.9	36	18	0.03B

Image source: 365DataScience

Model will find some coefficients(weights) to determine the relationship between input data and output value.

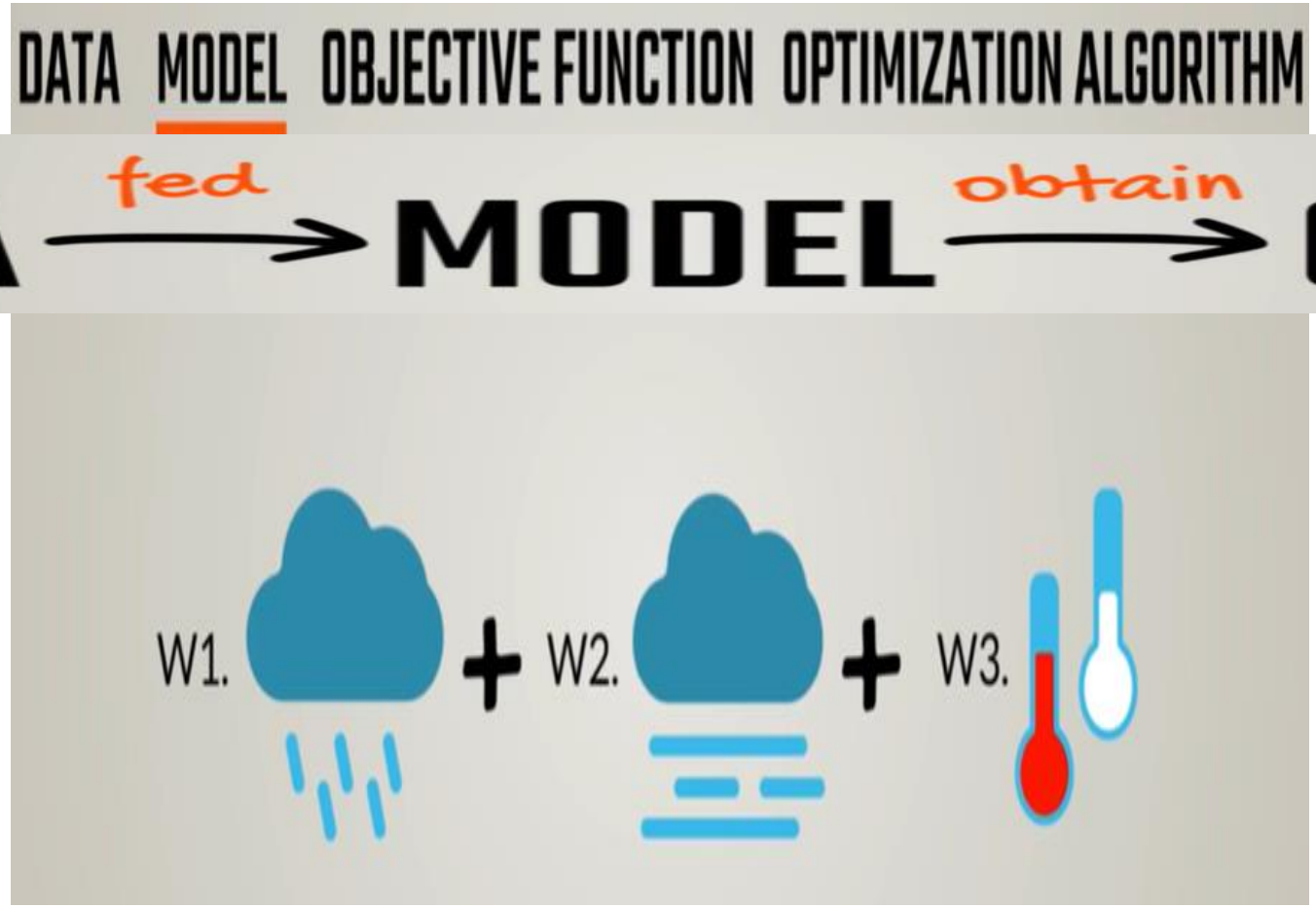
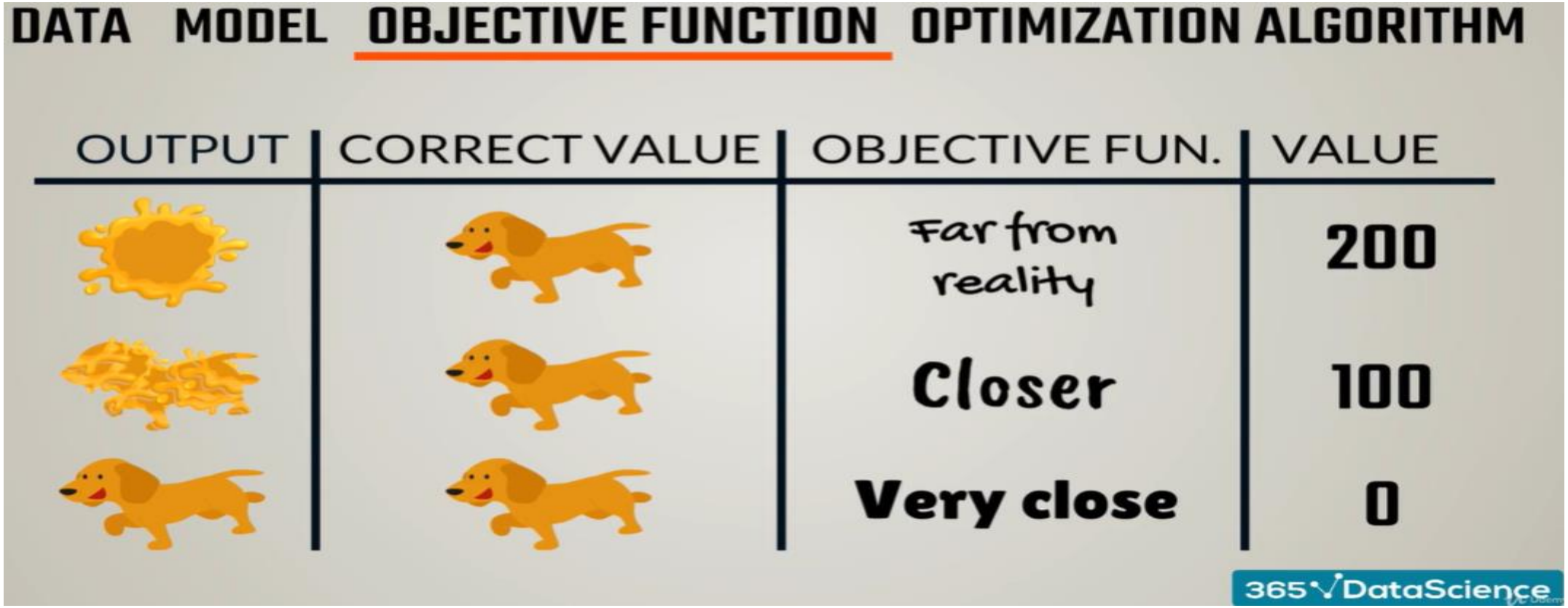


Image source: 365DataScience

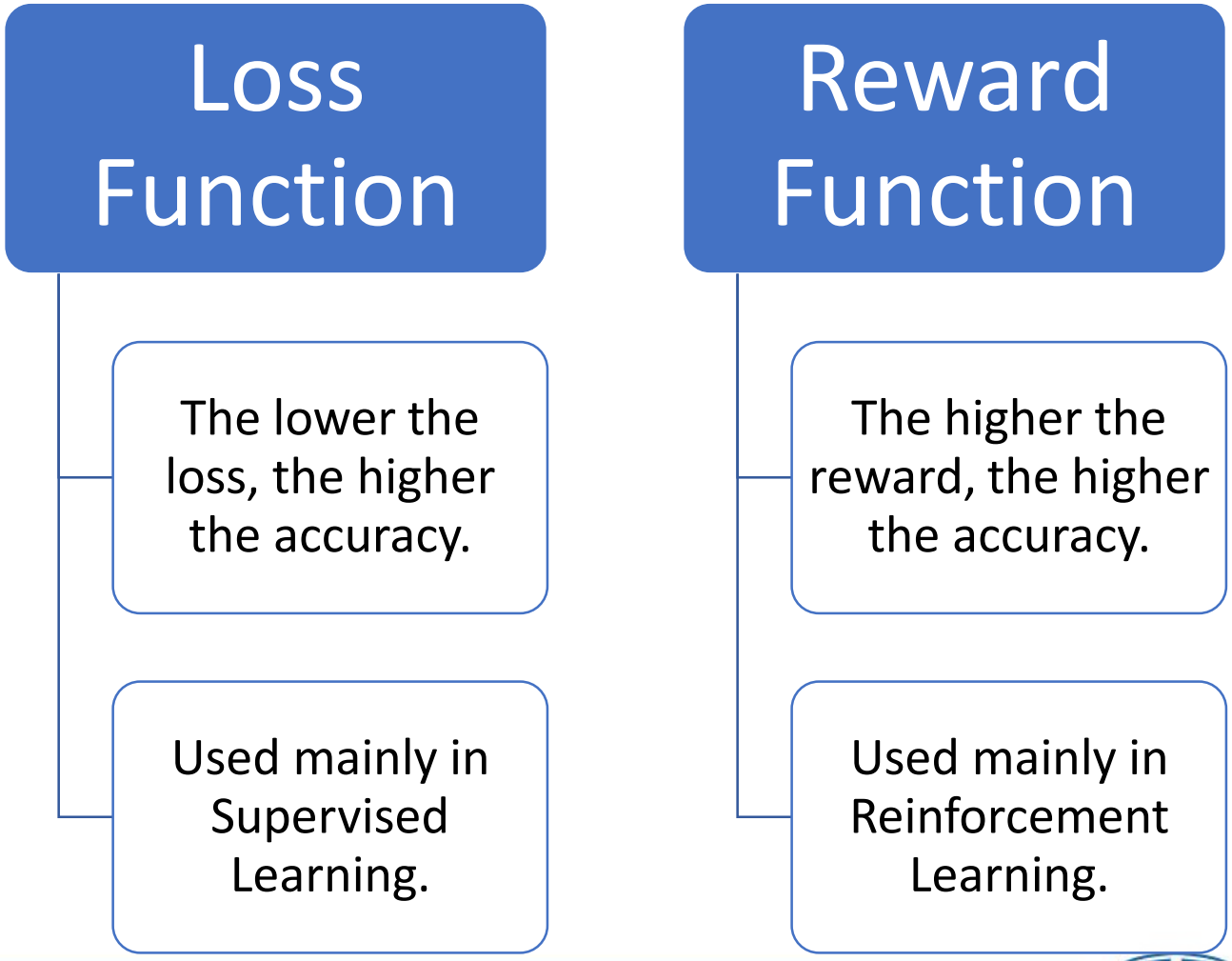
Objective function estimates how correct the model's output are on average:(Predicted vs. Actual)



Minimizing Error = Minimizing/Maximizing Objective Function.

Image source: 365DataScience

Objective Function are mainly 2 types:



Loss Function/Cost Function/Error Function calculates model's errors.

Measures the difference between our model's predictions and actual outcome that we want to predict.

Maximum Likelihood Estimation (MLE) is a framework for inference for finding the best statistical estimates of parameters from training data.

Two most commonly used loss functions when training Neural Network models used under MLE framework:

- Mean Squared Error (MSE)
  [Suitable for Regression]
- Cross-Entropy
  (Also known as Log loss/ Logistic Loss/Logarithmic Loss)
  [Suitable for Classification]

Commonly used combination of Activation Function and Loss Function:

Regression Problem:

- **Output Layer Configuration**: One node with a linear activation unit.
- **Loss Function**: Mean Squared Error (MSE), L-2 Norm(Euclidian Distance).

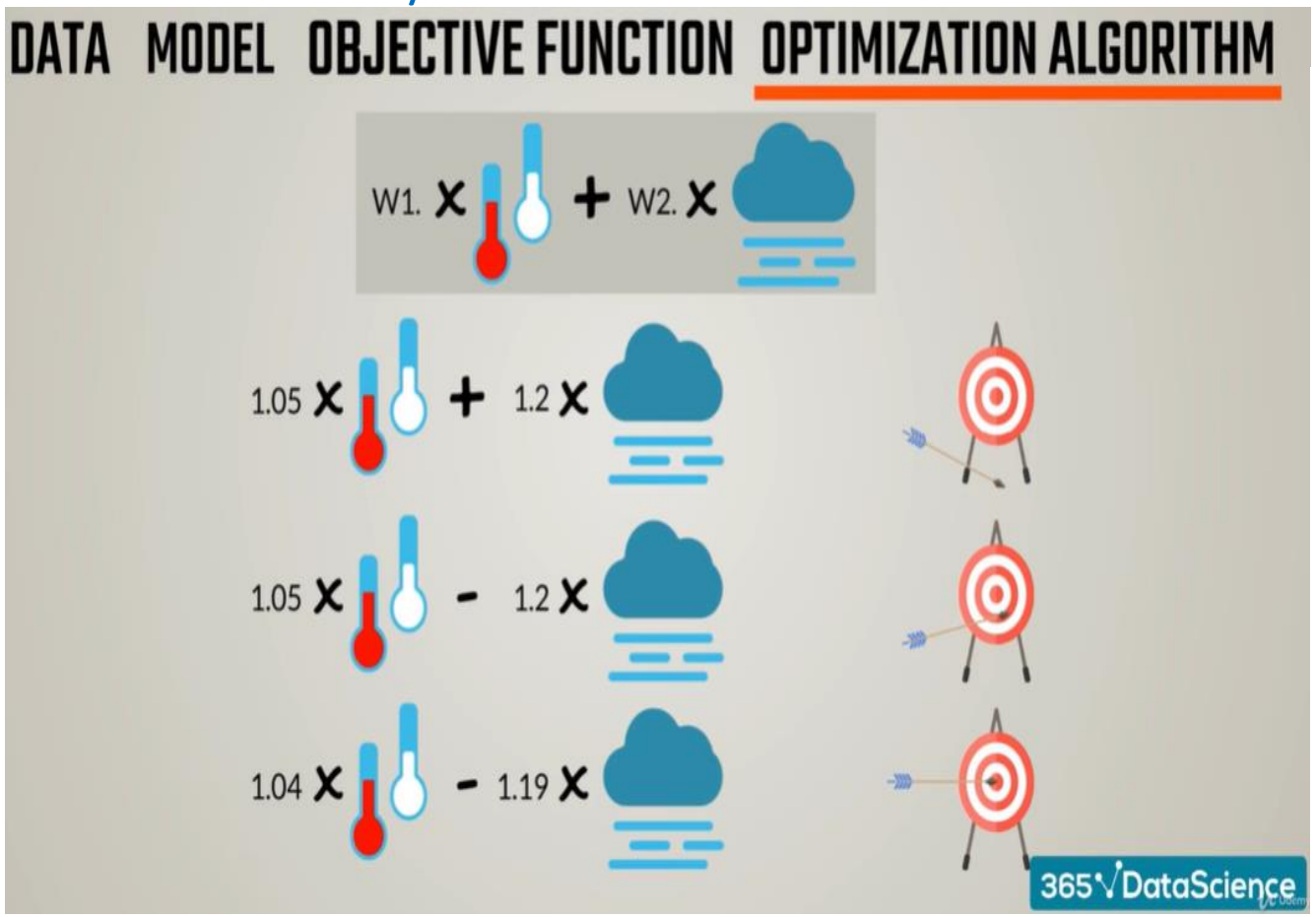
Binary Classification Problem:

- **Output Layer Configuration**: One node with a sigmoid activation unit.
- **Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

Multi-Class Classification Problem:

- **Output Layer Configuration**: One node for each class using the softmax activation function.
- **Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

Optimization Algorithm varies the model's parameters, i.e., weights and bias to optimize the objective function, i.e., loss function/reward function.



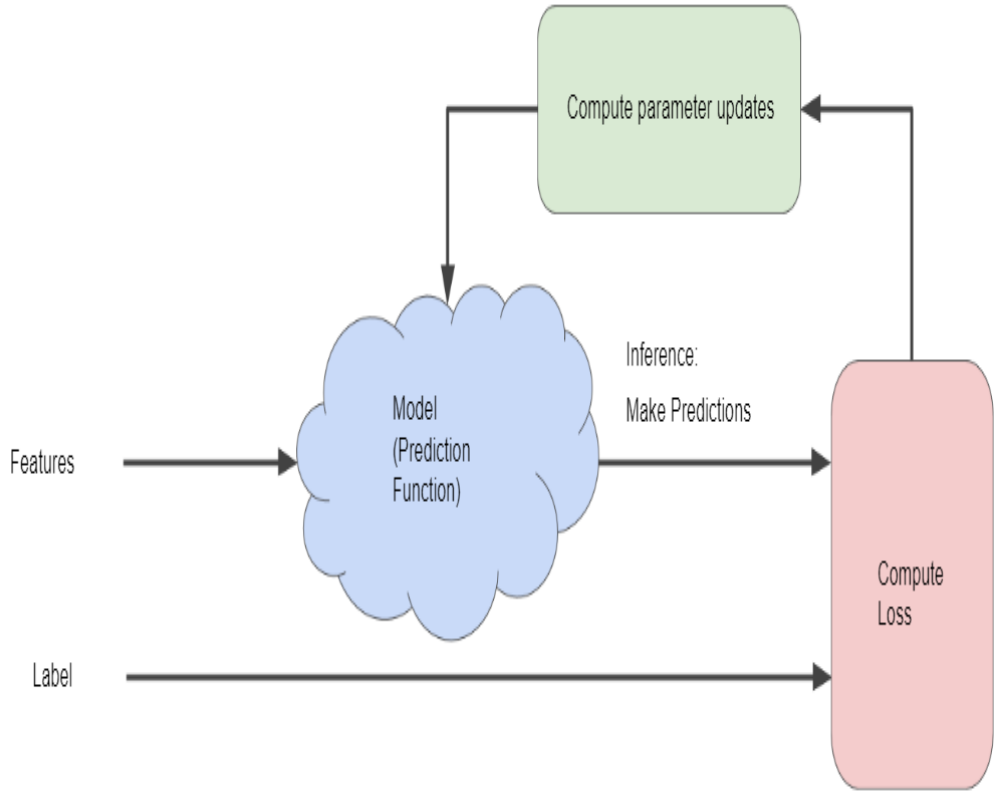
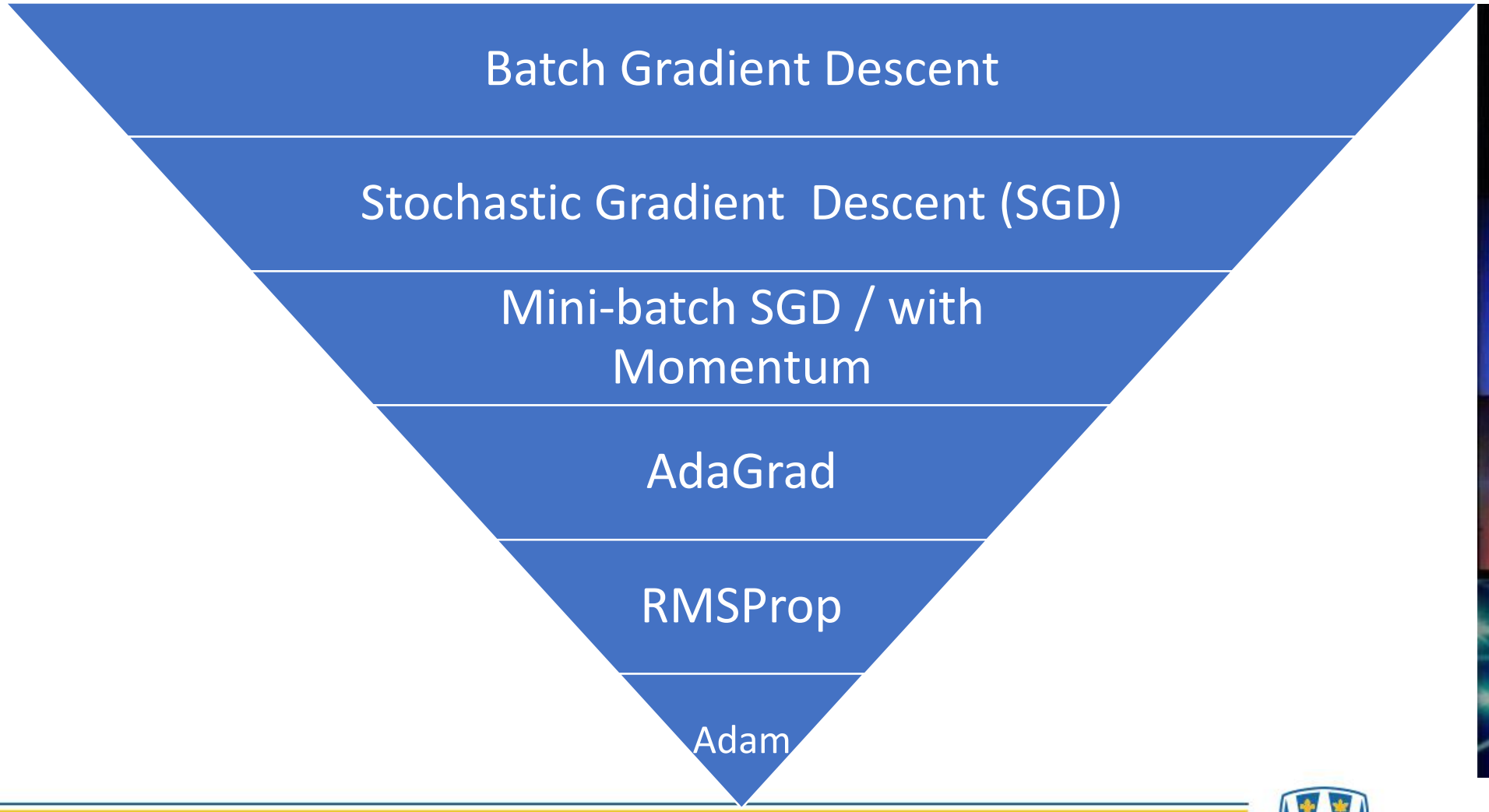
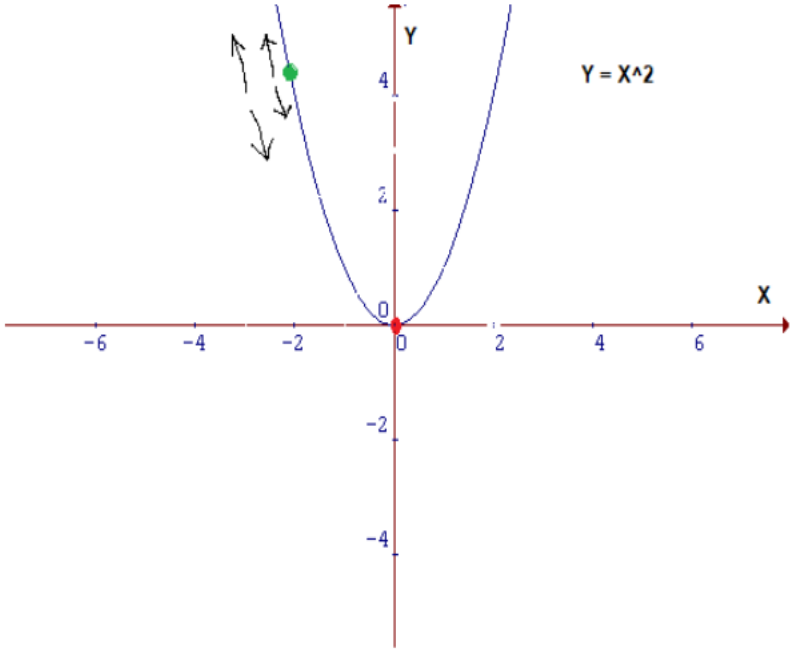


Image source: 365DataScience

Some commonly used Optimization Algorithms:



Optimization Algorithms help taking two crucial decisions:



1. Which way to go:

- Upward or Downward?

2. How you take the step to reach the destination:

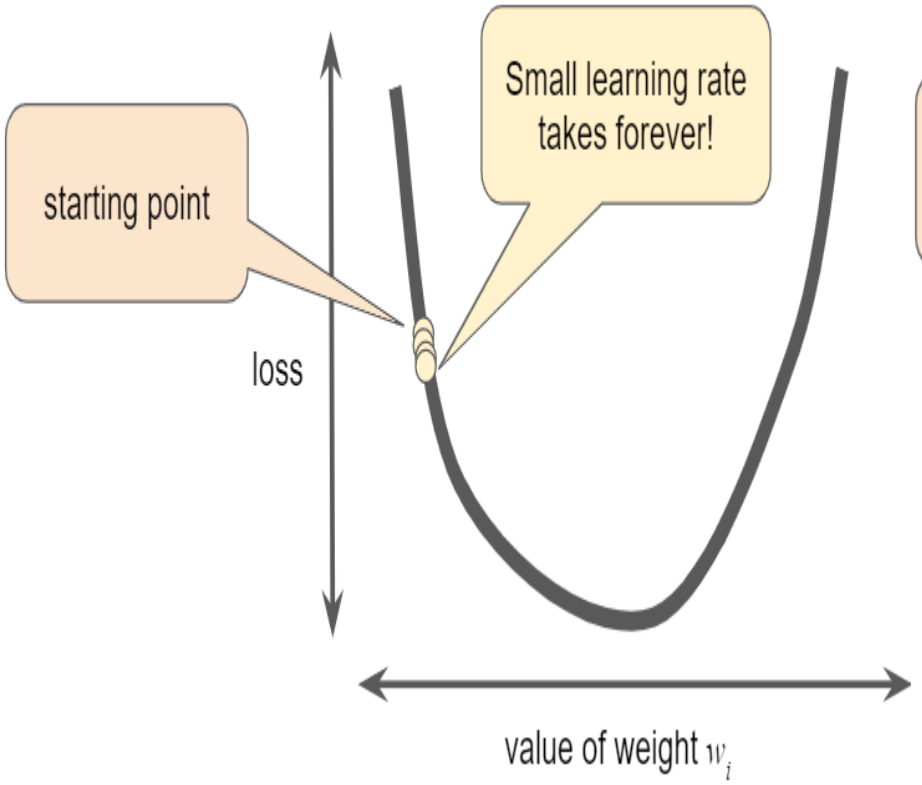
- Big or Small?

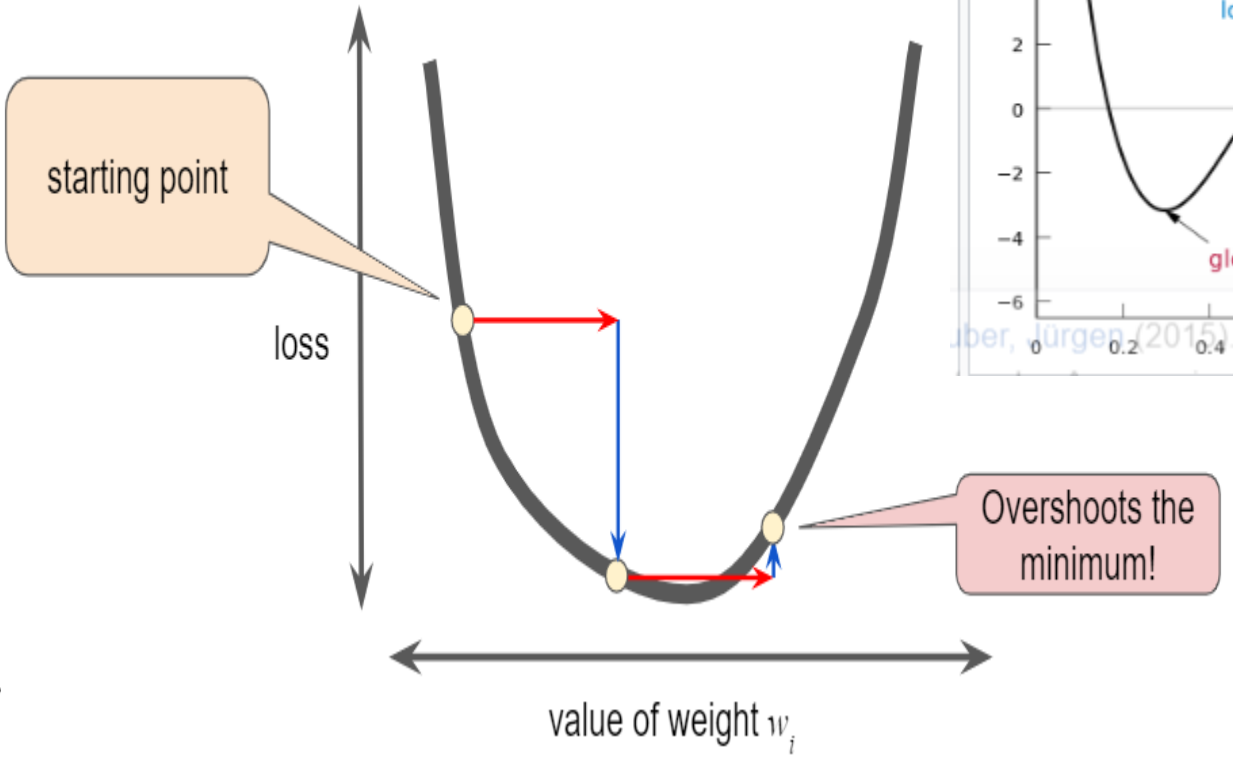
The idea is that by being able to compute the derivative/slope of the function, we can find the minimum of a function.

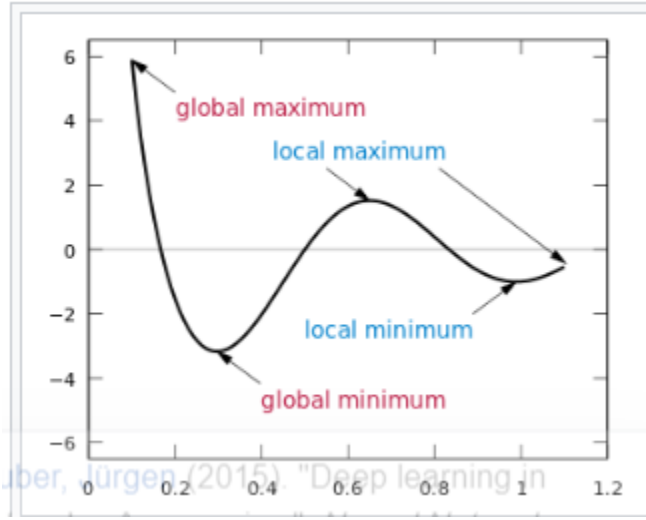
- If we are able to compute the derivative of a function, we know in which direction to proceed to minimize it.
- Learning Rate decides the steps size towards the direction. [learning rate * derivatives]
- Here, Function indicates the objective function of our model.

Learning Rate: size of steps taken to reach the minimum or bottom

Too Small vs Too Large







Try and visualize at : https://developers.google.com/machine-learning/crash-course/fitter/graph

Backpropagation: Calculates gradient descent(partial derivatives) of cost function and repeat the steps by going back and updating the weights (and biases) to get the optimum values.

Backpropagation was invented in 1960. Later, the term in neural networks was announced by Rumelhart, Hinton & Williams (1986).

What was actually wrong with backpropagation in 1986?

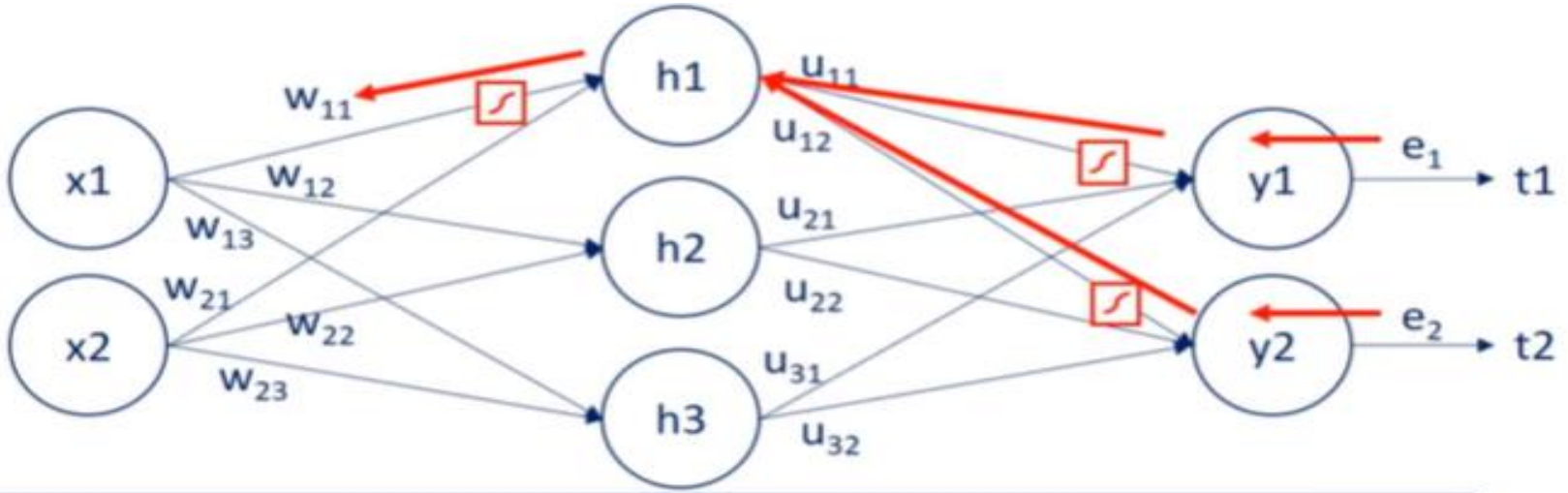
- We all drew the wrong conclusions about why it failed. The real reasons were:

1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.

What Was Actually Wrong With Backpropagation in 1986?
Slide by Geoff Hinton, all rights reserved.

Backpropagation: is one of the biggest challenges for the speed of an algorithm.

At the end of epoch the network back propagates and change the parameters accordingly



The algorithm adjusts:

the weights that have a **bigger** contribution to the errors by **more;**
the weights that have a **smaller** contribution to the errors by **less**

Image source: 365DataScience

Backpropagation: Why Computing Gradients?

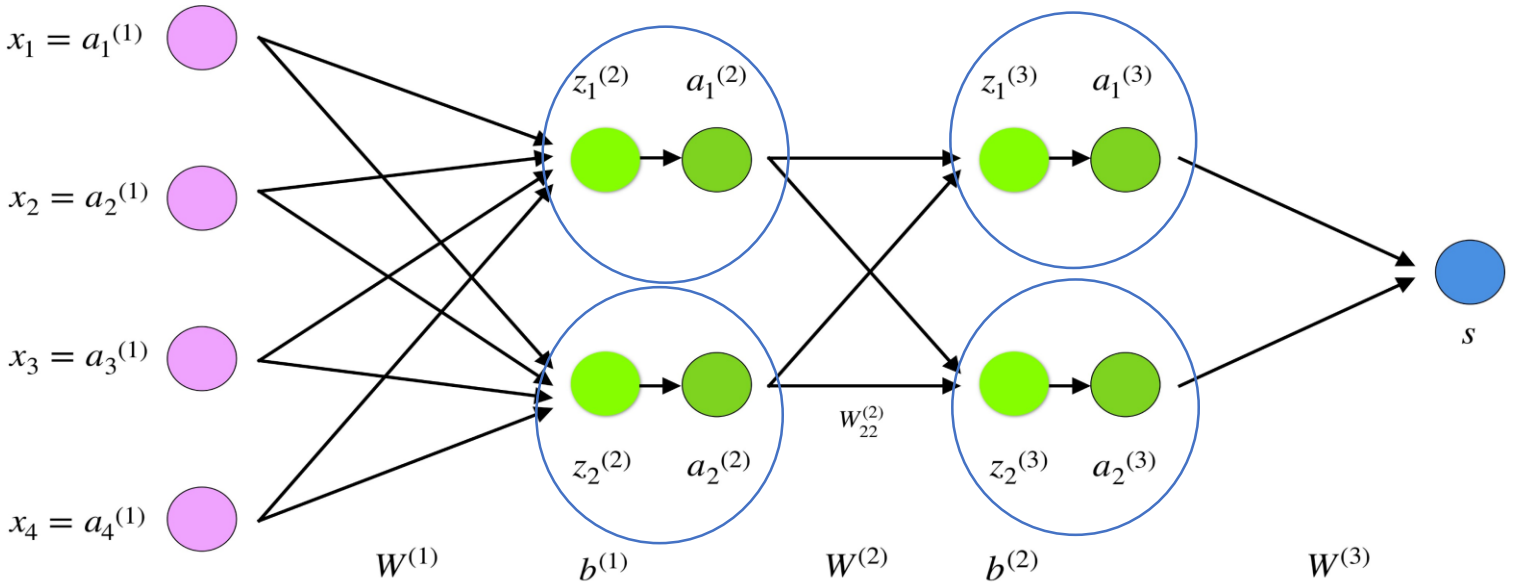
To answer this, we first need to revisit some calculus terminology:

- *Gradient of a function C($x_1, x_2, \ldots, x_m$) in point x is a vector of the partial derivatives of C in x.*

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$$

- *The derivative of a function C measures the sensitivity to change of the function value (output value) with respect to a change in its argument x (input value). In other words, the derivative tells us the direction C is going.*
- *The gradient shows how much the parameter x needs to change (in positive or negative direction) to minimize C.*

Backpropagation: How to Compute Gradients?



$x = a^{(1)}$ *Input layer*

$z^{(2)} = W^{(1)}x + b^{(1)}$ *neuron value at Hidden$_1$ layer*

$a^{(2)} = f(z^{(2)})$ *activation value at Hidden$_1$ layer*

$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$ *neuron value at Hidden$_2$ layer*

$a^{(3)} = f(z^{(3)})$ *activation value at Hidden$_2$ layer*

$s = W^{(3)}a^{(3)}$ *Output layer*

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix} \quad b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix}$$

Image source: towardsdatascience.com

Backpropagation: How to Compute Gradients?

Computing those gradients happens using a technique called **chain rules**:

For a single weight W_{jk}^{l}, the gradient is:

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} \qquad chain\ rule$$

$$z_{j}^{l} = \sum_{k=1}^{m} w_{jk}^{l} a_{k}^{l-1} + b_{j}^{l} \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} = a_{k}^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} a_{k}^{l-1} \qquad final\ value$$

Similar set of equations can be applied to b_{j}^{l} *is:*

$$\frac{\partial C}{\partial b_{j}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial b_{j}^{l}} \qquad chain\ rule$$

$$\frac{\partial z_{j}^{l}}{\partial b_{j}^{l}} = 1 \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial b_{j}^{l}} = \frac{\partial C}{\partial z_{j}^{l}} 1 \qquad final\ value$$

Backpropagation: How to Compute Gradients?

- The common part in both equations is often called *"local gradient"* and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

The *"local gradient"* can easily be determined using the chain rule.

- Algorithm for optimizing weights and biases (also called "Gradient descent")

while (termination condition not met)

$$w := w - \epsilon \frac{\partial C}{\partial w}$$

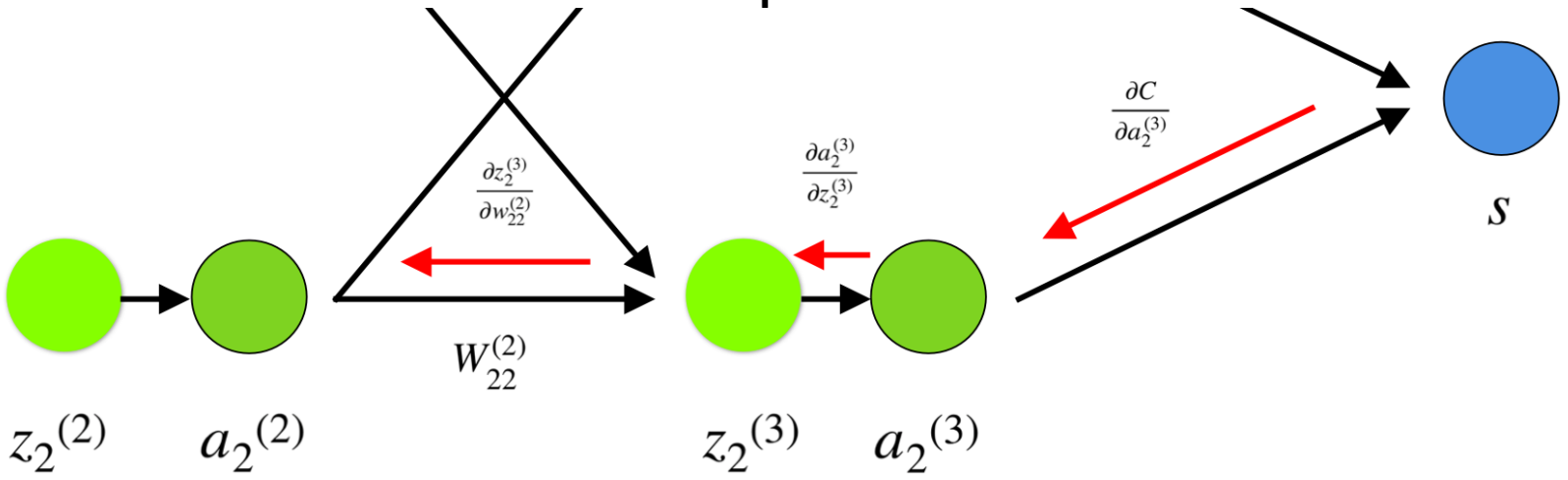
$$b := b - \epsilon \frac{\partial C}{\partial b}$$

end

- Initial values of w and b are randomly chosen.
- Epsilon (Ɛ) is the learning rate. It determines the gradient's influence (Also represented as Eta(ɳ).
- w and b are matrix representations of the weights and biases. Derivative of C in w or b can be calculated using partial derivatives of C in the individual weights or biases.
- Termination condition is met once the cost function is minimized (Can be terminated by setting number of epochs/early stopping)

Back Propagation: How to Compute Gradients?

Let's look into the bottom part of the neural network:



Weight W_{22}^2 connects a_2^2 and z_2^2, so computing the gradient requires applying the chain rule through z_2^3 and a_2^3

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

Calculating the final value of derivative of C in a_2^3 requires knowledge of the function C. Since C is dependent on a_2^3, calculating the derivative should be fairly straightforward.

Image source: towardsdatascience.com

Steps to train a Model:

1. • Randomly **initialized weights** to small number close to 0 (but not 0).
2. • Input first observation in your dataset in the input layer, **each feature in one input node.**
3. • **Forward-Propagation:** From left to right, the neuron are activated in such a way the impact of each neuron activation is limited by the weights associates with the neuron. Propagate the activation until getting the predicted result y.
4. • **Compare** the predicted result with the actual result and measure the error.
5. • **Back-Propagation:** From right to left, the error is back-propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much the model updates the weights.
6. • **Repeat** the steps **1 to 5** and update the weights after each observation (Reinforcement Learning).
   Or,
   • Repeat the steps 1 to 5 but update the weights after a batch of observations (Batch Learning).
7. • When the whole training set passed throw the model (ANN), that makes an epoch. **Redo more epochs**.

Training a model: Summary

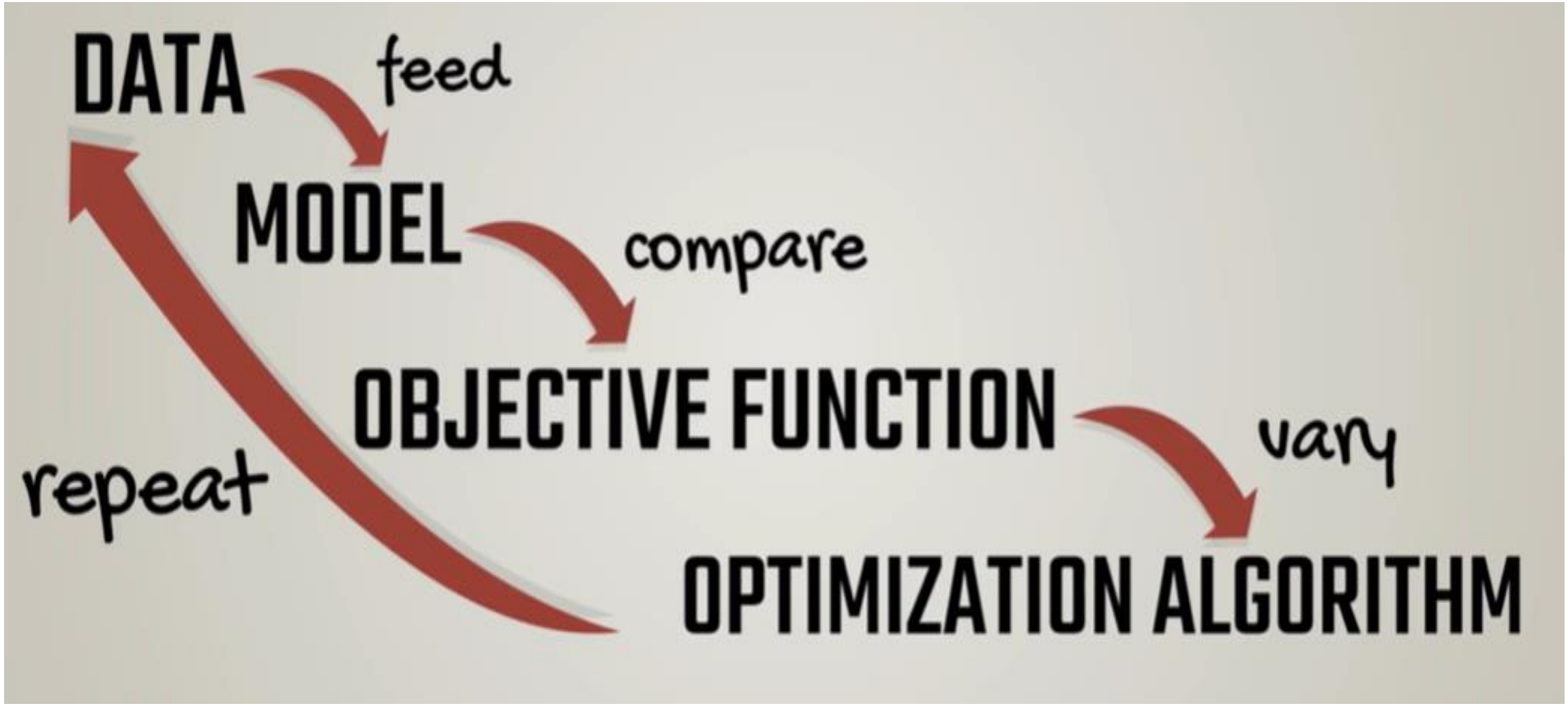


Image source: 365DataScience

How do Neural Networks Learn ?

[Single Neuron(Node) with Single observation]

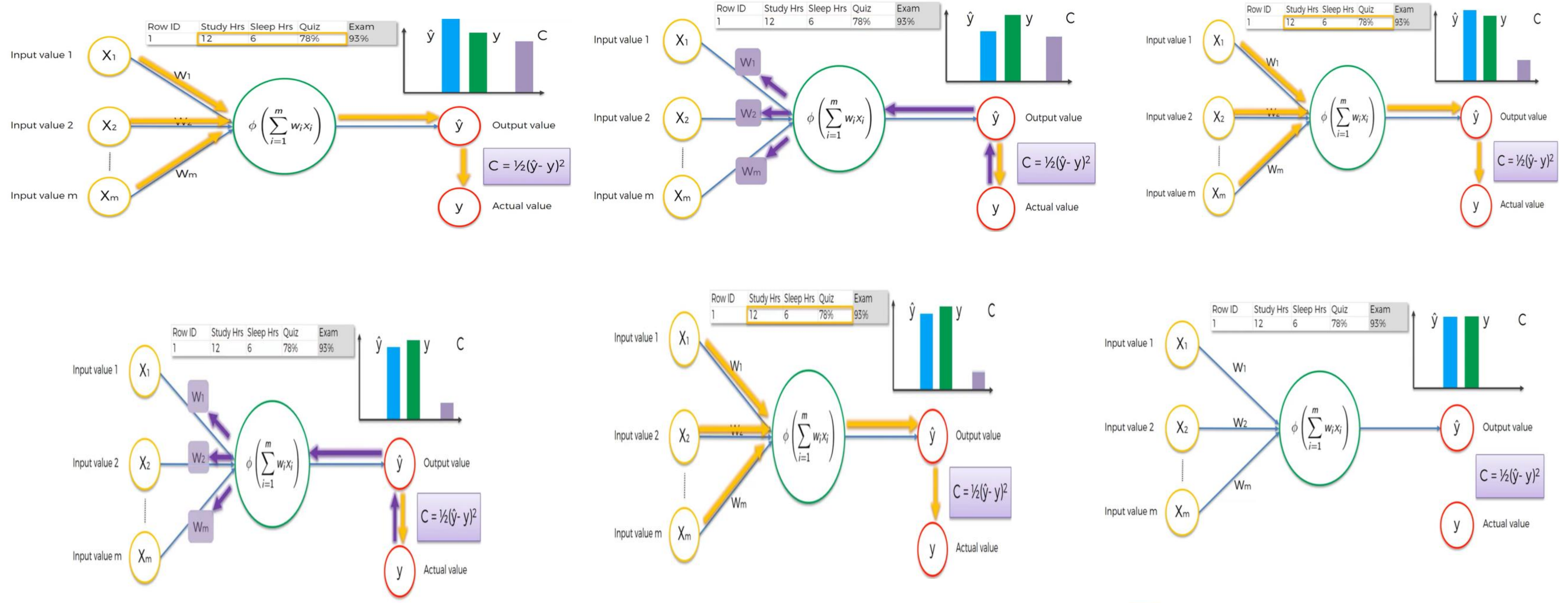
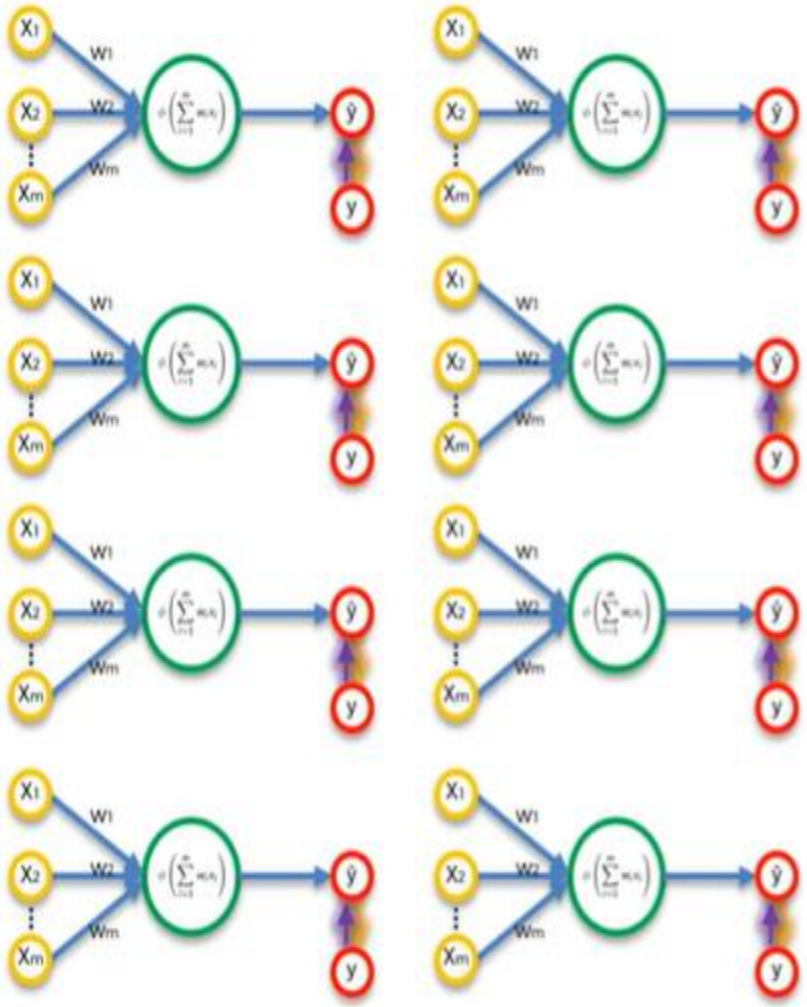


Image source: SuperDataScience

How do Neural Networks Learn ?

(Single Neuron(Node) with multiple observations)



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

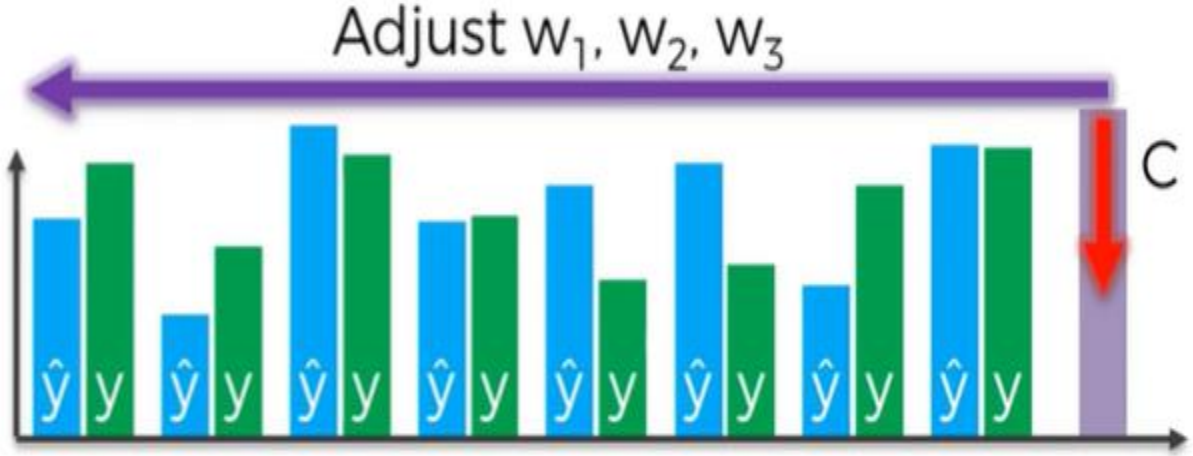


Image source: SuperDataScience

How do Neural Networks Learn ?

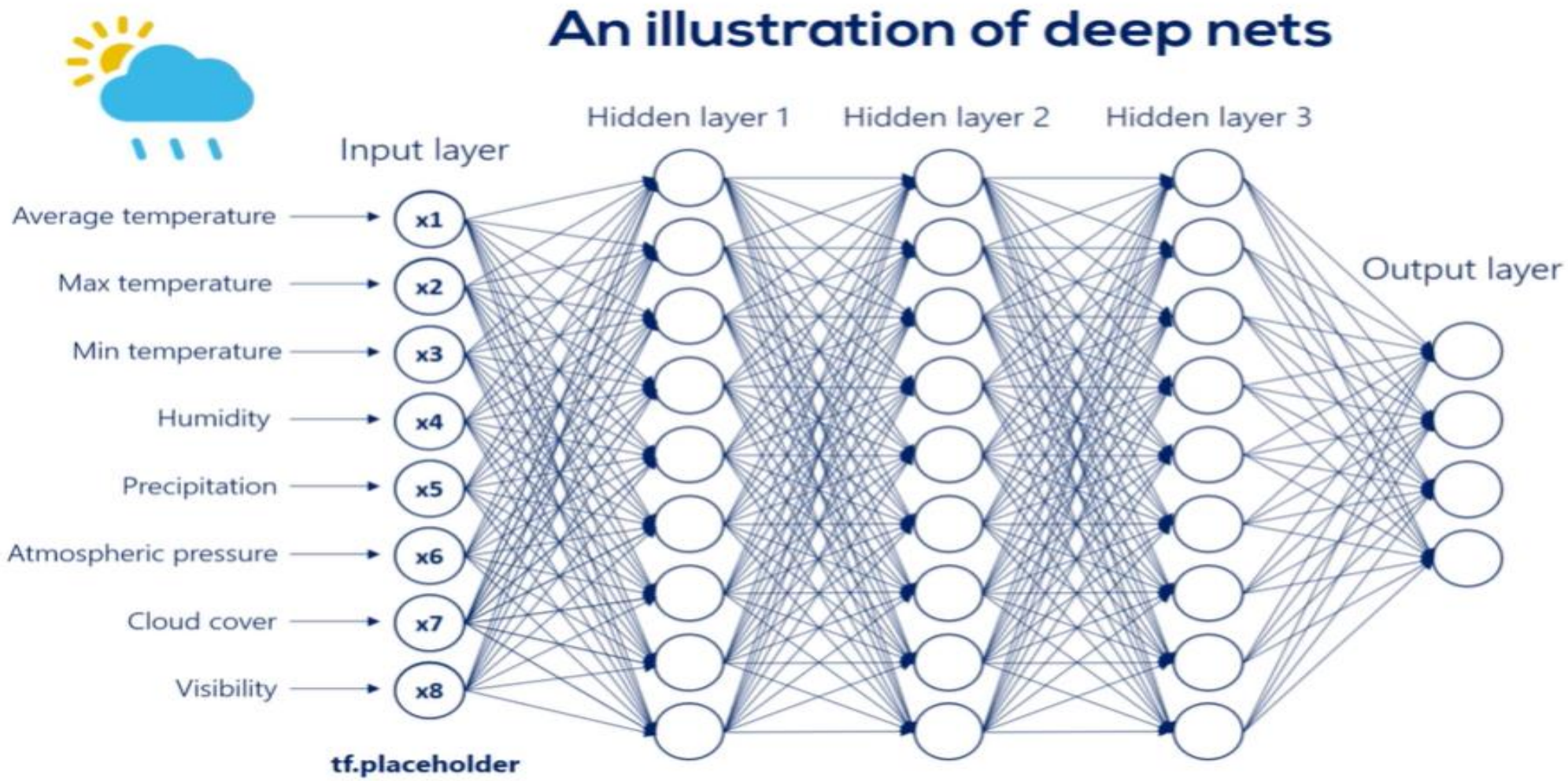
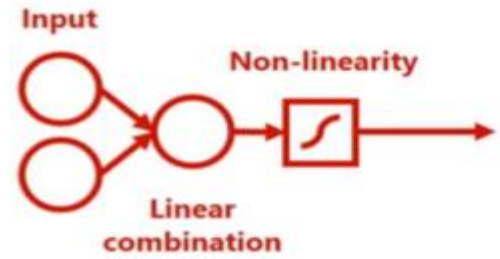
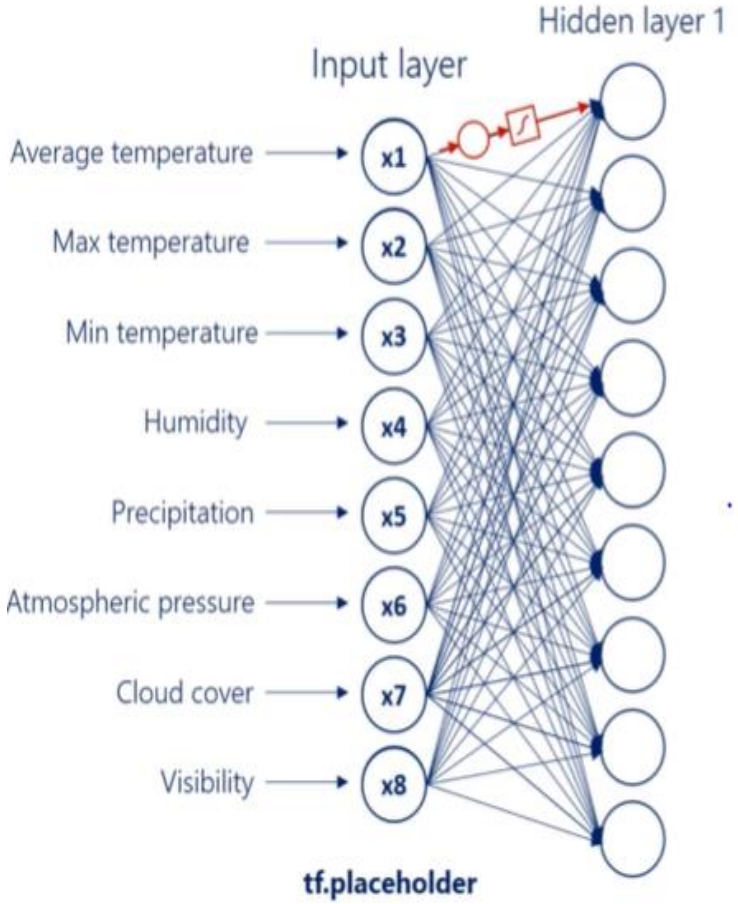


Image source: 365DataScience

How do Neural Networks Learn ?





w11	w12	w13	w14	w15	w16	w17	w18	w19
w21	w22	w23	w24	w25	w26	w27	w28	w29
w31	w32	w33	w34	w35	w36	w37	w38	w39
w41	w42	w43	w44	w45	w46	w47	w48	w49
w51	w52	w53	w54	w55	w56	w57	w58	w59
w61	w62	w63	w64	w65	w66	w67	w68	w69
w71	w72	w73	w74	w75	w76	w77	w78	w79
w81	w82	w83	w84	w85	w86	w87	w88	w89

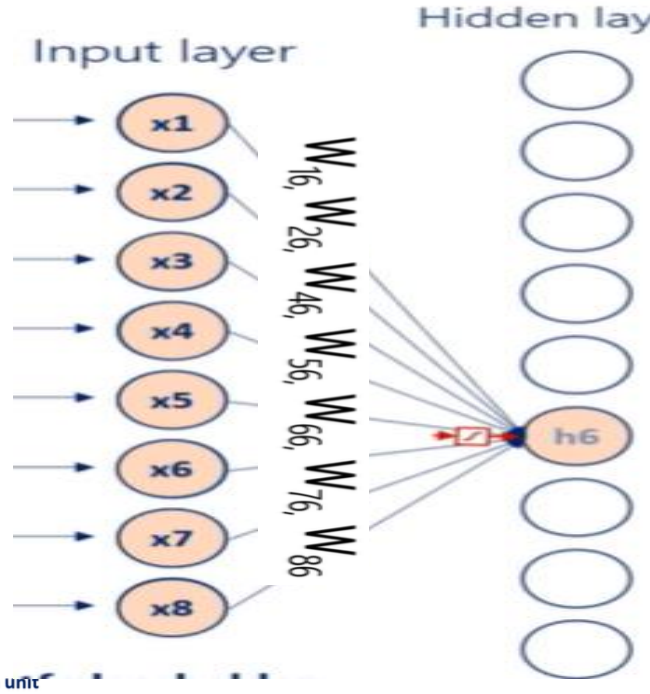
$X * W =$

$1 \times 8 * 8 \times 9 = 1 \times 9$

$8 \times 9 = 72$

W_{36}

Input unit Hidden unit



Each arrows multiplies weights (also added bias).
Then applies non-linearity.

Image source: 365DataScience

How do Neural Networks Learn ?

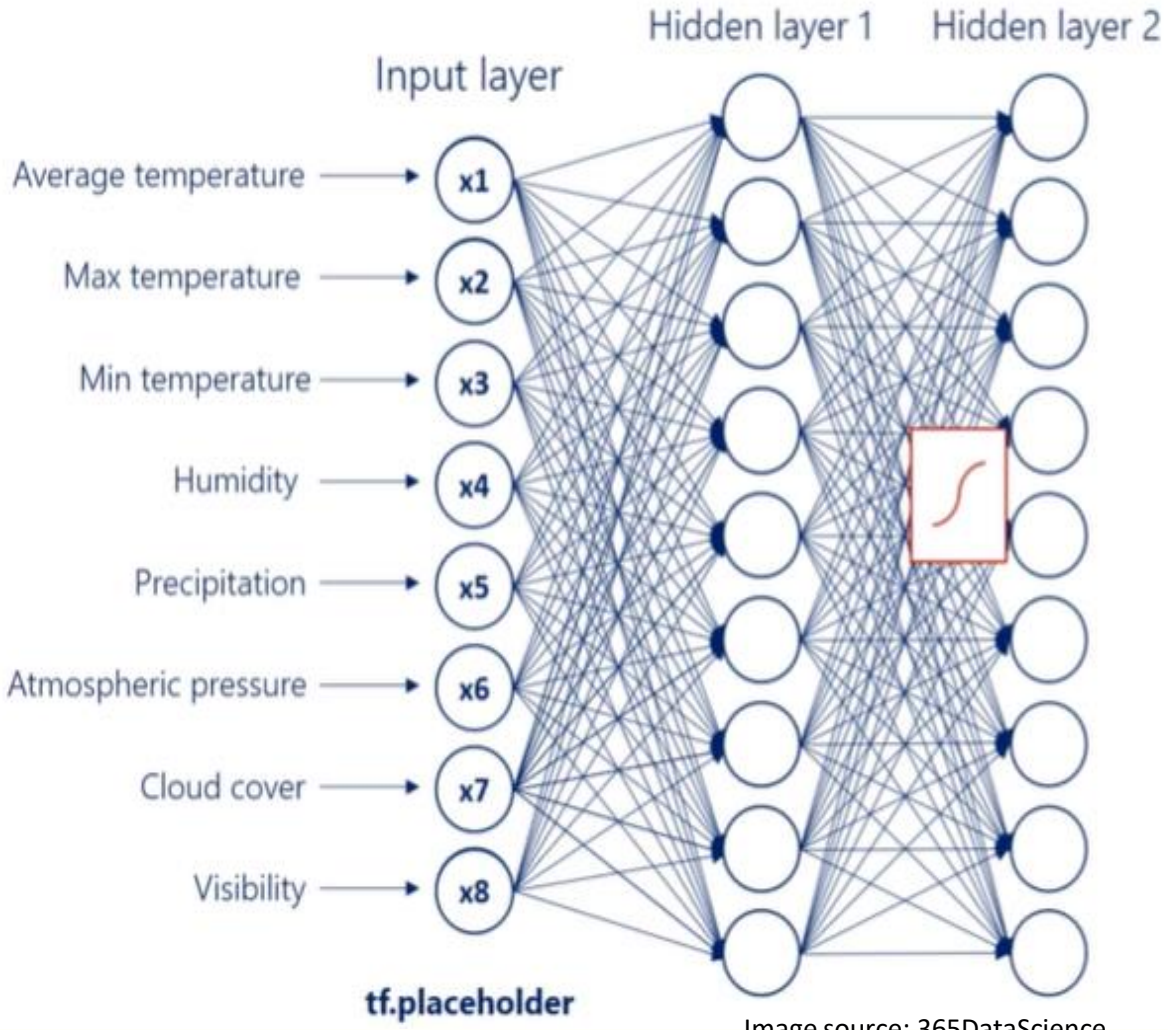
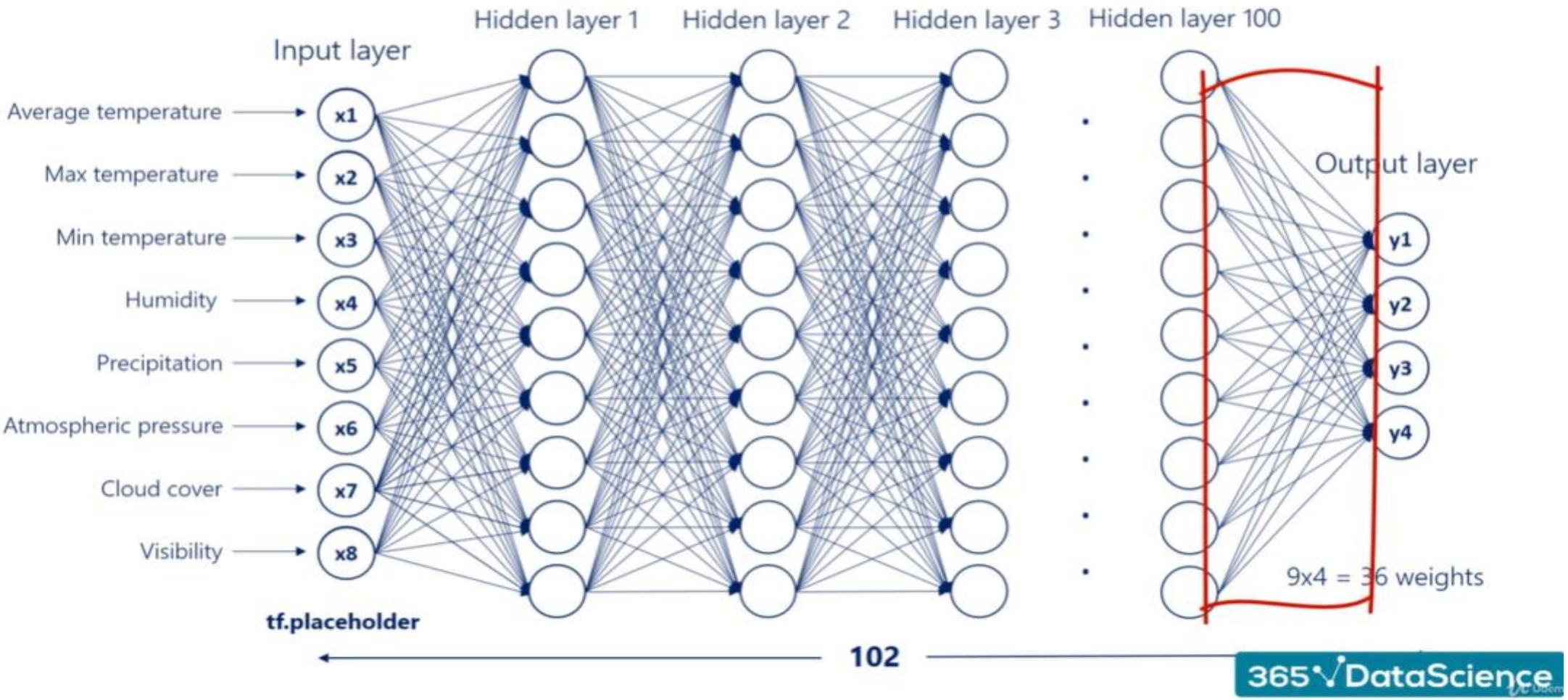


Image source: 365DataScience

w11	w12	w13	w14	w15	w16	w17	w18	w19
w21	w22	w23	w24	w25	w26	w27	w28	w29
w31	w32	w33	w34	w35	w36	w37	w38	w39
w41	w42	w43	w44	w45	w46	w47	w48	w49
w51	w52	w53	w54	w55	w56	w57	w58	w59
w61	w62	w63	w64	w65	w66	w67	w68	w69
w71	w72	w73	w74	w75	w76	w77	w78	w79
w81	w82	w83	w84	w85	w86	w87	w88	w89
w91	w92	w93	w94	w95	w96	w97	w98	w99

9 x 9 = 81

How do Neural Networks Learn ?



Try and visualize at: http://playground.tensorflow.org/

Parameters vs. Hyperparameters

Parameters

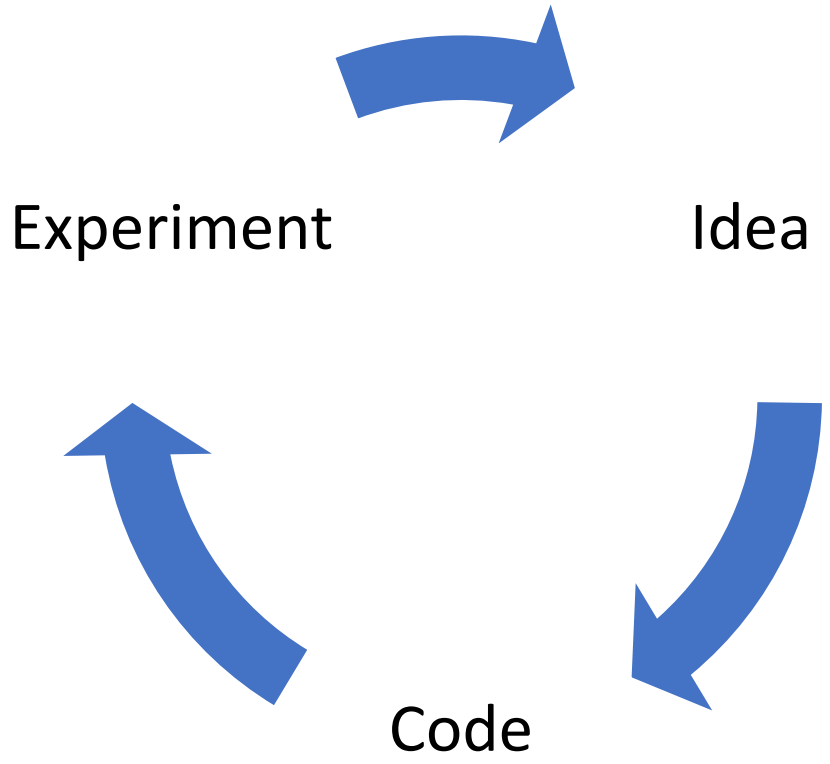
- Estimated or learned form Data (founded by optimization).
- Are not often set manually.
- Examples: Weights(w/theta/j), Biases(b) etc.

Hyperparameters

- To help estimate model's parameter.
- Are often pre-set manually.
- Examples:
  Width, Depth, Epochs, Learning Rate(η), Batch Size, Momentum Coefficient(α), Decay Coefficient(c) etc.

Experiment by tuning Hyperparameter to get the optimal result.

[Applied ML/Deep Learning is a highly iterative process]



Deep Learning Hands-On:

During the rest of the workshop, we will implement the theoretical concept we learned so far.

Please get the python codes from the following GitHub repository:

https://github.com/ShaonBhattaShuvo/Deep-Learning-Workshop/blob/master/Workshop%20(Part%201)/WorshopDL_(Part1).py

Some useful contents came in handy to prepare the lecture:

1. https://machinelearningmastery.com/what-is-deep-learning/
2. https://www.mathworks.com/discovery/deep-learning.html
3. https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/
4. http://content.time.com/time/interactive/0,31813,2048601,00.html
5. https://www.cs.toronto.edu/~tijmen/csc321/
6. https://cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf
7. https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/
8. https://www.sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html
9. https://www.coursera.org/specializations/deep-learning#courses
10. https://www.udemy.com/course/artificial-intelligence-az/
11. https://www.udemy.com/course/machinelearning/
12. https://www.udemy.com/course/the-data-science-course-complete-data-science-bootcamp/
13. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
14. http://neuralnetworksanddeeplearning.com/chap5.html#the_vanishing_gradient_problem
15. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
16. http://www.deeplearningbook.org/
17. https://mlfromscratch.com/activation-functions-explained/#/