南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

cnMed

CS 103 - 16

Introduction to AI Review

Jimmy Liu 刘江

2022-12-30

ALPHAFOLD 2022

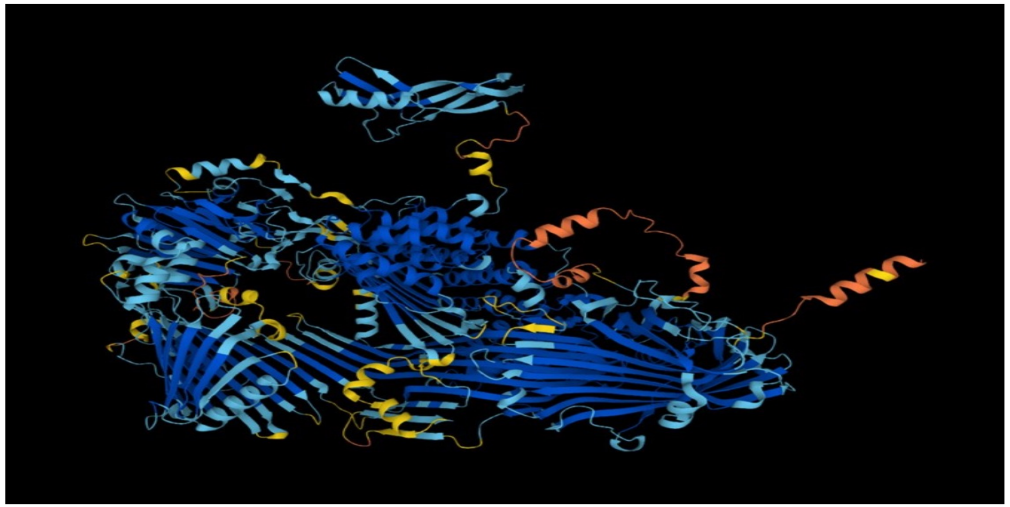
DeepMind chief executive Demis Hassabis says AlphaFold's new database covers "the entire protein universe". Credit: Jung Yeon-Je/AFP/Getty

NEWS | 28 July 2022 | Correction 29 July 2022

'The entire protein universe': AI predicts shape of nearly every known protein

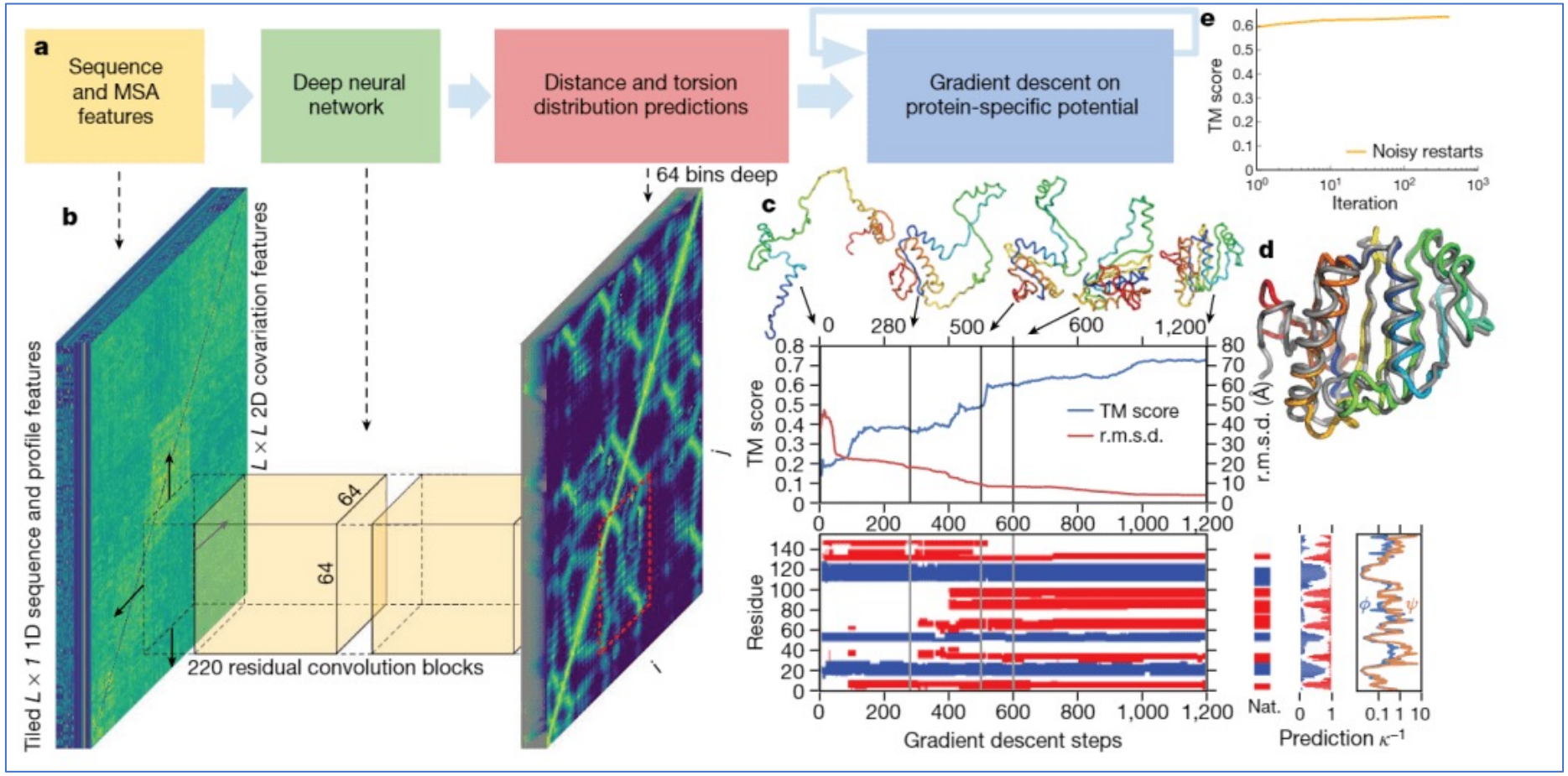
DeepMind's AlphaFold tool has determined the structures of around 200 million proteins.

Ewen Callaway



The structure of the vitellogenin protein — a precursor of egg yolk — as predicted by the AlphaFold tool. Credit: DeepMind

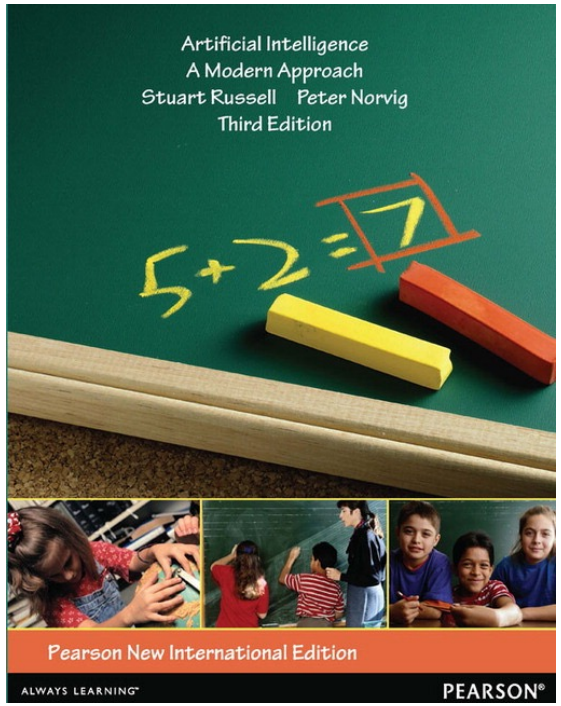
ALPHAFOLD Algorithm



Study Methods and Reference

Active learning: It is about how much you think and learn

Collective study: Let us study together



CS 103 Course Reference Book

已附加文件: Artificial_Intelligence_-_A_Modern_Approach,_3rd_.pdf (19.91 MB)

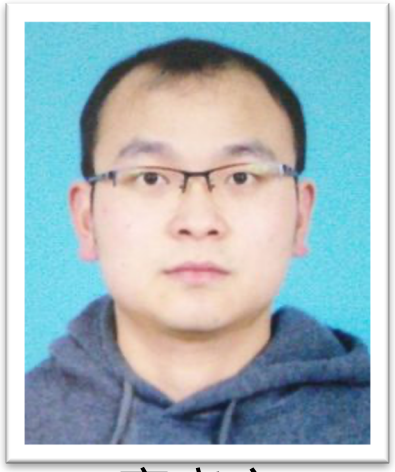
Artificial Intelligence – A Modern Approach (AIMA) (Russell/Norvig)

This is a very comprehensive textbook in AI, the PDF version of the book is attached and and much information about the book can be found at:

Web site: http://aima.cs.berkeley.edu/

Enjoy reading

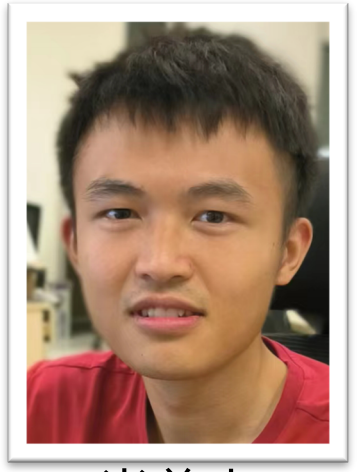
Study Methods and Reference- TAs



章晓庆

聂秋实

杨冰



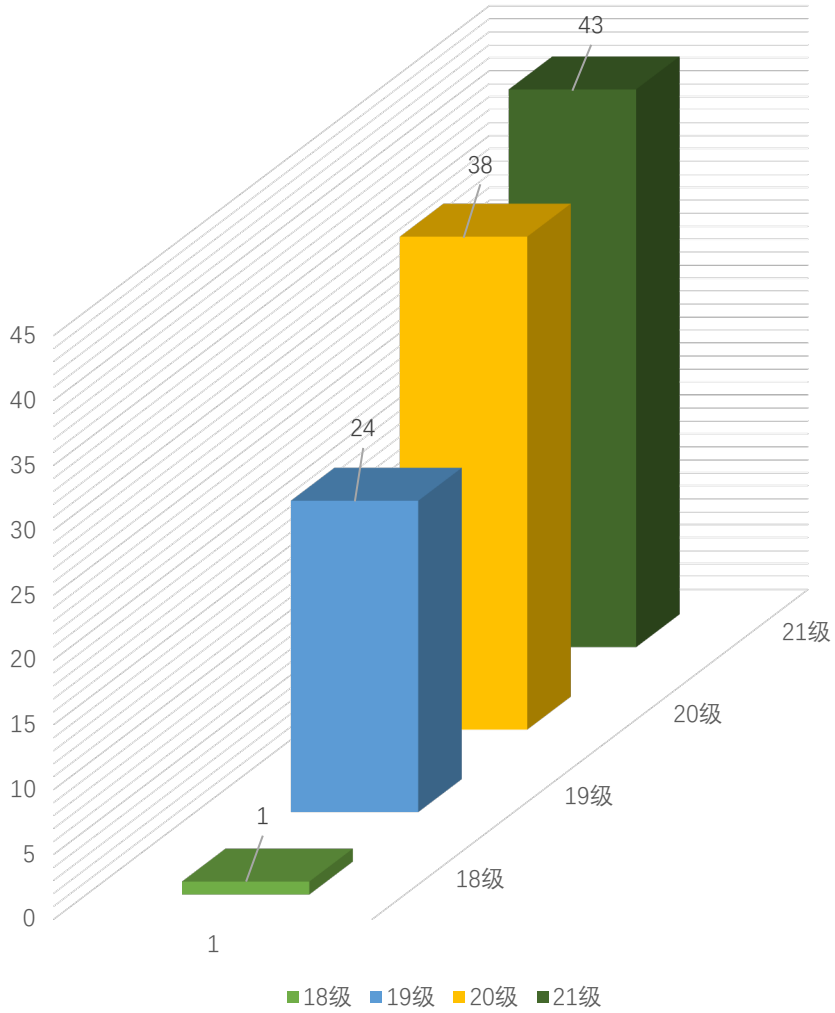
肖尊杰

巫晓

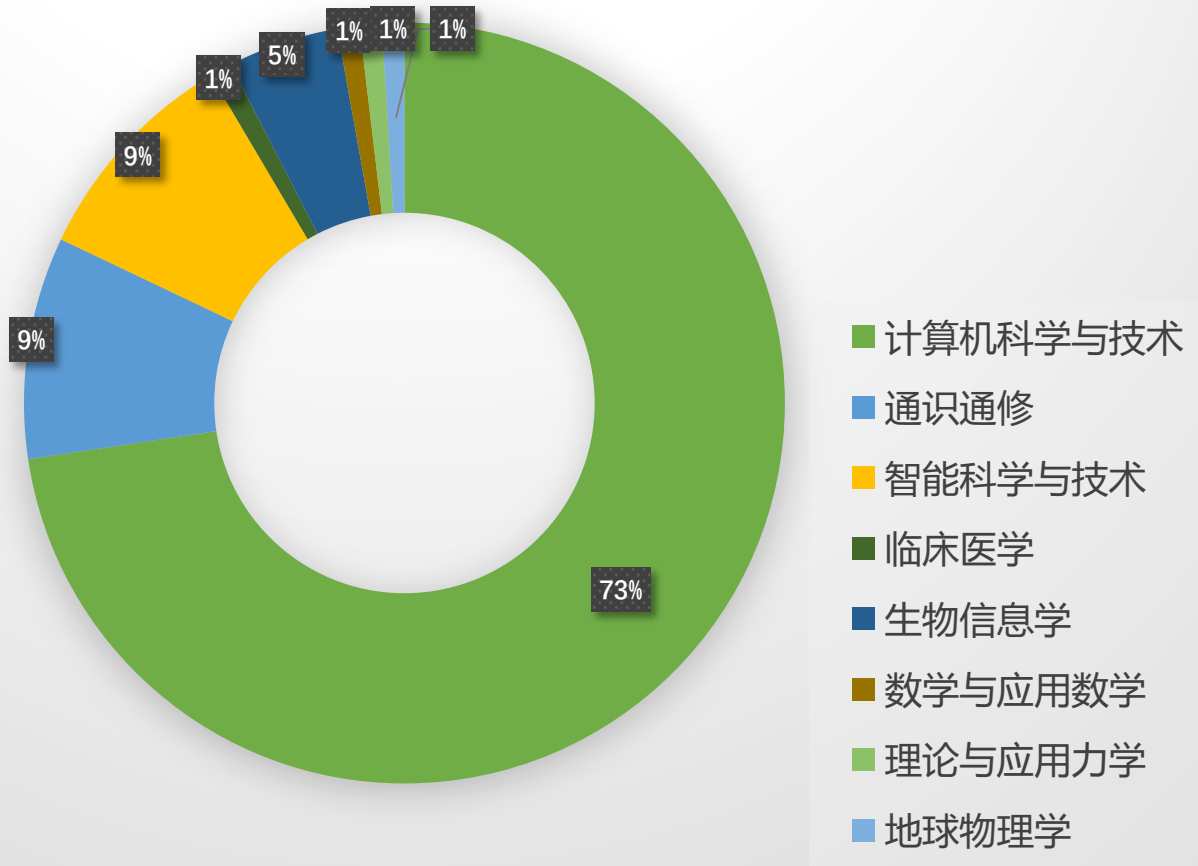
赵冀鲁

2022 CS103 student Summarization

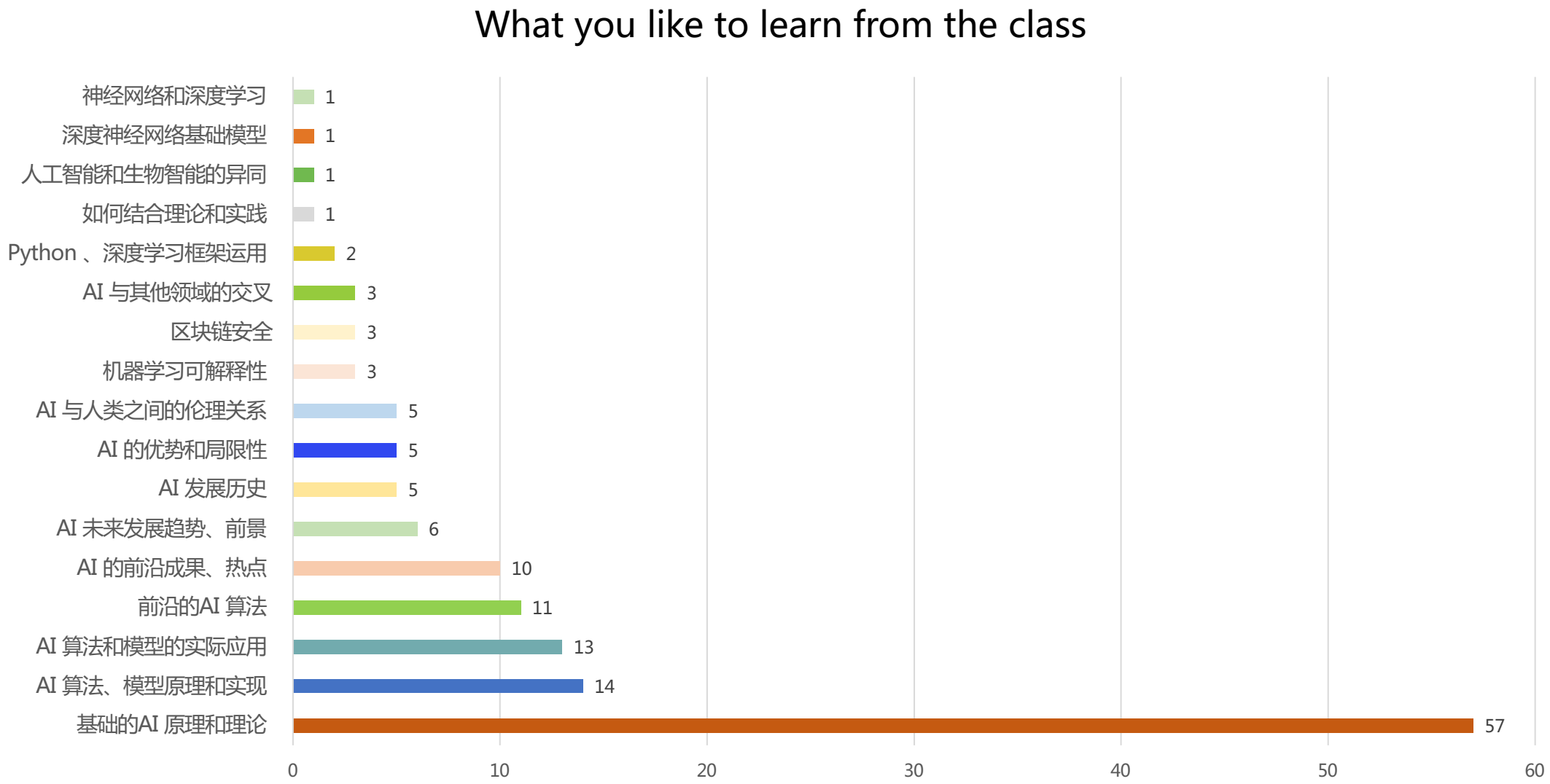
人工智能导论年级分布



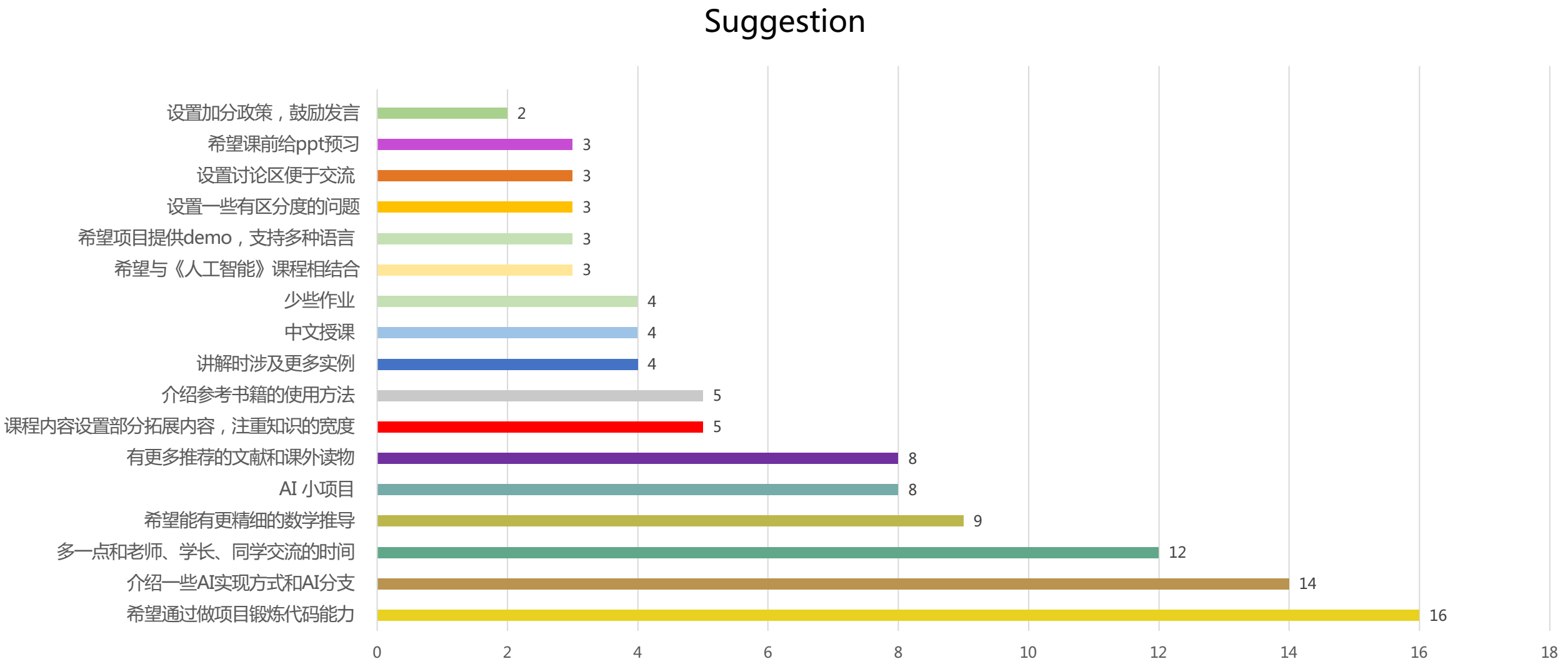
人工智能导论专业分布



CS103 Homework Summarization



CS103 Homework Summarization

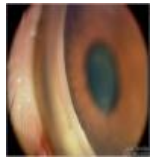


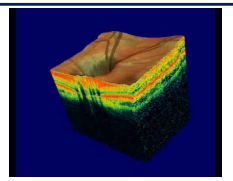
Ocular AI

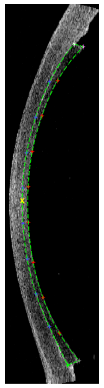
AS-OCT (Cornea and ACA)

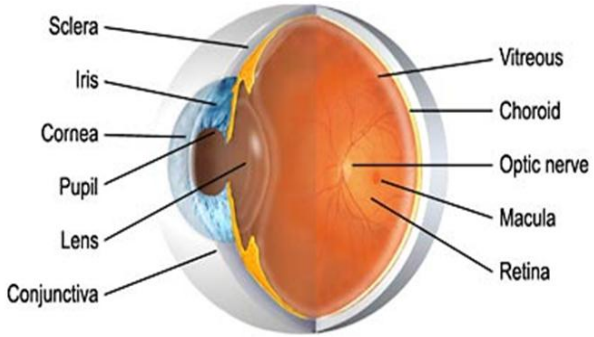
Fundus (Retina)

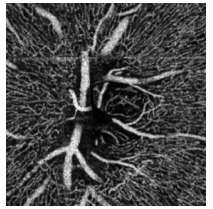
OCT (Optic nerve, Macula, Choroid)



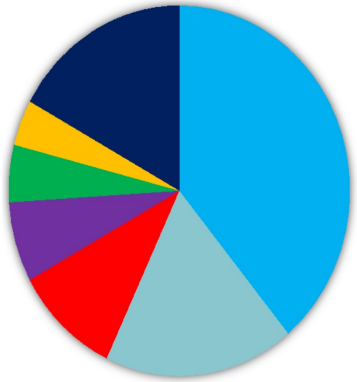


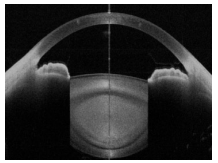


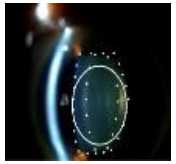


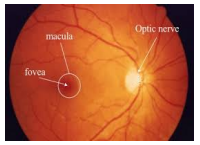


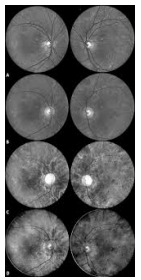
Blinding disease











AS-OCT (Lens)

Slit lamp (Lens)

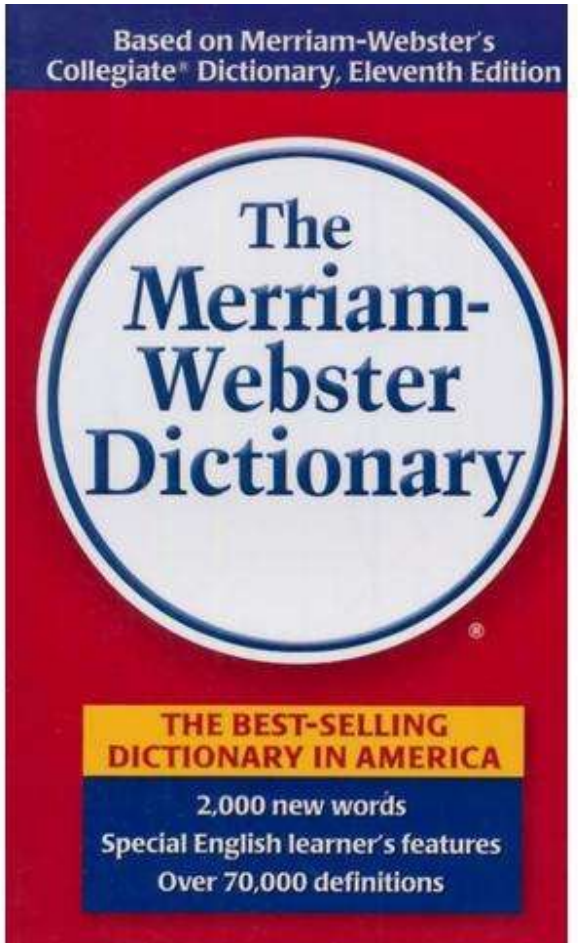
Multispectral image (Retina)

AI Concepts – "Intelligence" from Dictionaries

The ability to

- Learn
- Understand
- Deal with
- Try

new situations

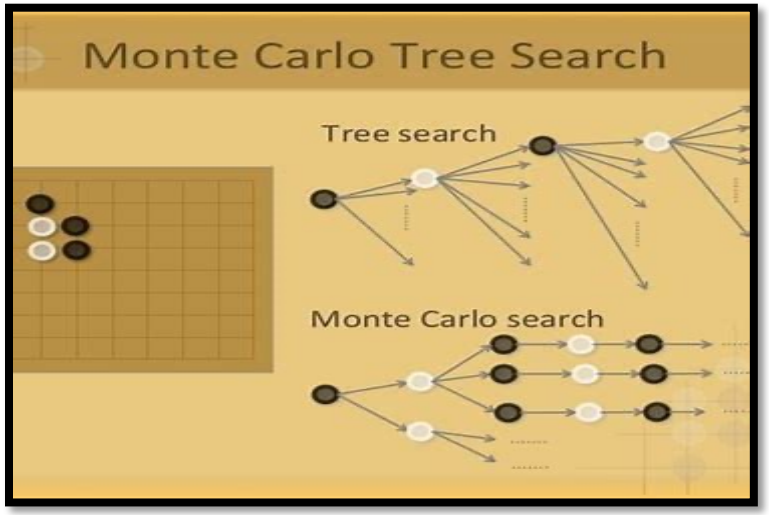


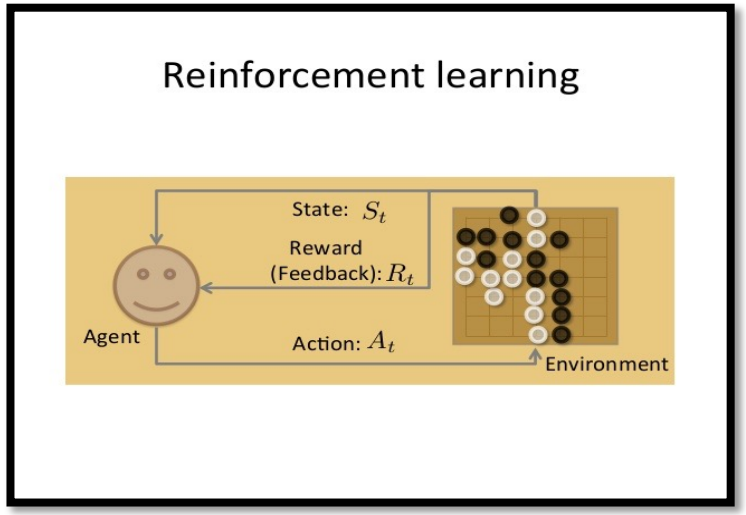
3 AI Categories:
In Real Life, AI is often the combination of the 3

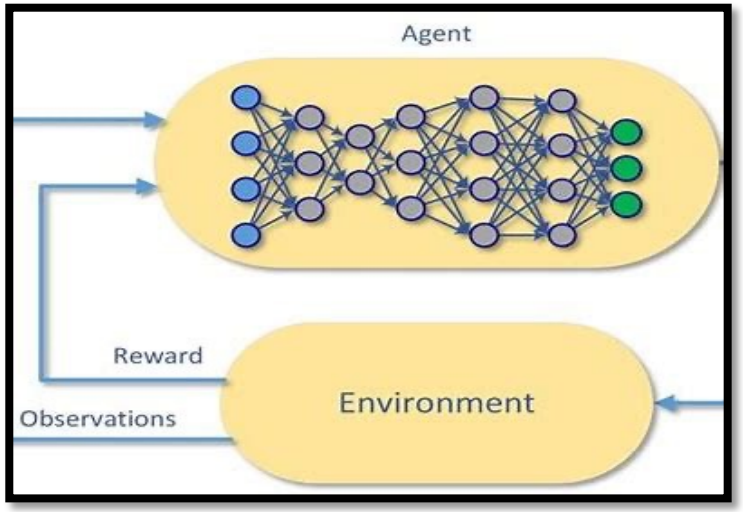
	符号主义	联结主义	行为主义
样本数量	小	大	小
计算性能	小	大	极大
相关学科	心理学	生理学	进化论
经典系统	专家系统	机器学习	遗传算法
模拟方式	功能模拟	结构模拟	行为模拟
是否犯错	从不犯错	偶尔犯错	经常犯错
逻辑特点	知其然知其所以然	知其然不知其所以然	不知其然不知其所以然

AI Concept- ALPHAGO

AlphaGo



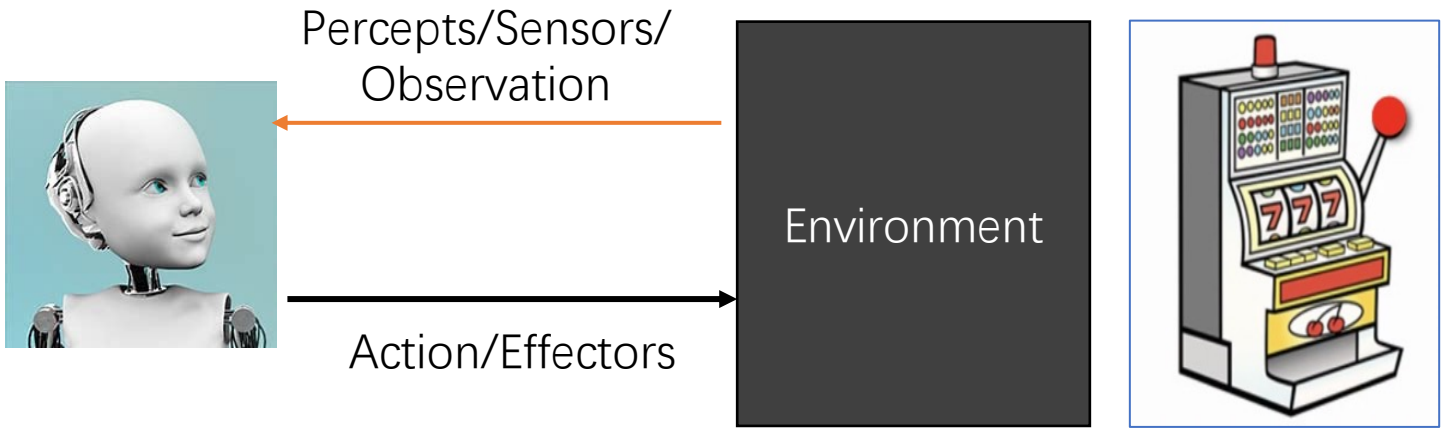




AI from Computer Science – Agent

- An agent is anything that can perceive its environment through sensors and acts upon that environment through effectors. Abstractly, an agent is a function from percept histories to actions:

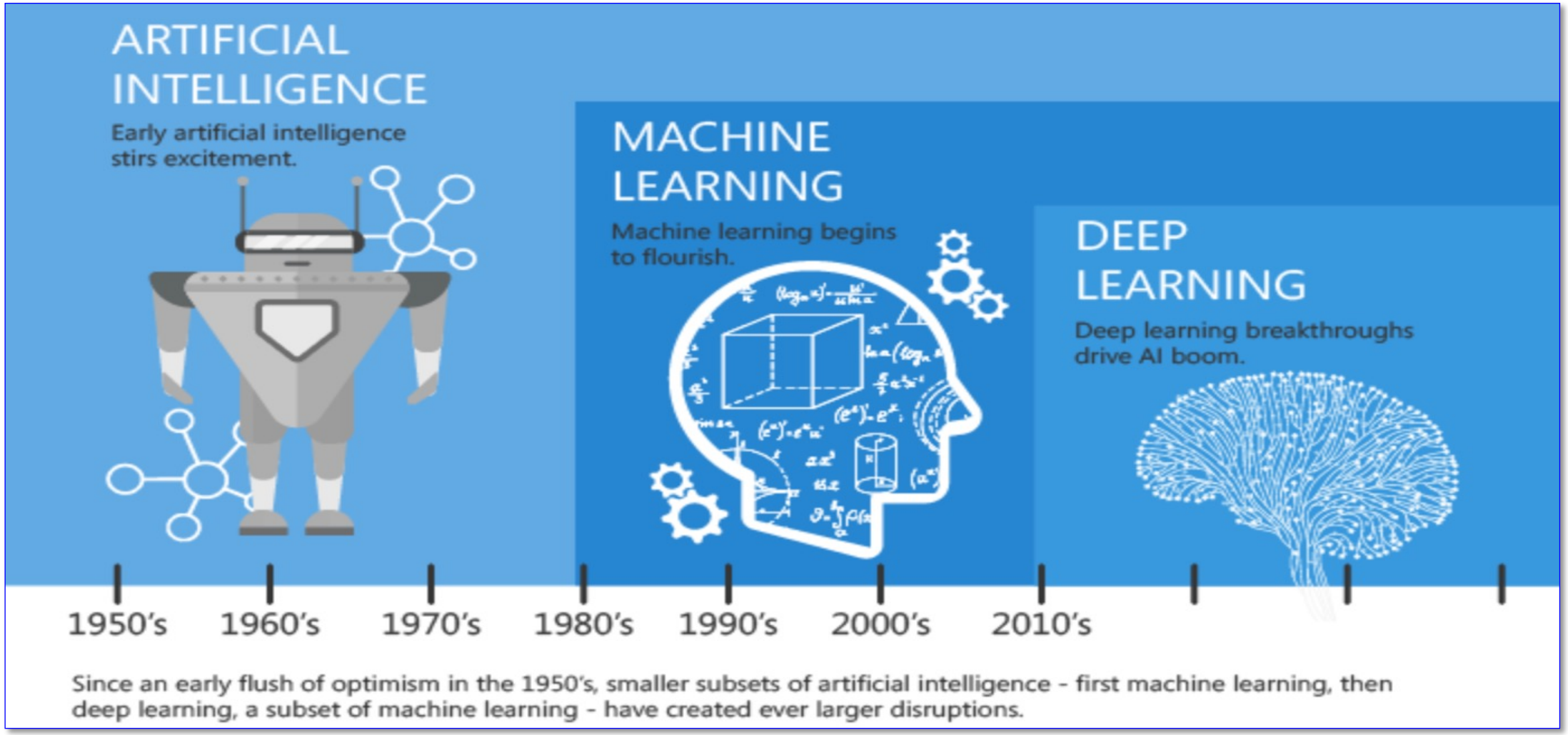
$$[f: P^* \rightarrow A]$$



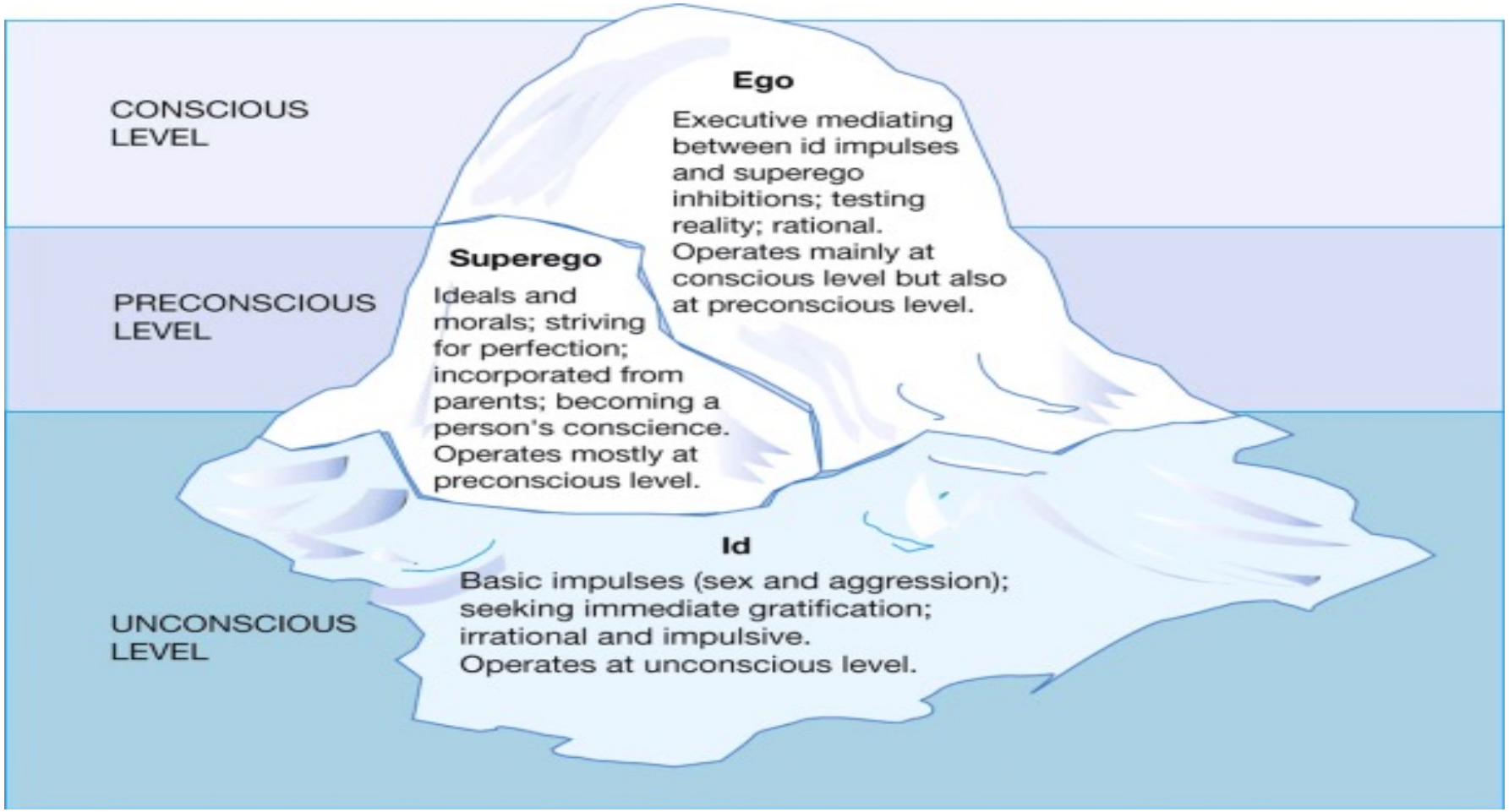
Reinforcement Learning and Agent with 5 Key Components

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. - Wiki

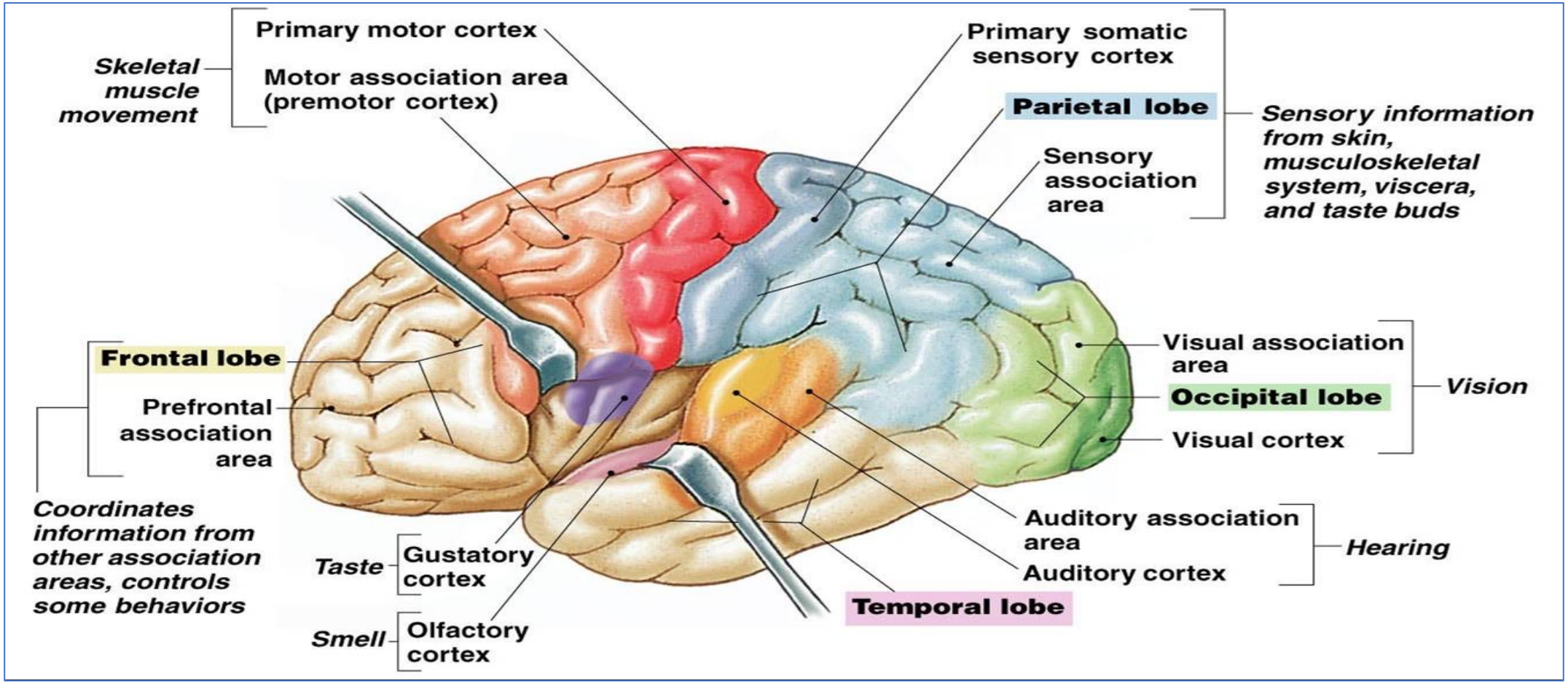
AI Algorithm Summary



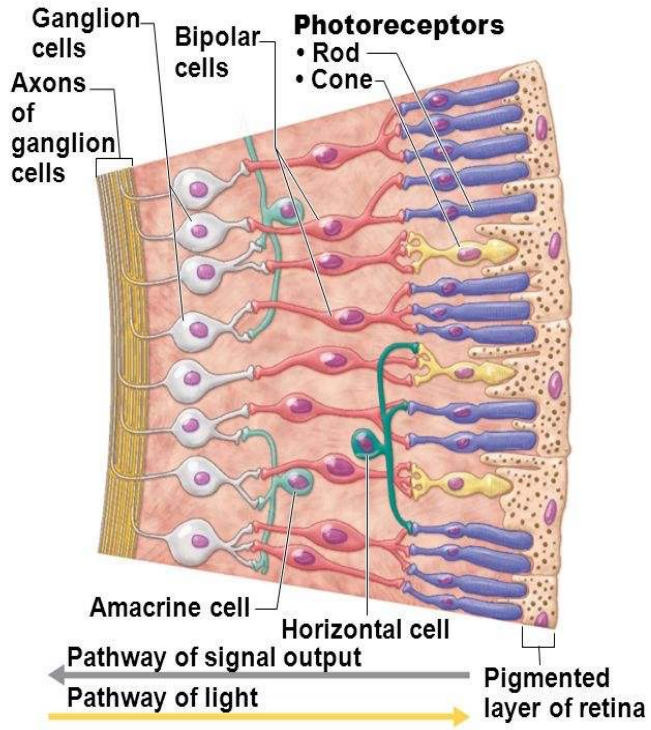
Human Mind from Freud



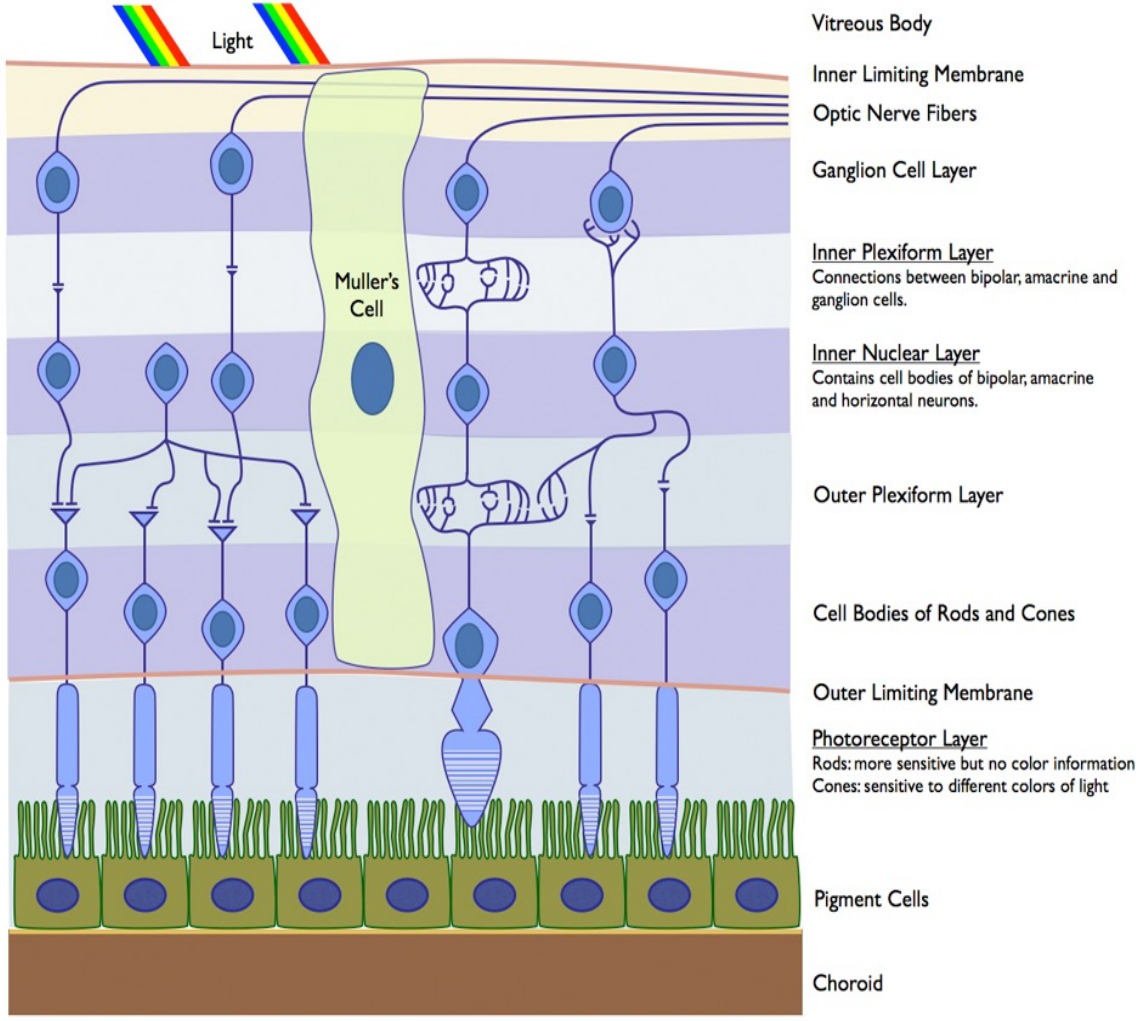
Human Brain – Lobes and Functions



Major Input to Brain: Eye Retina Layer Structure



The light rays that contribute to the retinal image have to pass through layer of nerve fibers and the blood vessels before reaching the light-sensitive photoreceptors.

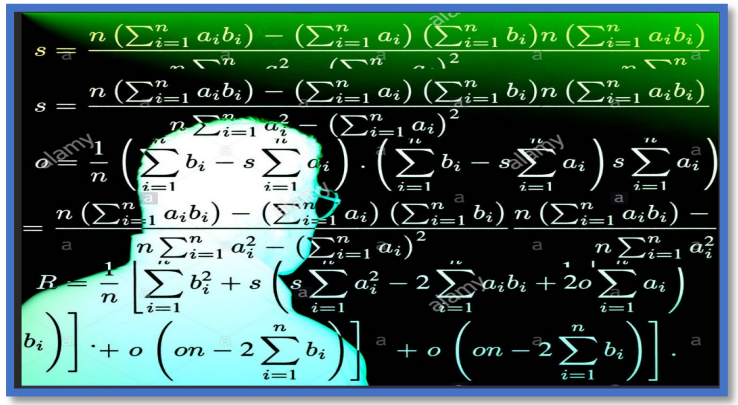


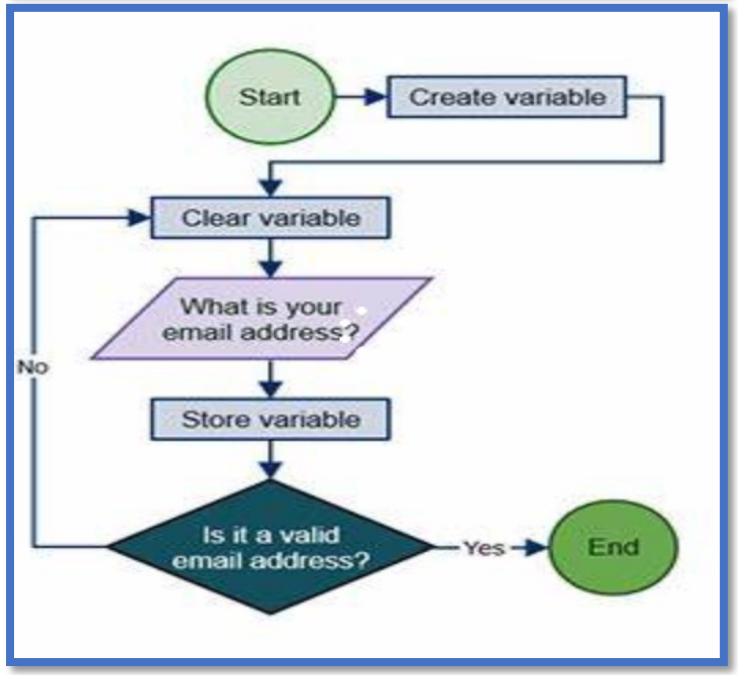
Computer Algorithm

Computer Algorithm:

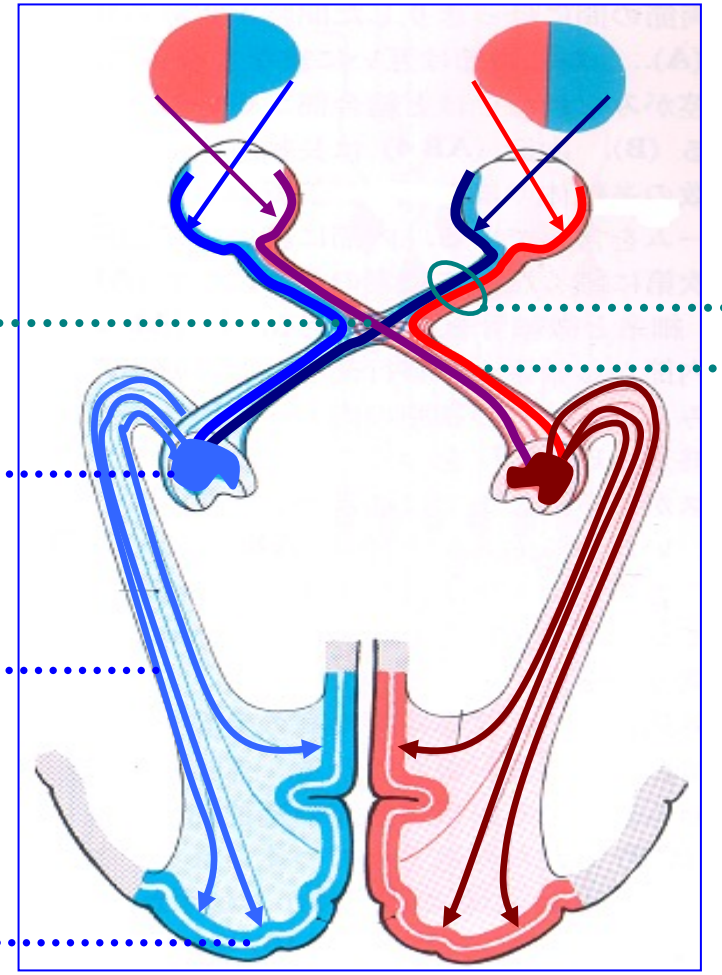
a well defined sequence of steps for solving a computational problem

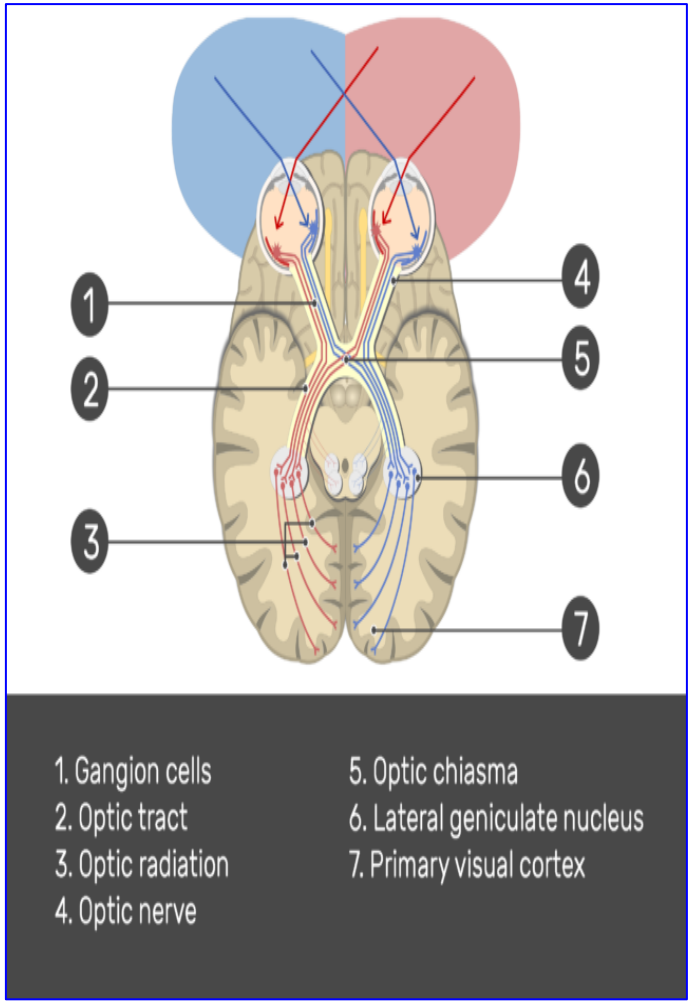
- It produces the correct output
- It uses basic steps / defined operations
- It finishes in finite time



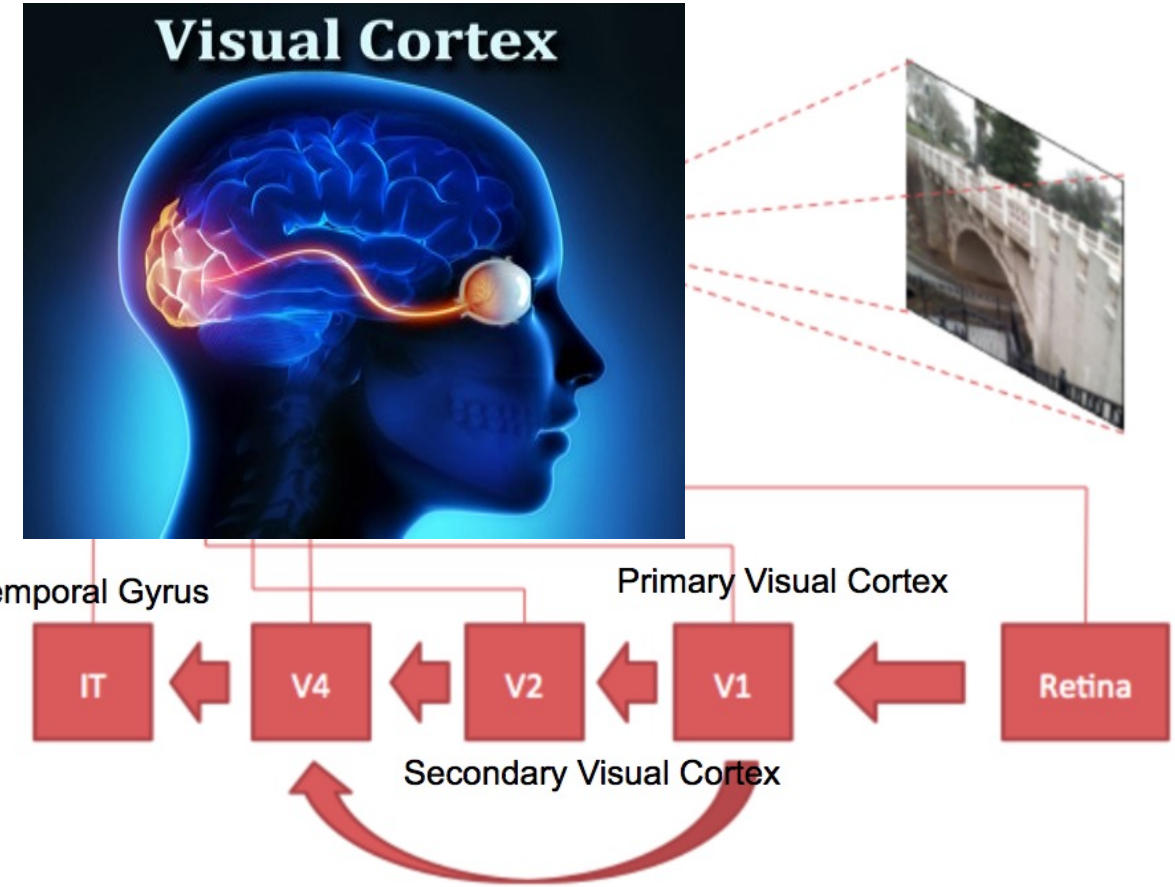


Visual Input to Brain: Visual Information Process





Simpler Model of Visual Input to Brain: Brain Computing



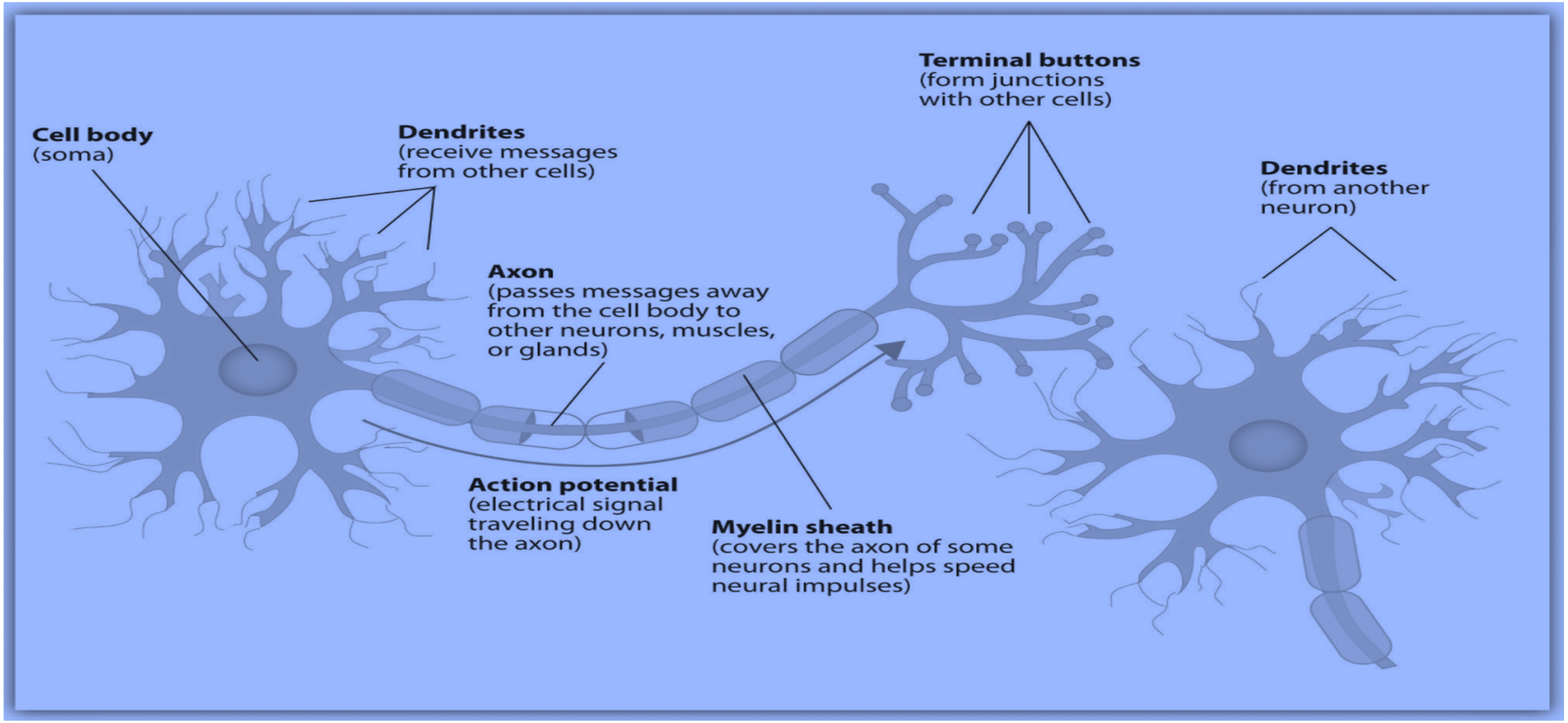
V1: Edge detection, etc.

V2: Extract simple visual properties (orientation, spatial frequency, color, etc)

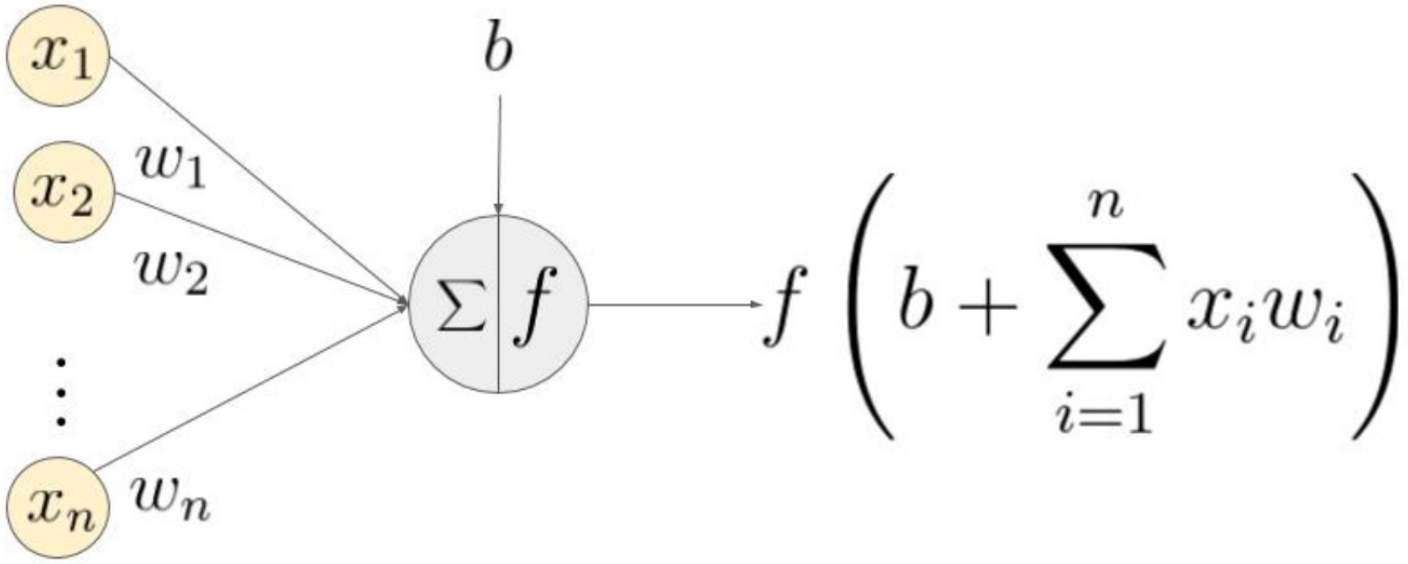
V4: Detect object features of intermediate complexity

TI: Object recognition.

What Really Happen in Neuron?

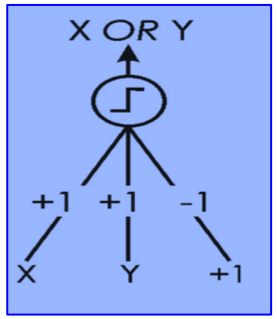


Artificial Neuron



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Prove "OR" MCP (McCulloch and Pitts) Neuron



A	B	Bias	W1	W2	W3	Transfer
0	0	1	1	1	-1	G(z)
0	1	1	1	1	-1	G(z)
1	0	1	1	1	-1	G(z)
1	1	1	1	1	-1	G(z)

$Z = \sum$ = A*W1+B*W2+ Bias*W3	Output F=G(Z) (Z>=0)	A OR B
0*1 + 0*1 + 1*(-1) = (-1)	0	0
0*1 + 1*1 + 1*(-1) = 0	1	1
1*1 + 0*1 + 1*(-1) = 0	1	1
1*1 + 1*1 + 1*(-1) = 1	1	1

Transfer Function is G(z)

$$g(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise.} \end{cases}$$

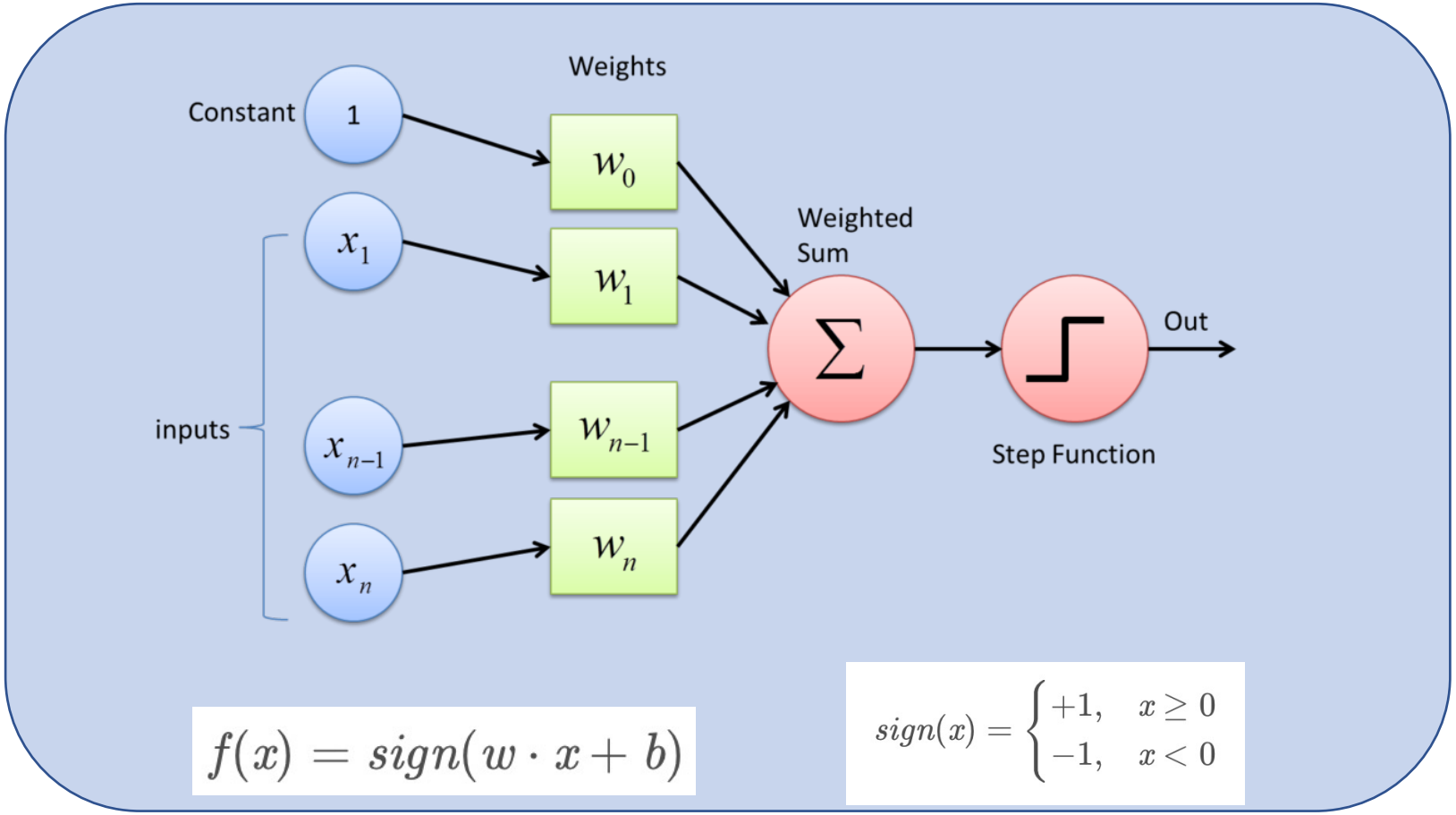
Turing Test – Operational Definition

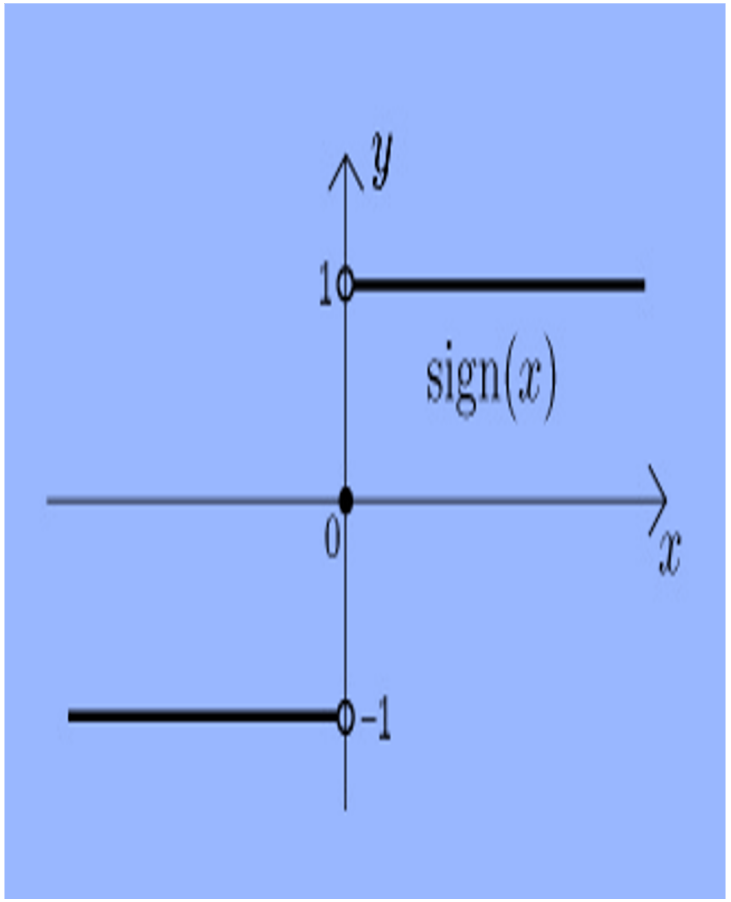
1. If the Turing Test was passed, Turing would conclude that the machine was intelligent.
2. Suggested as a way of saying when we could consider machines to be intelligent, or at least act intelligently
3. A satisfactory operational definition of intelligence

Chinese Room

The Chinese room argument holds that a digital computer executing a program cannot be shown to have a "mind", "understanding" or "consciousness", regardless of how intelligently or human-like the program may make the computer behave. The argument was first presented by philosopher John Searle in his paper, "Minds, Brains, and Programs", published in Behavioral and Brain Sciences in 1980. It has been widely discussed in the years since. The centerpiece of the argument is a thought experiment known as the Chinese room.

Traditional Perceptron

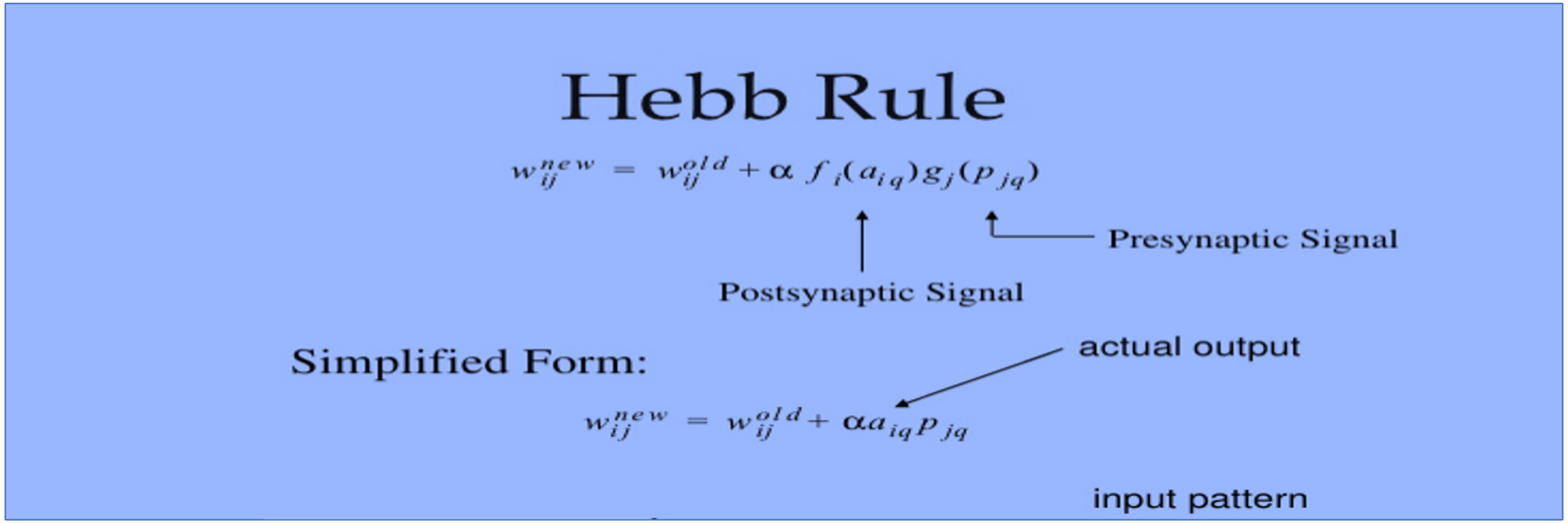




Perceptron Weights are Learned Y= F (W*X)

```
Initialize w⃗ = 0⃗                          // Initialize w⃗. w⃗ = 0⃗ misclassifies everything.
while TRUE do                              // Keep looping
    m = 0                                  // Count the number of misclassifications, m
    for (x_i, y_i) ∈ D do                  // Loop over each (data, label) pair in the dataset, D
        if y_i(w⃗^T · x⃗_i) ≤ 0 then          // If the pair (x⃗_i, y_i) is misclassified
            w⃗ ← w⃗ + yx⃗                    // Update the weight vector w⃗
            m ← m + 1                      // Counter the number of misclassification
        end if
    end for
    if m = 0 then                          // If the most recent w⃗ gave 0 misclassifications
        break                              // Break out of the while-loop
    end if
end while                                  // Otherwise, keep looping!
```

Hebb Rule to Update Weights



Typical Perceptron Weight Updates According to Perceptron Learning Rule (PLR)

- Weights modified for each example
- Update Rule:

$$w_i \leftarrow w_i + \Delta w_i$$

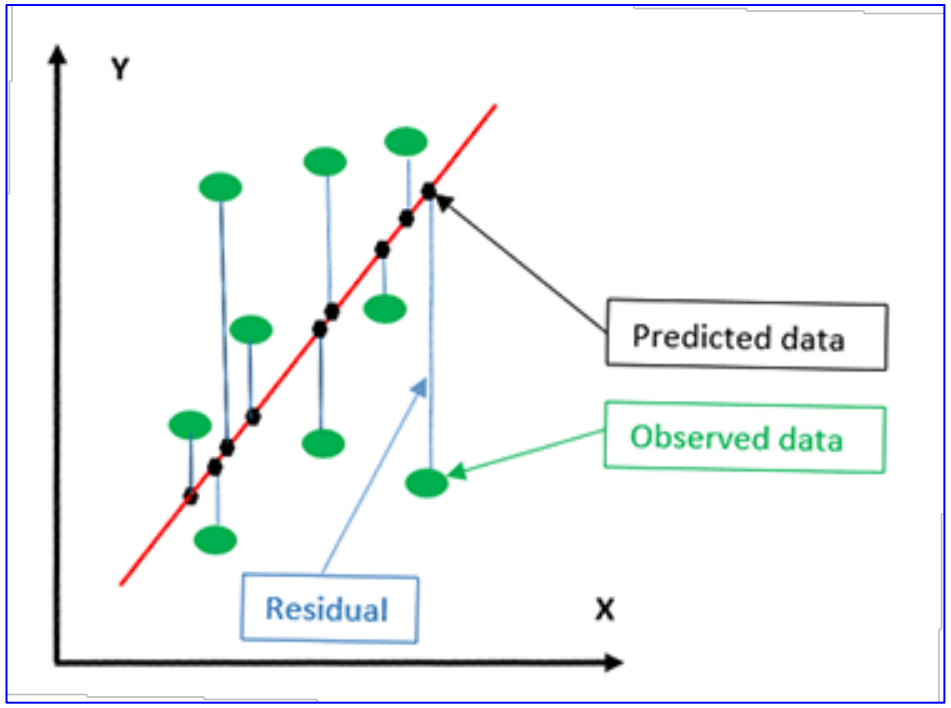
where

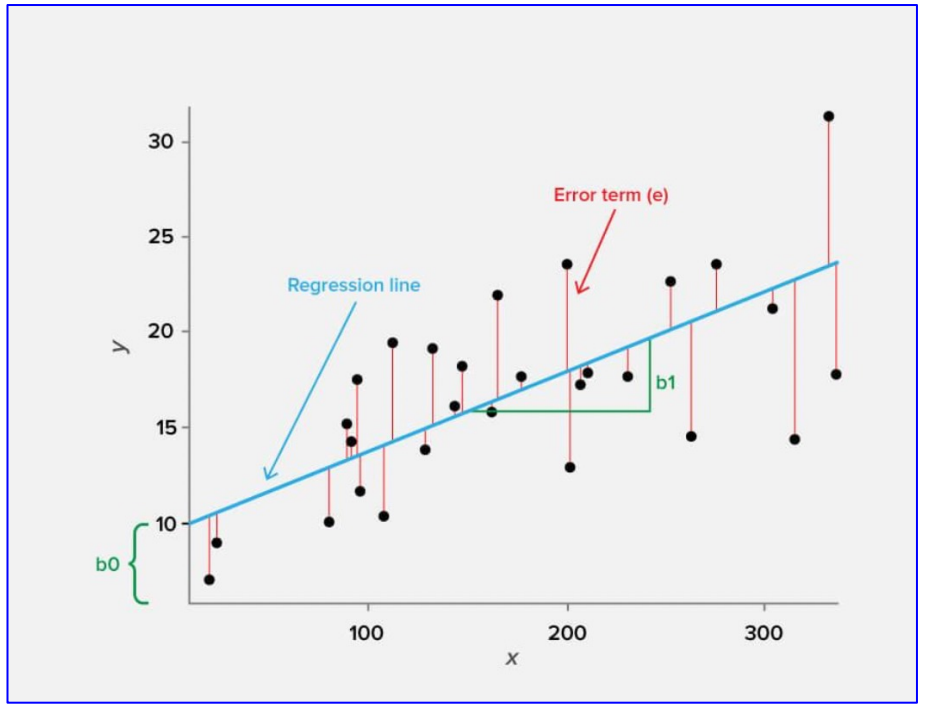
$$\Delta w_i = \eta (t - o) x_i$$

learning rate — target value — perceptron output — input value

Linear Regression

- Linear Regression is a statistical procedure that determines the equation for the straight line that best fits a specific set of data.



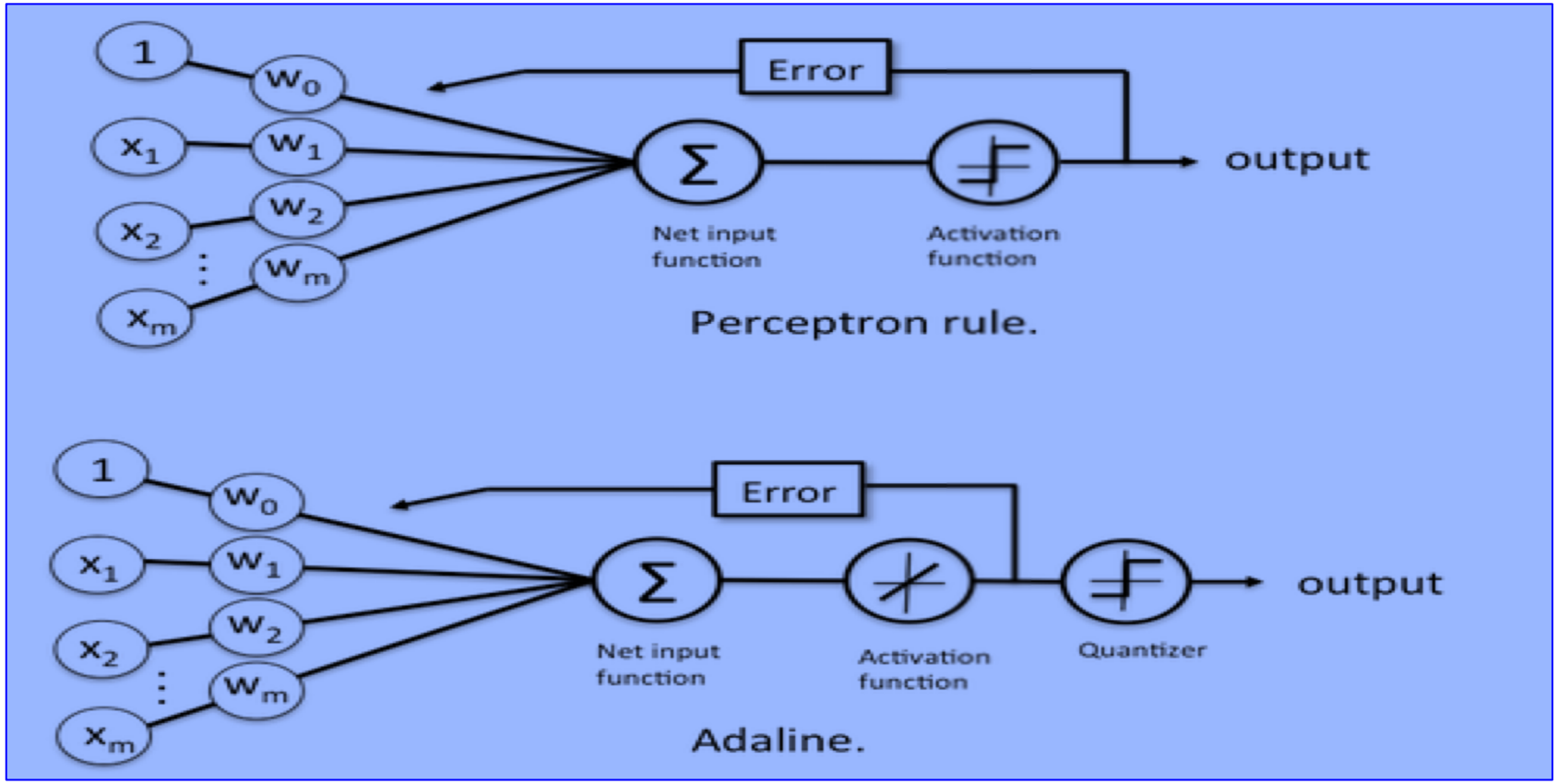


ADALINE (Adaptive Linear Neuron)

ADALINE is an early single-layer artificial neural network based on Least Mean Squares (LMS) algorithms. 最小均方算法

It was invented in 1960 by Stanford University science professor Bernard Widrow and his first Ph.D. student, Ted Hoff.

Perceptron and ADALINE



In the perceptron, we use the predicted class labels to update the weights, and in ADALINE, we use output to update, it tells us by "how much" we were right or wrong

Widrow Hoff Learning Algorithm

- Also known as **Delta Rule**. It follows gradient descent rule for linear regression. It updates the connection weights with the difference between the target and the output value. It is the least mean square learning algorithm falling under the category of the supervised learning algorithm.

- This rule is followed by ADALINE (ADAptive LInear NEuron or NEural Networks) and **MADALINE**. Unlike Perceptron, the iterations of Adaline networks do not always stop, but it converges by reducing the least mean square error. MADALINE is a network of more than one ADALINE.

Delta Learning Rule

- The motive of the delta learning rule is to minimize the error between the output and the target vector. The weights in ADALINE networks are updated by:

  Least Mean Square error (LMS) = $(t- O)^2$,

  ADALINE converges when the least mean square error is reached.

- Learning is an optimization search problem in weight space

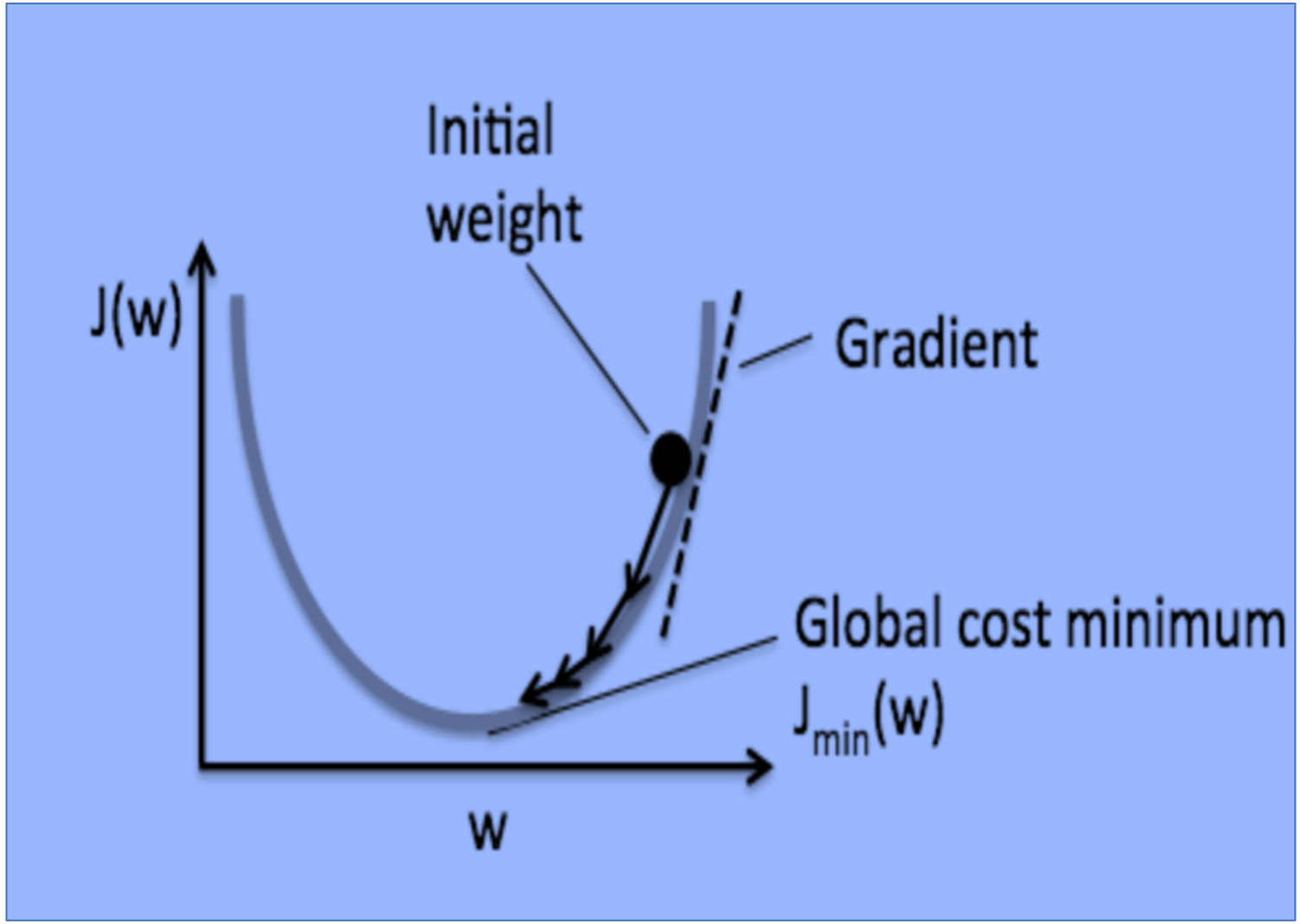
$$\Delta w_i = \eta(t-o)x_i$$

LMS Gradient Descent

Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. But if we instead take steps proportional to the positive of the gradient, we approach a local maximum of that function; the procedure is then known as gradient ascent. Gradient descent is generally attributed to Cauchy, who first suggested it in 1847, but its convergence properties for non-linear optimization problems were first studied by Haskell Curry in 1944.

Wikipedia



Simple Linear Regression ADALINE Example 1

Problem Definition:

$$\mathbf{X}=(x_0,x_1,x_2,\ldots,xn)^T$$

$$\mathbf{W}=(w_0,w_1,w_2,\ldots,wn)^T$$

Simply Define:

$$y=f(\mathbf{W}^T\mathbf{X})=\mathbf{W}^T\mathbf{X}$$

Simple Linear Regression ADALINE Example 2

LMS Rule:

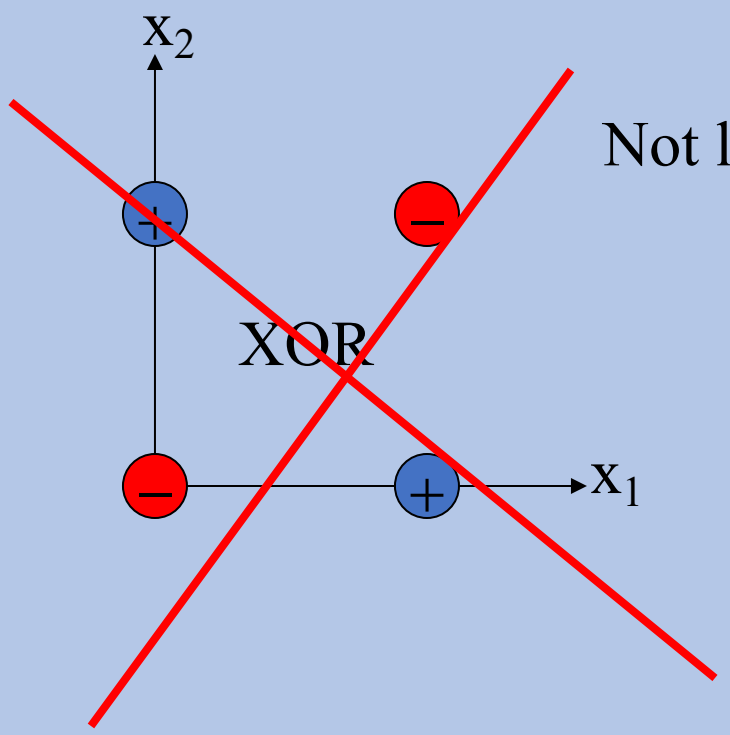
$$\Delta \mathbf{W} = \eta * (d - \mathbf{W}^T \mathbf{X}) \mathbf{X}$$

$$\varepsilon = d - y$$

Normalized LMS Rule:

$$\Delta \mathbf{W} = \eta * \varepsilon * \mathbf{X} / ||\mathbf{X}||^2$$

Perceptron XOR Problem (A Basic Logic Function)



Not linearly separable

Minsky & Papert (1969)

Bad News: Perceptrons can only represent linearly separable functions.

Perceptron XOR Problem Proof

- Consider a threshold perceptron for the logical XOR function (two inputs):

$$w_1 x_1 + w_2 x_2 > T$$

- XOR can be represented as:

	x1	x2	label
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	0

Given our examples, we have the following inequalities for the perceptron:

From (1) $0 + 0 \leq T$ → $T \geq 0$

From (2) $w_1 + 0 > T$ → $w_1 > T$

From (3) $0 + w2 > T$ → $w_2 > T$

From (4) $w1 + w2 \leq T$

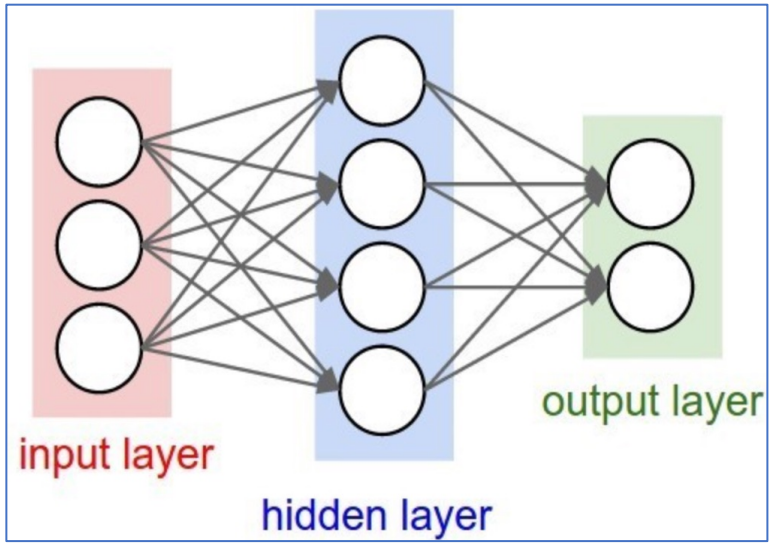
$w1 + w2 > 2T$

contradiction

So, XOR is not linearly separable

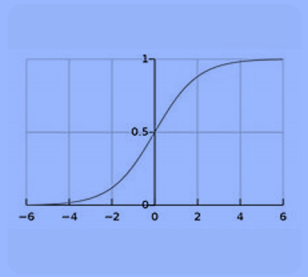
3 Key Network Components of Neural Network

- Network Architecture
- Transfer Function
- Learning Rule



Activation (Transfer) Function Extension

Activation Function



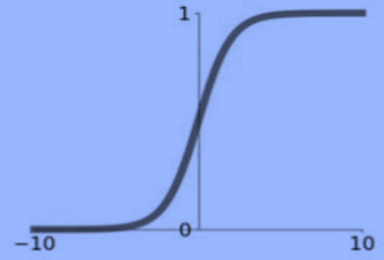
In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only nonlinear activation functions allow such networks to compute nontrivial problems using only a small number of nodes, and such activation functions are called nonlinearities.

Wikipedia

Activation Functions

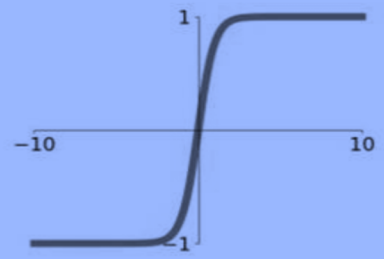
Sigmoid

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



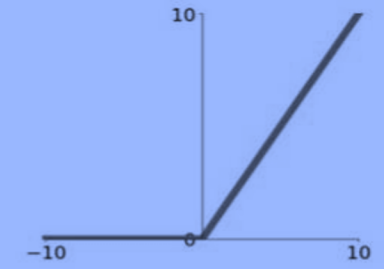
tanh

$$\tanh(x)$$

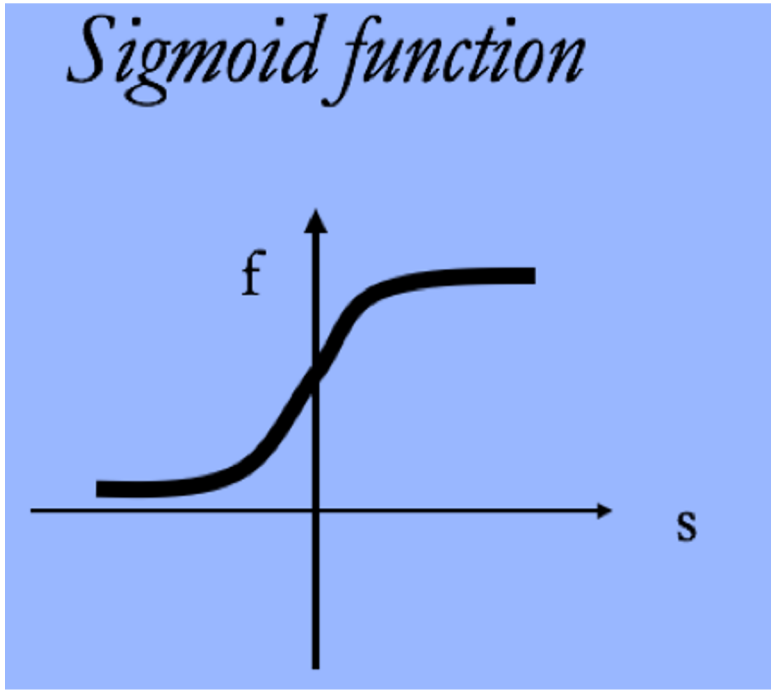


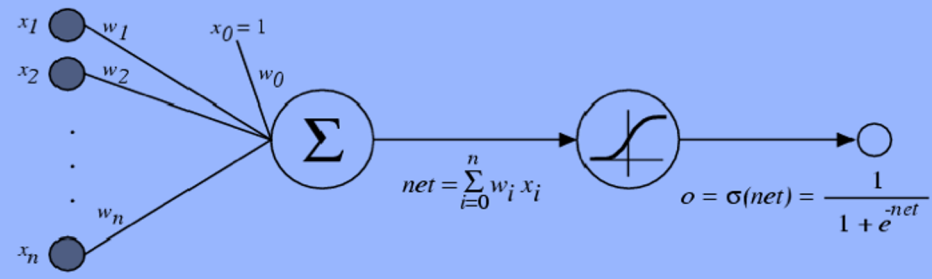
ReLU

$$\max(0, x)$$



Sigmoid Transfer Function





$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

Sigmoid Unit

Hidden Layers for NN

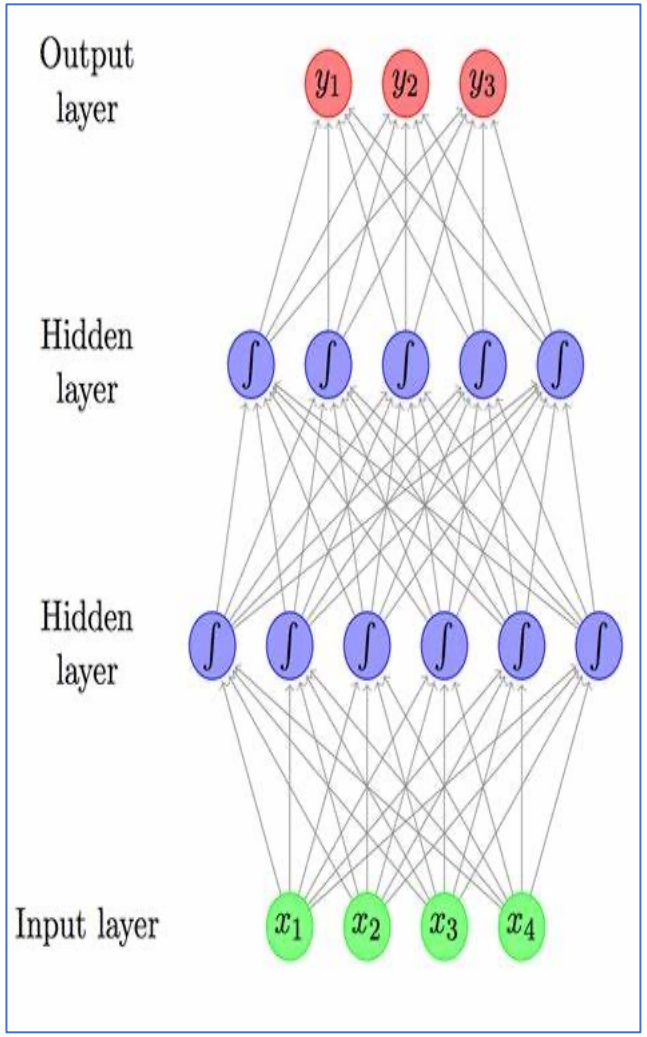
Theoretical answer:

- A neural network with 1 hidden layer is a **universal function approximator**
- Cybenko (1989): For any continuous function g(**x**), there exists a 1-hidden-layer neural net $h_\theta(\mathbf{x})$ s.t. $|h_\theta(\mathbf{x}) - g(\mathbf{x})| < \epsilon$ for all **x**, assuming sigmoid activation functions

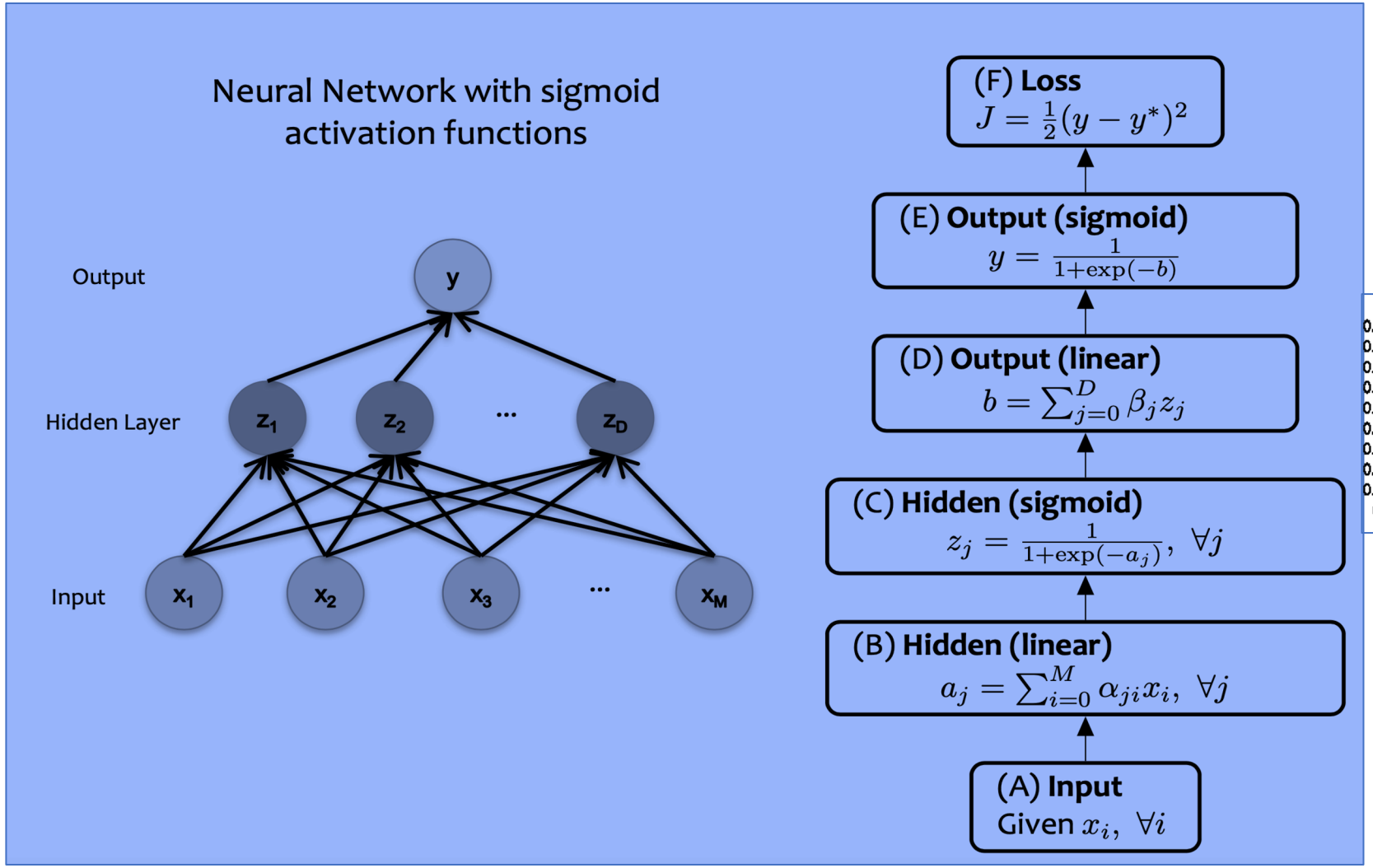
Empirical answer:

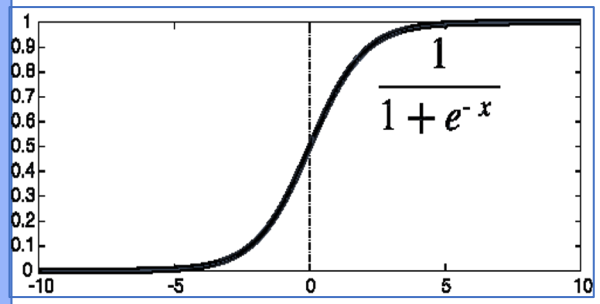
- Before 2006: "Deep networks (e.g. 3 or more hidden layers) are too hard to train"
- After 2006: "Deep networks are easier to train than shallow networks (e.g. 2 or fewer layers) for many problems"

Big caveat: You need to know and use the right tricks.



NN with Sigmoid Hidden Layer

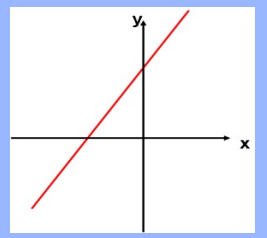


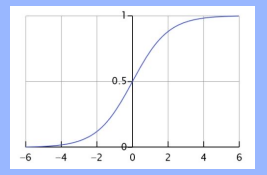


Loss (Objective) Functions

Objective Functions for NNs

1. Quadratic Loss:
   - the same objective as Linear Regression
   - i.e. mean squared error
2. Cross-Entropy:
   - the same objective as Logistic Regression
   - i.e. negative log likelihood
   - This requires probabilities, so we add an additional "softmax" layer at the end of our network





	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

Total-Sum-Squared-Error (TSSE)

$$TSSE = \frac{1}{2} \sum_{patterns} \sum_{outputs} (desired - actual)^2$$

Root-Mean-Squared-Error (RMSE)

$$RMSE = \sqrt{\frac{2 * TSSE}{\#\ patterns * \#\ outputs}}$$

Mean Squared Error (MSE)

$$Loss = \frac{1}{n} \sum_{i=1}^{n} (f_i - y_i)^2$$

Mean Absolute Error (MAE)

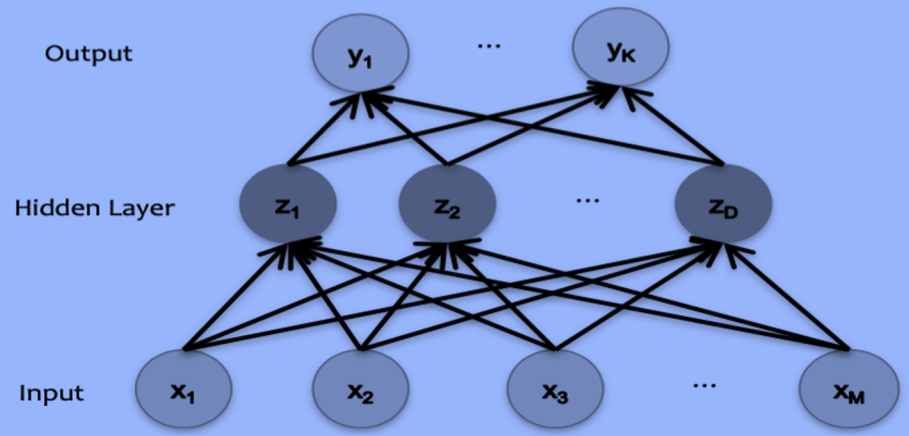
$$Loss = \frac{1}{n} \sum_{i=1}^{n} |f_i - y_i|$$

Multiple Classification with Softmax (Probability)

Multi-Class Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$



(F) **Loss**
$J = \sum_{k=1}^{K} y_k^* \log(y_k)$

(E) **Output (softmax)**
$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$

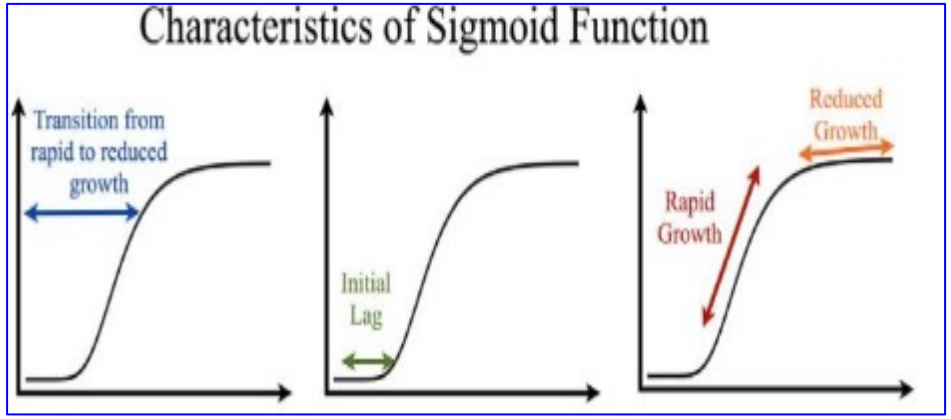
(D) **Output (linear)**
$b_k = \sum_{j=0}^{D} \beta_{kj} z_j \ \forall k$

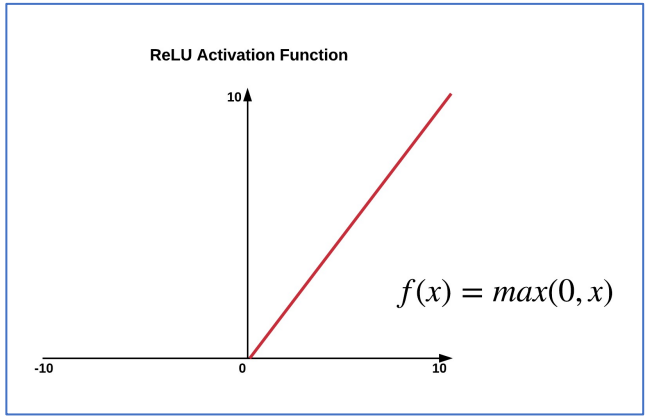
(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

Sigmoid and ReLU





Sigmoid: not blowing up activation

Relu : not vanishing gradient

Relu : More computationally efficient to compute than Sigmoid like functions since Relu just needs to pick max(0, *x*) and not perform expensive exponential operations as in Sigmoids

Relu : In practice, networks with Relu tend to show better convergence performance than sigmoid.

BP Algorithm with Sigmoid Transfer Function

1. Set initial weight and bias a random small number

$$w_{ij}^{k}(t),\quad \theta_{i}^{k}(t),(k=1,\ldots,m;i=1,\ldots,p_k;j=1,\ldots,p_{k-1};t=0)$$

2. Select a sample x from the N input samples and corresponding output y

$$\boldsymbol{x}=[x_1,x_2,\ldots,x_{p1}]^{\mathrm{T}} \qquad y_i \quad (i=1,\ldots,p_m)$$

3. Calculate all the outputs in all layer

$$y_i^k \quad (i=1,\ldots,p_k;\ \ k=1,\ldots,m)$$

4 Calculate the output error

$$e_i=y_i-y_i^m \quad (i=1,\ldots,p_m)$$

BP Algorithm with Sigmoid Transfer Function

- 5. Update the weight for both output and hidden layers

$$\Delta w_{ij}^{k} = -\alpha d_i^k y_j^{k-1}$$

$$d_i^m = y_i^m (1 - y_i^m)(y_i^m - y_i)$$

$$d_i^k = y_i^k (1 - y_i^k) \sum_{l=1}^{p_{k+1}} w_{li}^{k+1} d_l^{k+1}$$

- 6 t=t+1, repeat 2-5 until the error rate for the N samples reaches the set target or stops decreasing .

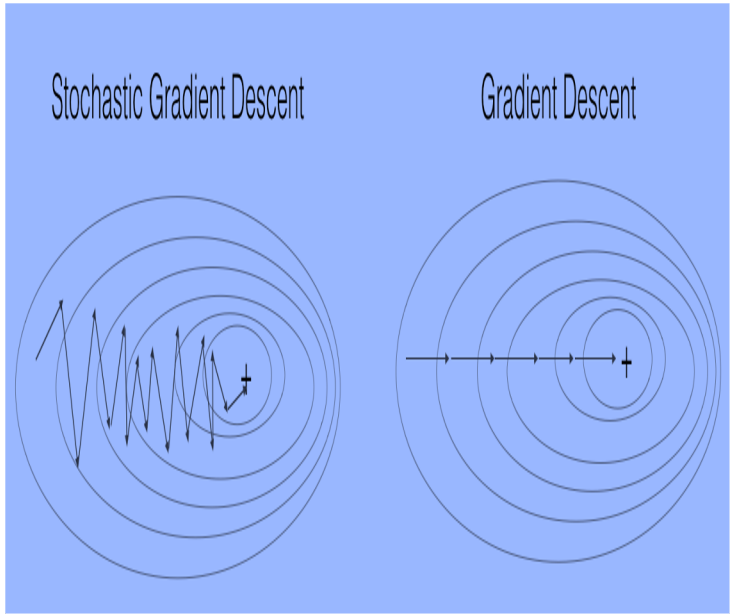
Gradient Descent Vs Stochastic Gradient Descent

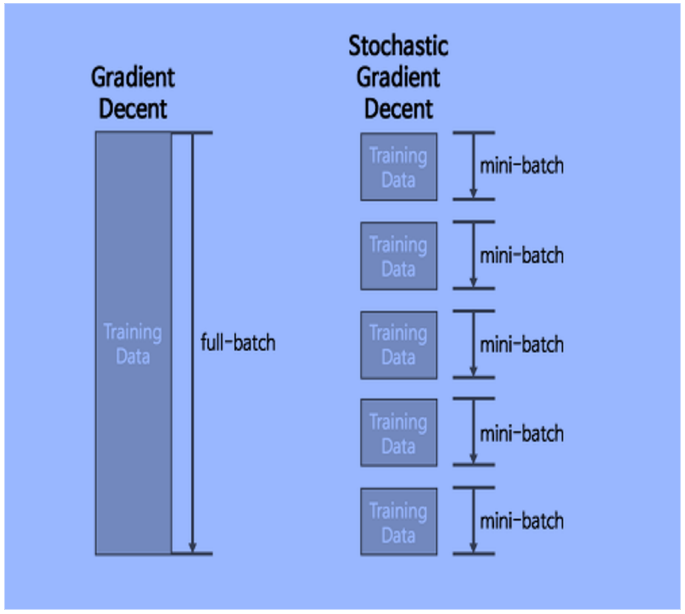
GD: This will update the weights only after calculating the mean loss of all the samples. Hence in such situation it will become very costly operation and it will converge very slowly.

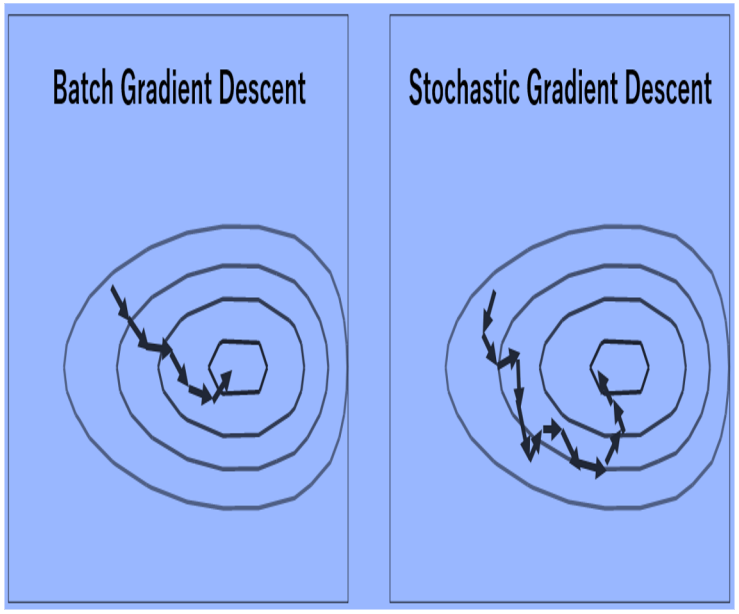
Batch GD/Mini-Batch GD: Alternative of GD. Selects a batch size and overcome the above said problem of GD. But still executes in batch.

SGD: Update weights with every sample and converge very fast.

Gradient Descent Vs Stochastic Gradient Descent







Support Vector Machine

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Developed at AT&T Bell Laboratories by Vapnik with colleagues (Boser et al., 1992, Guyon etal., 1993, Vapnik et al., 1997), it presents one of the most robust prediction methods, based on the statistical learning framework or VC theory proposed by Vapnik and Chervonenkis (1974) and Vapnik (1982, 1995). Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

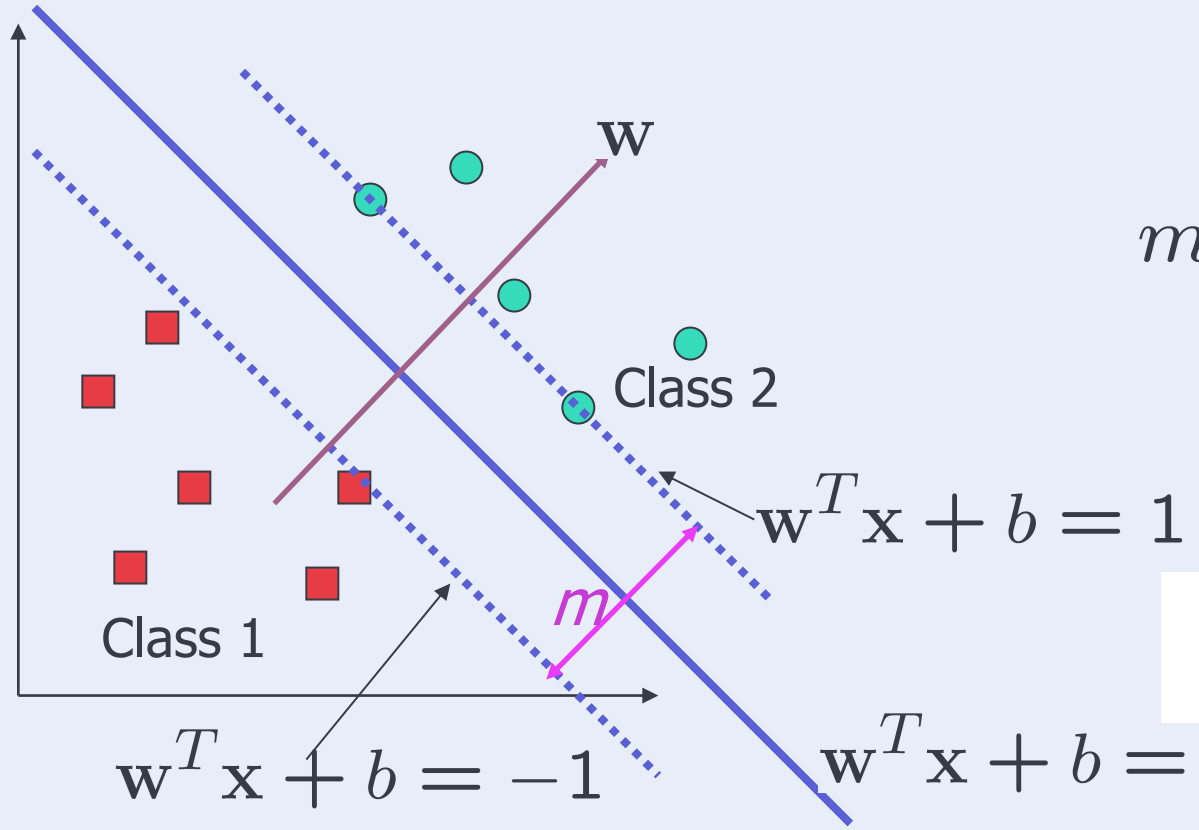
Wikipedia

Margin m

The decision boundary should be as far away from the data of both classes as possible

We should maximize the margin *m: smallest distance from observations to hyperplane*

Distance between the origin and the line $\mathbf{w}^t\mathbf{x}=-b$ is $b/||\mathbf{w}||$



$$m = \frac{2}{||\mathbf{w}||}$$

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b \geq +1, \quad \Rightarrow \quad y_i = +1$$
$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}_i + b \leq -1, \quad \Rightarrow \quad y_i = -1$$

SVM Unique Solution

Optimization Problem

(Cortes and Vapnik, 1995)

Constrained optimization:

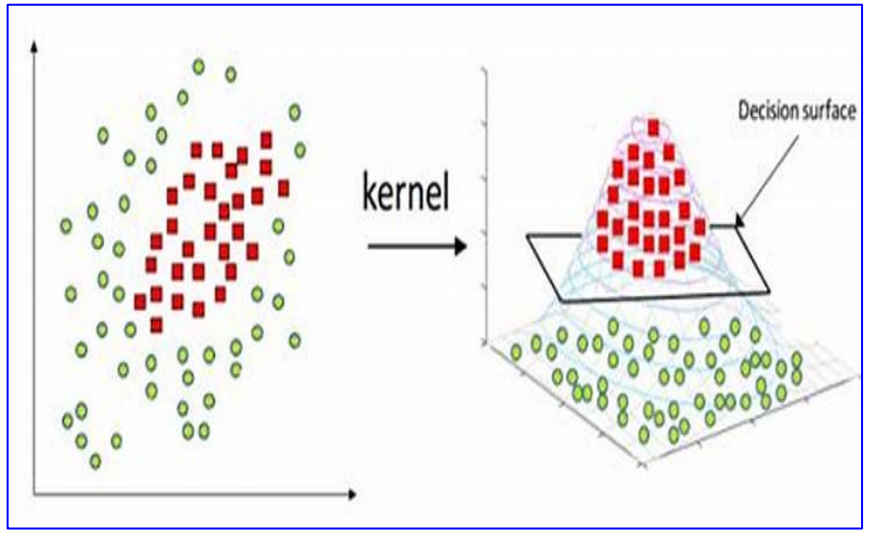
$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i$$

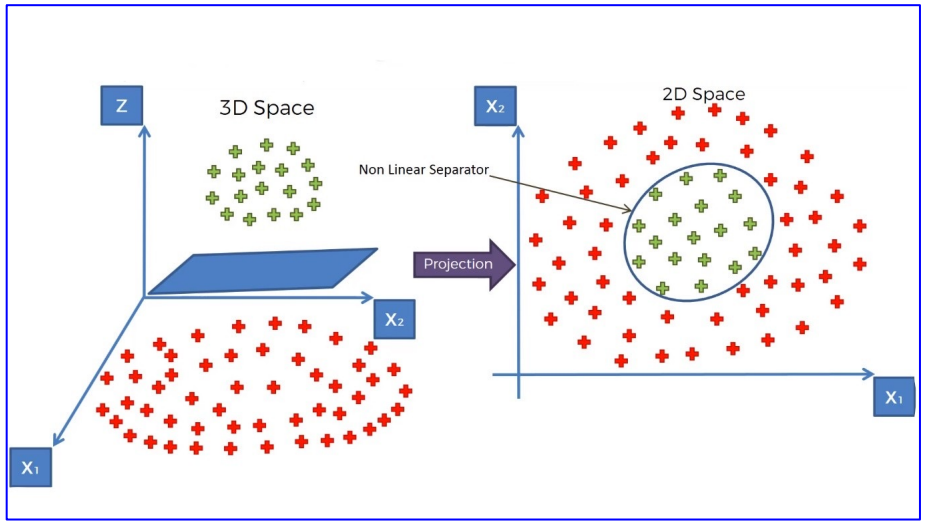
$$\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \ \wedge \ \xi_i \geq 0, i \in [1, m].$$

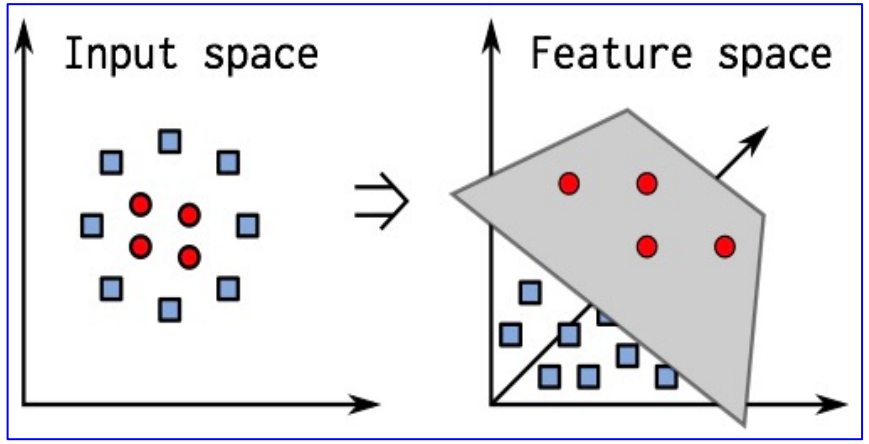
Properties:

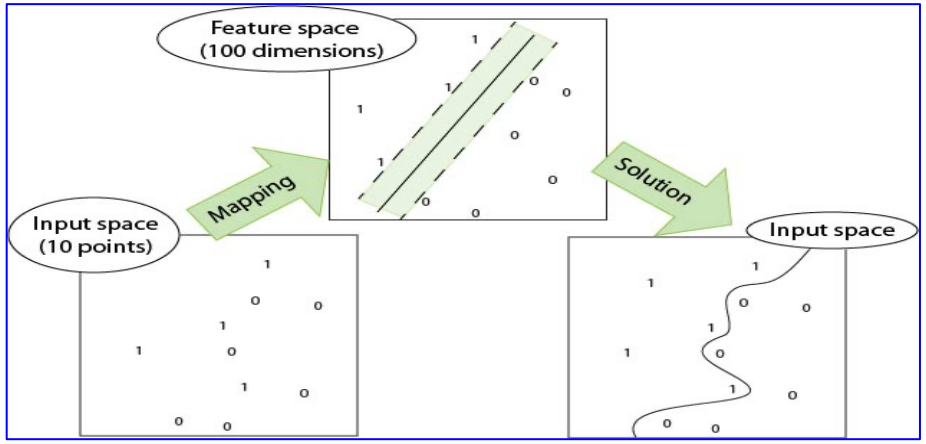
- $C \geq 0$ trade-off parameter.
- Convex optimization.
- Unique solution.

Non-Linear SVM Classifier with Kernel Mapping

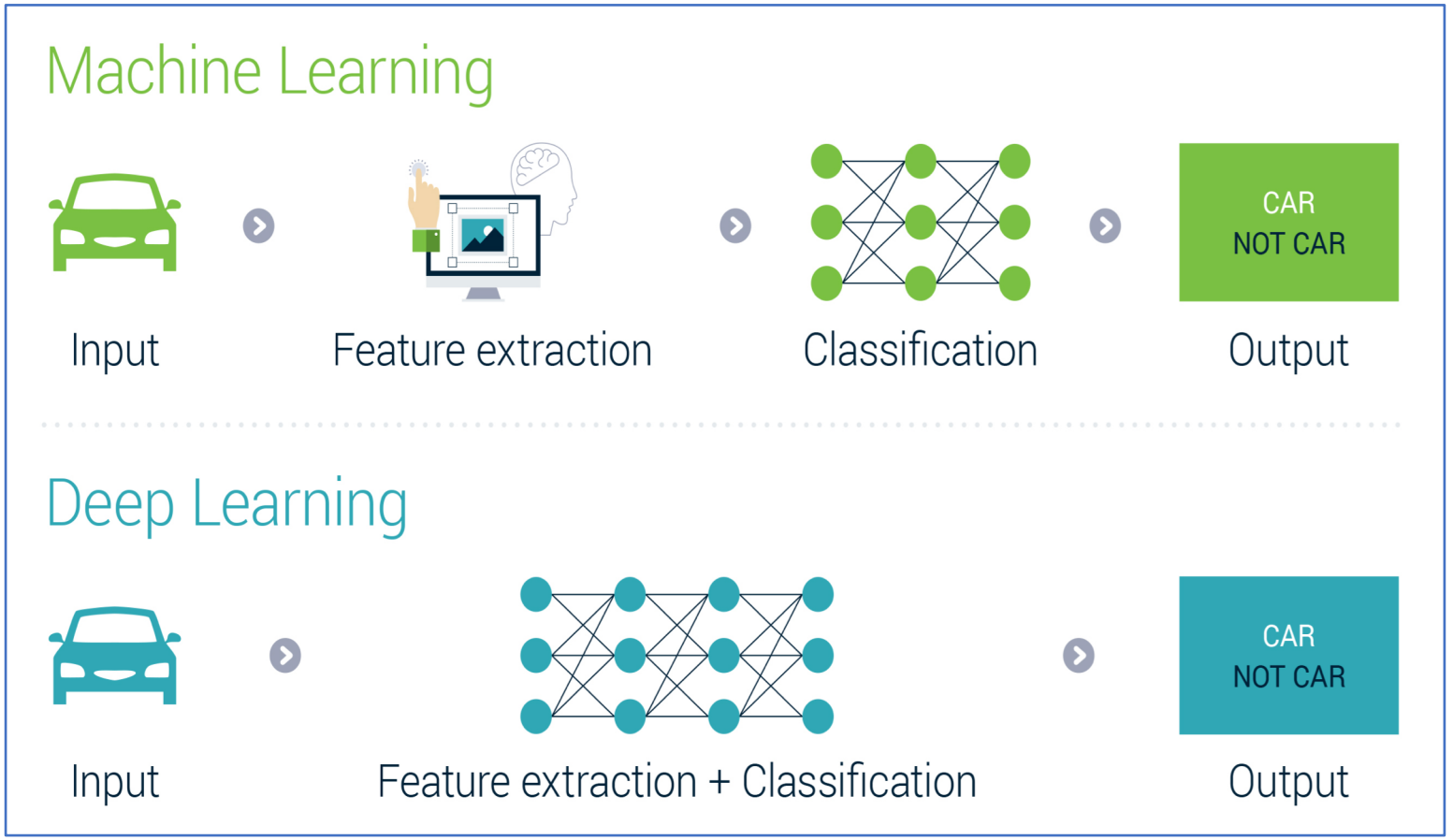




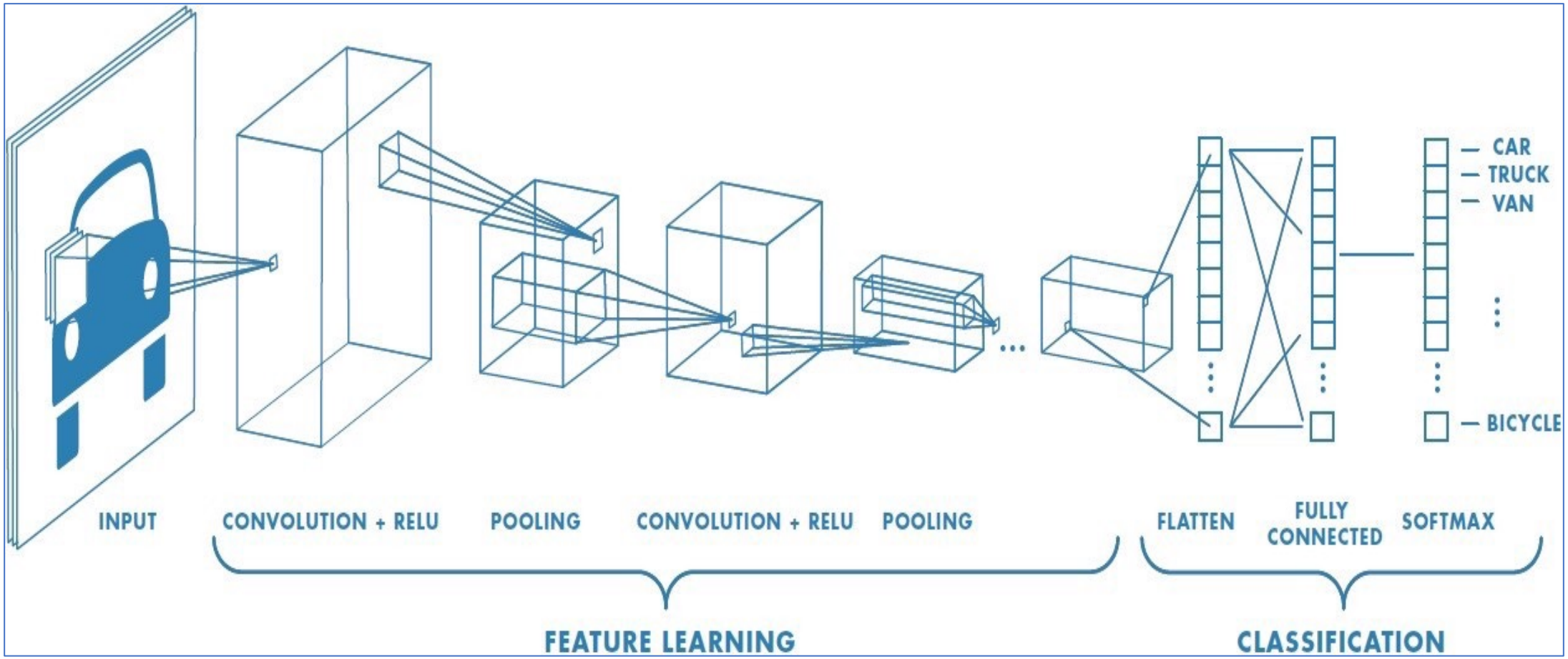




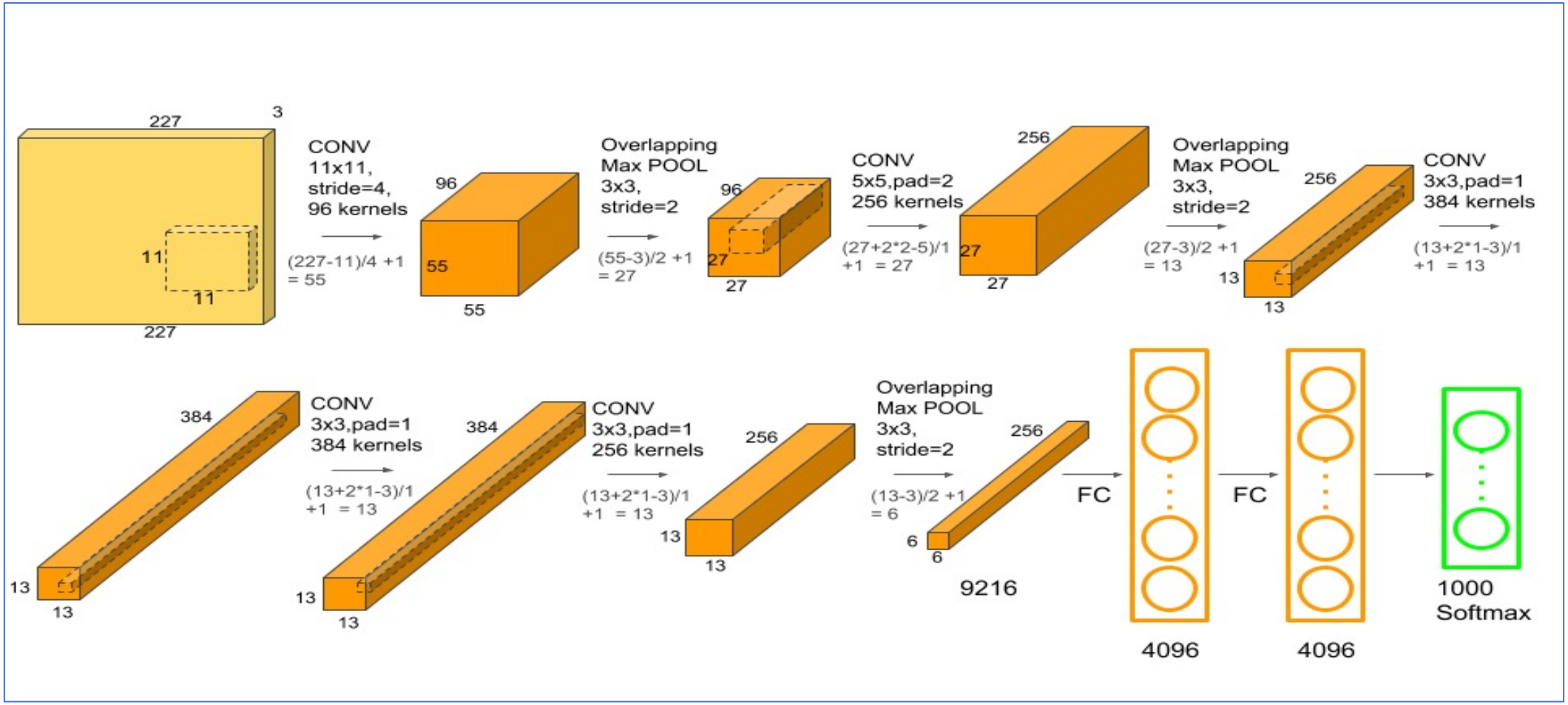
Machine Learning and Deep Learning



Convolutional Neural Network



Output Size AlexNet Computing



Number of Parameters of a Conv Layer

W_c = Number of weights of the Conv Layer.

B_c = Number of biases of the Conv Layer.

P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

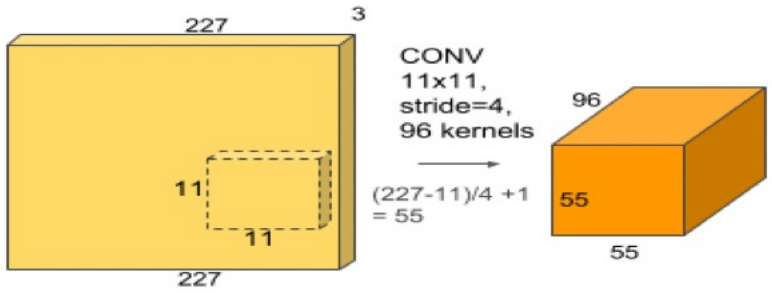
C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

In a Conv Layer, the depth of every kernel is always equal to the number of channels in the input image. So every kernel has $K^2 \times C$ parameters, and there are N such kernels. That's how we come up with the above formula.

Example: In AlexNet, at the first Conv Layer, the number of channels (C) of the input image is 3, the kernel size (K) is 11, the number of kernels (N) is 96. So the number of parameters is given by

$$W_c = 11^2 \times 3 \times 96 = 34,848$$
$$B_c = 96$$
$$P_c = 34,848 + 96 = 34,944$$



Number of Parameters of a Fully Connected (FC) Layer connected to a Conv Layer

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer.

B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer.

P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer.

F = Number of neurons in the FC Layer.

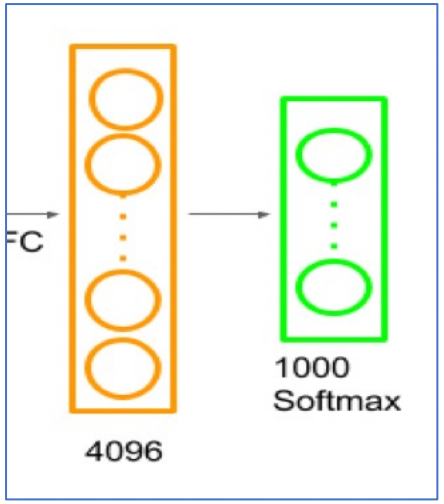
F_{-1} = Number of neurons in the previous FC Layer.

$$W_{ff} = F_{-1} \times F$$
$$B_{ff} = F$$
$$P_{ff} = W_{ff} + B_{ff}$$

In the above equation, $F_{-1} \times F$ is the total number of connection weights from neurons of the previous FC Layer the neurons of the current FC Layer. The total number of biases is the same as the number of neurons (F).

Example: The last fully connected layer of AlexNet is connected to an FC Layer. For this layer, $F_{-1} = 4096$ and $F = 1000$. Therefore,

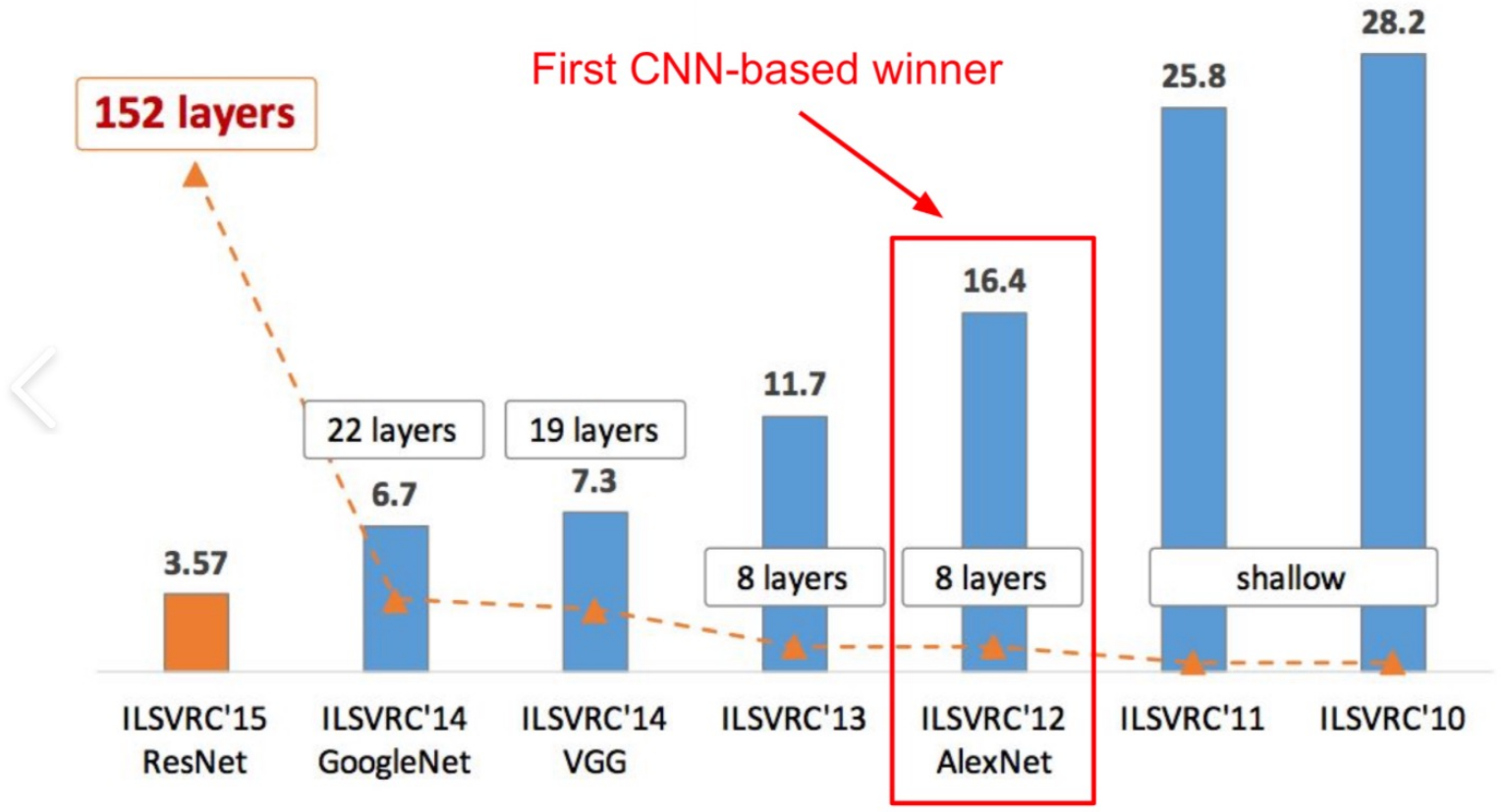
$$W_{ff} = 4096 \times 1000 = 4,096,000$$
$$B_{ff} = 1,000$$
$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$



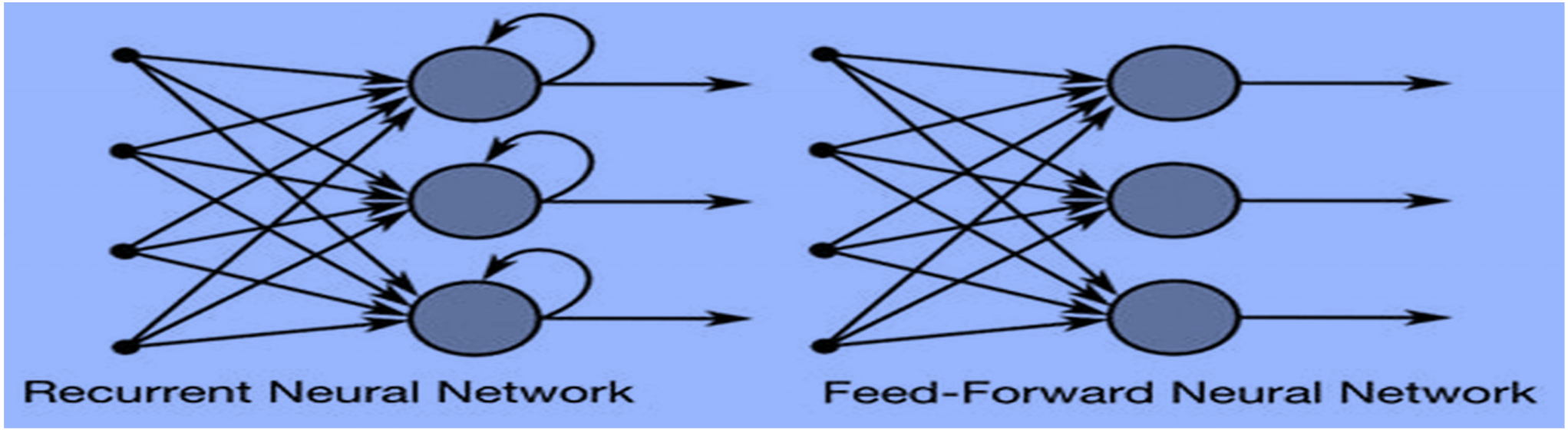
No. of Parameters and Tensor Sizes in AlexNet

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

Some Popular Algorithms



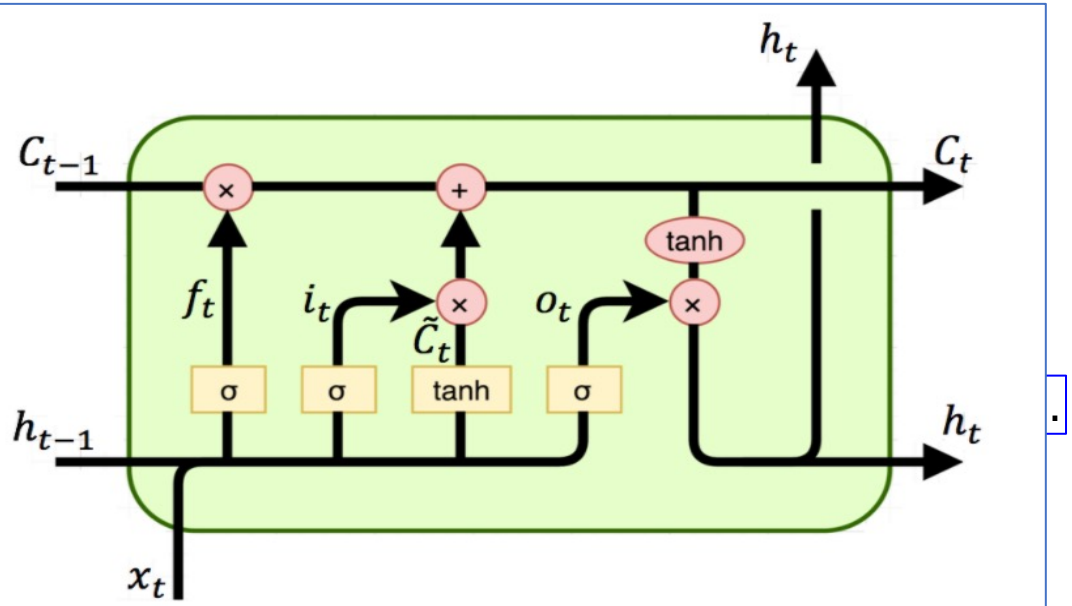
RNN and LSTM



A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

Long Short-Term Memory

遗忘门 (forget gate)、输入门 (input gate)、输出门 (output gate)



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

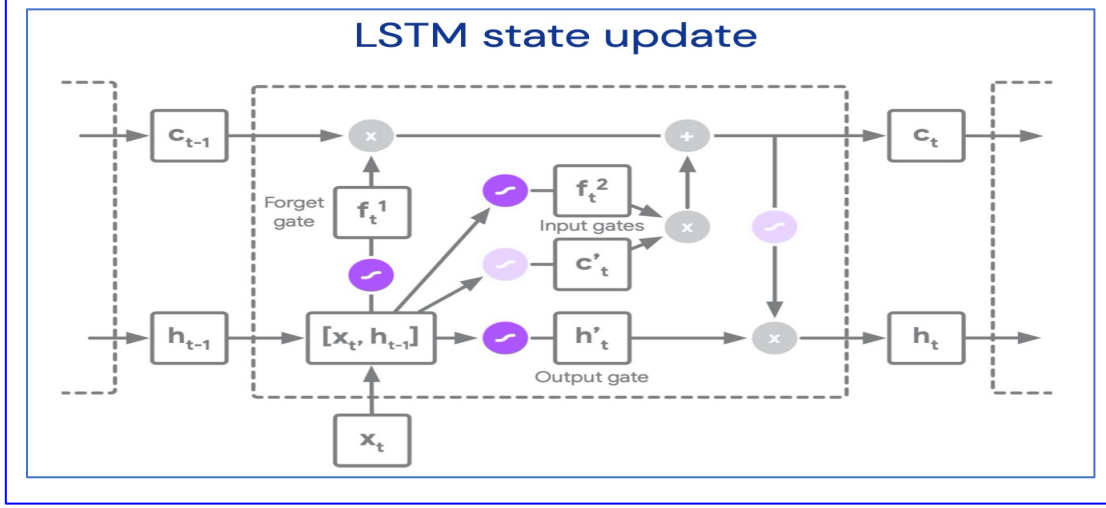
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

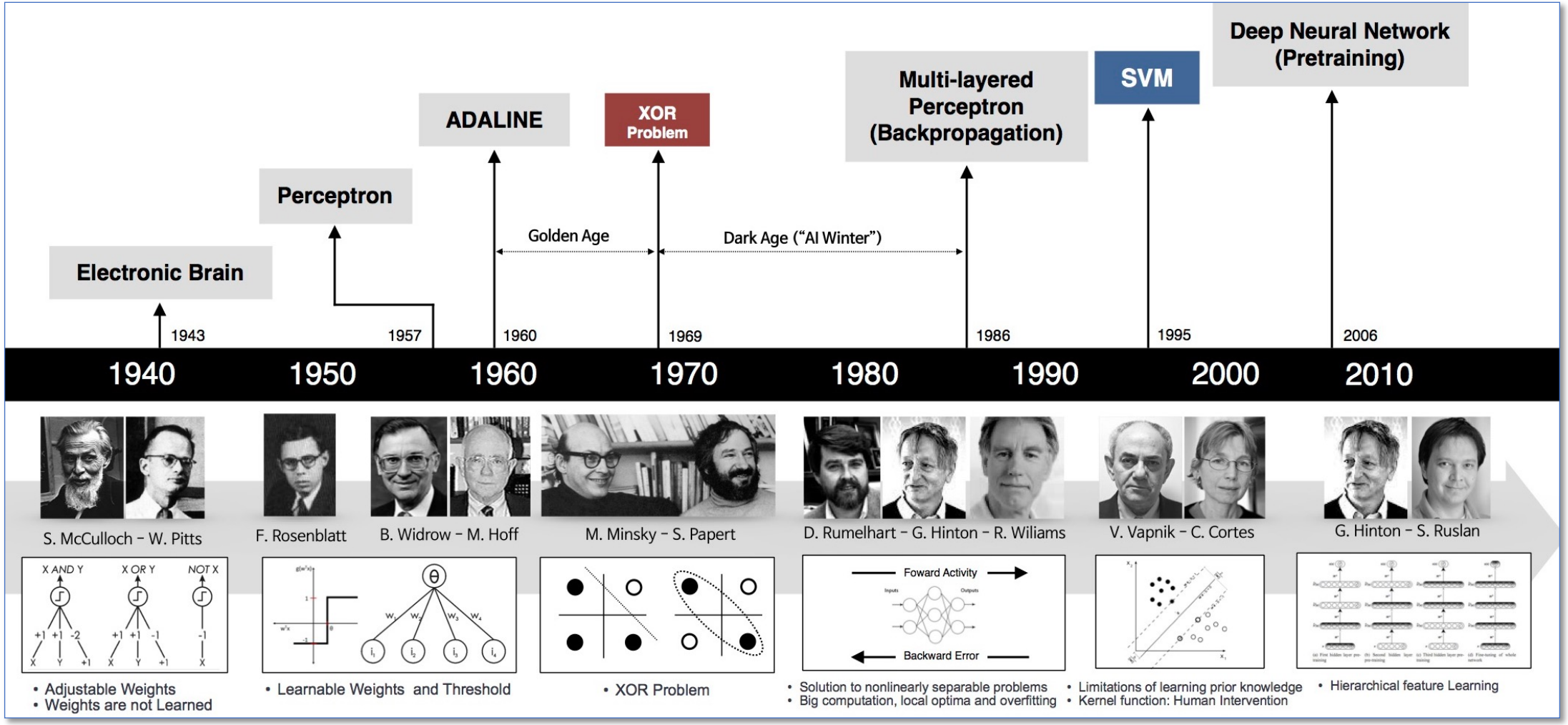
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$



- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in \mathbb{R}^h$: forget gate's activation vector
- $i_t \in \mathbb{R}^h$: input/update gate's activation vector
- $o_t \in \mathbb{R}^h$: output gate's activation vector
- $h_t \in \mathbb{R}^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in \mathbb{R}^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector

Introduction to AI



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

cnMed

CS 103 - 16

Introduction to AI Review

Jimmy Liu 刘江

2022-12-30