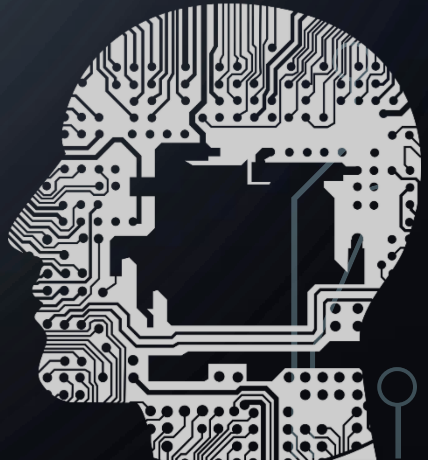
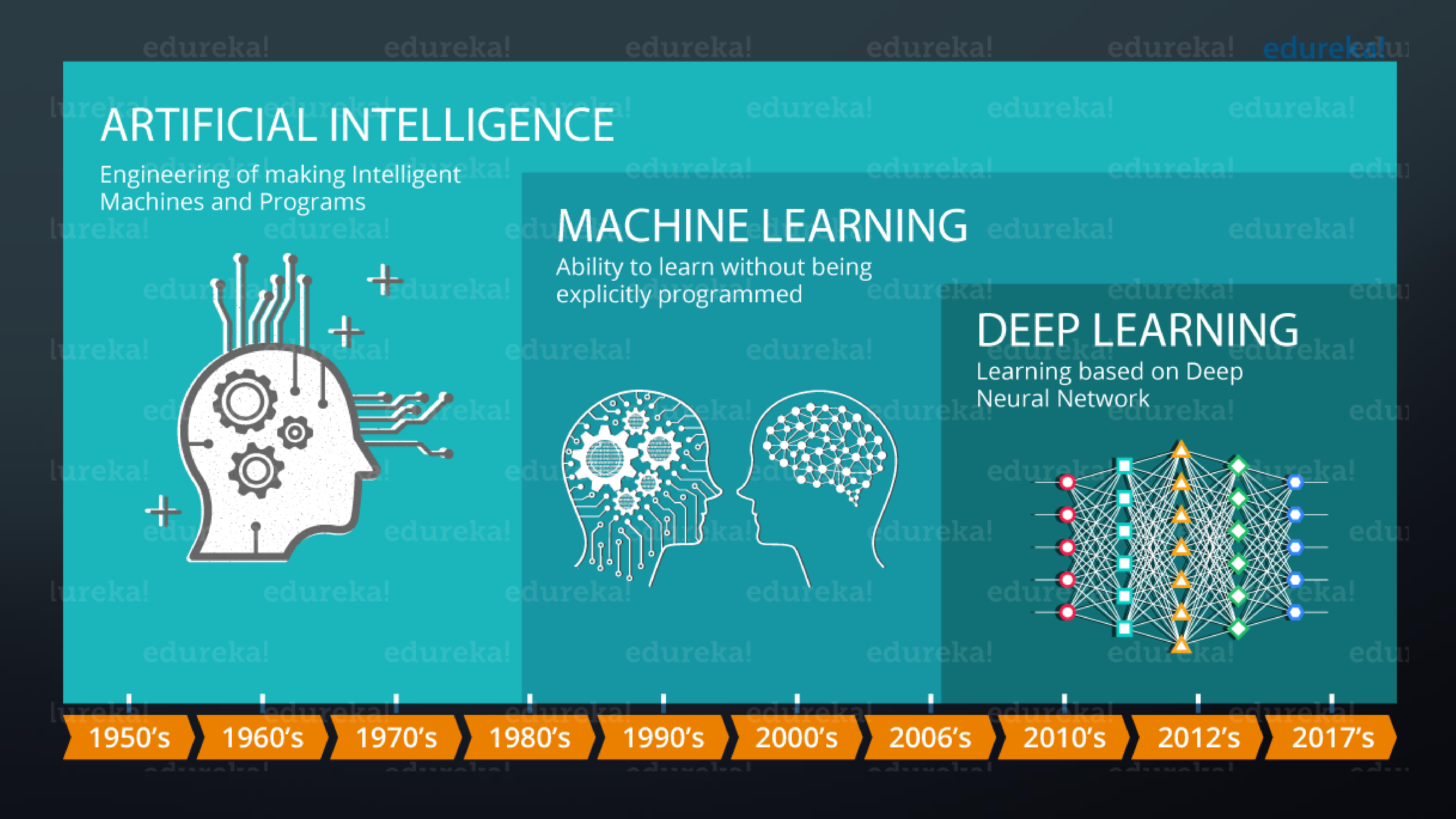
Machine learning

張傳育 (Chuan-Yu Chang Ph.D.)

國立雲林科技大學 資訊工程系 特聘教授

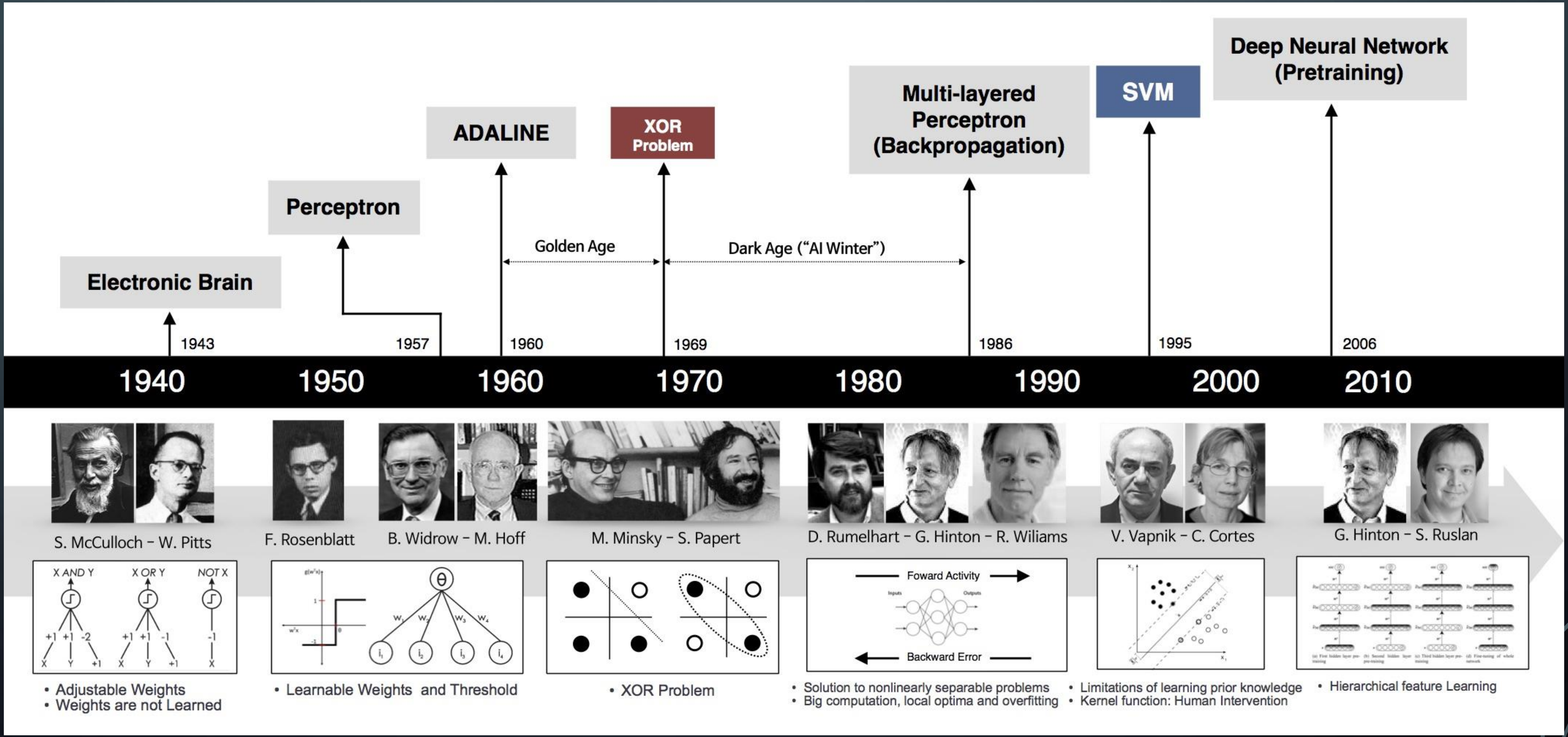
chuanyu@yuntech.edu.tw





ARTIFICIAL INTELLIGENCE
Engineering of making Intelligent Machines and Programs
MACHINE LEARNING
Ability to learn without being explicitly programmed
DEEP LEARNING
Learning based on Deep Neural Network
1950's
1960's
1970's
1980's
1990's
2000's
2006's
2010's
2012's
2017's

Milestones in the Development of Neural Networks



https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Historical notes

- McCulloch and Pitts (1943)
  - No training to neurons.
  - Act as certain logic functions.
- Hebb (1949)
  - Based on a neurobiological viewpoint, describes a learning process.
  - Hebb stated that information is stored in the connections of neurons and postulated a learning strategy for adjustment of the connection weight.
- Rosenblatt (1958)
  - Proposed the original concept of the perceptron

Historical Notes (Cont.)

- Widrow and Hoff (1960)
  - The Adaline (adaptive linear element) trained by the LMS learning rule.
- Minsky and Papert (1969)
  - Slowed down neural network research in 1969.
  - Perceptrons have limited capabilities, the XOR problem.
- Kohonen and Anderson (1972)
  - Content-addressable associative memories.
- Von der Malsburg (1973)
  - Proposed a cortex model with ability to modify and organize itself.
- Werbos (1974)
  - The first description of the backpropagation algorithm for training multilayer feed-forward perceptrons.

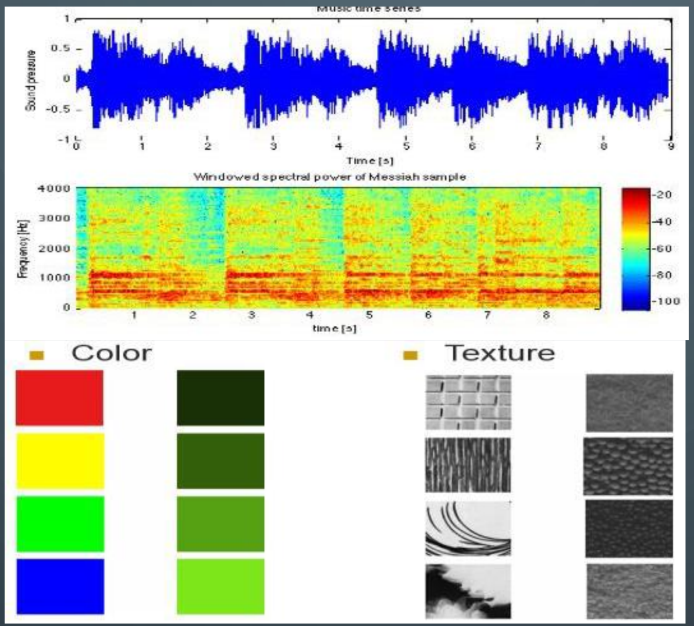
Historical Notes (Cont.)

- Little and Shaw (1975)
  - Use a probabilistic model of a neuron instead of a deterministic one.
- Lee (1975)
  - Presented the fuzzy McCulloch-Pitts neuron mode.
- Amari (1977)
  - Pattern associator. The input pattern induces an appropriates, but different, output pattern.
- Hopfield (1982)
  - Proposed a recurrent neural network
  - The network can store information in a dynamically stable storage and retrieval.
- Kohonen (1982)
  - Presented the self-organizating feature map.
  - It is an unsupervised, competitive learning, clustering network in which only one neuron is "on" at a time.

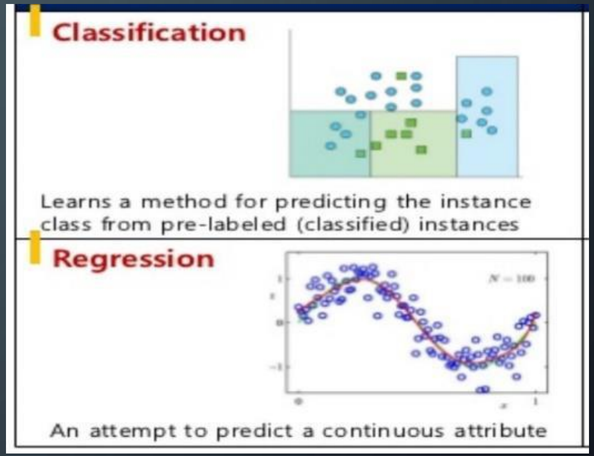
Historical Notes (Cont.)

- Oja (1982)
  - Presented a single linear neuron trained by a normalized Hebbian learning rule that acts as a principal-component analyzer.
  - The neuron is capable of adaptive extracting the 1st principal eignvector from the input data.
- Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams (1986)
  - "Learning Internal Representations by Error Propagation"
  - Multilayer perceptron (MLP) is a fully connected class of feedforward artificial neural network
  - MLP utilizes a supervised learning technique called backpropagation for training
- Carpenter and Grossberg (1987)
  - Developed self-organizing neural networks based adaptive resonance theory (ART)
- Sivilotti, Mahowald, and Mead (1987)
  - The first VLSI realization of neural networks.
- Broomhead and Lowe (1988)
  - First exploitation of radial basis function in designing neural networks.

Traditional Machine Learning



Supervised, Unsupervised, reinforcement learning, active learning, etc



classification or regression, prediction, detection, etc

Real World Data → Feature Representation → Machine Learning Algorithms → Applications

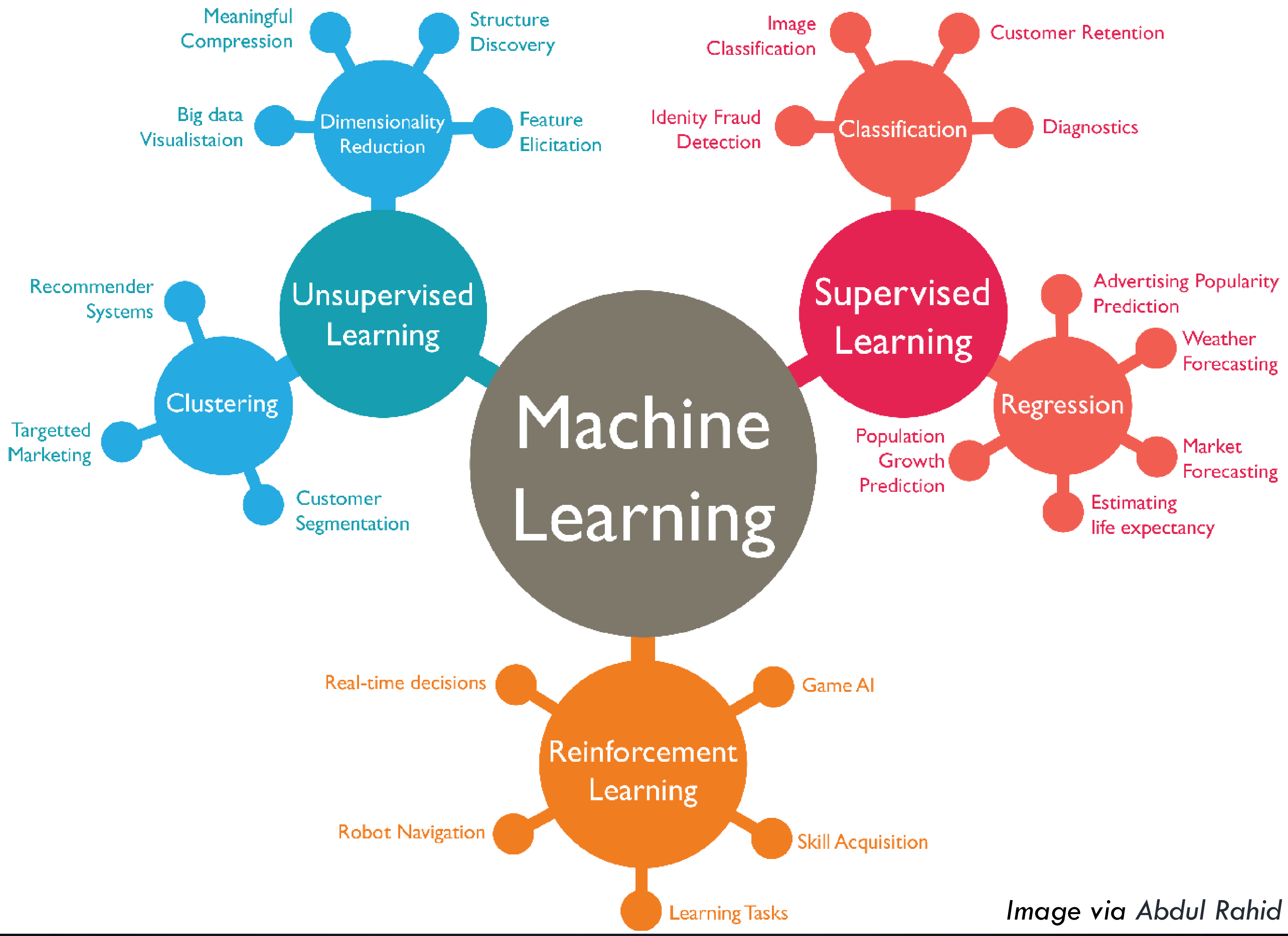
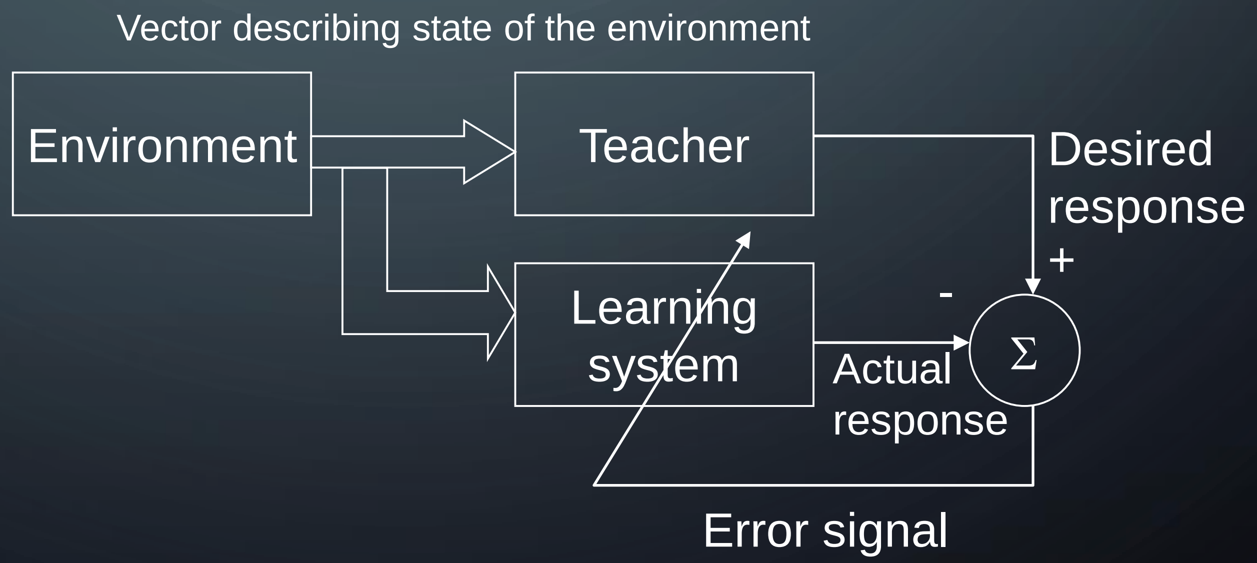


Image via Abdul Rahid

Learning with a teacher

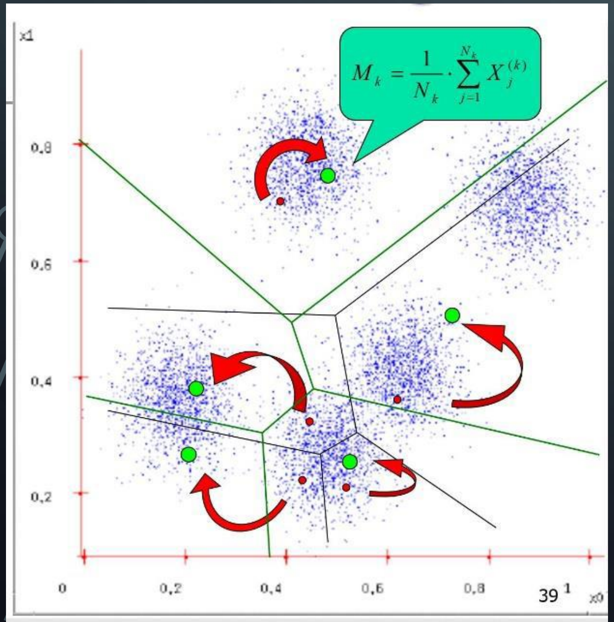
- Supervised learning
  - 藉由input-output examples來訓練網路
  - Error signal：desired response和actual response間的差異
  - 參數的調整是step-by-step反覆的進行。
  - Error-performance surface, error surface
  - Gradient Steepest Descent



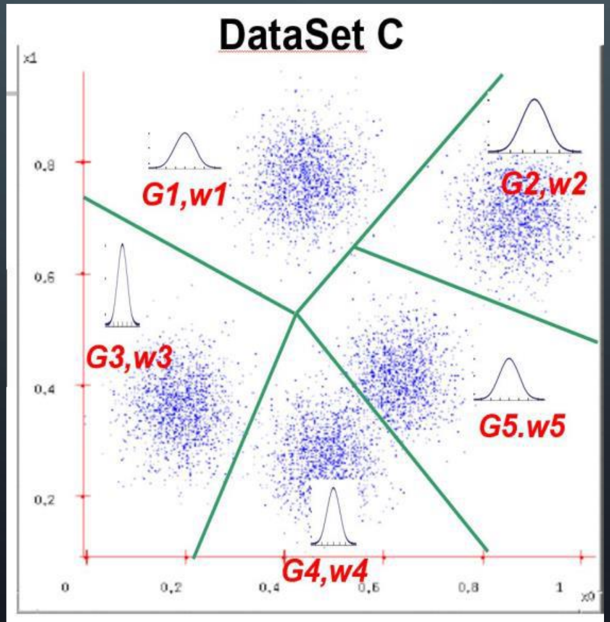
Learning without a teacher

- There is no teacher to oversee the learning process.
  - There are no labeled examples of the function to be learned by the network.
  - 1. Unsupervised learning
    - Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input.

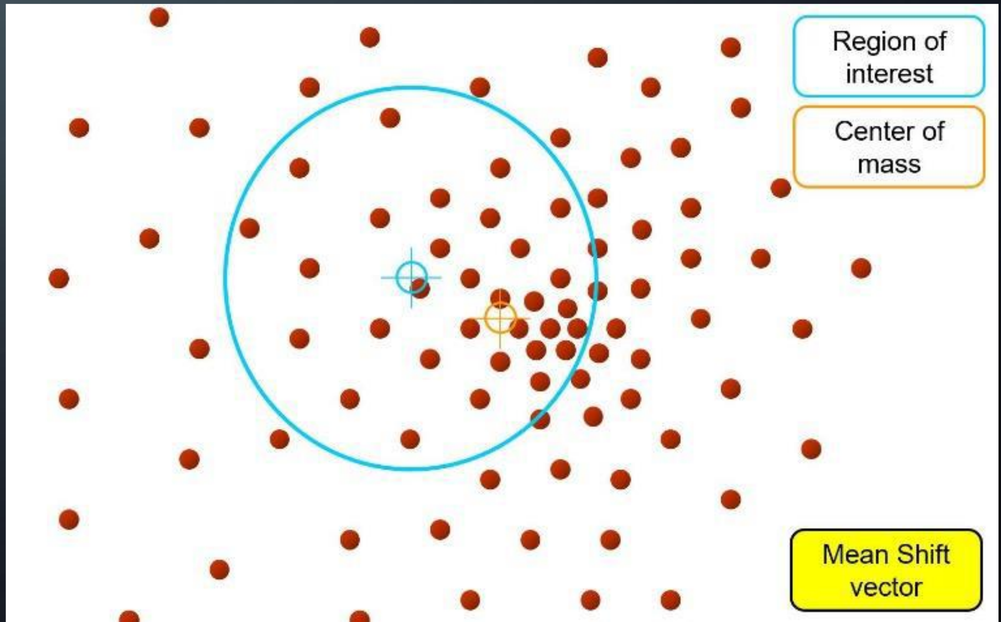
K-Means



Gaussian Mixture



Mean Shift



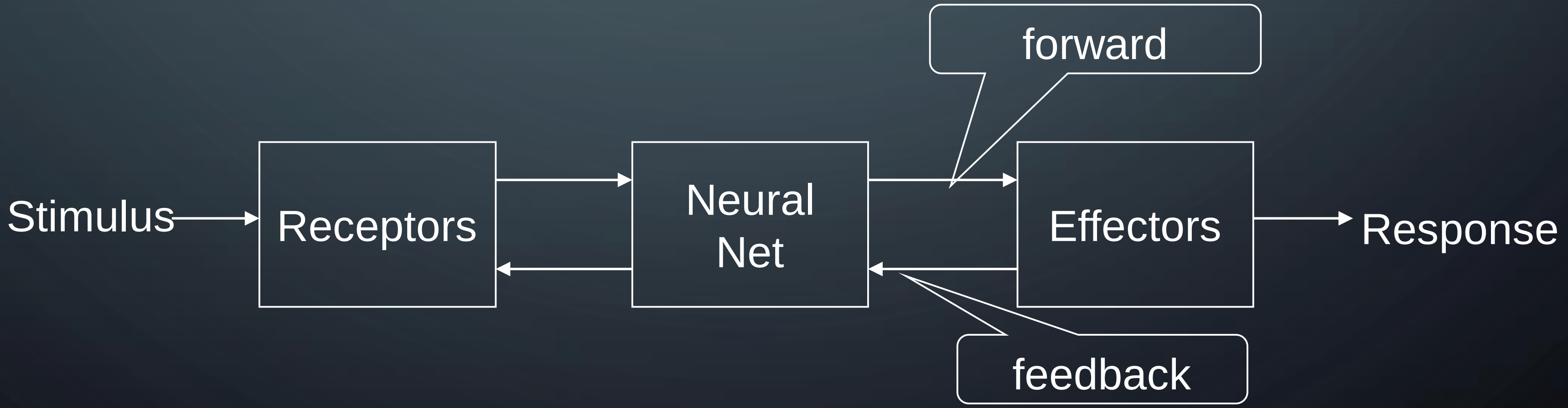
Learning without a teacher

- 2. Reinforcement learning
  - The learning of an input-output mapping is performed through continued interaction with the environment in order to minimize a scalar index of performance.
  - 系統觀察一段時間的環境刺激，最後產生heuristic reinforcement signal.
  - 此種學習的目的在於將一cost-to-go function最小化，亦即將一連串的步驟行動的累積成本期望值最小化。

The agent learn to sense and perturb the state of the environment using its actions to derive maximal reward.
The reward signal is the value function, which faithfully captures the ‘goodness’ of a state. While the reward signal represents the immediate benefit of being in a certain state, the value function captures the cumulative reward that is expected to be collected from that state on, going into the future. The objective of an RL algorithm is to discover the action policy that maximizes the average value that it can extract from every state of the system.

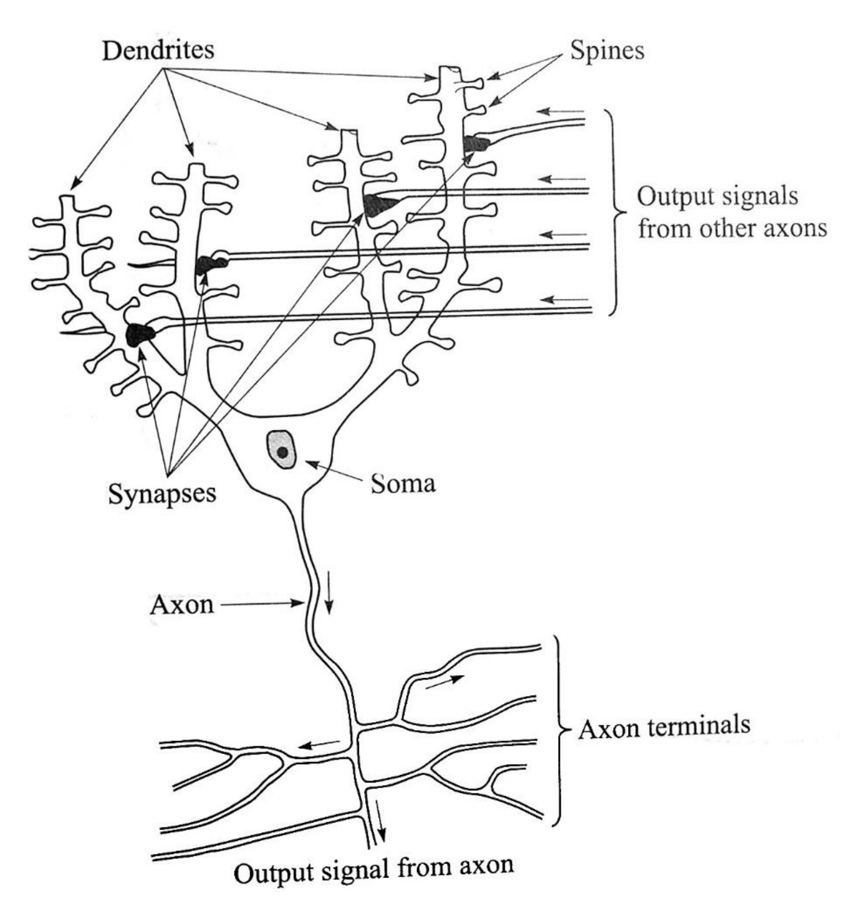
Human Brain

- The *neural (nerve) net* continually receives information, perceives it, and makes appropriate decisions.
- The *receptors* convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural net (brain).
- The *effectors* converts electrical impulses generated by the neural net into discernible responses as system output.



Neurocomputing and Neuroscience

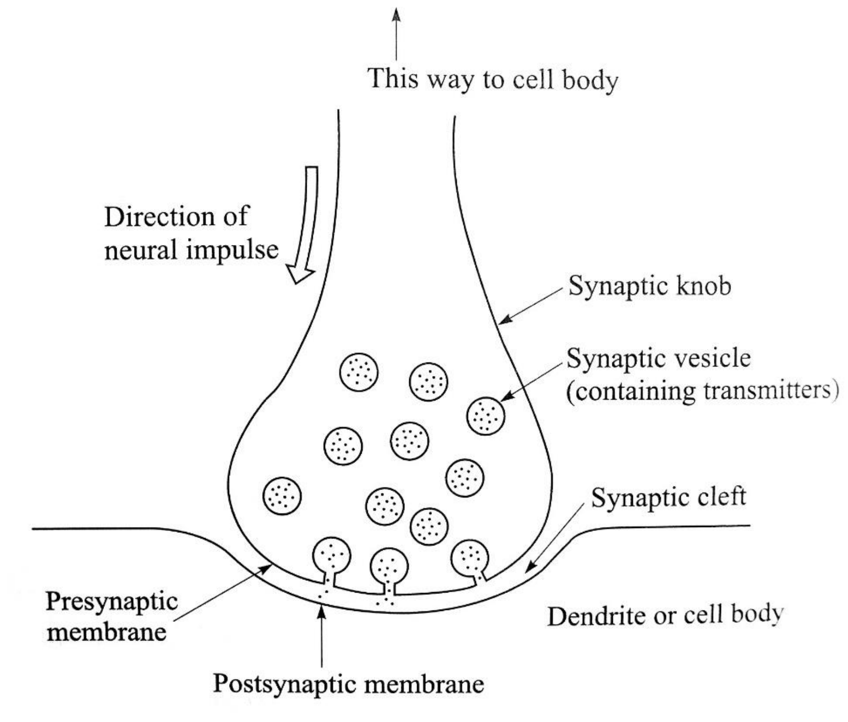
- A biological neuron consists of three main components：
  - Dendrites ：receive signals from other neurons。
  - Cell body (soma)：sums the incoming signals from the dendrites and sums the signals from the numerous synapses on its surface.
  - Axon ：the axons of other neurons connect to the dendrite and cell body surfaces by means of connectors called synapses。The number of synaptic connection from other neurons may range from a few hundred to 10,000.



(a)

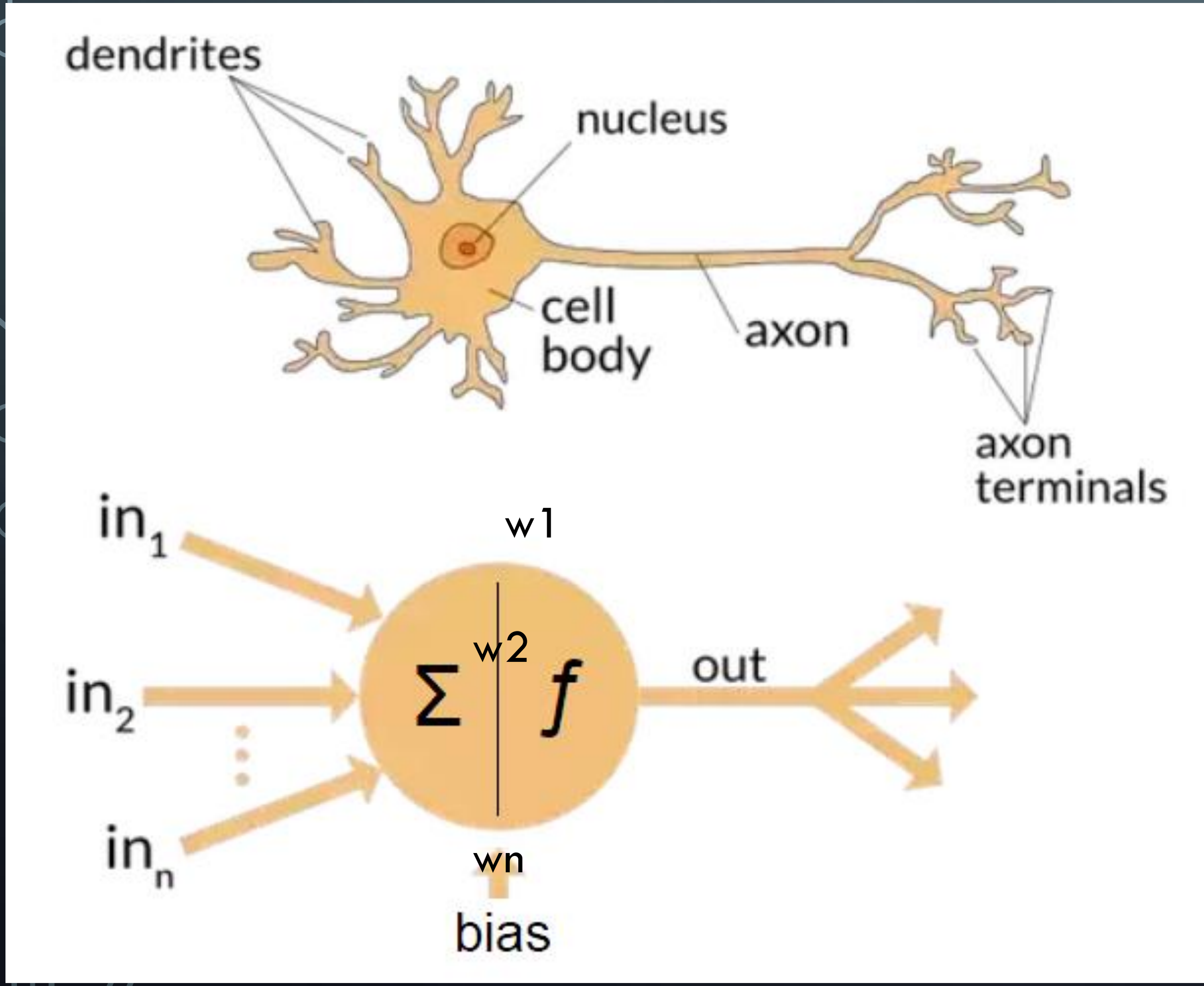
Neurocomputing and Neuroscience

- Synapses are the points of contact that connect the axon terminals to their targets.
- Synapse由下列所組成：
  - Nerve terminal
  - Synaptic cleft or gap
  - Postsynaptic membrane

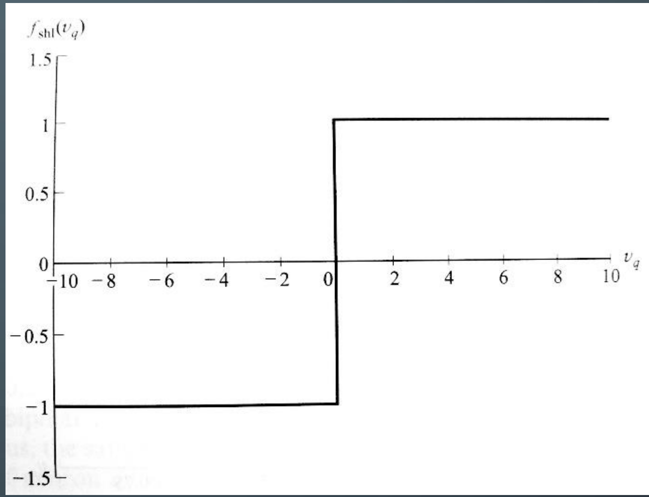


A biological and an artificial neuron

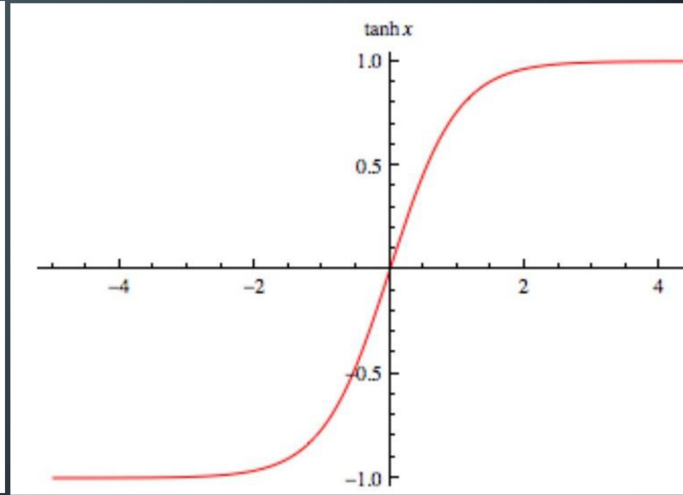
10^{11} neurons in our brain and 10^3 synapses per neuron.

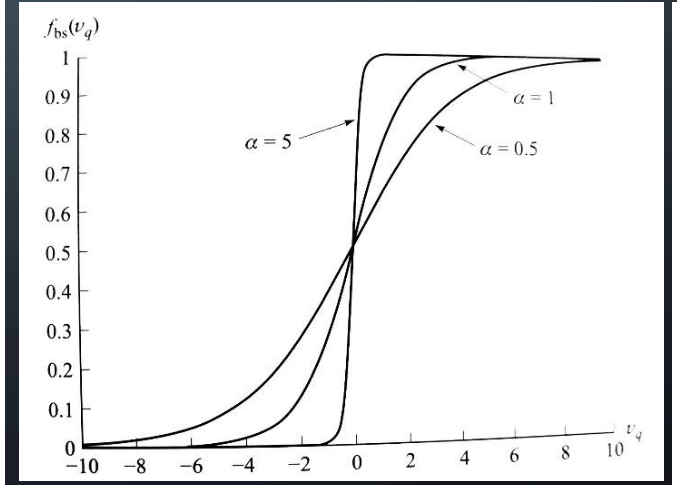


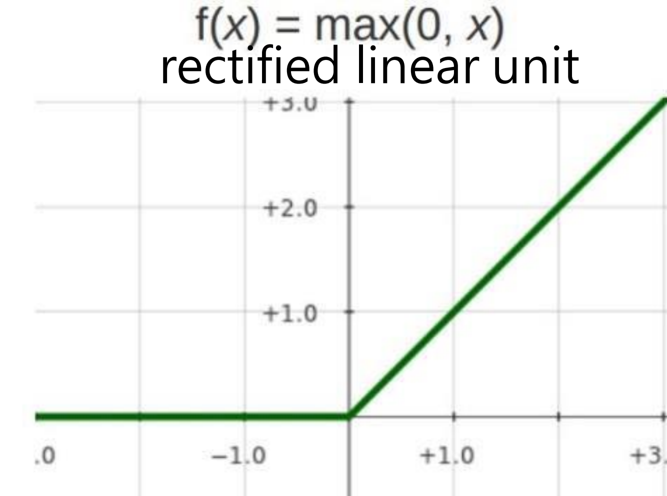
Activation Functions



hyperbolic tangent



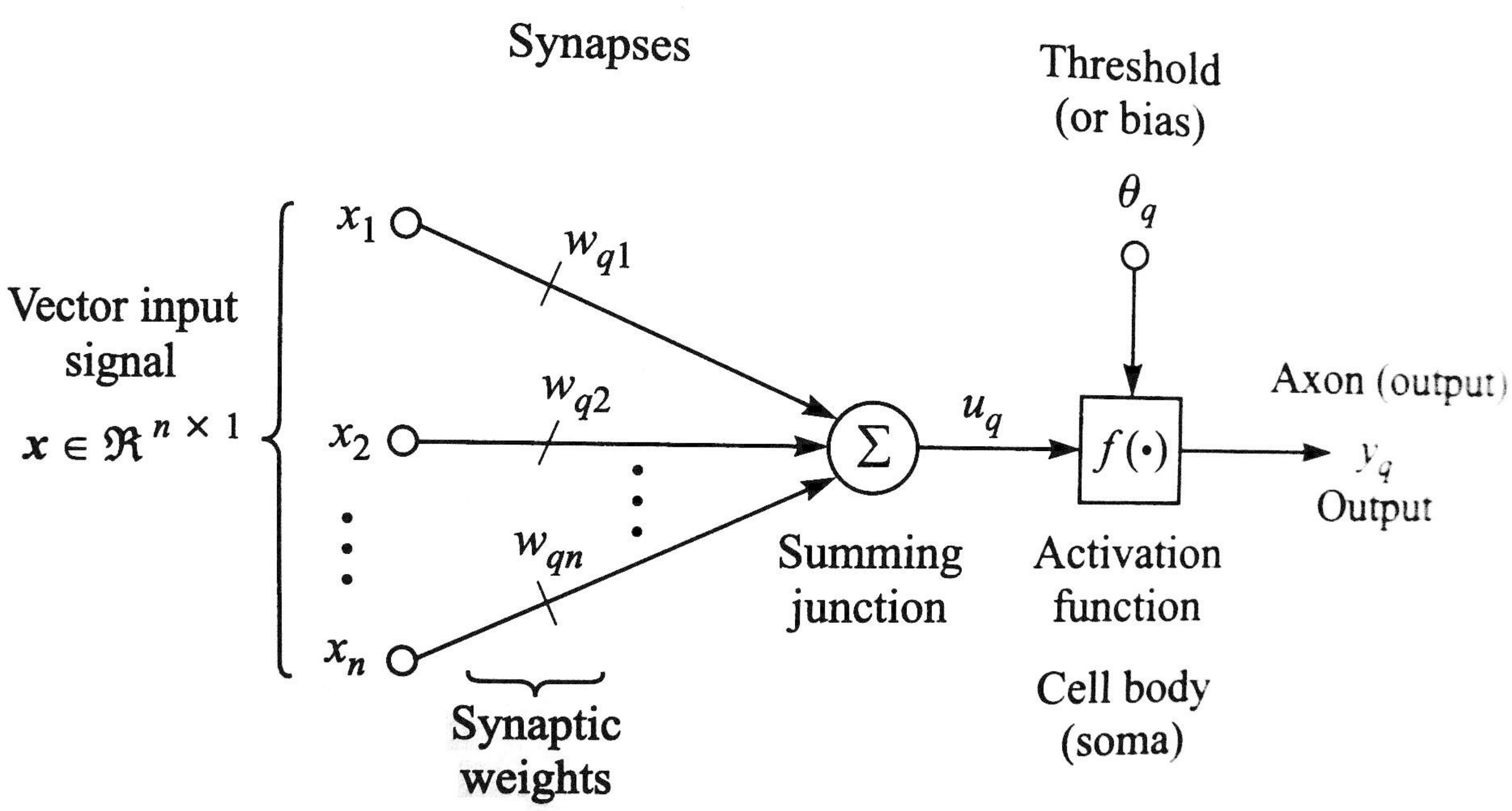




Basic Models of Artificial Neurons

- An artificial neuron can be referred to as a processing element, node, or a threshold logic unit.
- There are four basic components of a neuron
  - A set of *synapses* with associated synaptic weights
  - A *summing device*, each input is multiplied by the associated synaptic weight and then summed.
  - A *activation function*, serves to limit the amplitude of the neuron output.
  - A *threshold function*, externally applied and lowers the cumulative input to the activation function.

Basic Models of Artificial Neurons



the output of the linear combiner is

$$u_q = \sum_{j=1}^{n} w_{qj} x_j = \boldsymbol{w}_{\boldsymbol{q}}^{\boldsymbol{T}} \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{w}_q$$

where $\boldsymbol{w}_{\boldsymbol{q}} = \left[w_{q1}, w_{q2}, \ldots, w_{qn}\right]^T \in \mathrm{R}^{n\times1}$

the output of the activation function i

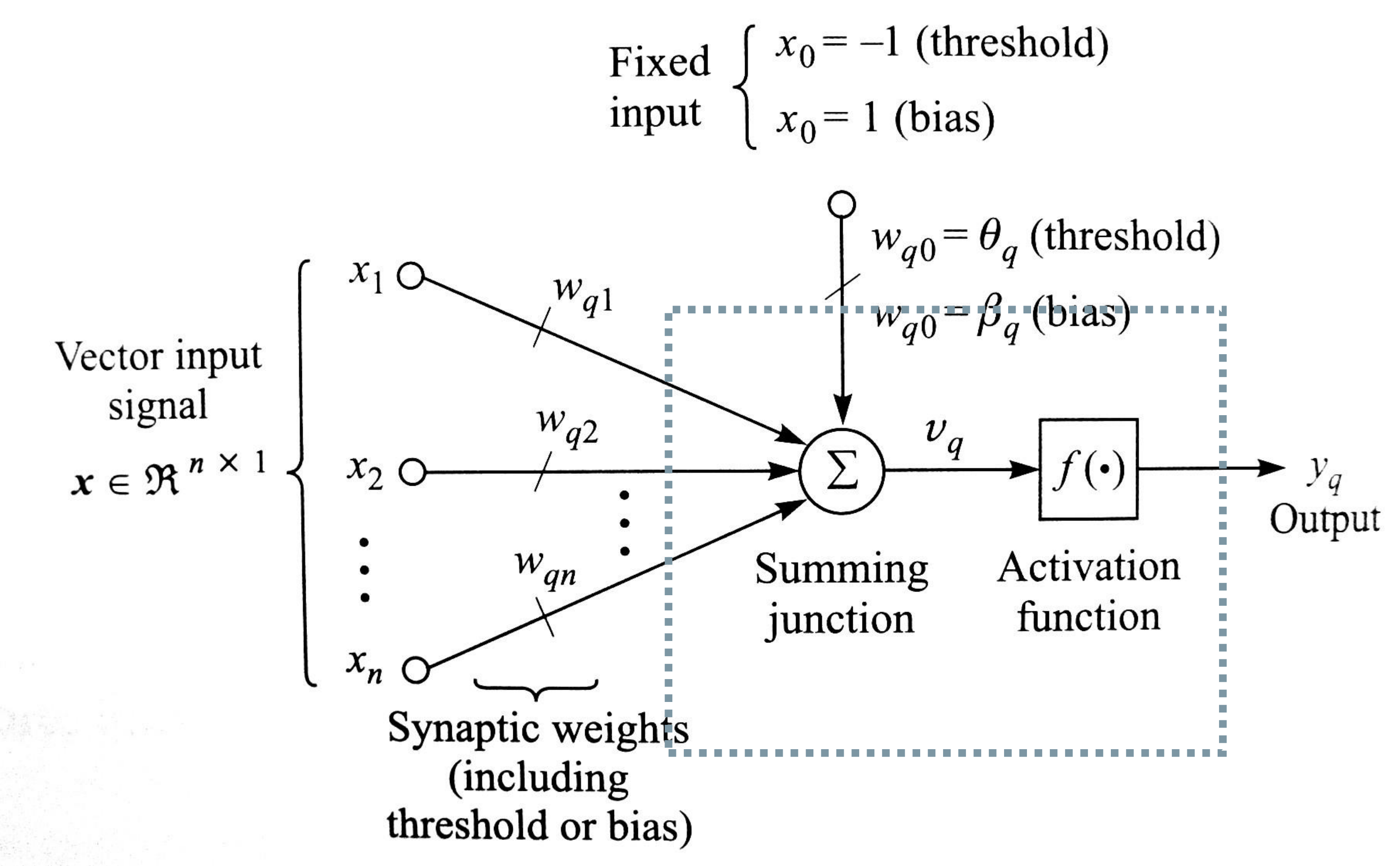
$$y_q = f(v_q) = f\left(u_q - \theta_q\right)$$

the output of the neuron is given by

$$y_q = f\left(\sum_{j=1}^{n} w_{qj} x_j - \theta_q\right)$$

Basic Models of Artificial Neurons

- The threshold (or bias) is incorporated into the synaptic weight vector $\boldsymbol{w}_q$ for neuron q.



Basic Models of Artificial Neurons

The effective internal activation potential is written as

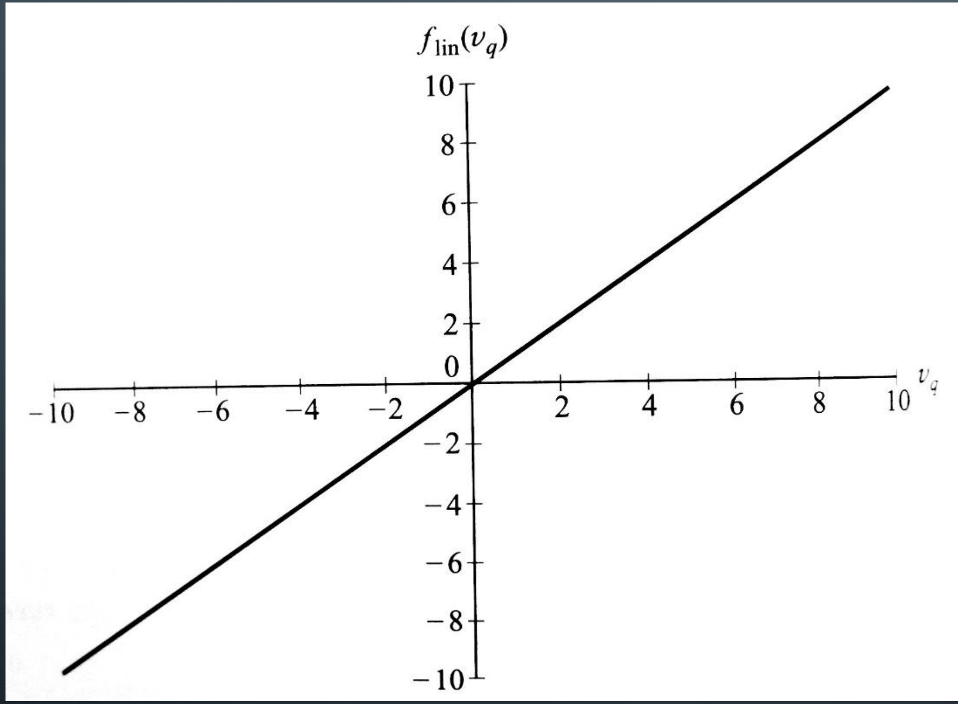
$$v_q = \sum_{j=0}^{n} w_{qj} \mathbf{x}_j$$

The output of neuron q is written as

$$y_q = f\left(v_q\right)$$

Basic Activation Functions

- The activation function, transfer function,
  - Linear or nonlinear



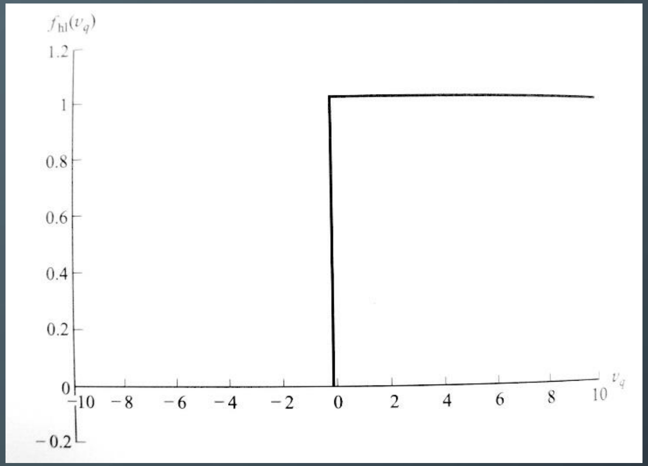
$$y_q = f_{lin}\left(v_q\right) = v_q$$

Linear (identity) activation function

Basic Activation Functions

- Hard limiter
  - Binary function, threshold function
    - (0,1)
    - The output of the binary hard limiter can be written as

$$y_q = f_{hl}(v_q) = \begin{cases} 0 & \text{if } v_q < 0 \\ 1 & \text{if } v_q \geq 0 \end{cases}$$



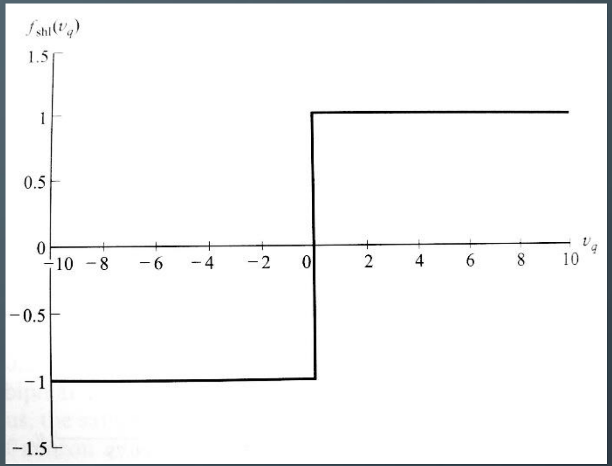
Hard limiter activation function

Basic Activation Functions

- Bipolar, symmetric hard limiter
  - (-1, 1)
  - The output of the symmetric hard limiter can be written as

$$y_q = f_{shl}\left(v_q\right) = \begin{cases} -1 & \text{if } v_q < 0 \\ 0 & \text{if } v_q = 0 \\ 1 & \text{if } v_q > 0 \end{cases}$$

  - Sometimes referred to as the *signum* (or *sign*) function.

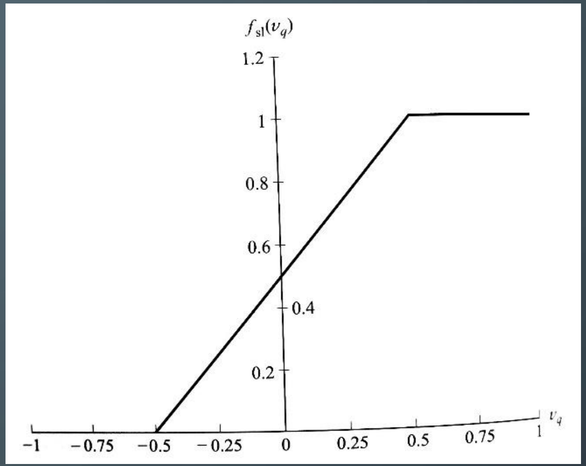


Symmetric limiter activation function

Basic Activation Functions

- Saturation linear function, piecewise linear function
  - The output of the saturation linear function is given by

$$y_q = f_{sl}\left(v_q\right) = \begin{cases} 0 & \text{if } v_q < -\frac{1}{2} \\ v_q + \frac{1}{2} & \text{if } -\frac{1}{2} \le v_q \le \frac{1}{2} \\ 1 & \text{if } v_q > \frac{1}{2} \end{cases}$$

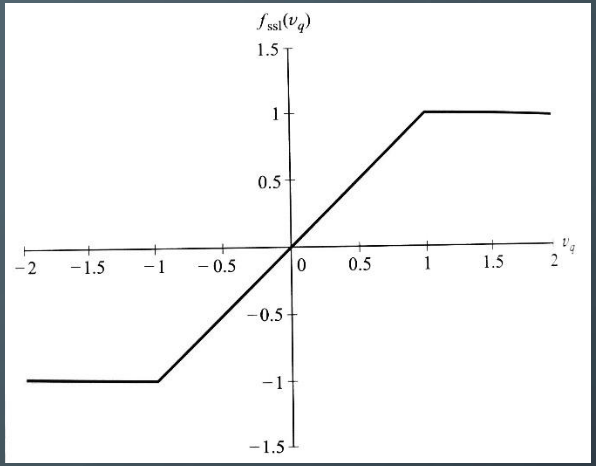


Saturation linear activation function

Basic Activation Functions

- Saturation linear function
  - The output of the symmetric saturation linear function is given by

$$y_q = f_{ssl}\left(v_q\right) = \begin{cases} -1 & \text{if } v_q < -1 \\ v_q & \text{if -1} \le v_q \le 1 \\ 1 & \text{if } v_q > 1 \end{cases}$$

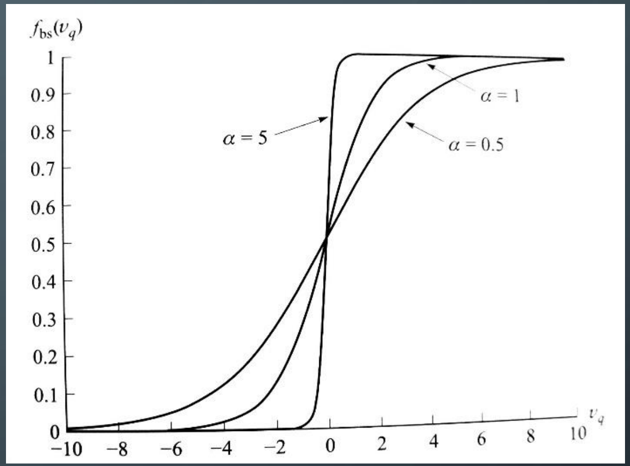


Saturation linear activation function

Basic Activation Functions

- Sigmoid function (S-shaped function)
  - Binary sigmoid function
  - The output of the binary sigmoid function is given by

$$y_q = f_{bs}(v_q) = \frac{1}{1+e^{-\alpha v_q}}$$

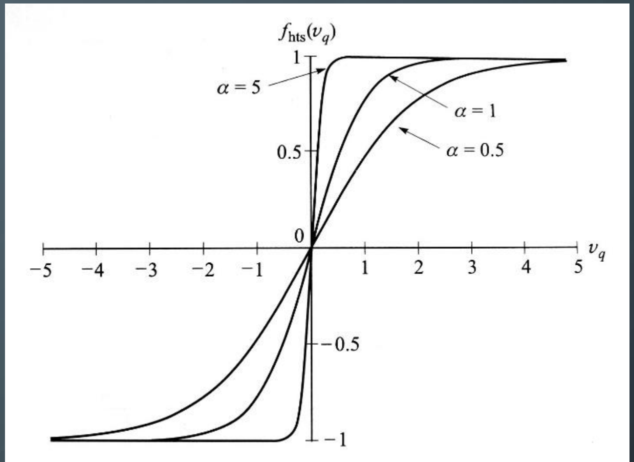


where α is the slope parameter of the binary sigmoid function

Hard limiter has no derivative at the origin, the binary sigmoid is a continuous and differentiable function.

Basic Activation Functions

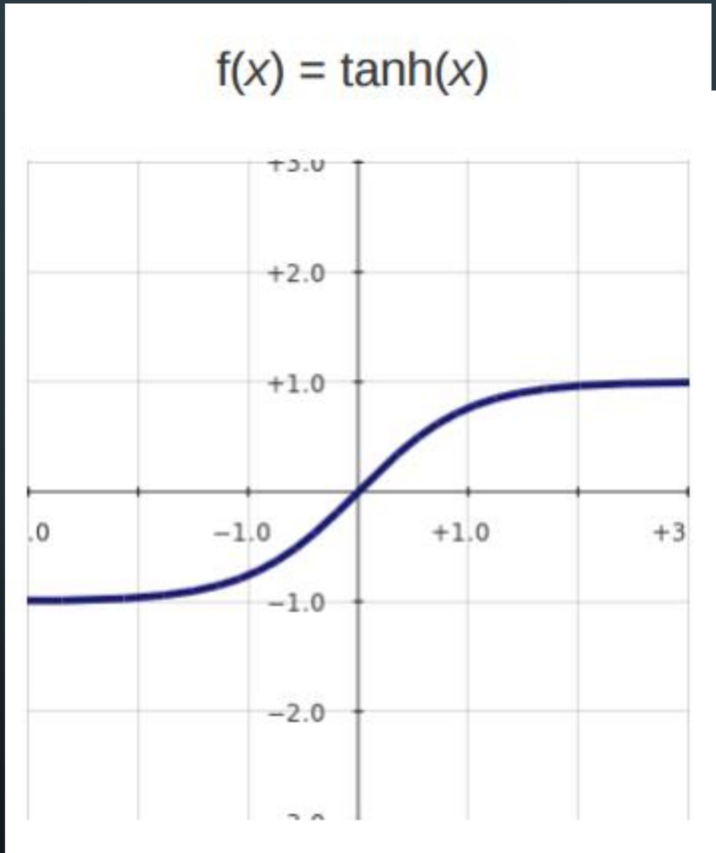
- Sigmoid function (S-shaped function)
  - Bipolar sigmoid function, hyperbolic tangent sigmoid
  - The output of the Binary sigmoid function is given by

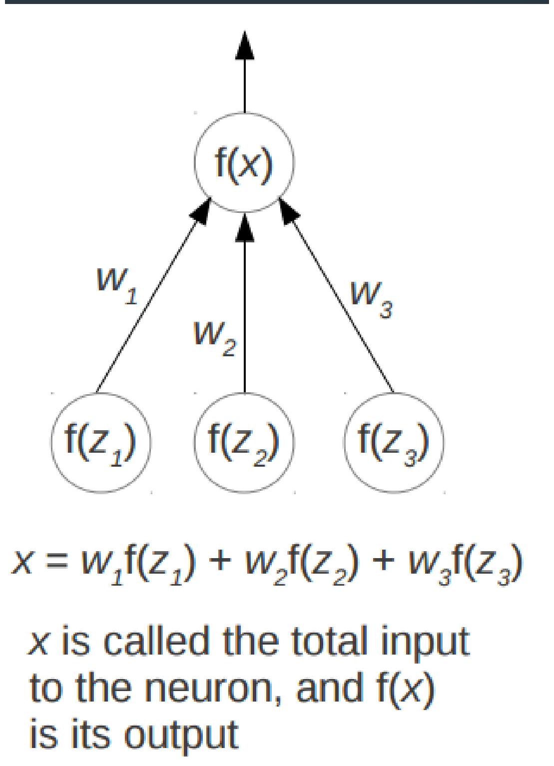


$$y_q = f_{hts}(v_q) = \tanh(\alpha v_q) = \frac{e^{\alpha v_q} - e^{-\alpha v_q}}{e^{\alpha v_q} + e^{-\alpha v_q}} = \frac{1 - e^{-2\alpha v_q}}{1 + e^{-2\alpha v_q}}$$

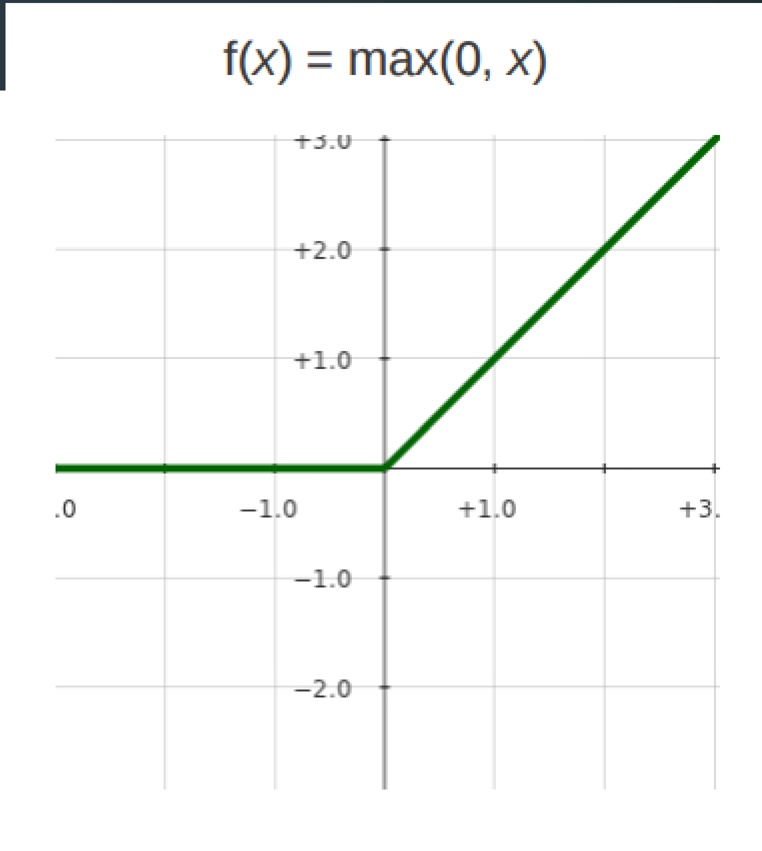
Basic Activation Functions

Sigmoid function





Rectified linear (ReLU)

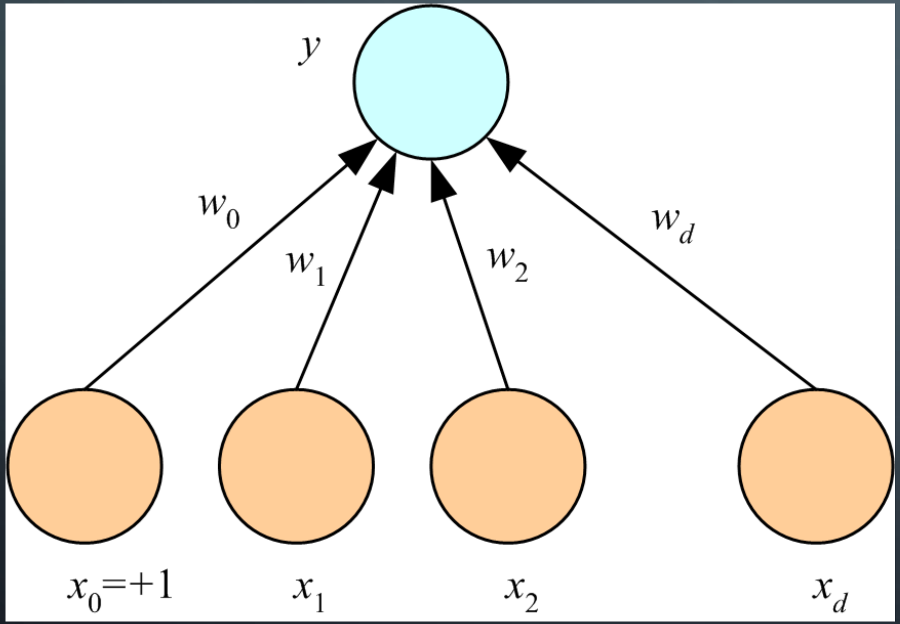


slow to train

quick to train

Simple Perceptron

- Simple perceptron (single-layer perceptron)
  - 由Frank Rosenblatt (1957)提出。

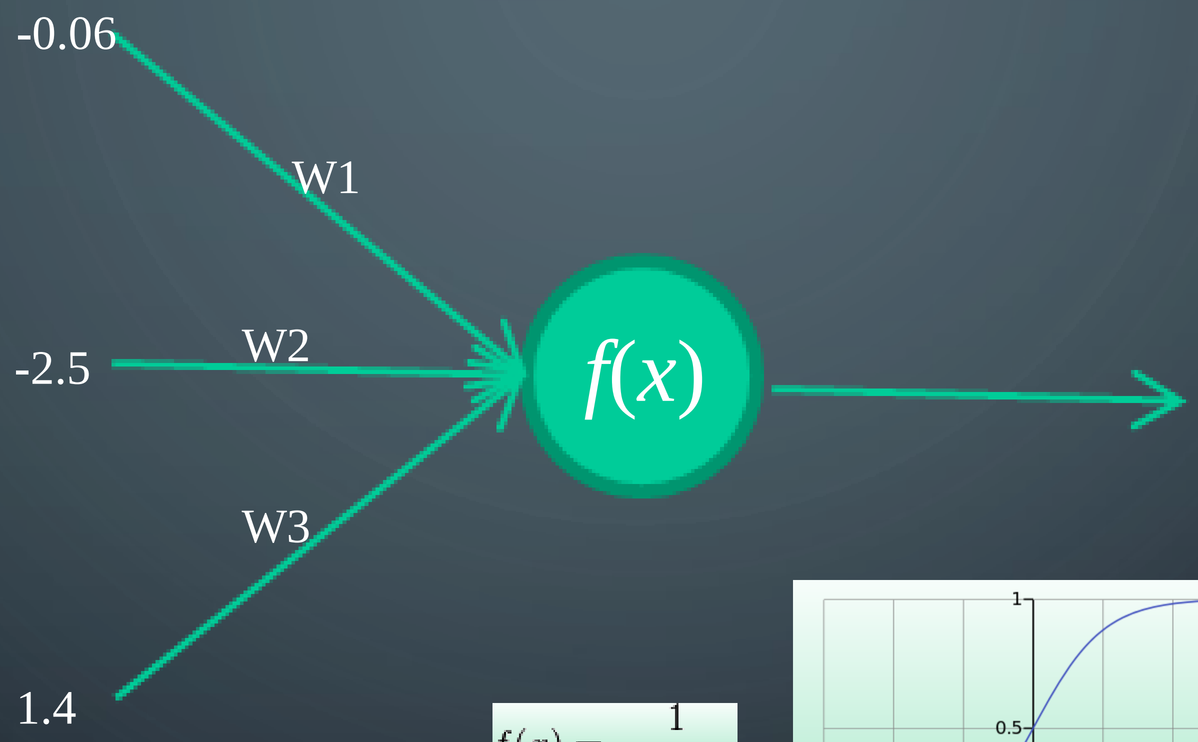


$$y = \sum_{j=1}^{d} w_j x_j + w_0 = \mathbf{w}^T \mathbf{x}$$

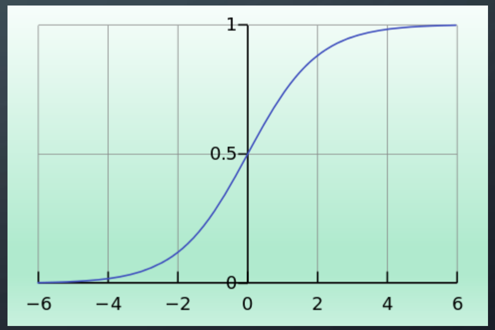
$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$$

$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$

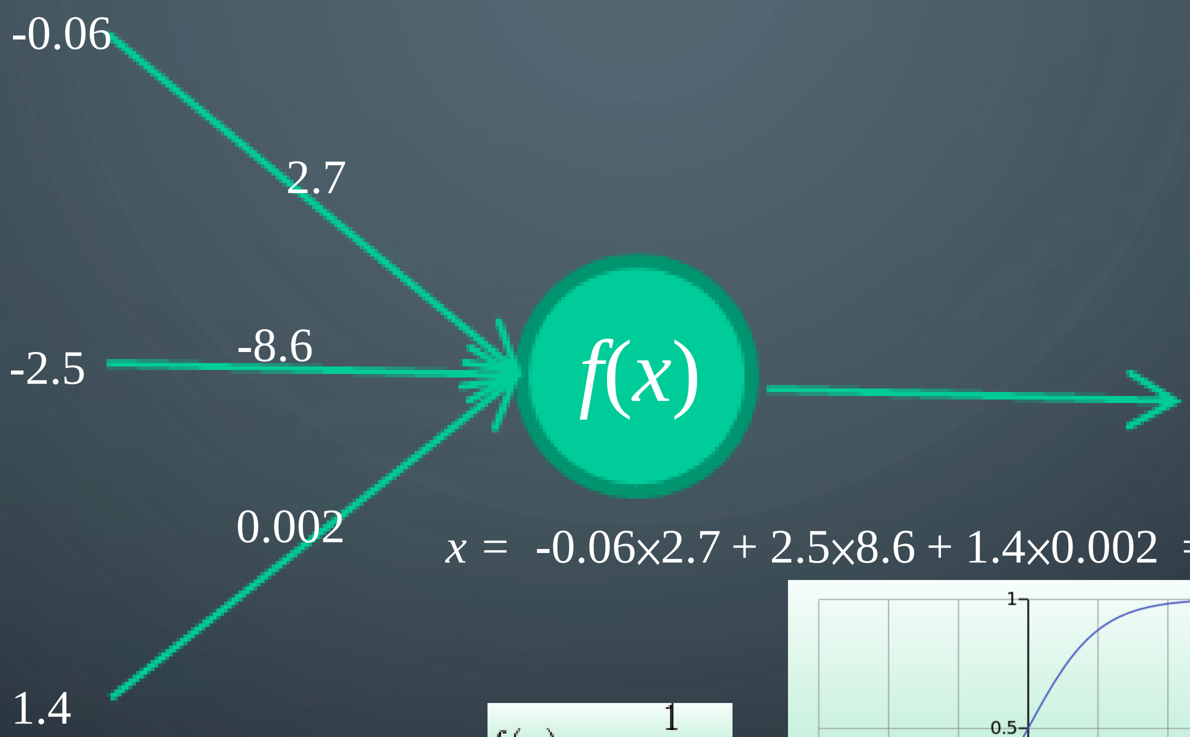
Example: the neuron model



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

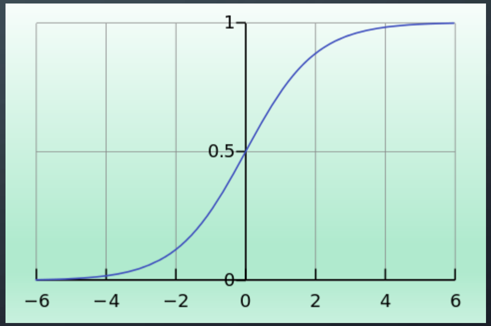


Example: the neuron model



$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$

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



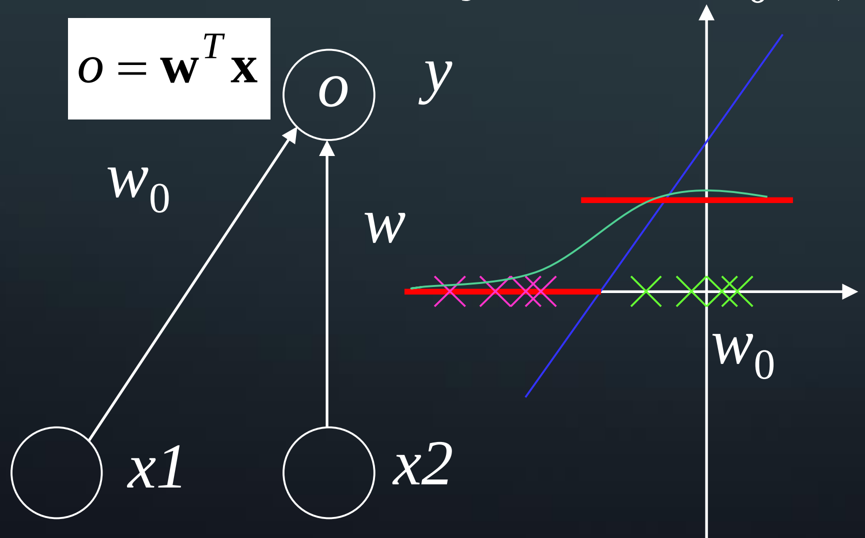
What a perceptron does?

- The simple perceptron can be regarded as a linear classifier, which belongs to supervised learning
- Minsky and Papert discovered a serious limitation: the perceptron cannot solve the XOR problem

- Classification: y=1 ($wx+w_0>0$)

  y=0 ($wx+w_0\leq0$)

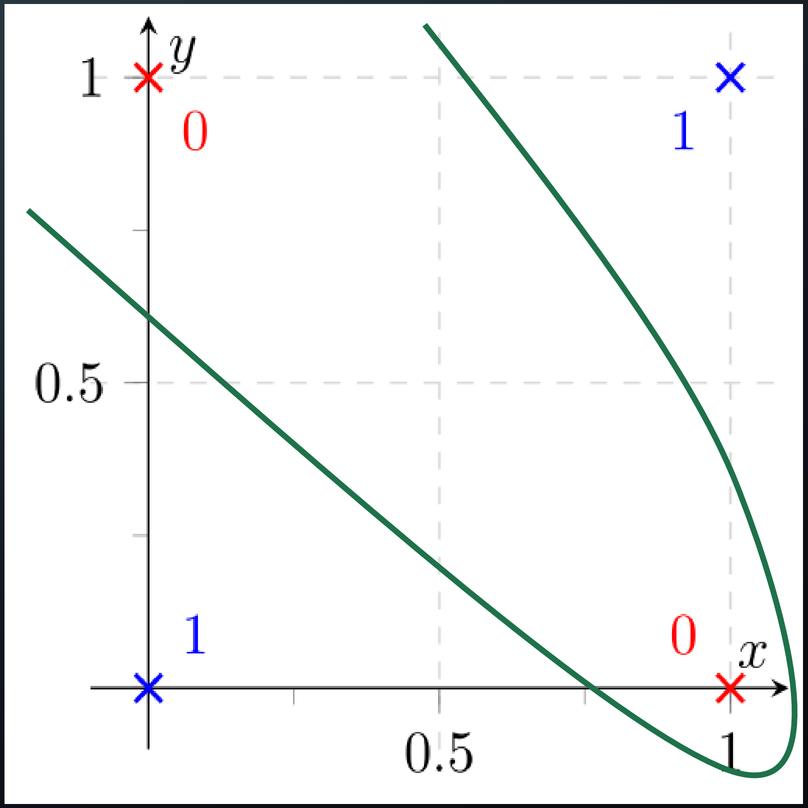
$o = \mathbf{w}^T\mathbf{x}$



$$y = \text{sigmoid}(o) = \frac{1}{1+\exp\left[-\mathbf{w}^T\mathbf{x}\right]}$$

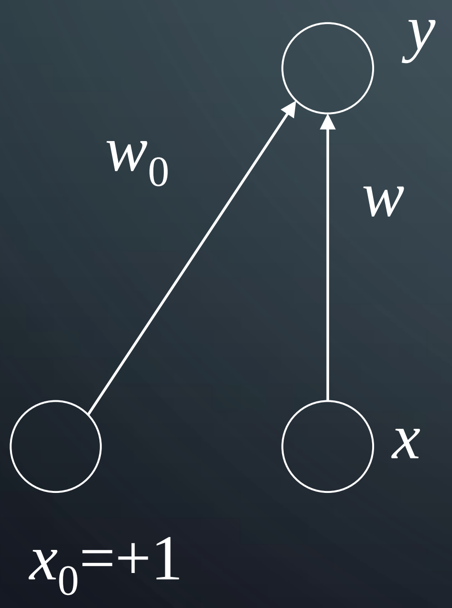
XOR problem

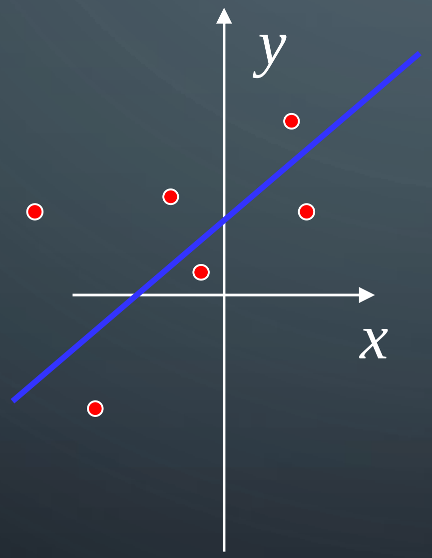
x	y	O
0	0	0
0	1	1
1	0	1
1	1	0



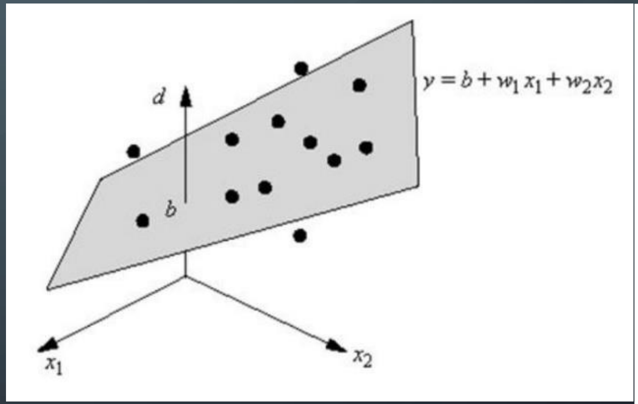
Regression

- A regressor models the relationship between a certain number of features and a continuous target variable
- $y = wx + w_0$

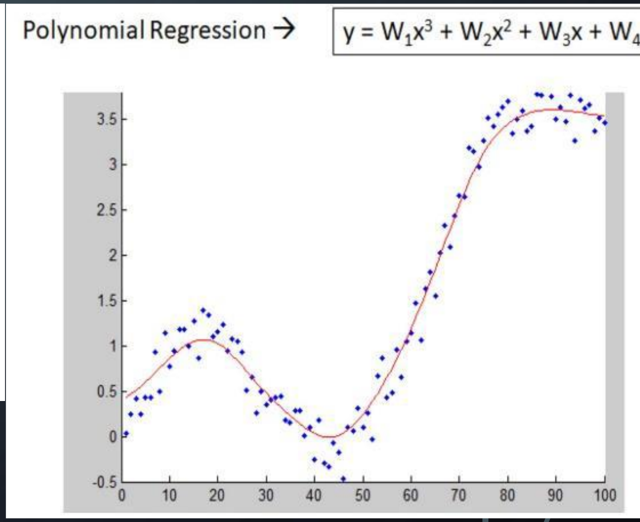




linear



Polynomial



What Is a Neural Network?

- A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It is similar to the brain in two respects:
  1. Knowledge is acquired by the network from its environment through a learning process.
  2. Interneuron connection strengths, known as synaptic weights, are used to store the knowledge.
- Also referred to as
  - Neurocomputing, connectionist networks, parallel distributed processor.

What Is Neurocomputing?

- Neurocomputing approach
  - Involves a learning process within an ANN
  - 一旦類神經網路訓練完成，該類神經網路即可進行特定的工作。例如pattern recognition
  - Associated (聯想)
- Why can we (human) perform certain tasks much better than a digital computer?
  - Our brain is organized
    - 大腦神經(nerve)的傳導速度比電子的速度慢(10^6)倍，但腦神經具有大量平行的計算能力。(約10^{11}個neuron)
    - 大腦是一個adaptive, nonlinear, parallel的計算機。

What Is Neurocomputing? (Cont.)

- 類神經網路的主要能力：
  - 由範例學習(learn by example)
  - 歸納(generalize)
    - The NN can classify input patterns to an acceptable level of accuracy even if they were never used during the training process.
- 大部分的類神經網路具有類似的特徵：
  - Parallel computational architecture
  - Highly interconnected
  - Nonlinearity output

What Is Neurocomputing? (Cont.)

- Applications of neural networks:
  - Image processing
  - Prediction and forecasting
  - Associative memory
  - Clustering
  - Speech recognition
  - Combinatorial optimization
  - Feature extraction
  - …

Benefits of Neural Networks

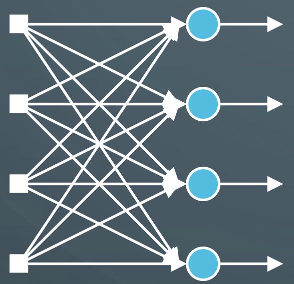
- The use of neural networks offers the following useful properties and capabilities:
  - Nonlinearity
    - A neuron can be linear or nonlinear
  - Input-Output Mapping
    - Learning with a teacher or supervised learning
  - Adaptively
    - Neural network adapts their synaptic weights to changes in the surrounding environment.
  - Evidential Response
    - Provide confidence information in decision made

Benefits of Neural Networks (Cont.)

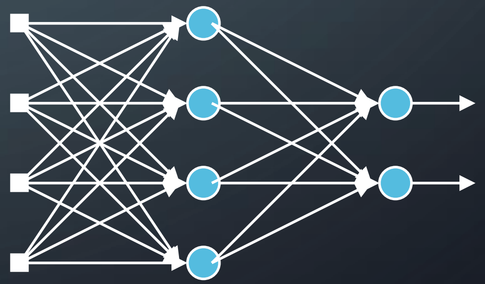
- Contextual Information
  - Every neuron in the network is potentially affected by the global activity of all other neurons in the network.
- Fault Tolerance
  - A neural network exhibits a graceful degradation in performance rather than catastrophic failure.
- VLSI Implement ability
  - The massively parallel nature of a neural network makes it well suited for implementation using VLSI.
- Uniformity of Analysis and Design
  - The same notation is used in all domains involving the application of neural networks.
- Neurobiological Analogy
  - The design of a neural network is motivated by analogy with the brain.

Network Architecture

- Single-layer feed-forward network (usually use threshold neuron function)
  - The input layer of source nodes project onto an output layer of neurons
  - Feedforward

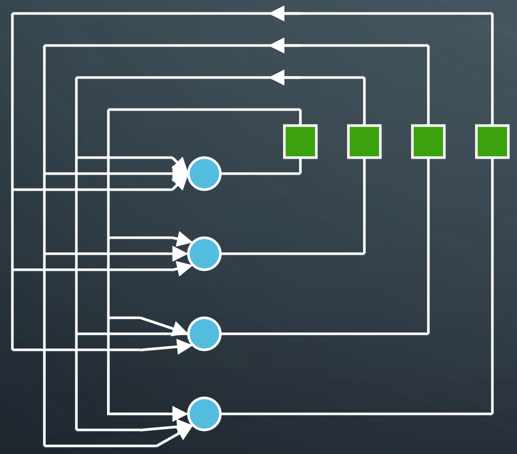


- Multi-layer feedforward network (fully connected, usually use sigmoid neuron function)
  - One or more hidden layers.
  - Enabled to extract higher-order statistics.

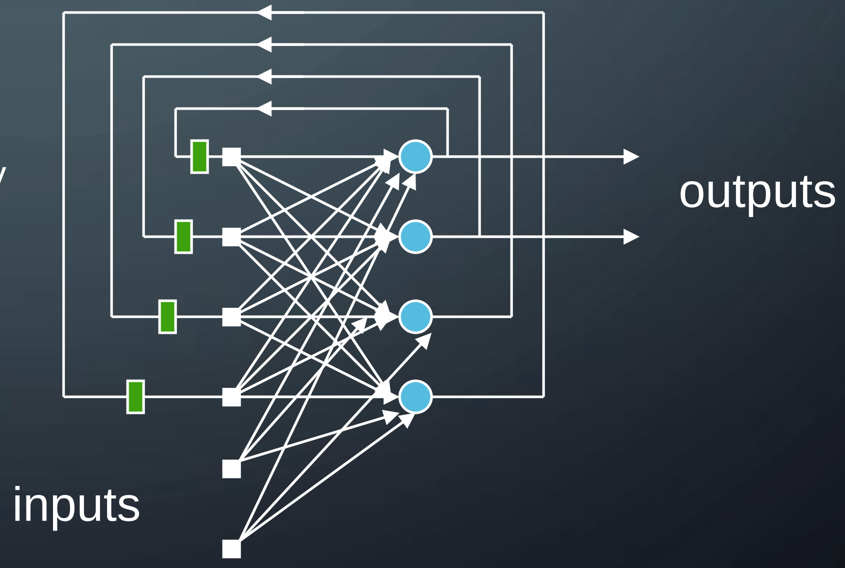


Network Architecture

- Recurrent network
  - It has at least one feedback
  - Recurrent network with no self-feedback loops and no hidden neurons
  - Recurrent network with hidden units



Unit delay operator

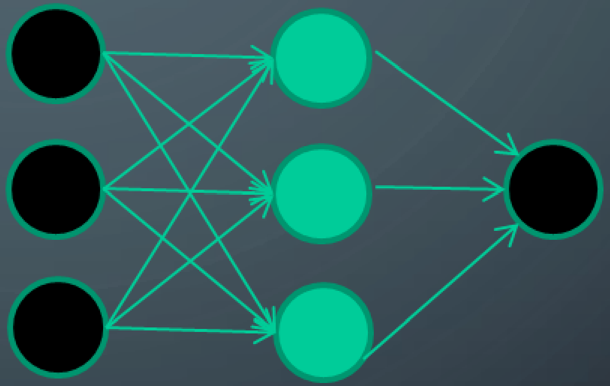


Neural Networks & Training Dataset

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

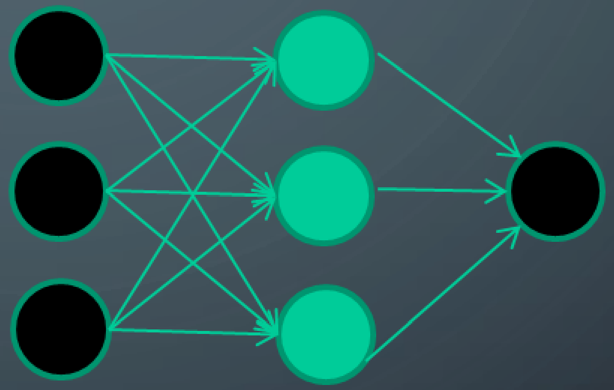


Training The Neural Networks

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



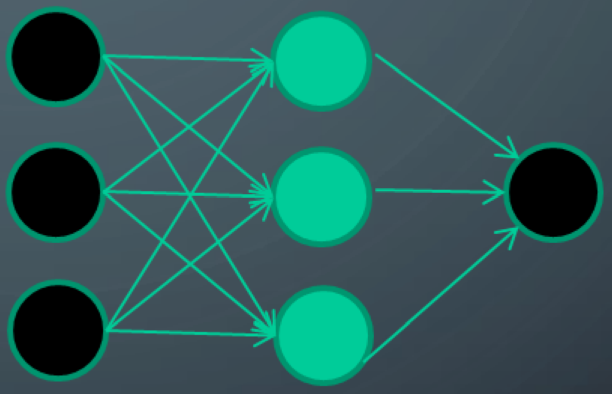
Training the Neural Networks

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Initialize with random weights



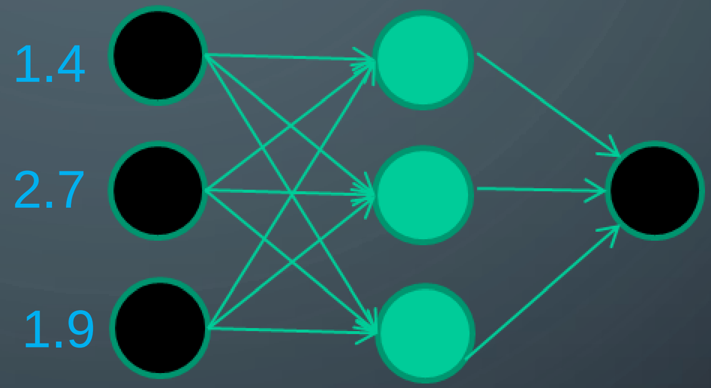
Training the Neural Networks

Present a training pattern

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



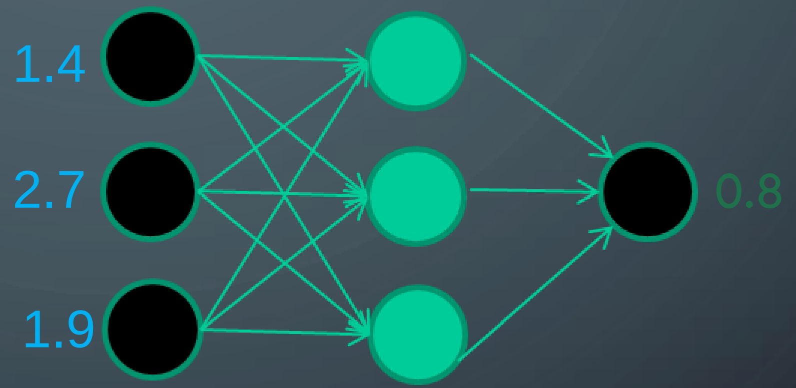
Training the Neural Networks

Feed it through to get output

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



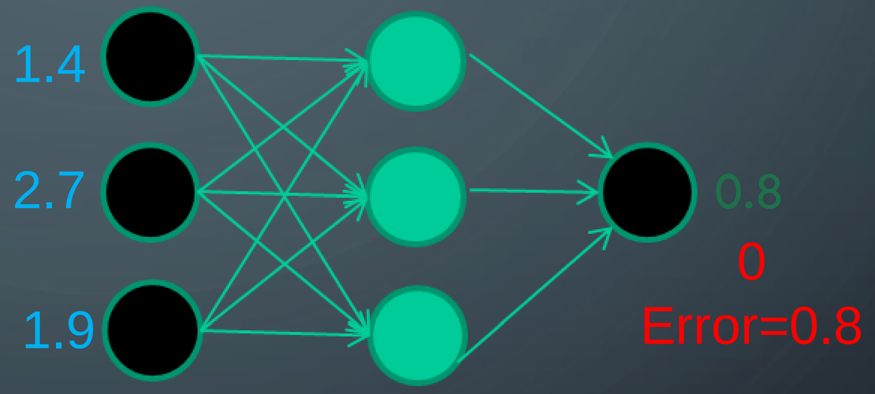
Training the Neural Networks

Compare with target output

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



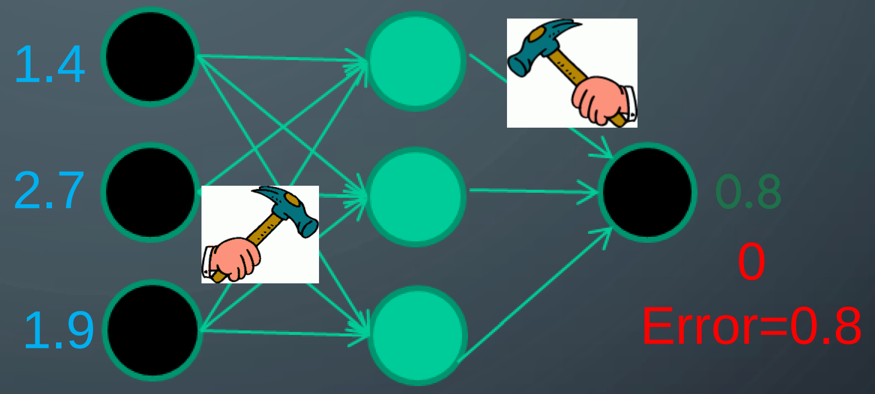
Training the Neural Networks

Adjust weights based on error

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



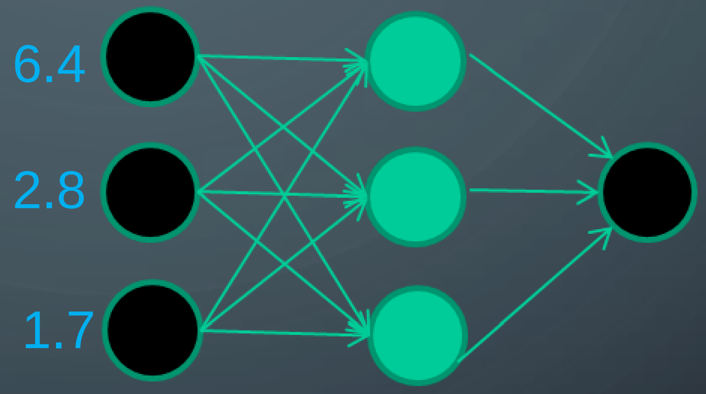
Training the Neural Networks

Present a training pattern

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



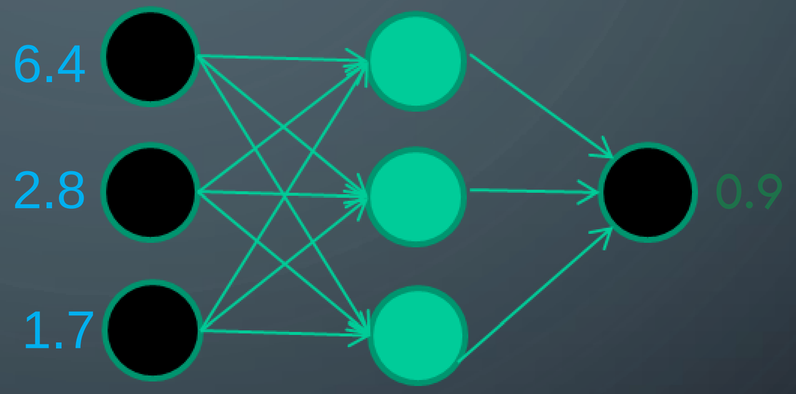
Training the Neural Networks

Feed it through to get output

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



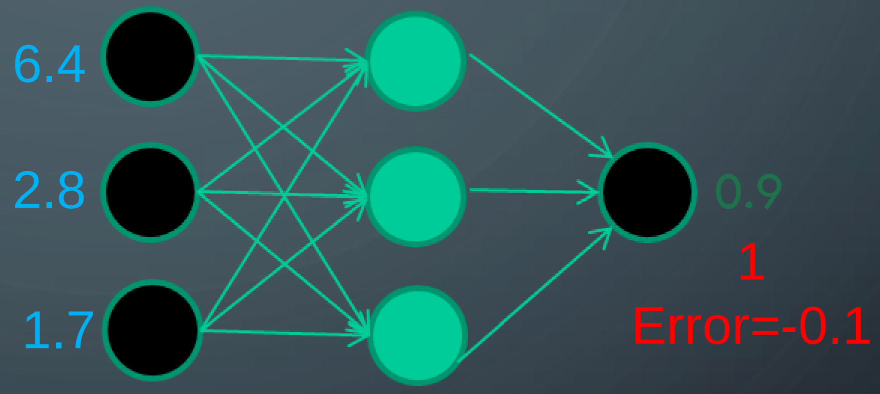
Training the Neural Networks

Compare with target output

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



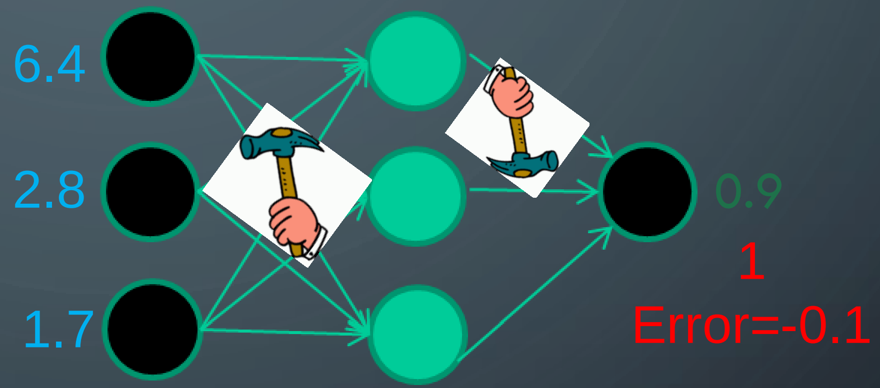
Training the Neural Networks

Adjust weights based on error

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



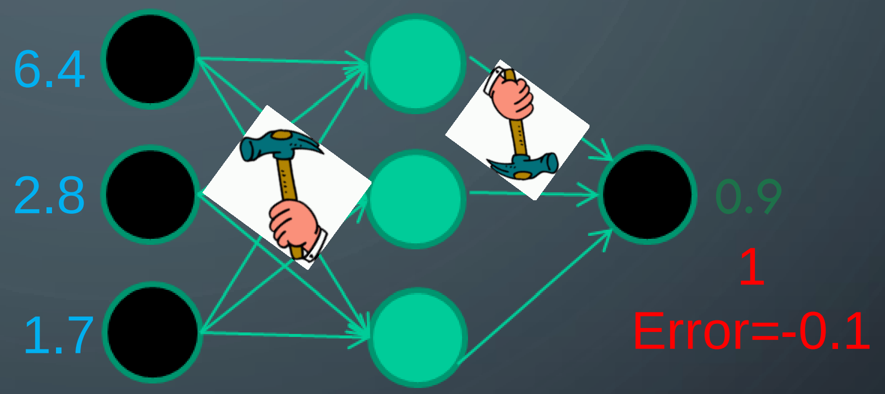
Training the Neural Networks

And so on ....

Training Dataset

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

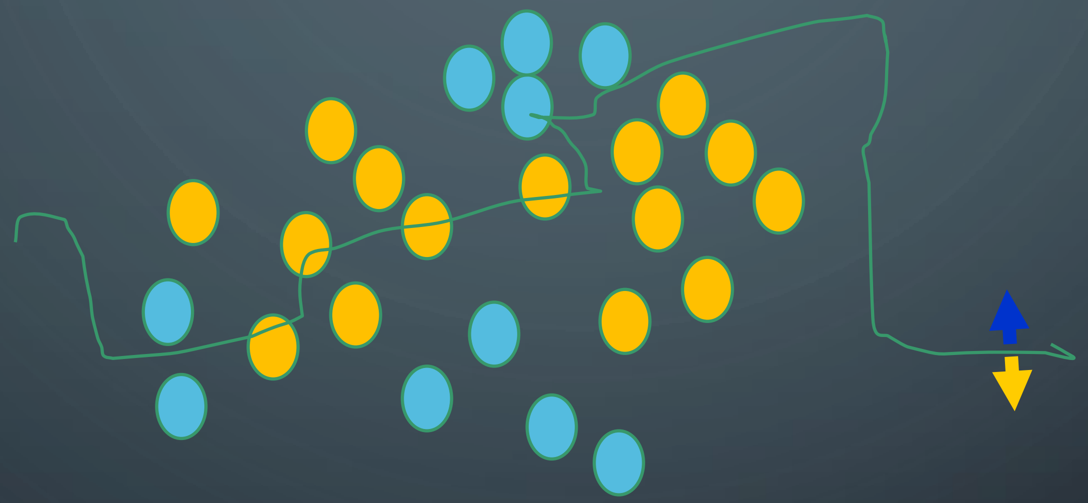
etc …



Repeat this thousands, maybe millions of times
Each time taking a random training instance, and
making slight weight adjustments *reduce the error*

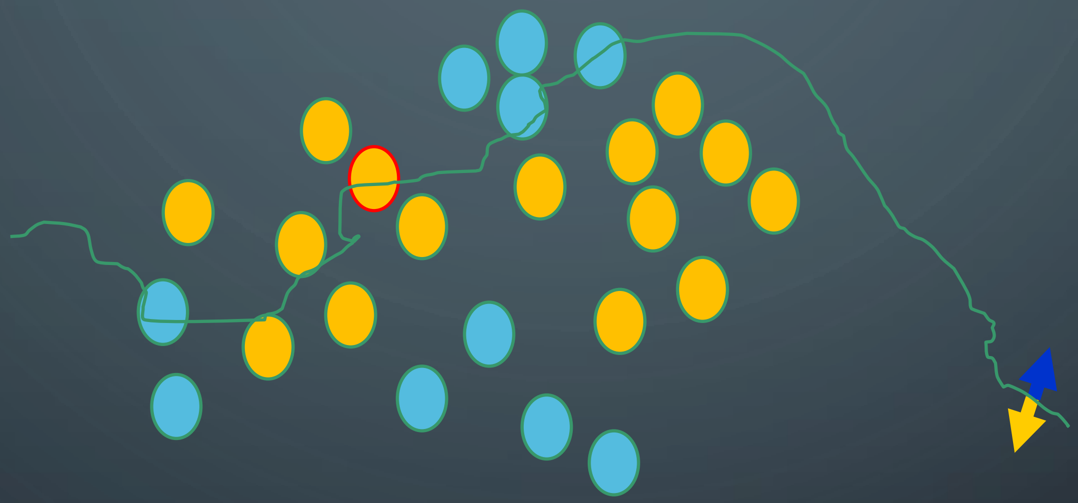
The Decision Boundary Perspective…

Initial random weights



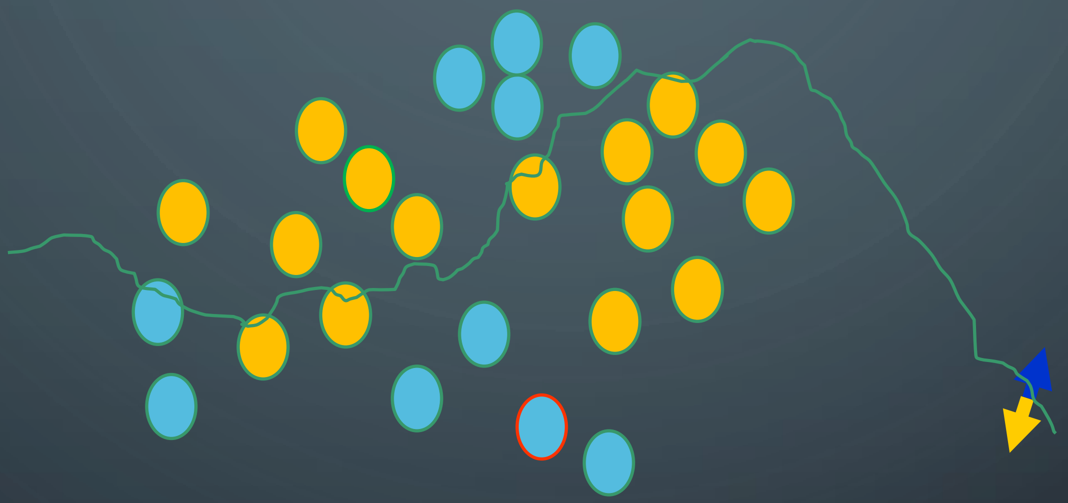
The Decision Boundary Perspective…

Present a training instance / adjust the weights



The Decision Boundary Perspective…

Present a training instance / adjust the weights

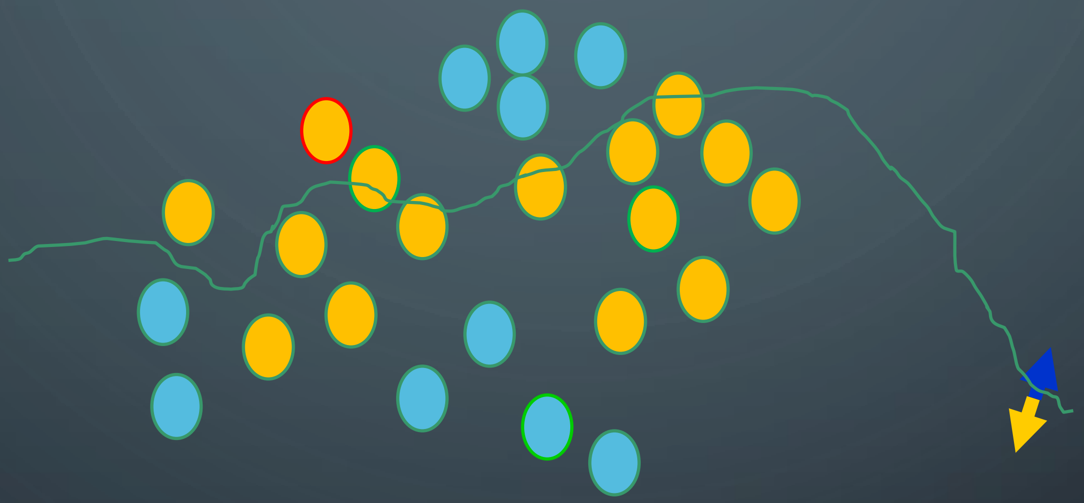


The Decision Boundary Perspective…

Present a training instance / adjust the weights

The Decision Boundary Perspective…

Present a training instance / adjust the weights



The Decision Boundary Perspective…

Eventually ….

Classification of Neural Networks

- Supervised learning vs. unsupervised learning