CS-713 (Artificial Intelligence) : Chapter 6

Parallel and Distributed AI, Connectionist Models, Expert Systems

Preetpal Kaur Buttar

Department of Computer Science and Engineering

Sant Longowal Institute of Engineering and Technology, Longowal

6.1 Parallel and Distributed AI

Parallelizing AI Programming Languages

- Writing parallel programs is a difficult task for humans, and parallel implementation of AI programming languages will make effective speedups more practical.
- Parallel LISP models include Multilisp, QLISP, the Paralation Model.
- Parallel PROLOG models include Concurrent PROLOG, PARLOG, Guarded Horn Clauses.

- Languages like PROLOG suggest two types of parallelism:
  - OR parallelism: Multiple paths to the same goal are taken in parallel.
    - For example,

      uncle(X, Y):- mother(Z, Y), sibling (X, Z)

      uncle(X, Y):- father(Z, Y), sibling(X, Z)
    - Then the query:

      ? – uncle(John, Bill)
      - could be satisfied in two different ways since John could be the sibling of Bill's mother or of Bill's father.
      - A sequential implementation would try to satisfy the first condition and then, if that failed, try the second condition.
  - In AND parallelism, the portions of a conjunctive goal are pursued in parallel.
    - For example,

      Infieldfly(X):- fly(X), infieldcatchable(X), occupiedbase(first), outs(zero)
    - Here, 4 conditions can be checked in parallel, possibly leading to a four-fold speedup in processing.
    - Such AND parallelism is not so straight-forward when variable are shared across goals as in:

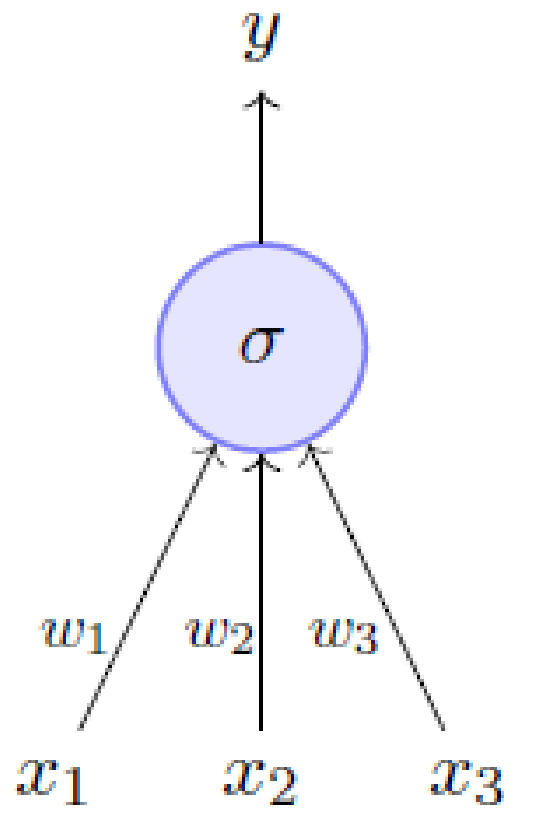
      uncle(X, Y):- mother(Z, Y), sibling (X, Z)

Parallelizing AI Algorithms

- Some problems are more amenable to parallel solutions than others.
- Some algorithms whose parallel aspects have been studied include best-first search, alpha-beta pruning, constraint satisfaction, natural language parsing, resolution theorem proving, property inheritance.

6.2 Connectionist Models

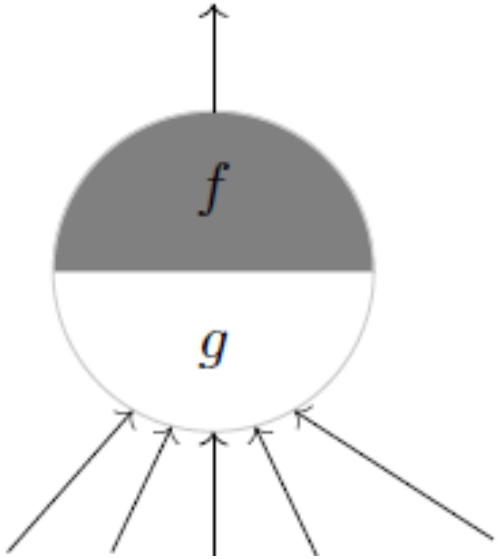
- The neural network architectures were dubbed “connectionist” architectures in the past years.
- The most fundamental unit of a neural network is called an artificial neuron.



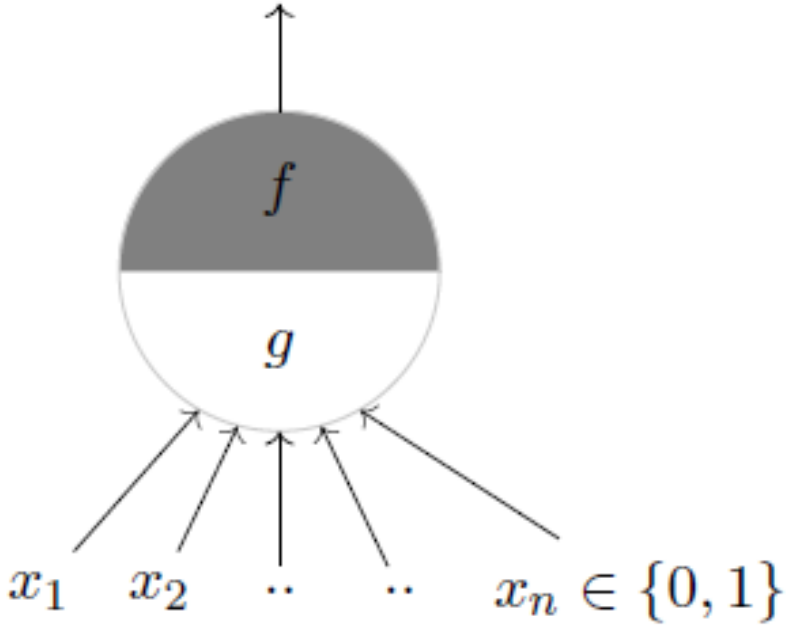
Artificial Neuron

6.2.1 McCulloch Pitts Neuron

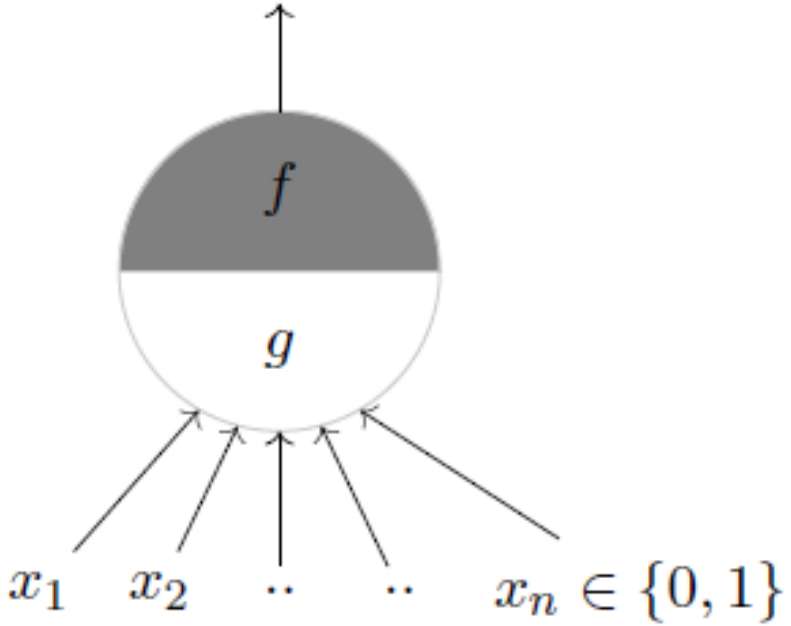
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).



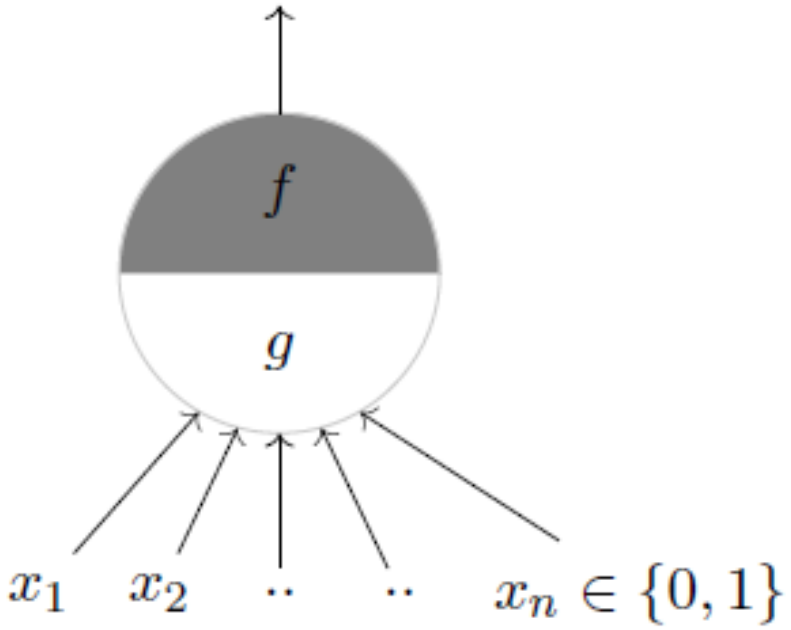
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).



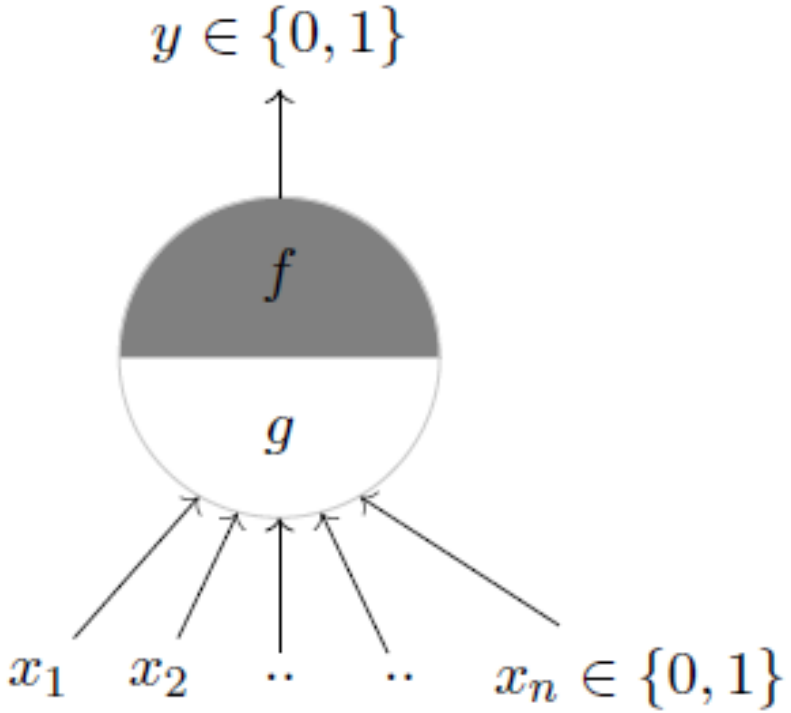
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs



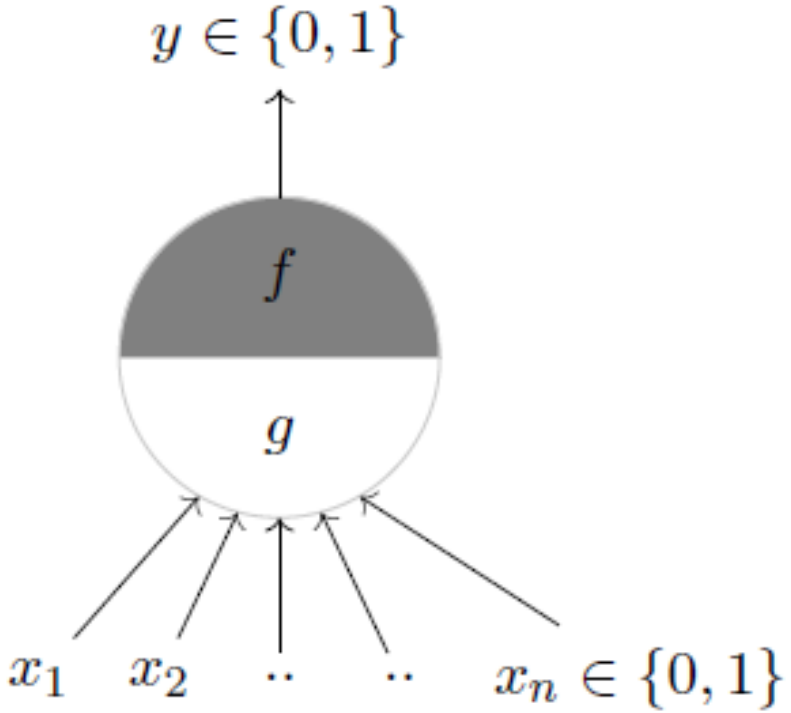
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.



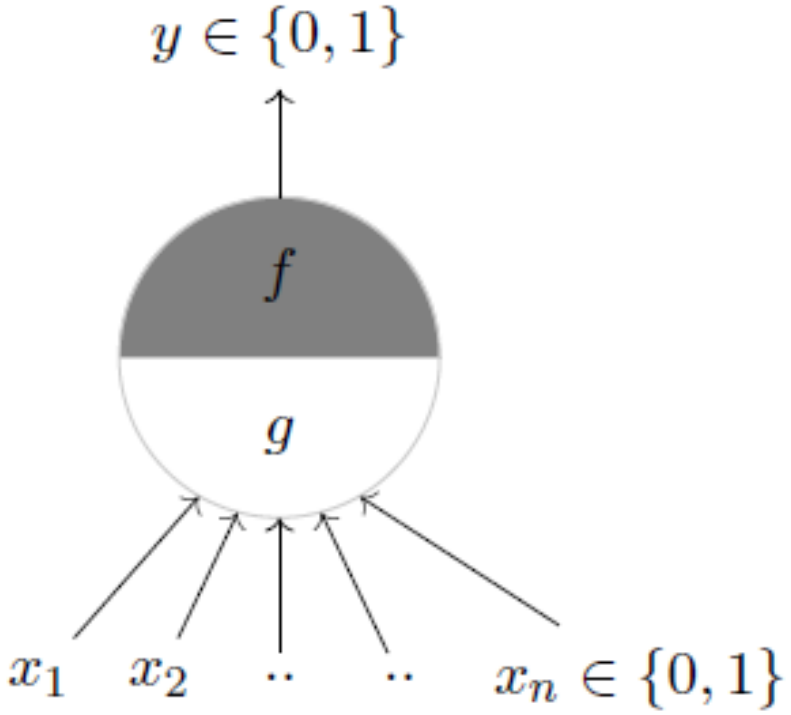
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.



- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory

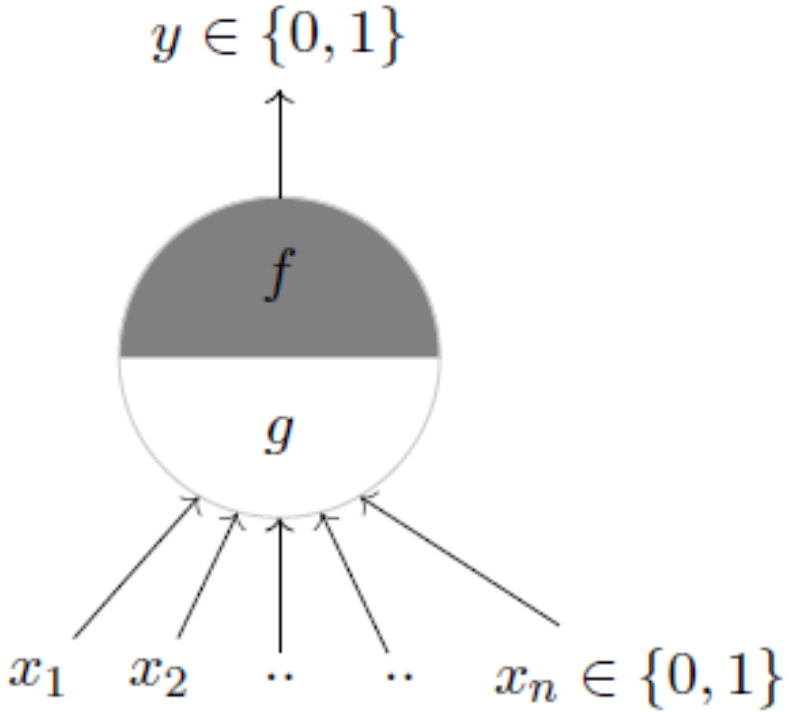


- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory
- y = 0 if any x_i is inhibitory, else



- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory
- y = 0 if any x_i is inhibitory, else

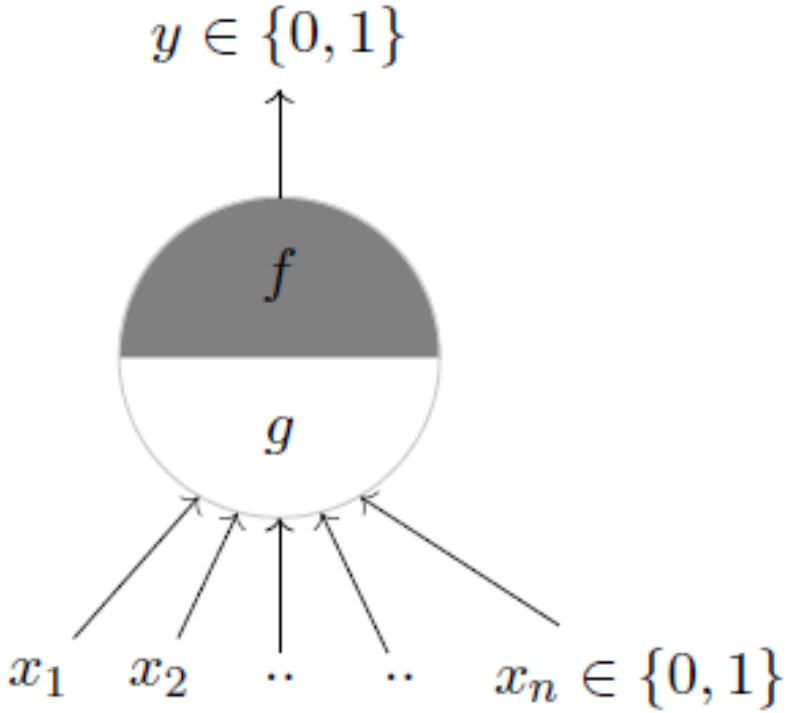
  $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$



- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory
- y = 0 if any x_i is inhibitory, else

  $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$

  $y = f(g(x)) = 1$ if $g(x) \geq \theta$

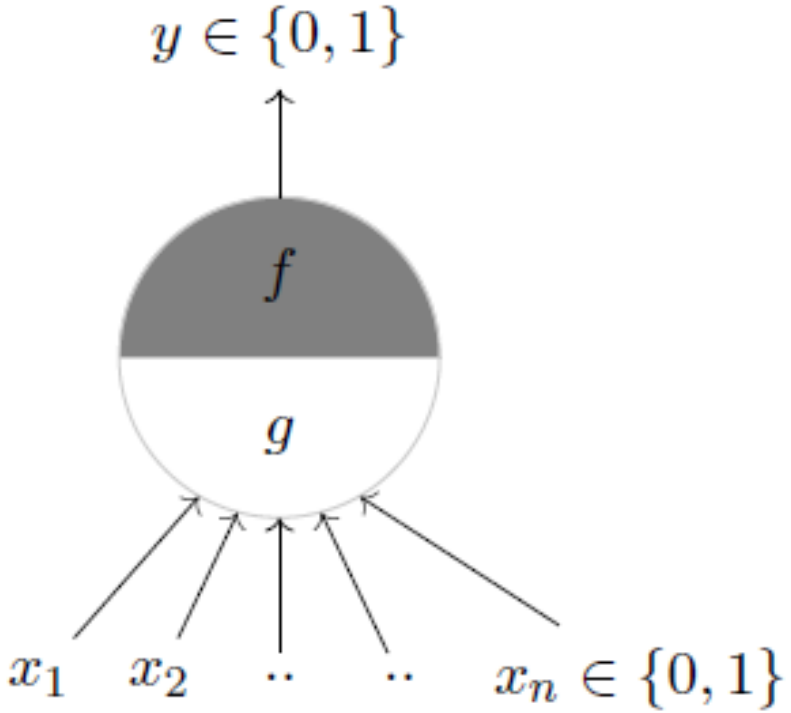


- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory
- y = 0 if any x_i is inhibitory, else

  $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$

  $y = f(g(x)) = 1$ if $g(x) \geq \theta$

  $= 0$ if $g(x) < \theta$

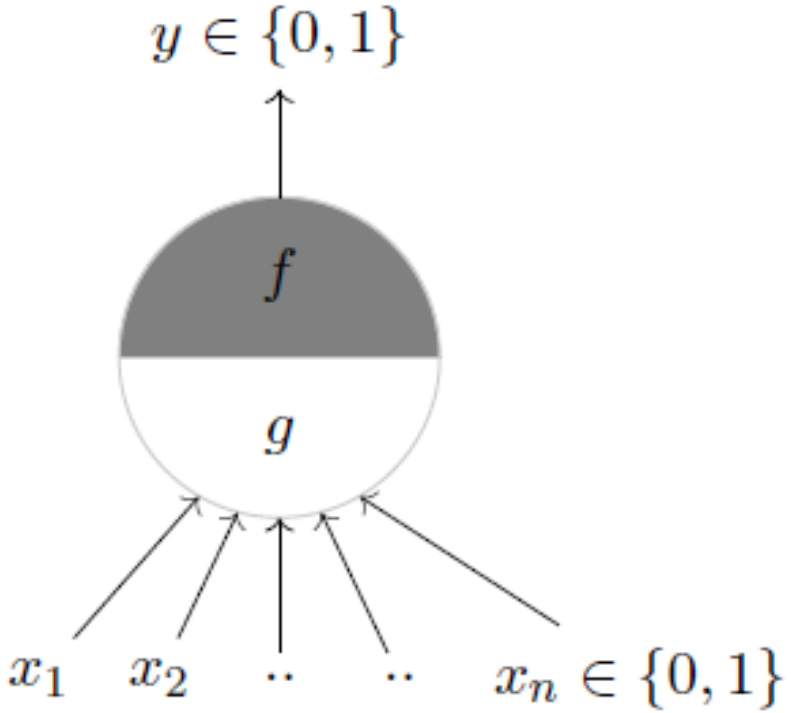


- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory
- y = 0 if any x_i is inhibitory, else

  $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$

  $y = f(g(x)) = 1$ if $g(x) \geq \theta$

  $= 0$ if $g(x) < \theta$
- θ is called the thresholding parameter

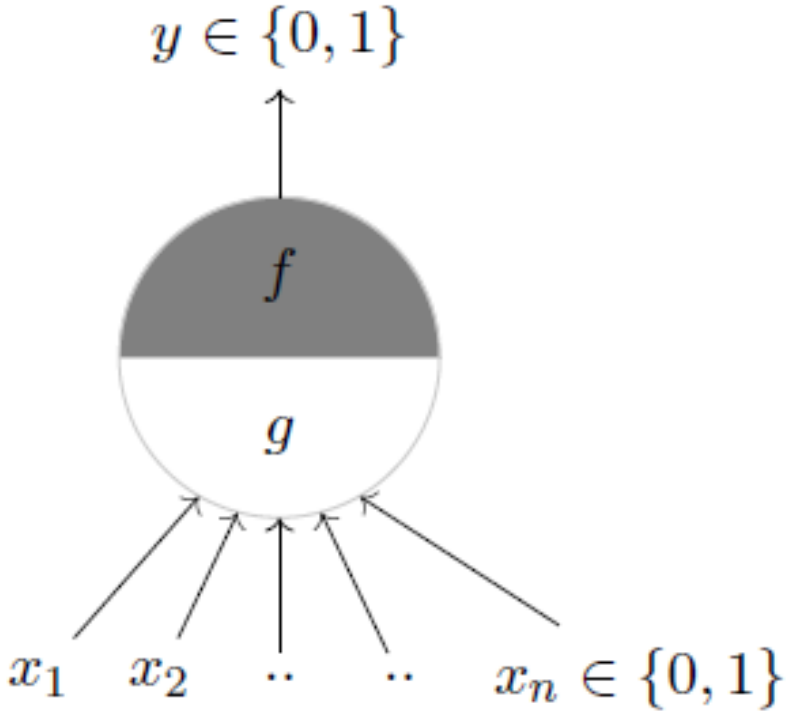


- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943).
- g aggregates the inputs and the function f takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory
- y = 0 if any x_i is inhibitory, else

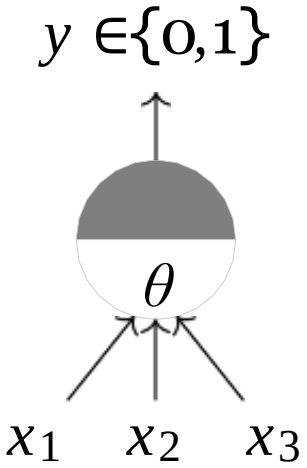
  $g(x_1, x_2, ..., x_n) = g(x) = \sum_{i=1}^{n} x_i$

  $y = f(g(x)) = 1$ if $g(x) \geq \theta$

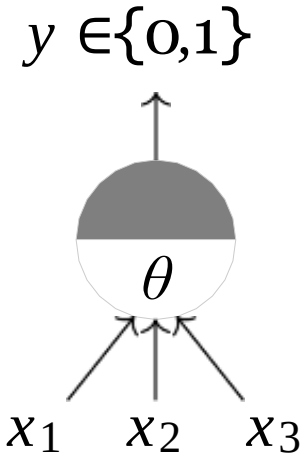
  $= 0$ if $g(x) < \theta$
- θ is called the thresholding parameter
- This is called thresholding logic



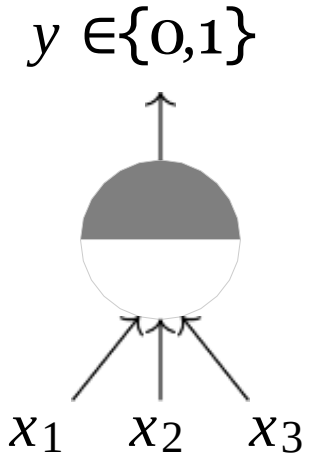
Let us implement some boolean functions using this McCulloch Pitts (MP) neuron...



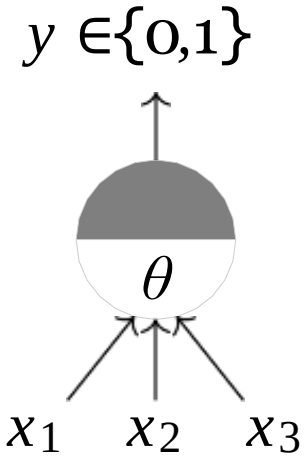
A McCulloch Pitts unit



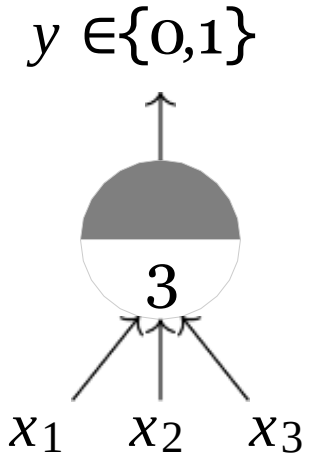
A McCulloch Pitts unit



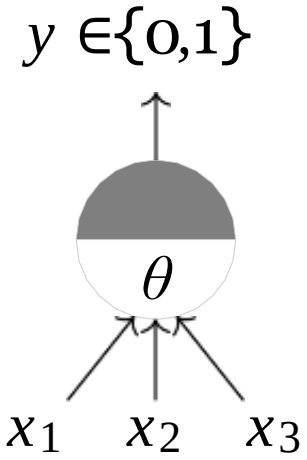
AND function



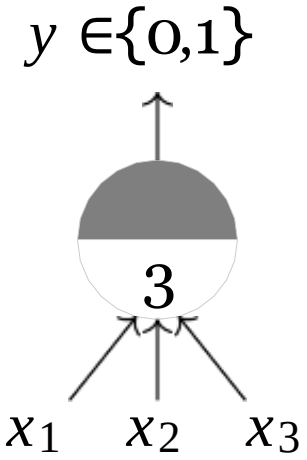
A McCulloch Pitts unit



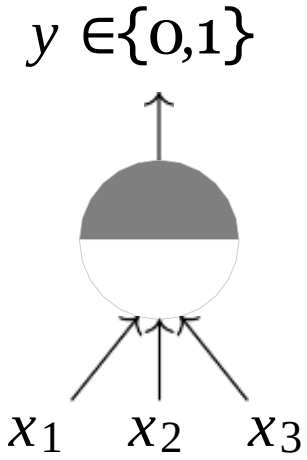
AND function



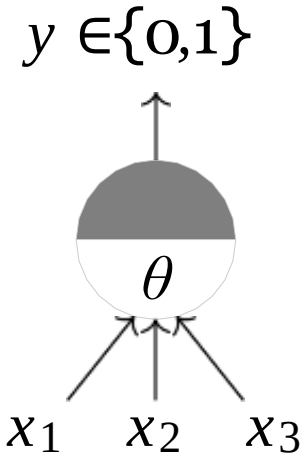
A McCulloch Pitts unit



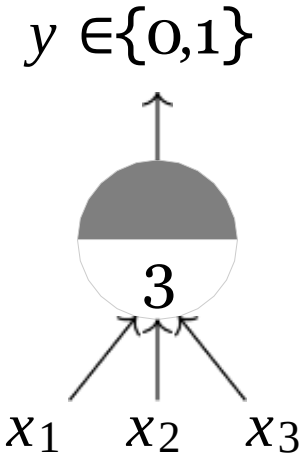
AND function



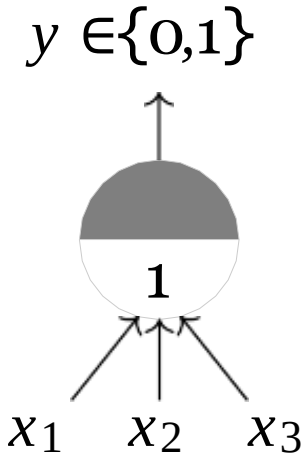
OR function



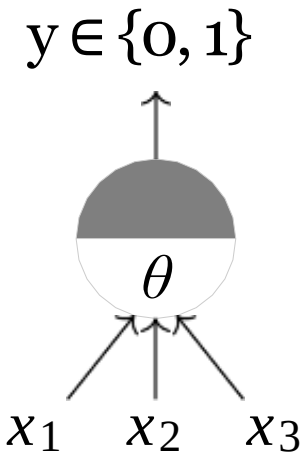
A McCulloch Pitts unit



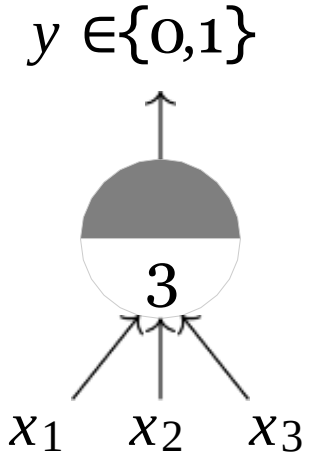
AND function



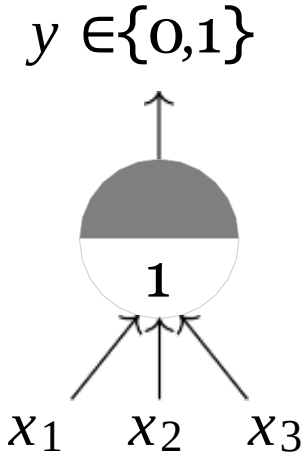
OR function



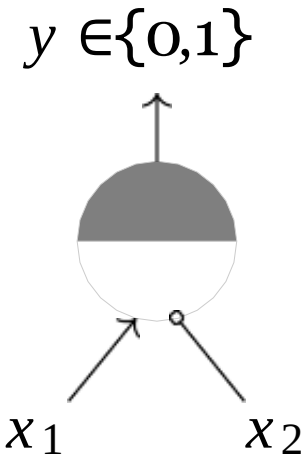
A McCulloch Pitts unit



AND function

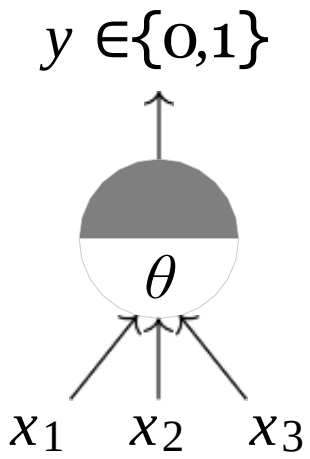


OR function

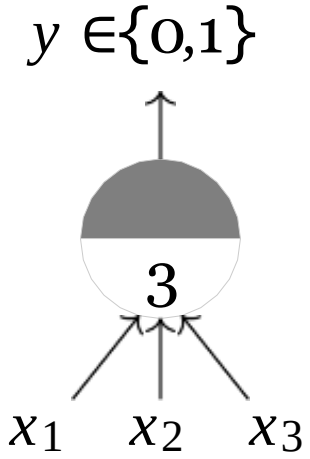


x_1 AND $!x_2$*

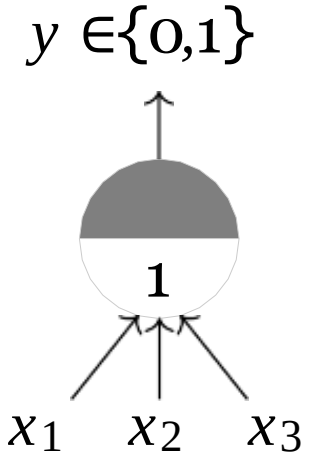
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0



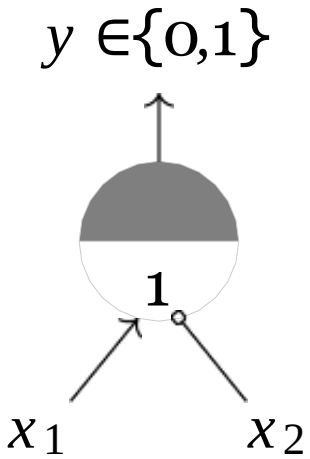
A McCulloch Pitts unit



AND function

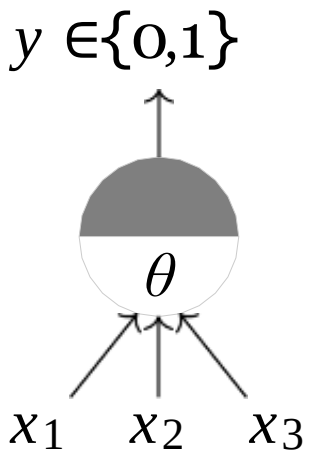


OR function

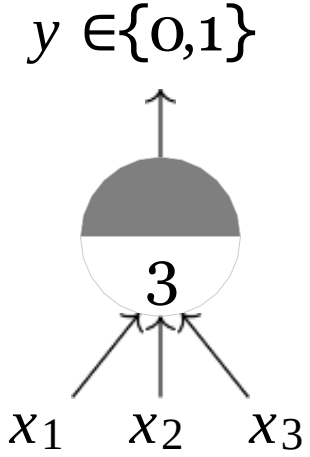


x_1 AND $!x_2$*

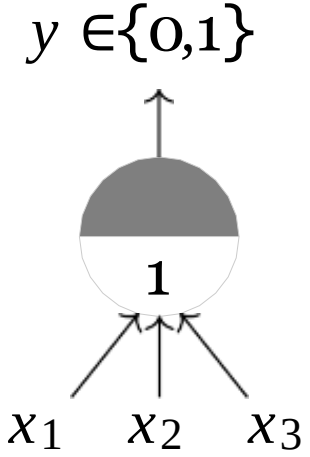
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0



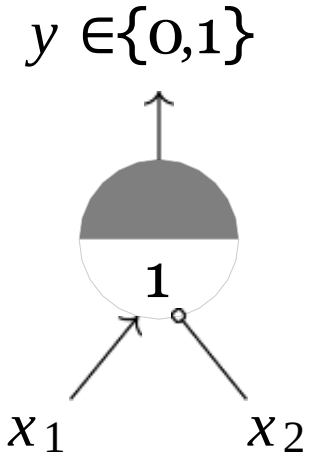
A McCulloch Pitts unit



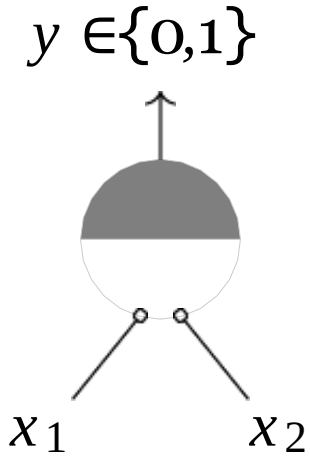
AND function



OR function

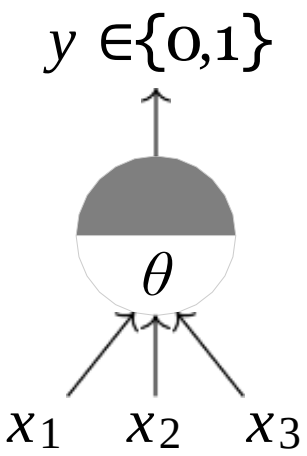


x_1 AND $!x_2$*

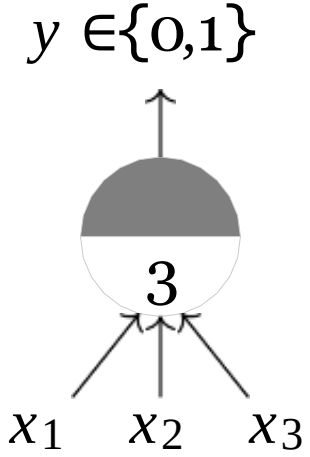


NOR function

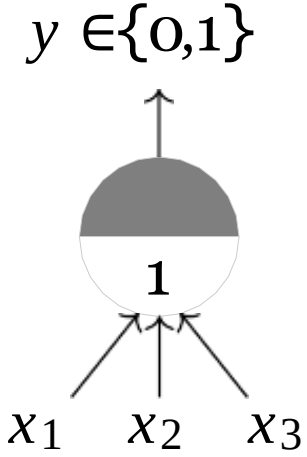
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0



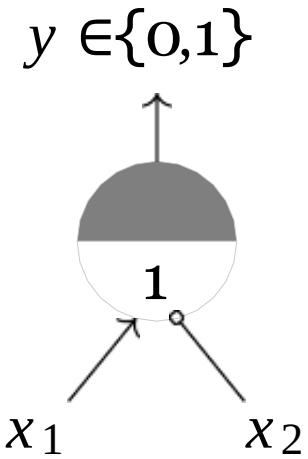
A McCulloch Pitts unit



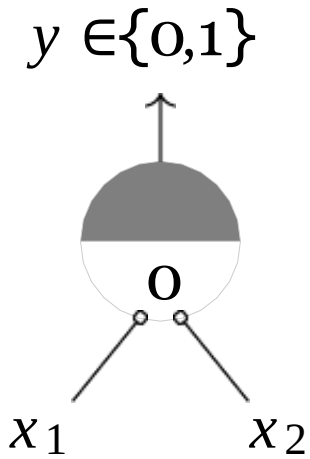
AND function



OR function

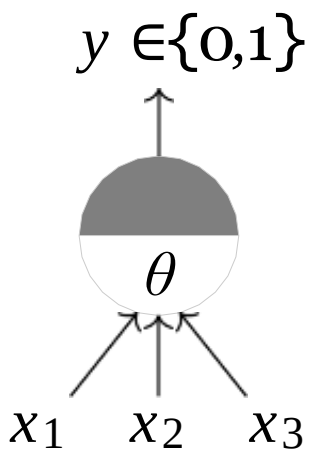


x_1 AND $!x_2$*

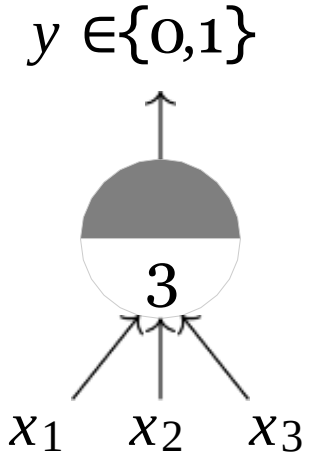


NOR function

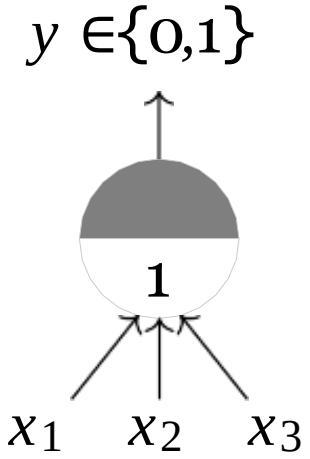
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0



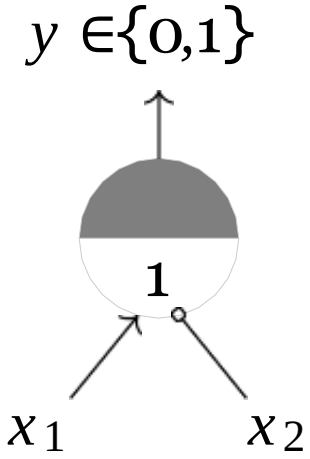
A McCulloch Pitts unit



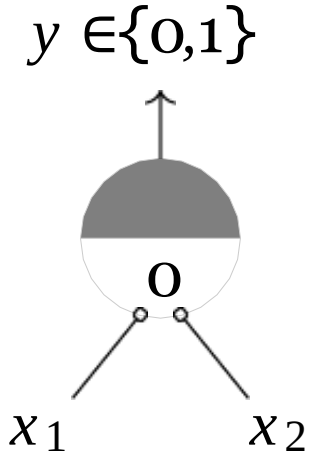
AND function



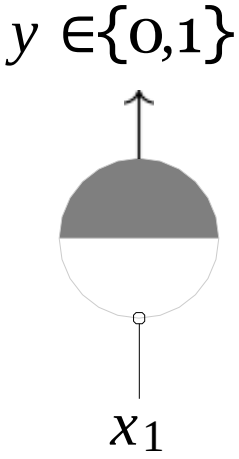
OR function



x_1 AND $!x_2$*

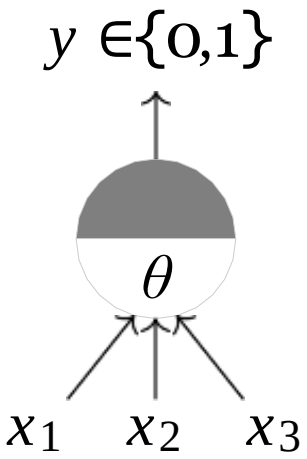


NOR function

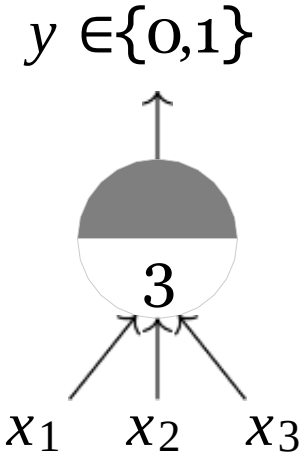


NOT function

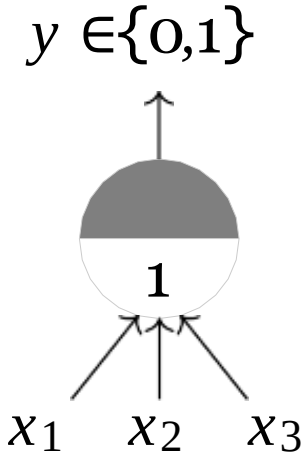
*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0



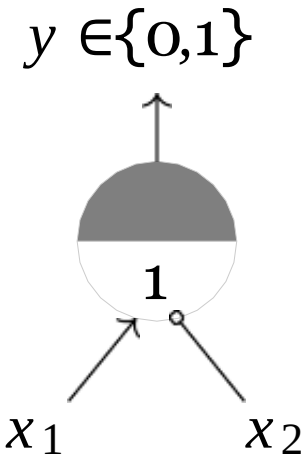
A McCulloch Pitts unit



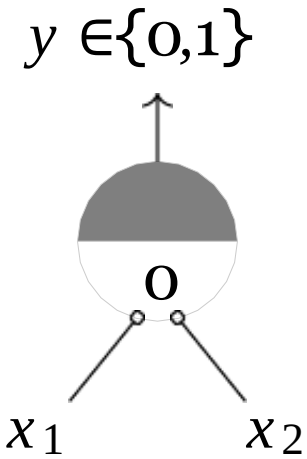
AND function



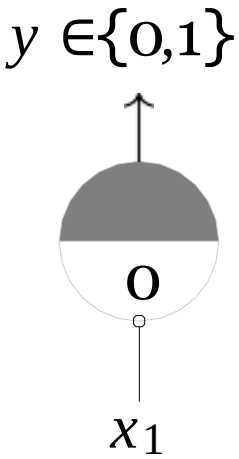
OR function



x_1 AND $!x_2$*



NOR function



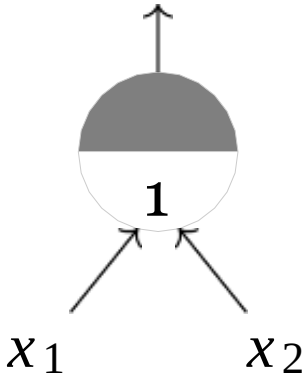
NOT function

*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0

Can any boolean function be represented using a McCulloch Pitts unit ?

Can any boolean function be represented using a McCulloch Pitts unit ?
Before answering this question let us first see the geometric interpretation of a MP unit...

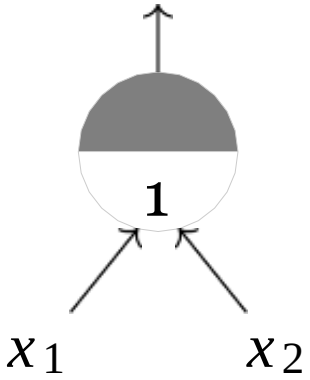
$y \in \{0,1\}$



OR function

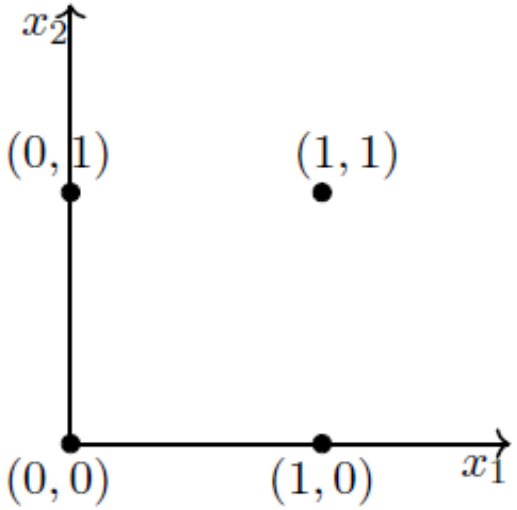
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

$y \in \{0,1\}$

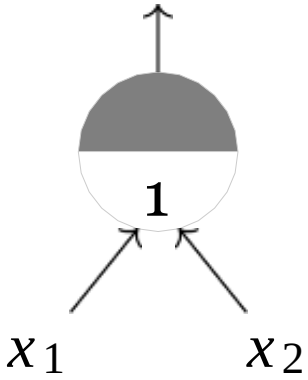


OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

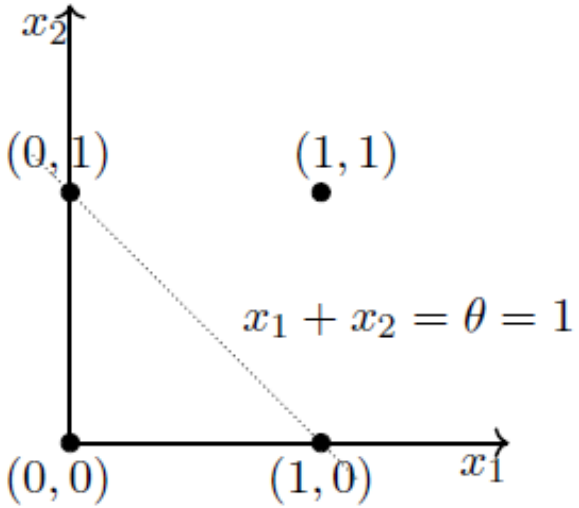


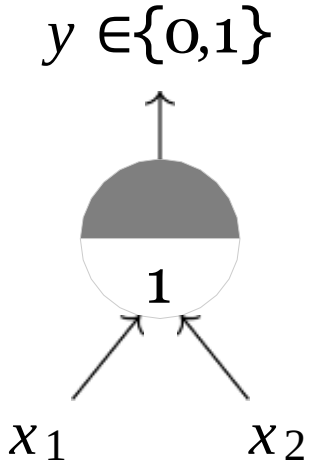
$y \in \{0,1\}$



OR function

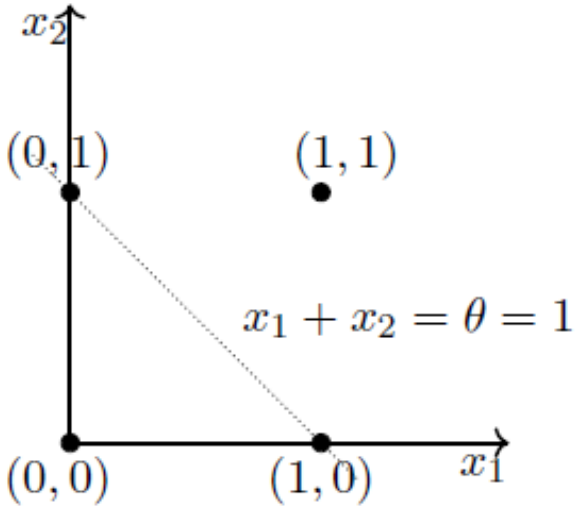
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$



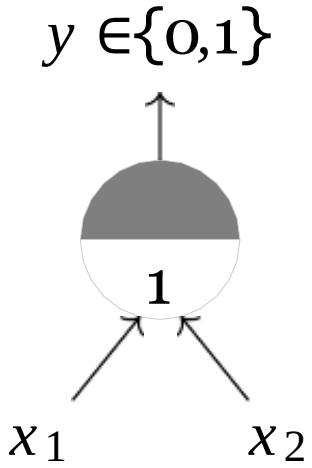


OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

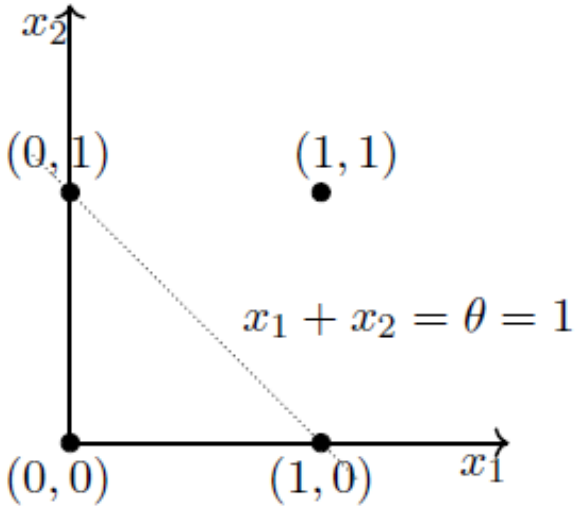


- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves

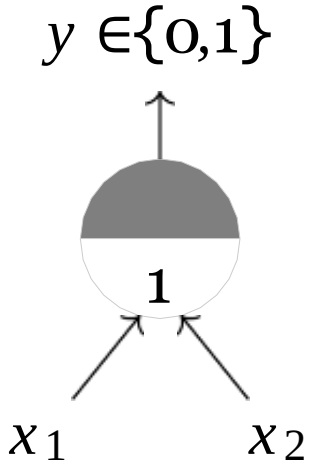


OR function

$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$

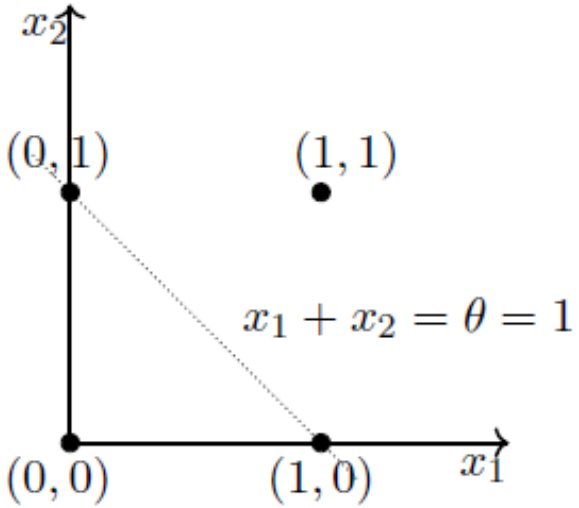


- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves
- Points lying on or above the line $\sum_{i=1}^{n} xi - \theta = 0$ and points lying below this line

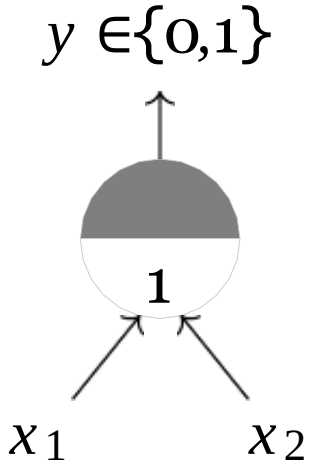


OR function

$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$

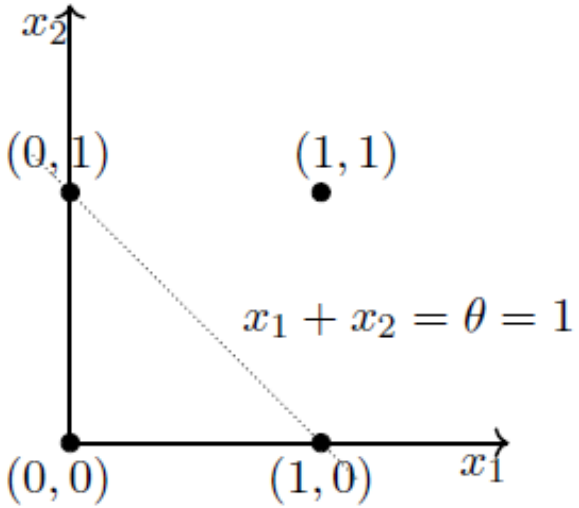


- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves
- Points lying on or above the line $\sum_{i=1}^{n} xi - \Theta = 0$ and points lying below this line
- In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} xi < \Theta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^{n} xi \geq \Theta$)of this line



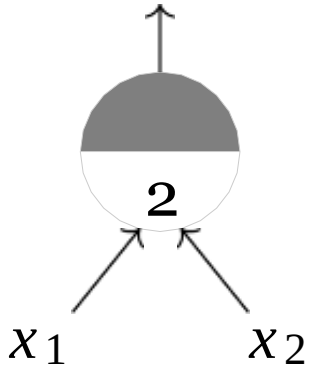
OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$



- A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves
- Points lying on or above the line $\sum_{i=1}^{n} xi - \theta = 0$ and points lying below this line
- In other words, all inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} xi < \theta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^{n} xi \geq \theta$)of this line
- Let us convince ourselves about this with a few more examples (if it is not already clear from the math)

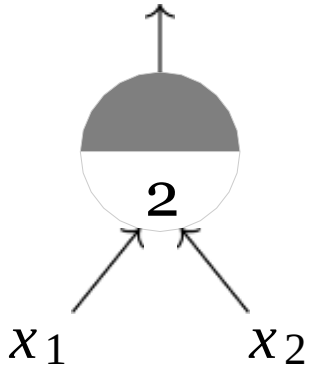
$y \in \{0,1\}$



AND function

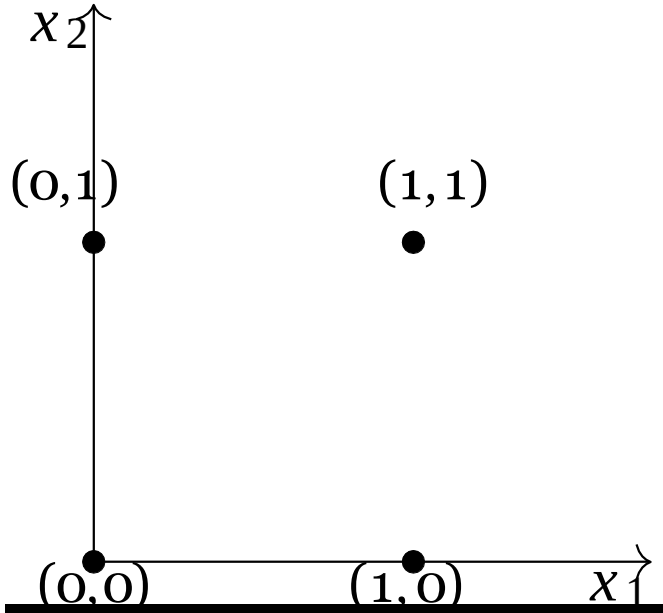
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

$y \in \{0,1\}$

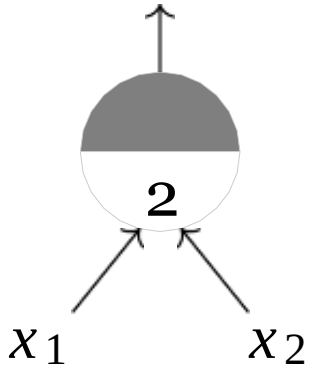


AND function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

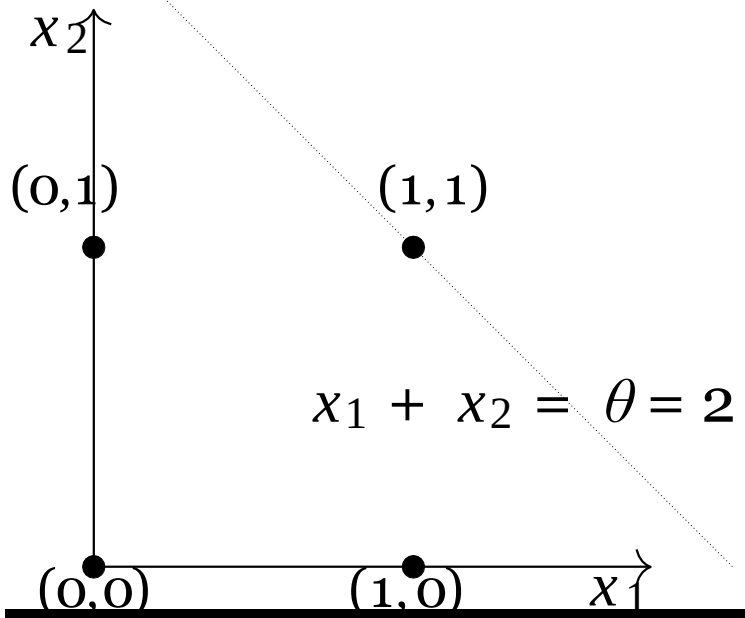


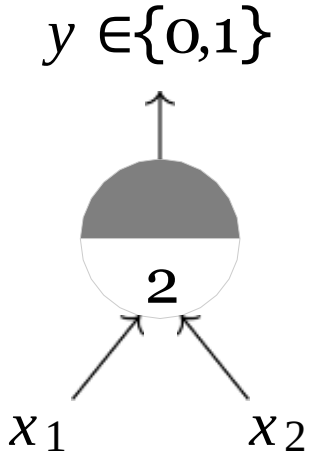
$y \in \{0,1\}$



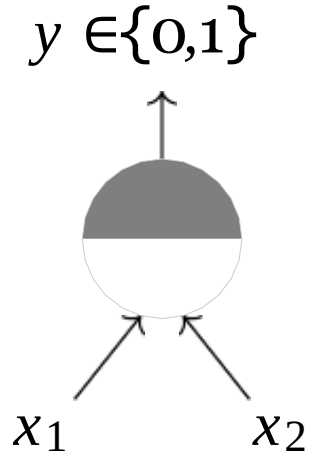
AND function

$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$



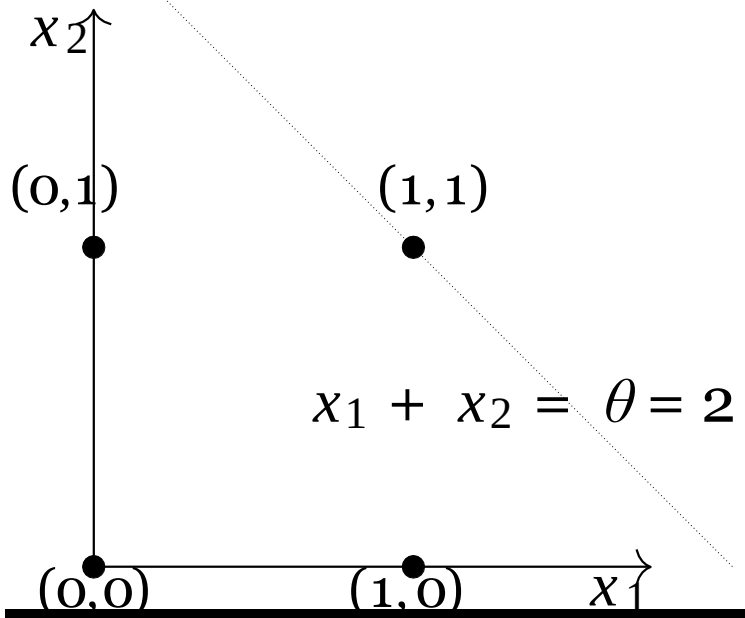


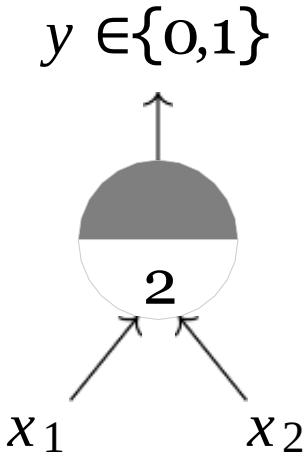
AND function



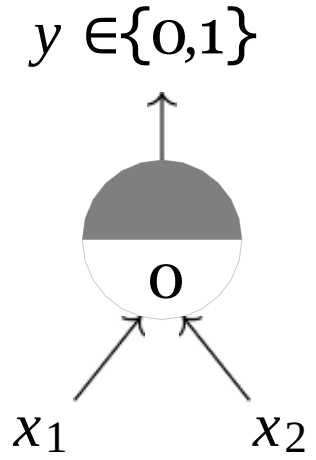
Tautology (always ON)

$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$



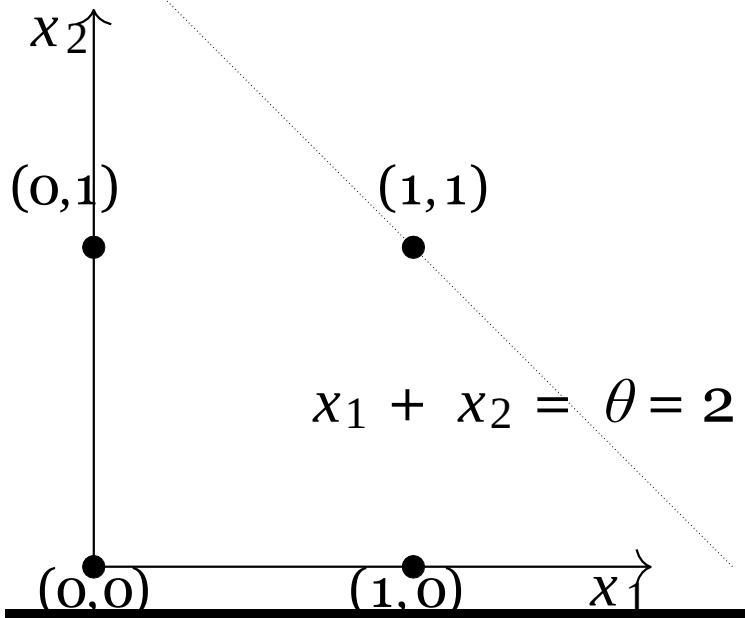


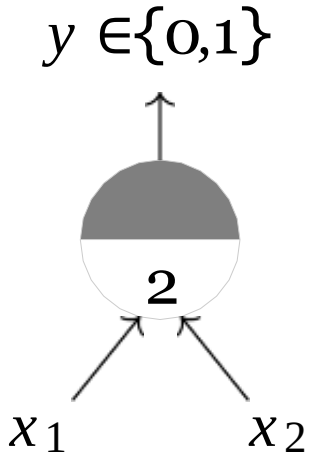
AND function



Tautology (always ON)

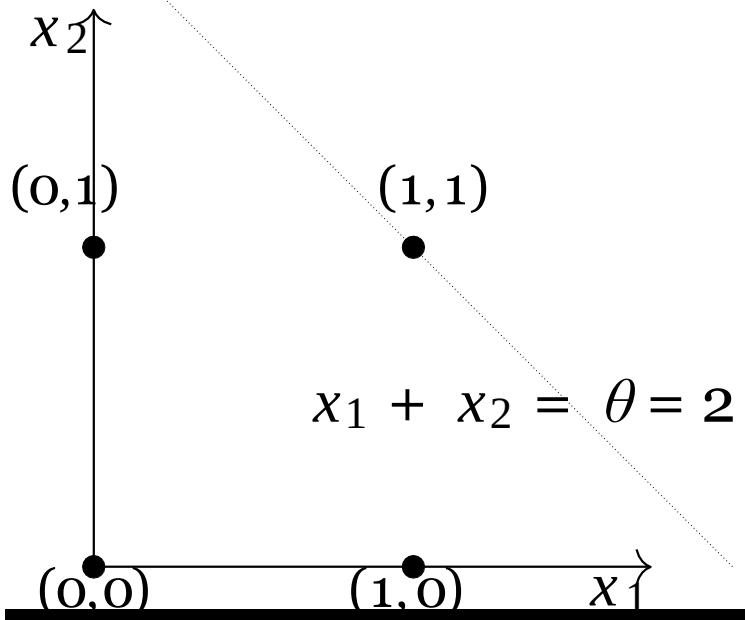
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

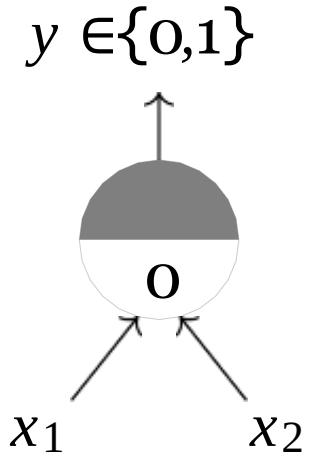




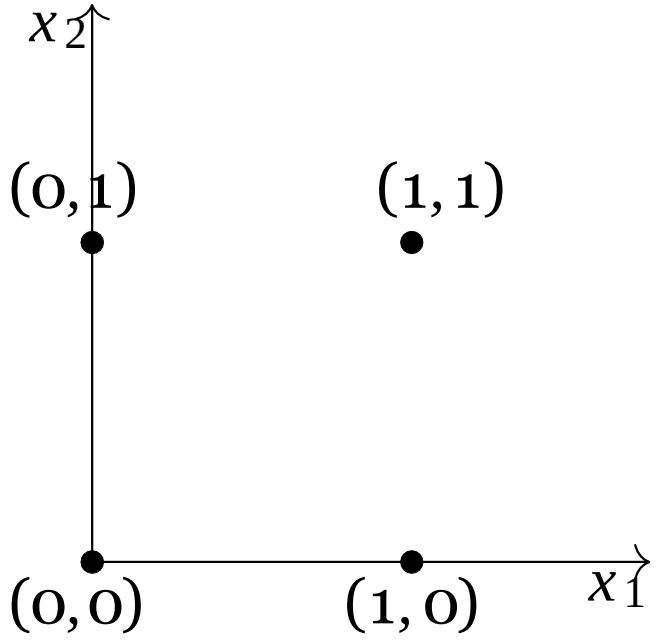
AND function

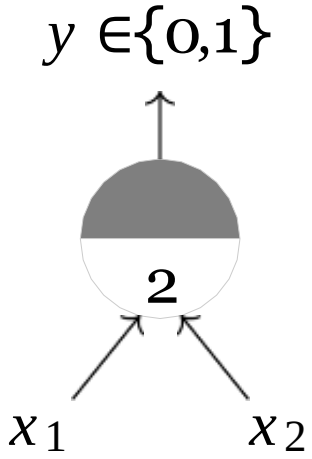
$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$





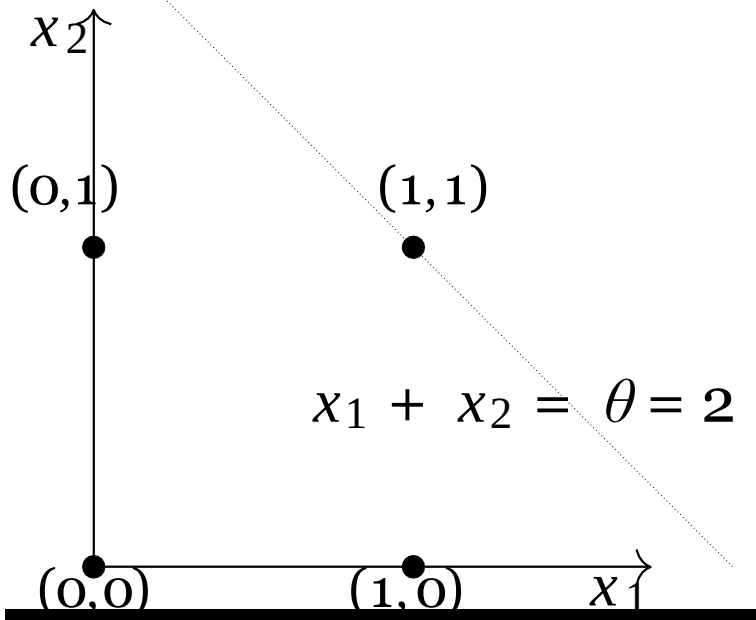
Tautology (always ON)

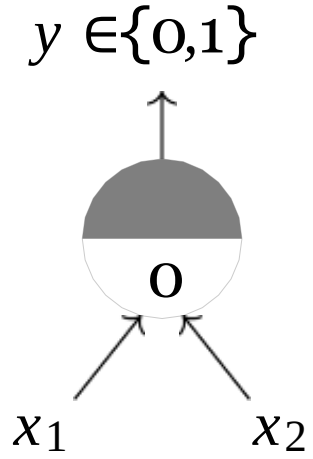




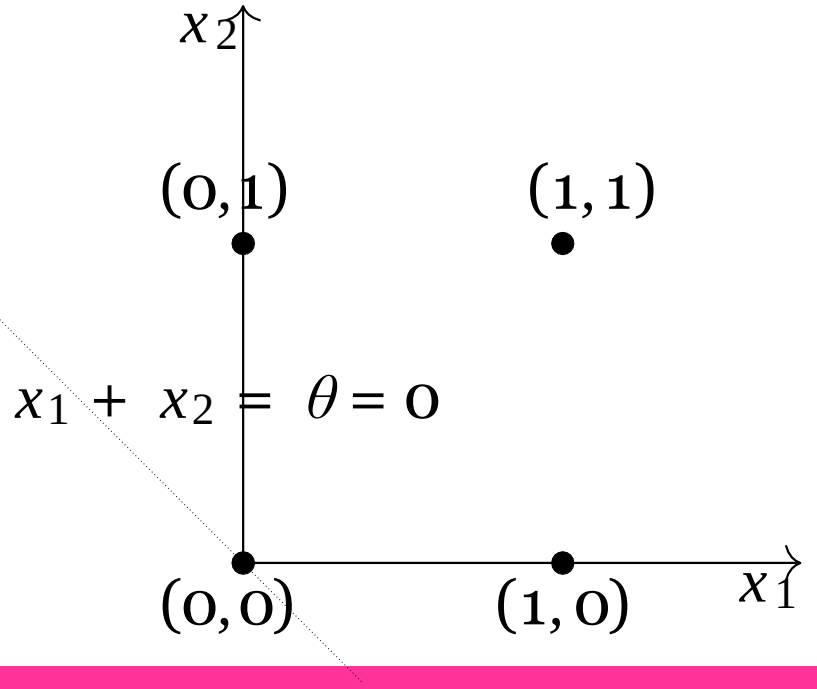
AND function

$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$





Tautology (always ON)



The XOR Problem

- On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s.

The XOR Problem

- On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s.
- A limitation of perceptron architecture is that it is only capable of separating data points with a single line.

The XOR Problem

- On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s.
- A limitation of perceptron architecture is that it is only capable of separating data points with a single line.
- This is unfortunate because the XOR inputs are not linearly separable. This is particularly visible if you plot the XOR input values to a graph.

The XOR Problem

- On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s.
- A limitation of perceptron architecture is that it is only capable of separating data points with a single line.
- This is unfortunate because the XOR inputs are not linearly separable. This is particularly visible if you plot the XOR input values to a graph.
- There is no way to separate the 1 and 0 predictions with a single classification line.

The story so far…

A single McCulloch Pitts Neuron can be used to represent boolean functions which are linearly separable

The story so far…

A single McCulloch Pitts Neuron can be used to represent boolean functions which are linearly separable

Linear separability (for boolean functions): There exists a line (plane) such that all inputs which produce a 1 lie on one side of the line (plane) and all inputs which produce a 0 lie on other side of the line (plane)

6.2.2 Perceptron

The story ahead…

What about non-boolean (say, real) inputs ?

The story ahead…

What about non-boolean (say, real) inputs ?

Do we always need to hand code the threshold ?

The story ahead…

What about non-boolean (say, real) inputs ?

Do we always need to hand code the threshold ?

Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?

The story ahead…

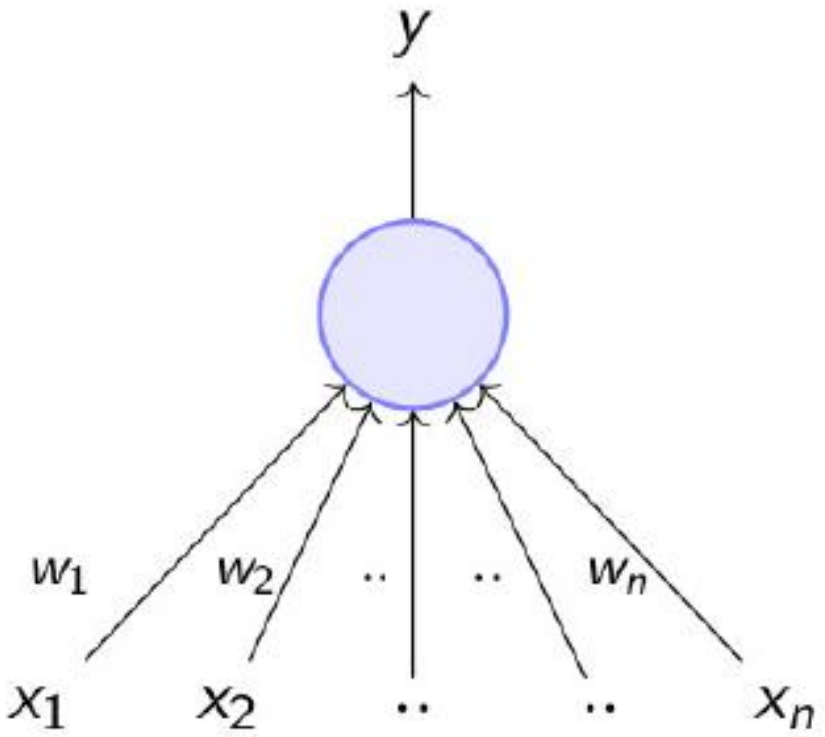
What about non-boolean (say, real) inputs ?

Do we always need to hand code the threshold ?

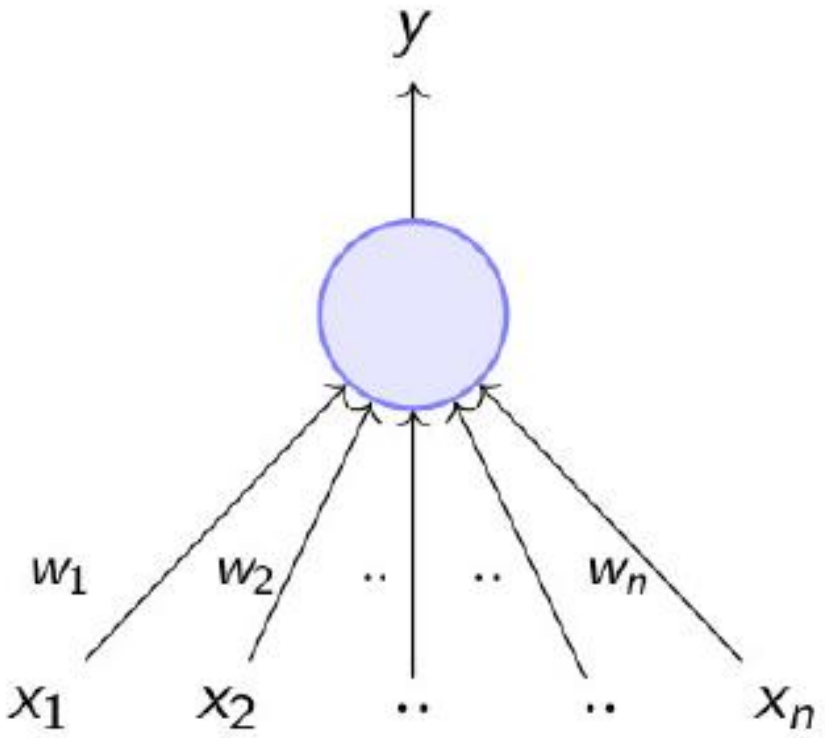
Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?

What about functions which are not linearly separable ?

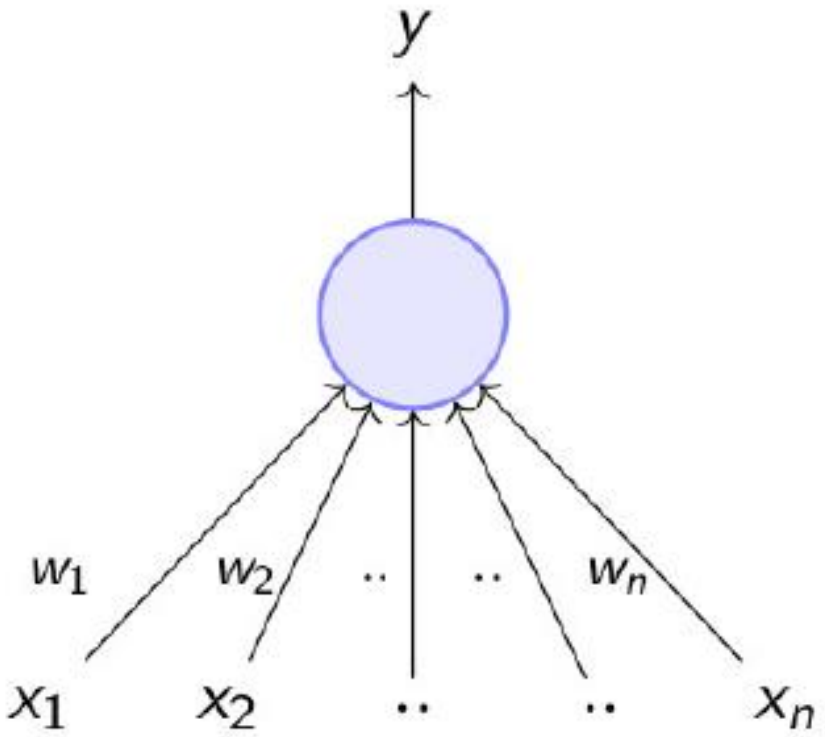
- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model (1958)



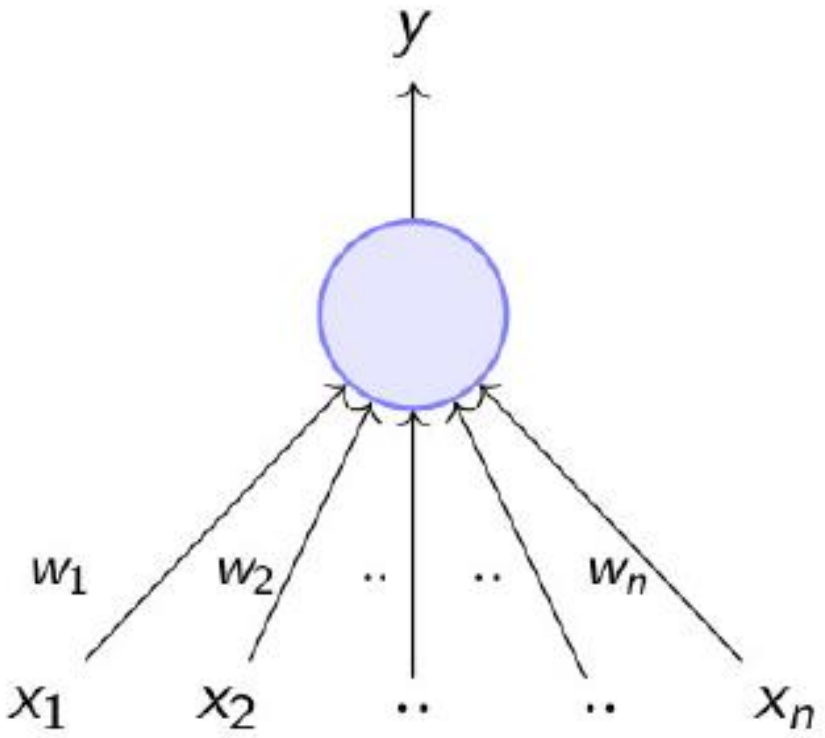
- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model (1958)
- A more general computational model than McCulloch–Pitts neurons



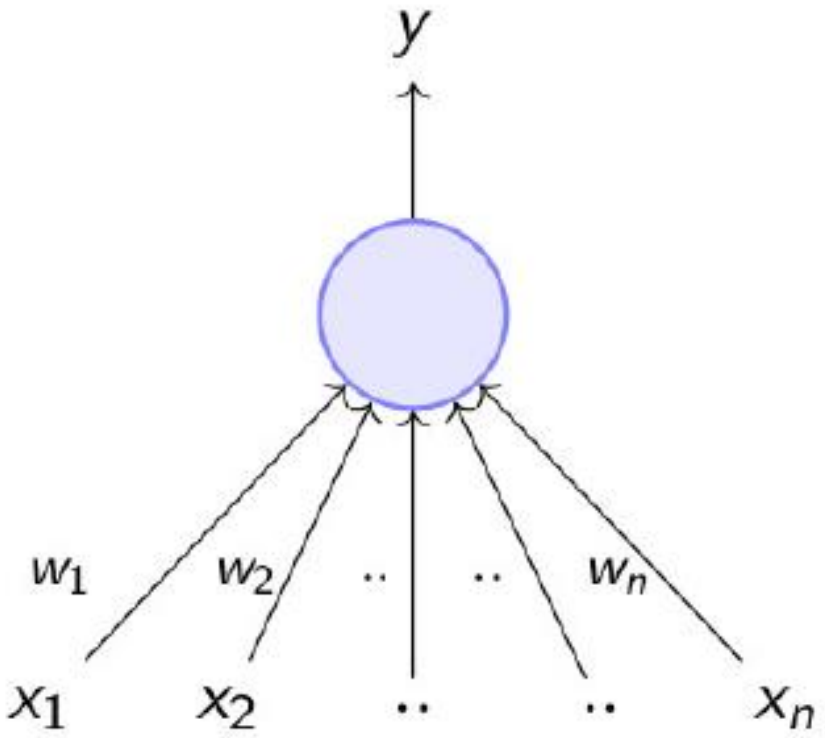
- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model (1958)
- A more general computational model than McCulloch–Pitts neurons
- Main differences: Introduction of numerical weights for inputs and a mechanism for learning these weights

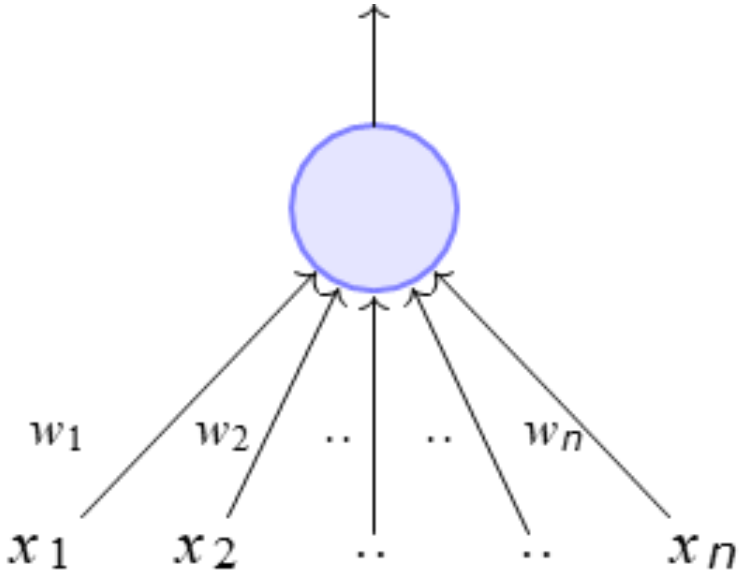


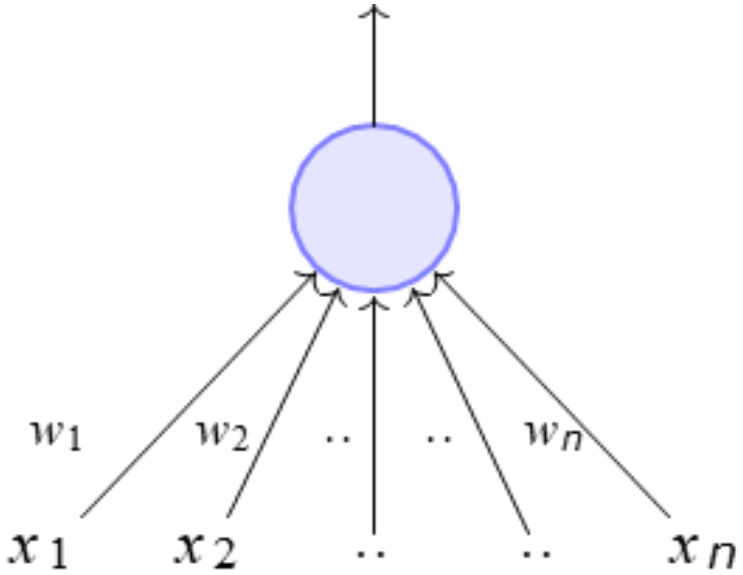
- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model (1958)
- A more general computational model than McCulloch–Pitts neurons
- Main differences: Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values



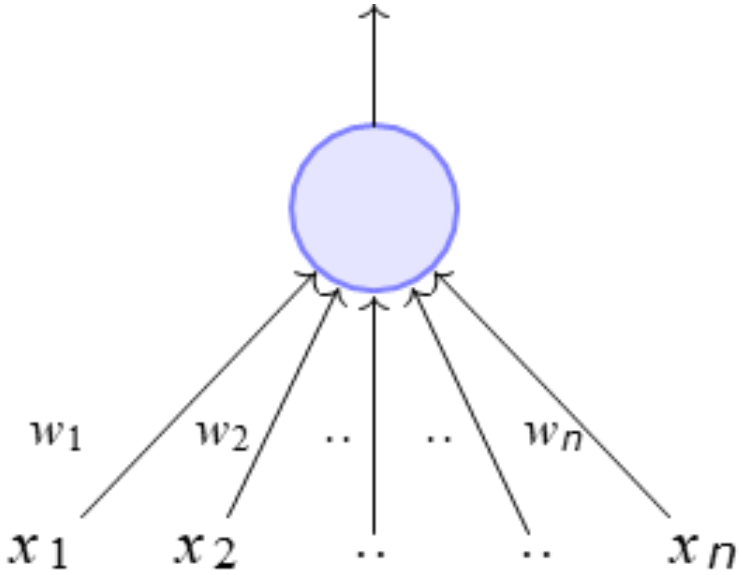
- Frank Rosenblatt, an American psychologist, proposed the classical perceptron model (1958)
- A more general computational model than McCulloch–Pitts neurons
- Main differences: Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values
- Refined and carefully analyzed by Minsky and Papert (1969) - their model is referred to as the perceptron model here




w_1
w_2
..
..
w_n
x_1
x_2
..
..
x_n

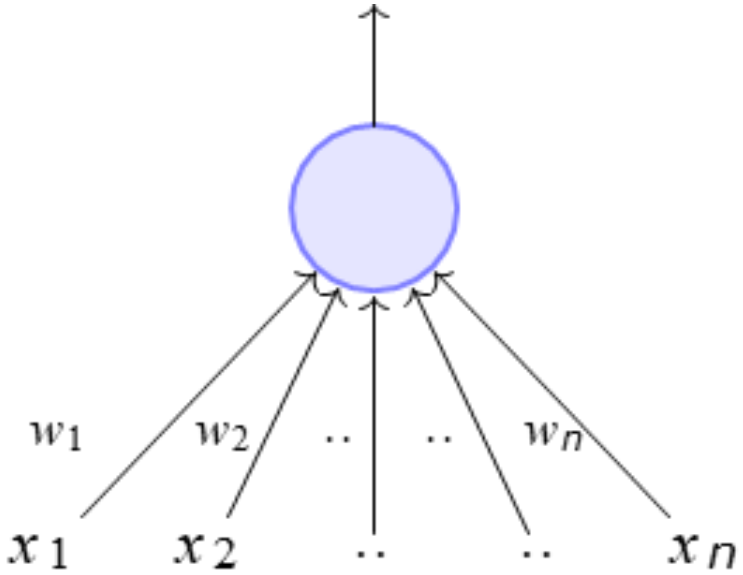


$$y = 1 \text{ if } \sum_{i=1}^{n} w_i * xi \geq \theta$$



$$y = 1 \text{ if } \sum_{i=1}^{n} w_i * xi \geq \theta$$

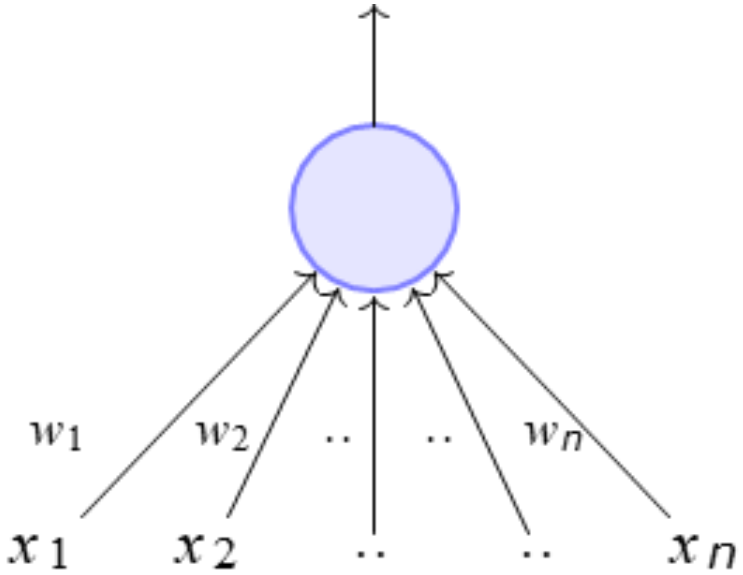
$$= 0 \text{ if } \sum_{i=1}^{n} w_i * xi < \theta$$



$y = 1$ if $\sum_{i=1}^{n} w_i * xi \geq \theta$

$= 0$ if $\sum_{i=1}^{n} w_i * xi < \theta$

Rewriting the above,

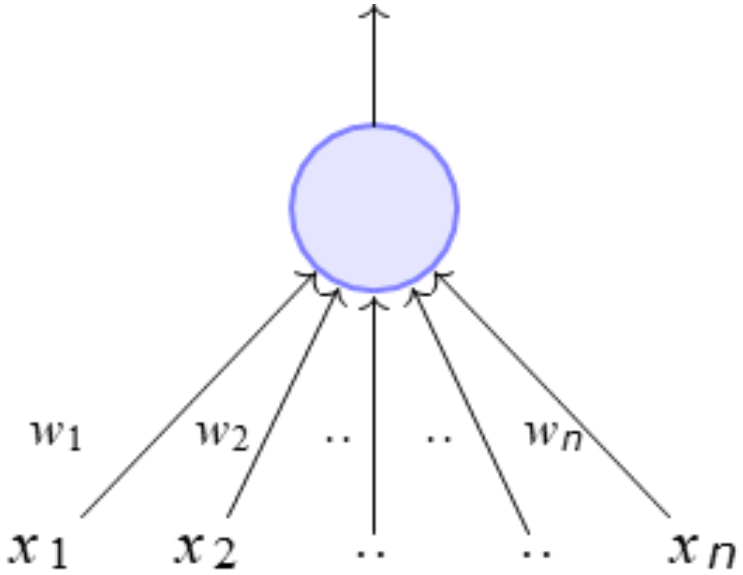


$$y = 1 \text{ if } \sum_{i=1}^{n} w_i * xi \geq \theta$$

$$= 0 \text{ if } \sum_{i=1}^{n} w_i * xi < \theta$$

Rewriting the above,

$$y = 1 \text{ if } \sum_{i=1}^{n} w_i * xi - \theta \geq 0$$



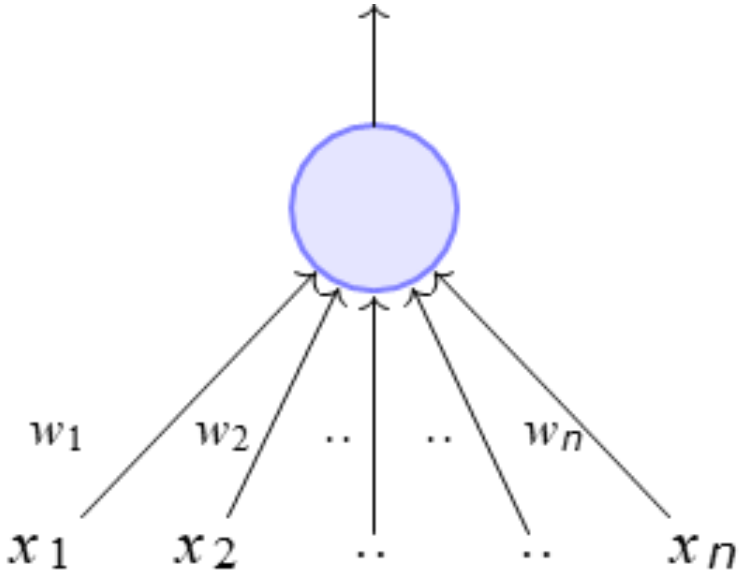
$$y = 1 \text{ if } \sum_{i=1}^{n} w_i * xi \geq \theta$$

$$= 0 \text{ if } \sum_{i=1}^{n} w_i * xi < \theta$$

Rewriting the above,

$$y = 1 \text{ if } \sum_{i=1}^{n} w_i * xi - \theta \geq 0$$

$$= 0 \text{ if } \sum_{i=1}^{n} w_i * xi - \theta < 0$$



$y = 1$ if $\sum_{i=1}^{n} w_i * xi \geq \theta$

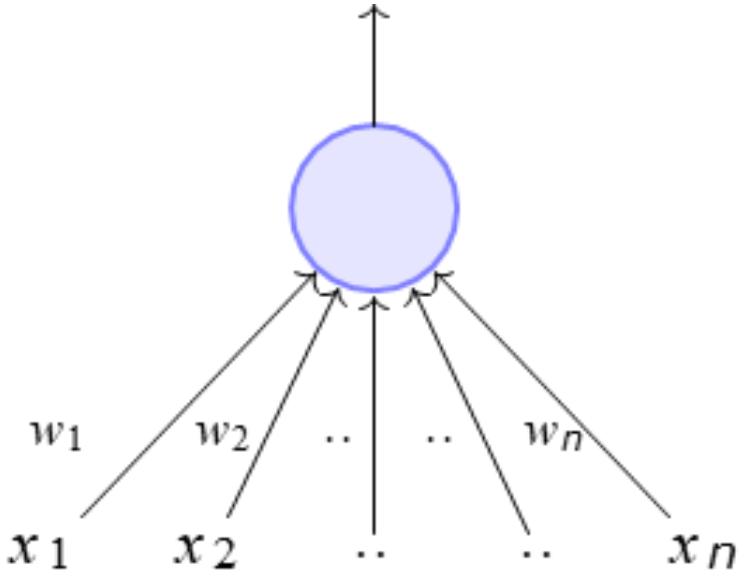
$= 0$ if $\sum_{i=1}^{n} w_i * xi < \theta$

Rewriting the above,

$y = 1$ if $\sum_{i=1}^{n} w_i * xi - \theta \geq 0$

$= 0$ if $\sum_{i=1}^{n} w_i * xi - \theta < 0$

A more accepted convention,



$y = 1$ if $\sum_{i=1}^{n} w_i * xi \geq \theta$

$= 0$ if $\sum_{i=1}^{n} w_i * xi < \theta$

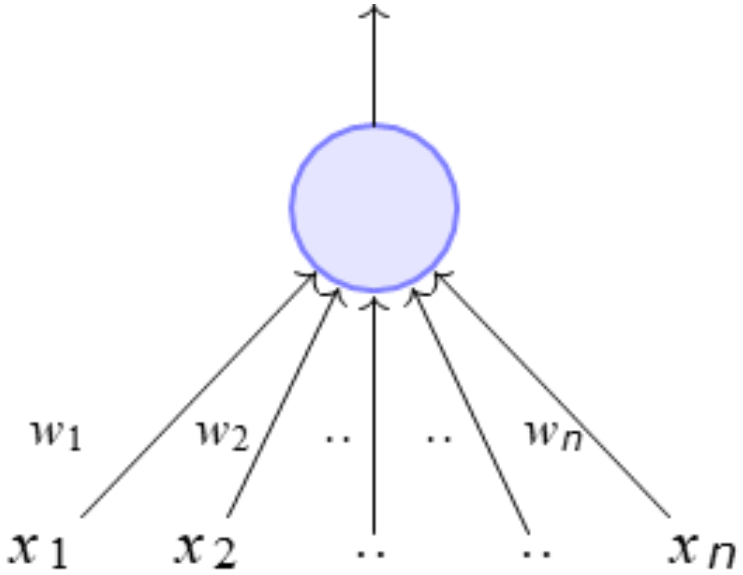
Rewriting the above,

$y = 1$ if $\sum_{i=1}^{n} w_i * xi - \theta \geq 0$

$= 0$ if $\sum_{i=1}^{n} w_i * xi - \theta < 0$

A more accepted convention,

$y = 1$ if $\sum_{i=0}^{n} w_i * xi \geq 0$



$y = 1$ if $\sum_{i=1}^{n} w_i * xi \geq \Theta$

$= 0$ if $\sum_{i=1}^{n} w_i * xi < \Theta$

Rewriting the above,

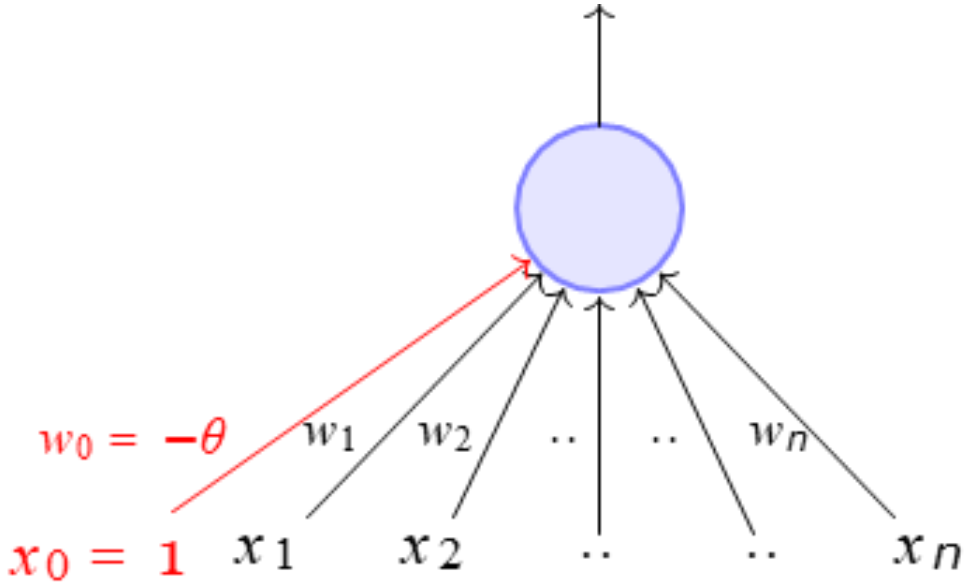
$y = 1$ if $\sum_{i=1}^{n} w_i * xi - \Theta \geq 0$

$= 0$ if $\sum_{i=1}^{n} w_i * xi - \Theta < 0$

A more accepted convention,

$y = 1$ if $\sum_{i=0}^{n} w_i * xi \geq 0$

where, $x_0 = 1$ and $w_0 = -\Theta$



$y = 1$ if $\sum_{i=1}^{n} w_i * xi \geq \Theta$

$= 0$ if $\sum_{i=1}^{n} w_i * xi < \Theta$

Rewriting the above,

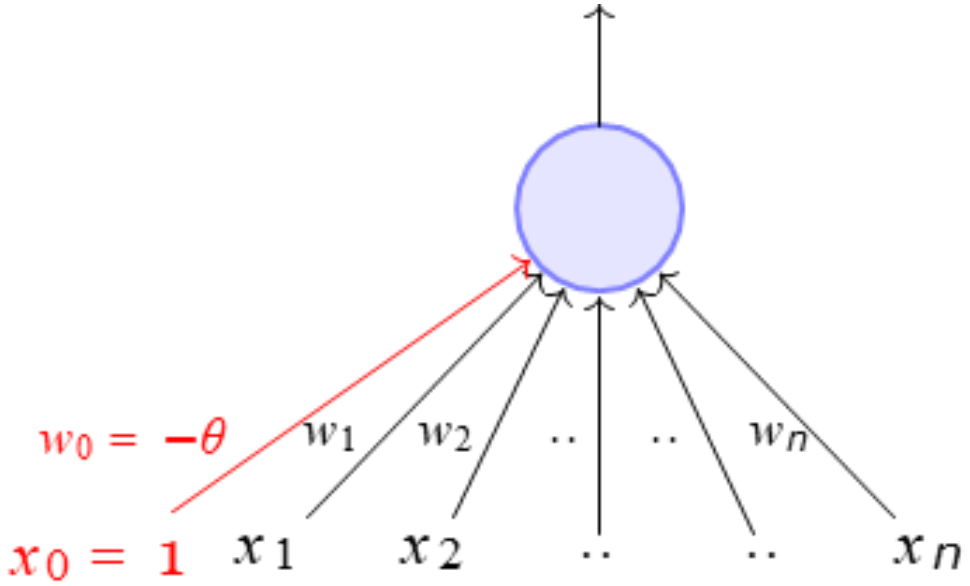
$y = 1$ if $\sum_{i=1}^{n} w_i * xi - \Theta \geq 0$

$= 0$ if $\sum_{i=1}^{n} w_i * xi - \Theta < 0$

A more accepted convention,

$y = 1$ if $\sum_{i=0}^{n} w_i * xi \geq 0$

where, $x_0 = 1$ and $w_0 = -\Theta$



$y = 1$ if $\sum_{i=1}^{n} w_i * xi \geq \Theta$

$= 0$ if $\sum_{i=1}^{n} w_i * xi < \Theta$

Rewriting the above,

$y = 1$ if $\sum_{i=1}^{n} w_i * xi - \Theta \geq 0$

$= 0$ if $\sum_{i=1}^{n} w_i * xi - \Theta < 0$

A more accepted convention,

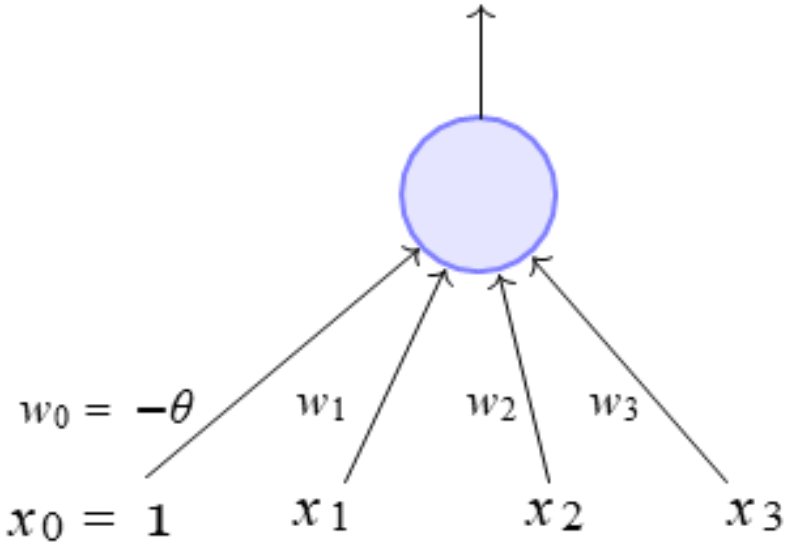
$y = 1$ if $\sum_{i=0}^{n} w_i * xi \geq 0$

$= 0$ if $\sum_{i=0}^{n} w_i * xi < 0$

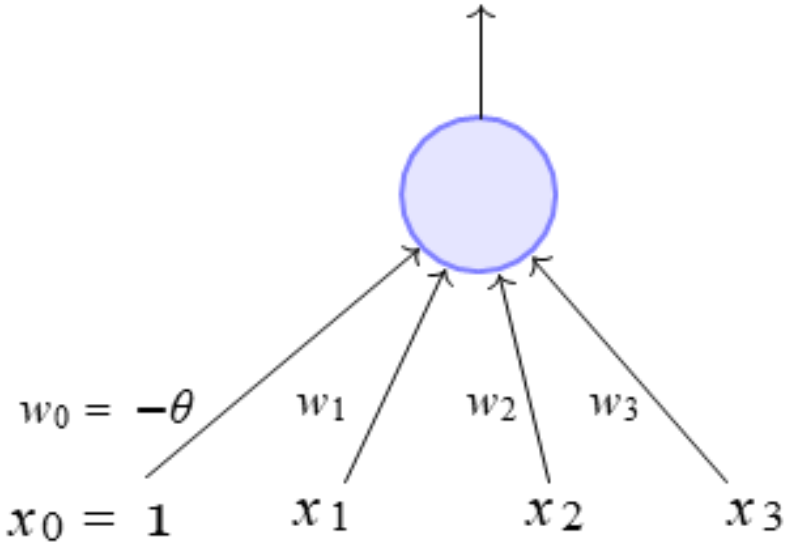
where, $x_0 = 1$ and $w_0 = -\Theta$

We will now try to answer the following questions:

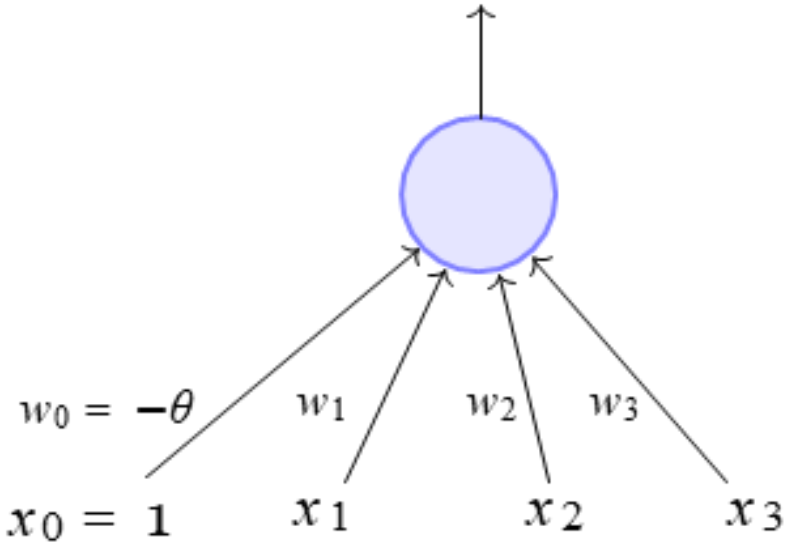
- Why are we trying to implement boolean functions?
- Why do we need weights ?
- Why is $w_0 = -\theta$ called the bias ?



- Consider the task of predicting whether we would like a movie or not



- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)

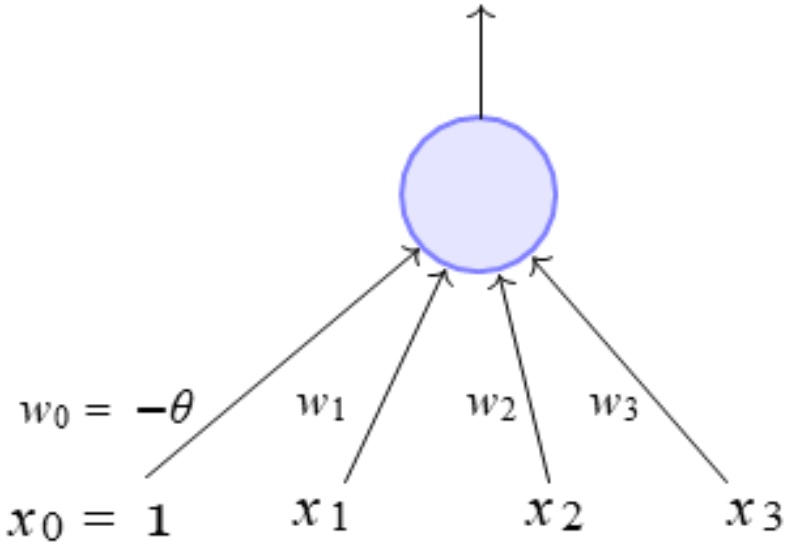


x_1 = isActorDamon

x_2 = isGenreT hriller

x_3 = isDirectorNolan

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)
- Based on our past viewing experience (data), we may give a high weight to isDirectorNolan as compared to the other inputs

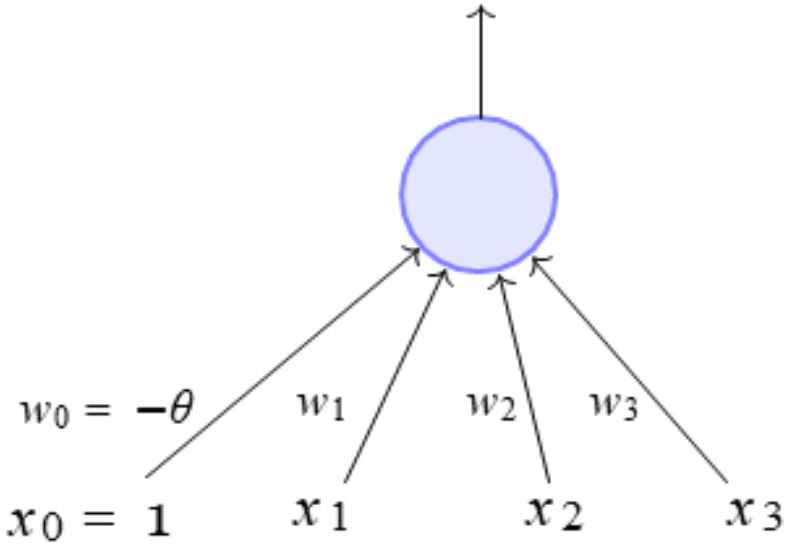


x_1 = isActorDamon

x_2 = isGenreT hriller

x_3 = isDirectorNolan

- Consider the task of predicting whether we would like a movie or not
- Suppose, we base our decision on 3 inputs (binary, for simplicity)
- Based on our past viewing experience (data), we may give a high weight to isDirectorNolan as compared to the other inputs
- Specifically, even if the actor is not Matt Damon and the genre is not thriller we would still want to cross the threshold by assigning a high weight to isDirectorNolan

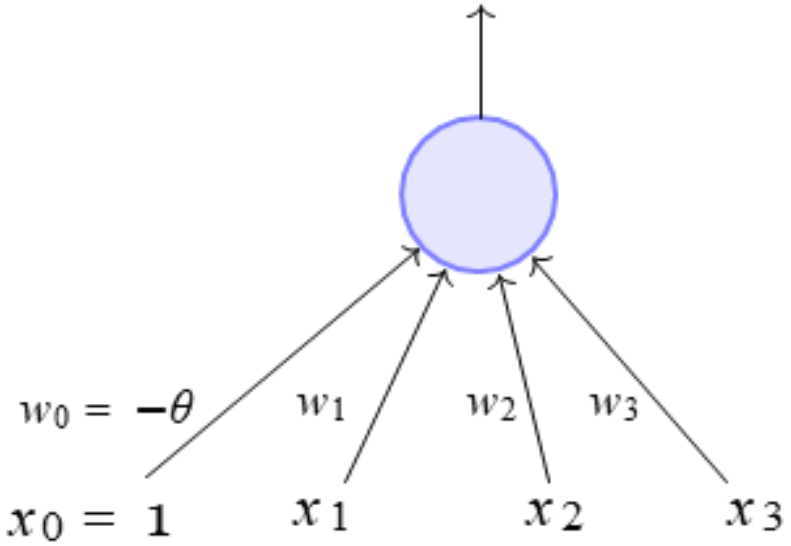


- w0 is called the bias as it represents the prior (prejudice)

x_1 = isActorDamon

x_2 = isGenreT hriller

x_3 = isDirectorNolan

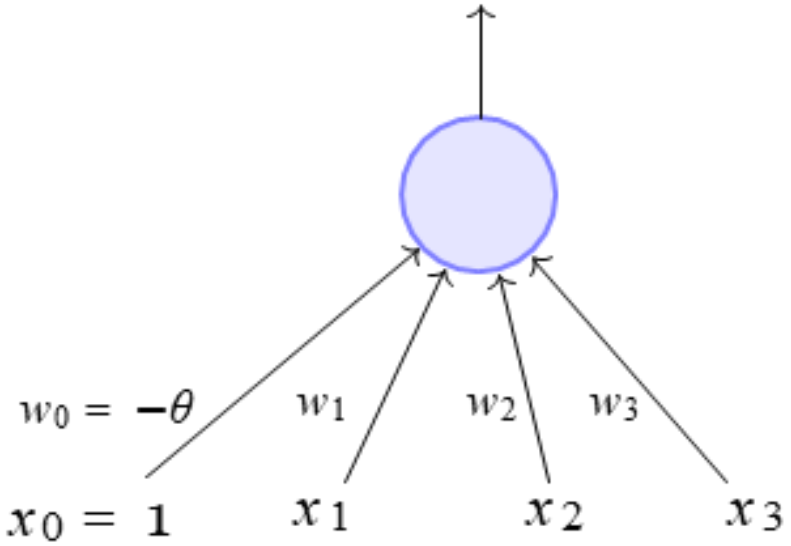


- w0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director [Θ = 0]

x_1 = isActorDamon

x_2 = isGenreT hriller

x_3 = isDirectorNolan

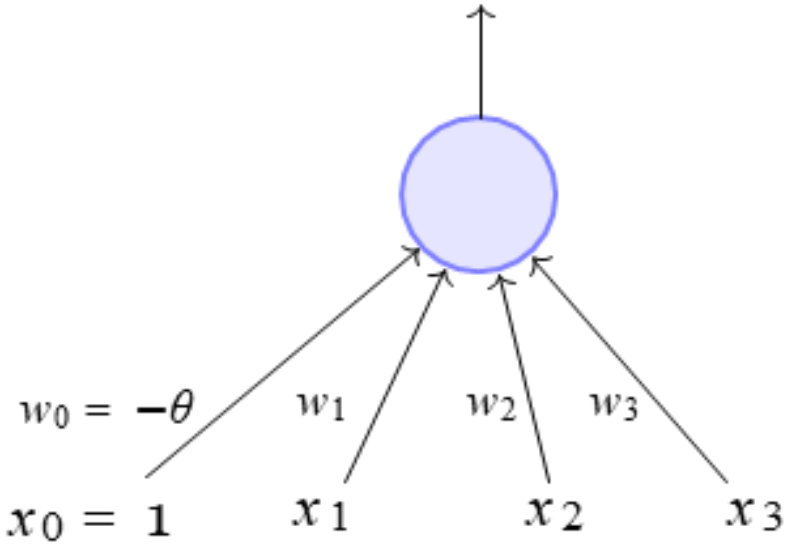


x_1 = isActorDamon

x_2 = isGenreT hriller

x_3 = isDirectorNolan

- w0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director [Θ = 0]
- On the other hand, a selective viewer may only watch thrillers starring Matt Damon and directed by Nolan [Θ = 3]



x_1 = isActorDamon

x_2 = isGenreT hriller

x_3 = isDirectorNolan

- w0 is called the bias as it represents the prior (prejudice)
- A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor, director [Θ = 0]
- On the other hand, a selective viewer may only watch thrillers starring Matt Damon and directed by Nolan [Θ = 3]
- The weights (w_1, w_2, …, w_n) and the bias (w_0) will depend on the data (viewer history in this case)

What kind of functions can be implemented using the perceptron? Any difference from McCulloch Pitts neurons?

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \text{ if } \sum_{i=0}^{n} xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} xi < 0$$

Perceptron

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i * xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i * xi < 0$$

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \text{ if } \sum_{i=0}^{n} xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} xi < 0$$

Perceptron

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i * xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i * xi < 0$$

- From the equations it should be clear that even a perceptron separates the input space into two halves

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \text{ if } \sum_{i=0}^{n} xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} xi < 0$$

Perceptron

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i * xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i * xi < 0$$

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \text{ if } \sum_{i=0}^{n} xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} xi < 0$$

Perceptron

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i * xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i * xi < 0$$

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side
- In other words, a single perceptron can only be used to implement linearly separable functions

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \text{ if } \sum_{i=0}^{n} xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} xi < 0$$

Perceptron

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i * xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i * xi < 0$$

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side
- In other words, a single perceptron can only be used to implement linearly separable functions
- Then what is the difference?

McCulloch Pitts Neuron

(assuming no inhibitory inputs)

$$y = 1 \text{ if } \sum_{i=0}^{n} xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} xi < 0$$

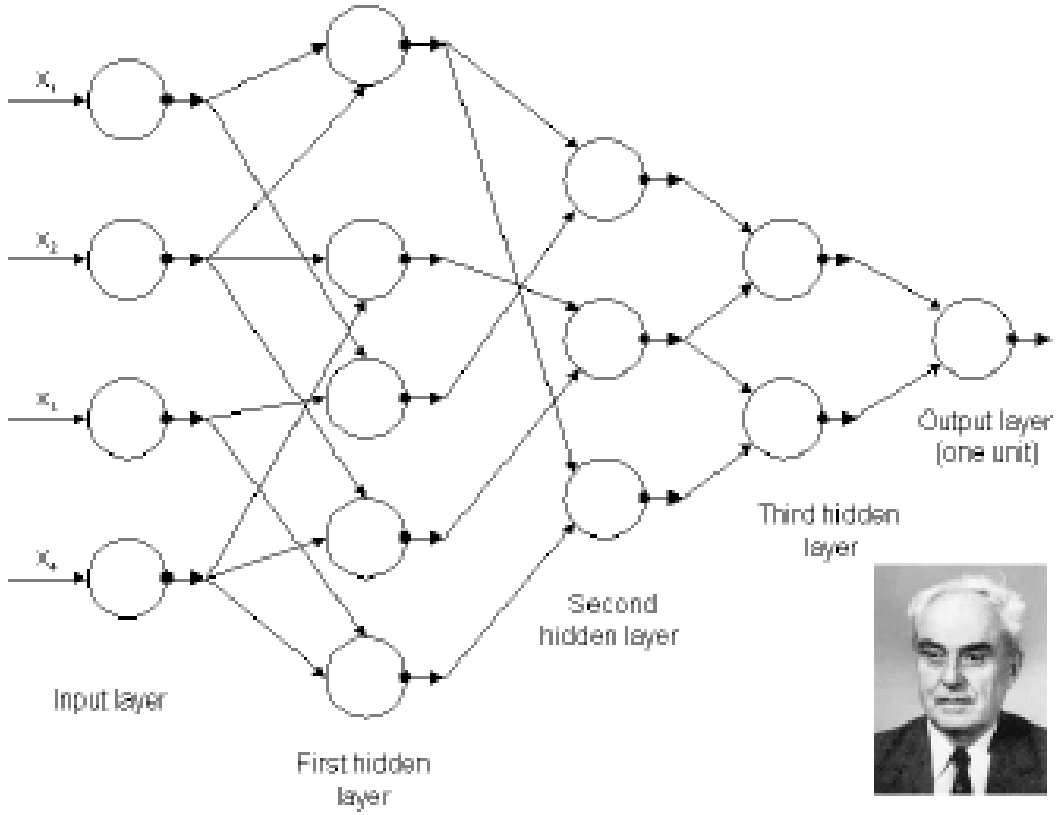
Perceptron

$$y = 1 \text{ if } \sum_{i=0}^{n} w_i * xi \geq 0$$
$$= 0 \text{ if } \sum_{i=0}^{n} w_i * xi < 0$$

- From the equations it should be clear that even a perceptron separates the input space into two halves
- All inputs which produce a 1 lie on one side and all inputs which produce a 0 lie on the other side
- In other words, a single perceptron can only be used to implement linearly separable functions
- Then what is the difference? The weights (including threshold) can be learned and the inputs can be real valued

First Generation Multilayer Perceptrons

- Came around in 1965-68
- Ivakhnenko et. al



6.3 Expert Systems

- According to Feigenbaum, **expert system (ES)** is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution.
- An expert system is a computer system that emulates the decision making of a human expert.
- The expert knowledge is stored in the computer in an organized manner.
- This so called knowledge base is used to provide advice.
- ES does the same reasoning process that a human decision maker would go through to arrive at a decision.

- Expert system is a branch of Artificial Intelligence but it differs from others in that:
  - It deals with subject matter of realistic complexity
  - It must exhibit high performance
  - It must be plausible

Knowledge Engineering

- The process of building an expert system is called knowledge engineering.
- Knowledge Engineers acquire the knowledge from a human expert or other source and code in the expert system.
- The problem of transferring human knowledge into an expert system is so major that it is called the knowledge acquisition bottleneck.
- Major bottlenecks are due to: cognitive barrier, linguistic barrier, representation barrier and the problem of creating model.

Conventional Programs vs ES

- ESs differ from the conventional computer programs in the following aspects:
  - ESs are knowledge intensive programs
  - ESs are highly interactive
  - ESs mimic human experts in decision making and reasoning process
  - ESs divides expert knowledge into number of separate rules
  - ESs are user friendly and intelligent.

ES Development

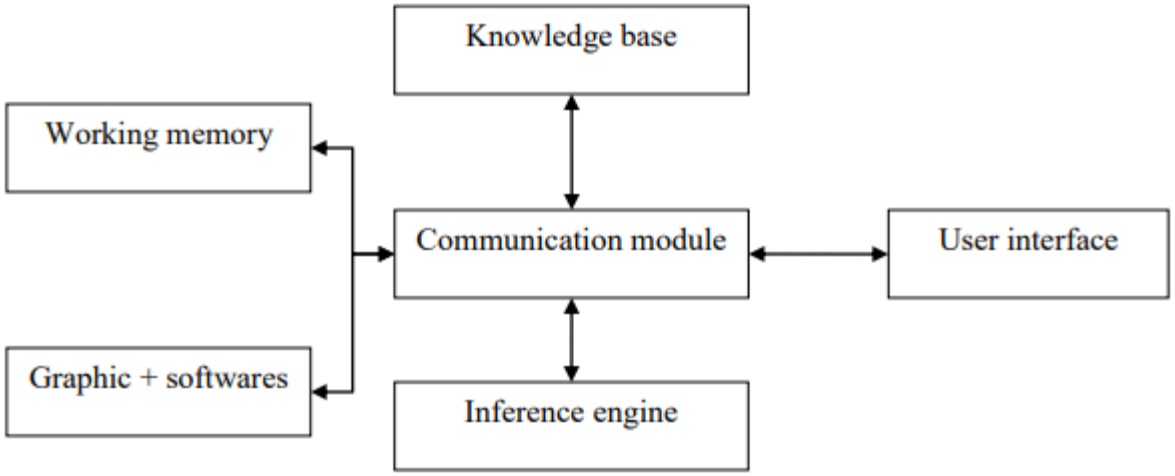
- Hayes – Roth and Lenat (1983) have recommended five stages in the development of ES:
  1. Identification – determining characteristics of the problem
  2. Conceptualisation – finding concepts to represent the knowledge
  3. Formalisation – designing structures to organize knowledge
  4. Implementation – formulating rules embodying the knowledge
  5. Testing – validating the rules A good coordination between the knowledge engineer and the expert is necessary.

ES Tools

- Language: A translator of commands written in a specific syntax.
  - An expert system language will also provide an inference engine to execute the statement of the language
- Shells: A special purpose tool designed for certain types of applications in which the user must only supply the knowledge base (e.g. EMYCIN)
- Tools: A language + utility programs to facilitate the development debugging, and delivery of application programs

ES Architecture

- A ES is specific to one problem domain.
- However, it is not for domain-modeling but for problem-solving.
- The expert system consists of:
  - a knowledge base,
  - a working memory,
  - an inference engine,
  - system analysis, graphic and other softwares and
  - user interface.



- Knowledge base: A declarative representation of the expertise; often in IF THEN rules;
- Working storage: The data which is specific to a problem being solved;
- Inference engine: The code at the core of the system which derives recommendations from the knowledge base and problem-specific data in working storage;
- User interface: The code that controls the dialog between the user and the system.

Roles of Individuals who Interact with the System

- Domain expert: The individuals who currently are experts in solving the problems; here the system is intended to solve;
- Knowledge engineer: The individual who encodes the expert's knowledge in a declarative form that can be used by the expert system;
- User: The individual who will be consulting with the system to get advice which would have been provided by the expert.

Knowledge Base

- Knowledge base module contains domain specific knowledge.
- Knowledge can be either:
  - Priori knowledge which comes before and is independent of knowledge from the senses. It is considered to be universally true and cannot be denied without contradiction. Ex: All triangles in the plane have 180 degrees.
  - Posteriori knowledge that is derived from the senses. It can be denied on the basis of new knowledge without the necessity of contradictions. Ex: The light is green.
- Knowledge can be represented in various forms as: rules, semantic nets, frames, scripts, object-oriented, others- KL-1, KRYPTON, conceptual graphs and so on.

Rules

- The most popular format of rules are the IF–condition– THEN – action statements.
- This is useful when the knowledge is in the form of condition action.
- P_1, ..., P_m ==> Q_1, ..., Q_n means if premises P_1 and …, and P_m are true, then perform actions Q_1 and …, and Q_n.
- An example of a rule is:

  IF Inflow < 0.7 * Average

  AND Storage < Capacity/2

  THEN irrigation release = 0.6* Demand

Inference Engine

- This module examines the knowledge base and answers the questions (how and why) from the user.
- It is the most crucial component of ES.
- It derives the knowledge i.e., guides the selection of a proper response to a specific situation which is called pruning.
- Three formal approaches used in this case are:
  - production rules,
  - structured objects and
  - predicate logic.

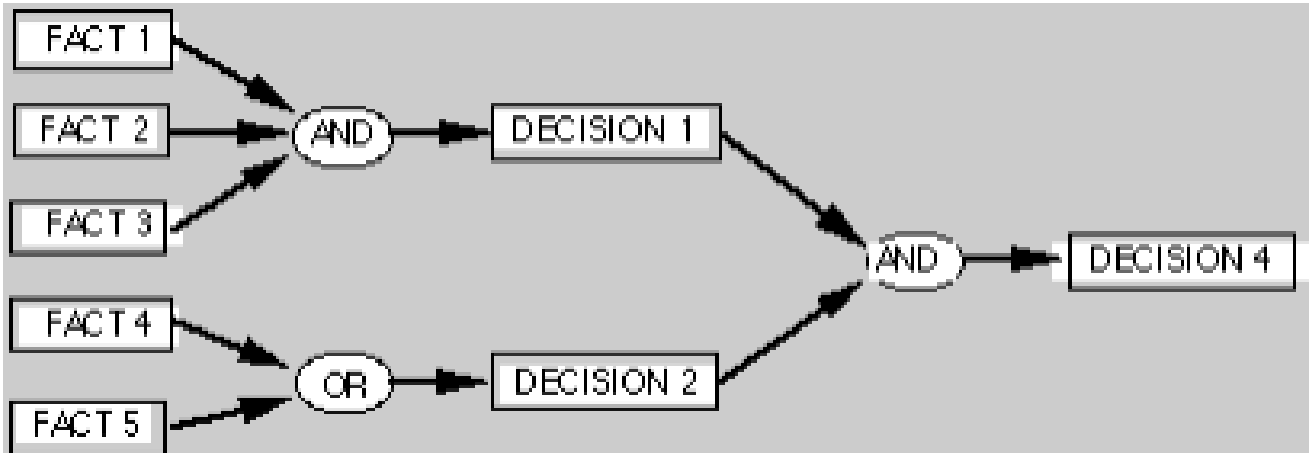
- Production rules consist of a rule set, a rule interpreter which specifies when and how to apply the rules and a working memory which holds the data, goals and intermediate results.
- Structured objects use vector representation of essential and accidental properties.
- Predicate logic uses propositional and predicate calculi.
- The inference engine can work in the following ways:
    1. Forward Chaining
    2. Backward Chaining
    3. Abduction
    4. Reasoning under Uncertainty

Inference engine – Explanations

- An expert system seeks to make problem solving knowledge explicit.
- The knowledge applied to a problem must be available to the user.
- The system must be able to explain how it arrived at a conclusion and why it is performing some computation.
- It may also be required to answer what if questions.

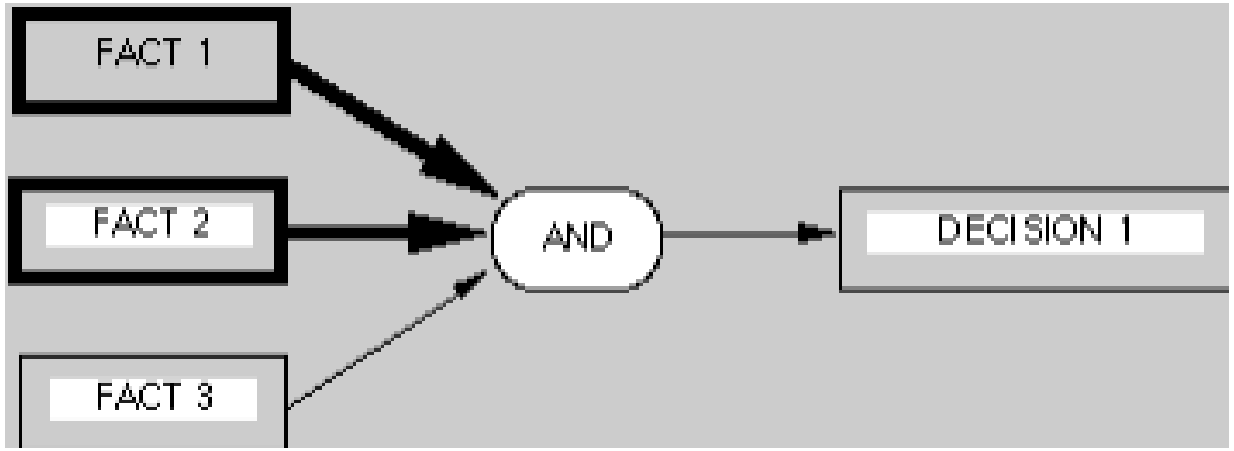
Answering HOW?

- To answer how a conclusion was reached, work back through the inference chain.
- Decision 4 was made as a result of making decision 1 and decision 2.
- Decision 1 was made because facts 1, 2 & 3 are true, etc.



Answering WHY?

- To answer why a computation is being performed, the system must state its current goal.
- The system may ask the user if fact 3 is true because it is trying to determine if decision 1 should be made.



6.4 Perception and Action

- Perception involves interpreting sights, sounds, smells, and touch.

- Perception involves interpreting sights, sounds, smells, and touch.
- Action includes the ability to navigate through the world and manipulate objects.

- Perception involves interpreting sights, sounds, smells, and touch.
- Action includes the ability to navigate through the world and manipulate objects.
- In basic perceptual and motor skills, even lower animals possess phenomenal capabilities compared to computers. In order to build robots that live in the world, we must come to understand these processes.

Key Difference between AI Programs and Robots

- While AI programs usually operate in computer-simulated worlds, robots must operate in the physical world.

Key Difference between AI Programs and Robots

- While AI programs usually operate in computer-simulated worlds, robots must operate in the physical world.
- Example: Making a move in chess

Key Difference between AI Programs and Robots

- While AI programs usually operate in computer-simulated worlds, robots must operate in the physical world.
- Example: Making a move in chess
- An AI program can search millions of nodes in a game tree without ever having to sense or touch anything in the real world.

Key Difference between AI Programs and Robots

- While AI programs usually operate in computer-simulated worlds, robots must operate in the physical world.
- Example: Making a move in chess
- An AI program can search millions of nodes in a game tree without ever having to sense or touch anything in the real world.
- A complete chess-playing robot, on the other hand, must be capable of grasping pieces, visually interpreting board positions, and carrying on a host of other actions.

Implications of the Distinction between Real and Simulated Worlds

- Input to an AI program is symbolic in form, e.g., an 8-puzzle configuration or a typed English sentence.
- AI programs require only general-purpose computers.

- Input to a robot is typically an analog signal, such as a 2-D image or a speech waveform.
- Robots need special hardware for perceiving and affecting the world.
- Robot sensors are inaccurate, and their effectors are limited in precision. There is always some degree of uncertainty about exactly where the robot is located and where objects and obstacles stand in relation to it. Robot effectors are also limited in precision.
- Many robots must react in real time. A robot fighter plane, for example, cannot afford to stop monitoring the world during a LISP garbage collection.
- Real world is unpredictable, dynamic, and uncertain. A robot cannot hope to maintain a correct and complete description of the world.

Perception

- Humans perceive through: sight, sound, touch, smell, taste.
- Robots, too, can process visual and auditory information, and they can also be equipped with more exotic sensors, such as laser rangefinders, speedometers, and radar.
- Applications of machine vision include mobile robot navigation, complex manufacturing tasks, analysis of satellite images, medical image processing etc.

Action

- Navigation means moving around the world, planning routes, reaching desired destinations without bumping into things etc.
- Manipulation: Robot manipulators are able to perform simple repetitive tasks such as bolting and fitting automobile parts, but these robots are highly task-specific.