

## Chapter 6

### Machine Learning

#### What is Learning?

"Learning denotes changes in the systems that are adaptive in the sense that they enable the system to do the same task (or tasks drawn from the same population) more effectively the next time." --Herbert Simon

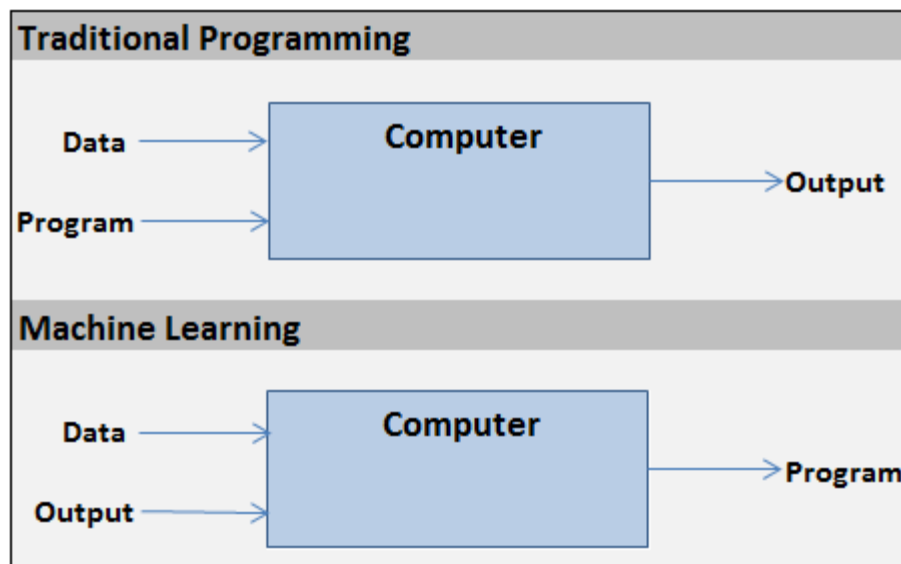
"Learning is constructing or modifying representations of what is being experienced." --Ryszard Michalski

"Learning is making useful changes in our minds." --Marvin Minsky

#### Machine Learning Definitions:

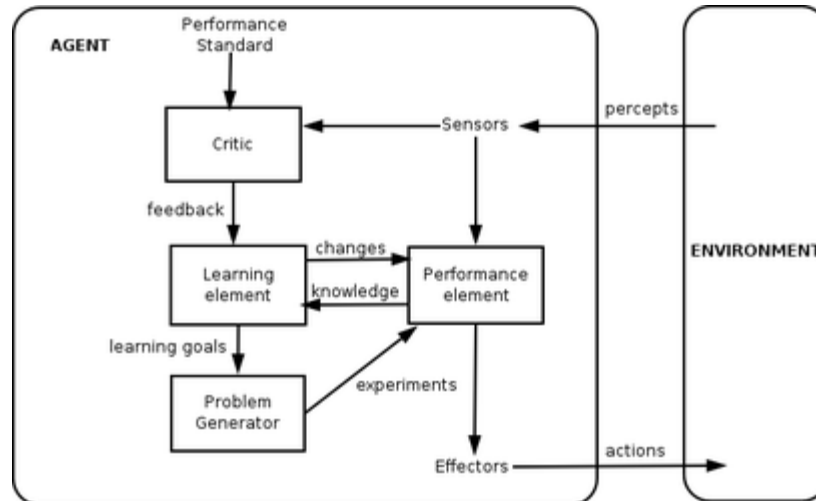
**According to Tom Mitchell** "A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$  if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ".

The difference between traditional programming and machine learning can be illustrated with following figures (source: [www.datasciencecentral.com](http://www.datasciencecentral.com))



#### General Machine Learning agent Framework or architecture:

The following figure represents the general steps involved in machine learning agent. It learns from inputs and corrects its decision based on the feedback provided by critic.



It consists of the following components.

1. Learning element
2. Knowledge base
3. Performance element
4. Feedback element (Critic)
5. Standard system.

### 1. Learning element

It receives and processes the input obtained from a person ( i.e. a teacher), from reference material like magazines, journals, etc, or from the environment at large.

### 2. Knowledge base

This is somewhat similar to the database. Initially it may contain some basic knowledge. Thereafter it receives more knowledge which may be new and so be added as it is or it may replace the existing knowledge.

### 3. Performance element

It uses the updated knowledge base to perform some tasks or solves some problems and produces the corresponding output.

### 4. Feedback element

It is receiving the two inputs, one from learning element and one from standard (or idealized) system. This is to identify the differences between the two inputs. The feedback is used to determine what should be done in order to produce the correct output.

### 5. Standard system

It is a trained person or a computer program that is able to produce the correct output. In order to check whether the machine learning system has learned well, the same input is given to the standard system. The outputs of standard system and that of performance element are given as inputs to the feedback element for the comparison. Standard system is also called idealized system.

### Types of Learning:

The strategies for learning can be classified according to the amount of inference the system has to perform on its training data. In increasing order we have

1. **Rote learning** – the new knowledge is implanted directly with no inference at all, e.g. simple memorization of past events, or a knowledge engineer's direct programming of rules elicited from a human expert into an expert system.

Rote learning is the basic learning activity. It is also called memorization because the knowledge, without any modification is, simply copied into the knowledge base. As computed values are stored, this technique can save a significant amount of time.

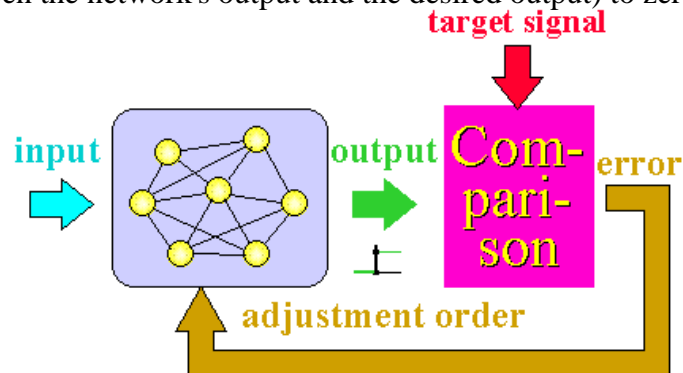
**Learning by Memorization** which avoids understanding the inner complexities the subject that is being learned; Rote learning instead focuses on memorizing the material so that it can be recalled by the learner exactly the way it was read or heard.

**Learning something by Repeating** over and over and over again; saying the same thing and trying to remember how to say it; it does not help us to understand; it helps us to remember, like we learn a poem, or a song, or something like that by rote learning.

2. **Supervised learning** – the system is supplied with a set of training examples consisting of inputs and corresponding outputs, and is required to discover the relation or mapping between them, e.g. as a series of rules, or a neural network.

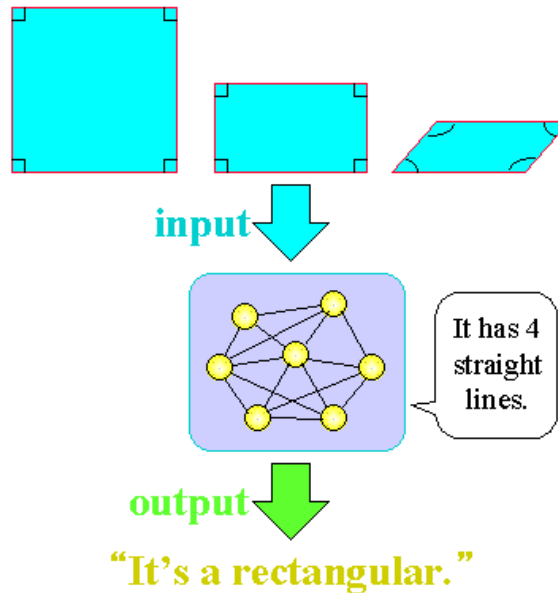
In supervised learning, the network is presented with inputs together with the target (teacher signal) outputs. Then, the neural network tries to produce an output as close as possible to the target signal by adjusting the values of internal weights. The most common supervised learning method is the “error correction method”.

Error correction method is used for networks which their neurons have discrete output functions. Neural networks are trained with this method in order to reduce the error (difference between the network's output and the desired output) to zero.



3. **Unsupervised learning** –

In unsupervised learning, there is no teacher (target signal) from outside and the network adjusts its weights in response to only the input patterns. A typical example of unsupervised learning is **Hebbian learning**.



Consider a machine (or living organism) which receives some sequence of inputs  $x_1, x_2, x_3, \dots$ , where  $x_t$  is the sensory input at time  $t$ . In supervised learning the machine is given a sequence of input & a sequence of desired outputs  $y_1, y_2, \dots$ , and the goal of the machine is to learn to produce the correct output given a new input. While, in unsupervised learning the machine simply receives inputs  $x_1, x_2, \dots$ , but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.

### **The need for Learning:**

As with many other types of AI system, it is much more efficient to give the system enough knowledge to get it started, and then leave it to learn the rest for itself. We may even end up with a system that learns to be better than a human expert.

The *general learning approach* is to generate potential improvements, test them, and discard those which do not work. Naturally, there are many ways we might generate the potential improvements, and many ways we can test their usefulness. At one extreme, there are model driven (top-down) generators of potential improvements, guided by an understanding of how the problem domain works. At the other, there are data driven (bottom-up) generators, guided by patterns in some set of training data.

## Learning through Examples: (A type of Concept learning)

(References: P. Winston, "Learning by Managing Multiple Models")

Concept learning also refers to a learning task in which a human or machine learner is trained to classify objects by being shown a set of example objects along with their class labels. The learner will simplify what has been observed in an example. This simplified version of what has been learned will then be applied to future examples. Concept learning ranges in simplicity and complexity because learning takes place over many areas. When a concept is more difficult, it will be less likely that the learner will be able to simplify, and therefore they will be less likely to learn.

This learning by example consists of the idea of **version space**.

A **version space** is a hierarchical representation of knowledge that enables you to keep track of all the useful information supplied by a sequence of learning examples without remembering any of the examples.

The **version space method** is a concept learning process accomplished by managing multiple models within a version space.

### Version Space Characteristics

A version space represents all the alternative plausible **descriptions** of a heuristic. A plausible description is one that is applicable to all known positive examples and no known negative example.

A version space description consists of two complementary trees:

1. One that contains nodes connected to overly **general** models, and
2. One that contains nodes connected to overly **specific** models.

Node values/attributes are **discrete**.

### Fundamental Assumptions

1. The data is correct; there are no erroneous instances.
2. A correct description is a conjunction of some of the attributes with values.

### Diagrammatical Guidelines

There is a **generalization** tree and a **specialization** tree.

Each **node** is connected to a **model**.

Nodes in the generalization tree are connected to a model that matches everything in its subtree.

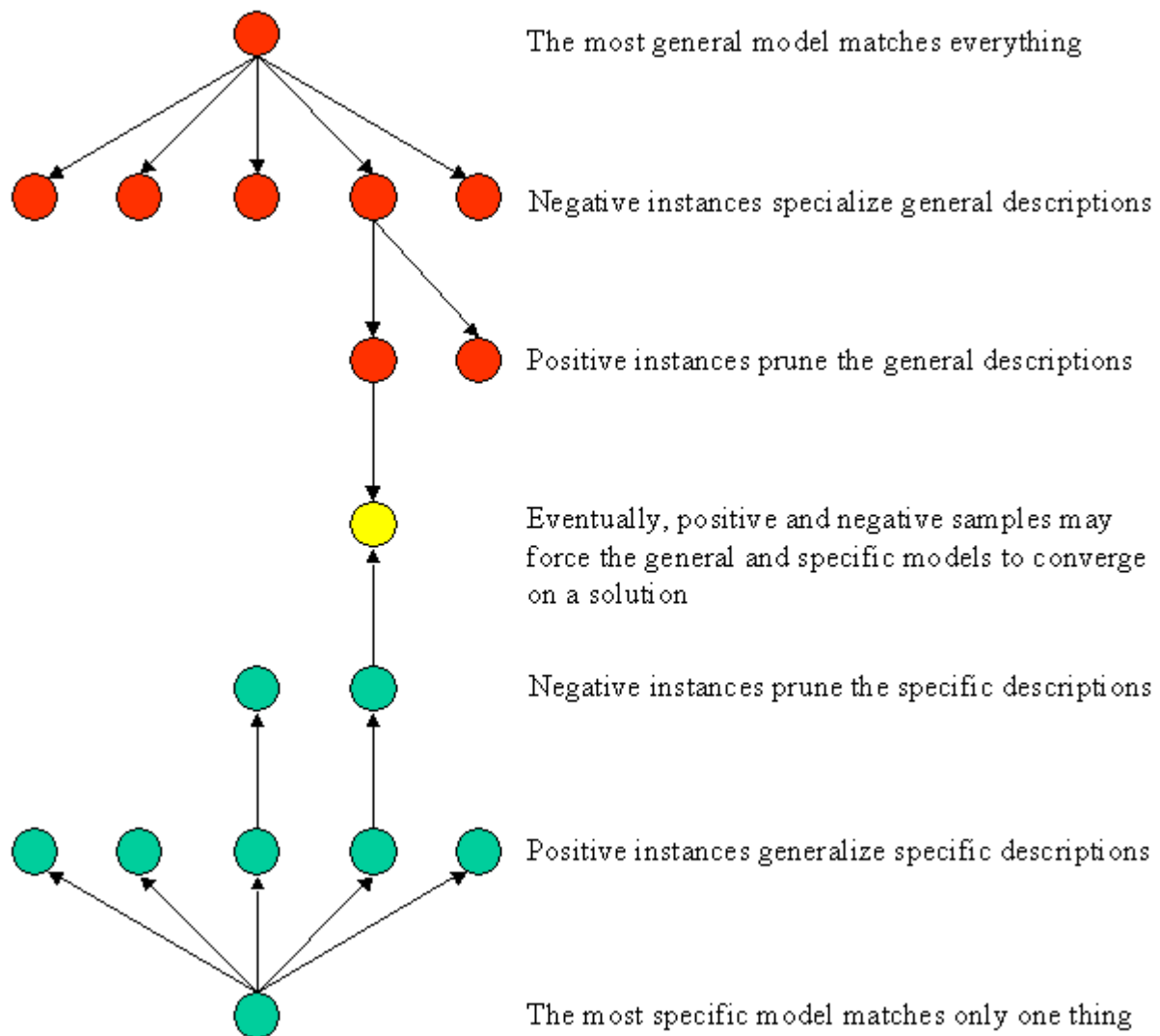
Nodes in the specialization tree are connected to a model that matches only one thing in its subtree.

Links between nodes and their models denote

- generalization relations in a generalization tree, and
- specialization relations in a specialization tree.

### Diagram of a Version Space

In the diagram below, the specialization tree is colored **red**, and the generalization tree is colored **green**.



## **Generalization and Specialization Leads to Version Space Convergence**

The key idea in version space learning is that specialization of the general models and generalization of the specific models may ultimately lead to just one correct model that matches all observed positive examples and does not match any negative examples.

That is, each time a negative example is used to specialize the general models, those specific models that match the negative example are eliminated and each time a positive example is used to generalize the specific models, those general models that fail to match the positive example are eliminated. Eventually, the positive and negative examples may be such that only one general model and one identical specific model survive.

## **Version Space Method Learning Algorithm: Candidate-Elimination**

The version space method handles positive and negative examples symmetrically.

### **Given:**

- A representation language.
- A set of positive and negative examples expressed in that language.

**Compute:** a concept description that is consistent with all the positive examples and none of the negative examples.

### **Method:**

- Initialize G, the set of maximally general hypotheses, to contain one element: the null description (all features are variables).
- Initialize S, the set of maximally specific hypotheses, to contain one element: the first positive example.
- Accept a new training example.
  - If the example is **positive**:
    1. Generalize all the specific models to match the positive example, but ensure the following:
      - The new specific models involve minimal changes.
      - Each new specific model is a specialization of some general model.
      - No new specific model is a generalization of some other specific model.
    2. Prune away all the general models that fail to match the positive example.
  - If the example is **negative**:
    1. Specialize all general models to prevent match with the negative example, but ensure the following:
      - The new general models involve minimal changes.

- Each new general model is a generalization of some specific model.
  - No new general model is a specialization of some other general model.
2. Prune away all the specific models that match the negative example.
    - If S and G are both singleton sets, then:
      - if they are identical, output their value and halt.
      - if they are different, the training cases were inconsistent. Output this result and halt.
      - else continue accepting new training examples.

The algorithm stops when:

1. It runs out of data.
2. The number of hypotheses remaining is:
  - 0 - no consistent description for the data in the language.
  - 1 - answer (version space converges).
  - $2^+$  - all descriptions in the language are implicitly included.

Example: Learning the concept of "Japanese Economy Car"


**Features:** (Country of Origin, Manufacturer, Color, Decade, Type)


Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive

### Solution:

1. **Positive Example:** (Japan, Honda, Blue, 1980, Economy)

Initialize G to a singleton set that includes everything. Initialize S to a singleton set that includes the first positive example.	$G = \{ (?, ?, ?, ?, ?) \}$ $S = \{ (Japan, Honda, Blue, 1980, Economy) \}$
----------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------

  $(?, ?, ?, ?, ?)$

$(Japan, Honda, Blue, 1980, Economy)$  

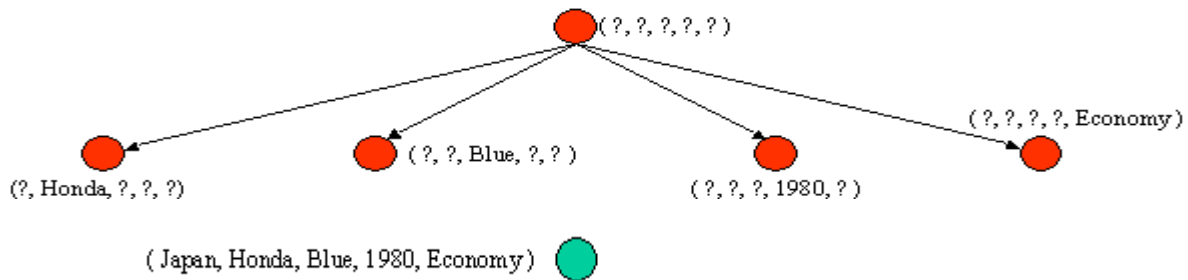
These models represent the most general and the most specific heuristics one might learn. The actual heuristic to be learned, "Japanese Economy Car", probably lies between them somewhere within the version space.



2. **Negative Example:** (Japan, Toyota, Green, 1970, Sports)

Specialize G to exclude the negative example.

G =	{ (?, Honda, ?, ?, ?), (?, ?, Blue, ?, ?), (?, ?, ?, 1980, ?), (?, ?, ?, ?, Economy) }
S =	{ (Japan, Honda, Blue, 1980, Economy) }



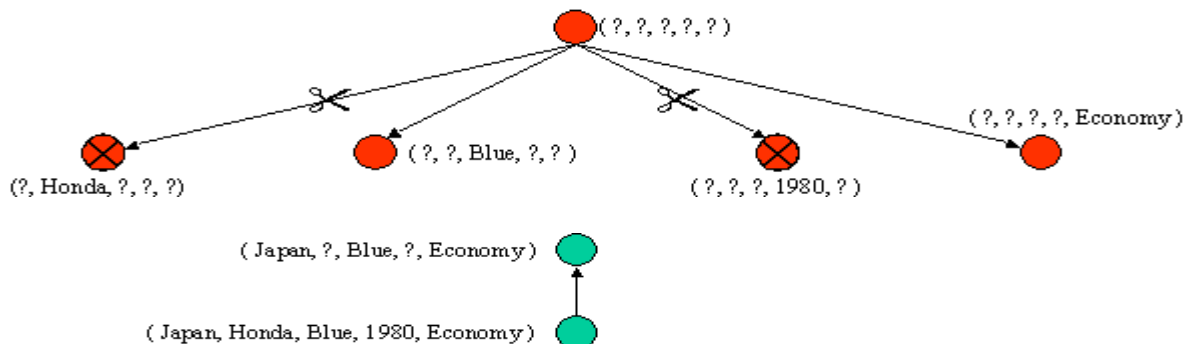
Refinement occurs by generalizing S or specializing G, until the heuristic hopefully converges to one that works well.

3. **Positive Example:** (Japan, Toyota, Blue, 1990, Economy)

Prune G to exclude descriptions inconsistent with the positive example. (Prune = ✂)

Generalize S to include the positive example.

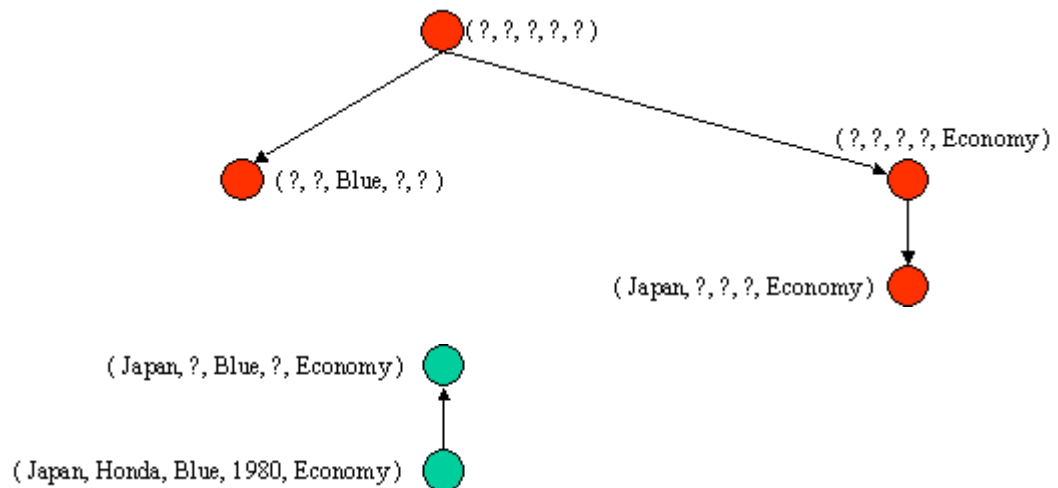
G =	{ (?, ?, Blue, ?, ?), (?, ?, ?, ?, Economy) }
S =	{ (Japan, ?, Blue, ?, Economy) }



4. **Negative Example:** (USA, Chrysler, Red, 1980, Economy)

Specialize G to exclude the negative example (but stay consistent with S)

G =	{ (?, ?, Blue, ?, ?), (Japan, ?, ?, ?, Economy) }
S =	{ (Japan, ?, Blue, ?, Economy) }



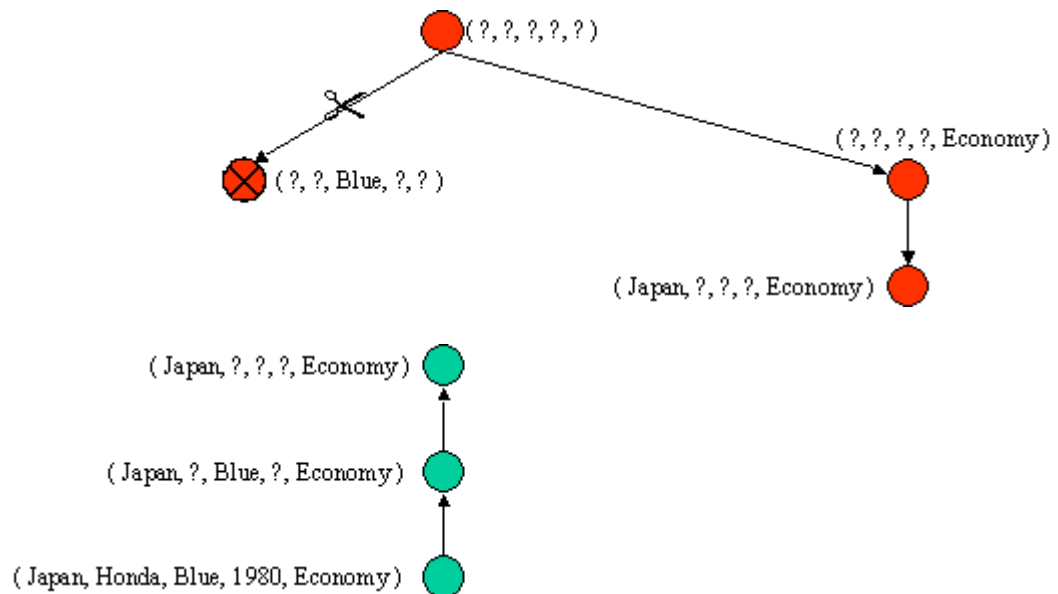
5. **Positive Example:** (Japan, Honda, White, 1980, Economy)

Prune G to exclude descriptions inconsistent with positive example.

Generalize S to include positive example.

$G = \{ (Japan, ?, ?, ?, Economy) \}$

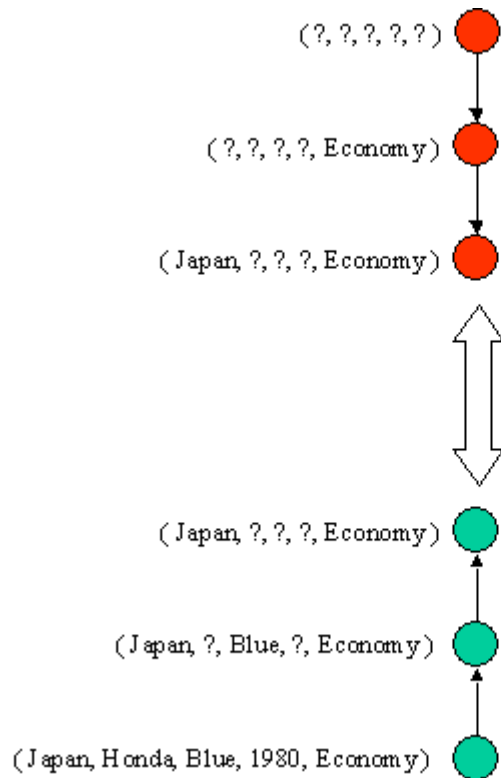
$S = \{ (Japan, ?, ?, ?, Economy) \}$



G and S are singleton sets and  $S = G$ .

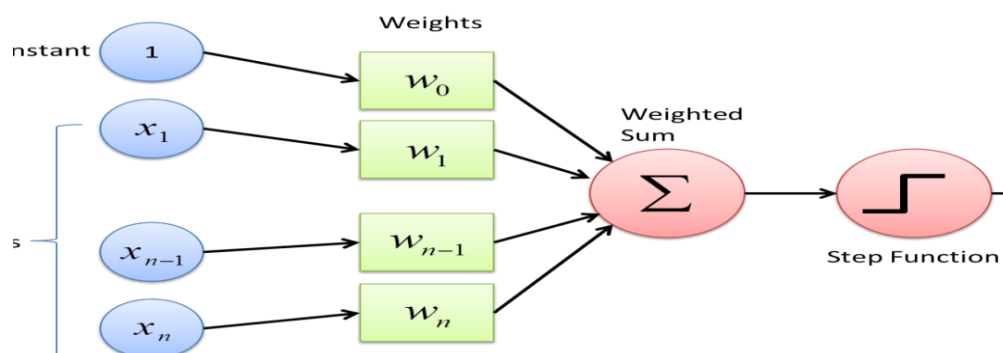
Converged.

No more data, so algorithm stops.



## Learning by Perceptrons Algorithm

It was originally developed by Frank Rosenblatt in the late 1950s. Training patterns are presented to the network's inputs; the output is computed. Then the connection weights  $w_j$  are modified by an amount that is proportional to the product of the difference between the actual output,  $y$ , and the desired output,  $d$ , and the input pattern,  $x$ . The summarized algorithm is given below:



### Variables used:

- $y=f(x)$  denotes the *output* from the perceptron for an input vector  $x$ .
- $D=(x_1,d_1), (x_2,d_2), \dots, (x_n, d_n)$  is the *training set* of  $n$  samples, where:
  - $x_j$  is the  $m$ -dimensional input vector.
  - $d_j$  is the desired output value of the perceptron for that input  $x_j$ .

We show the values of the features as follows:

- $x_{j,i}$  is the value of the  $i^{\text{th}}$  feature of the  $j^{\text{th}}$  training *input vector*.

To represent the weights:

- $w_i$  is the  $i^{\text{th}}$  value in the *weight vector*, to be multiplied by the value of the  $i^{\text{th}}$  input feature.
- $w_i(t)$  is the weight  $i$  at  $t$  time .

### Steps in Algorithms

1. Initialize weights and threshold.
2. For each example  $j$  in our training set  $D$ , perform the following steps over the input  $x_j$  and desired output  $d_j$ :
  - Calculate the actual output:
    - $y(t) = g [w_0(t)x_0(t) + w_1(t)x_1(t) + \dots + w_n(t)x_n(t)]$
  - Update the weights
    - $w_i(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t)$  , where  $0 \leq \alpha \leq 1$  (learning rate) is a positive gain function that controls the adaption rate.

*Examples: Perceptron Learning algorithm (implement a neuron to simulate logical OR)*

## Explanation Based Machine Learning:

**Explanation-based learning (EBL)** is a form of machine learning that exploits a very strong, or even perfect, domain theory to make generalizations or form concepts from training examples. This is a type of *analytic* learning. The advantage of explanation-based learning is that, as a deductive mechanism, it requires only a single training example ( inductive learning methods often require many training examples)

An Explanation-based Learning (**EBL** ) system accepts an example (i.e. a training example) and explains what it learns from the example. The **EBL** system takes only the relevant aspects of the training.

EBL accepts four inputs:

**A training example:** what the learning *sees* in the world. (specific facts that rule out some possible hypotheses)

**A goal concept:** a high level description of what the program is supposed to learn. (the set of all possible conclusions)

**A operational criterion:** a description of which concepts are usable. (criteria for determining which features in the domain are efficiently recognizable, e.g. which features are directly detectable using sensors).

**A domain theory:** a set of rules that describe relationships between objects and actions in a domain. (axioms about a domain of interest)

From this EBL computes a generalization of the training example that is sufficient not only to describe the goal concept but also satisfies the operational criterion.

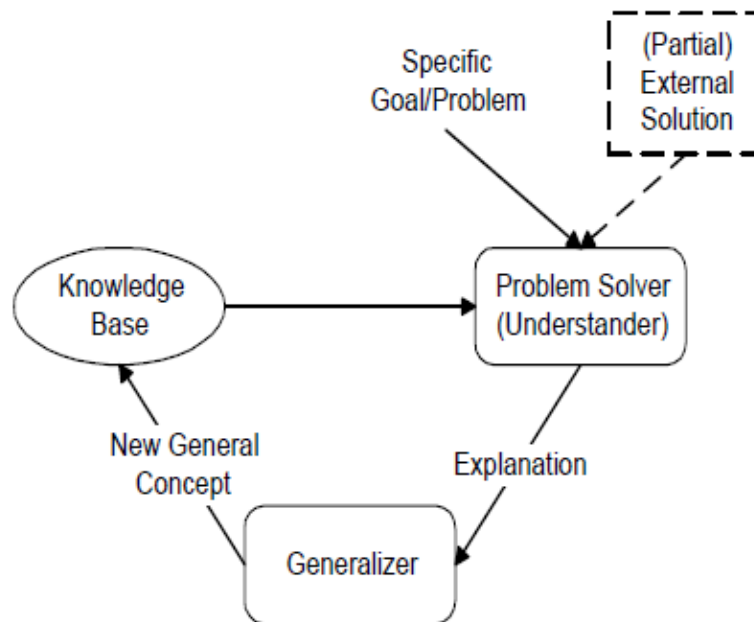
This has two steps:

**Explanation:** the domain theory is used to prune away all unimportant aspects of the training example with respect to the goal concept.

**Generalisation:** the explanation is generalized as far possible while still describing the goal concept.

An example of EBL using a perfect domain theory is a program that learns to play chess by being shown examples. A specific chess position that contains an important feature, say, "Forced loss of black queen in two moves," includes many irrelevant features, such as the specific scattering of pawns on the board. EBL can take a single training example and determine what the relevant features are in order to form a generalization.

The general architecture of EBL system is;



### **Learning By analogy (Reference: Lecture notes By Jagadish Bhatt)**

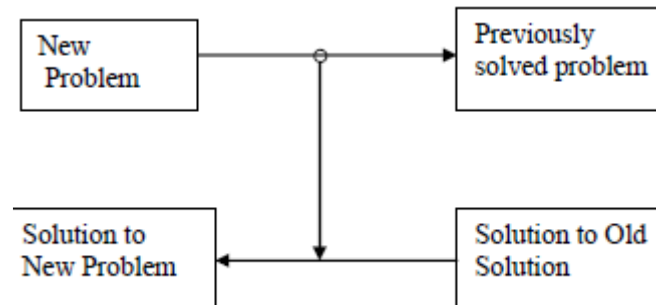
Reasoning by analogy generally involves abstracting details from a particular set of problems and resolving structural similarities between previously distinct problems.

Analogical reasoning refers to this process of recognition and then applying the solution from the known problem to the new problem. Such a technique is often identified as *case based reasoning*.

Analogical learning generally involves developing a set of mappings between features of two instances.

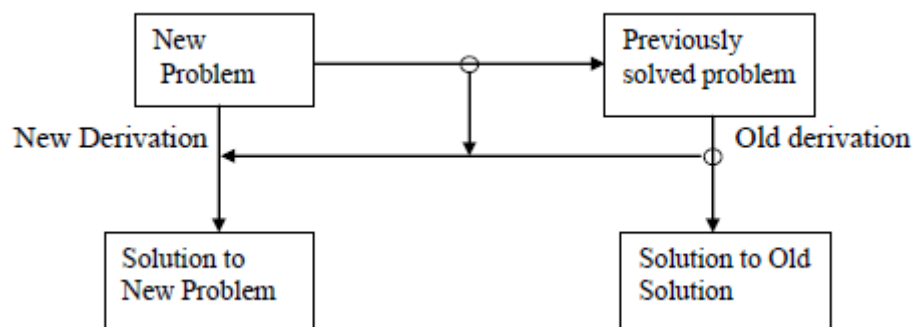
**Transformational Analogy:**

Suppose you are asked to prove a theorem in plane geometry. You might look for a previous theorem that is very similar and copy its proof, making substitutions when necessary. The idea is to transform a solution to a previous problem in to solution for the current problem. The following figure shows this process.



### Derivational Analogy:

Notice that transformational analogy does not look at how the old problem was solved, it only looks at the final solution. Often the twists and turns involved in solving an old problem are relevant to solving a new problem. The detailed history of problem solving episode is called derivation, Analogical reasoning that takes these histories into account is called derivational analogy.



### Learning by simulating Evolutions

Evolution is change in the heritable characteristics of biological populations over successive generations. Evolutionary processes give rise to biodiversity at every level of biological organization, including the levels of species, individual organisms, and molecules.

The Darwin theory of Evolution explain natural selection as

- Individuals pass on traits to offspring
- Individuals have different traits
- Fittest individuals survive to produce more offspring
- Over time, variation can accumulate leading to new species

A genetic or evolutionary algorithm **applies the principles of evolution found in nature** to the problem of finding an optimal solution to a Solver problem. In a "genetic algorithm," the problem is encoded in a series of bit strings that are manipulated by the algorithm;

Five phases are considered in a genetic algorithm as shown in figures.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

## **Initial Population**

The process begins with a set of individuals which is called a **Population**. Each individual is a solution to the problem you want to solve. An individual is characterized by a set of parameters (variables) known as **Genes**. Genes are joined into a string to form a **Chromosome** (solution)

## **Fitness Function**

The **fitness function** determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a **fitness score** to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

## **Selection**

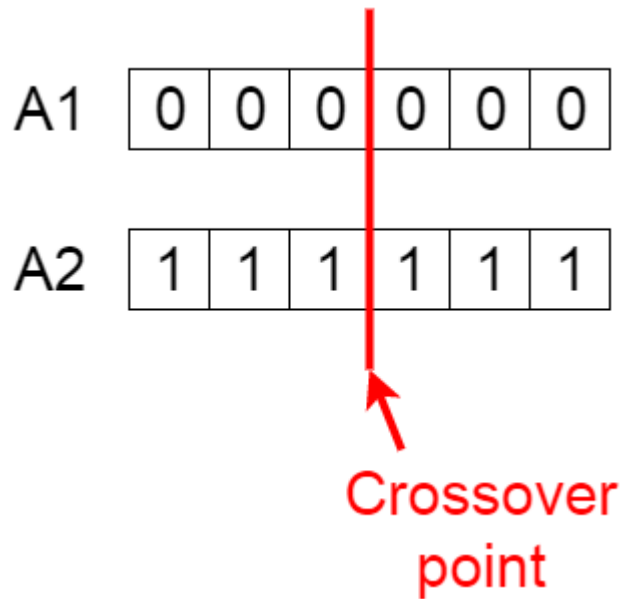
The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation.

Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

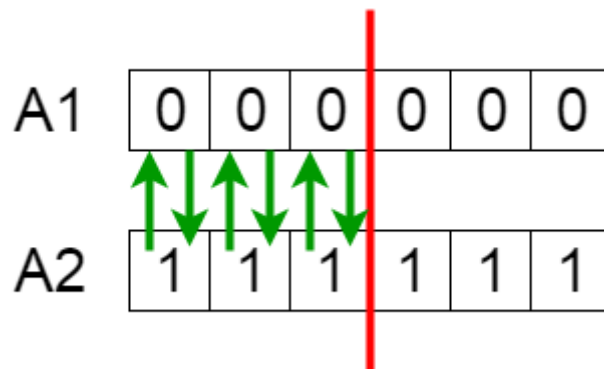
## **Crossover**

**Crossover** is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a **crossover point** is chosen at random from within the genes.

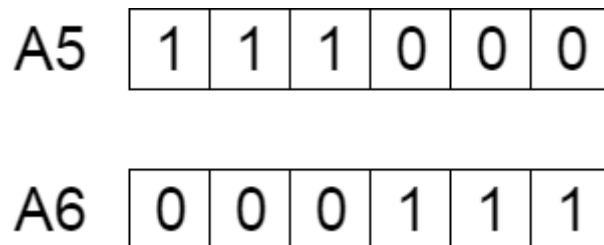
For example, consider the crossover point to be 3 as shown below.



**Offspring** are created by exchanging the genes of parents among themselves until the crossover point is reached.



The new offspring are added to the population.



## Mutation

In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.



**Before Mutation**

A5 

1	1	1	0	0	0
---	---	---	---	---	---

**After Mutation**

A5 

1	1	0	1	1	0
---	---	---	---	---	---

## **Termination**

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.