

What is Learning?

“Learning denotes changes in the system 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

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.
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.
3. **Unsupervised learning** – the system is supplied with a set of training examples consisting only of inputs and is required to discover for itself what appropriate outputs should be, e.g. a *Kohonen Network* or *Self Organizing Map*.

Early expert systems relied on rote learning, but for modern AI systems we are generally interested in the supervised learning of various levels of rules.

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.

Machine Learning:

As regards machines, we might say, very broadly, that a machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves. Some of these changes, such as the addition of a record to a data base, fall comfortably within the province of other disciplines and are not necessarily better understood for being called learning. But, for example, when the performance of a speech-recognition machine improves after hearing several samples of a person's speech, we feel quite justified in that case saying that the machine has learned.

Machine learning usually refers to the changes in systems that perform tasks associated with artificial intelligence (AI). Such tasks involve recognition, diagnosis, planning, robot control, prediction, etc. The changes might be either enhancements to already performing systems or synthesis of new systems.

Supervised Learning

In supervised learning, you use input/output pairs (labeled data) to train the machine. You show the algorithm both input variables (x) and an output variable (Y) and then have the algorithm infer the mapping function from the input to the output.

The main goal here is getting the machine to produce a function that's approximated enough to be able to predict outputs for new inputs when you introduce them.

Supervised learning problems can be further grouped into regression problems and classification problems.

A regression problem is when outputs are actual objects whereas a classification problem, as the name suggests, is when outputs are categories.

Gradient Descent Learning

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

Minimizing any function means finding the deepest valley in that function. Keep in mind that, the cost function is used to monitor the error in predictions of an ML model. So minimizing this, basically means getting to the lowest error value possible or increasing the accuracy of the model. Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

Intuition for Gradient Descent

Think of a large bowl like what you would eat cereal out of or store fruit in. This bowl is a plot of the cost function (f).



A random position on the surface of the bowl is the cost of the current values of the coefficients (cost).

The bottom of the bowl is the cost of the best set of coefficients, the minimum of the function.

The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost.

Repeating this process enough times will lead to the bottom of the bowl and you will know the values of the coefficients that result in the minimum cost.

Gradient Descent Procedure

The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.

$$\text{coefficient} = 0.0$$

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

$$\text{cost} = f(\text{coefficient})$$

or

$$\text{cost} = \text{evaluate}(f(\text{coefficient}))$$

The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

$$\text{delta} = \text{derivative}(\text{cost})$$

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.

$$\text{coefficient} = \text{coefficient} - (\text{alpha} * \text{delta})$$

This process is repeated until the cost of the coefficients (cost) is 0.0 or close enough to zero to be good enough.

You can see how simple gradient descent is. It does require you to know the gradient of your cost function or the function you are optimizing, but besides that, it's very straightforward. Next we will see how we can use this in machine learning algorithms.

Least Mean Square

The LMS algorithm was introduced by Widrow and Hoff in 1959. It has several names, including the Widrow-Hoff rule and also Delta rule. LMS is an example of supervised learning algorithm in NN similar with the perceptron learning algorithm (refer to the previous article, May 2011). In the perceptron learning algorithm, the algorithm trains the perceptron until it correctly classifies the output of the training set but LMS uses another termination criterion in order to train the perceptron. So instead of training the perceptron until a solution is found, another criterion is to continue training while the Mean-Square Error (MSE) is greater than a certain value. This is the basis for the LMS algorithm.

LMS is a fast algorithm that minimizes the MSE. The MSE is the average of the weighted sum of the error for N training sample.

In order to train the perceptron by using LMS, we can iterate the test set, taking a set of inputs, computing the output and then using the error to adjust the weight. This process can be done either randomly by the test set, or for each test of the set in succession. The learning rule of LMS is given as:

$$\Delta w_{ij} = -\eta \nabla E(w_{ij}) = \eta (d_i - y_i) x_j$$

The learning rule adjusts the weight based on the error (R-C or expected output minus actual output). Once the error is calculated, the weights are adjusted by a small amount, η in the direction of the input, E . This has the effect of adjusting the weights to reduce the output error.

The implementation of LMS is very simple. Initially, the weights vector is initialized with small random weights. The main repetition then randomly selects a test, calculates the output of the neuron, and then calculates the error. Using the error, the formula of learning rule is applied to each weight in the vector. Then continues the repetition to check the MSE to see if it has reached an acceptable value, and if so, exit and emit the computed truth table for the neuron.

The least mean square algorithm uses a technique called “method of steepest descent” and continuously estimates results by updating filter weights. Through the principle of algorithm convergence, the least mean square algorithm provides particular learning curves useful in machine learning theory and implementation. Many of these ideas are part of dedicated work on refining machine learning models, matching inputs to outputs, making training and test processes more effective, and generally pursuing “convergence” where the iterative learning process resolves into a coherent (reasonable or logical) final result instead of getting off track.

The LMS algorithm for a p th order algorithm can be summarized as

Parameters: p = filter order

μ = step size

Initialisation: $\hat{\mathbf{h}}(0) = \text{zeros}(p)$

Computation: For $n = 0, 1, 2, \dots$

$$\mathbf{x}(n) = [x(n), x(n-1), \dots, x(n-p+1)]^T$$

$$e(n) = d(n) - \hat{\mathbf{h}}^H(n)\mathbf{x}(n)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu e^*(n)\mathbf{x}(n)$$

Back Propagation

The Delta rule is a gradient descent learning rule for updating the weights of the inputs to artificial neurons, making connections between inputs and outputs with layers of artificial neurons.

The Delta rule is also known as the Delta learning rule.

Back propagation is a supervised learning method, and is an implementation of the Delta rule. It requires a teacher that knows, or can calculate, the desired output for any given input. It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". Back propagation requires that the activation function used by the artificial neurons (or "nodes") is differentiable.

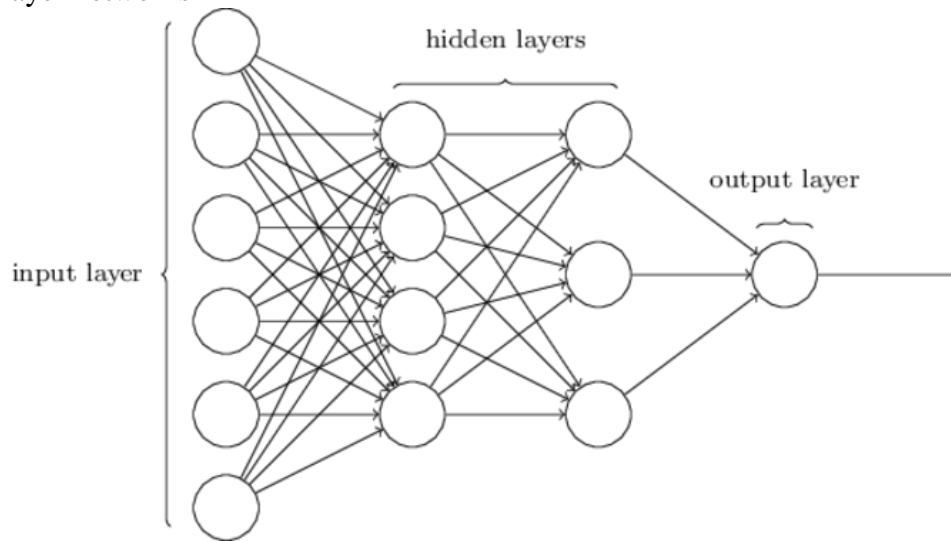
As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, back propagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights. This gradient is almost always then used in a simple stochastic gradient descent algorithm, is a general optimization algorithm, but is typically used to fit the parameters of a machine learning model, to find weights that minimize the error. Often the term "back propagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Back propagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Back propagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network.

Characteristics:

- A multi-layered perceptron has three distinctive characteristics
 - The network contains one or more layers of hidden neurons
 - The network exhibits a high degree of connectivity
 - Each neuron has a smooth (differentiable everywhere) nonlinear activation function, the most common is the sigmoidal nonlinearity.

The back propagation algorithm provides a computationally efficient method for training multi-layer networks



Summary of the backpropagation technique:

1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.
5. Assign "blame" for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat from step 3 on the neurons at the previous level, using each one's "blame" as its error.

Backpropagation Algorithm

1. Initialize weights to small random values
 2. Choose a random sample training item, say (\mathbf{x}^m, y^m)
 3. Compute total input z_j and output y_j for each unit (forward prop)
 4. Compute δ_n for output layer $\delta_n = y_n(1-y_n)(y_n-y_n^m)$
 5. Compute δ_j for all preceding layers by backprop rule
 6. Compute weight change by descent rule (repeat for all weights)
- Note that each expression involves data **local** to a particular unit, we don't have to look around summing things over the whole network.
 - It is for this reason, simplicity, locality and, therefore, efficiency that backpropagation has become the dominant paradigm for training neural nets.

Stochastic Gradient Descent for Machine Learning

Gradient descent can be slow to run on very large datasets.

Because one iteration of the gradient descent algorithm requires a prediction for each instance in the training dataset, it can take a long time when you have many millions of instances.

In situations when you have large amounts of data, you can use a variation of gradient descent called stochastic gradient descent.

In this variation, the gradient descent procedure described above is run but the update to the coefficients is performed for each training instance, rather than at the end of the batch of instances.

The first step of the procedure requires that the order of the training dataset is randomized. This is to mix up the order that updates are made to the coefficients. Because the coefficients are updated after every training instance, the updates will be noisy jumping all over the place, and so

will the corresponding cost function. By mixing up the order for the updates to the coefficients, it harnesses this random walk and avoids it getting distracted or stuck.

The update procedure for the coefficients is the same as that above, except the cost is not summed over all training patterns, but instead calculated for one training pattern.

The learning can be much faster with stochastic gradient descent for very large training datasets and often you only need a small number of passes through the dataset to reach a good or good enough set of coefficients, e.g. 1-to-10 passes through the dataset.

Stochastic gradient descent is a popular algorithm for training a wide range of models in machine learning, including (linear) support vector machines, logistic regression and graphical models. When combined with the backpropagation algorithm, it is the de facto standard algorithm for training artificial neural networks..

Unsupervised learning

Unsupervised learning is the training of an artificial intelligence (AI) algorithm using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.

In unsupervised learning, an AI system may group unsorted information according to similarities and differences even though there are no categories provided. AI systems capable of unsupervised learning are often associated with generative learning models, although they may also use a retrieval-based approach (which is most often associated with supervised learning). Chatbots, self-driving cars, facial recognition programs, expert systems and robots are among the systems that may use either supervised or unsupervised learning approaches.

In unsupervised learning, an AI system is presented with unlabeled, uncategorised data and the system's algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.

Unsupervised learning algorithms can perform more complex processing tasks than supervised learning systems. However, unsupervised learning can be more unpredictable than the alternate

model. While an unsupervised learning AI system might, for example, figure out on its own how to sort cats from dogs, it might also add unforeseen and undesired categories to deal with unusual breeds, creating clutter instead of order.

Hebbian Learning:

The oldest and most famous of all learning rules is Hebb's postulate of learning:

“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased”

From the point of view of artificial neurons and artificial neural networks, Hebb's principle can be described as a method of determining how to alter the weights between model neurons. The weight between two neurons increases if the two neurons activate simultaneously—and reduces if they activate separately. Nodes that tend to be either both positive or both negative at the same time have strong positive weights, while those that tend to be opposite have strong negative weights.

Hebb's Algorithm:

Step 0: initialize all weights to 0

Step 1: Given a training input, s , with its target output, t , set the activations of the input units: $x_i = s_i$

Step 2: Set the activation of the output unit to the target value: $y = t$

Step 3: Adjust the weights: $w_i(\text{new}) = w_i(\text{old}) + x_i y$

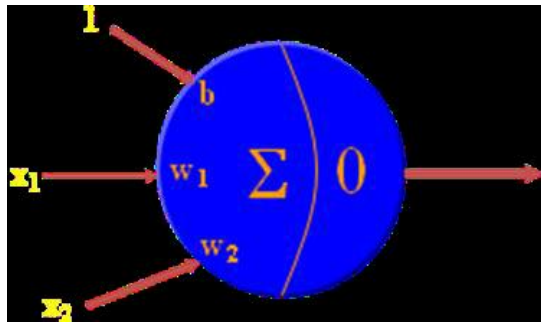
Step 4: Adjust the bias (just like the weights): $b(\text{new}) = b(\text{old}) + y$

Example:

PROBLEM: Construct a Hebb Net which performs like an AND function, that is, only when both features are “active” will the data be in the target class.

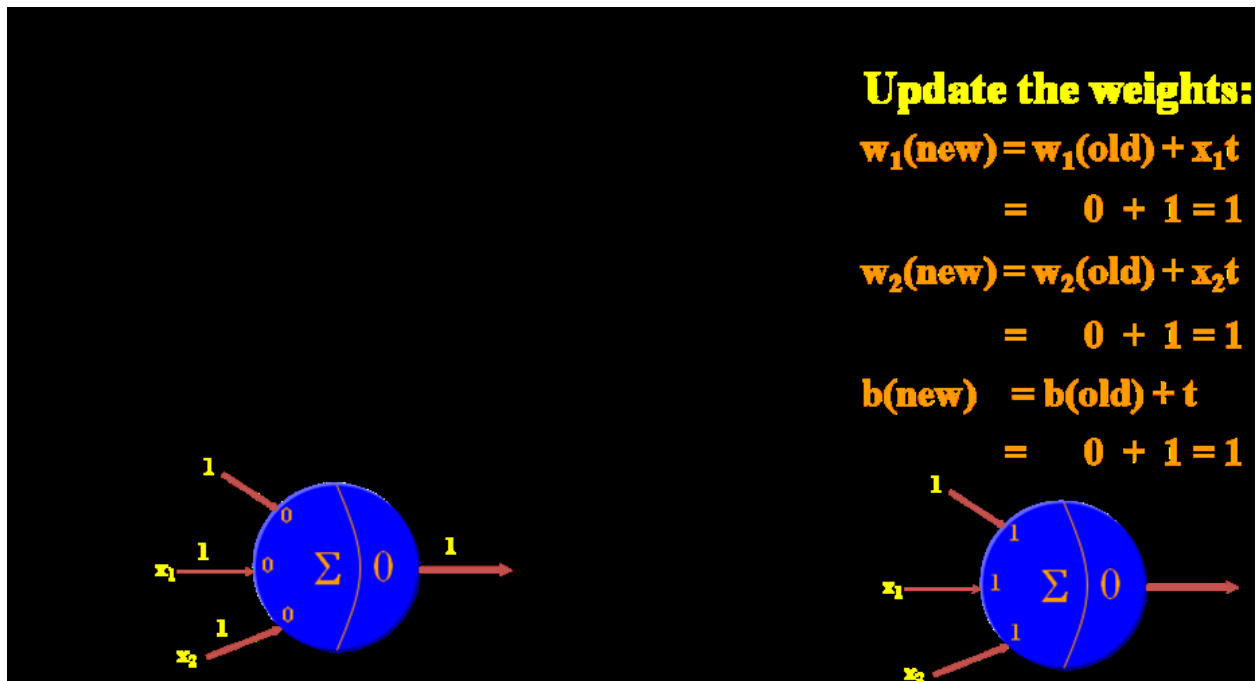
TRAINING SET (with the bias input always at 1):

X1	X2	Bias	target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



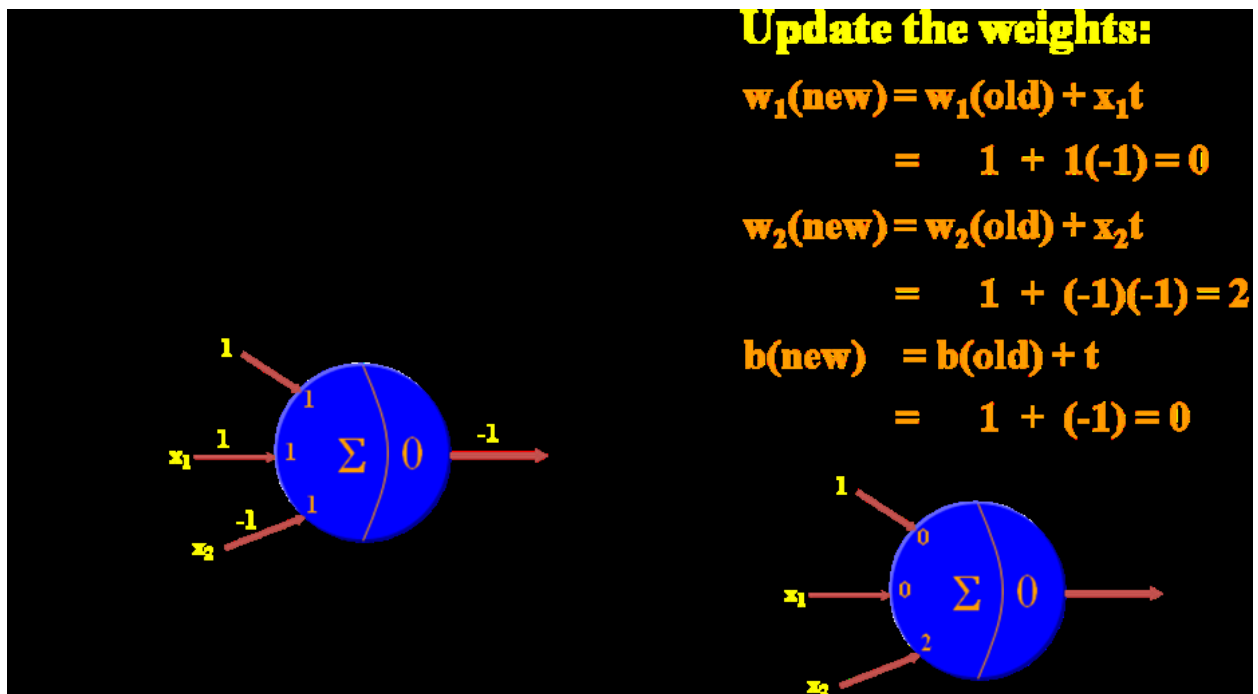
Training-First Input:

Present the first input (1 1 1) with the target of 1



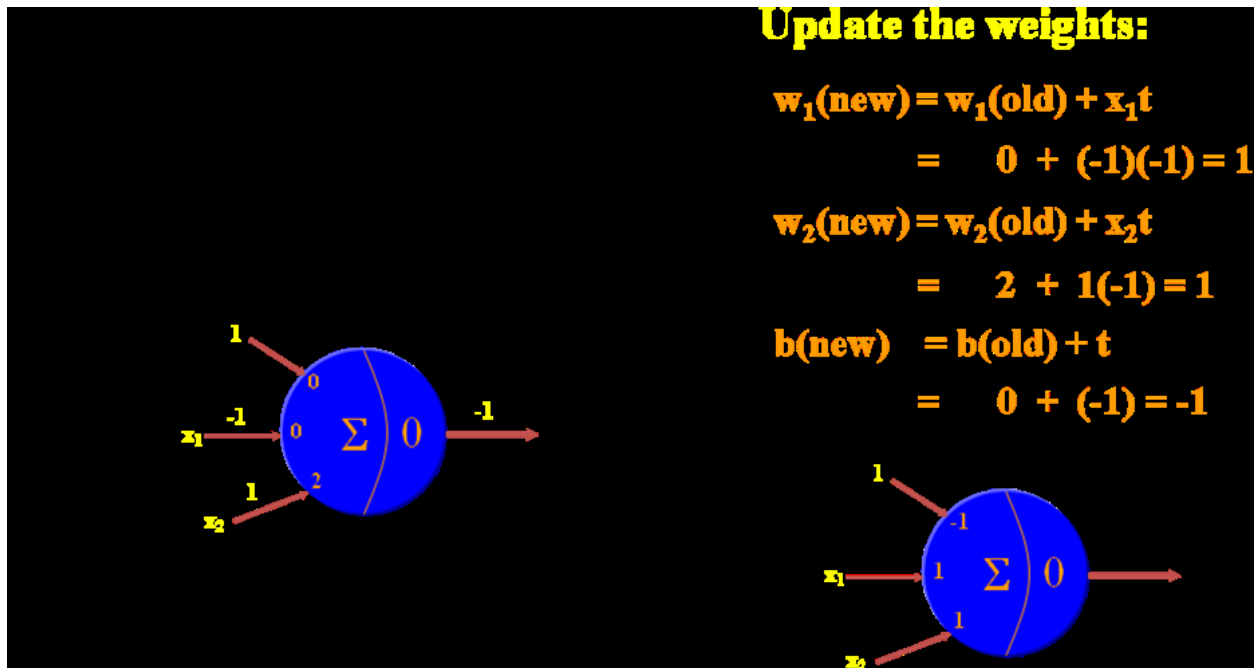
Training- Second Input:

Present the input (1 -1 1) with the target of -1



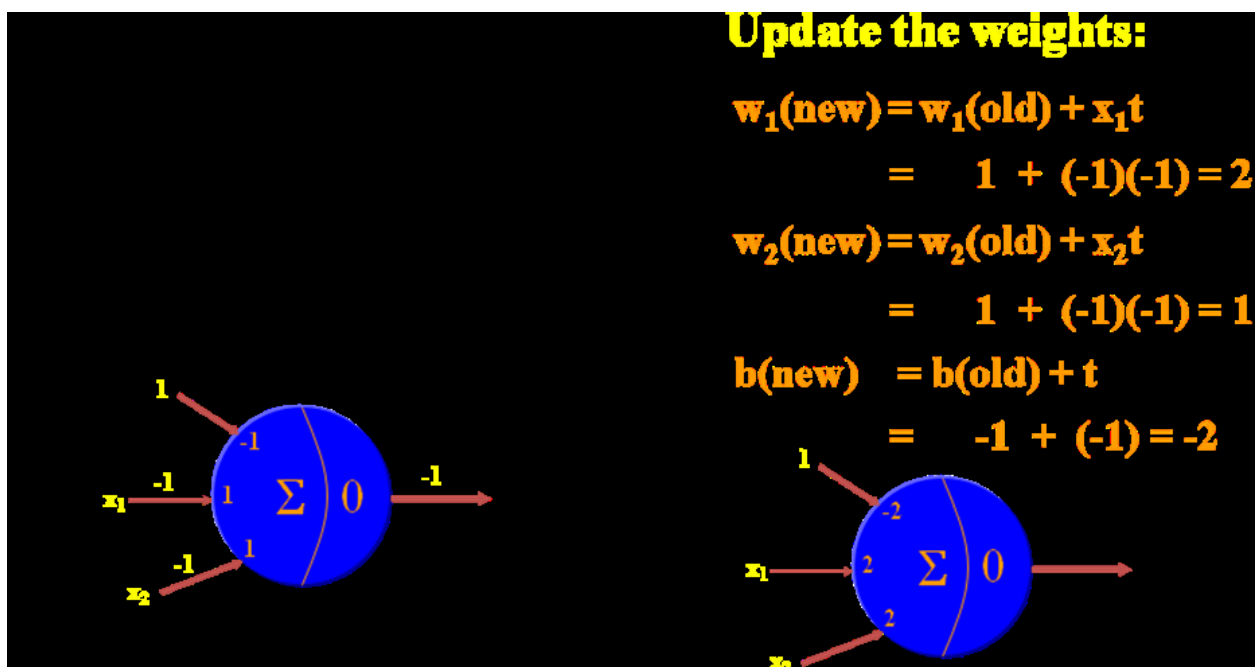
Training- Third Input:

Present the input (-1 1 1) with the target of -1



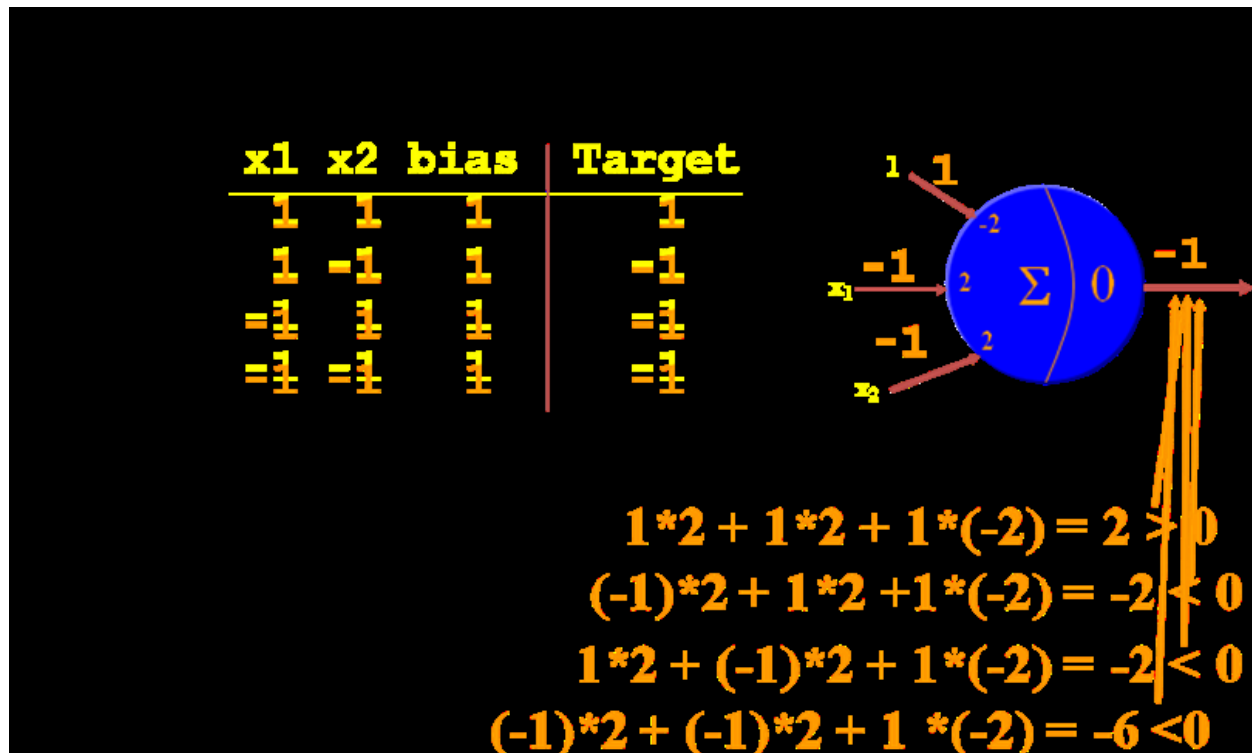
Training- Fourth Input:

Present the input (-1 -1 1) with the target of -1



Final Neuron:

This neuron works:



Competitive learning

Competitive learning is a form of unsupervised learning in artificial Neural Networks. The nodes compete for the right to respond to a subset of the input data. Competitive learning works by increasing the specialization of each node in the network. It is well suited to finding clusters within data.

Examples include: Vector quantization and Kohonen maps (self-organizing maps)

Principles of Competitive Learning:

There are three basic elements to a competitive learning rule:

- A set of neurons that are all the same, except for some randomly distributed synaptic weights, which respond differently to a given set of input patterns
- A limit which is imposed on the "strength" of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron (or only one neuron per group), is active (i.e. "on") at a time. The neuron that wins the competition is called a "winner-take-all" neuron.

Individual neurons of the network learn to specialize on ensembles of similar patterns and become 'feature detectors' for different classes of input patterns.

The competitive networks recode sets of correlated inputs to one of a few output neurons essentially removes the redundancy in representation.

Architecture & Implementation:

Competitive Learning is usually implemented with Neural Networks that contain a hidden layer which is commonly known as “competitive layer”.

Every competitive neuron is described by a vector of weights :

$\mathbf{w}_i = (w_{i1}, \dots, w_{id})^T, i = 1, \dots, M$ and calculates the similarity measure between the input data $\mathbf{x}^n = (x_{n1}, \dots, x_{nd})^T \in \mathbb{R}^d$ and the weight vector \mathbf{w}_i .

For every input vector, the competitive neurons “compete” with each other to see which one of them is the most similar to that particular input vector. The winner neuron m sets its output to:
 $O_m = 1$

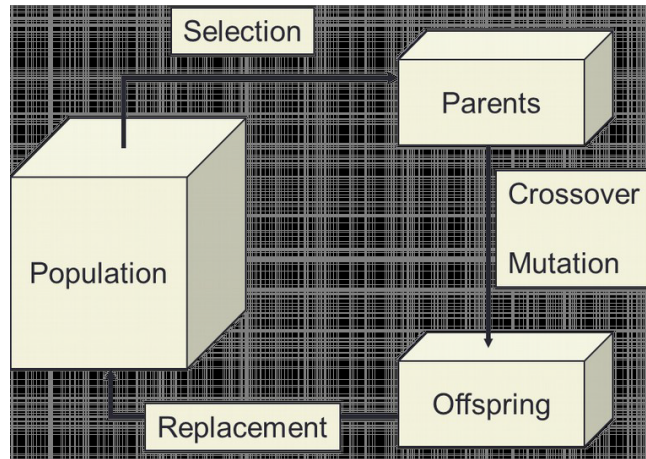
All the other competitive neurons set their output to:

$$o_i = 0, i = 1, \dots, M, i \neq m.$$


Usually, in order to measure similarity the inverse of the Euclidean distance is used.

Genetic Algorithm

- A genetic algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a set of stochastic operators.
- It is a variation of stochastic beam search.
- Inspired by biological evolution process
- Uses concepts of “Natural Selection” i.e. “Survival of the fittest” and “Genetic Inheritance”
- Particularly well suited for hard problems where little is known about the underlying search space
- Widely used in business, science and engineering.



Genetic process in Nature



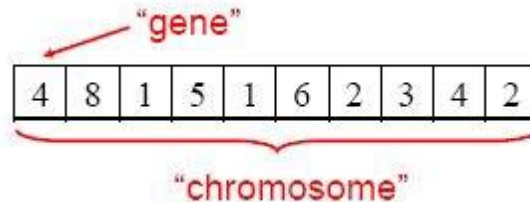
Basic Operators of Genetic Algorithm

- **Reproduction:** It is usually the first operator applied on population. Chromosomes are selected from the population of parents to cross over and produce offspring. It is based on Darwin's evolution theory of "Survival of the fittest". Therefore, this operator is also known as '**Selection Operator**'.
- **Cross Over:** After reproduction phase, population is enriched with better individuals. It makes clones of good strings but doesnot create new ones. Cross over operator is applied to the mating pool with a hope that it would create better strings.
- **Mutation:** After cross over, the strings are subjected to mutation. Mutation of a bit involves flipping it,changing 0 to 1 and vice-versa.

- Selection replicates the most successful solutions found in a population at a rate proportional to their relative quality
- Crossover decomposes two distinct solutions and then randomly mixes their parts to form novel solutions
- Mutation randomly produces a candidate solution.

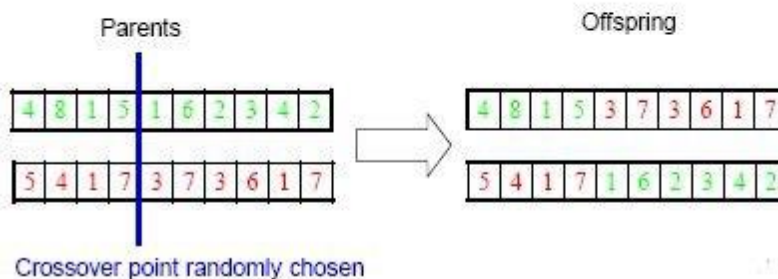
Genetic Algorithm

- GA starts with k randomly generated states (population)
- A state is represented a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function) defines fitness value of each states.
- Produce the next generation of states by selection, crossover, and mutation.
- The primary advantage of GA comes from crossover operation.



Definitions

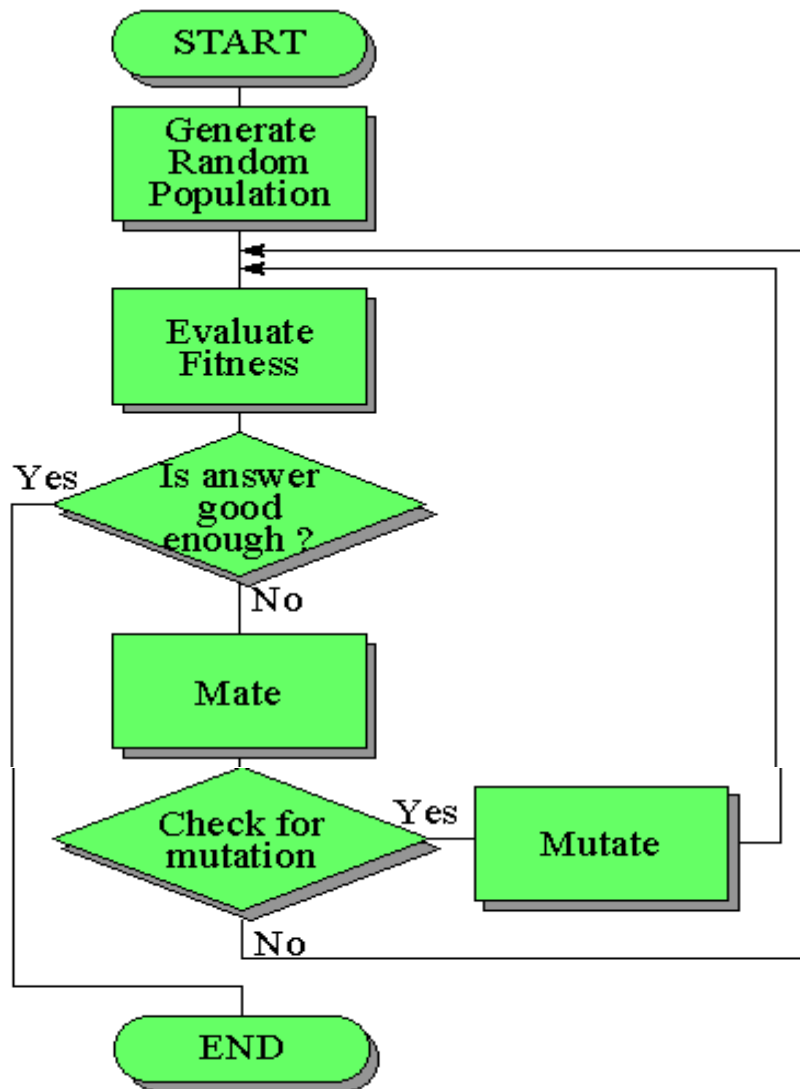
- **Selection:** Pick two random individuals for reproduction
- **Crossover:** Mix the two parent strings at the crossover point



Algorithm

```
produce an initial population of individuals
evaluate the fitness of all individuals
while (solution not found)
    select fitter individuals for reproduction
    recombine between individuals
    mutate individuals
    evaluate the fitness of the modified individuals
    generate a new population
end while
```

GA flowchart



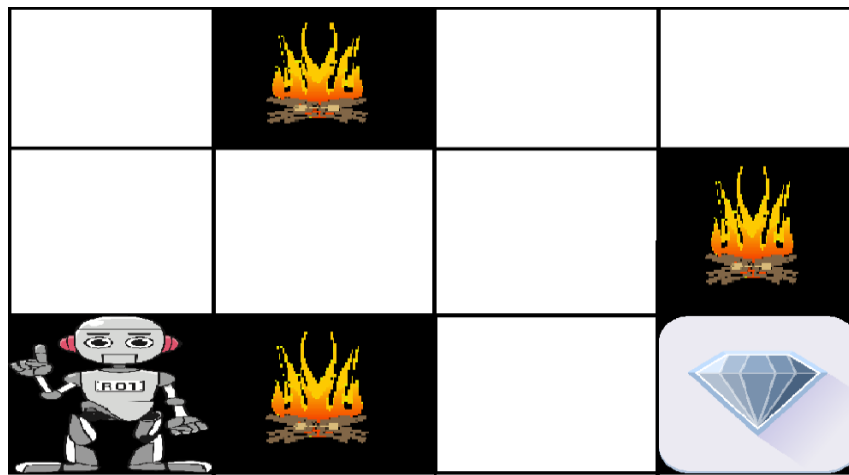
Disadvantage

- GA is better if the problem does not have any mathematical model for the solution.
- GA is less efficient in terms of speed of convergence.
- GA has tendency to get stuck in local maxima rather than global maxima.

Reinforced Learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.

Example : The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.



The above image shows robot, diamond and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that is fire. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning :

1. Input: The input should be an initial state from which the model will start
2. Output: There are many possible output as there are variety of solution to a particular problem
3. Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
4. The model keeps continues to learn.
5. The best solution is decided based on the maximum reward.

Difference between Reinforcement learning and Supervised learning:

REINFORCEMENT LEARNING	SUPERVISED LEARNING
Reinforcement learning is all about making decisions sequentially. In simple words we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	Supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game	Example: Object recognition

Types of Reinforcement: There are two types of Reinforcement:

Positive – Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words it has a positive effect on the behavior.

Advantages:

- Maximizes Performance
- Sustain Change for a long period of time

Disadvantages:

- Too much Reinforcement can lead to overload of states which can diminish the results

Negative – Negative Reinforcement is defined as strengthening of a behavior because a negative condition is stopped or avoided.

Advantages of reinforcement learning:

- Increases Behavior
- Provide defiance to minimum standard of performance

Disadvantages of reinforcement learning:

- It Only provides enough to meet up the minimum behavior

Various Practical applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.