[Unit 5: Concepts of Machine Learning]

Artificial Intelligence (MDS 556)

Master in Data Science

Introduction to Machine Learning

An agent is **learning** if it improves its performance on future tasks after making observations about the world. Learning can range from the trivial, as exhibited by jotting down a phone number, to the profound, as exhibited by Albert Einstein, who inferred a new theory of the universe.

Why would we want an agent to learn? If the design of the agent can be improved, why wouldn't the designers just program in that improvement to begin with?

There are three main reasons. First, the designers cannot anticipate all possible situations that the agent might find itself in. For example, a robot designed to navigate mazes must learn the layout of each new maze it encounters.

Second, the designers cannot anticipate all changes over time; a program designed to predict tomorrow's stock market prices must learn to adapt when conditions change from boom to bust.

Third, sometimes human programmers have no idea how to program a solution themselves. For example, most people are good at recognizing the faces of family members, but even the best programmers are unable to program a computer to accomplish that task, except by using learning algorithms.

Forms of Learning

Any component of an agent can be improved by learning from data. The improvements, and the techniques used to make them, depend on four major factors:

- Which *component* is to be improved?
- What *prior knowledge* the agent already has?
- What *representation* is used for the data and the component?
- What *feedback* is available to learn from?

Components to be learned

The components of agent designs include;

1. A direct mapping from conditions on the current state to actions.
2. A means to infer relevant properties of the world from the percept sequence.
3. Information about the way the world evolves and about the results of possible actions the agent can take.
4. Utility information indicating the desirability of world states.
5. Action-value information indicating the desirability of actions.
6. Goals that describe classes of states whose achievement maximizes the agent's utility.

Each of these components can be learned. Consider, for example, an agent training to become a **taxi driver**.

Every time the instructor shouts "Brake!" the agent might learn a condition– action rule for when to brake (component 1); the agent also learns every time the instructor does not shout.

By seeing many camera images that it is told contain buses, it can learn to recognize them (2).

By trying actions and observing the results—for example, braking hard on a wet road—it can learn the effects of its actions (3).

Then, when it receives no tip from passengers who have been thoroughly shaken up during the trip, it can learn a useful component of its overall utility function (4).

Feedback to learn from

There are three *types of feedback* that determine the three main types of learning:

In **unsupervised learning** the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is **clustering**: detecting potentially useful clusters of input examples. For example, a taxi agent might gradually develop a concept of "good traffic days" and "bad traffic days" without ever being given labeled examples of each by a teacher.

In **reinforcement learning** the agent learns from a series of reinforcements—rewards or punishments. For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong. The two points for a win at the end of a chess game tells the agent it did something right. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it.

In **supervised learning** the agent observes some example input–output pairs and learns a function that maps from input to output.

In component 1 above, the inputs are percepts and the output are provided by a teacher who says "Brake!" or "Turn left."

In component 2, the inputs are camera images and the outputs again come from a teacher who says "that's a bus."

In 3, the theory of braking is a function from states and braking actions to stopping distance in feet. In this case the output value is available directly from the agent's percepts (after the fact); the environment is the teacher.

Learning with a Teacher

Learning with a teacher is also referred to as **supervised learning**. Following figure shows a block diagram that illustrates this form of learning. In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of input–output examples. The environment is, however, unknown to the neural network.

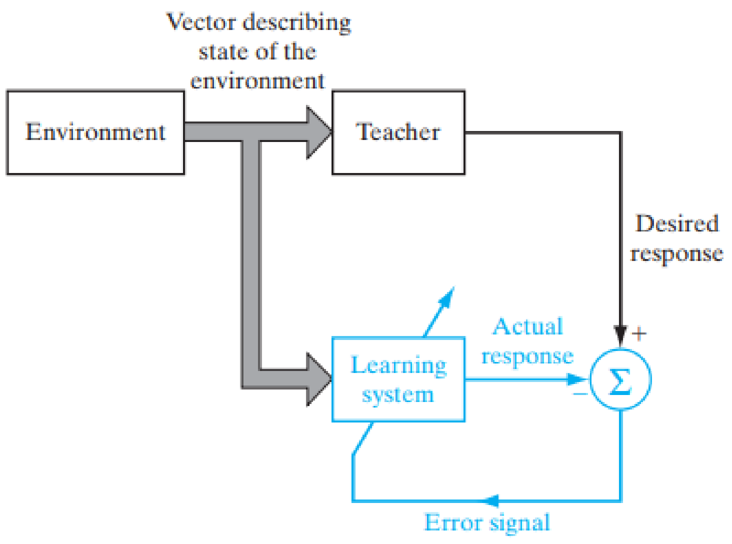


FIGURE 24 Block diagram of learning with a teacher; the part of the figure printed in red constitutes a feedback loop.

Suppose now that the teacher and the neural network are both exposed to a training vector (i.e., example) drawn from the same environment. By virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the "optimum" action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. The error signal is defined as the difference between the desired response and the actual response of the network.

This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network emulate the teacher; the emulation is presumed to be optimum in some statistical sense. In this way, knowledge of the environment available to the teacher is transferred to the neural network through training and stored in the form of "fixed" synaptic weights, representing long-term memory

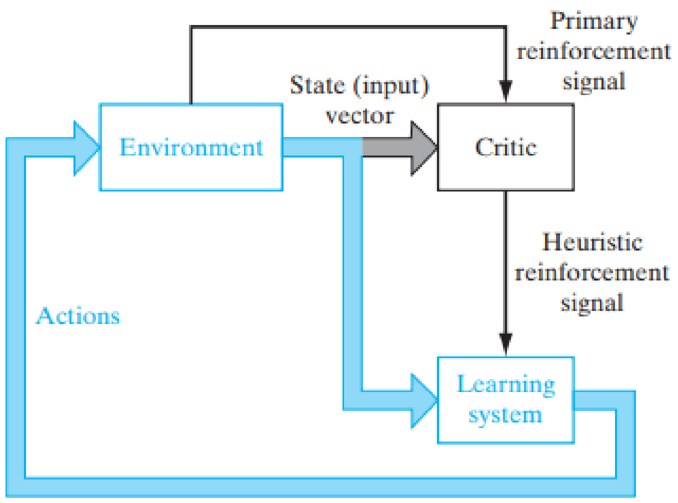
Learning without a Teacher

In supervised learning, the learning process takes place under the tutelage of a teacher. However, in the paradigm known as learning without a teacher, as the name implies, there is no teacher to oversee the learning process. That is to say, there are no labeled examples of the function to be learned by the network. Under this second paradigm, two subcategories are identified:

Reinforcement Learning

In reinforcement learning, the learning of an input–output mapping is performed through continued interaction with the environment in order to minimize a scalar index of performance. Following figure shows the block diagram of one form of a reinforcement-learning system built around a critic that converts a primary reinforcement signal received from the environment into a higher quality reinforcement signal called the heuristic reinforcement signal. The system is designed to learn under delayed reinforcement, which means that the system observes a temporal sequence of stimuli also received from the environment, which eventually result in the generation of the heuristic reinforcement signal.

FIGURE 25 Block diagram of reinforcement learning; the learning system and the environment are both inside the feedback loop.



The goal of reinforcement learning is to minimize a cost-to-go function, defined as the expectation of the cumulative cost of actions taken over a sequence of steps instead of simply the immediate cost. It may turn out that certain actions taken earlier in that sequence of time steps are in fact the best determinants of overall system behavior. The function of the learning system is to discover these actions and feed them back to the environment.

Unsupervised Learning

In unsupervised, or self-organized, learning, there is no external teacher or critic to oversee the learning process, as indicated in following figure. Rather, provision is made for a task-independent measure of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. For a specific task-independent measure, once the network has become tuned to the statistical regularities of the input data, the network develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically.

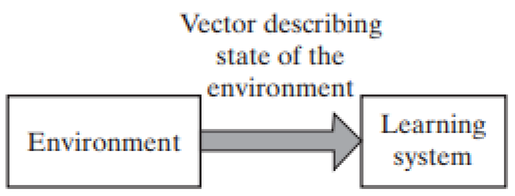
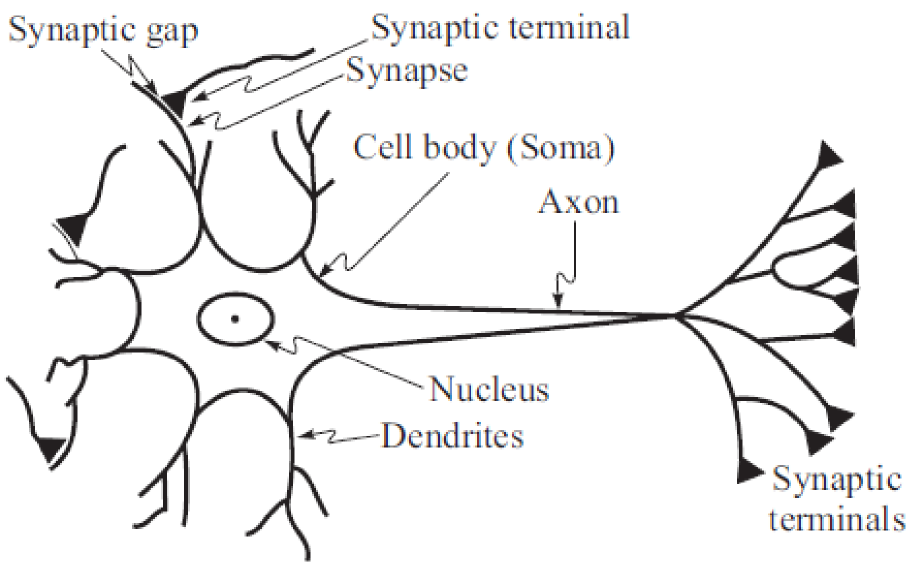


FIGURE 26 Block diagram of unsupervised learning.

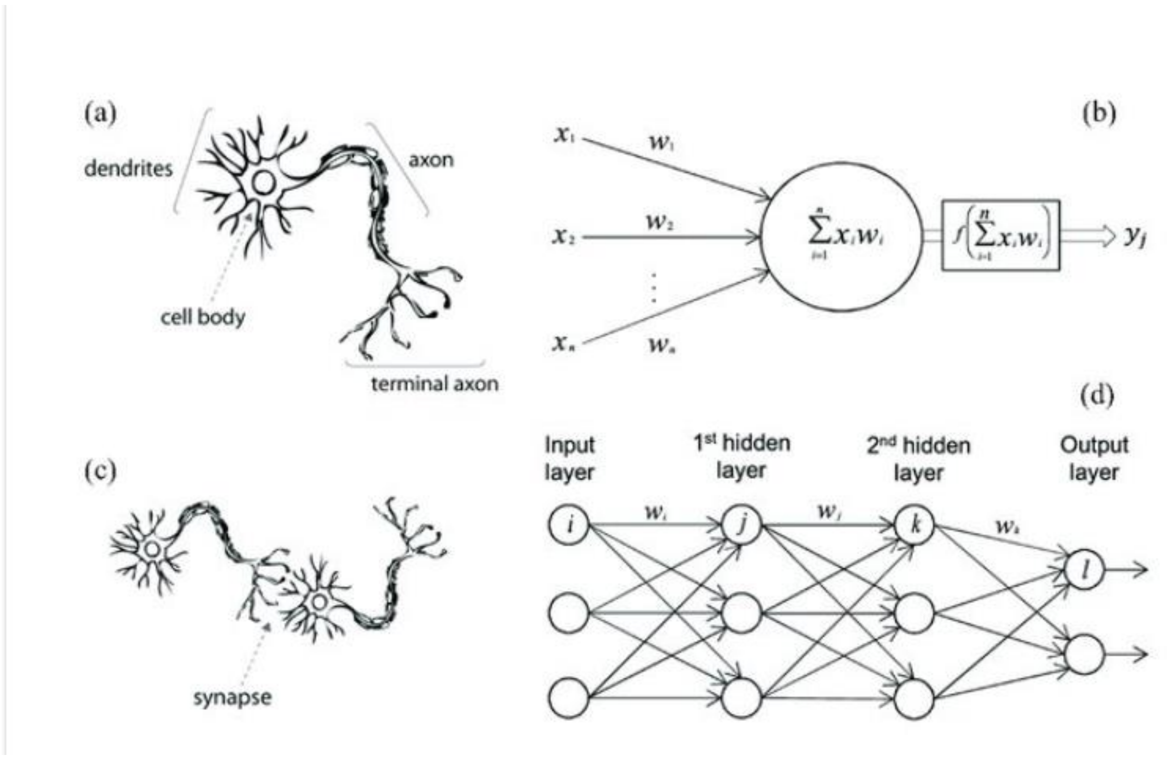
Neural Networks:

A neuron is a cell in brain whose principle function is the collection, Processing, and dissemination of electrical signals. Brains Information processing capacity comes from networks of such neurons. Due to this reason some earliest AI work aimed to create such artificial networks. (Other Names are Connectionism; Parallel distributed processing and neural computing).



What is a Neural Network?

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.



Why use neural networks?

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage

Neural networks versus conventional computers

Neural networks take a different approach to problem solving than that of conventional computers. **Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve.** But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. **The network is composed of a large number of highly interconnected processing elements (neurones) working in parallel to solve a specific problem.** Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Units of Neural Network:

Nodes (units):

Nodes represent a cell of neural network.

Links:

Links are directed arrows that show propagation of information from one node to another node.

Activation:

Activations are inputs to or outputs from a unit.

Weight:

Each link has weight associated with it which determines strength and sign of the connection.

Activation function:

A function which is used to derive output activation from the input activations to a given node is called activation function.

Bias Weight:

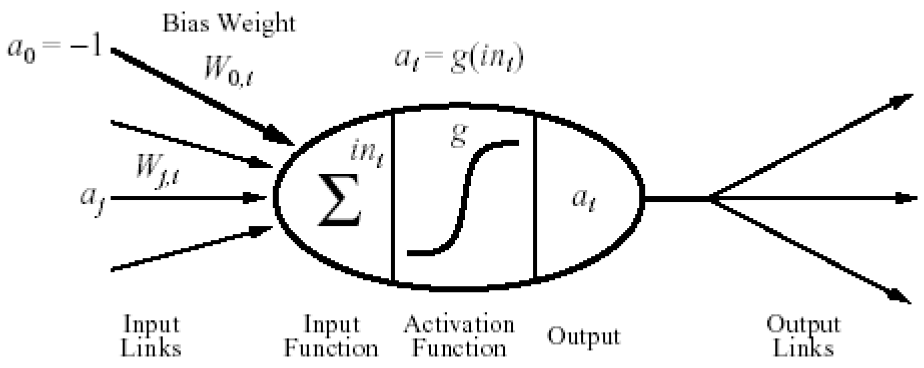
Bias weight is used to set the threshold for a unit. Unit is activated when the weighted sum of real inputs exceeds the bias weight.

Biological neuron	**Artificial neuron**
dendrites	inputs
synapses	weight or inter connection
axon	output
cell body (Soma)	summation and threshold

Simple Model of Neural Network

A simple mathematical model of neuron is devised by McCulloch and Pit is given in the figure given below:

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



It fires when a linear combination of its inputs exceeds some threshold.

A neural network is composed of nodes (units) connected by directed links A link from unit j to i serve to propagate the activation a_j from j to i. Each link has some numeric weight $W_{j,i}$ associated with it, which determines strength and sign of connection.

Each unit first computes a weighted sum of it's inputs:

$$in_i = \sum_{J=0}^{n} W_{j,i}\ a_j$$

Then it applies activation function g to this sum to derive the output:

$$a_i = g(in_i) = g(\sum_{J=0}^{n} W_{j,i}\, a_j)$$

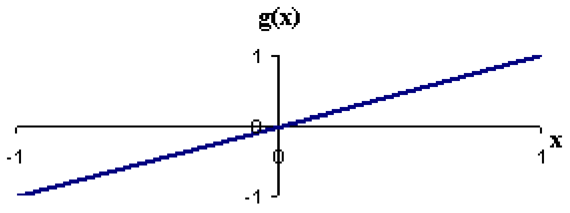
Here, a_j output activation from unit j and $W_{j,i}$ is the weight on the link j to this node.

Activation function can be of various categories. For example;

- Linear
- Threshold (*Heaviside function)*
- Sigmoid
- Sign

For **linear activation functions**, the output activity is proportional to the total weighted output.

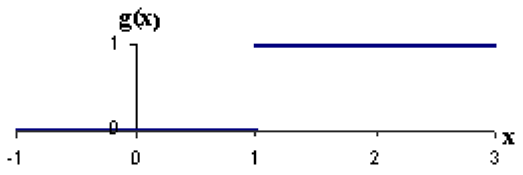
$g(x) = k x + c$, where k and c are constant



For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

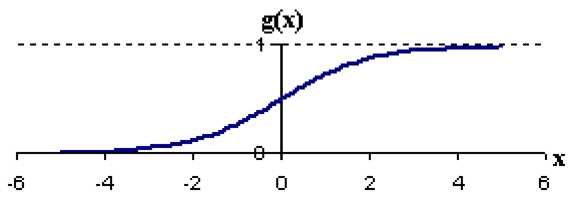
$g(x) = 1$ if $x >= k$

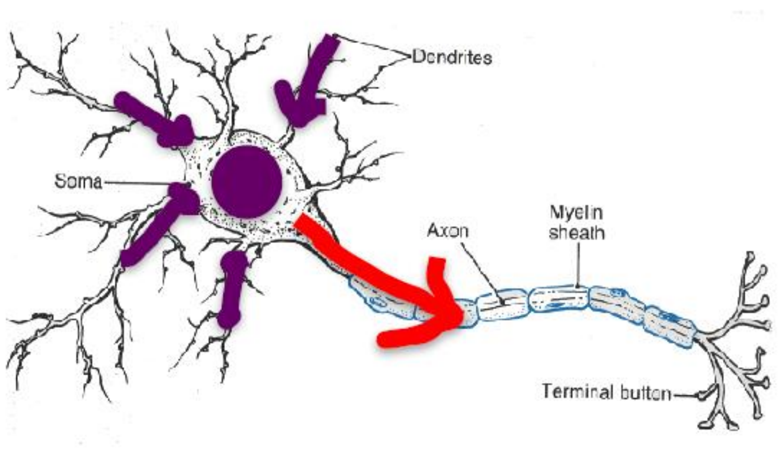
$= 0$ if $x < k$

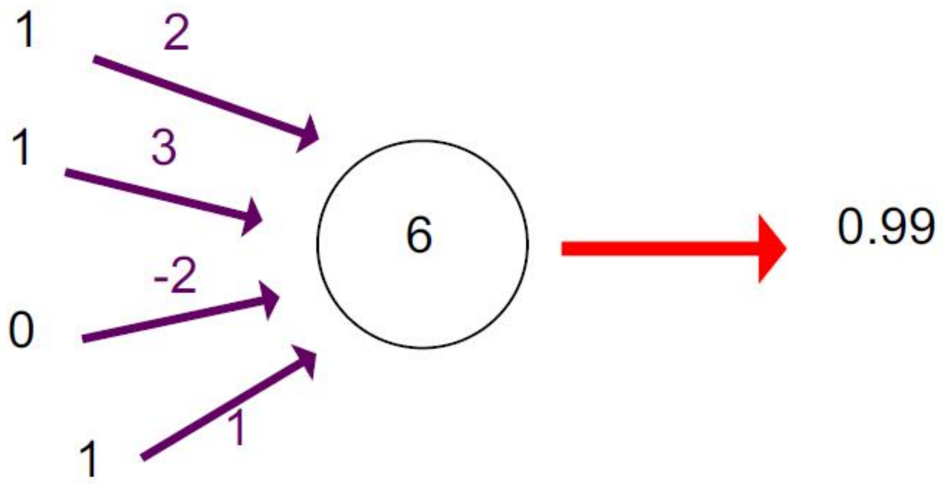


For **sigmoid activation functions**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units. It has the advantage of differentiable.

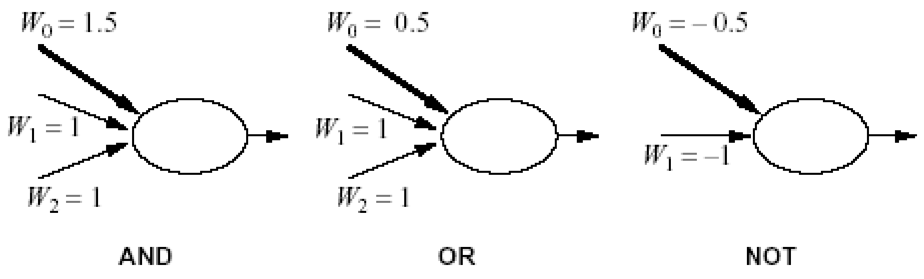
$g(x) = 1/ (1 + e^{-x})$




Dendrites
Soma
Axon
Myelin sheath
Terminal button


1
2
1
3
-2
0
1
1
6
0.99

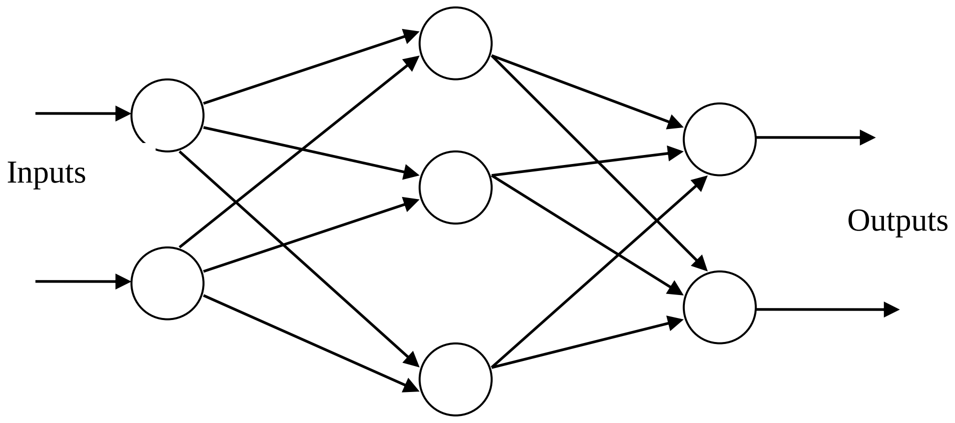
Realizing logic gates by using Neurons:



Network structures:

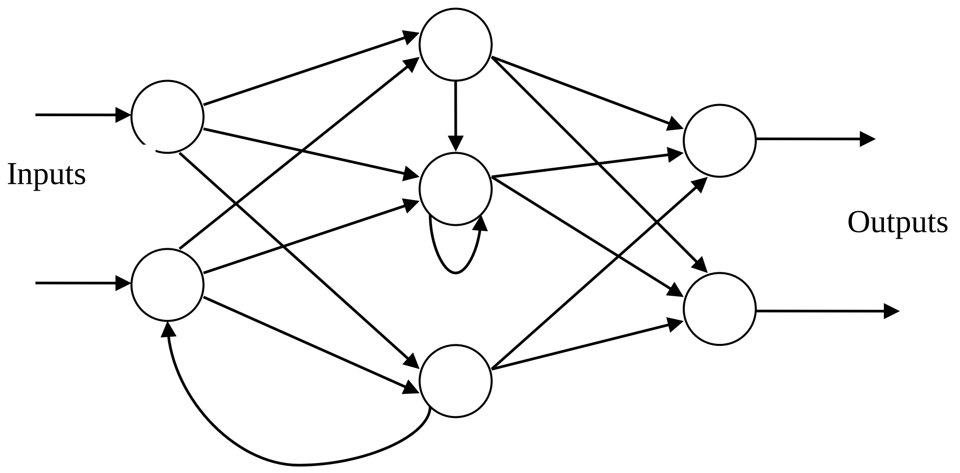
Feed-forward networks:

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.



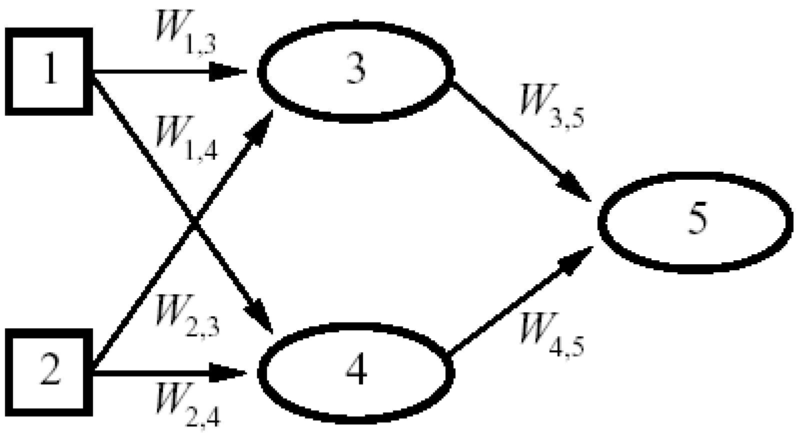
Feedback networks (Recurrent networks :)

Feedback networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new



equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent.

Feed-forward example

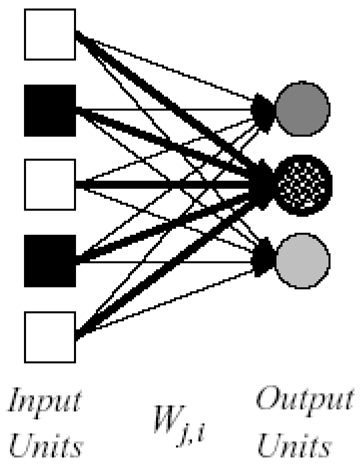


Here;
$y_5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$
$= g(W_{3;5}\ g(W_{1;3}\ a_1 + W_{2;3}\ a_2) + W4;5\ g(W_{1;4}\ a_1 + W_{2;4}\ a_2)$

Types of Feed Forward Neural Network:

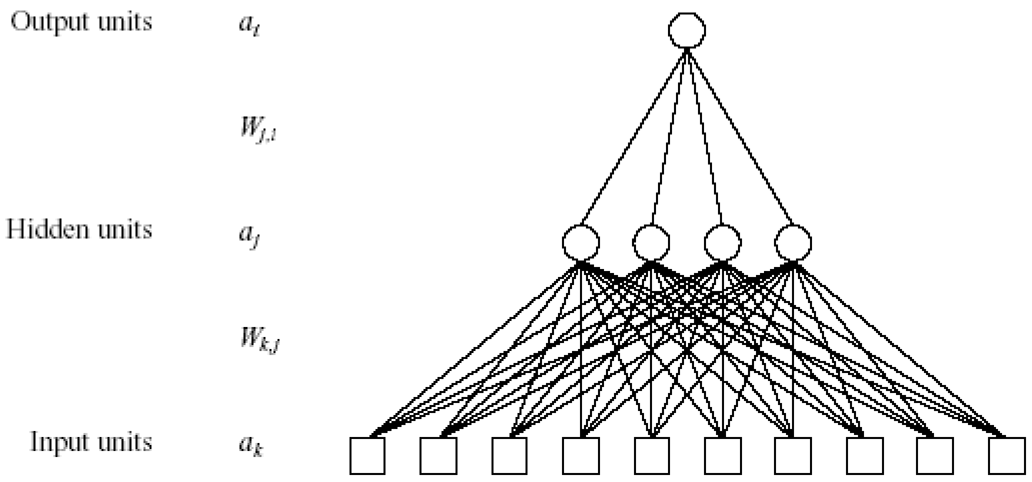
Single-layer neural networks (perceptrons)

A neural network in which all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network. Since each output unit is independent of the others each weight affects only one of the outputs.



Multilayer neural networks (perceptrons)

The neural network which contains input layers, output layers and some hidden layers also is called multilayer neural network. The advantage of adding hidden layers is that it enlarges the space of hypothesis. Layers of the network are normally fully connected.



Once the number of layers, and number of units in each layer, has been selected, training is used to set the network's weights and thresholds so as to minimize the prediction error made by the network

Training is the process of adjusting weights and threshold to produce the desired result for different set of data.

Learning in Neural Networks:

Learning: One of the powerful features of neural networks is learning. **Learning in neural networks is carried out by adjusting the connection weights among neurons**. It is similar to a biological nervous system in which learning is carried out by changing synapses connection strengths, among cells.

The operation of a neural network is determined by the values of the interconnection weights. There is no algorithm that determines how the weights should be assigned in order to solve specific problems. Hence, the weights are determined by a learning process

Perceptron Learning Theory:

The term "Perceptrons" was coined by Frank RosenBlatt in 1962 and is used to describe the connection of simple neurons into networks. These networks are simplified versions of the real nervous system where some properties are exagerrated and others are ignored. For the moment we will concentrate on Single Layer Perceptrons.

So how can we achieve learning in our model neuron? We need to train them so they can do things that are useful. To do this we must allow the neuron to learn from its mistakes. There is in fact a learning paradigm that achieves this, it is known as supervised learning and works in the following manner.

i. set the weight and thresholds of the neuron to random values.
ii. present an input.
iii. caclulate the output of the neuron.
iv. alter the weights to reinforce correct decisions and discourage wrong decisions, hence reducing the error. So for the network to learn we shall increase the weights on the active inputs when we want the output to be active, and to decrease them when we want the output to be inactive.
v. Now present the next input and repeat steps iii. - v.

Perceptron Learning Algorithm:

The algorithm for Perceptron Learning is based on the supervised learning procedure discussed previously.

Algorithm:

i. Initialize weights

   Set $w_i(t)$, $(0 <= i <= n)$, to be the weight i at time t..

   Set $w_i(0)$ to small random values, thus initializing the weights and threshold.

ii. Present input and desired output

Jagdish Bhatta

Present input x_0, x_1, x_2, ..., x_n and desired output $d(t)$

iii. Calculate the actual output

$y(t) = g\,[w_0(t)x_0(t) + w_1(t)x_1(t) + ... + w_n(t)x_n(t)]$

iv. Adapts weights

$w_i(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t)$, where $0 <= \alpha <= 1$ (learning rate) is a positive gain function that controls the adaption rate.

Steps iii. and iv. are repeated until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

The weights only change if an error is made and hence this is only when learning shall occur.

Back-propagation Algorithm

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, backpropagation 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 "backpropagation" 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. Backpropagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

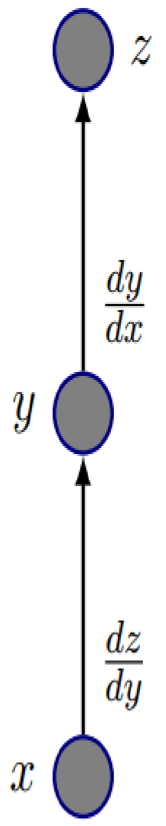
Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer).

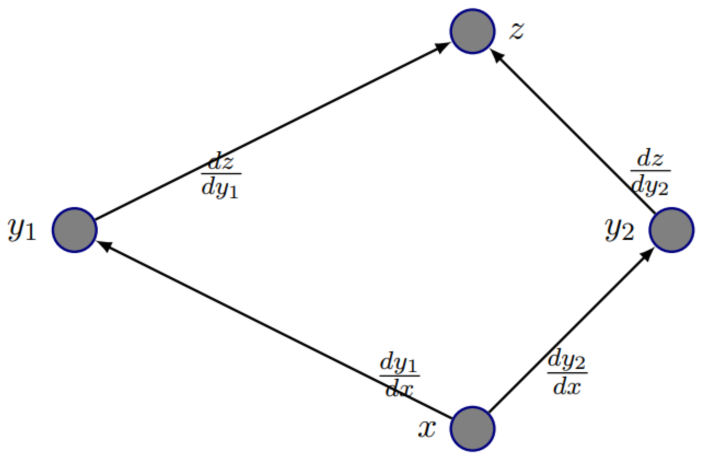
Backpropagation computes the chain rule, in a manner that is highly efficient.

Let f, g : R → R

Suppose y = g(x) and z = f(y) = f(g(x))

Chain rule: dz/dx = dz/dy * dy/dx

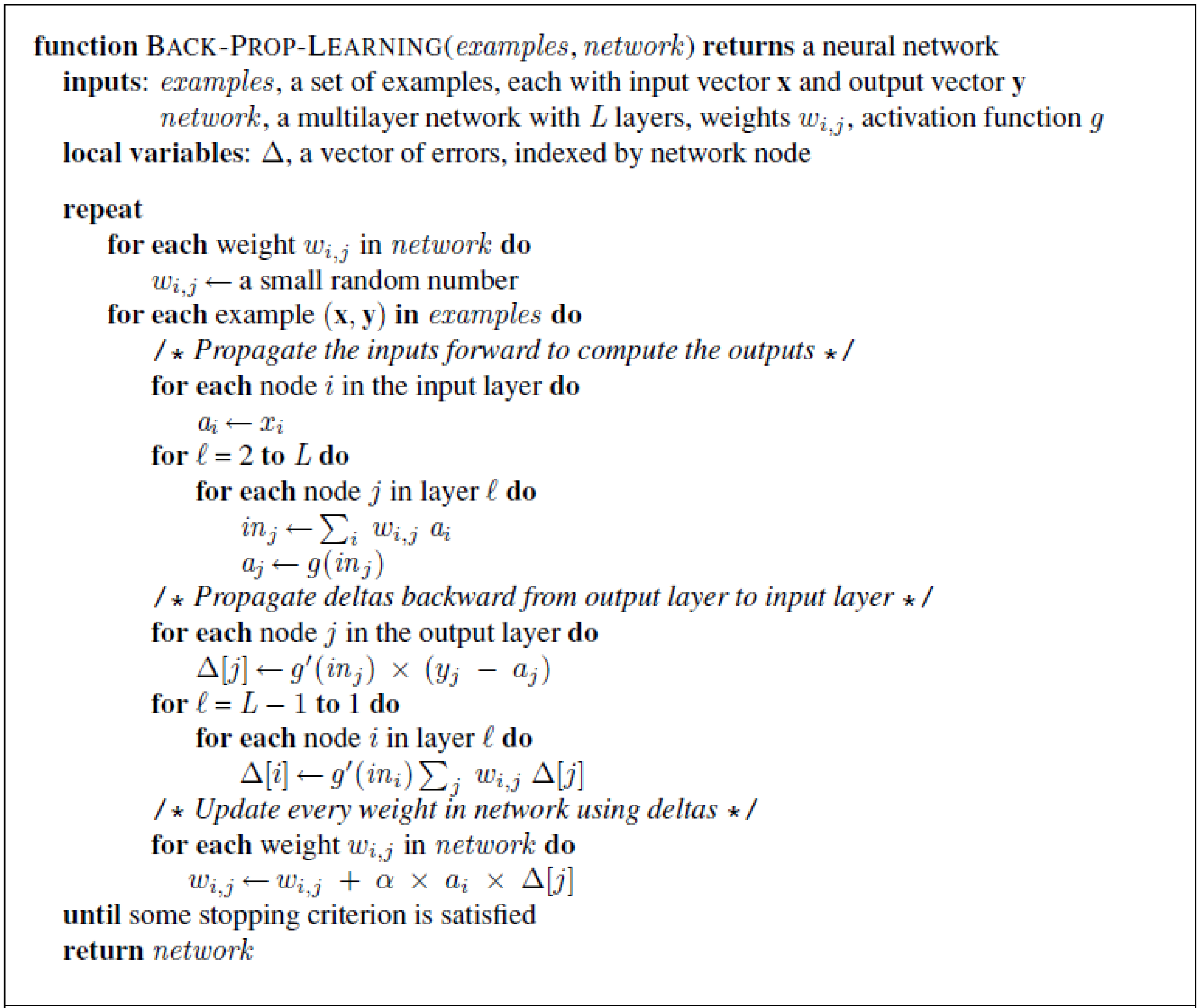




Multiple Paths: $\frac{dz}{dx} = \frac{dz}{dy_1}\frac{dy_1}{dx} + \frac{dz}{dy_2}\frac{dy_2}{dx}$

Chain rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

To calculate the direction of steepest descent along the error surface, we have to compute the derivative with respect to each component of the vector w.



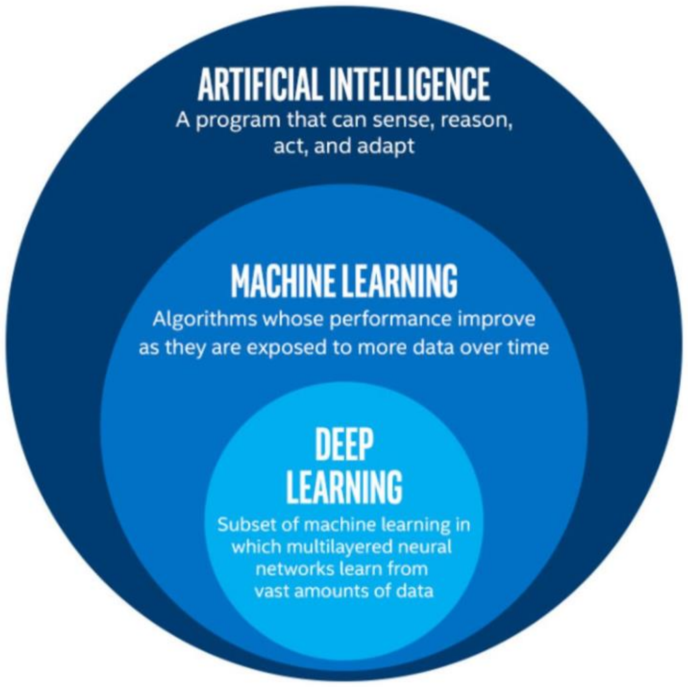
```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
  local variables: $\Delta$, a vector of errors, indexed by network node

  repeat
     for each weight $w_{i,j}$ in network do
        $w_{i,j} \leftarrow$ a small random number
     for each example (x, y) in examples do
        /* Propagate the inputs forward to compute the outputs */
        for each node $i$ in the input layer do
           $a_i \leftarrow x_i$
        for $\ell = 2$ to $L$ do
           for each node $j$ in layer $\ell$ do
              $in_j \leftarrow \sum_i w_{i,j}\, a_i$
              $a_j \leftarrow g(in_j)$
        /* Propagate deltas backward from output layer to input layer */
        for each node $j$ in the output layer do
           $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
        for $\ell = L - 1$ to $1$ do
           for each node $i$ in layer $\ell$ do
              $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
        /* Update every weight in network using deltas */
        for each weight $w_{i,j}$ in network do
           $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return network
```

Figure 18.24 The back-propagation algorithm for learning in multilayer networks.

The back-propagation process can be summarized as follows:

- Compute the Δ values for the output units, using the observed error.
- Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
  - Propagate the Δ values back to the previous layer.
  - Update the weights between the two layers.

Deep Learning


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Deep learning is an aspect of artificial intelligence (AI) that is to simulate the activity of the human brain specifically, pattern recognition by passing input through various layers of the neural network.

Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs.

Deep learning algorithms are constructed with connected layers.

- The first layer is called the Input Layer
- The last layer is called the Output Layer
- All layers in between are called Hidden Layers.

The word deep means the network join neurons in more than two layers.

Each Hidden layer is composed of neurons. The neurons are connected to each other. The neuron will process and then propagate the input signal it receives the layer

above it. The strength of the signal given the neuron in the next layer depends on the weight, bias and activation function.

The network consumes large amounts of input data and operates them through multiple layers; the network can learn increasingly complex features of the data at each layer.

The traditional machine learning methods including decision trees, SVM, naïve Bayes classifier and logistic regression. These algorithms are also called flat algorithms. “Flat” here refers to the fact these algorithms cannot normally be applied directly to the raw data (such as .csv, images, text, etc.). We need a preprocessing step called feature extraction.

The result of feature extraction is a representation of the given raw data that these classic machine learning algorithms can use to perform a task. For example, we can now classify the data into several categories or classes. Feature extraction is usually quite complex and requires detailed knowledge of the problem domain. This preprocessing layer must be adapted, tested and refined over several iterations for optimal results.

Deep learning’s artificial neural networks don’t need the feature extraction step. The layers are able to learn an implicit representation of the raw data directly and on their own.

How Deep Learning Algorithms Work?

While deep learning algorithms feature self-learning representations, they depend upon ANNs that mirror the way the brain computes information. During the training process, algorithms use unknown elements in the input distribution to extract features, group objects, and discover useful data patterns. Much like training machines for self-learning, this occurs at multiple levels, using the algorithms to build the models.

Deep learning models make use of several algorithms. While no one network is considered perfect, some algorithms are better suited to perform specific tasks. To choose the right ones, it’s good to gain a solid understanding of all primary algorithms.

Types of Algorithms used in Deep Learning

Here is the list of top 10 most popular deep learning algorithms:

1. Convolutional Neural Networks (CNNs)
2. Long Short Term Memory Networks (LSTMs)
3. Recurrent Neural Networks (RNNs)
4. Generative Adversarial Networks (GANs)
5. Radial Basis Function Networks (RBFNs)

6. Multilayer Perceptrons (MLPs)
7. Self Organizing Maps (SOMs)
8. Deep Belief Networks (DBNs)
9. Restricted Boltzmann Machines(RBMs)
10. Autoencoders

Convolutional Neural Networks (CNNs)

CNN's, also known as ConvNets, consist of multiple layers and are mainly used for image processing and object detection. Yann LeCun developed the first CNN in 1998 when it was called LeNet. It was used for recognizing characters like ZIP codes and digits.

CNN's are widely used to identify satellite images, process medical images, forecast time series, and detect anomalies.

How Do CNNs Work?

CNN's have multiple layers that process and extract features from data:

Convolution Layer

CNN has a convolution layer that has several filters to perform the convolution operation.

Rectified Linear Unit (ReLU)

CNN's have a ReLU layer to perform operations on elements. The output is a rectified feature map.

Pooling Layer

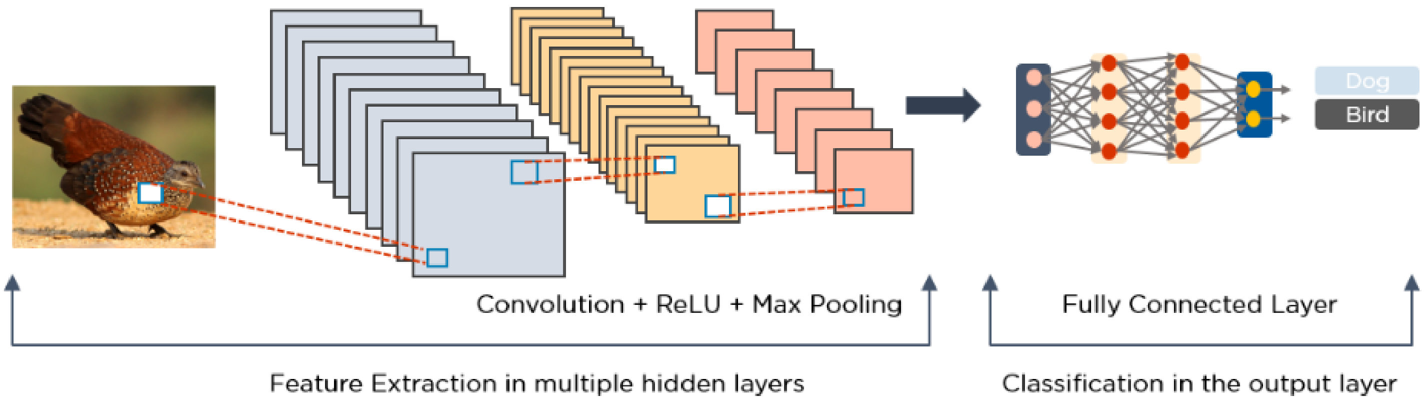
The rectified feature map next feeds into a pooling layer. Pooling is a down-sampling operation that reduces the dimensions of the feature map.

The pooling layer then converts the resulting two-dimensional arrays from the pooled feature map into a single, long, continuous, linear vector by flattening it.

Fully Connected Layer:

A fully connected layer forms when the flattened matrix from the pooling layer is fed as an input, which classifies and identifies the images.

Below is an example of an image processed via CNN.


Dog
Bird
Convolution + ReLU + Max Pooling
Fully Connected Layer
Feature Extraction in multiple hidden layers
Classification in the output layer

Statistical-based Learning: Naive Bayes Model

Naïve Bayes is a probabilistic machine learning algorithm based on the Bayes Theorem, used in a wide variety of classification tasks. Naïve Bayes is a probabilistic machine learning algorithm based on the **Bayes Theorem**, used in a wide variety of classification tasks.

In this model, the "class" variable C (which is to be predicted) is the root and the "attribute" variables Xi are the leaves. The model is "naive" because it assumes that the attributes are conditionally independent of each other, given the class.

Bayesian classification is based on Bayes' theorem. It is also called Naïve Bayes Classification or Naïve Bayesian Classification. Bayes Theorem is given by:

$$P(H \mid X) = \frac{P(X \mid H)P(H)}{P(X)}$$

Bayes' theorem is useful in that it provides a way of calculating the posterior probability, $P(H|X)$, from $P(H)$, $P(X|H)$, and $P(X)$. Here P(X) and P(H) are prior probability.

Let D be a set of data and C_1, C_2, ……, C_m are m classes labels of the data. Now above Bayes rule can be written as:

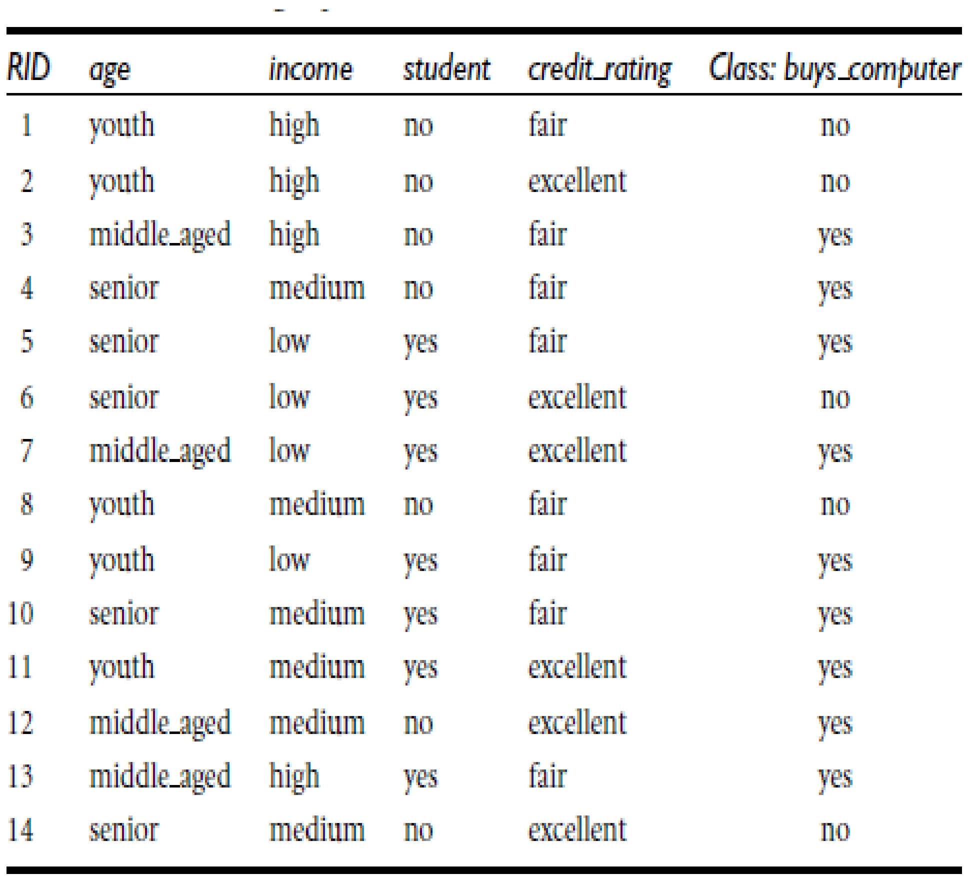
$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)}$$

Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. Thus we need to maximize $P(C_i|X)$. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need to be maximized.

Let X is the set of attributes {x_1, x_2, x_3……x_n} where attributes are independent of one another. Now the probability P(X|C$_i$) is given by the equation given below:

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) = P(x_1 \mid C_i) \times P(x_2 \mid C_i) \ldots\ldots \times P(x_n \mid C_i)$$

Consider an example;



RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Predict class level of the tuple: *X* = (*age* = *youth*, *income* = *medium*, *student* = *yes*, *credit_rating* = *fair*) using Bayesian classification.

Solution
Prior probability of each class can be computed based on the training tuples:

$$P(buys\_computer = yes) = 9/14 = 0.643$$
$$P(buys\_computer = no) = 5/14 = 0.357$$

Compute the following conditional probabilities:

$P(age = youth \mid buys\_computer = yes) = 2/9 = 0.222$

$P(age = youth \mid buys\_computer = no) = 3/5 = 0.6$

$P(income = medium \mid buys\_computer = yes) = 4/9 = 0.444$

$P(income = medium \mid buys\_computer = no) = 2/5 = 0.4$

$P(student = yes \mid buys\_computer = yes) = 6/9 = 0.667$

$P(student = yes \mid buys\_computer = no) = 1/5 = 0.2$

$P(credit\_rating = fair \mid buys\_computer = yes) = 6/9 = 0.667$

$P(credit\_rating = fair \mid buys\_computer = no) = 2/5 = 0.4$

Using the above probabilities, we obtain

***P*(*X*|*buys_computer=yes*)**

=$P(age=youth|buys\_computer=ye \times P(income=medium|buys\_computer=yes) \times P(student=yes|buys\_computer=yes) \times P(credit\_rating=fair|buys\_computer=yes)$

= 0.222×0.444×0.667×0.667 = 0.044

***P*(*X*|*buys computer=no*)**

=$P(age=youth|buys\_computer=no) \times P(income=medium|buys\_computer=no) \times P(student=yes|buys\_computer=no) \times P(credit\_rating=fair|buys\_computer=no)$

= 0.6*0.4*0.2*0.4 = 0.019

Now,

***P*(*buys_computer=yes*|X)**

=$P(X|buys\_computer=yes) \times P(buys\_computer=yes)$= 0.044*0.643 = 0.028

$P(buys\_computer=no|\text{X})=P(X|buys\_computer=no) \times P(buys\_computer=no)$
= 0.019*0.357 = 0.007

Since,

$P(buys\_computer=yes|\text{X}) > P(buys\_computer=no|\text{X})$,

Therefore, the Naïve Bayesian Classifier prediction for Given X is: buys_computer = yes.

Learning by Evolutionary Approach: Genetic Algorithm

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a **pool or a population of possible solutions** to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. **Each individual (or candidate solution) is assigned a fitness value and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest".** In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.

The basic terminologies behind GA are;

Population − It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings. It is the set of chromosomes.

There are two primary methods to initialize a population in a GA. They are −

Random Initialization − Populate the initial population with completely random solutions.
Heuristic initialization − Populate the initial population using a known heuristic for the problem.

Chromosomes − A chromosome is one such solution to the given problem.

Gene − A gene is one element position of a chromosome.

Genotype − Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system. Complete set of genetic material (all chromosomes) is called genome. Particular set of genes in genome is called genotype.

Phenotype − Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations. The genotype is with later development after birth base for the organism's phenotype, its physical and mental characteristics, such as eye color, intelligence etc.

Decoding and Encoding − for simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. **Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space.** Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

Binary Encoding

Binary encoding is the most common one, mainly because the first research of GA used this type of encoding and because of its relative simplicity.

In **binary encoding**, every chromosome is a string of **bits - 0** or **1**.

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Permutation Encoding

Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem.

In **permutation encoding**, every chromosome is a string of numbers that represent a position in a **sequence**.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Permutation encoding is useful for ordering problems like TSP.

Value Encoding

Direct value encoding can be used in problems where some more complicated values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.

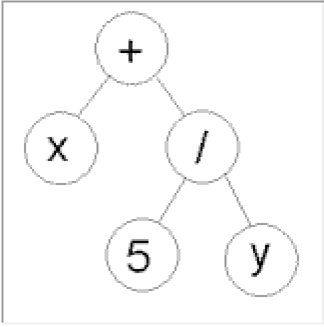
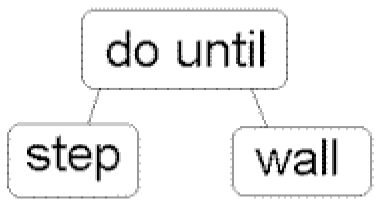
In the **value encoding**, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, chars or any objects.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, i.e. for **genetic programming**.

In the **tree encoding** every chromosome is a tree of some objects, such as functions or commands in programming language.

Chromosome A	Chromosome B
	
(+ x (/ 5 y))	(do_until step wall)

Tree encoding is useful for evolving programs or any other structures that can be encoded in trees.

Fitness Function − A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output.

Genetic Operators − They alter the genetic composition of the offspring. These include ***selection, crossover, and mutation.***

Selection:

The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation.

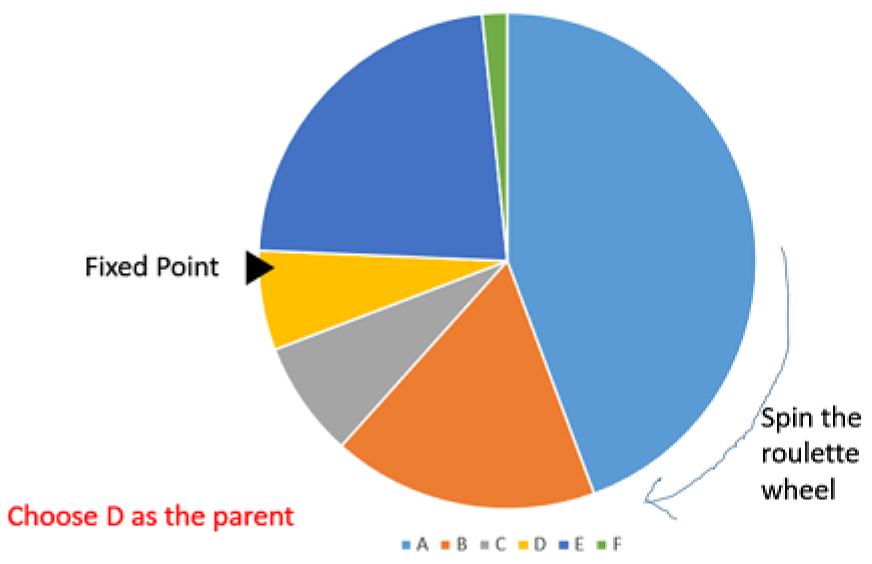
There are three types of Selection methods available, which are:

- Roulette wheel selection
- Stochastic universal sampling
- Tournament selection
- Rank-based selection

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided into slots of region. **A fixed point is chosen on the wheel circumference as shown and the wheel is rotated**. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated**.**

The probability for selection of an individual is proportional to its fitness. If all slots have same fitness probability of being selected is same. Instead, we can implement a weighted version of the roulette. **With it, the larger the fitness of an individual is, the more likely is its selection**:



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

If f_i is the fitness of individual *I* in the population, its probability of being selected is

$$p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j},$$

Here the sum of fitness is = 18.5

Here the percentage of area in the wheel for the chromosome A = 8.2/18.5=0.44=44%. Similarly for D=0.648=6.5%

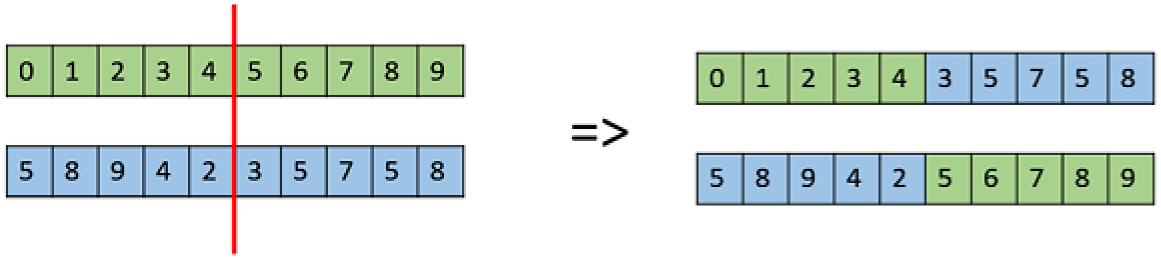
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. **Therefore, the probability of choosing an individual depends directly on its fitness.**

Crossover

The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents.

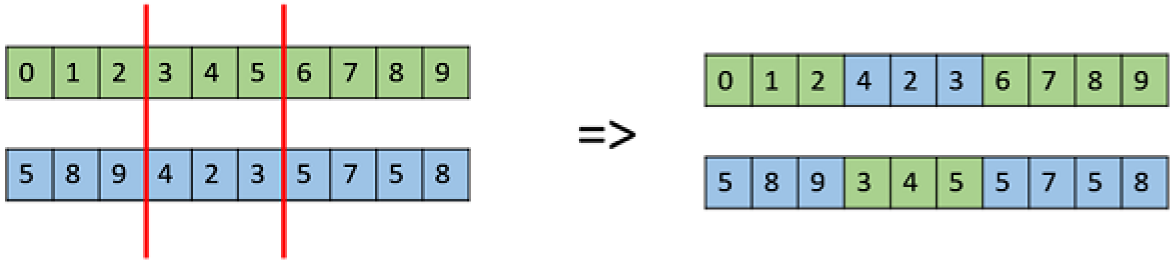
One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



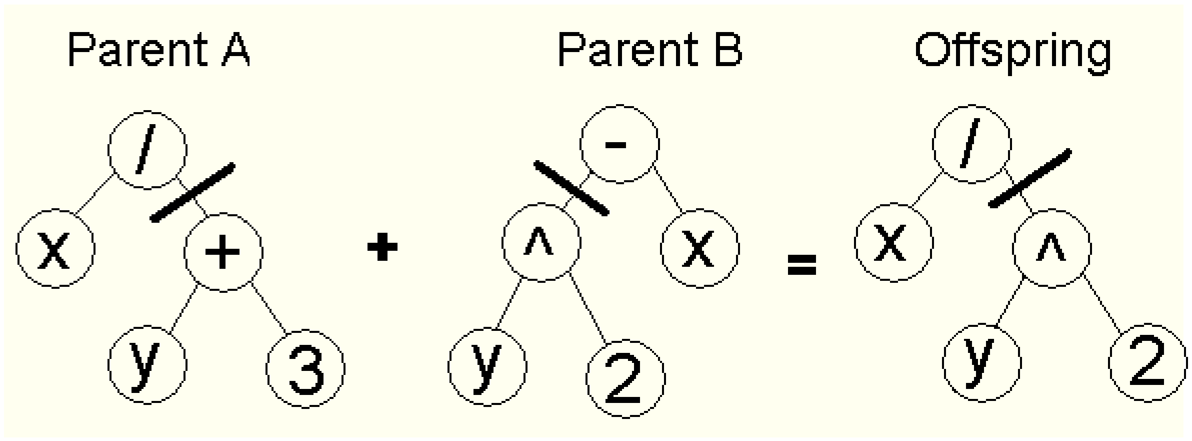
Multi Point Crossover

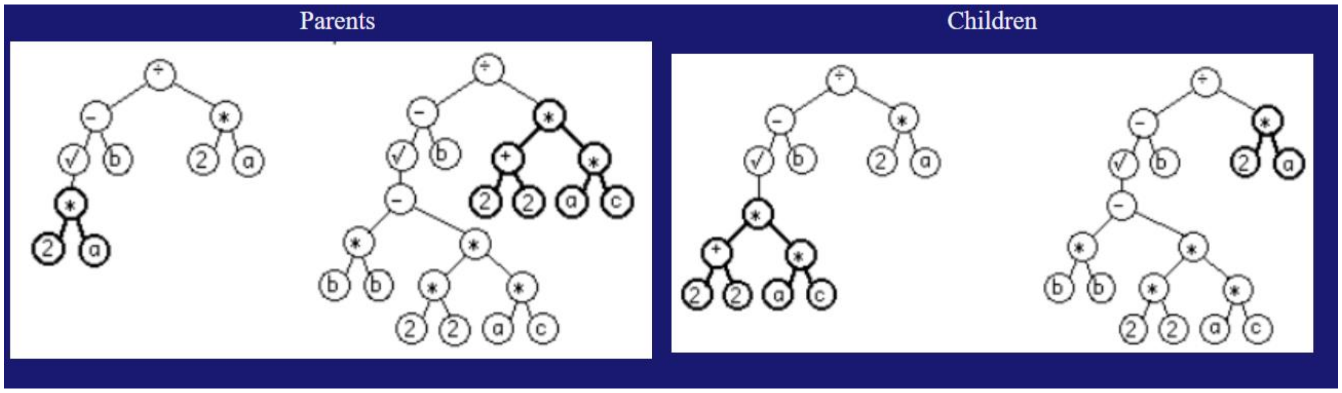
Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



Tree crossover

One crossover point is selected in both parents, parents are divided in that point and the parts below crossover points are exchanged to produce new offspring.





Mutation

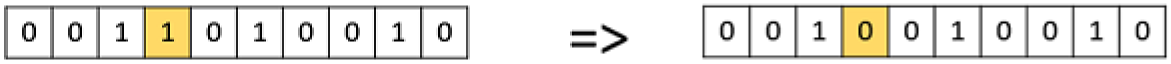
Mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population. The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes.

Mutation operators can be

- Bit flip mutation
- Random resetting
- Swap mutation and so on.

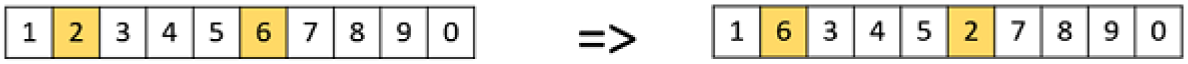
Bit Flip Mutation

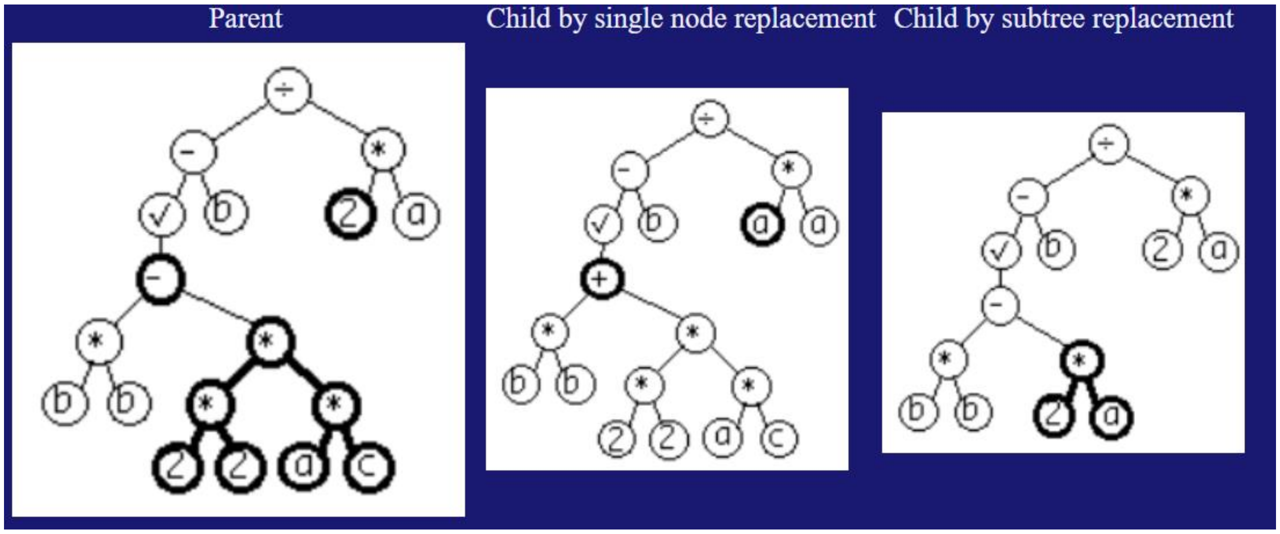
In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.



Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.




Parent
Child by single node replacement
Child by subtree replacement

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability: how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. **If crossover probability is 100%, then all offspring are made by crossover.** If it is 0%, whole new generation is made from exact copies of chromosomes from old population.

Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survive to next generation.

Mutation probability: how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. **If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.**

If the chromosome is mutated, normally a few bits are changed and so the solution is a bit changed. But **if all bits in chromosome** are mutated (because of 100% mutation probability), then the chromosome **string is in fact inverted**. Then we **have inversion and no mutation, so whole population degenerates very quickly**. It behaves like with no mutation (0% mutation probability).

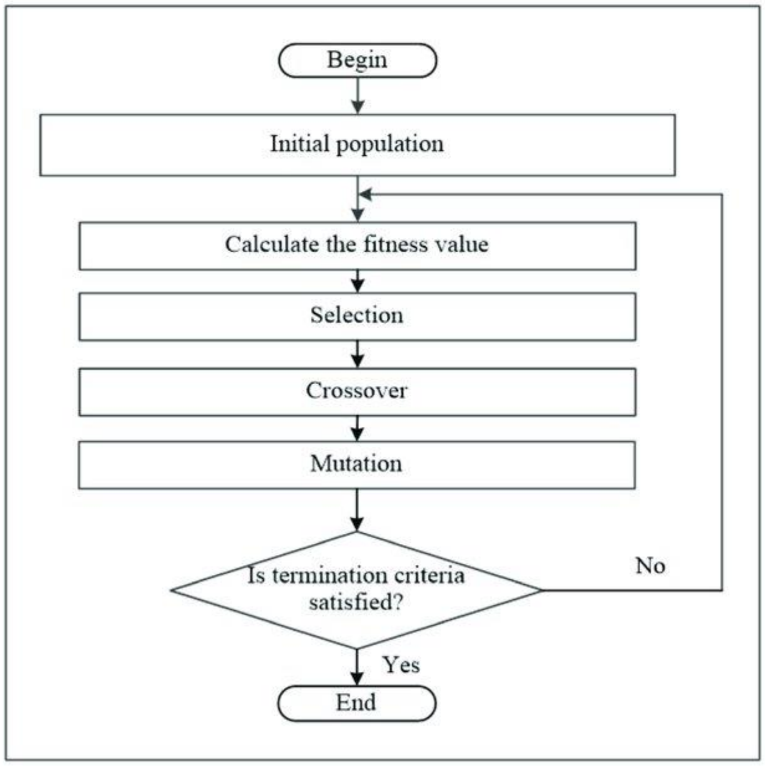
Termination Condition in GA

There are different termination conditions, which are listed below:

1. There is no improvement in the population for over x iterations.
2. We have already predefined an absolute number of generation for our algorithm.
3. When our fitness function has reached a predefined value.

GA Algorithm:

1. Generate random population of n individuals.
2. Evaluate the fitness of each individual.
3. Create a new population.
   a. Select two parents from a population according to their fitness.
   b. Crossover operation performed between the two parents
   c. Mutation operation is performed this will change the trip from the output of crossover step, if not the children will be the same
   d. Add new children in a new population.
4. Evaluate the fitness of each individual.
5. If the stopping criteria is satisfied, the algorithm stops and shows the best trip, if not it will start over from step 3 and continues the iteration.

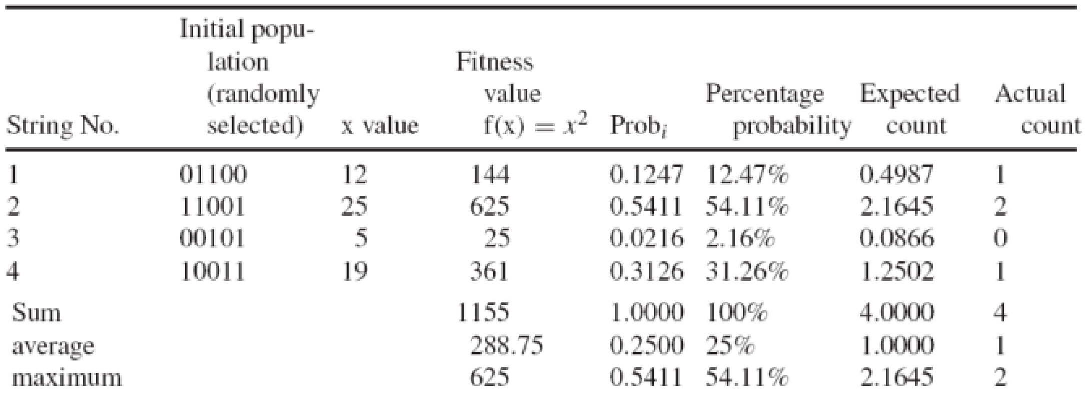

Begin
Initial population
Calculate the fitness value
Selection
Crossover
Mutation
Is termination criteria satisfied?
No
Yes
End

Consider the problem of maximizing the function, f(x) = x^2 where x is permitted to vary between 0 to 31.

Step 1: For using genetic algorithms approach, one must first code the decision variable 'x' into a finite length string. Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained. The objective function here is f(x) = x^2 which is to be maximized.

A single generation of a genetic algorithm is performed here with encoding, selection, crossover and mutation. To start with, select initial population at random.

Here **initial population** of size 4 is chosen, but any number of populations can be elected based on the requirement and application. Table 1 shows an initial population randomly selected.



String No.	Initial population (randomly selected)	x value	Fitness value $f(x) = x^2$	Prob_i	Percentage probability	Expected count	Actual count
1	01100	12	144	0.1247	12.47%	0.4987	1
2	11001	25	625	0.5411	54.11%	2.1645	2
3	00101	5	25	0.0216	2.16%	0.0866	0
4	10011	19	361	0.3126	31.26%	1.2502	1
Sum			1155	1.0000	100%	4.0000	4
average			288.75	0.2500	25%	1.0000	1
maximum			625	0.5411	54.11%	2.1645	2

*Table 1.**Selection***

Step 2: Obtain the decoded x values for the initial population generated. Consider string 1, thus for all the four strings the decoded values are obtained.

Step 3: Calculate the **fitness function**. This is obtained by simply squaring the 'x' value, since the given function is f(x) = x^2. When, x = 12, **the fitness value** is, f(x) = 144 for x = 25, f(x) = 625 and so on, until the entire population is computed

Step 4: Compute **the probability of selection**,

$$\text{Probi} = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i}$$

where n = no of populations f(x)=fitness value corresponding to a particular individual in the population

Σf(x)- Summation of all the fitness value of the entire population.

Considering string 1, Fitness f(x) = 144 Σf(x) = 1155. The probability that string 1 occurs is given by, P1 = 144/1155 = 0.1247. The **percentage probability** is obtained as, 0.1247∗100 = 12.47%.

The same operation is done for all the strings. It should be noted that, summation of probability select is 1.

Step 5: The next step is to calculate the **expected count**, which is calculated as,

$$\text{Expected count} = \frac{f(x)_i}{(\text{Avg}f(x))_i}$$

$$\text{where } (\text{Avg } f(x))_i = \left[\frac{\sum_{i=1}^{n} f(x)_i}{n} \right]$$

For string 1, Expected count = Fitness/Average = 144/288.75 = 0.4987. Computing the expected count for the entire population. The **expected count gives an idea of which population can be selected for further processing in the mating pool**.

Note: Expected count is not mandatory always. Selection can be based on probabilities only.

Step 6: **Now the actual count is to be obtained to select the individuals, which would participate in the crossover cycle using Roulette wheel selection**.

The Roulette wheel is formed as shown in following figure. Roulette wheel is of 100% and the probability of selection as calculated in step 4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the no of occurrences of population is noted to get **actual count**.

String 1 occupies **12.47%,** so there is a chance for it to occur at least once. Hence its actual **count may be 1** since **expected count** for string 1 is **0.4987=0.5(rounded)=1(rounded)** .

With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of being selected twice. Thus its **actual count can be considered as 2 since the expected count is 2.1645=2(rounded).**

On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, it actual count is 0 since its expected count is 0.0866.

String 4 with 31.26% has at least one chance for occurring while Roulette wheel is spun, thus its actual count is 1 since its expected count is 1.2502. The above values of actual count are tabulated as shown is Table 1

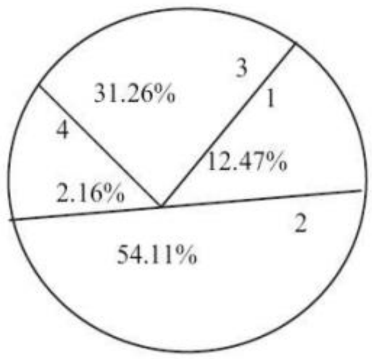
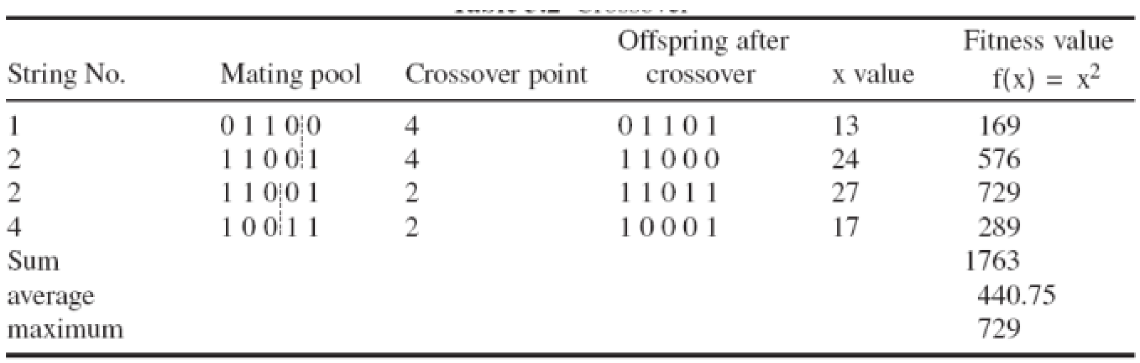


Fig: Roullete Wheel Selection

Step 7: Now, writing the mating pool based upon the actual count as shown in Table 2. The actual count of string no 1 is 1, hence it occurs once in the mating pool. The actual count of string no 2 is 2, hence it occurs twice in the mating pool. **Since the actual count of string no 3 is 0, it does not occur in the mating pool.** Similarly, the actual count of string no 4 being 1, it occurs once in the mating pool. Based on this, the mating pool is formed.



String No.	Mating pool	Crossover point	Offspring after crossover	x value	Fitness value $f(x) = x^2$
1	0 1 1 0 | 0	4	0 1 1 0 1	13	169
2	1 1 0 0 | 1	4	1 1 0 0 0	24	576
2	1 1 0 | 0 1	2	1 1 0 1 1	27	729
4	1 0 0 | 1 1	2	1 0 0 0 1	17	289
Sum					1763
average					440.75
maximum					729

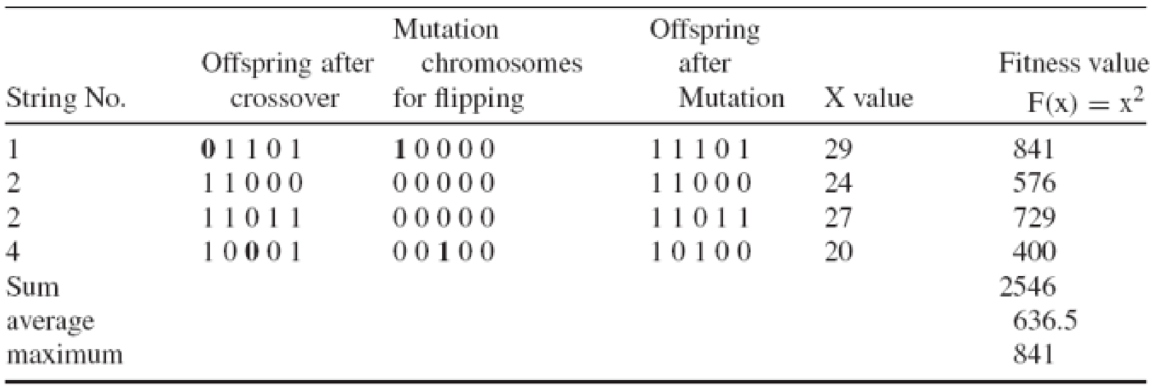
*Table 2.**Crossover***

**Crossover point should be 3 not 2 in above table for the last two strings 2 and 4.*

Step 8: Crossover operation is performed to produce new offspring (children). The crossover point is specified and based on the crossover point, single point crossover is performed and new offspring is produced.

Step 9: After crossover operations, new off springs are produced and 'x' values are decodes and fitness is calculated.

Step 10: In this step, mutation operation is performed to produce new off springs after crossover operation. As discussed in mutation-flipping operation is performed and new off springs are produced. Table 3 shows the new offspring after mutation. Once the off springs are obtained after mutation, they are decoded to x value and find fitness values are computed. This completes one generation. The mutation is performed on a bit-bit by basis.



String No.	Offspring after crossover	Mutation chromosomes for flipping	Offspring after Mutation	X value	Fitness value $F(x) = x^2$
1	0 1 1 0 1	1 0 0 0 0	1 1 1 0 1	29	841
2	1 1 0 0 0	0 0 0 0 0	1 1 0 0 0	24	576
2	1 1 0 1 1	0 0 0 0 0	1 1 0 1 1	27	729
4	1 0 0 0 1	0 0 1 0 0	1 0 1 0 0	20	400
Sum					2546
average					636.5
maximum					841

Table 3. ***Mutation***

Once selection, crossover and mutation are performed, the new population is now ready to be tested. This is performed by decoding the new strings created by the simple genetic algorithm after mutation and calculates the fitness function values from the x values thus decoded.

The results for successive cycles of simulation are shown in Tables 1–3. From the tables, it can be observed how genetic algorithms combine high performance notions to achieve better performance.

In the tables, it can be noted how maximal and average performance has improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 6 841 during same period.

Although random processes make this best solution, its improvement can also be seen successively.

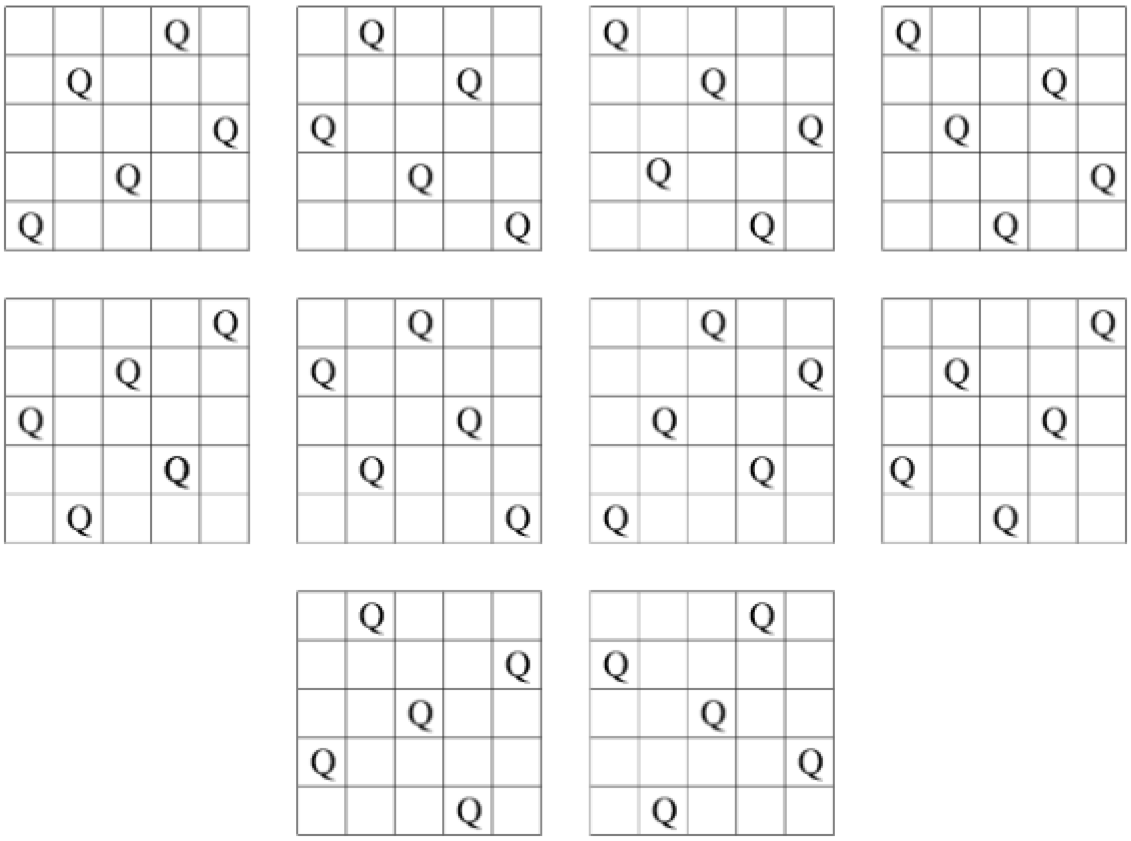
The best string of the initial population (1 1 0 0 1) receives two chances for its existence because of its high, above-average performance.

When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 3.2), one of the resulting strings (1 1 0 1 1) proves to be a very best solution indeed.

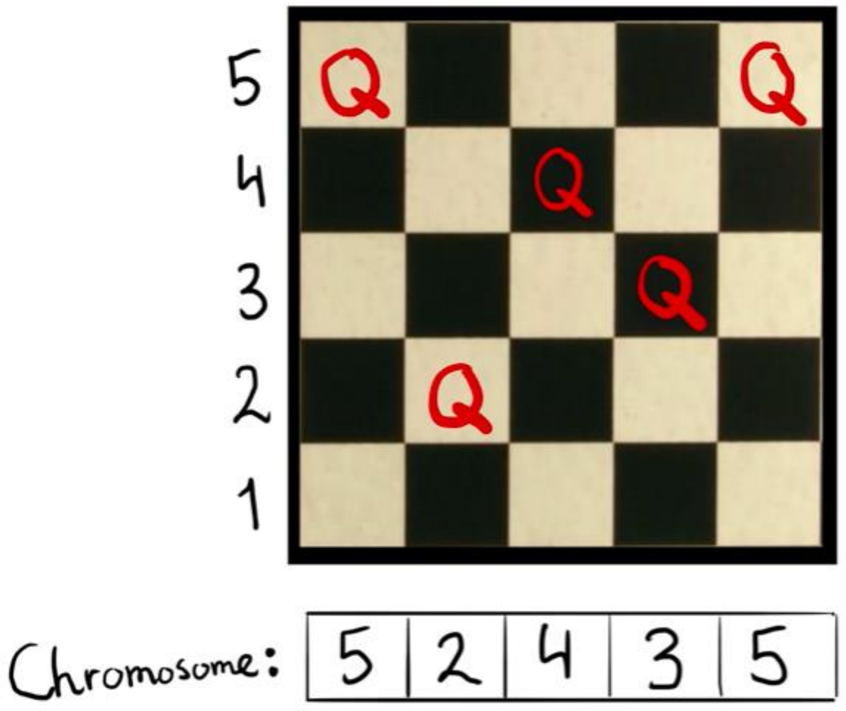
Thus after mutation at random, a new offspring (1 1 1 0 1) is produced which is an excellent choice. This example has shown one generation of a simple genetic algorithm.

Solving the 5-Queens Problem Using Genetic Algorithm

In 5-Queens problem we need to place 5 chess queens on a 5×5 chessboard so that no two queens attack each other. When we find all possible cases, it would look like the following:



Firstly, we need to create a chromosome representation. For showing a chromosome, the best way is to represent it as a list of length N where in our case N=5. The value of each index shows the row of the queen in a column. The value of each index is from 1 to 5.



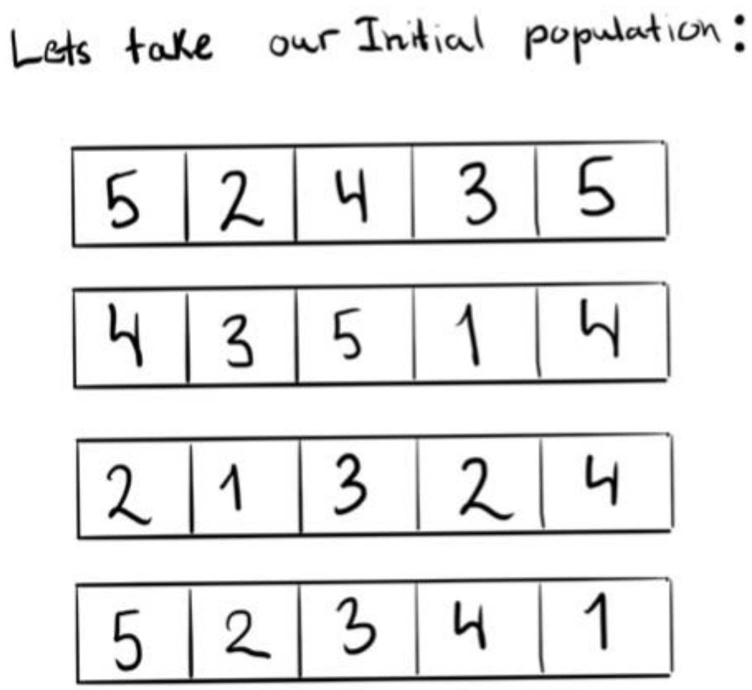
In the initialization process, we need to arrange a random population of chromosomes (potential solutions) are created. Here is the initial population, let us take 4 chromosomes, each of which has a length 5. They are

[5 2 4 3 5]

[4 3 5 1 4]

[2 1 3 2 4]

[5 2 3 4 1]

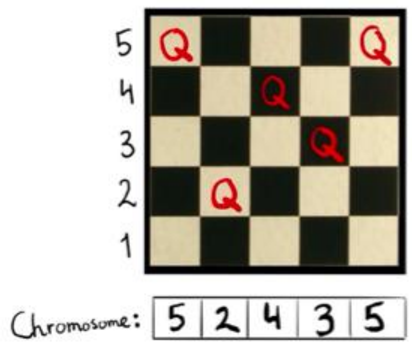


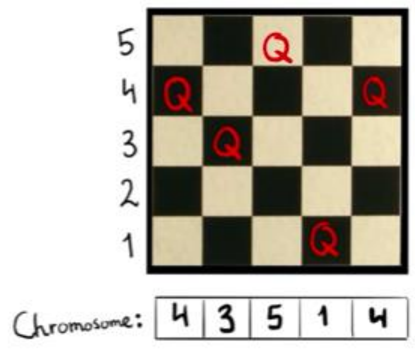
Lets take our Initial population:

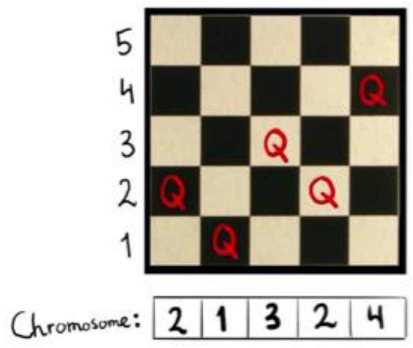
5	2	4	3	5
4	3	5	1	4
2	1	3	2	4
5	2	3	4	1

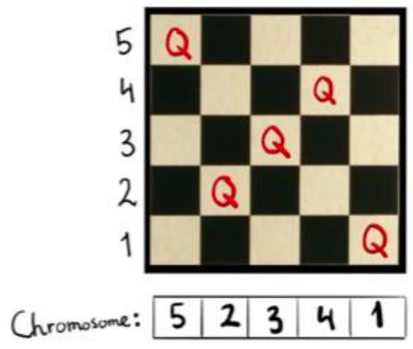
In particular, these chromosomes can be shown as the following on the chessboard:

In particularly, chromosomes represented as the following on board:









First of all, the fitness function is pairs of non-attacking queens. So, higher scores are better is better for us. In order to solve the fitness function for the chromosome [5 2 4 3 5], let us assign each queen uniquely as Q1, Q2, Q3, Q4 and Q5. To find the fitness function value consider following equation:

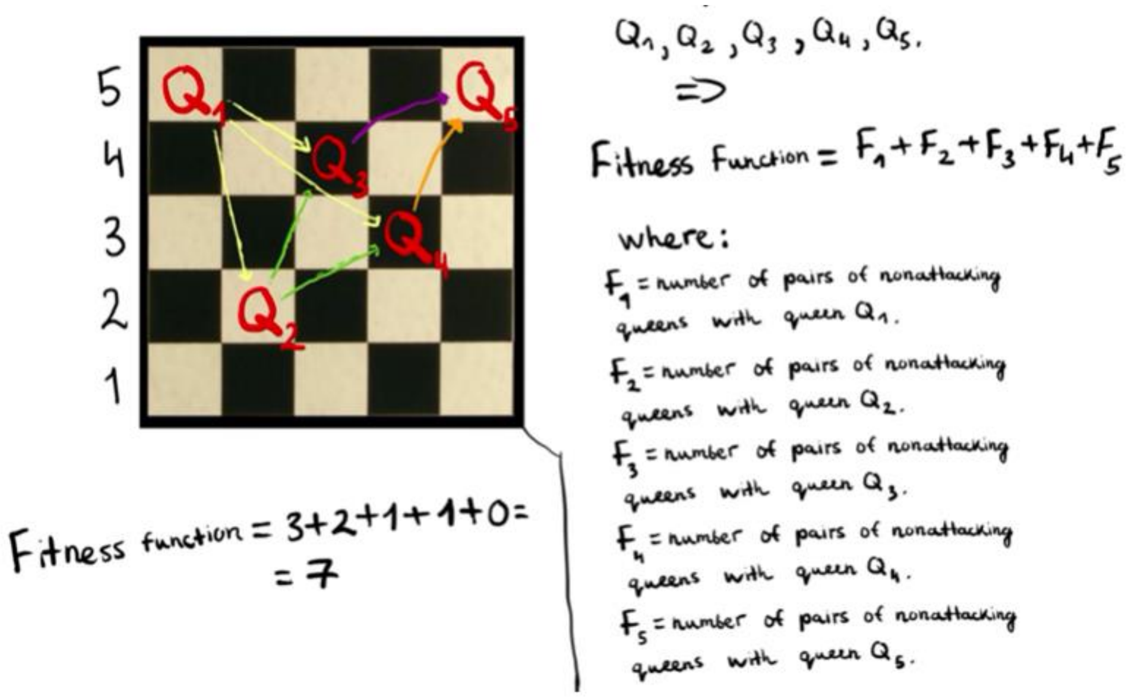
Fitness function = F1+F2+F3+F4+F5

Where:

F1 = number of pairs of non-attacking queens with queen Q1.
F2 = number of pairs of non-attacking queens with queen Q2.
F3 = number of pairs of non-attacking queens with queen Q3.
F4 = number of pairs of non-attacking queens with queen Q4.
F5 = number of pairs of non-attacking queens with queen Q5.

The repetitive non-attacking pairs are discarded. Here for example if we already counted pair Q1 and Q2 to F1, we should not count the same pair Q2 and Q1 to F2.

Thus for chromosome [5 2 4 3 5] the fitness function will be equal to 7.


5
4
3
2
1
Q1
Q2
Q3
Q4
Q5
Fitness function = 3+2+1+1+0= = 7
Q1, Q2, Q3, Q4, Q5. =>
Fitness Function = F1+F2+F3+F4+F5
where:
F1 = number of pairs of nonattacking queens with queen Q1.
F2 = number of pairs of nonattacking queens with queen Q2.
F3 = number of pairs of nonattacking queens with queen Q3.
F4 = number of pairs of nonattacking queens with queen Q4.
F5 = number of pairs of nonattacking queens with queen Q5.

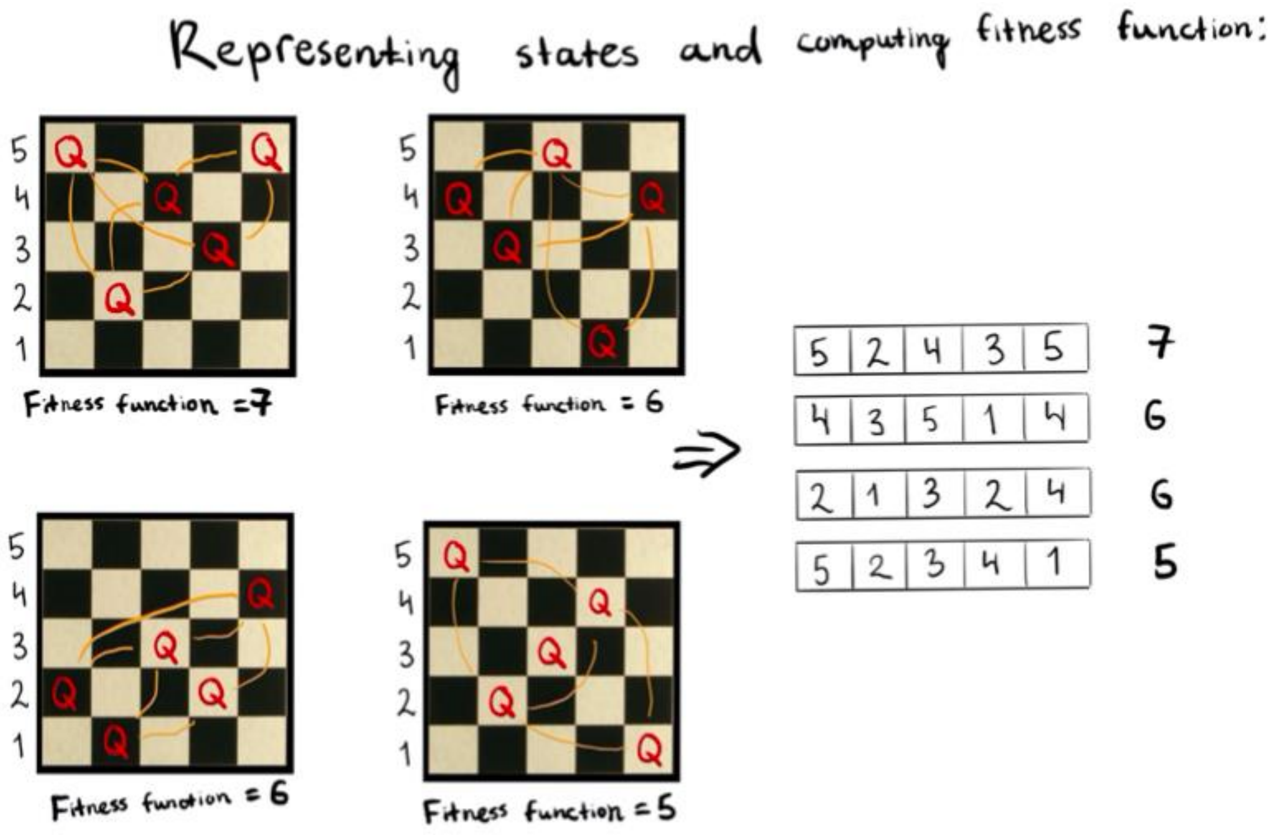
We should evaluate all of our population individuals (chromosomes) using the fitness function. So fitness functions will be the following:

[5 2 4 3 5] fitness function=7

[4 3 5 1 4] fitness function=6

[2 1 3 2 4] fitness function=6

[5 2 3 4 1] fitness function=5



Then we need to compute the probability of being chosen from the fitness function. This will be needed for the next selection step. First, we need to add all fitness functions which will be equal as the following:

7+6+6+5=24

Then we need to compute the probability of being chosen from the fitness function. We need to divide the fitness function by the sum of the fitness function and multiply it to 100%.

[5 2 4 3 5] probability of being chosen = 7/24 *100% = 29%

[4 3 5 1 4] probability of being chosen =6/24 * 100% = 25%

[2 1 3 2 4] probability of being chosen =6/24 * 100% = 25%

[5 2 3 4 1] probability of being chosen =5/24 * 100% = 21%

In the next step, **we randomly choose the two pairs to reproduce based on probabilities which we counted on the previous step.** In other words, a certain number of chromosomes will survive into the next generator using a selection operator. Here selected chromosomes to act as parents that are combined using crossover operator to make children. In addition to this, we pick a crossover point per pair.

Here we took randomly following chromosomes based on their probabilities:

[4 3 5 1 4]

[5 2 4 3 5]

[4 3 5 1 4]

[2 1 3 2 4]

We can notice that we did not take the chromosome [5 2 3 4 1] because its probability of being chosen is the least among chromosomes.

For the first pair

[4 3 5 1 4]

[5 2 4 3 5]

Consider for this pair, the crossover point will be picked after two genes.

In the case of the second pair

[4 3 5 1 4]

[2 1 3 2 4]

Consider for this pair, the crossover point will be picked after three genes.

In the crossover, selected chromosomes act as parents that are combined using crossover operator to make children. In other words, it combines the genetic information of two parents to generate new offspring.

Here we can see that children generated from the first pair ([4 3 5 1 4] and [5 2 4 3 5]) are the following:

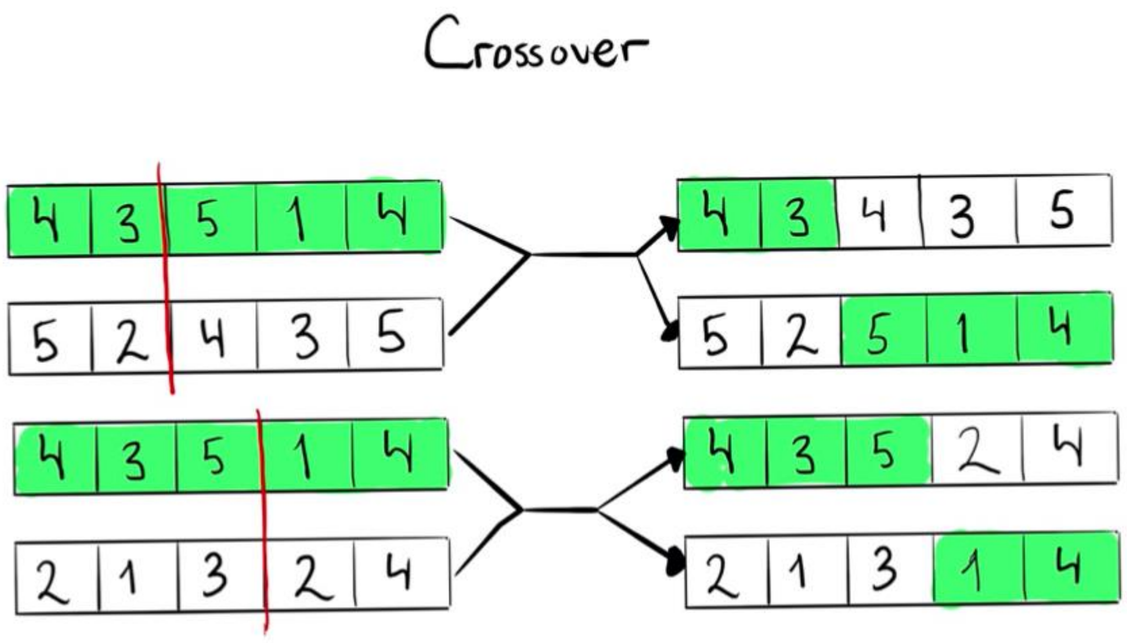
[4 3 4 3 5]

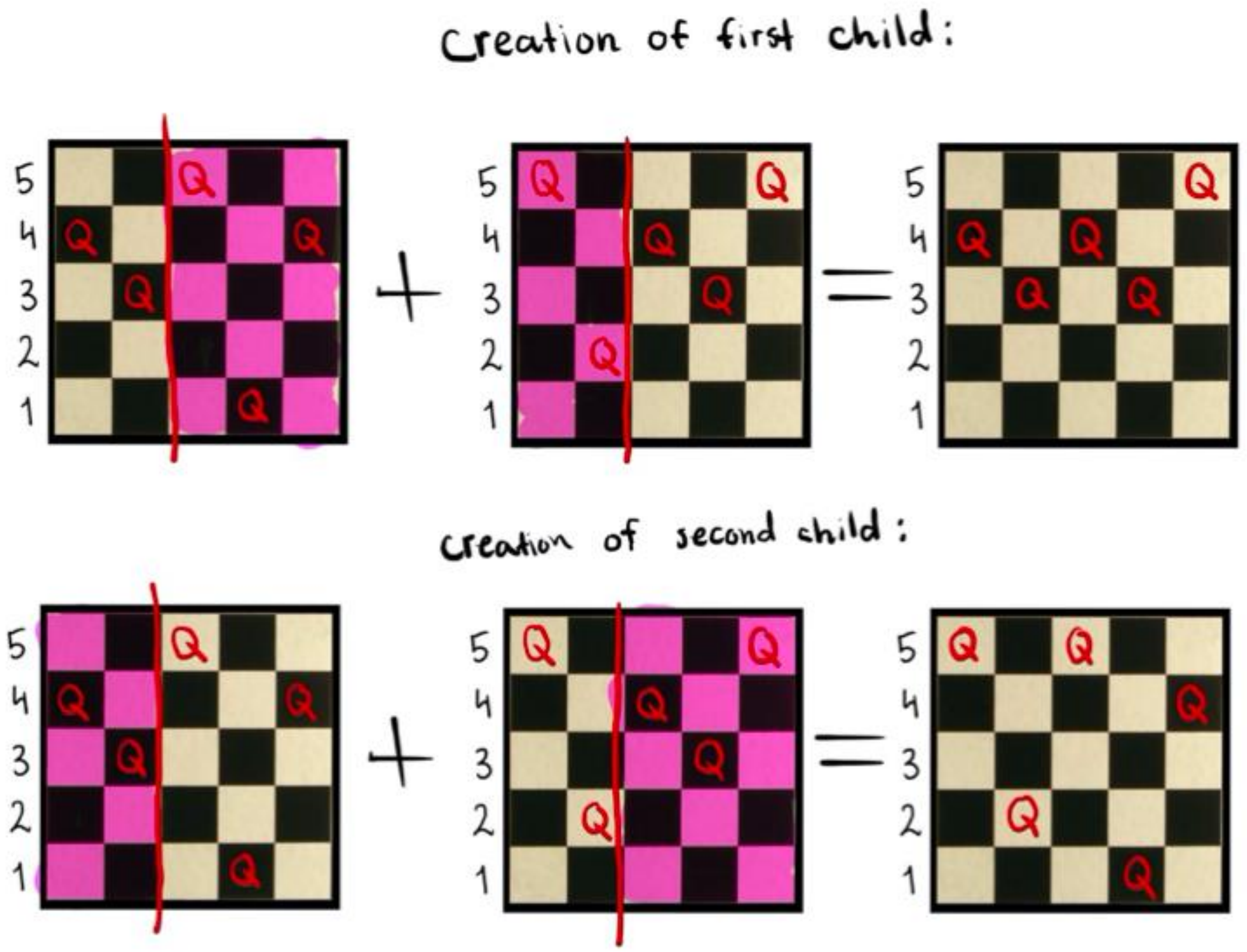
[5 2 5 1 4]

From the second pair ([4 3 5 1 4] and [2 1 3 2 4]) the children are the following:

[4 3 5 2 4]

[2 1 3 1 4]





The next step is mutation. In the mutation process, we alter one or more gene values in chromosomes which we found after crossover. So it randomly changes a few gens and the mutation probability is low. So in our example, our mutation will look like the following:

[4 3 4 3 5] →[4 3 ***1*** *3 5]*

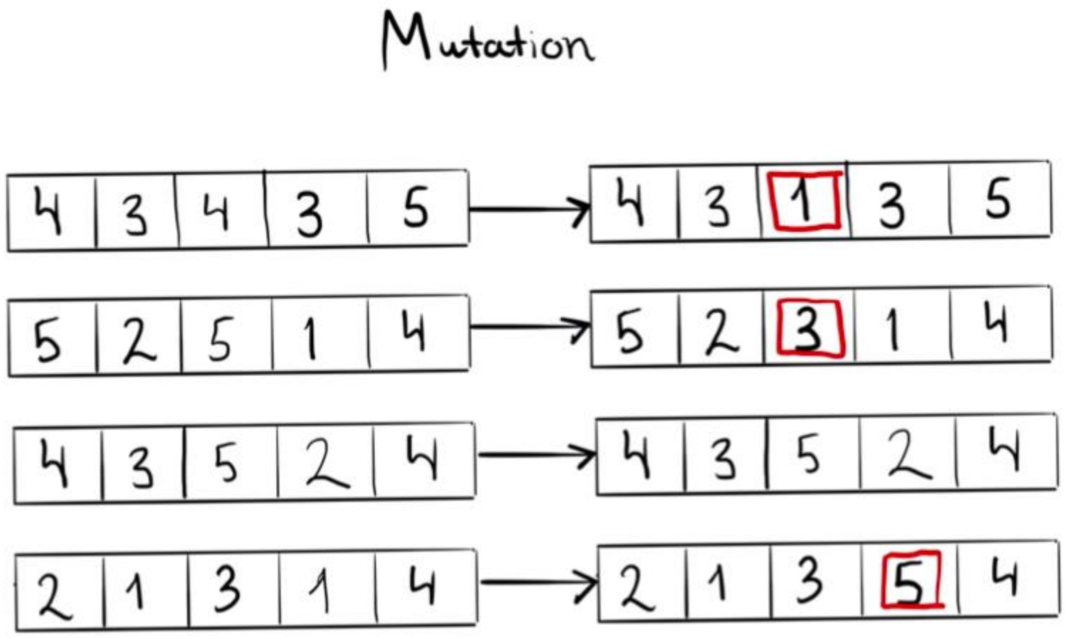
[5 2 5 1 4] →[5 2 ***3*** *1 4]*

[4 3 5 2 4] →[4 3 5 2 4]

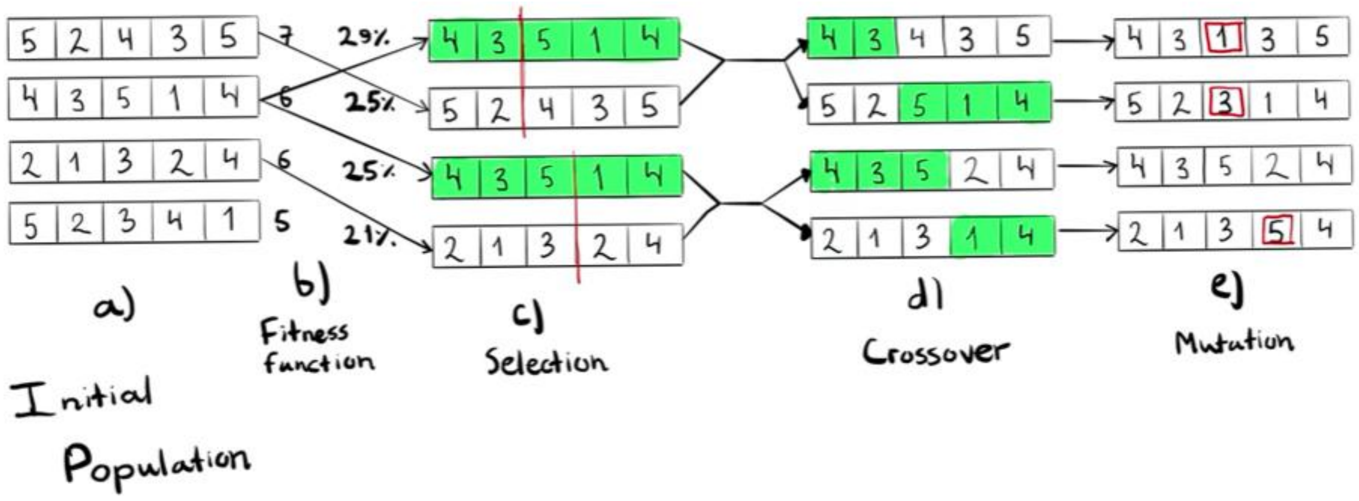
[2 1 3 1 4] →[2 1 3 ***5*** *4]*

where we can notice that the third gene in the chromosome [4 3 4 3 5] changed from 4 to 1.

Also the third gene in the chromosome [5 2 5 1 4] changed from 5 to 3. In addition to this, the fourth gene in the chromosome [2 1 3 1 4] changed from 1 to 5.



So until this, the genetic algorithm to solve the 5-Queen algorithm will look like the following:



In the next step, we need to update the generation. New chromosomes will update the population but the population number will not change. So the chromosomes

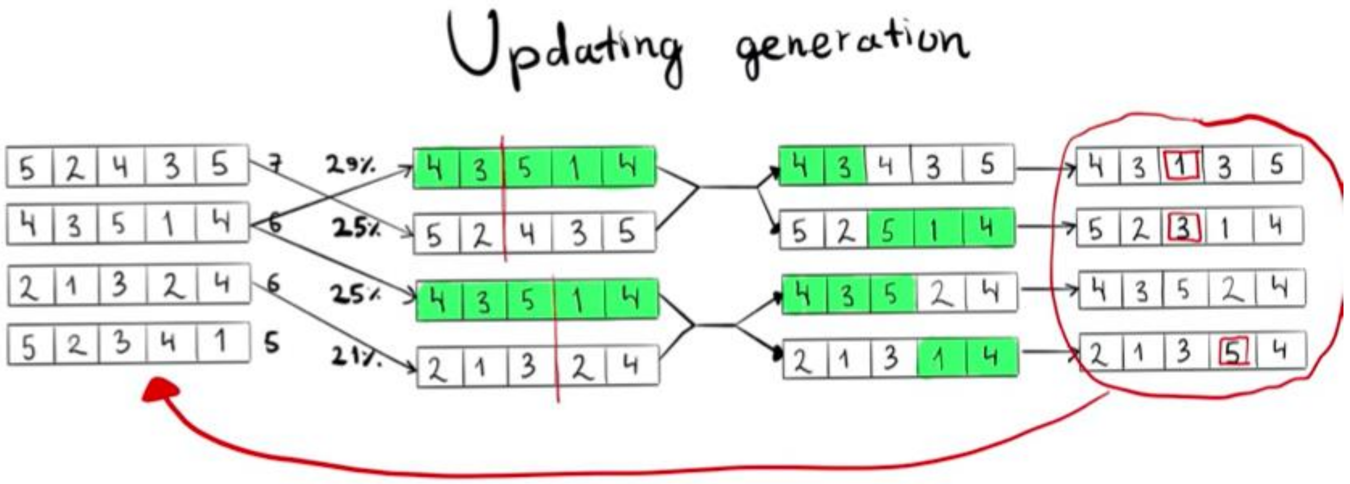
[4 3 1 3 5]

[5 2 3 1 4]

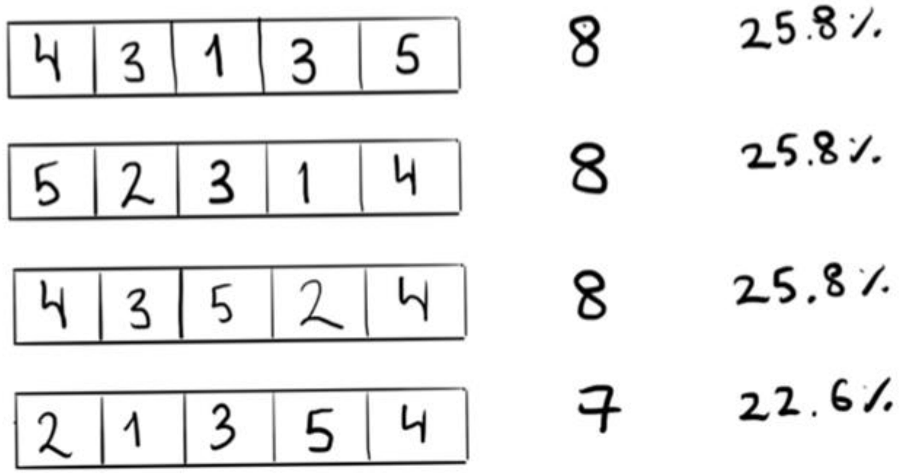
[4 3 5 2 4]

[2 1 3 5 4]

Will be our new population.



So on the next step, we need to undergo fitness evaluation again to find the fitness function of our updated population.



4	3	1	3	5	8	25.8%
5	2	3	1	4	8	25.8%
4	3	5	2	4	8	25.8%
2	1	3	5	4	7	22.6%

The above steps are repeated until chromosome (solution) will satisfy the some termination condition like having *Fitness value to some maximum or to solution convergence.*