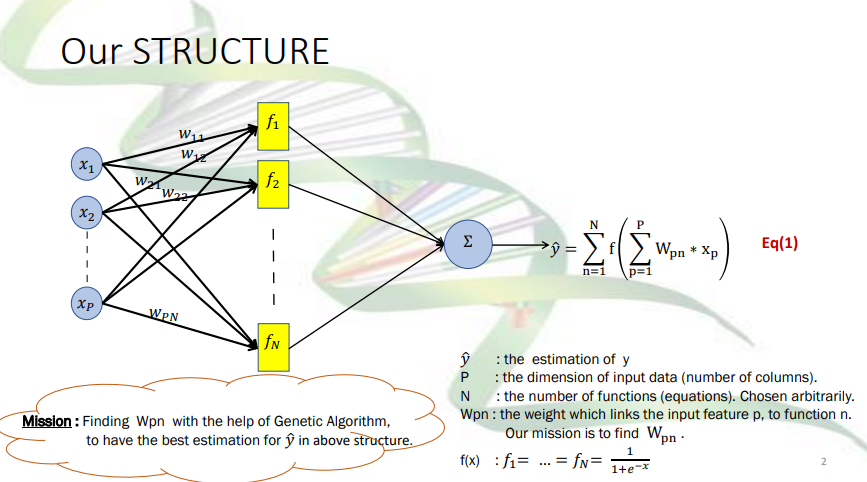
A brief outline of the genetic algorithm is as follows:

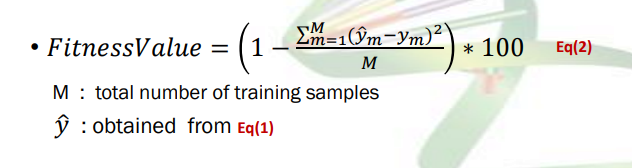
We were provided with a variety of body measurements (in cm), and a corresponding body fat percentage.

The objective is to find a mapping from the measurements (Xi in Eq 1 below) to predict body fat percentage (y in Eq 1 below)

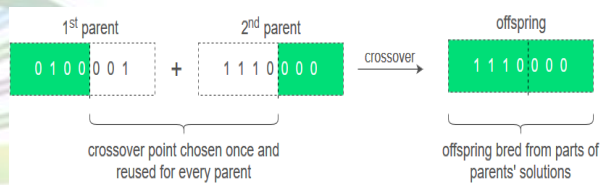
Eq 1) Describes the structure of the mapping we are trying to create by using a genetic algorithm.



Eq 2 Provides an equation to evaluate the fitness of any given mapping (“Chromosome”):



Initially, a population of random mappings is produced. These are standardised and binarised. The mapping with the best fitness score is chosen as the “Stud”. The Stud then “mates'' with every other mapping to create 2 offspring via random single point cross-over (as in the following diagram)



5% of the bits in each mapping were then randomly mutated. Fitness was calculated (via eq 2) for all mappings, and a new stud was chosen. This was repeated until best fitness scores began to plateau.

The following pseudo-code steps were provided to us. For ease of marking, comments in my code mark where all of these steps were carried out.

1 – Import the dataset from the .csv file provided.

2 - Choose N=10 (see STRUCTURE on slide #2) .

3 – Only select the first 5 columns as input, and the very last column as output (target). You can eliminate the rest of the columns.

4 - Choose 25% of the dataset (random) as testing and the rest 75% as training samples. Leave the testing dataset on the side for the time being.

5 – Normalise the training dataset with values between 0 and 1.

6 – Calculate the number of parameters (weights) you need to tune in the STRUCTURE (refer to slide #2). You need to tune PxN parameters (weights). NOTE : For each weight (parameter) you need 10 bits for binarized version of it. Therefore, your long ‘chromosomes’ are having length 10xPxN bits (genes).

7 – Create randomly around Npop=500 (should be large number, so feel free to have more even) initial population of parameters(solutions).

8 – Calculate the fitness\_values via Eq(2) for each solution.

9 – Select the solution with highest fitness\_values (fittest). This is parent now (or you can call it sire).

10 - Binarize the parent and all other population according to the following procedure : I ) For each single parameter (weight) - it should be a figure between -1 and 1 , normalise the weights to numbers between 0 and 1 II ) Multiply the normalised figures to 1000. Your figures are now float numbers less than 1000. III ) Round the numbers to closest integer. Now you have integer numbers less than 1000. IV ) Get the base-2 (binary) 10-bit conversion of the weights. NOTE : Make sure for each binary weight you have fix 10 bits allocated.

11 – Concatenate all 10-bit weights along each other and make a ‘chromosome’ . NOTE : Please remember the order you align the weights in the chromosome , because later you need to de-segment the chromosome and put each weight in its own place in the STRUCTURE (refer to slide #2), to produce 𝑦ො .

12– Do the Cross-Over of the parent, with each single member of Npop and create two offsprings from each. Now your population is increased by 2xNpop

13– Do the mutation for each newly born chromosome.

14– Do the de-binarization of the chromosomes according to following procedure : I ) De-segment each chromosome to its 10-bits components. II ) Make a binary to decimal conversion of each single 10-bit weight. III ) Divide them by 1000 IV ) De-normalise weights to values between -1 and 1

15 – Calculate the fitness\_value for all population from Eq(2).

16 – Eliminate the lowest fitness\_value chromosomes. Now the population is reduced back from 2xNpop to Npop again.

17 – Save the chromosome with highest fitness\_value as the parent. If by any chance the highest fitness\_value was less than previous iteration, keep previous iteration parent as current parent.

18 – Iterate from step 12 to 17. Each time you do steps 12 -17, one iteration is elapsed. You iterate until the highest fitness\_value reaches to a plateau (like a ‘while’ loop)