**Heuristics CA1 – Using a Genetic Algorithm evolve sets of weights for a Neural Network**

*By Conor Fahey, Alex Jevdokimovs*

Table of Contents

[1. Description of the Problem (with Feasible Solutions) 2](#_Toc2695672)

[Data Normalization & Pre-Processing 3](#_Toc2695673)

[2. Justification of using Artificial Neural Network 4](#_Toc2695674)

[3. Genetic Algorithm Selection and Breeding Process 5](#_Toc2695675)

[Encoding 5](#_Toc2695676)

[Selection 6](#_Toc2695677)

[Crossover 6](#_Toc2695678)

[Mutation 7](#_Toc2695679)

[Fitness Function 8](#_Toc2695680)

[4. Neural Network 9](#_Toc2695681)

[Neural Network Architecture 9](#_Toc2695682)

[Example NN Generation 10](#_Toc2695683)

[Round 1 (1st set of inputs) 10](#_Toc2695684)

[Process 11](#_Toc2695685)

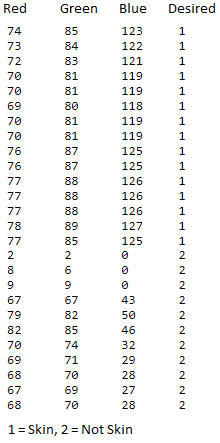
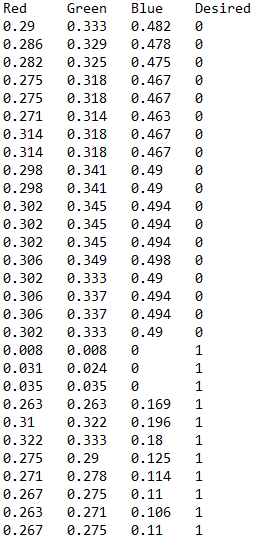
[5. Altered Parameters 12](#_Toc2695686)

[6. References 13](#_Toc2695687)

# 1. Description of the Problem (with Feasible Solutions)

The problem we are trying to solve is whether a set of RGB values is “skin colored”. Our focus is to use the Genetic Algorithm to find a fit set of weights for our Neural Network. We retrieved our test data from UCI in the Machine Learning Repository **[1]**.

Each entry is made up of 3-columns - R G B and, Desired output (1 skin, 2 not skin, [Re-arranged: 0 skin, 1 not skin]). RGB is given as a value between 0 and 255.

*Example of Data Set*

## Data Normalization & Pre-Processing

When training the network, the ideal fitness we’re striving for is a fitness of 0 where 0 indicates that all provided desired outputs have matched against the output calculated from the neural network. As a result, the larger the error of the neural network, the less fit the chromosome will be.

Our first step will be to normalize the data.

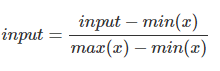
**Original Inputs:**

**Red: 74**

**Green: 85**

**Blue: 123**

**Desired Outcome: 1**



74 = (74 – 0 / 255 – 0)

74 = (74/255)

74 = 0.290 85 = 0.333 123 = 0.482

**New Normalized Inputs:**

**Red: 0.290**

**Green: 0.333**

**Blue: 0.482**

The inputs will be normalized using the method; *input = (input − min(x)) / (max(x) − min(x))*, which will put each value in the range of 0 – 1. As the possible outputs are either skin or not skin, it seems immediately appealing to use 0 or 1 for our output.

# 2. Justification of using Artificial Neural Network

Our problem is a classification/pattern recognition problem. This means an artificial network would be considered as a go to method, as they are used mostly when dealing with these types of problems. We want to solve the problem of pattern matching so that the program can identify if an input of RGB values from the dataset is skin or not skin.

It cannot be solved using conventional methods, i.e. procedural instruction execution, because the steps required to solve it are not known to the computer beforehand, therefore neural networks can be used as they offer a different approach of solving problems.

**Calculating error:** Use one of Heavy-Side/**Sigmoid** (Logistics Function)/ReLU activation functions.

Assumption that our dataset is not linearly separable.

# 3. Genetic Algorithm Selection and Breeding Process

## Encoding

**Value encoding** will be used instead of binary encoding.

Binary encoding would be difficult as we will be using float values.

Chromosomes will be a **combination of weights** for the overall neural network architecture.

Weights B Weights A

{0.231, -0.543, 0.998, 0.223, 0.521, -0.175}

**Variable:** Number of Parents

Too many: Persistent bad traits

Too few: stagnant gene pool

**Example:**

* Total population consists of 5 by default
* 2 elite members will be selected for the next generation at all times.
* Will always require 3 offspring to be created which will be achieved from 6 selected parents in total.

|  |  |
| --- | --- |
| Parents | Fitness |
| {0.231, -0.543, 0.998, 0.223, 0.521, -0.175} | 0.645 |
| {0.856, -0.354, -0.576, -0.321, 0.412, 0.112} | 0.332 |
| {0.654, 0.337, -0.576, -0.889, 0.311, 0.856} | 0.745 |
| {0.050, 0.199, -0.644, 0.707, 0.256, -0.555} | 0.375 |
| {0.878, -0.577, 0.153, 0.122, -0.798, -0.334} | 0.875 |

**Sort by Fitness – Higher number = better fitness**

|  |  |
| --- | --- |
| Parents | Fitness |
| {0.878, -0.577, 0.153, 0.122, -0.798, -0.334} | 0.875 |
| {0.654, 0.337, -0.576, -0.889, 0.311, 0.856} | 0.745 |
| {0.231, -0.543, 0.998, 0.223, 0.521, -0.175} | 0.645 |
| {0.050, 0.199, -0.644, 0.707, 0.256, -0.555} | 0.375 |
| {0.856, -0.354, -0.576, -0.321, 0.412, 0.112} | 0.332 |

## Selection

**Roulette Wheel**

|  |  |  |  |
| --- | --- | --- | --- |
| Parents | 1 - Fitness |  |  |
| {0.878, -0.577, 0.153, 0.122, -0.798, -0.334} | 1 - 0.875 =0.125 | 0.125 | 0.125/1.968 = 0.064 |
| {0.654, 0.337, -0.576, -0.889, 0.311, 0.856} | 1 -0.745 = 0.225 | 0.35 | 0.35/1.968 = 0.178 |
| {0.231, -0.543, 0.998, 0.223, 0.521, -0.175} | 1 -0.645 = 0.325 | 0.675 | 0.675/1.968 = 0.343 |
| {0.050, 0.199, -0.644, 0.707, 0.256, -0.555} | 1 -0.375 = 0.625 | 1.3 | 1.3 / 1.968 = 0.661 |
| {0.856, -0.354, -0.576, -0.321, 0.412, 0.112} | 1 -0.332 = 0.668 | 1.968 | 1.968/1.968 = 1 |

**Variable:** Offspring

More children = faster evolution

6 random values for 3 offspring

Random Values: {0.023, 0.122, 0.354, 0.665, 0.644, 0.8}

**Parents 1 2 4 5 4 5**

**Parent combos: {1,2} {4,5} {4,5}**

## Crossover

Chromosome 1: {0.878, -0.577, 0.153, 0.122, -0.798, -0.334}

Chromosome 2: {0.654, 0.337, -0.576, -0.889, 0.311, 0.856}

Child: {0.654, 0.337, 0.153, 0.122, 0.311, 0.856}

## Mutation

Mutation chance + mutation factor to random number of genes in the chromosome.

A number randomizer between 0 and 1 decided of two types of mutation

0 we take the mutation factor and add it to the selected location in the chromosome

1 we do the opposite and subtract.

**Variables**

Type of mutation (Add/subtract)[0, 1] = {**0**, 0, 1,0,1}

Mutation value[0.001, 0.010] = {**0.005**, 0.003, 0.009, 0.01, 0.001}

Mutation chance [0.000 - 1.000]: {0.376, 0.567, 0.674, 0.123, 0.231, 0.576}

Mutation gene [1 - 6]: {**2**,1,1,3,5,4}

How many mutations for one child?

Child: {0.654, 0.337, 0.153, 0.122, 0.311, 0.856}

After Mutation

Child: {0.654, 0.337, 0.158, 0.122, 0.311, 0.856}

**2 Selected Elite added:**

|  |  |
| --- | --- |
| Parents | Fitness |
| {0.878, -0.577, 0.153, 0.122, -0.798, -0.334} | 0.875 |
| {0.654, 0.337, -0.576, -0.889, 0.311, 0.856} | 0.745 |
| {0.654, 0.337, 0.158, 0.122, 0.311, 0.856} | Back to Neural Network |
|  |  |
|  |  |

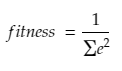
## Fitness Function

Each chromosome, built of encoded weights, make up our population. Each chromosome needs to be assigned a fitness. We decode each chromosome in turn to a set of weights, which then pass through our fitness function. The fitness function is a feed forward through the neural network, the maximum value from the outputs is rounded up to 1 while all others are rounded down to 0, errors are then calculated by subtracting these outputs from the desired outputs.



*e – error | d - desired output |o - outcome*

Now we have errors which are then squared and added together to give the total error of this input output set. This is done for all the input output sets and total errors are added together, 1 / (sum of total errors) is the fitness we give to this chromosome (set of weights).



# 4. Neural Network

To obtain the fitness of the respective weights that have been chosen, we have to run the inputs & weights through a neural network which in result will return an output.

To determine the error, we subtract the desired outcome from our output of the neural network as stated in ***Fitness Function*** above.

## Neural Network Architecture



## 

## Example NN Generation

|  |  |  |  |
| --- | --- | --- | --- |
| **Example Inputs (Un-normalized)** | **Example Inputs (Normalized) (1x3)** | **Example Weight B (4x4)** | **Example Weight A (5x1)** |
| |  |  |  |  | | --- | --- | --- | --- | | 74 | 85 | 123 | 1 | | 77 | 88 | 126 | 1 | | 68 | 70 | 28 | 2 | | 2 | 2 | 0 | 2 | | |  |  |  |  | | --- | --- | --- | --- | | 0.290 | 0.333 | 0.482 | 0 | | 0.302 | 0.345 | 0.494 | 0 | | 0.266 | 0.275 | 0.110 | 1 | | 0.008 | 0.008 | 0 | 1 | | |  |  |  |  | | --- | --- | --- | --- | | -0.5 | 0 | -0.1 | -0.6 | | 0.3 | 0.9 | 0.1 | 0.1 | | -0.7 | -0.1 | 0.3 | 0.2 | | 0.6 | 0.1 | 0.4 | -0.4 | | |  | | --- | | 0.3 | | -0.6 | | -0.7 | | 0.3 | | 0.9 | |

### 

### Round 1 (1st set of inputs)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x\* | x | xB | h\* = F(xB) | h | hA | O = F(hA) | d | e | E |
| [0.290 0.333 0.482] | [1 0.290 0.333 0.482] | [-0.357 0.276 0.222 -0.697] | [0.412 0.569 0.555 0.332] | [1 0.412 0.569 0.555 0.332] | [0.120] | 0.530 | 0 | -0.530 | 0.281 |
| [0.302 0.345 0.494] | [1 0.302 0.345 0.494] | [-0.355 0.287 0.231 -0.698] | [0.412 0.571 0.557 0.332] | [1 0.412 0.571 0.557 0.332] | [0.119] | 0.530 | 0 | -0.530 | 0.281 |
| [0.266 0.276 0.110] | [1 0.266 0.276 0.110] | [-0.547 0.223 0.053 0.562] | [0.367 0.556 0.513 0.637] | [1 0.367 0.556 0.513 0.637] | [0.418] | 0.603 | 1 | 0.397 | 0.158 |
| [0.008 0.008 0] | [1 0.008 0.008 0] | [-0.503 0.006 0.100 -0.600] | [0.380 0.501 0.525 0.354] | [1 0.380 0.501 0.525 0.354] | [0.200] | 0.550 | 1 | 0.450 | 0.203 |

### Process

**Example – First Input Set:**

**xB** --> *(Convert x into inputs by padding first row with 1)* **[1x4 matrix = 1x4 • 4x4]**

1. (1)(-0.5) + (0.290)(0.3) + (0.333)(-0.7) + (0.482)(0.6) = -0.3569
2. (1)(0) + (0.290)(0.9) + (0.333)(-0.1) + (0.482)(0.1) = 0.2759
3. (1)(-0.1) + (0.290)(0.1) + (0.333)(0.3) + (0.482)(0.4) = 0.2217
4. (1)(-0.6) + (0.290)(0.1) + (0.333)(0.2) + (0.482)(-0.4) = -0.6972

**h\*** --> **f(xB)**

**f(xB) -->** 1/1+e^-x [Sigmoid Activation Function]

1. 1/1+e^-(-0.357) = 0.412
2. 1/1+e^-(0.276) = 0.509
3. 1/1+e^-(0.222) = 0.555
4. 1/1+e^-(-0.697) = 0.332

**hA**

* (1)(0.3)+(0.412)(-0.6)+(0.569)(-0.7)+(0.555)(0.3)+(0.332)(0.9) = 0.1198 🡪 0.120

**Output (o)** --> **f(hA)**

**f(hA) -->** 1/1+e^-x [Sigmoid Activation Function]

* 1/1+e^-(0.120) = 0.529964… --> 0.530

**Error (e)** --> **e = d – o**

* 0 – 0.530 = -0.530

**Overall Error (E)** --> e^2

* (-0.530) ^ 2 = 0.2809 --> 0.281

# 5. Altered Parameters

Weights – number of nodes

Population

// “Roulette Wheel” parent values

Stochastic Universal Sampling

// Rank Selection

Elite member count

Dropped Chromosomes

Mutation rate/chance

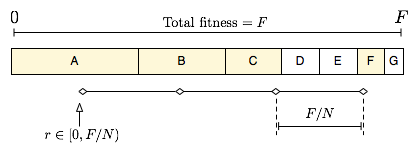
Number of generations

# 6. References

**[1]** - **UCI Skin Segmentation Data Set** - *https://archive.ics.uci.edu/ml/machine-learning-databases/00229/Skin\_NonSkin.txt*

**Selection:**

* *Stochastic Universal Sampling (SUS)*selection.
* SUS is a zero bias and minimum spread. Ensuring that selection process is most fair to those weights that are fitter.



SUS(Population, N)

F := total fitness of Population

N := number of offspring to keep

P := distance between the pointers (F/N)

Start := random number between 0 and P

Pointers := [Start + i\*P | i in [0..(N-1)]]

return RWS(Population,Pointers)

RWS(Population, Points) Keep = []

for P in Points i := 0

while fitness sum of Population[0..i] < P i++

add Population[i] to Keep

return Keep

F = 0.332 + 0.375 + 0.645 + 0.745 + 0.875 = 2.972

N = 3

P =

Start =

Pointers = [Start +i\*P | i in [0 . . (3 - 1)]]

Intermediate recombination

Mean of 2 values is new value for offspring

(ValA + ValB) / 2 = new value

Chromosome 1: {0.878, -0.577, 0.153, 0.122, -0.798, -0.334}

Chromosome 2: {0.654, 0.337, -0.576, -0.889, 0.311, 0.856}

Child : {0.766, -0.12, -0.212, -0.384, -0.244, 0.261}