**Application Of Neural Networks To Robot Animals**

**Final Project CS547**

Matthew A. Letter

&

Lin sun

University of New Mexico

Neural Networks 547

mletter1@unm.edu

sun@unm.edu

Table of Contents

Abstract 3

1. Introduction 3

2. Approach 3

2.1 Architecture 0: No movement, measure lifetime 3

2.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food 3

2.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food 4

2.4 Architecture 3: Neuronal Classification of food using Food RGB Values 4

2.5 Architecture 4: Neuronal Movement towards brightest object 4

2.6 Architecture 5: Move robot based on intensity and eat based on classification 6

2.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage 6

2.8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage 7

2.9 Architecture 8: Classify objects based on visual lights and acoustic sounds: 7

3. Results 7

3.1 Architecture 0: No movement, measure lifetime 7

3.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food 7

3.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food 9

3.4 Architecture 3: Neuronal Classification of food 10

3.5 Architecture 4: Neuronal Movement towards brightest object 11

3.6 Architecture 5: Eat objects based on neuronal classification 12

4. Discussion 14

5. Summaries and Conclusion 14

6. Acknowledgements 14

7. References 14

8. Appendix A 15

# Abstract

Add abstract here, single space, 12pt intentionally blank

# 1. Introduction

This paper rigorously establishes that neural network design models can be used to manipulate learning on simulated biological system. All the research was done using professor Thomas Caudell’s animal robot environmental model. (Caudell) The starting point of the research involved 3 architectures, used to establish the parameters of life with respect to the simulated organism and its environment. These provide a basis for analyzing the implemented neural network algorithms, the goal of which is to create a “neuron” based brain for the robot organism to live as long as possible in its environment, and determine what neuronal design structures degrade or enhance this attribute.

# 2. Approach

### 2.1 Architecture 0: No movement, measure lifetime

For the first architecture, we will test the basic metabolic consumption for the robot. The robot’s speed was set to 0; therefore the robot was programmed to do nothing until it dies. Finally the lifespan was measured for analysis.

### 2.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food

This test is designed to explore the effect of the robots speed variation on its run lifespan and varying the robots direction correlated to lifespan. The goal of this experiment is to determine a correlation between the speed of the robot and its metabolism. Furthermore, the direction change should yield insight into the world around the robot. The robot will start at the same position where the direction and speed will be varied. To try test different directions, we will turn the neuron head to 5 degree counter-clockwise each time. When the neuron is moving, it will always head to the brightest object in the space, no matter if the food is poisonous or not. The robot will be set to eat nothing in the world.

### 2.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food

This test is designed to explore the effect eating on robot’s lifetime. This is different from Architecture 1 in that neuron will eat all of the food it encounters. When the robot is moving, it will always head to the brightest object in the space, no matter what type of object it is. To get a well-distributed random speed, the initial robot head angle will be changed for each run, but the initial position will be held constant at the point of origination.

### 2.4 Architecture 3: Neuronal Classification of food using Food RGB Values

This test will check classification of food as either good or bad based on whether the food could increase or decrease energy of the robot using LMS neuron architecture. If the food increases the energy of the robot, the food is classified as good; otherwise, we say that the food is bad. For a simple starting neural architecture an LMS neuron to classify the food. In order to classify the objects, we should know the weights for classification neuron. Using the objects eaten in the world, training of the neuron will be conducted.



Figure 1: Network diagram of the LMS neuron for classifying food

### 2.5 Architecture 4: Neuronal Movement towards brightest object

The goal of this architecture is to move towards the brightest object using a winner takes all neuronal networks implementation. Movement is the next logical step after classification of food since we have to have a way of moving toward objects. In order to give the robot the ability to seek out objects a neuron will be added that will make a direction choice.

The neuron will choose the direction based on the incoming light intensity. It is known that the light intensity is inversely related to the distance between the robot and an object. Meaning that objects having higher light intensity are closer to the robot than other objects with lower light intensity. In the simulation world, the robot wants to survive for as long as possible. Therefore, it should try to move towards the objects in the world closer to it. Equation 1.1 shows the equation for deciding which direction to move towards. Based on the thought process above, a neuron like the one in Fig 2 will be tested. The inputs for the input are RGB lights waves from 31 directions, while the output is the angle of the lightest object with respect to the horizontal line. So the direction that the neuron will change towards is the direction of the robot automatically based on light intensity.

(eq 1.1)



Figure 2: This is a three layer neural network where the RGB circles denote the intensity signal bands being received by the first layer. The second layer is a winner-take all network which selects the brightest cone. The third layer is a neuron, which computes the angle change for movement

### 2.6 Architecture 5: Move robot based on intensity and eat based on classification

Architecture 5 is very similar to architecture 3 except that we are normalizing the RGB input vectors to solve the problem of intensity that is given off from objects at different distances. The goal of this neuron is to classify objects without bias on the distance.

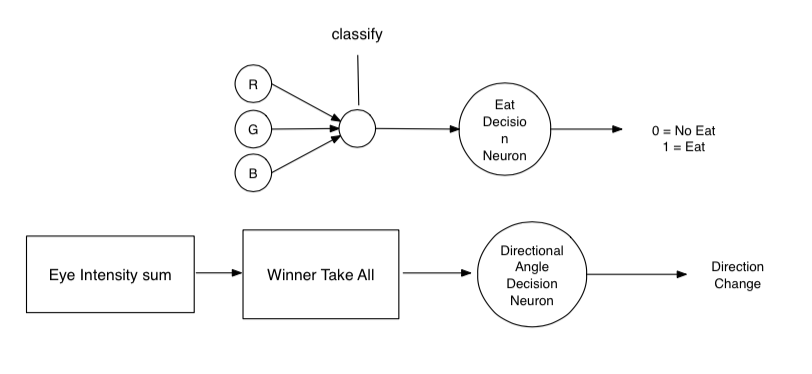


Figure 3: Network diagram of the LMS neuron for classifying food with the caveat of normalized inputs

### 2.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage

In architecture 3, we use RGB values to classify objects. However this does not address a certain problem subset, as robot moves farther away from an object, the light value for said object is decreased uniformly over each signal band. The un-normalized input of RGB signals cannot be trusted to classify objects, due to this fact. When the robot moves, it will pick the direction with the highest intensity light value based off equation 1.1. It is possible that the direction picked will lead the robot to poisonous objects; this is because direction is currently only based on the intensity of light not what type of object is producing the light. Therefore, we need to implement a direction neuron, which can move towards the direction with the closest nonpoisonous object. To solve this problem a neuronal network will be created that takes normalized RGB values and classifies these values as either good or bad one caveat is that neutral objects will be classified as bad. The goal of architecture 7 is to see if a normalized RGB value can be used to generate better direction selection.

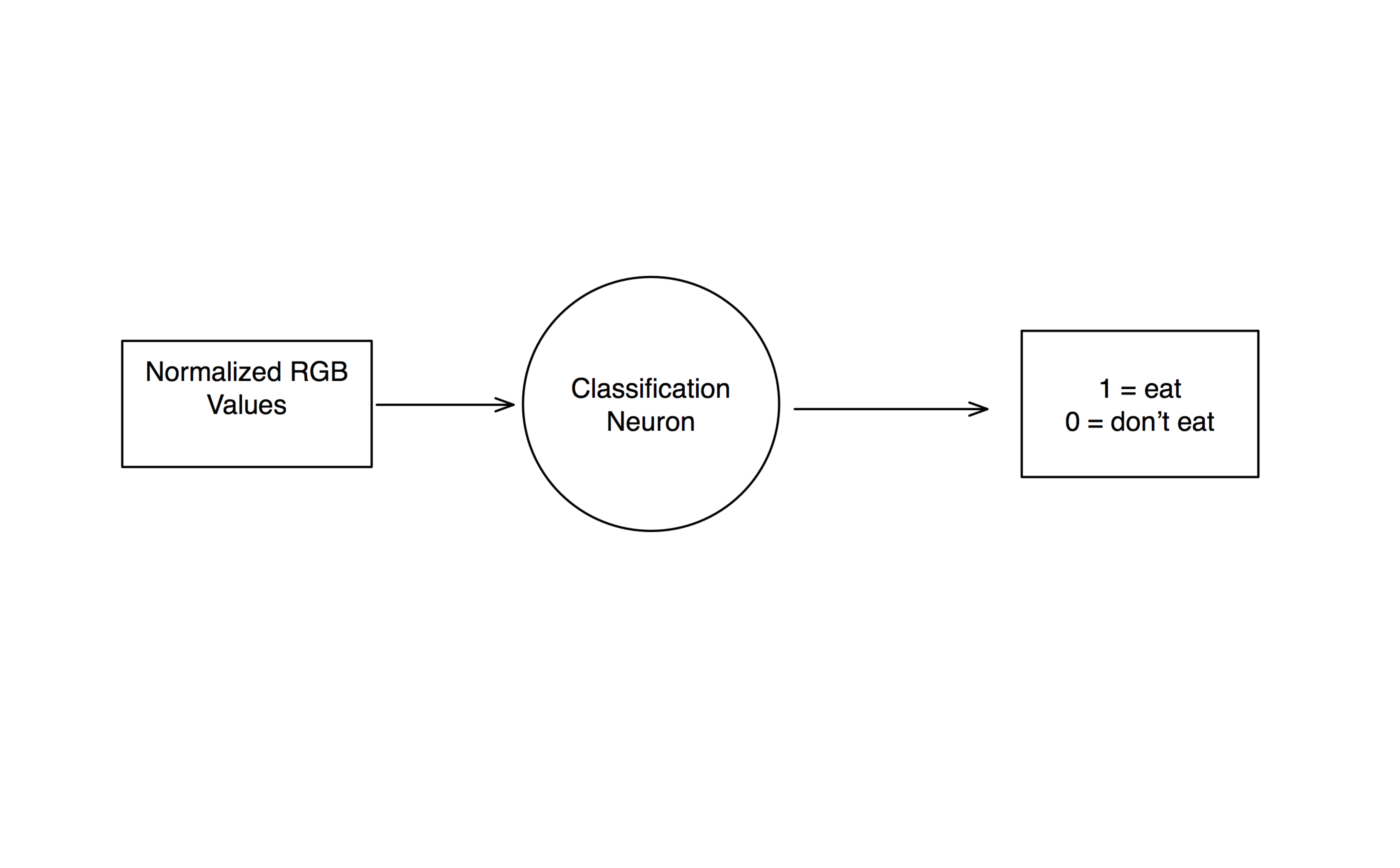


Figure 4: Representation of the normalized classification approach

### 2.8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage

In architecture 6, we used normalized RGB value to classify objects. Now the neuronal architecture will choose the direction based on both intensity and whether an object is classified as poisonous, with the goal being to choose the most intense good object direction. In our direction neuron, it will take into consideration the output of LMS neuron. If the LMS neuron classifies the direction as containing a poisonous then the next highest intensity direction neuron will be chosen for classification.

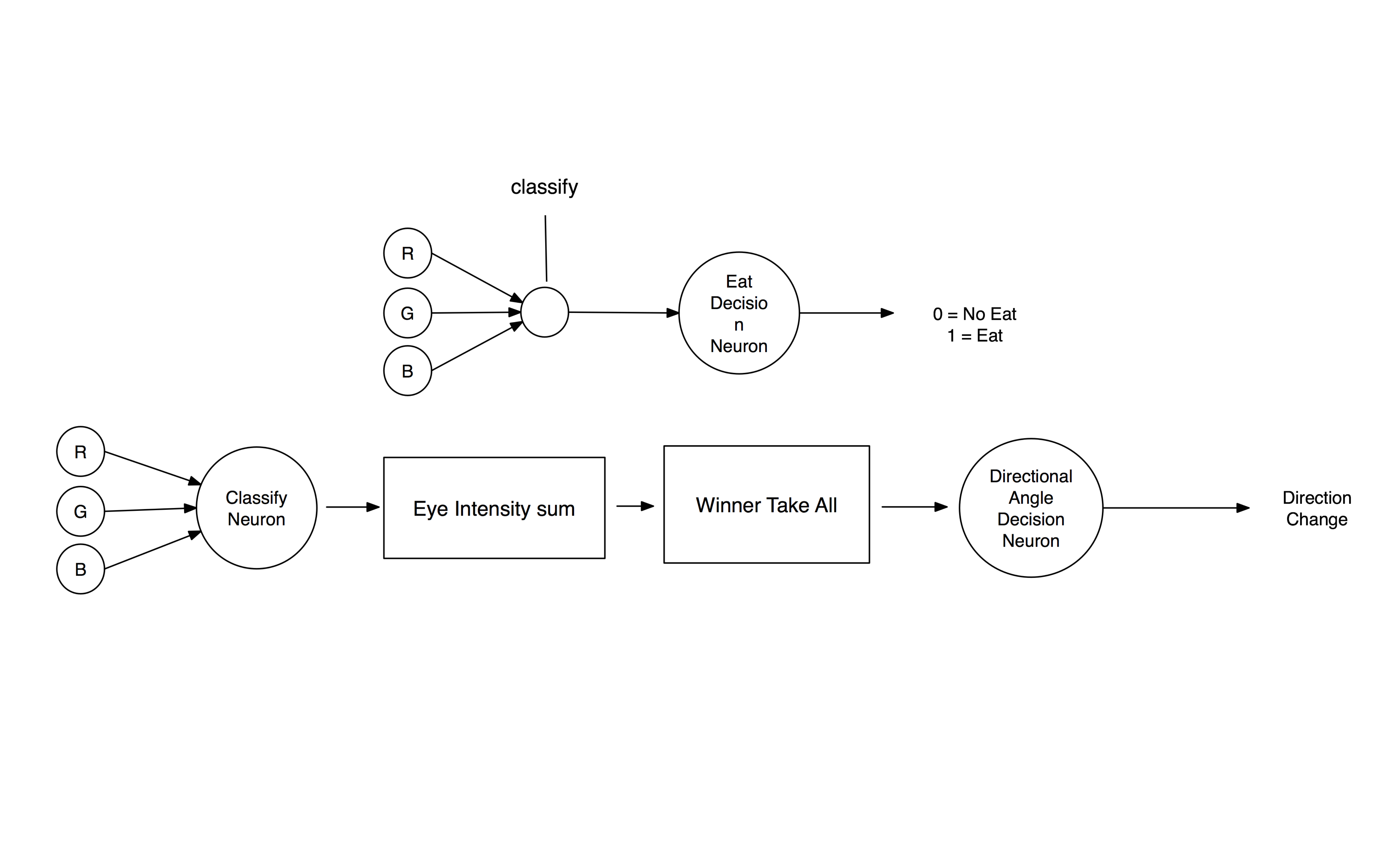


Figure 5: Network diagram of the LMS neuron for classifying food with pre-classification before winners take all network layer.

### 2.9 Architecture 8: Eat Few Objects based on Neuron Classification using RGB Percentage

In previous architectures, food was eaten from three sides 0, 1 and 7. In architecture 8 we will only eat food in one direction as show in figure 6.

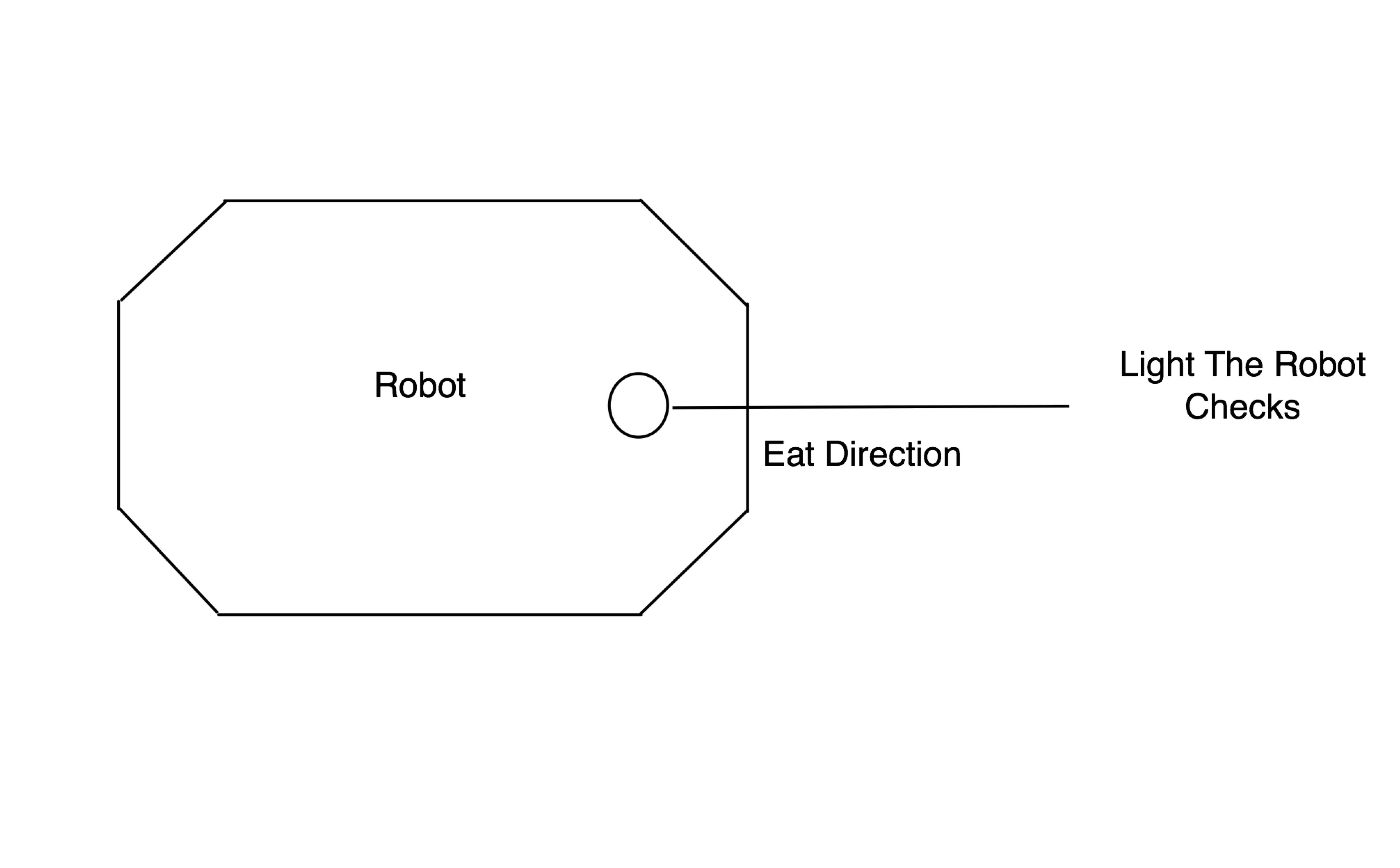


Figure 6: Diagram of the new eat direction for the robot.

We know that the light we check is straight in front of the robot. The front incoming light direction is index at 0 in the code. So in this architecture, we will only eat food directly in front of the Robot. The problem from the previous architecture is that it would eat classify the forward object as good but the objects at positions indexed at 1 and 7 are bad and eaten giving bad classification results. The architecture from figure 5 is still applied in this case with the change in eating.

# 3. Results

### 3.1 Architecture 0: No movement, measure lifetime

| Table 1: Over 200 runs, every run of a stationary robot had a lifespan of 5001 cycles | | |
| --- | --- | --- |
| Speed Rate | Lifetime Mean (Cycles) | Lifetime Standard Deviation |
| 0 | 5001 | 0 |

From Table 1, we see that the robot life is 5001 cycles if it does not move and does not eat any food. The robot needs one cycle to jump out the loop initial start. So totally there are 5000 cycles of life given initially by the world to the robot at birth. Therefore, we could see that the basic metabolic rate should be 0.0002 unit/cycle.

### 3.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food



Figure 7: Shows the max speed of the robot to be 0.1



Figure 8: Linear function of the life cycles with respect to the Robots speed.

From Figure 1, shows lifetime is directly correlated with the robot’s speed. When speed is less than 0.1 and increasing, then the lifetime of the robot has a nearly linear functional correlation with speed, where as speed increases, the lifetime is decreased. Furthermore, when the speed is over 0.1, the lifetime is constant. Therefore, the speed does not change, even though the speed is increasing. Changing the head angle originating from the same origin has limited effect on the life span of the robot, as the life span, with constant speed, showed no change. Therefore, the standard deviation for the lifetime at the same speed with head rotation is 0

### 3.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food



Figure 9: Error bar graph of varying speeds and directions where the standard deviation is represented as bars and the trend line represents the mean.

| **Table 2: Robot lifetime mean and standard deviation for different speed rate for comparison with future architectures.** | | |
| --- | --- | --- |
| Speed Rate | Lifetime Mean (Cycles) | Lifetime Standard Deviation |
| 0.02 | 4541 | 884 |
| 0.04 | 4233 | 973 |
| 0.06 | 3517 | 849 |
| 0.08 | 3093 | 765 |
| 0.10 | 2861 | 832 |
| 0.12 | 2858 | 827 |
| 0.14 | 2858 | 827 |

Figure 3 shows the final result of lifetime vs. speed, with the robot eating food all food it encounters. Like Architecture 1, the lifetime is correlated with speed, further supporting our hypothesis from the previous architecture of their relation. As speed increases, the mean lifetime decreases. When the robots speed goes over 0.1, the mean lifetime is stabilized. There is a standard deviation greater than 0 for this architecture because the robot is eat everything it encounters which is effecting its life either positively or negatively based on the standard deviations range around the trend line lifespan mean.

### 3.4 Architecture 3: Neuronal Classification of food

| **Table 3: Root mean squared error (RMSE) vs. lifetime training result** | | | | |
| --- | --- | --- | --- | --- |
| Weight Index | 0 | 1 | 2 | 3 |
| Value | -0.0593627 | -0.0574903 | 0.643285 | -0.0570887 |

Table 3 shows the training results for the classification neuron. We could see that at the beginning the RMS drops very quickly and dramatically. However, after a few cycles, the RMS values is stabilized but still tremble within 0.2 and 0.35. So we could see that the neuron is trained well after that. Table … shows the weight for the final trained neuron using RGB values. Therefore, in the following experience, we could use these weight to classify the objects robot meets and decide if robot should eat the objects or not.

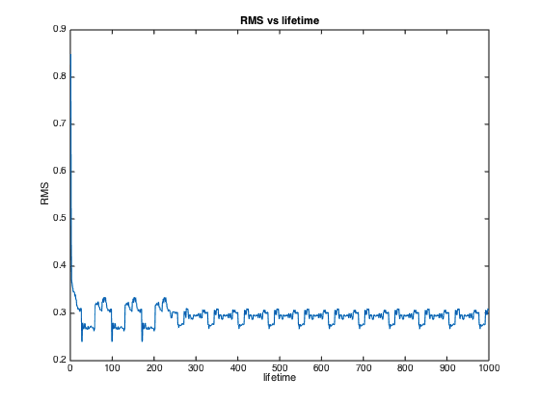


Figure 10: Direction decision neuron checking with a 5o rotation change after each life. This graph shows the RMSE dropping from 0.9 to asymptote around 0.3 suggesting that the LSM neuron has learned to classify objects it has encountered.

### 3.5 Architecture 4: Neuronal Movement towards brightest object

In section 2.5, we postulated a theory about neuronal directionality decision-making. We now need to verify the neuronal algorithms correctness. There is an internally implemented function, called intensity\_winner\_takes\_all, which returns the index of the receptor that has the highest intensity. We will compare the result from our implementation of a neuronal function with that of the internally coded function to see if they have the same result. Figure 5 shows the result of this comparison. We can see that the neuron implemented here generates the same result as the internal functions. Therefore, our neuronal function yields the same result as the internally coded function.

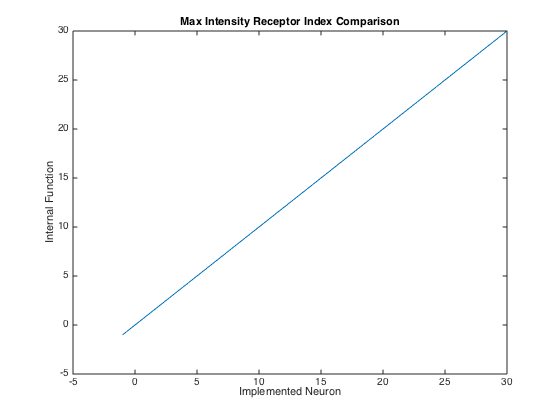


Figure 11: Intensity receptor index comparison between the internal function and the implemented neuronal function.

### 3.6 Architecture 5: Eat objects based on neuronal classification

In this architecture, the neuron architecture will classify food will be eaten based on the classification. Also, a direction change will be made based on the network in figure 3.

| **Table 4: Robot lifetime mean and standard deviation for different speed rate for comparison with future architectures.** | | |
| --- | --- | --- |
| Speed Rate | Lifetime Mean (Cycles) | Lifetime Standard Deviation |
| 0.02 | 4353 | 878.8 |
| 0.04 | 4323 | 965.2 |
| 0.06 | 3709 | 963.7 |
| 0.08 | 3661 | 1044.4 |
| 0.10 | 4884 | 1661.2 |
| 0.12 | 4862 | 1644.7 |
| 0.14 | 4862 | 1644.7 |

Table 4 shows a significant improvement over the mean lifespan found in table 2. This is attributed to the classification of food before eating, thereby helping the robot selectively choose only food that is classified as good by the LMS neuron.

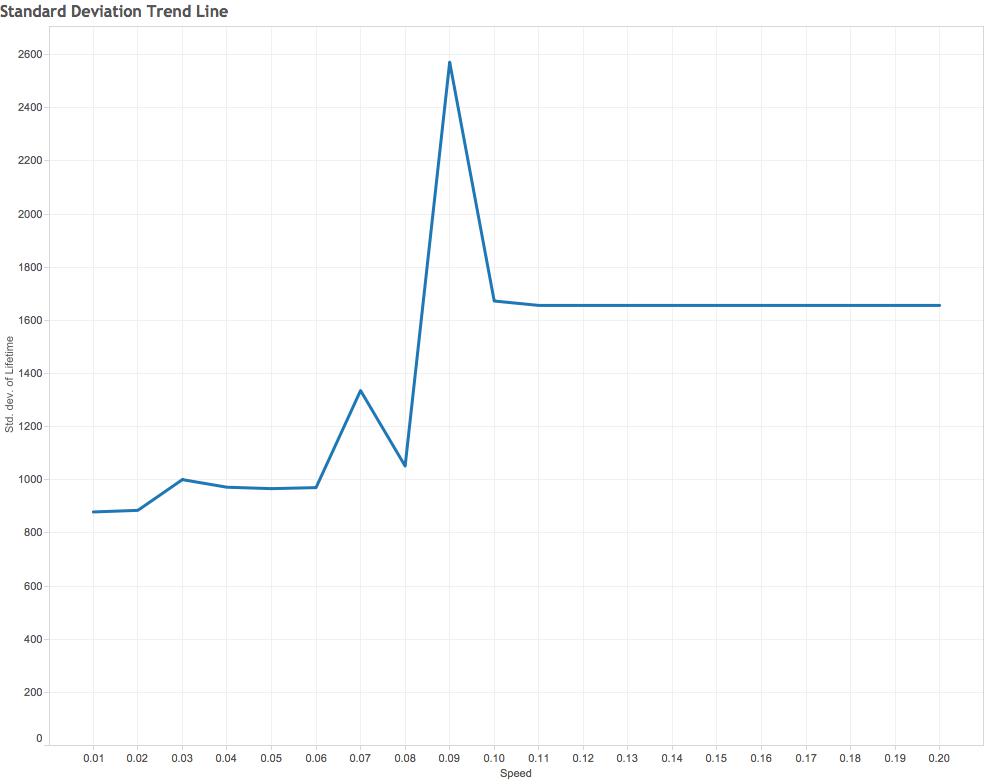


Figure 12: Standard deviation plotted as a line. We can see the standard deviation levels out once the speed is constant.

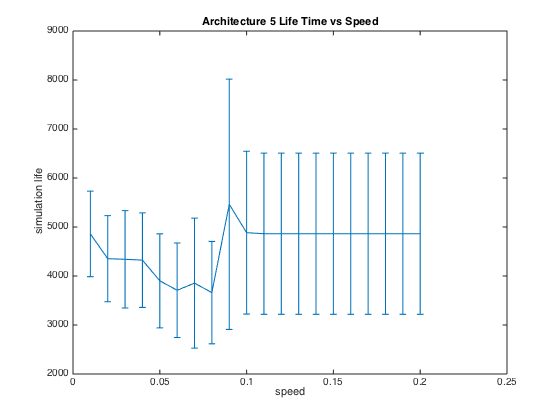
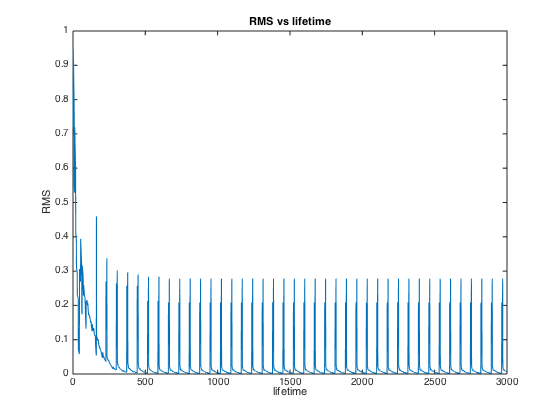


Figure 13: Intensity Receptor Index Comparison between the internal function and the implemented neuronal function. Note that the highest life span of the robot was 8000 and the minimum lifespan was 2500.

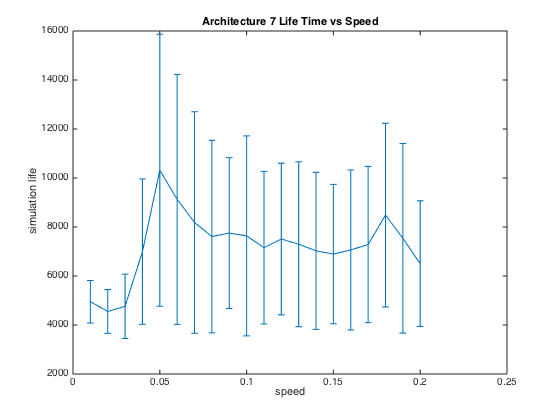
### 3.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage

| Table 5 Classification Neuron Weights after training with normalized inputs | | | | |
| --- | --- | --- | --- | --- |
| Weight Index | 0 | 1 | 2 | 3 |
| Value | 0.0364013 | -0.462995 | 1.26562 | -0.833772 |



### 3. 8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage

| Robot Lifetime Mean and Standard Deviation for Different Speed Rate with Direction selection for food | | |
| --- | --- | --- |
| Speed Rate | Lifetime Mean (Cycles) | Lifetime Standard Deviation |
| 0.02 | 4553 | 893.9 |
| 0.04 | 6995 | 2970.6 |
| 0.06 | 9127 | 5100.8 |
| 0.08 | 7609 | 3929.6 |
| 0.10 | 7639 | 4081.9 |
| 0.12 | 7510 | 3094.4 |
| 0.14 | 7026 | 3205.2 |
| 0.16 | 7063 | 3264.4 |
| 0.18 | 8485 | 3747.1 |
| 0.20 | 6504 | 2564.8 |



# 4. Discussion

# 5. Summaries and Conclusion

# 6. Acknowledgements

# 7. References

1. Caudell, Thomas. "Flatworld."
2. Haykin, Simon S. *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice Hall, 1999. Print

# 8. Appendix A

1. Perceptron.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Perceptron header file \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef PERCEPTRON

#define PERCEPTRON

typedef struct perceptron

{

int input\_num;

double \*weights;

double v;

double output;

double error;

int is\_inner\_neuron; /\*inner neuron has no error, but sigma instead\*/

void \*param; /\*Reserved variable for future use\*/

}perceptron;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Global variable \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Exportable functions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void perceptron\_default(perceptron \*neuron);

void perceptron\_clear(perceptron \*neuron);

#endif

2. Perceptron.c

#include "Perceptron.h"

#include <stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Exportable functions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void perceptron\_default(perceptron \*neuron)

{

neuron->weights = (double \*)malloc(sizeof(double) \* (neuron->input\_num + 1));

}

void perceptron\_clear(perceptron \*neuron)

{

free(neuron->weights);

}

3. LMSAlgorithm.h

/\*

\* LMSAlgorithm.h

\*

\* Created on: Oct 24, 2014

\* Author: lin

\*/

#include "Perceptron.h"

#include <math.h>

#ifndef LMSALGORITHM\_H\_

#define LMSALGORITHM\_H\_

extern perceptron neuron\_brain;

extern double accumulated\_rms;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Exportable functions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int LMScalculate(float \*inputs, int input\_num, int isCal, float expected);

#endif /\* LMSALGORITHM\_H\_ \*/

4. LMSAlgorithm.c

/\*

\* LMSAlgorithm.c

\*

\* Created on: Oct 24, 2014

\* Author: lin

\*/

#define OUTPUT\_NUM 1

#define LEARNING\_RATE 0.01

#define NEURON\_NUM 1

#include "LMSAlgorithm.h"

#include "Perceptron.h"

int initialized = 0;

perceptron neuron\_brain;

double accumulated\_rms = 0;

float forwardspeed;

void v\_function(float\* inputs, struct perceptron\* p)

{

int it = 0;

p->v = 0;

p->v += p->weights[0];/\*Bias\*/

for(it = 0; it < p->input\_num; it++)

{

p->v += (inputs[it] \* p->weights[it+1]);

}

}

void y\_function(struct perceptron\* p)

{

p->output = p->v;

}

void adjust\_function(float\* inputs, struct perceptron\* p, void \*params)

{

double target =\*((double \*) params);

int idx = 0;

p->error = target - p->output;

p->weights[0] += (p->error \* 1 \* LEARNING\_RATE);/\*Bias\*/

for(idx = 0; idx < p->input\_num; idx++)

{

p->weights[idx+1] += (p->error \* inputs[idx] \* LEARNING\_RATE);

}

}

void initialize(int input\_num)

{

int idx = 0;

neuron\_brain.input\_num = input\_num;

perceptron\_default(&neuron\_brain);

/\*This part uses the old data\*/

neuron\_brain.weights[0] = 0.493975;

neuron\_brain.weights[1] = -0.658991;

neuron\_brain.weights[2] = 0.720632;

neuron\_brain.weights[3] = -0.144448;

/\*This part is used to train the neuron\*/

// for(idx = 0; idx <= input\_num; idx++)

// {

// neuron\_brain.weights[idx] = 0.1 \* (rand()%10);

// }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Exportable functions \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int LMScalculate(float \*original\_inputs, int input\_num, int isCal, float expected)

{

int ret = 0;

double e = (double)expected;

float \*inputs = (float\*)malloc(input\_num \* sizeof(float));

memcpy(inputs, original\_inputs, sizeof(float) \* ( input\_num));

if(!initialized)

{

initialize(input\_num);

initialized = 1;

}

v\_function(inputs, &neuron\_brain);

y\_function(&neuron\_brain);

if(neuron\_brain.output > 0)

ret = 1;

if(isCal)

{

adjust\_function(inputs, &neuron\_brain, &e);

/\*Log rms\*/

accumulated\_rms += (pow(neuron\_brain.error, 2));

}

else

{

}

return ret;

}

void reset()

{

accumulated\_rms = 0;

}

5. DirectionControlNeuron.c

//

// DirectionControlNeuron.c

// FlatWorldIIV1.0ClassVersion\_dist2014

//

// Created by lin sun on 11/22/14.

// Copyright (c) 2014 lin sun. All rights reserved.

//

int set\_direction(WORLD\_TYPE \*world, AGENT\_TYPE \*agent, int eye\_idx)

{

VISUAL\_SENSOR\_TYPE \*\*eyes = agent->instate->eyes;

int num\_receptors;

int num\_bands;

int receptor\_idx = 0;

int band\_idx = 0;

int max\_receptor = -1 ;

float intensity = 0;

float maxintensity = 0;

float bodyx = 0;

float bodyy = 0;

float bodyh = 0;

int ret = 0;

num\_receptors = eyes[0]->nreceptors ;

num\_bands = eyes[0]->nbands ;

read\_visual\_sensor(world, agent) ;

extract\_visual\_receptor\_values\_pointer(agent, 0) ;

for(receptor\_idx = 0; receptor\_idx < num\_receptors; receptor\_idx++)

{

intensity = 0 ;

for(band\_idx = 0; band\_idx < num\_bands; band\_idx++)

intensity += eyes[0]->values[receptor\_idx][band\_idx] ;

ret = LMScalculate(eyes[0]->values[receptor\_idx], agent->instate->eyes[0]->nbands, 0, 0);

if(intensity > maxintensity && ret == 1)

{

max\_receptor = receptor\_idx;

maxintensity = intensity ;

}

}

return max\_receptor;

}

6. Controller.c