**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 4

2. Approach 4

2.1 Architecture 0: No movement, measure lifetime 4

2.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food 4

2.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food 5

2.4 Architecture 3: Neuronal Classification of food using Food RGB Values 5

2.5 Architecture 4: Neuronal Movement towards brightest object 6

2.6 Architecture 5: Move robot based on intensity and eat based on classification 7

2.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage 8

2.8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification 9

2.9 Architecture 8: Eat Objects with Small Mouse based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification 10

3. Results 11

3.1 Architecture 0: No movement, measure lifetime 11

3.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food 11

3.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food 13

3.4 Architecture 3: Neuronal Classification of food using Food RGB Values 14

3.5 Architecture 4: Neuronal Movement towards brightest object 15

3.6 Architecture 5: Eat objects based on neuronal classification 16

3.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage 18

3. 8 Architecture 7: Eat objects based on neuron classification using RGB percentage and choose direction based on light intensity and classification 20

3. 9 Architecture 8: Eat Few Objects based on Neuron Classification using RGB Percentage 21

4. Discussion 22

5. Conclusion 23

6. Acknowledgements 24

7. References 24

8. Appendix A 25

# Abstract

The Problem of improving robot lifetime in the simulated world created by professor Caudell can be used as a great learning platform for the implementation, understanding, and improvement of neural networks. The LMS neuron can be used to classify objects according to their properties. In order to classify objects effectively and efficiently in a simulated world, a strong understanding of how to use these properties is more important than knowledge of the neuronal algorithm itself. Neurons, such as the LMS neuron, need to be implemented and tested in an environment in order to determine whether they are working and trustable.

In this project, we will use the animal simulation world, provided by Thomas Caudell, as a platform for testing various neural network architectures. The starting point of our overall case study involves a non-neuron robot study, which is used as an environmental control point. After a based line was determined, a simple single neuron object classifying architecture was introduced leading to the implement more neuronal architecture for the purpose of direction determination. These studies come to a crescendo involving the combination, implement, and integration of all the neural architecture studies. Yielding results, which emphasize the LMS neuron’s ability for classifying objects in the simulated world. Furthermore, it was found that different inputs affect the neuron classification abilities. The conclusions draw at the end of this paper will examine and discuss what potential future works could be constructed from our final architecture and how the final architecture performed when compared with controls and with factoring in previous architectural improvements and failures.

Key words: LMS, Classification, Neuron, Neural networks, robot learning

# 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 robot environmental model. (Caudell) The research, in total, involved 9 architectures. These architectures were used to establish the parameters of lifespan with respect to the simulated organism and its environment. Providing a basis for analyzing the implemented neural network algorithms.

The goal of building these neural networks is to create a “brain” for the robot entity to live as long as possible in its environment. A determination on what neuronal design structures degrade or enhance lifespan will be used to support architectural changes in the process of achieving this goal. There will be two major kinds of neurons implemented, classification neurons and direction choice neurons. Classification neurons will be used to classify objects within the robots environment and direction neurons could be activated subsequently to determine a direction of the nearest non-poisonous objects around it.

# 2. Approach

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

For the first architecture, the basic metabolic rate for the robot was tested. The robot’s speed was set to 0 to hold the robots metabolism at a constant baseline. The robot was programmed to do nothing until it dies and the lifespan was measured to calculate the basic metabolic rate.

### 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. The robots direction was also varied to see if other environmental factors were correlated to lifespan. The goal of this experiment is to determine a relation between the speed of the robot and its metabolism.

The robot will start at the same position at the beginning of every new life cycle. The direction and speed will be varied every thing else will be held constant. The robots head will be moved 5 degree counter-clockwise, upon reset. Lastly, 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 robot’s lifetime when eating everything it encounters. 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 lifespan from the environmental makeup, the initial robot’s head angle will be changed for each run. The initial position will be held constant at the point of origination. This architecture will be used as a control, for the future neuronal architectures. This control will be used for comparisons (like architectures 0 and 1) when deciding whether future architectures improve the lifespan of the robot.

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

Architecture 3 is designed for classifying food, in the world, as either good or bad based on whether environmental objects increase or decrease the energy of the robot. This architecture will rely on the use of the LMS neuron algorithm. The inputs are visual readings, as RGB values, and the output is 0 for bad or 1 for good. If the food increases the energy of the robot the food is classified as good; all other changes are considered bad.

In order to classify the objects, the LMS neuron’s classification weights need to be determined. The objects encountered by the robot will be used as supervised training sets. The learning rate is set to 0.01. The stopping criteria for training is set as when the Root Mean Squared Error (RMSE) change between two consecutive life times is less than 0.00000001 or the life time is greater than 3000. This is done in order to avoid being caught in an infinite loop. Figure 1 shows a diagram of the neural design.



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 let the robot move towards the brightest object in its environment using a winner-takes-all neuronal network implementation. In the architectural design process, movement is the next logical step after classification of food. The robot has to have a way of moving toward objects in its environment. In order to give the robot the ability to seek out objects a direction neuron will be added. This new neuron will make a direction choice.

The direction 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. This means that objects having higher light intensity are closer to the robot than other objects with lower light intensity. In the simulation world, the robot goal is to survive for as long as possible. Therefore, it should try to move towards the objects in the world closest to it. In order to compare the intensity of different lights, a function to calculate the intensity of the light must be postulated.

Equation 1.1 shows how to calculate the intensity of lights based on its RGB values. A neuronal model, like the one in Figure 2, will be implemented. This is a two-layer neuron structure. The inputs for the neuron are RGB lights bands from 31 directions and the output is the angle of the brightest object with respect to the current horizontal axis. The first layer of neurons is used to calculate the intensity for each input, and the second layer of neurons will pick the highest input intensity. The direction that the robot will change towards is the direction of the winning light intensity.

Intensity = sqrt(R^2 + G^2 + B^2) (eq 1.1)



Fig 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

Figure 3 shows the frame for architecture 5, where we will integrate classification neuron and direction neuron. Each time the robot stops, classification neuron is activated to classify the objects encountered by the robot. If the objects are not poisonous, the robot will eat it. Otherwise, the robot will just skip the objects. After that, direction neuron is activated to pick up the direction with highest intensity, so that the robot will move towards that direction. If direction neuron can’t find any object, the robot will rotate 45 degree clockwise and then move towards that direction. Because the neuron could classify the objects, the robot will not eat poisonous objects. Therefore, the robot will not waste energy on poisonous objects, which means that robot in this architecture should have larger average lifespan than the robot in architecture 2.



Fig 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 that object is decreased uniformly over each signal band. The un-normalized input of RGB values cannot be trusted to classify objects, due to this fact. Moreover, when the robot moves, it will pick up the direction with the highest intensity light value based on equation 1.1. However, it is possible that the direction picked up by direction neuron 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 smart direction neuron, which can move towards the direction with the closest nonpoisonous object. To solve this problem we need implement a classification neuron which can classify objects regardless of the distance. Thus, a new classification neuronal network will be created that takes normalized RGB values and classifies these values as either good or bad based on the frame in Fig 4. Similar to classification in architecture 3, the inputs are normalized RGB values instead of original RGB values. The output is 0 or 1. If the objects can increase the energy, the output is 1; otherwise, it is 0. Equation 1.2 is used to normalized RGB values in this architecture, where V is individual band of the light, R, G or B. In this architecture, we will train the new classification neuron using objects encountered by the robot. To randomize the objects encountered by the robot, we will change the head direction each time. The learning rate for the classification neuron is still 0.01 while the stopping criteria is when RMS change between two consecutive life time is less than 0.00000001 or the life time is greater than 300 to avoid infinite loop.

Normalized V = V/(R + G + B) (Eq 1.2)



Fig 4: Representation of the normalized classification approach

### 2.8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification

In architecture 6, we implements a new classification neuron, which can classify the light based on normalized RGB values. Therefore, for the object to eat, we can ignore the distance factor. It is known that the robot wants to move towards the direction where the objects are closest to it and are non-poisonous. Therefore, we need to implement a upgraded direction neuron, which can select the direction with highest light density and the light is classified as non-poisonous. Fig 5 shows two kinds of neurons we will implement for the robot. To make it easier, we will integrate new classification neuron with direction neuron to make a smarter direction neuron. Each time when the direction checks a light, the classification neuron will classify the neuron first. Then we will multiply the output by classification by the intensity output from first layer direction neuron. Therefore, if the light is classified as poisonous, the intensity is set as 0. So the light which smart direction neuron picks up has the highest intensity and also non-poisonous. Therefore, like architecture 5, each time the robot stops, classification neuron is activated to classify the objects encountered. If it is poisonous, the robot just skip it. Otherwise, the robot will eat it. Then smart direction neuron is activated to choose a direction to move. If there is no suitable direction made from direction neuron, the robot will rotate 45 degrees clockwise, and move towards that direction.



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

### 2.9 Architecture 8: Eat Objects with Small Mouse based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification

Fig 6 shows the shape of the robot we are controlling in the simulation world. When the robot stops, it will first use classification neuron to classify the objects it encounters. In the architecture, we use the middle light to classify objects, which is the beam in fig 6. However, when robot eat objects, it will eat objects on three sides. Therefore, a new problem comes. Are we sure that the objects on the other two sides are non-poisonous? If the objects on the other two sides are poisonous but the objects on side 0 are good. Then the robots will eat all objects, therefore, the robot is wasting energy on poisonous objects. So in this architecture, we will narrow down the mouse of the robot, so that it can only eat the objects on side 0, which can be classified directly by classification neuron.



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

# 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. It is know that the robot needs one cycle to jump out the loop initial start. So totally there are 5000 cycles of lifespan 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



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



Fig 8: Average and standard deviation of the lifespan with respect to the robot’s speed.

Figure 7 shows that the lifetime of robot is directly correlated with its speed. When robot speed is less than 0.1 and increasing, the lifetime of the robot has a nearly linear functional correlation with speed, where as speed increases, the lifetime is decreased. However, when the speed is over 0.1, the lifetime is constant. Therefore, the lifespan does not change anymore, even though the speed is increasing. What’s more, 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 for the same speed with different head orientations is 0.

From this experiment, we could see that the movement metabolic rate is fixed when the movement rate is reached at 0.1. Therefore, in the following experiment for neuron trainings, we could fix the movement speed in order to get rid of the effect of speed on lifespan.

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



Fig 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 9 shows the final result of lifetime vs. speed, with the robot eating 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 again like architecture 1. However, instead of getting standard deviation as 0, there is a standard deviation greater than 0 for this architecture because the robot can eat everything it encounters in the world, which is effecting its life either positively or negatively based on the standard deviations range around the trend line lifespan mean. Moreover, table 2 provides the basic line for our future neural network implementation. If neuron network could classify the objects correctly, theoretically the robot should survive longer than current robot for each speed.

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

| **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 |

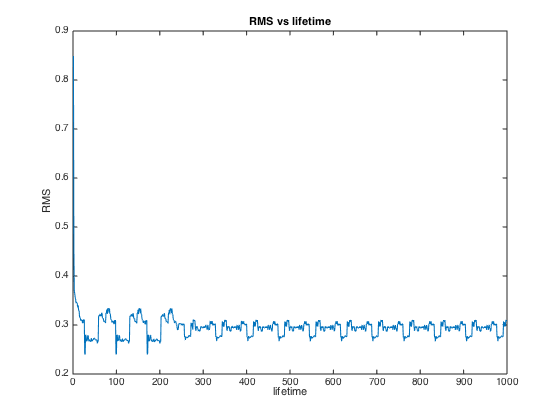
Fig 10 shows the training results for the classification neuron. We could see that at the beginning the RMS drops very quickly and dramatically. However, after a few cycles, the RMS values is stabilized but still tremble within 0.2 and 0.35. So we could see that the neuron is trained well after iterations. Table 3 shows the final weight for the trained neuron using RGB values. Weight 0 is used for bias in the calculation. Upon the observation of three weights, we could get a preliminary conclusion that the second bandwidth yields good food while the other two bandwidths are related with bad food. Then in the following experience, we will try to use these weights to classify the objects robot meets and decide if robot should eat the objects or not. Then we can verify our primary conclusion.

Fig 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

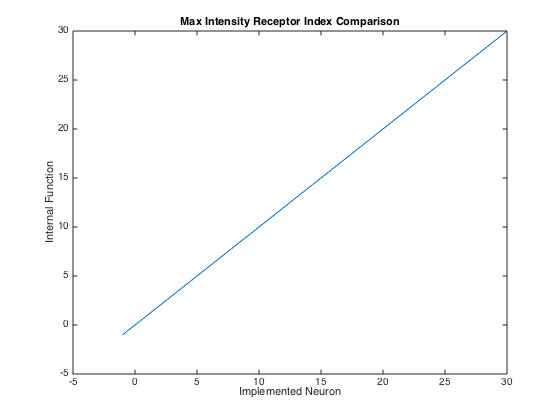
In section 2.5, we postulated a theory about neuronal directionality decision-making. We now need to verify the neuronal algorithms correctness. There is an internally implemented function, called intensity\_winner\_takes\_all, which returns the index of the receptor that has the highest intensity. We 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 11 shows the result of this comparison. We can see that the neuron implemented here generates the same result as the internal functions. Therefore, our direction neuron yields the same result as the internally coded function and we can trust the direction picked up by direction neuron.

Fig 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 robot could activate both classification neuron and direction neuron to search and eat good foods in the world. After implementation, we could see how the lifetime changes.

| **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 |

By comparing table 4 and table 2, we could see that architecture 5 gains a significant improvement over the mean lifespan found in architecture 2. The improvement 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. Like architecture, there is a significant standard deviation for the same speed. More interesting, the standard deviation is stabilized when the speed reaches 0.1, as shown in Fig12. Probably the max speed of the robot in the simulation world is 0.1. But this experiment proves directly that our new network architecture works well in classifying objects in the world.



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



Fig 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 |



Figure 14: RMSE error drops as the LMS neuron adjusts its weights and learns how to classify objects found in its environment.

Normalization of the data appears to not be detrimental to the overall performance of the classification of incoming light based on figure 14. The large oscillation is hypothesized to be from classification of good food in front of the robot while there is bad food to the sides of the robot, since the robot classified the food as good all the food around it gets eaten. This will be addressed as a fine-tuning to architecture 7 with architecture 8. Take note that a biased input was included in the classification of the objects. Besides, upon observation of the weights in table 5, we can see get the same conclusion that the second light bean contributes to the good food while the other two beams are related to bad food. What is more, it seems that the third beam has higher contribution to bad food than the first beam by comparing their weights. And the following architecture will use the new classification neuron and can prove our hypothesis.

| Table 6 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 | 7337 | 1253 |
| 0.04 | 12763 | 4349 |
| 0.06 | 33165 | 15722 |
| 0.08 | 28193 | 11208 |
| 0.10 | 23935 | 9429 |
| 0.12 | 23911 | 9543 |
| 0.14 | 23911 | 9543 |
| 0.16 | 23911 | 9543 |
| 0.18 | 23911 | 9543 |
| 0.20 | 23911 | 9543 |

### 3. 8 Architecture 7: Eat objects based on neuron classification using RGB percentage and choose direction based on light intensity and classification



Fig 15: Mean lifespan trend line with standard deviation bars.

By comparing table 6 and table 4, we could see that our architecture gains a big improvement in robot’s lifespan. The lifetime is almost 4 times as big as before. Therefore, it solids conclusion in architecture 6 that the second beam is related with good foods, while the other two beams are related with bad foods. By comparing fig 13 and fig 15, we could see that the best speed for robot to survive is changing if we change the architecture. From table 6, we can see that the best speed is 0.06, where the robot could survive for more than 30000 cycles in average for one life time.

### 3. 9 Architecture 8: Eat Few Objects based on Neuron Classification using RGB Percentage

| Table 7 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 | 8491 | 1200 |
| 0.04 | 24987 | 8612 |
| 0.06 | 51701 | 8355 |
| 0.08 | 28601 | 13893 |
| 0.10 | 32059 | 4848 |
| 0.12 | 30550 | 4426 |
| 0.14 | 30550 | 4426 |
| 0.16 | 30550 | 4426 |
| 0.18 | 30550 | 4426 |
| 0.20 | 30550 | 4426 |

Fig 16: Mean lifespan trend line with standard deviation bars.

By comparing table 6 and table 7, we could see that our new architecture works well. At the best speed, our robot could survive more than 50000 cycles per life time. Therefore, we could conclude that objects on side 1 and 7 could not be classified directed by single beam. However, by classifying the correct objects, the robot could survive longer than other architectures.

# 4. Discussion

In this project, we try to implement a neuronal network for the robot to survive in the simulation world. Simply, there are two kinds of neurons here, classification neurons and direction neurons. Classification neurons are used to classify objects seen by the robot to check if it is good or not, while direction neurons are used to pick up a direction for the robot after eating.

For our first try, we implemented a neuron network, which can classify the objects based on their RGB values and choose direction based on the light intensity. In the real experiments, by comparing figure 9 and figure 13, we can see that our neurons works well. The lifespan average for each speed is extended, which means that the robot can survive longer.

After the first try, we realized that there is a big issue in our architecture, that is how to classify objects regardless of the distance between the robot and the objects. Therefore, we implemented a new neuronal architecture, which takes normalized RGB values instead of the original ones. Equation 1.2 is supplied to transferred the original RGB values to normalized values. Then we realized that when the robot picks up a direction, it can try to pick up the direction which leads it to good foods. Therefore, we upgraded our direction neurons and make it to pick up the direction which has the nearest non-poisonous foods. After that, we tried our new architecture to check the lifespan of robot with new brain. As shown in figure 15, the lifespan of the robot is improved a lot at individual speed. Sometimes, the liftspan can reach 30000 cycles, which is amazing compared the original lifespan.

Afterwards, we got a new insight, whether the objects on two adjacent sides are classified by our neurons correctly. It is known that our neuron will classify objects only based on the straight light. Therefore, we did an improvement and let the robot just eat the objects on one side, where objects are classified correctly. As shown in figure 16, it did a big improvement again, and sometimes the lifespan average can reach 50000, which suggests that our neuron architecture works well in the simulation world.

# 5. Conclusion

Through this projects, we see that neurons works well in classification. In our trials, we tried to use neurons to classify objects in the world and use direction neuron to help robot pick up directions. From our results, we can see clearly that the robot survives longer and longer. The algorithms used in the architecture are mainly LMS and winner-take-all algorithms. Besides, we use visual lights to classify objects, which means that light is highly related with the object property in the simulation world.

However, there are still more improvements needed to be done. First, the robot only eats objects on one side of the robot. So if there are some good objects lying on the other sides, the robot will miss it probably. So we need to implement a new architecture to classify these objects and try to eat these objects. Our ideas is to rotate the body after eating one side. However, there is a new issue, whether the rotation energy consumption is bigger than the energy got from other sides. Therefore, it needs more experiments to explore it. Secondly, there is another signals which can also be used in the simulation world, that is sound. But we can’t guess how the sound works in classification. We knows that the light beam got from the eye is from only one object. But how about sounds? Are they mixed signals from a lot of objects? Can we distinguish the sound just use LMS neuron? These questions are very hard to answer and need more experiments to test them. We did not try to explore more on it because we think that the sound can’t be used to distinguish objects so far in our algorithm.

# 6. Acknowledgements

Great Thanks should be given to Thomas Caudell for createing the simulated environment the robot lives in and for being one of the best teachers at the University of New Mexico. Also, Thank you Fengshu Xu for being of great help ironing out the kinks in our architectures. Finally, Special thanks should be given to our team for making all architectures possible and for all the hard work they put in to the project.

# 7. References

1. Caudell, Thomas: Simulated world
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

//

// Controller.c

// FlatWorldIIV1.0ClassVersion\_dist2014

//

// Created by lin sun on 10/02/14.

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

//

#include "LMSAlgorithm.h"

#include "Perceptron.h"

typedef struct object

{

float \*inputs;

int input\_num;

float output;

struct object \*next;

}object;

typedef struct object\_list

{

object \*header;

object \*cur;

int object\_num;

}object\_list;

static object\_list list;

int epoch\_num = 0;

double rmss[10000];

int object\_num = 0;

float forwardspeed = 0.01;

float init\_x;

float init\_y;

float init\_head\_position = 0;

/\*Unexported functions\*/

object \*addObject(float \*input, int input\_num, float output);

void clearObjects();

void agents\_controller( WORLD\_TYPE \*w )

{ /\* Adhoc function to test agents, to be replaced with NN controller. tpc \*/

AGENT\_TYPE \*a ;

int collision\_flag=0 ;

int i,k ;

int maxvisualreceptor = -1 ;

int nsomareceptors ;

int nacousticfrequencies ;

float delta\_energy, old\_delta\_energy;

float dfb , drl, dth, dh ;

float headth ;

//float forwardspeed ;

float maxvisualreceptordirection ;

float bodyx, bodyy, bodyth ;

float \*\*eyevalues, \*\*ear0values, \*\*ear1values, \*\*skinvalues ;

float ear0mag=0.0, ear1mag=0.0 ;

time\_t now ;

struct tm \*date ;

char timestamp[30] ;

int is\_poisonous = 0;

//temporary file

char eye\_data\_file\_name[20] = "./dat/eye\_data\_";

char direction\_data\_file\_name[20] = "./dat/direction\_data";

char eye\_data\_file\_name\_str[50] = "";

int idx = 0;

float stopping\_criteria = 0.00000001;

FILE \*fp = 0x0;

int ret = 0;

float \*input\_data = 0x0;

a = w->agents[0] ; /\* get agent pointer \*/

/\* test if agent is alive. if so, process sensors and actuators. if not, report death and

reset agent & world \*/

if(a->instate->metabolic\_charge > 0.0)

{

/\* get current motor rates and body/head angles \*/

read\_actuators\_agent(a, &dfb, &drl, &dth, &dh ) ;

read\_agent\_body\_position( a, &bodyx, &bodyy, &bodyth ) ;

read\_agent\_head\_angle( a, &headth );

/\* read somatic(touch) sensor for collision \*/

collision\_flag = read\_soma\_sensor(w, a) ;

skinvalues = extract\_soma\_receptor\_values\_pointer( a ) ;

nsomareceptors = get\_number\_of\_soma\_receptors( a ) ;

/\* read visual sensor to get R, G, B intensity values to collect x values\*/

read\_visual\_sensor(w, a) ;

eyevalues = extract\_visual\_receptor\_values\_pointer( a, 0 ) ;

for(k = 0 ; k < nsomareceptors ; k++)

{

ret = LMScalculate(eyevalues[a->instate->eyes[0]->nreceptors/2], a->instate->eyes[0]->nbands, 0, 0);

if((k == 0) && skinvalues[k][0] > 0.0 && ret == 1)

{

delta\_energy = eat\_colliding\_object(w, a, k) ;

// /\*Train the neuron\*/

// if(delta\_energy != 0)

// {

// object\_num++;

// LMScalculate(eyevalues[a->instate->eyes[0]->nreceptors/2], a->instate->eyes[0]->nbands, 1, (delta\_energy > 0 ? 1.0 : 0.0));

// }

}

}

/\*Move robot\*/

ret = set\_direction(w, a, 0);

read\_agent\_body\_position(a, &bodyx, &bodyy, &bodyth) ;

if(ret == -1)

{

set\_agent\_body\_angle(a, bodyth + 45);

}

else

{

set\_agent\_body\_angle(a, bodyth + a->instate->eyes[0]->receptor\_locations[ret] + a->instate->eyes[0]->receptor\_directions[ret]);

}

/\* move the agents body \*/

set\_forward\_speed\_agent(a, forwardspeed) ;

move\_body\_agent(a) ;

/\* decrement metabolic charge by basil metabolism rate. DO NOT REMOVE THIS CALL \*/

basal\_metabolism\_agent(a) ;

simtime++ ;

} /\* end agent alive condition \*/

else

{

/\* Example of agent is dead condition \*/

printf("agent\_controller- Agent has died, eating %d objects. simtime: %d\n",a->instate->itemp[0], simtime ) ;

now = time(NULL) ;

date = localtime( &now ) ;

sprintf(eye\_data\_file\_name\_str, "./dat/timevsspeed.csv");

if((fp = fopen(eye\_data\_file\_name\_str, "a+")) != 0x0)

{

fprintf(fp, "%f,%f,%d\n", forwardspeed, init\_head\_position, simtime);

fclose(fp);

}

/\* Example as to how to restore the world and agent after it dies. \*/

restore\_objects\_to\_world(Flatworld) ; /\* restore all of the objects back into the world \*/

reset\_agent\_charge( a ) ; /\* recharge the agent's battery to full \*/

a->instate->itemp[0] = 0 ; /\* zero the number of object's eaten accumulator \*/

/\* keep starting position the same and change head angle \*/

init\_head\_position += 20;

nlifetimes++ ;

if(nlifetimes%18 == 0)

{

init\_head\_position -= 360;

forwardspeed += 0.01;

}

init\_x = (Flatworld->xmax + Flatworld->xmin)/2;

init\_y = (Flatworld->ymax + Flatworld->ymin)/2;

printf("\nagent\_controller- new coordinates after restoration: x: %f y: %f h: %f f: %f\n", init\_x, init\_y, init\_head\_position, forwardspeed) ;

set\_agent\_body\_position(a, init\_x, init\_y, init\_head\_position) ;

/\* Accumulate lifetime statistices \*/

simtime = 0 ;

//maxnlifetimes

if(nlifetimes >= 360) //|| (epoch\_num > 10 && fabs(rmss[epoch\_num - 1] - rmss[epoch\_num - 2]) < stopping\_criteria)) /\*Add stopping condition for the neuron training to stop\*/

{

/\*plot data and clean up data\*/

// sprintf(eye\_data\_file\_name\_str, "%sdata.csv", eye\_data\_file\_name);

// printf("store the data and exit with epoch %d\n", epoch\_num);

//

// if((fp = fopen(eye\_data\_file\_name\_str, "w+")) != 0x0)

// {

// /\*Log rms\*/

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

// {

// fprintf(fp, "%d,%lg\n", idx+1, rmss[idx]);

// }

//

// fprintf(fp, "\nThe final weights are:\n");

//

// /\*Log weights\*/

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

// {

// fprintf(fp, "%lg,", neuron\_brain.weights[idx]);

// }

//

// fclose(fp);

// }

// epoch\_num = 0;

exit(0) ;

}

else

{

// rmss[epoch\_num] = pow(accumulated\_rms/object\_num, 0.5);

// printf("This is the %dth epoch with %f random speed with %f rms\n", epoch\_num, forwardspeed, rmss[epoch\_num]);

// epoch\_num++;

// accumulated\_rms = 0;

// object\_num = 0;

}

} /\* end agent dead condition \*/

}/\*end agents\_controller()\*/

object \*addObject(float \*inputs, int input\_num, float output)

{

object \*new\_object = (object \*)malloc(sizeof(object));

new\_object->inputs = inputs;

new\_object->input\_num = input\_num;

new\_object->output = output;

list.object\_num++;

list.cur->next = new\_object;

list.cur = list.cur->next;

list.cur->next = NULL;

return new\_object;

}

void clearObjects()

{

object \*o1;

object \*o2 = list.header->next;

list.header->next = NULL;

list.object\_num = 0;

while(o2)

{

o1 = o2;

o2 = o2->next;

free(o1);

}

}