Combining Q-Learning with Artificial Neural Networks in an Adaptive Light Seeking Robot

Steve Dini and Mark Serrano

May 6, 2012

Abstract

Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of performing a given action in a given state and following a fixed policy thereafter. The basic implementation uses a q-table to store the data. With increasing complexity in the environment and the agent, this approach fails to scale well as the space requirements become prohibitive. In this paper, we investigate an alternative implementation in which we use an artificial neural network as a function approximator and eliminate the need for an explicit table.

1 Introduction

It is hard to be a robot, but its even harder to be the programmer. For decades, researchers have attempted to simulate intelligence and, more specifically, create agents that can learn. An important technique in machine learning is Reinforcement learning. Reinforcement learning systems learn, by trial and error, what the optimal action to take in any state is. Feedback is given in the form of a reward. The reward is defined in terms of the task to be achieved; positive reward is given for successfully achieving the task or for any action that brings the agent closer to solving the task while negative reward is given for any actions that impede the agent from successfully achieving the task. According to Gasket, "Reinforcement learning lies between the extremes of supervised learning, where the policy is taught by an expert, and unsupervised learning where no feedback is given and the task is to find structure in the data" [1].

Q-learning is an example of a reinforcement learning technique used to train robots to develop an optimal strategy to solving a task. A table is often used to store the utility data as the agent wanders in the environment. In this paper, we will investigate the practical limitations of using tables as well as the viability of using artificial neural networks to approximate the utility in place of using tables.

2 Q-Learning

Q-Learning uses an action-value function that calculates the expected utility of performing a specific action in a discrete state and following a fixed policy thereafter. Three different functions are involved: memorization, exploration and updating (fig 1) [2]. In response to the present situation, an action is chosen. In order to ensure that both exploration and exploitation are performed, a set probability value is used alongside a randomly generated number to de-

termine whether the agent will explore or exploit. If the random number is above the probability threshold, the optimal action yielding the highest q-value is selected (exploitation). Otherwise, a random action is selected (exploration). This promotes both aspects of exploitation and exploration which are necessary to ensure the success of the learning technique [2].

Each time the agent performs an action a in state s at time t, the agent gets a reward r which represents how close it is to solving the task. The utility for performing this action is then updated according to the update rule:

$$Q(s,a) = Q_t(s,a) + \alpha[r + \gamma.maxQ(s',a') - Q(s,a)] \quad (1)$$

where s' is the next state, a' is the optimal action in s', α is the learning rate and γ is the discount factor. After applying this rule, all the other entries in the table remain unchanged. Since there is an expected reward for each action in a multitude of states, the total size of the table (state-space) grows exponentially as the complexity of the problem increases. This rapid growth results in dismal scalability in addition to physical space limitations in terms of data storage.

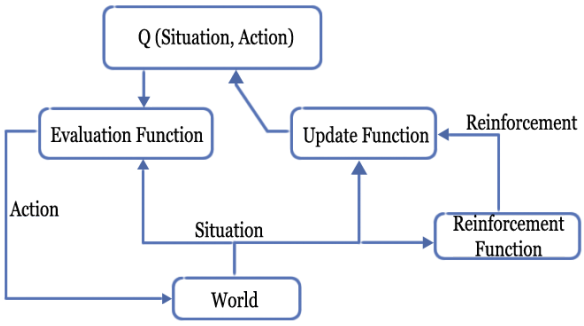

Figure 1: The 3 functions of q-learning

3 Artificial Neural Networks (ANN)

Artificial Neural Networks are models heavily studied in both cognitive and computer science. They are inspired by the structure and function of biological neural networks. The basic building blocks of neural nets are nodes that generate an output value given an input value. At the net's extremities, input nodes usually consist of a single input but can have multiple outputs. Inversely, output nodes usually have a single output, but can have a multitude of input connections. The neural net uses these interconnected nodes to relay signals across the structure, from an input source through to an output. Each connection between nodes has a weight parameter that affects what values the nodes receive. The neural network is trained by comparing the output to a target for a given set of inputs, and generating an error value. This error is then used to update the connections and/or the weight of those connections in the neural net.

While much of the literature regarding the use of neural nets for q-learning use recurrent neural networks, we decided to use simple feed forward nets instead. The difference between these two types of neural net structures lies in types of connections that are permitted and consequently, the directional flow of data. In a recurrent neural net, connections are allowed to establish loops in such a manner where top layers can recursively feed data back down to previous layers. This kind of net allows for an abstract memory-retention structure to emerge. In contrast, feed-forward nets are only allowed to feed data upwards from lower layers to higher layers. This results in a static directional data flow.

4 Q-learning on the light finding task

The idea of developing a control system for a robot by q-learning has been around for quite a while [3]. The domain of problems has generally been limited to light finding tasks, obstacle avoidance tasks and the pole balancing problem [4][5]. For our purposes, we are going to investigate q-learning on a light seeking task. The light-seeking problem consists of an agent in an environment with a single light source, where the agent is expected to develop a strategy for locating and subsequently reaching the light[6][7].

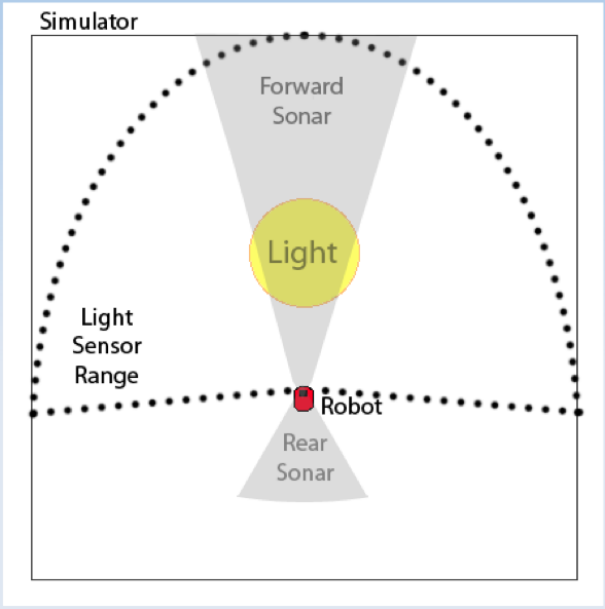

Figure 2: Experiment Setup

4.1 Experiment Setup

The environment consists of a single robot with a centrally located light source as shown in fig 2. The experiment was setup and run in the pyrobot simulator using a single pioneer robot.

In order to record useful measurements, the robot was given the ability to sample two light value readings, one to the left and one to the right, and then contrast them. The robot also had the means to detect obstacles, both directly ahead and behind it. It achieved this by observing the readings it obtained from the sonar sensors and determining if they were below a threshold of 0.5. The total sensory input this robot had is summarized fig 3 below.

4.2 State Representations

The sensors on the robot are used to generate a representation of the state the robot is currently in. Therefore, the state is simply a summary of what the robot is currently perceiving. In addition to these perceptions, we also include limited internal state information when generating a state representation for the robot. Specifically, we include the value of a variable that reports whether light sensor values have not changed in 5 consecutive time steps. The motivations behind having this extra piece of information lie in the fact that our goal is to motivate the robot to move towards the light source. If the light values are not changing this is an indication that the robot might be stuck somewhere, or the robot is repeating a sequence of forward and backward movements. Since this is something undesirable, offering negative reward in these situations may assist the learning process. In order to offer the negative reward however, it is necessary to know that the above-mentioned situation is happening, hence the need for that variable.

In order to fully explore the strengths and weaknesses of q-learning, we propose evaluating the performance of q-learning in both a very rich as well as a constrained representation of the environment. In

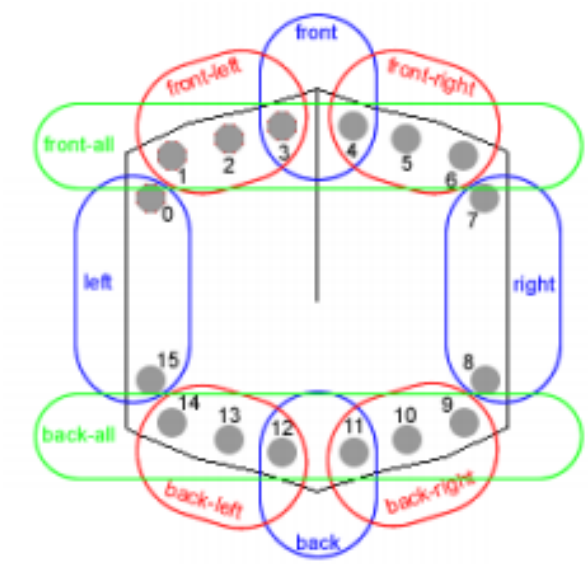

Figure 3: The sensor groups used in the experiment were front and back for sonar as well as front-left and front-right for the light sensors

both cases, q-learning will be done with the q-table and the artificial neural net implementation.

In the constrained (simple) representation of the environment, the state is represented as a 4-tuple (w, x, y, z). The variable w is associated with the light values. If the reading in the left light sensor is higher than the one on the right, w is given a value of 0. If the right light sensor is higher, w is set to 1. Finally, if both light sensors are equal, w is set to 2. Since the pyrobot light sensors are highly sensitive, we have defined equal to mean the light values are within 0.01 of each other. The next variable, x, is a binary variable which monitors proximity to an obstacle in front of the robot. The sonar sensors are used to determine this. If the front sonar sensors report a value less than 0.5 (meaning an obstacle is close), x is set to 1. The same applies for y except that it refers to the rear sonar sensors. The final variable, z, is a binary variable that is activated if the light sensor values have not changed in the last 5 consecutive time steps. Again, due to the sensitivity of the light sensors, the values are deemed to have not changed is they are all within 0.01 units of each other. The entire state space is therefore made of $3 \times 2 \times 2 \times 2$ combinations of the percepts, or 24 unique states.

Percept	0	1	2
Light Readings (w)	L>R	R>L	R=L
Obstacle in Front (x)	N	Y	
Obstacle at Rear (y)	N	Y	
Light readings not changing in last 5 time steps (z)	N	Y	

Figure 4: The different percepts that are used to define a state in simple representation

In the rich representation of the environment, things are a bit different. Instead of using a 4-tuple to represent the environment, we use a 6-tuple in order to capture more information about the environment hence making the representation more complex. The 6-tuple (u, v, w, x, y, z) is constructed as follows: u represents the left light sensor values. Since the light values go between 0 and 1, we decided to descretize these values by demarcating them into 5 possible ranges: 0-0.2, 0.2-0.4, 0.4-0.6, 0.6-0.8 and 0.8-1.0 with the values for u being 0,1,2,3,4 respectively. v represents right light sensor value with the same setup as u. Next, w, x and y are binary and represent the presence of an obstacle in front, to the right and to the left of the robot respectively. The final variable, z represents whether light readings have not changed in the last 5 time steps. Notice that proximity to an obstacle at the rear of the robot was omitted since the robot was not allowed

to move backwards in this part of the experiment. We removed the robots ability to move backward because it drastically slowed the learning process by making repeated forward and backward movements. In this case the entire state space is made up of $5 \times 5 \times 2 \times 2 \times 2 \times 2$ different combinations or 400 unique states.

4.3 Actions

For every time-step, the robot is capable of finding the reinforcement value that it is awarded for performing a specific action in its current state. In order to follow a completely optimal strategy, the robot will have to choose the action with the highest q-value in any given state. When the q-value function has converged to the true payoffs for the state and action pair, then this policy is optimal. In the early stages of learning however, this is not the case and as such it is necessary to encourage exploration by choosing actions that do not necessarily yield the highest reinforcement values. Additionally, an effective exploration strategy is also necessary because if the strategy is inefficient, it could lead to an exponential increase in time and solutions length [3]. One way to achieve this is to use the Boltzmann probability distribution [6].

$$P(a|x) = \frac{e^{Q(x,a)/T}}{\sum_{b \in A} e^{Q(x,b)/T}} \tag{2}$$

where T balances the effect of exploration vs exploitation with a strong bias towards exploration during the early stages of the learning curve. Gradually this is reduced as the experiment progresses so as to favor exploitation.

The actions the robot can take in any given state are represented as a 2-tuple (r, l) which represents right and left motor values respectively. The total action space is shown in figure 5 below.

Action	Motor Commands (L,R)
0	(stop, stop)
1	(stop, fwd)
2	(stop, rvs)
3	(fwd, stop)
4	(fwd, fwd)
5	(fwd, rvs)
6	(rvs, stop)
7	(rvs, fwd)
9	(rvs, rvs)

Figure 5: The entire action space for the robot

4.4 Reinforcement Values

The core of the entire q-learning algorithm revolves around the premise of a reward scheme. In order for q-learning to function correctly, it is imperative that the robot be able to determine the approximate value of an action immediately after performing the action. A successful reward scheme for the light finding task is one that awards the robot for getting closer to areas of brighter light thus implicitly motivating the robot to find the light [7].The following is a possible reward scheme for this problem.

Since the goal is to find the light, it makes sense to reward the robot for actually getting to a light source. As we are using pyrobot light sensors for this experiment, getting to the light is defined as the instance when either (or both) of the light sensor readings registers a value above 0.95. This works because the brightness of the light source was set to 1.0 in this experiment. For reaching light, the robot is given a reward of 3.

Approaching the light is also a desirable control

sequence. In the rich representation of the environment, any approach to the light was given a reinforcement value of 2. For the simple representation of the environment however, the value of the larger current light reading minus the larger reading in the last time step was given as the reward. This method works well in that provides positive reward when the robot approaches a light source, and negative reward in instances where it moves away. We avoided having a similar scheme for the rich environment as we felt this complexity might impede the learning speed of the robot and as such we resorted to using a static reinforcement value of 2 in the rich environment.

Since the goal of this experiment is to get to the light source, it is implicit that the robot be able to avoid obstacles so as be successful in reaching the light. If at any point the robot collides with the walls either at the front or the rear, a negative reinforcement value of -2 is given.

It is also necessary to motivate the robot to continuously move (towards the areas of brighter light). If for any reason the light values remain the same over 5 consecutive time steps, the robot is given a negative reward of -2.

5 Experiments

5.1 Q-table Implementation

Under this implementation, the objective is to build a table for storing the q-values. This implementation is suitable for small size problems, as it may take really long time for the q-function to converge if the problem is exceedingly complex [4]. With this implementation, storage size is a function of the state and action space. For the simple environment, we have 24 states $\times$ 9 actions resulting in a table with 216 entries. In the rich representation of the world, there are 400 states $\times$ 9 actions, producing a table with 3600 entries. Incidentally, we propose that the task should be more difficult to learn in the rich environment since its q-table is significantly larger.

	Actions								
State	[0,0]	[0,1]	.	.	.	.	.	.	[-1,-1]
(0,0,0,0)	-0.01	0.11							0.1
(0,0,0,1)	0.03	-0.13							-0.1
(0,0,1,0)	0.12	0.09							0.14
.									
.									
.									
.									
.									
.									
(2,1,1,1)	-0.04	0.01							-0.03

Figure 6: Q-table just after initializing

The learning procedure for the robot when using the table can be described in the following steps:

1. Initialize entries to random values in the interval -0.15 to 0.15.
2. Place the robot at a random location in the environment.
3. Get current state
4. Determine an action according to equation 2.
5. Perform the action and determine the new state.
6. Get the associated reinforcement values and update the table according to the update rule in equation 1.
7. Repeat 3-5 above until agent gets to the light.
8. Repeat 2-6 above for the total number of runs during the experiment.

The action-perception loop can be summarized by the following diagram:

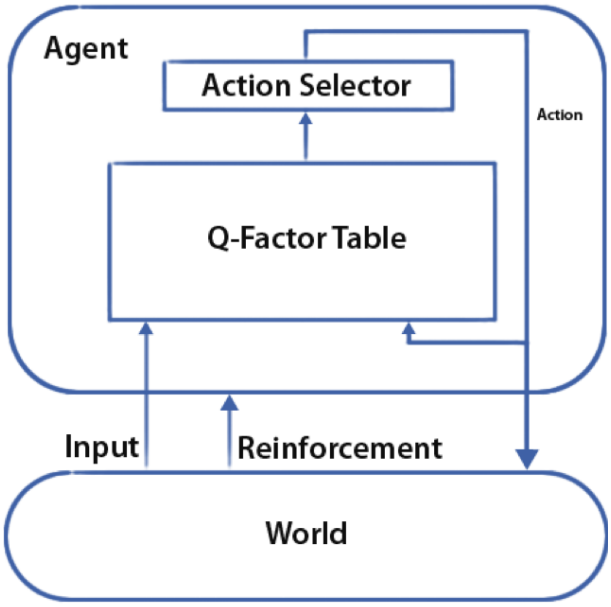

Figure 7: The action-perception loop

5.2 Neural Net Implementation

The q-table implementation of q-learning is favorable for its simplicity of implementation. However, as the state-action space becomes more complex, it becomes increasingly difficult to train and update the table as the environment is explored. Furthermore, if the environment is very complex, then the size of the state-action space may become prohibitively large. In order to overcome these problems, q-learning can be implemented using a neural network as a function approximator for q-values [3].

For the purposes of this experiment, we are going to use the conx library provided as part as pyrobot. For the simple environment, a neural network with 6 input nodes (4 for the state representation and 2 for the motor actions), 4 hidden nodes and 1 output node is ideal. For the rich environment, we use a network with 8 inputs nodes (6 for the state and 2 for the motor action), 4 hidden nodes and 1 output node. The network accepts a state and an action as input and outputs the approximated q-value for that state-action pair.

Additionally, it is important to note that the neural network only accepts values in the range of 0.0-1.0 hence it is necessary to scale some of the inputs to the network to fit in that range. Furthermore, the motors require instructions in the range of -1 (full reverse) to 1 (full forward). The function $f(x) = (x + 1)/2$ is used to translate values into the necessary range for this purpose. Lastly, the reinforcement values also need to be made neural network friendly. Mapping the reinforcement values of -2, 0, 2, 3 to 0, 0.5, 0.75 and 1 respectively seems to work for this experiment.

The learning procedure using this implementation can be described as follows[5]:

1. Initialize the neural network
2. Place the robot at a random location in the environment
3. Obtain the current state
4. Obtain $Q(x, a)$ for each action by substituting the state and action pairs into the neural net, keeping track of those values.
5. Determine an action according to equation 2, so as to be able to balance between exploration and exploitation.
6. Make the move and get new state
7. Get the associated reinforcement values for taking the particular action in the previous state
8. Generate $Q^{target}(x, a)$ according to equation 1and use Q^{target} to train the net as shown in fig 8 below.

9. Repeat 3-8 until the robot finds the light.

10. Repeat 2-9 for the number of runs of the experiment.

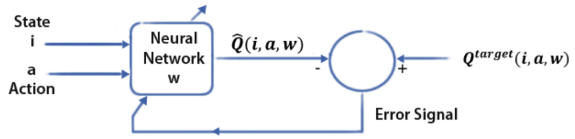

Figure 8: Neural net layout for approximating the target q-values

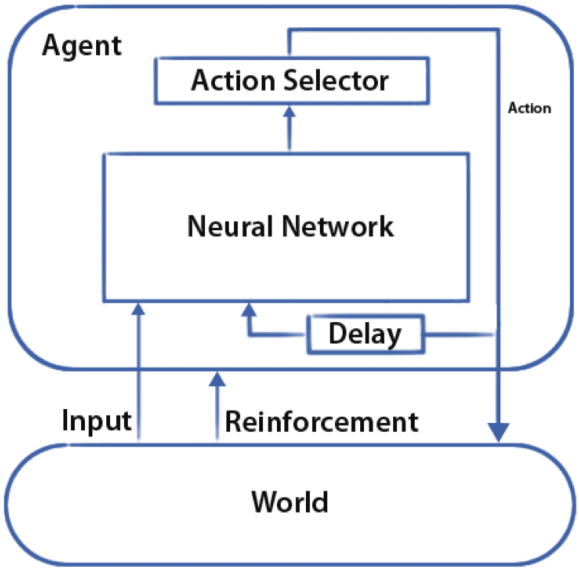

Figure 9: The action-perception loop

6 Results

Our goal was to encourage the robot to learn a control strategy to solve the light finding task. In order to achieve this, we allowed the robot to wander around in the environment until is appeared to have learned a successful approach to the problem.

In as far as the reward schemes for the two implementations are concerned, there were slight modifications we implemented into the simple environment experiment that did not exist in the complex environment. As stated earlier, the robot was given a reward of 3 for getting to the light and a reward of 2 for inching closer to the light. In the simple environment experiment, we altered this scheme so that for getting to the light the robot was given a reward of 3 but for getting closer to the light the reinforcement value was given as current light reading - previous current reading. The advantage with this metric is that is has the ability to reward the robot as well as punishing it for getting further away from the light by giving it a negative reinforcement.

We were only able to identify the new reward scheme after having run the experiments on the complex environment, and as such a fair comparison of the progression of reward between the two environment representation is therefore not available at this time.

6.1 Complex Environment

For the complex environment, the results were not very promising. We ran the experiment for 5000, 10000 and 30000 epochs and in all instances it was apparent that the robot had not learned how to successfully perform the light finding task. Both implementations were not able to achieve a success rate of more than 30%. It was interesting to note that even though the experiment did not go as expected, the neural net implementation appeared to perform better than the table implementation. This is probably due to the fact that such an immensely sized table is difficult to populate. However, we believe that if we ran the experiment indefinitely, we might have achieved better success rates.

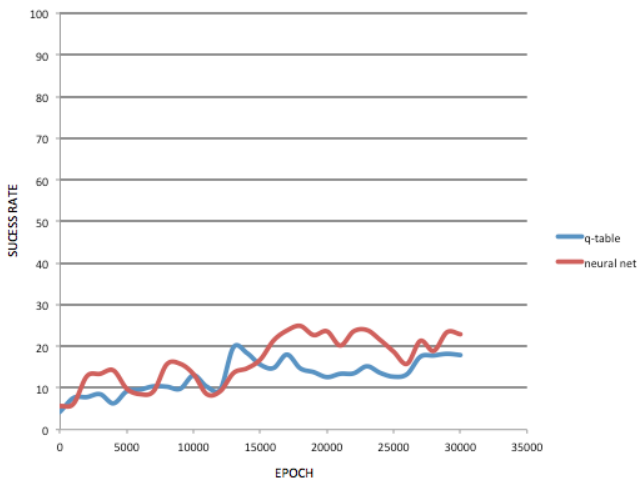

Figure 10: After 30000 epochs of learning in the complex environment the task was still not learnt

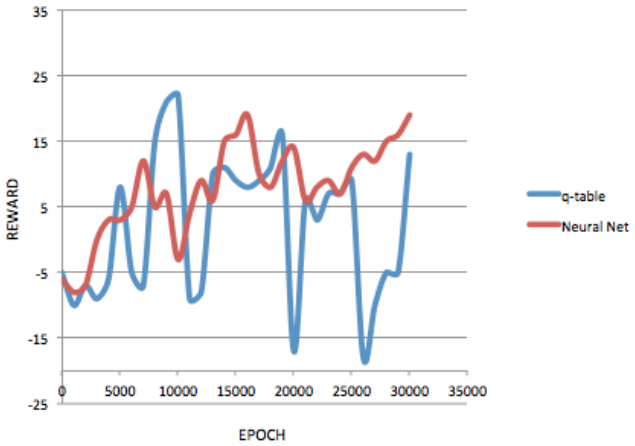

Figure 11: After 30000 epochs the progression of the total acquired reward was hard to interpret

The progression of accumulated reward over time also did not appear as anticipated. The progression for the neural net implementation appeared to be more stable, an indication that learning might have been going on more smoothly albeit slowly. The progression for the table implementation was very haphazard an indication that the robot was having a hard time to learn the task.

6.2 Simple Environment

For the simple environment, the results were more promising. After only 3000 epochs (compared to 30000) for the previous experiment, success rates were going up to upwards of 80%. In this experiment, the neural net implementation performed more poorly than the table implementation. The reason for this might be the simplicity of the environment resulting in a smaller table, hence the ease with which the q-table values converged. We feel that due to the disproportionate allocation of time to the different states, the neural net might have become overtrained on some states rendering it useless to some states it might not have seen as often.

As a result of the difference in the reward schemes between the two experiments, we felt it unfair to compare the progression of accumulated reward over time for the two implementations hence we decided to omit the reward results for this experiment for that very fact. We hope in due course, to perform another set of experiments with the reward functions being similar so that the analysis between of the reward progression over time can actually be performed.

Figure 13 shows the allocation of time spent in each state by the robot. It is interesting to note that the robot spent a great deal of time in state $(1, 0, 0, 1)$ and $(2, 0, 0, 1)$. Both of these states have the property that the light values had remained constant over the last 5 time steps. At first, we were shocked by these results but on closer inspection we realized the robot would sometimes get in situations

where it would move forward and then immediately backwards and do this repeatedly until it changed its sequence of moves. We are not entirely sure why this was the case but we can confidently state that this didn't impede the learning process as shown by our results. State $(0, 0, 0, 0)$ was the next most common state This was the state where the light was on the left, away from walls and light values were not repeating. This was in line with our expectations as were anticipated that the robot would be in this state a lot of the time.

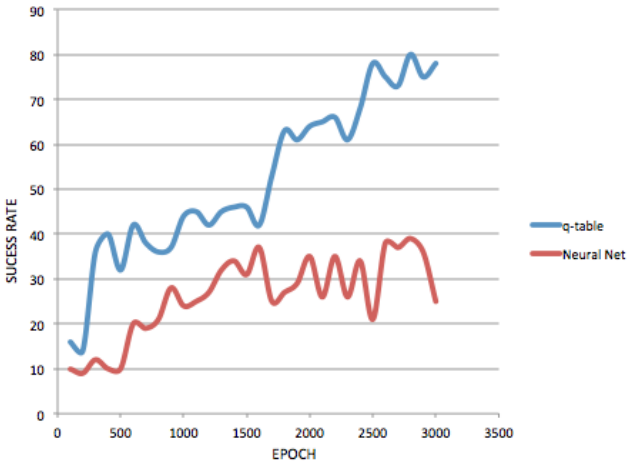

Figure 12: After only 3000 epochs the table implementation was doing pretty well

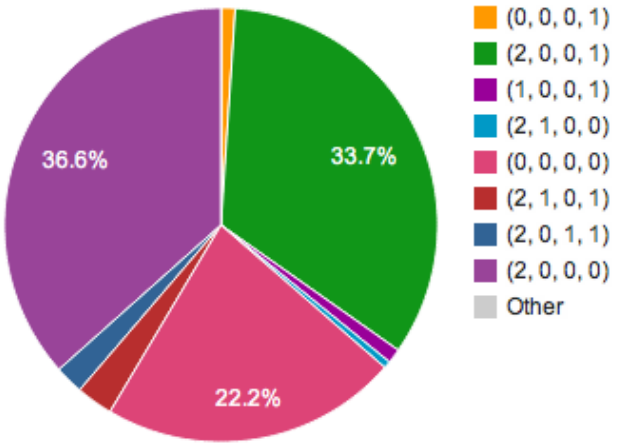

Figure 13: Simple Environment: Distribution of the time spent in each state

7 Summary and Conclusions

In this paper we have shown an implementation of q-learning using both a neural network and a traditional q-table across simplified and complex representation of the environment to solve the light finding task. The results showed that q-learning in general is not a viable approach to machine learning when the environment is very complex and in such instances, other methods might work better. The results also show that the more complex the environment gets, the better the neural net implementation does over the q-table implementation. We anticipate that the difference between the two approaches in the complex environment might only become apparent after more than 30000 learning iterations. The results also show that in simpler environments in which the state-action space is containable, a neural network implementation of q-learning is not the best approach and a traditional q-table implementation is probably better.

8 Future Directions

There are many aspects of the neural net implementation that can be profitably improved, many of which could not be done simply due to the timeframe in which we had to complete the project.

There exists literature that discuses using linear activation functions with regards to neural networks[6]. This might prove to be an interesting ex-

tension as it would allow for a more continuous representation of the q-values. We expect this to work better than the sigmoid function we used in this experiment.

As far as the neural net structure is concerned, one possible extension might be to have a neural network for each action rather than having a universal neural net that works for all actions [6]. The obvious advantage to this is that since the net has to be trained on fewer cases (1/9 of the current), training should inherently be faster in this system. The system will also be immune to the problem of overtraining that might exist in the current implementation in the event that the net is subjected to multiple consecutive encounters with the same action.

A large percentage of related work on this subject also included some notion of using a recurrent Elman net in their implementation rather the standard back-propagation feed-forward network we used for this experiment[4][5]. While there has been widespread use of this implementation, we are not sure and cannot speculate on what the potential advantages of doing it this way might be.

References

[1] C. Gaskett, D. Wettergreen and A. Zelinsky, *Q-learning in Continuous State and Action Spaces*, Lecture Notes in Computer Science, Vol 1747/1999, pp 417-428, 1999.

[2] C. Touzet, *Neural Networks and Q-Learning for Robotics*, IJCNN '99 Tutorial, pp 2071, International Joint Conference on Neural Networks, 1999.

[3] L.J. Lin, *Reinforcement Learning for Robots using Neural Networks*, Unpublished, , School of Computer Science, Carnegie Mellon University, 1993.

[4] A. Onat, *Q-learning with recurrent Neural Networks as a Controller for the Inverted Pendulum Problem*, 5th International Conference on Neural Information Processing, pp 837-840, 1998.

[5] C. Y. Liou, *Q-learning with Look-up Table and Recurrent Neural Networks as a Controller for the Inverted Pendulum Problem*, Unpublished, Information Engineering Department, National Taiwan University.

[6] B. Huang, G. Cao and M. Guo, *Reinforcement Learning Neural Network To The Problem of Autonomous Mobile Robot Obstacle Avoidance*, Proceedings of the 4th International Conference on Machine Learning and Cybernetics, Guangzhou, 2005.

[7] B. Bagnall, *Building a Light-seeking Robot with Q-learning*, http://www.informit.com/articles, April 19, 2002.