GNG-Based Q-Learning

Ivana Ng
Sarah Chasins

May 13, 2010

Abstract

In this paper, we present a new developmental architecture that joins the categorizational power of Growing Neural Gas networks with an action policy for discrete states and actions. The result is a robot brain that can choose its next move by associating its current sensor inputs with a particular subsection of the possible input vectors. GNG networks are used for vector quantization, to generate a set of input prototypes and a separate set of output prototypes. A Q-learning process that treats the input model vectors as states and the output model vectors as actions yields a Q-table that should efficiently guide a robot through the state-action space on which it is trained. We apply this reinforcement learning system to a Pioneer robot in simulation, operating in a simple maze in which the goal is to reach a stationary light located in a far quadrant of the world. We examine the effects of varying the maximum permissible error in the category creation, and of varying the number of steps for which we run Q-learning. Additionally, we investigate two methods of rewarding goal-finding behaviors in the Q-learning algorithm.

1 Introduction and Related Work

Robot navigation is a fundamental goal of adaptive robotics. Without the ability to explore its world thoroughly and fruitfully, a robot cannot accrue experiences to categorize the world in an accurate way. This greatly hinders any potential open-ended learning models with which we might want to train a robot. Localization is perhaps an even more fundamental issue than navigation. Localization is defined as the ability to determine one's location within an environment, and it is currently a highly active field of research [2]. Tingle et al discuss the difficulties of localization and the noisiness of sensor data:

> In theory, a robot could integrate its motor outputs to determine its location, but in practice small errors in the motor outputs lead to large errors in position. Additionally, using sensor inputs to localize can lead to the problem of perceptual aliasing: multiple distinct locations in an environment might give indistinguishable sensor inputs [6].

We approach the problem of localization through the lens of Q-learning, a reinforcement learning technique in which the robot is able to walk through its environment and learn the consequences of its actions. We approach the issue of robot navigation by hybridizing GNG and Q-learning. Q-learning uses the concept of schemas, which originates from psychology and neurology, to configure a "control system that continually monitors feedback from the system it controls to determine the appropriate pattern of action for achieving the motor schema's goals" [1]. According to Arkin, "motor schemas serve as the basic unit of behavior specification for the navigation of a mobile robot" [1]. As a result, we have developed GNG-based Q-learning (GBQL), which uses GNGs to inform the motor schemas developed by Q-learning. In other words, GBQL is a developmental architecture that applies categories formed by GNG networks to a Q-learning table, which the robot then uses to determine its plan of action in a given environment. First, we provide an overview of Q-learning and a brief discussion of related works. Then we introduce GBQL, our hybrid approach to robot navigation. Next, we describe the experiment, which was executed in simulation. Finally, we present and analyze the results and discuss future directions.

1.1 Q-Learning

Q-learning is a reinforcement learning model that allows the robot to learn from the consequences of its actions. By comparing the expected utility of taking a given action in a given state, which is determined by the rewards and penalties it receives, the robot decides the best action policy to follow. Developed by Watkins in 1989, "it provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains" [7]. Markov localization is a "technique to globally estimate the position of a robot in its environment? [It] uses a probabilistic framework to maintain a position probability density over the whole set of possible robot poses" [8].

The Q-learning algorithm assigns rewards and penalties for taking a certain action in a given state. The value of a state-action pair (Q(s,a)), or Q-value, represents the consequences of following an optimal action policy and is calculated by considering the expected future payoffs from taking subsequent actions in that policy. The optimal action from any state is the one with the highest Q-value. These values and their corresponding state-action pairs are stored in a table. Q-values are initialized to zero, and then they follow the algorithm [8] below:

1. From the current state s, select an action a. An immediate reward r is assigned, and the robot moves into a new state s'.
2. Update the table entry for Q(s,a) as follows: Q(s,a) = Q(s,a) + x * r + $y$max(a') - Q(s,a), where x is the learning rate, y is the discount factor (0 ¡ y ¡ 1), and max(a') is the maximum value of Q(s',a'), i.e. the maximum future value. The discount factor determines the importance of future rewards. If the discount factor is 0, the robot will only consider current rewards. If the discount factor approaches 1, the robot will work toward a long-term high reward.
3. Repeat 1.

Note that there is a degree of random exploration in Q-learning that can be manipulated. The higher the random exploration factor, the more likely the robot will choose a random action instead of the action with the highest Q-value. This ensures that the robot will learn about all possible state-action policies., rather than what seems optimal from its current state.

1.2 Growing Neural Gas

Fritzke's Growing Neural Gas (GNG) [3] is a neural network model that can be used for clustering or vector quantization. It is a modified version of Martinetz and Schulten's neural gas (NG) mode, which takes a fixed number of units to create a topology of an input space [4]. GNG is incremental, which allows the network to continue learning by adding units and connections and means that a pre-specified network size is no longer necessary. Instead, a GNG network stops growing when a user-defined performance criterion or network size is met. GNG is an improvement over NG because it is able to continue to grow to discover still smaller clusters and thus makes more explicit the important topological relations in a given set of inputs.

The GNG algorithm categorizes units by forming edges among them [3]. First, two random units are selected and connected by an edge for each input signal j generated. Then the two nearest units, $s1$ and $s2$, are determined. The ages of all edges connected to $s1$ are updated. (Note that the age of edge is calculated based on Euclidean distance). The squared distance between $s1$ and j is added to $s1$. Then $s1$ is moved toward j and each of $s1$'s neighbors is moved slightly toward j, by a fraction of the total distance between them called the error in $s1$. The error in each unit is constantly updated, and edges are removed from the graph when they reach a certain user-defined age. Every j steps, a new unit is added between the unit with the highest error and its neighbor with the highest error.

In GNG-based Q-learning, we use equilibrium GNG, a variation developed by Provost et al [5] in which new units are added only when necessary. In this algorithm, a new node is only added when the average error of all units in the graph is above some user-defined threshold.

1.3 Related Works

Tingle et al's Maze Solving by Learning State Topologies (MSLST) [6] algorithm is specifically designed for maze localization and solving. First, the robot wanders through the maze randomly and creates a set of GNG states. These GNG states are then used to create a graph, or map, that represents the maze. The robot matches "its current and past sensory states and actions to part of the graph, and then follow[s] the actions encoded in the graph that lead to the nearest destination node" [6].

Like MSLST, Provost et al's Self-Organizing Distinctive-state Abstraction (SODA) [5] algorithm also creates a map of the environment. SODA uses GNG to quantize sensory vectors, and then feeds these into a self-organizing map (SOM). The robot uses the SOM to learn a set of reusable motor routines, called trajectory-following and hill-climbing, to navigate the world.

1.4 GNG-Based Q-Learning in Comparison

GBQL combines GNG's ability to categorize sensorimotor data in a relevant way and Q-learning's schema-based reinforcement learning model to approach robot navigation. Like MSLST and SODA, GNG networks are used to make sense of sensorimotor data. But unlike those two architectures, robot navigation is not achieved by creating a representative map of the environment. Instead, GBQL uses reinforcement learning to teach the robot how to navigate in a given environment.

2 The GNG-Based Q-Learning Framework

Any run of the GBQL must begin with a walkthrough of the task environment. During this stage, GBQL runs parallel GNG networks. To the first network, the system must pass all essential sensor values that will distinguish unique areas of the robot's world. The second network is simultaneously fed whatever outputs (usually motor values of some nature) accomplish the walkthrough. The aim is to be sure that there is an appropriate category for every possible sensory state the robot might encounter, and one for every useful action it might need to take. To this end, it is essential that the walkthrough expose the robot to all possible situations. This should allow the first GNG network to generate nodes for all important features of the world, and should allow the second GNG network to develop nodes for all actions that may be necessary to navigate it.

The GNG networks that result from the above process produce lists of categories, in the form of model vectors. All sets of sensor or motor data that can arise in the world can be placed into one of these categories by simply calculating the Euclidian distance from each, and selecting as representative the model vector to which it is closest. This calculation of geometric distance is important only for the input data, and not for motor outputs. It is only the sensor data that the robot will gather in its world and then identify as part of a predefined state.

Creating the Q-learning table requires only the initialization of two dictionaries, one which stores the model vectors of the first GNG network, and one which stores the model vectors of the second network. These become the discrete states and actions that facilitate Q-learning.

With the table set up, Q-learning proceeds in its usual way. The robot's sensor values are passed in, GBQL calculates geometric distances to identify the closest existent GNG unit, and the robot is treated as being in that state. The Q-learning algorithm selects from among the possible actions. The prototype vector for the selected action is passed to the robot as its motor outputs, and the robot moves in its environment. The entry in the Q-learning table for that state-action combination is updated. The robot, possibly in a new state, or possibly still in the previous one if the nearest GNG node remains the same, passes in a new input vector. The process is repeated.

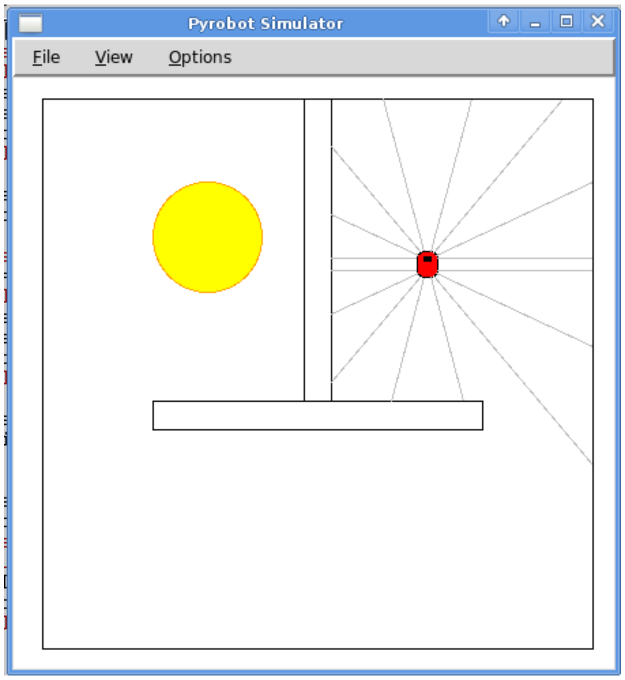

Figure 1: The simple maze used to train the robot.

3 The GBQL Maze Experiment

This experiment took place entirely in simulation, utilizing the Pyrobot simulator. The world was a simple maze, as pictured in Figure 1. The task of the single Pioneer robot in the world was to reach a light inside the maze. The light appeared always in the box at the top left of the environment, as seen in Figure 1.

The inputs selected as essential to situation categorization were three sonar sensors, and one light sensor. The sonar sensors measured the distance from the nearest wall in a particular direction. The light sensor measured the intensity of light present in a particular direction. The sonar values passed to the network were always the minimum value from among the left sensors, the minimum value from among the front sensors, and the minimum value from among the right sensors. The left, front, and right divisions can be seen in Figure 2. The right light sensor was also passed in as part of the input vectors. The outputs passed into the other GNG network were a left motor value and a right motor value, indicating the amount of power passed to each motor.

The walkthrough was a simple loop through the world, beginning in the box at the upper right. It traveled up around the small box, down into the lower half of the environment, up to the light, then around the light until it faced the left wall. During this portion, it made only right turns, between periods of moving in a straight line. Once it reached this stage, it turned left and followed the wall until it returned to its initial position. This section of the walkthrough consisted entirely of left turns and straight lines. For this experiment, Equilibrium GNG was chosen as the ideal GNG structure. Recall that Equilibrium GNG differs from traditional GNG in that it does not add new units at arbitrary intervals. Instead, at each step, the average error (that is, the average distance of new vectors from the prototype vectors to which they are matched) is calculated and compared to a maximum permissible error threshold. Higher values result in the formation of fewer categories, with prototypes less representative of some of the vectors matched to them. Lower values lead to a greater number of more specific categories. The walkthrough was run with multiple different values of this maximum error threshold (MET). Eight runs were completed for each of the MET values: 0.3, 0.4, 0.5, and 0.7.

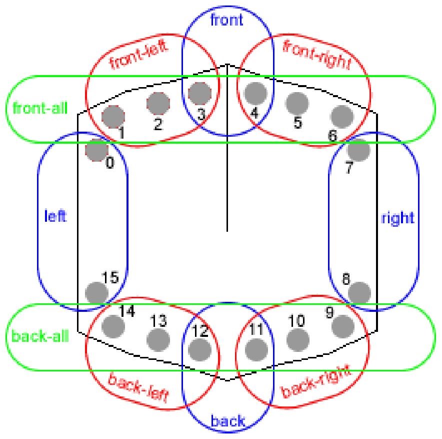

Figure 2: The sensor groups utilized in this experiment were the groups marked "left," "right," and "front."

The resulting lists of GNG nodes were then used as the basis for Q-learning runs. Each pair of GNG networks was used to create four different action policies, each by running a slightly different version of Q-learning. Sets of GNG Networks that were made together were always used together. That is, the sensor categories and motor categories created in a single walkthrough would form the states and actions for one Q-table, rather than being randomly associated with the categories produced by different walkthroughs.

Two main considerations went into designing the reward structure inside the Q-learning algorithm. The central aim of the reward was to incentivize light-finding. Thus, there had to be a way to determine which state (or states) should qualify as target states. The ideal would have been to ensure the existence in each GNG of a unit that specifically represented the light, or to create one afterwards. However, since this would have subverted the underlying goal of automated categorization, that option was unavailable. Instead, the key was to select existing units which could qualify as destination states. The most effective criteria was one which gave a +1 reward for entering any state for which the prototype vector had a light sensor value higher than 0.5. This was the only positive reward distributed.

However, there were other secondary goals. Most pressingly, it was preferable to train a robot that would avoid stalling. To achieve this, a punishment was added to the reward function. If ever the robot indicated that it was stalled during Q-learning, the reward for that time step was -1. The robot would then be randomly relocated. Circumventing the usual requirement that a punishment be associated with a state necessitated checking the current simulation instead of the state model vector. This ensured that all stalling behavior was punished. A rule of this nature should lead to entries in the Q-learning table indicating, for example, that for all nodes with small left sonar values, turning left yields a negative payoff. Except during exploratory behavior, this prevents the robot from choosing those actions which would lead to stalling.

A second treatment approached rewards differently. Stalling corrections remained the same. However, the reward was distributed without regard to the state, by simply checking the robot's light sensor, and giving a reward if the light sensed was over a 0.5 intensity. This treatment was conceived as a safety treatment, because of the fact that three sonar values and only one light value constituted the input vectors. It was possible that the light value would have so little weight that a robot near the light would still find a closer model vector than a high-light-intensity node, because the sonar values would be so similar. With that in mind, it was important to design a version of GBQL that accounts for the possibility of a single model vector being used as the closest node in multiple locations. For clarity, we will call this treatment the Target Intensity Treatment, and the traditional method the Target State Treatment.

One of the central characteristics of the GBQL version of Q-learning is its method of exploring the environment. After thirty steps, if the robot has not stalled, the robot is randomly replaced in the world.

Average Number of GNG Categories		
GNG MET	Average Number of State Categories	Average Number of Action Categories
0.3	14.4	8.0
0.4	11.8	6.6
0.5	9.8	5.3
0.7	8.0	4.4

Figure 3: This figure shows the average number of state and action GNG categories created, as determined by the maximum error threshold value.

Its initial location is always a location near the light, so that it can begin with knowledge of reward states. However, Q-learning's tendency to lead to exploration of a single area - especially in a world in which the robot may move for many steps without ever leaving a single state - made randomized placement the most effective method of ensuring that the robot is exposed to all available rewards and punishments. While this description of the random replacement and start location is clearly tailored to the particular task on which this experiment tested the architecture, it is trivial to extend the modification to any other GBQL task.

Q-learning was run with a discount factor of 0.9, a learning rate of 0.3, and a rate of exploration of 0.5. Each pair of GNG networks was turned into an action policy in each of three ways. First, Q-learning was run for 500 steps, with the Target State Treatment. A second version of the action policy was created by running Q-Learning for 1500 steps with the Target State Treatment. A third version was created by running Q-learning for 500 steps with the Target Intensity Treatment. The fourth action policy was created by running Q-learning for 1500 steps with thet Targe Intensity Treatment.

After the creation of these action policies, each action policy was tested five times. For these runs, the robot's actions were dictated strictly by the action policy, with all random exploratory behavior removed. Actions were only chosen randomly when multiple actions had the same expected reward for a particular state and that reward was the highest available payoff. For these runs, the robot always started in the upper right box in the world, far from the light. Stalling ended the run, preventing success if it had not already located the goal light intensity. Success was defined as sensing a light value of 0.5 or greater.

4 Results and Discussion

As is evident from Figure 3, the maximum permitted average error did affect the number of categories formed, even in the motor outputs GNG network. The number of actions was generally approximately half the number of states.

This suggests that varying the MET could indeed have an effect on the results of the subsequent Q-learning. Presumably the networks generated with lower METs yield sets of more specific, tailored categories. Whether these smaller and more strictly defined categories would positively or negatively influence Q-learning's ability to make general rules is unclear. However, the implication is that the success of high-MET action policies suggests that the use of more generalized states helps learning. The reverse that the success of low-MET action policies would suggest it hinders learning should also be true.

The first and most crucial observation is that of all the action policies created, only one led to successful completion of the task. That is, only one of the robots could perfectly navigate from the right box to the left without stalling at all, to sense a light intensity of 0.5 or higher. The successful robot came from a set of GNGs with 0.4 METs, and was trained with the Target Intensity version of Q-learning, for 1500 steps.

It is clear from this single victory that GBQL met with only limited success in the area of perfect task-completion. However, from the fact that any runs succeeded, it is clear that this developmental architecture could become a viable option in the future. With some amount of modification and further refinement, it is likely that such a structure could reliably lead to effective action policies. While the full extent of the learning is not reflected in these statistics, it was observable in a more qualitative way. Many of the robot brains in

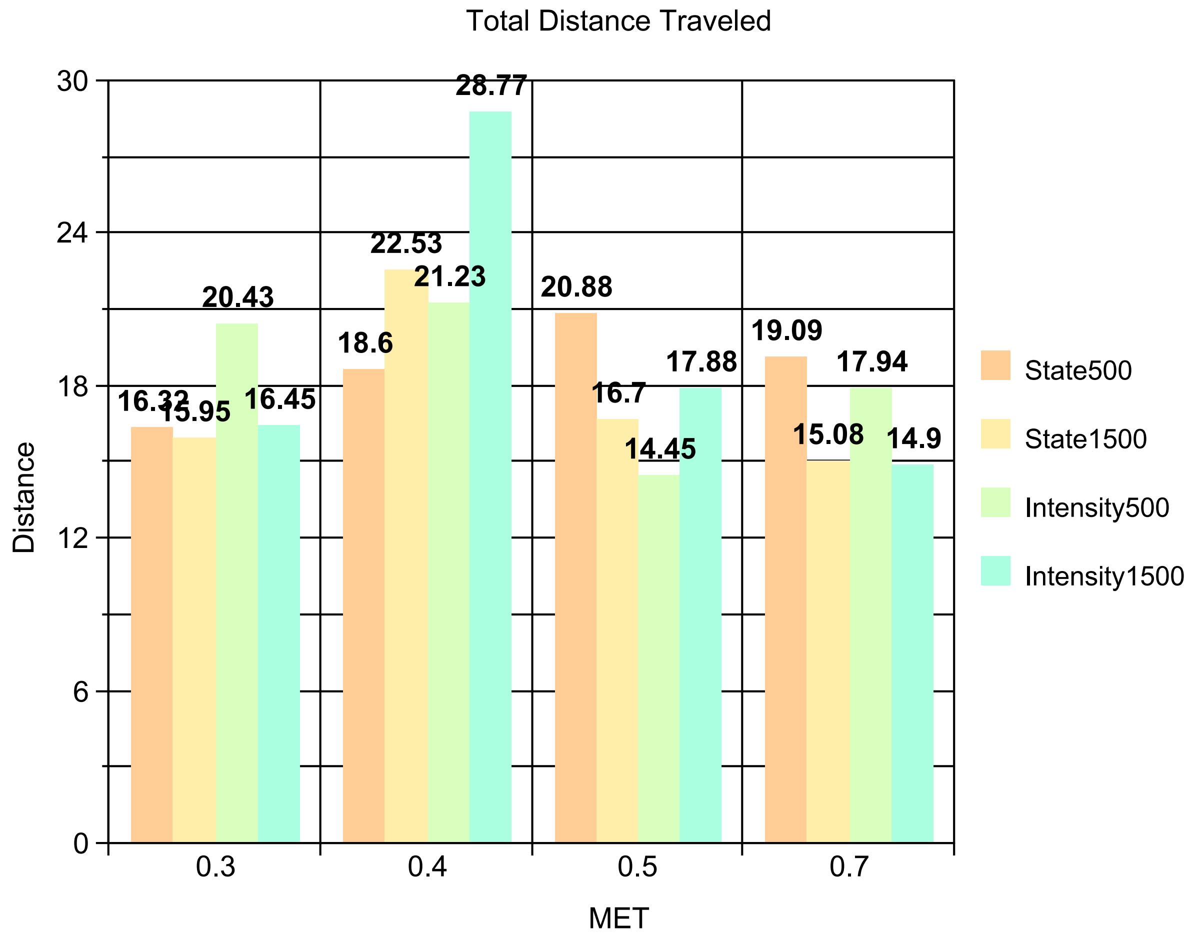

Figure 4: This figure shows the total distance traveled across five trials before stalling or success, by number of steps and treatment (Target State Treatment or Target Intensity Treatment). A higher distance total reflects better performance in the maze, because it indicates that the robot was able to leave the right box. Gaining high distance values generally required progress towards the goal, and roughly corresponded with travel along the goal path. Additionally, gaining high values was impossible without avoiding stalling for some significant period of time.

the categories in the two left columns did extremely well, following a consistent path, and making significant progress towards the light. Only a few robots from the Target State Treatment managed to navigate into the left half of the world. In the Target Intensity Treatment, such a path was common. However, traveling into the left box to sense light proved to be more difficult. Several other trends also appeared. First, GNG networks with medium METs gave rise to better-performing action policies. This could be counterintuitive, since one might imagine that more specific categories would lead to a more precise understanding of what action to take in various scenarios. However, the likely reason behind this finding is suggested by the second trend. When Q-learning was run for more steps, the action policies tended to perform better.

From the fact that the Target State Treatment received disappointing results, it becomes clear that the choice of input vectors, and of the weighting of the various items that constitute them, is essential in the design of the task. The fact that only the Target Intensity Treatment received successful results indicates that the flaw anticipated early in the process - the possibility that even a situation that a human viewer would count a success would fail to be rewarded during the creation of the action policy - did indeed affect the results.

This can be more clearly seen by examining the action policies that resulted from these different Q-learning algorithms. On closer inspection, it is clear that almost none of the action policies produced in the Target State Treatment have any positive Q-table values at all. There are many negative ones, from stalling, indicating that the reward structure is in fact functioning. However, the robot never enters a situation in which it would receive a positive reward. In fact, there are many rows (that is, states) which simply have zero entries for each possible action. This suggests that these states are never, or only very rarely, encountered. In many cases, this includes the vectors which would qualify, under the reward structure, as target vectors, those with light intensity values higher than 0.5. It appears that the robot does not ever find that that vector most closely represents its current state. For this to be the case, there must be other vectors which are geometrically closer. This does not, however, mean that the robot in its simulated world is not encountering a light value of 0.5 or greater. However, because of the high weights of the sonar values, there is a nearer node. This issue could perhaps have been corrected by a more carefully designed input vector. Scaling the light value up by 3 or possibly a little more would certainly have made the light value more powerful in determining Euclidian distance. This would have made it more likely that a step in which the robot is sensing light would select as its nearest neighbor a model vector that includes a high light value. Making the system more sensitive to whether or not each node is a target node would very likely improve the experimental results.

Another solution is, of course, the Target Intensity Treatment. In the Target Intensity Treatment, the robot can develop its Q-learning table in the normal way, developing appropriate actions for each possible combination of sensor values, action preferences that will lead it to avoid stalling, given the physical situation that it faces. However, despite its sensitivity to wall placement and sonar values, it cannot fall prey to the flaw of never achieving rewards despite being in reward-worthy situations. In the Target State Treatment, placing itself into a non-target vector means that it is not recognized as having accomplished anything, and no reward is ever given. In the Target Intensity Treatment, the state structure is irrelevant to rewards, and only relevant for determining the appropriate next action. To determine whether the robot will be rewarded, the system checks the light sensor value from the robot's current situation, and grants a higher payoff if the *true value* is greater than 0.5. Thus, even model vectors that would not have incurred positive entries in the traditional Q-learning setup do receive positive rewards here. However, they do so only if a robot can experience itself as being in that category while in fact being in a goal location. The result is an effective action policy that contains positive entries only for actions that do lead to the goal location and do not lead to stalling, with negative values for actions that lead to stalling, and with zero values for actions taken in states that are rarely or never reached. These GNG state units could essentially be omitted without any effect on the action policy that is formed.

The fact that the greater number of steps in the 1500 run generally led to better results than the 500 run is unsurprising. It indicates that placing the robot in more situations, giving it more training, causing it to explore its environment more fully, leads to an action policy that more accurately represents the consequences in the world. The greater the amount of training, the better the performance in later tests.

Total Timesteps				
GNG MET	500 Steps (State)	1500 Steps (State)	500 Steps (Intensity)	1500Steps (Intensity)
0.3	68.75	65.12	96.25	75.88
0.4	71.62	91	96.25	141
0.5	93.5	93.38	64.38	77.5
0.7	76.63	86.88	86.25	62.13

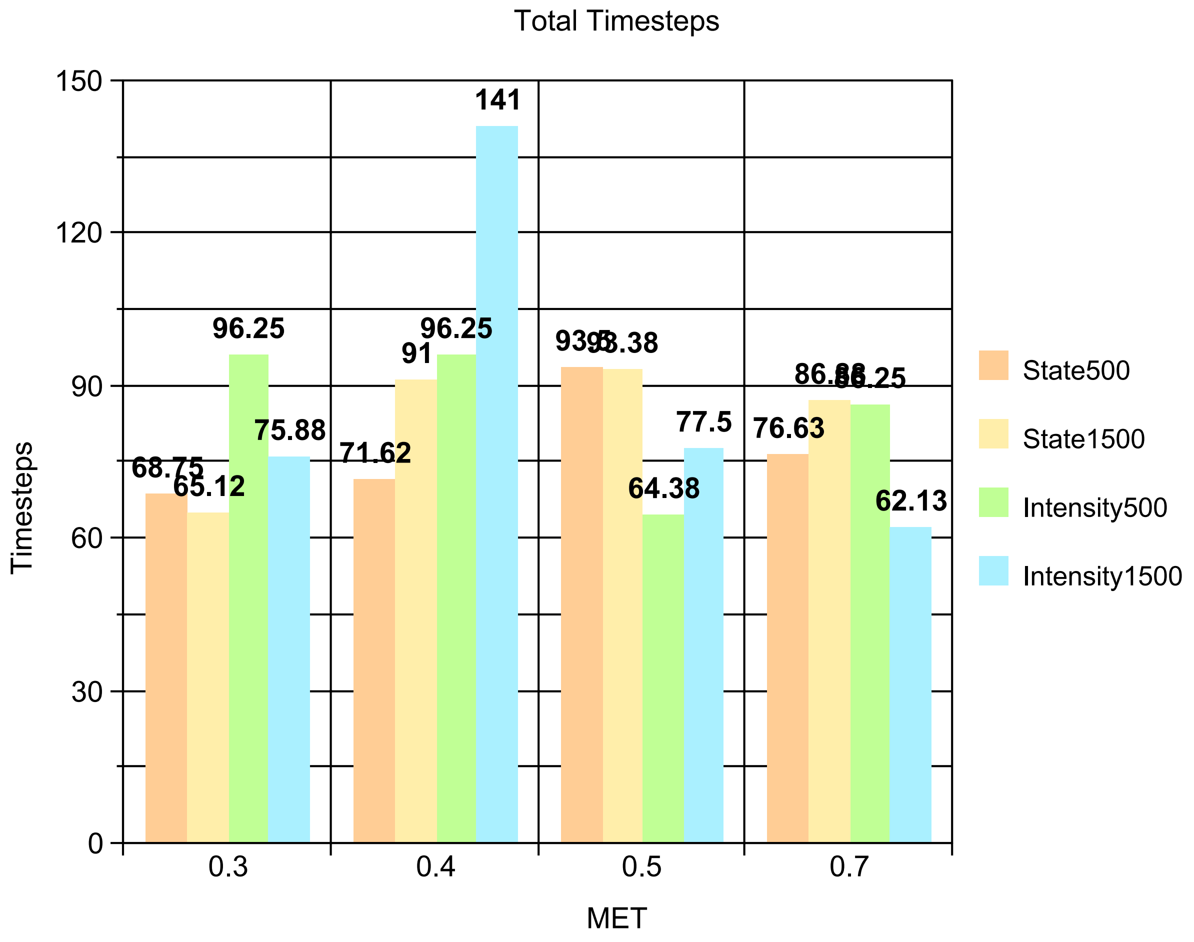

Figure 5: This figure shows the total timesteps accrued across five trials before stalling or success, by number of steps and treatment (Target State Treatment or Target Intensity Treatment). A higher timestep total reflects better performance in the maze, as it is a measure of the amount of time the robot was able to avoid stalling.

All further Q-learning simply improves the robot's knowledge of the simulated world.

The above finding also suggests the likely reason for the worsening of action policy performance as MET declined (and units present in the networks rose). It is probable that the larger Q-tables that stem from larger GNG networks cannot be as exhaustively explored as tables with fewer rows and columns. The various state-action combinations would be revisited less, and their expected values therefore less accurate. Even higher numbers of Q-learning steps could perhaps lead to low MET networks that would also have good Q-learning results. However, the time that such training would take could be a prohibitive factor.

It is not unexpected, in fact, to find that extremely low MET action policies perform poorly when little training is possible. A related concept was, after all, the central premise of this experiment. If one could create a Q-learning table with entries for every possible combination of sensor inputs and every possible combination of motor outputs, the result would of course be the most accurate action policy that could be created. However, the computational power that would be required makes this unfeasible and inefficient. It is for this reason that we use the Growing Neural Gas to form categories. To a certain extent, we want to generalize, to be able to apply learning gained in one area to other similar areas. This is more effective with systems of fewer categories, in which each must represent multiple similar spaces. The fact that the spaces are comparable - a fact ensured by the GNG algorithm - makes it likely that the results in different situations that fall under the same category will be congruent too. With a small table, this can be tested effectively.

On the other end of the spectrum, very high values for the MET, which led to extremely broad categories, were not as useful as some lower values. The reasons for this is likely the over-generalization of behaviors learned in other situatios. Having some greater ability to distinguish between states made the robots better able to handle the peculiarities of a given situation.

5 Future Research

Preliminary results indicate that GBQL is a promising approach to robot navigation and environment exploration. There are, however, still many avenues of exploration open for improvements to GBQL.

5.1 Tweaking Variables

GBQL uses GNG and Q-learning, both of which have several user-defined variables that could change results drastically. In equilibrium GNG, the maximum-error threshold (MET) affects the rate at which new units are added to the GNG network. In our experiments, we looked at four METs, 0. 3, 0.4, 0.5 and 0.7, but we can look at other values and analyze how higher or lower values may affect the number of and the usefulness of the categories formed. A very high MET may increase perceptual aliasing and create too many similar categories, but a very low MET may create too few categories that do not accurately describe the robot's environment. Finding the optimal MET is a difficult but worthy endeavor. In addition, in our experiments, the two GNG networks always used the same MET, which may have caused an imbalance in the number of categories formed in the sensor-GNG compared to the motor-GNG. For example, the average 14.4 sensor input categories created with an MET of 0.3 may be an appropriate summary of the environment, while the 8.0 action categories created by that same MET may be far more than necessary.

In Q-learning, the rate of exploration (RE) affects how newly acquired information affects old information. If RE is low, the robot will not learn anything, but if RE is high, then the robot will consider only the most recent information. Also, the discount factor (DF) affects what type of rewards the robot strives for. If DF is low, then the robot focuses only on rewards resulting from current actions. If DF is high, then the robot focuses on getting a long-term high reward. DF is crucial to the concept of reinforcement learning, but too high a DF could make it a very difficult for a robot to learn a task at all. The optimal RE and DF could largely be determined by the complexity of the environment and the difficulty of the task. This is something that could be further examined in the future.

5.2 Stalling

As seen in our results, the state-action policies derived from GBQL do not always adequately address the problem of stalling. Despite following an action policy developed through reinforcement learning, the robot still stalled in most trials. This essentially brings robot navigation to an abrupt halt. A future direction may be to vary treatment of stalling behavior in the Q-learning algorithm. One possibility is to add stalling as one of the discrete states in the Q-learning table.

5.3 Different Environments

We only tested GBQL in one environment, a simple T-shaped maze with a stationary light in the upper-left quadrant of the world. By testing GBQL in other environments, both in simulation and in the real world, we can strengthen the case for GBQL as a viable approach to robot navigation and exploration.

5.4 Compare to Hard-coding

GBQL is unique in that the states and actions of the Q-learning table are determined by GNG networks, which are built from the bottom up and continually changing. We should compare GBQL to Q-learning in which the states and actions are hard-coded. States could be as simple as the intuitive encodings for being in a corner or being near a wall. Actions could simply be basic motor functions like forward, backward, left and right. We expect GBQL to be more successful at robot navigation than regular Q-learning, because the states and actions of the table are more tailored to the robot's individual experience in the environment.

6 Conclusion

In conclusion, we find that GBQL is a promising mechanism for robot navigation and exploration. The action policies were not always successful, as evidenced by the stalling observed in most trials, but this may be more indicative of the difficulties of localization, which continue to plague the field of adaptive and mobile robotics, than of shortcomings specific to GBQL.

The categories created by the two GNG networks, one of which took in sensor inputs and the other of which took in motor inputs, proved to be quite effective in defining discrete states and actions in the Q-learning table. As a result, the Q-learning algorithm yielded action policies that were precise and adapted to the robot's experiences in the environment. Further development of the system could make it a viable option for future machine learning projects.

References

[1] Arkin, R. (1989). *Motor Schema-Based Mobile Robot Navigation. International Journal of Robotics Research.* 8(4):92-112.

[2] Fox, D., Burgard, W. and Thrun, S. (1999). *Markov Localization for Mobile Robots in Dynamic Environments.* Journal of Artificial Intelligence Research, 11:391-427.

[3] Fritzke, B. (1995). *A growing neural gas network learns topologies.* In Tesauro, G., Touretzky, D.S., and Leen, T.K., (Eds.). Advances in Neural Information Processing Systems 7, pages 624-632. MIT Press.

[4] Martinez, T.M. and Schulten, K.J. (2009). *A "neural-gas" network learns topologies.* In O. Simula T. Kohonen, K. Makisara and J. Kangas, editors, Artifical Neural Networks, pages 397-402. Springer.

[5] Provost, J., Kuipers, B., and Miikkulainen, R. (2006). *Self-organizing distinctive-state abstraction for learning robot navigation.* Connection Science, 18(2):159-172.

[6] Tingle, D., Ball, E. and Augat, M. (2009). *Maze Solving by Learning State Topologies*, CS 81, Spring 2009.

[7] Watkins, C. (1989). *Learning from Delayed Rewards*, Thesis, University of Cambidge, England.

[8] Watkins, C. and Dayan, P. (1992). *Technical Note: Q-Learning.* Machine Learning, 8:279-292.