Modular Neural Network Tasking of Space Situational Awareness Systems

Daniel J. Regan
Ball Aerospace

ABSTRACT

Modular neural networks are evaluated for their ability to automatedly task an SSA system. For this study, an optical, steerable space situational awareness (SSA) system is modeled within a MATLAB simulation and its behavior is automated by a modular neural network. This network-of-networks approach begins with the development of three artificial neural network blocks designed to perform specific tasks essential to the execution of the SSA mission - Search, the discovery of new objects; Reacquisition, the subsequent measurement of an object to refine its state model; and Tasking, the decision to Search or Reacquire at a specific time given the state of the system. These network blocks are trained individually through deep Q reinforcement learning to optimize reward values, which have been designed to maximize the performance of the respective functions. The Tasking, Search, and Reacquisition blocks are then integrated together to form a modular neural network architecture capable of executing a basic SSA mission, with individually optimized functional components. The integrated network can then be retrained as a whole through transfer learning to demonstrate this architecture's ability to adapt and re-optimize to changing mission performance metrics and system configurations. The advantages of this modality are explored in its adaptability and cost savings over traditional, hard-coded algorithms approaches by obviating a man-in-the-loop.

1. INTRODUCTION

Optical, ground-based Space Situational Awareness (SSA) systems are becoming more and more prevalent across the globe as interest from governments, academics, businesses, and amateurs become more and more piqued by the growing population of earth-orbiting objects and debris. With orbital debris numbers on the rise [1], the advent of populous constellations of commercial small satellites soon approaching, and a continuously contested space environment looming overhead, effective SSA has never been so important. In an effort to simplify – or at least optimize – the tasking of the growing number of upward-looking SSA sensors, this study investigates the role that artificial neural networks may play in the future of SSA tasking. Certainly, deep learning networks have proven themselves in their ability to classify images [2], detect changes and anomalies in large datasets, and even learn to perform track correlations and orbital determination (OD) on detected objects [3]. It is the aim of this study to demonstrate that not only can neural algorithms be used for the processing of SSA data but can be trained to optimally task an SSA system without a human-in-the-loop.

This study centers around a simulation environment developed in MATLAB that replicates the essential challenges of the SSA problem. Within this simulation are placed a number of moving targets as well as a commandable observer that can slew across the environment and make observations to detect and catalog these targets. After each observation, the observer requests new tasking instructions, and it is to this interface that algorithms are connected to direct the observer's actions. By commanding the observer, detecting targets within the environment, and tracking their positions and ephemeris errors within an object catalog, we can derive mission performance metrics with which we can measure the effectiveness of the observer (or rather, the algorithms) at performing this SSA-like mission.

This study focuses on the three primary functions of SSA – "search," systematic observations with the intent of discovering new objects; "reacquisition," secondary observations of discovered objects for the purpose of catalog maintenance and error management; and "tasking," the decision at any point in time to execute either the search or the reacquisition function. While there are many more aspects to true SSA in an operational setting, these three functions at least distill the basic functionality of an SSA system to the point where an algorithmic architecture can be evaluated for its effectiveness. These three SSA functions are encoded into a modular neural network architecture, a framework for deep learning networks that separates the different functions of a complex algorithm into individual modules in the same way that neural functions are separated into different structures in a biological brain [4][5].

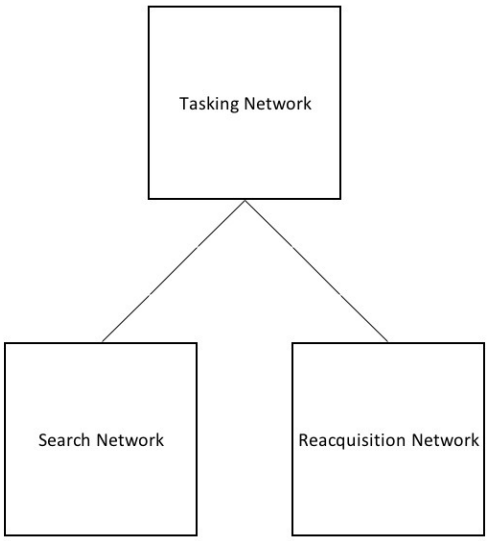


Fig. 1. Modular Network Outline

To test the hypothesis that a modular neural network architecture is capable of driving a ground-based optical SSA system to some degree of optimality, a network of deep reinforcement networks is developed, each module of which is designed to execute one of the three basic functions of the SSA mission. As shown in Fig. 1, the functional flow through these networks begins at the tasking block, which takes as an input the state of the environment and decides whether to execute either the search or reacquisition networks. Following tasking, either the search or reacquisition network is presented state information about the system to specifically direct the observer's movement within the environment. By dividing the functionality of the driving algorithms into three network modules, the training and optimization of each individual network should be simplified and expedited compared to training a singular deep neural network with the burden of optimizing all three functions at once.

Because these networks are trained through reinforcement, they have an inherent ability to re-optimize to new simulation parameters through the concept of transfer learning, with little to no hard-coded modifications. The result is a network of algorithms that can be developed independent of a final sensor design and can adapt to changing system parameters caused by upgrades, failures, or normal degradation. It is this flexibility that makes modular neural network tasking a profoundly unique and advantageous modality for future SSA systems.

2. SIMULATION ENVIRONMENT

To accurately assess the capability of a modular neural network architecture to perform an SSA mission, the algorithms must be exercised within an environment that emulates the basics of an SSA mission. The environment should be a rule-based simulation in which an agent (in this case, our observer) can act based upon its understanding of the environment's state at any given time. For the purposes of this study, the environment must effectively emulate similar characteristics to a real-world space environment such that it presents a challenging but solvable problem for our SSA algorithms without being overly constraining or computationally burdensome. To accomplish this, the following requirements are levied upon the simulation environment:

1. The environment shall simulate the deterministic movement of targets within an observable region.
2. The environment shall provide the agent (observer) with the ability to measure (observe) the position and velocity of targets within a limited field-of-view (FOV) with some nominal measurement error.
3. The extent of the observable region shall be significantly larger than the agent's FOV such that the agent cannot observe all targets within the environment at once, therefore making the accumulation of accurate target information sufficiently challenging for catalog management.
4. The execution of a simulation within the environment shall be sufficiently timely such that thousands of simulations can be executed within a matter of hours. This is essential to allow for comprehensive network training within the limited time period of this study.

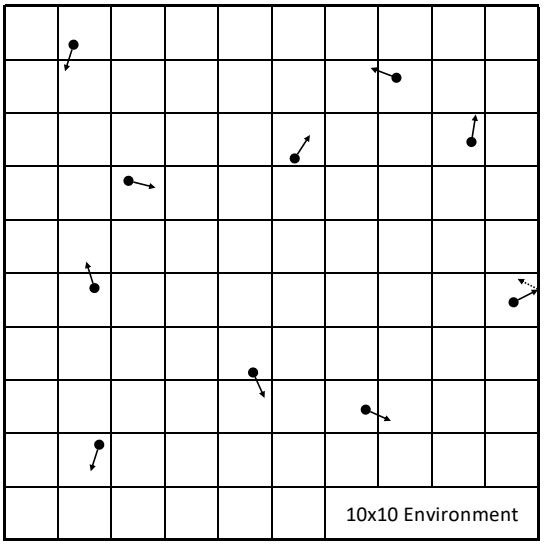


Fig. 1. Simulation Environment with 10 Targets

The environment ultimately used in this study is developed in MATLAB using SAND, a proprietary, object-oriented framework developed by Ball Aerospace for discrete-time based, event-driven simulation and mission analysis. SAND allows for efficient management of interactive objects within a simulation, and therefore provides an ideal backbone for an SSA environment. The environment is comprised of an observable, 10 by 10 unit region, within which 50 target objects are spawned with random positions and velocities. The targets are modeled as 0.2 unit diameter "balls," which travel linearly across the environment until they "bounce" off the edges of the region. The targets do not interact with one another, and the collisions off of the edges of the region are completely elastic. Clearly, this simulated target movement differs from that of orbital objects, but this model is sufficient to meet the needs of this study in that it is deterministic, and therefore predictable.

2.1 Observer

To emulate an electro-optical telescope, an observer is modeled within the environment. The observer has a square, 1-unit wide FOV, and emulates a step-stare sensor with a set slew rate, settle time, and integration time. As it performs observations, the observer is fed positions and velocities of targets within its FOV with a random gaussian error of 0.1 units in position and velocity per axis, 6σ. Target observations are stored in the observer's catalog. Target positions and velocities are then tracked and propagated over simulation time since the previous observation, and sequential observations of a target are compounded to reduce the target error. Note: target identification and correlation are taken for granted in this model – the observer has perfect knowledge of which targets it is observing at any given time. The problem of observation correlation is beyond the scope of this initial study but should be applied in future investigations. For the purpose of validating the algorithm architecture, this simplified problem set is sufficient for this initial study.

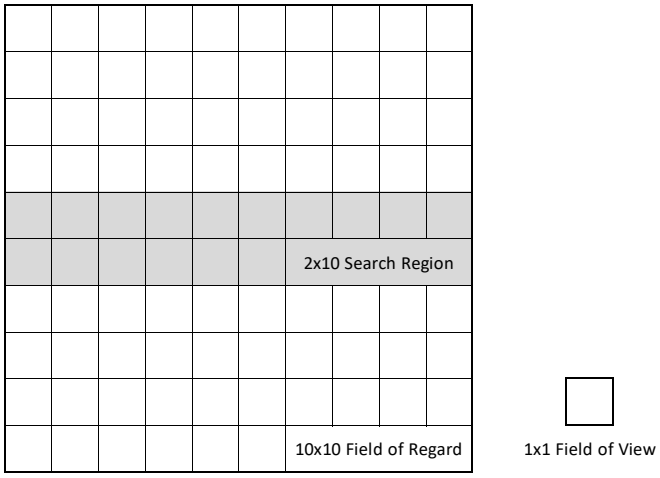


Fig. 2. Observer Proportions

The observer can be commanded to any one of 100 locations on the 10x10 grid of the environment. By discretizing the environment into a grid, the number of possible actions that can be taken by the observer is limited, simplifying the simulation both visually and computationally. As illustrated in Fig. 2, the 5th and 6th rows of the grid are designated as the search region – a 2 by 10-unit sub-region of the environment that can be systematically revisited by the observer for the detection of new targets. The age, or time since last observation of these 20 search region squares, can be tracked as a metric of search performance. Ultimately, it is the goal of the agent to command the observer from location to location to minimize the age of the search region grid while simultaneously minimizing the accumulated position error of cataloged targets. It is the purpose of this study to create a series of networks capable of optimally balancing these two policies to maximize the performance of the mission.

Measurement information about discovered targets is stored in a target catalog by the observer. The target catalog maintains the estimated position and velocity of each target based upon any measurements that have been made and propagates error ellipses from these measurements. The mathematics of orbital determination and covariance is not the subject of this study, so this arithmetic is largely taken for granted. In the name of simplicity, error is modeled as a probability density function that expands over time based upon the specified velocity error of the simulated sensor. Every time the target is reacquired the state error is statistically reduced, resulting in a minimal position error ellipse that expands at a reduced rate due to the reduced velocity error from compounded measurements.

With the simulation environment fully defined, we can begin the development of our SSA tasking algorithms.

3. NETWORK DEVELOPMENT

3.1 Search

The first algorithm examined in this study is search, or the commanding of the observer to discover new targets and minimize revisit time to each search region square. The search function is limited to a 2 by 10 region within the larger environment to emulate the behavior of a ground-based SSA system that searches the GEO belt for new or previously undiscovered objects. Limiting the number of search region squares also puts a cap on the achievable efficiency of the system to perform search – in other words, it makes search performance more transparent as we can compare the algorithm's performance at any given time to a known ideal.

3.1.1 Q-Learning

Before endeavoring to create a deep learning network for the search function, we can validate the concept of machine learning for search within this environment by setting up a simple Q-Learning example. Q-Learning was first published by Watkins for his PhD thesis in 1989 [6] and is a method of reinforcement machine learning that is inspired by the behaviorist theories of psychologists such as John B. Watson and B.F. Skinner [7]. It uses values in a

matrix, Q, to execute a policy that optimizes an agent's behavior based upon the learned values of state-action relationships. These Q values are calculated through experiences through which the algorithm is taught that a certain reward can be expected should it choose to act a certain way given a specific system state. For example, if the observer is located in the bottom left corner of the search region and decides to search a neighboring square, it may receive a large reward. If instead the algorithm sends the observer to the opposite side of the search region, a far less efficient move, it may receive a far lesser reward or even a punishment. Through these experiences, the Q matrix is developed, and an optimal search policy is derived.

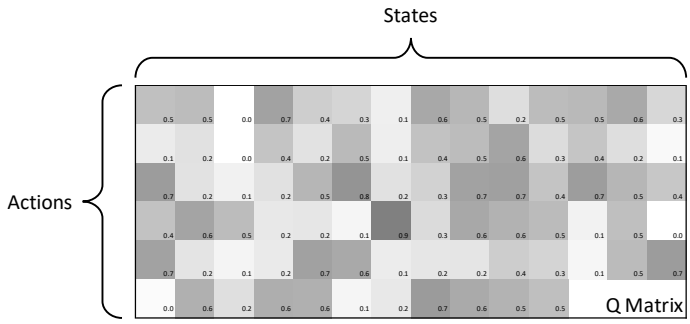


Fig. 3. Example Q Matrix

To implement Q-Learning for the search function, we can encode the states and actions of the search problem into a limited range of possibilities. Actions are simple enough as there are only 20 search region grid locations, and therefore only 20 possible actions the observer (agent) can take. Conversely, the state can be described in a far greater number of enumerations, especially considering that "age" is a continuous value.

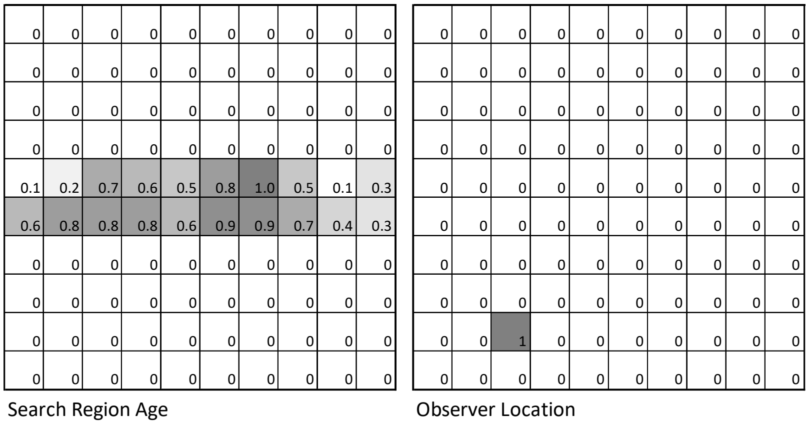
The environment provides 100 unique locations in which the observer may be at any given time. In actuality, there are infinite locations the observer may be at any given time as the observer slews from one location to another, but in this simulation decision making is only executed between movements when the observer is stationary. Decisions for search should also take into account the age of the search region locations, or at a minimum it should include data about the most aged search region locations. Because we need to limit the number of states our system can have for Q-Learning (or else the Q matrix grows too large and training time increases), we need to develop a function that expresses the state of the simulation in a limited number of possible outputs. For this study, search age is encoded as a 1 by 3 vector containing the IDs of the three oldest search region squares in descending order of age. With 20 squares to choose from, this limits the number of possible permutations to 6840. Concatenating this onto the ID of the observer location, the state input vector to the Q-Learning algorithms is a 1 by 4 vector with 684000 different possible states. Since there are 20 different possible actions, the final size of the Q matrix is 20 x 684000.

With the environment states, actions, and Q matrix prepared, the final component needed to create a Q-Learning algorithm is the reward function. To derive a policy, a Q-Learning algorithm uses a reward function to learn the value of choosing a specific action given a specific state. The reward function for the search algorithm should take into account only the reduction of the search region age and should not reward the function based on objects discovered after each action. From one perspective, this creates an algorithm that is robust to the number of objects that are actually discoverable – i.e. it is not the algorithm's fault if there was nothing there to discover in the first place, nor is it the sign of a good algorithm if everything happens to be in the same spot at the same time when the observer decided to look there. It is therefore a much more robust goal for a search algorithm is to have it complete a sweep of all search region locations in the least amount of time. In essence, while we cannot guarantee efficacy, we can enforce efficiency. To teach the algorithm this, rewards were valued based upon the search location age as a percentage of the oldest location's age, minus a factor of the distance the observer had to move.

3.1.2 Deep Q Search Network

The limitation of Q-Learning, especially for our search problem, is the massive loss of information that occurs through the encoding of system states. Deep Q-Learning gets around this problem by allowing for continuous states of far more information using a convolutional neural network (CNN) architecture [8]. Deep Q Networks (DQNs) do not require us to explicitly define states as Q-Learning does. Instead, input data is provided to the network in matrix

form, and the state is interpreted by the convolutional layers of the network, which are followed by fully connected (FC) layers and a regression output layer that use this state information to calculate the Q value for each action.



Search Region Age

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0.1	0.2	0.7	0.6	0.5	0.8	1.0	0.5	0.1	0.3
0.6	0.8	0.8	0.8	0.6	0.9	0.9	0.7	0.4	0.3
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

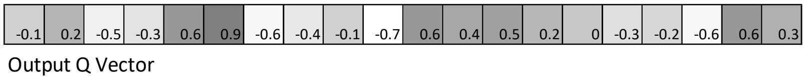
Observer Location

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Fig. 4. Search Network Input

The search algorithm can now be presented with far more input information, allowing it to make more nuanced and potentially optimizable decisions based on a more complete state of the environment. As an input to the network, we define a 10x10x2 matrix of data. The first channel of the input represents the ages of the 2x10 grid, normalized by the maximum, and padded with rows of zeros to make the matrix 10x10. The second channel is a one-hot matrix where the location of the observer within the 10x10 environment has value 1 and all other squares have value 0.

The advantage to expressing the state information in this matrix format is that it preserves spatial information that was lost in the Q-Learning vector input. The Q-Learning network can infer some distance information between the current observer location and a given action based upon the values returned by the reward function, but this not be gleaned from the input data itself. Additionally, quantitative age information is now provided for each individual search region square; in the Q-Learning example we only provided a vector of the three oldest region IDs. This extra information is important when considering the efficiency of the DQN to predict Q values without a formal state definition.



-0.1	0.2	-0.5	-0.3	0.6	0.9	-0.6	-0.4	-0.1	-0.7	0.6	0.4	0.5	0.2	0	-0.3	-0.2	-0.6	0.6	0.3

Output Q Vector

Fig. 5. Search Network Output

The output of the regression network is a 1x20 vector of Q values, where the maximum corresponds to the highest valued action to take. To translate this vector into an action, the index of the maximum value on the vector is tagged to a location within the search region, as shown in Fig. 6 below. It is this ID which is sent to the observer to direct its movement.

2	4	6	8	10	12	14	16	18	20
1	3	5	7	9	11	13	15	17	19

Fig. 6. Search Region IDs

The design of the search CNN, as is often the case with deep learning, is an exercise in trail and error. The DQN framework typically requires a series of convolutional layers, each followed by batch normalization and nonlinear activation layer. The convolutional layer block is then followed by a set of fully connected layers, and finally a regression output layer of the proper size. While a nuanced discussion of the design of the networks used in this study is omitted, I have included outlines for each network such as the one in Fig. 7 below. Note, batch normalization layers are omitted in the table, but occur between each convolutional and non-linear activation layer pair.

Layer	Filter Size	Output
Input	-	10 x 10 x 2
Conv1	2 x 2	9 x 9 x 16
ELU	-	9 x 9 x 16
Conv1	2 x 2	8 x 8 x 32
ELU	-	8 x 8 x 32
Conv1	2 x 2	7 x 7 x 64
ELU	-	7 x 7 x 64
FC1	-	512
ELU	-	512
FC2	-	20
Reg.	-	20

Fig. 7. Search Regression Network Outline

An important point to note, the CNN network outlined above contains no max pooling layers. Max pooling is often used in CNNs to limit the total number of neurons in the network by down sampling between convolutional layers. To avoid the loss of information caused by down sampling, and because the 10 x 10 input size is sufficiently small to avoid long computation and training times, we avoid max pooling altogether.

3.1.3 Performance

The following figure shows the result of reinforcement training through back-propagation of Q values for the search network. Each episode of training is a 300 second (environment time) simulation with random tasking, i.e. a 50% of executing the search function at any tasking event. Reacquisitions are simulated as a move to a random grid location within the environment to ensure that the observer may be located at any grid location in the environment at the time search is executed. If training is run with 100% chance of search, the observer will always remain within the search region, and the network will not be exposed to states where the observer is outside of the search region.

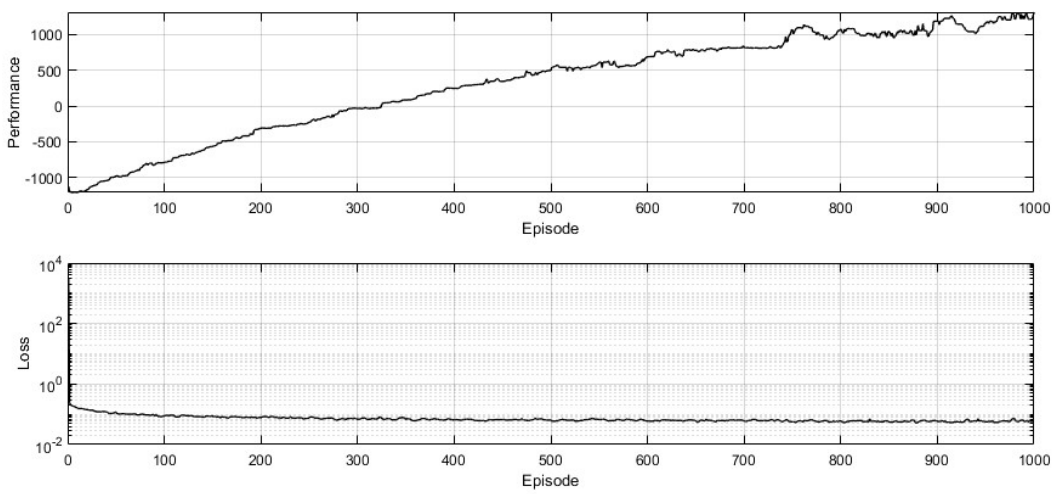


Fig. 8. Search Training Performance Snapshot

As shown in Fig. 8 above, performance (in this case plotted as the sum of rewards over a simulation episode) shows a steady increase over time as the search network is reinforced through experience. This is an encouraging result, as it demonstrates that the search algorithm can optimize itself in a relatively short amount of training time. Note that the noise seen in the performance graph is due to the randomness of the simulation, not an irregularity in the performance of the search algorithm.

3.2 Reacquisition

Following the successful development of a DQN network for the search function, we can develop a network capable of driving the reacquisition function of the mission. Reacquisition in this simulation is a two-step process. First, the agent must consider the observer's current position as well as the current state of the target catalog to decide which

target to reacquire. A first approximation, which might be sufficient, is to always go after the target with the largest accumulated error. Alternatively, one might elect to go after the closest target to the observer to minimize slew time. One would assume the optimal policy would fall somewhere in between these two options. Second, the agent must propagate the selected target's position forward in time, considering its own slew and settle times, to determine which grid location to move to. The grid location is ultimately the command that is returned to the observer. Upon consideration, we elect in this study to frame only the first step of the reacquisition function in a deep learning network, as this is truly where the decision and optimization is made. The second component – that of forward propagation of target motion – is deterministic and does not leave much room for optimization and is therefore hard-coded. This second function should be explored further in future work.

Reacquisition, like search, is an algorithm that decides between several different actions given the state of the system. Therefore, the reacquisition network is similarly well suited to a DQN framework. The reacquisition network developed for this study takes in a stack of 10 x 10 matrices, similarly to the search network. The first 9 channels express catalog state information, and the 10th channel has the observer location as a one-hot matrix like we had for search. The 9 catalog channels are intended to provide information about target positions and their associated error. There are 50 simulated targets within the simulation environment, any number of which may be tracked in the observer's catalog at any given time. To enforce some realism into the development of the reacquisition network, the network is not provided information about all tracked targets. Instead, a preprocessor selects no more than 9 targets from the catalog for consideration. In this study, the 9 targets with the largest velocity error terms (i.e. the fewest previous measurements) are selected. The position and error of each of these targets is converted into a 10 x 10 matrix representing the environment grid. The grid square in which the center of the target error ellipse falls is given a value equal to the area of the ellipse. This therefore captures the spatial information about target positions, as well as the relative error sizes of each catalog entry.

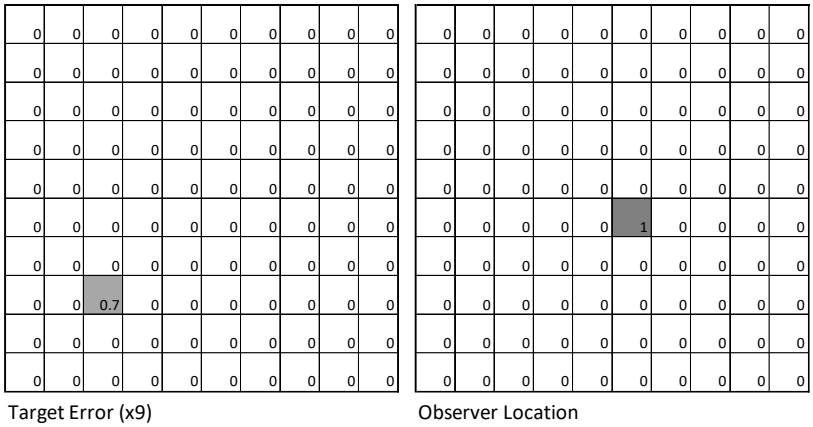


Fig. 9. Reacquisition Network Input

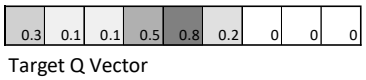


Fig. 10. Reacquisition Network Output

The output vector of the reacquisition deep Q network is a 1 by 9 vector of Q values, where the maximum value corresponds to the chosen target to reacquire. The indices of the output vector are then correlated to the 9 targets from which the input was created, and the chosen target information is fed to the propagation algorithm to direct the observer's motion.

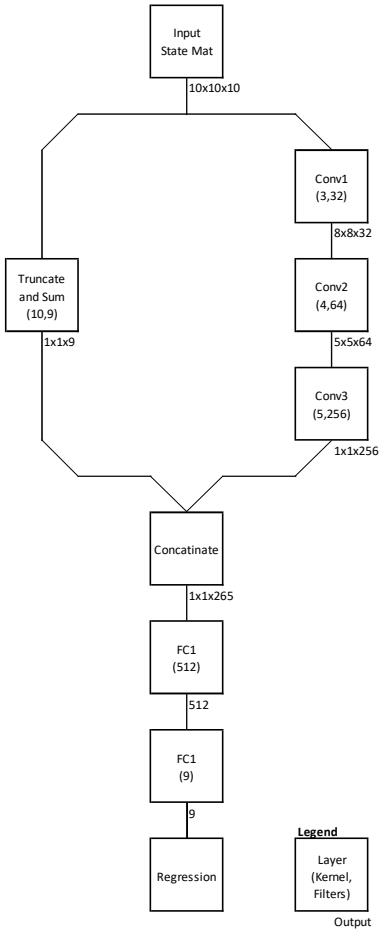


Fig. 11. Reacquisition Directed Acrylic Graph

More license was taken when crafting the reacquisition network compared to the more vanilla search CNN. The reacquisition network is a directed acrylic graph (DAG) network, meaning the layers are not arranged in linear sequence. Instead, two distinct pathways are created through the convolutional layers, each serving a different purpose. The left-hand path, shown on Fig. 11, simply truncates the input matrix down to only the target error channels, and sums each channel to create a 1 x 1 x 9 vector of error magnitudes. The right-hand path, alternatively, performs a series of convolutions on all input channels to maintain spatial information about the target and observer locations. The resultant 1 x 1 x 256 vector is concatenated with the error magnitude vector, and the data is forwarded through two FC layers and finally through a regression output layer.

3.2.1 Performance

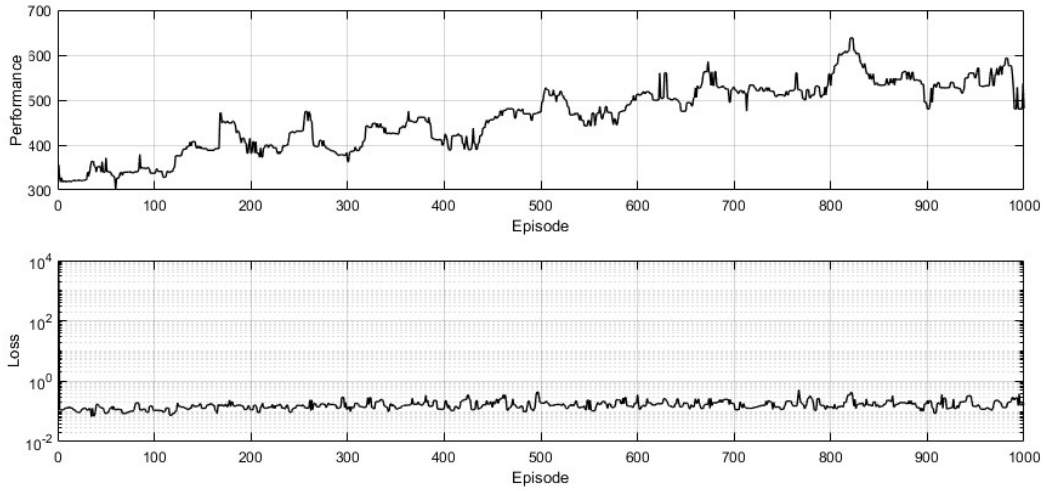


Fig. 12. Reacquisition Training Performance Snapshot

Performance of the reacquisition network is promising. Given the far wider array of possible states than the search network, the reacquisition network necessarily takes more time to train to an optimal level. Additionally, the effect of randomness in the simulation is greater on the reacquisition network, as the performance metric/reward function used is heavily dependent on the number of objects discovered by the observer. Taking this into account, the steady increase in performance of the reacquisition network over 1000 episodes of reinforcement training can be taken as a validation of our approach.

3.3 Tasking

The tasking network is the keystone of the modular network architecture investigated in this study. While it is the most critical of the three functional networks, it is still rather simple in its design and function. The tasking network needs to be provided enough information to decide between the search and acquisition functions but does not have the responsibility to determine which target or search region to go after.

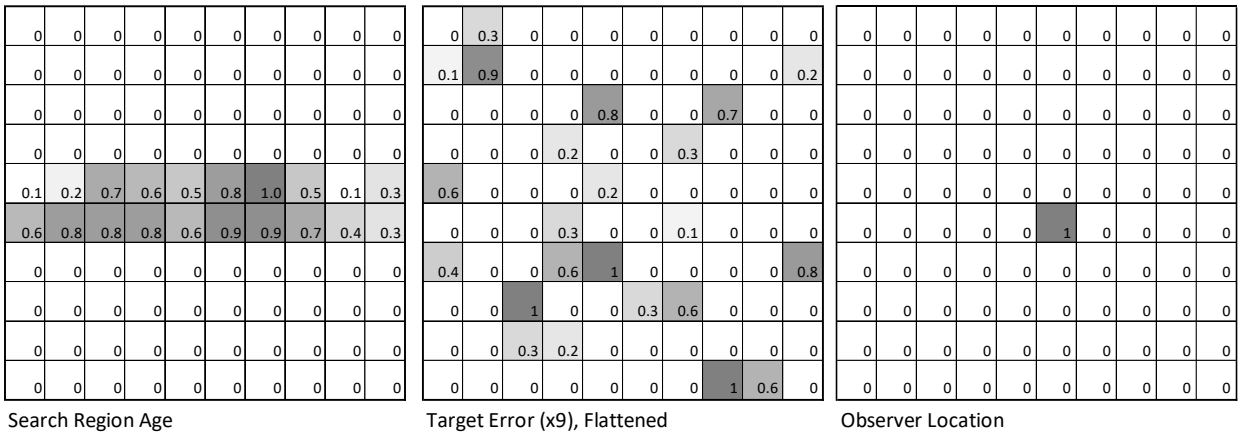


Fig. 13. Tasking Network Input

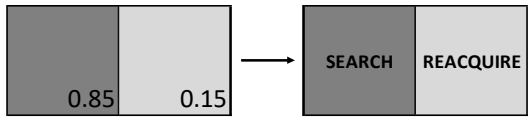


Fig. 14. Tasking Classification Output Vector

The inputs to the tasking network are similar in form to the other two networks we have already examined. The inputs need to convey state information about the search region, the target catalog, and the observer location, but it does not need to be as granular in detail as the information provided to the networks responsible for driving the observer position. To accomplish this, the input to the tasking network is a 10 by 10 by 3 matrix input, shown in Fig. 13. The first channel contains search region information and is identical to the matrix provided to the search network. The second channel is a single catalog error matrix similar to the input to the reacquisition network but flattened through cross-channel addition and normalization. The third and final channel is the standard one-hot observer location matrix provided to all three of our networks. The output of the tasking network is simply a 1 x 2 Q vector where the elements correspond to search or reacquisition. The greater of these two values determines which function is executed next.

Layer	Filter Size	Output
Input	-	10 x 10 x 3
Conv1	3 x 3	8 x 8 x 32
ELU	-	8 x 8 x 32
Conv1	3 x 3	6 x 6 x 64
ELU	-	6 x 6 x 64
Conv1	5 x 5	2 x 2 x 128
ELU	-	2 x 2 x 128
FC1	-	512
ELU	-	512
FC2	-	2
Reg.	-	2

Fig. 14. Tasking Network Outline

The architecture for the tasking network is a simple CNN. Unlike the frameworks used in the other networks, however, it is less sensitive to information loss because it does not need to drive the observer to a specific grid location. As such, the tasking network could employ max pooling layers between convolutions to limit the size of the network. However, given the rather small 10 x 10 size of each channel, max pooling is not required to reduce computation time and training. Should the environment be expanded beyond the 10 x 10 unit environment used in this study, one might choose to employ max pooling to reduce computations.

3.3.1 Performance

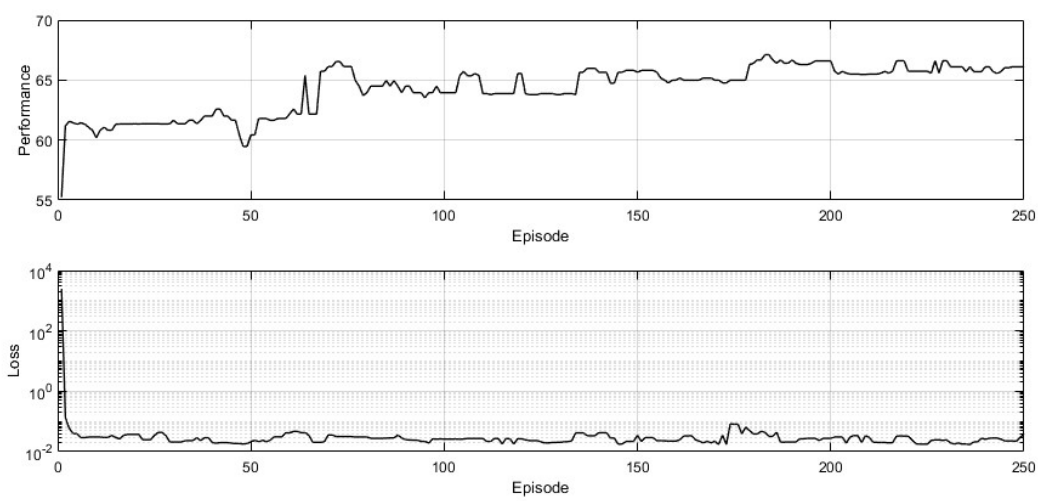


Fig. 15. Tasking Training Performance Outline

The tasking network need far fewer episodes than the other two networks before it begins to reach asymptotic performance. The performance metric used to gauge tasking performance is a 0 – 100 value, where 50 points represent search performance and the other 50 represent reacquisition performance. A truly random tasking engine should result in a performance around 50, as it should evenly task between search and reacquisition functions. This is indeed the case at the start of training, as episode 1 has an approximate value of 55. Within 100 episodes the tasking network shows a 15% increase in performance, and within 250 episodes this trend continues toward a 25% performance increase above randomness. This is an encouraging result, as it demonstrates the tasking network's ability to learn and optimize in a short amount of time.

4. INTEGRATED NETWORK TRAINING AND ADAPTABILTY

At this point the three functional modules have undergone isolated development and training for the execution of their individual functions. The performance enabled by this integrated network is quite competent as each function is well tuned to its own task given the state of the system at any point in time. To exercise the adaptability of the full modular network, however, we may wish to retrain the all three modules together to fine-tune our optimization against our performance metrics, or even train against new performance metrics. No physical system is constant. Rarely, over the life of an operational program, are mission requirements constant. Furthermore, hardware degrades over time, undergoes upgrades, and is moved from one location to another. Given these facts, the adaptability of a system to new and changing parameters is an invaluable characteristic, especially when that adaptability can be automated and does not contribute significantly to operations costs. Artificial neural networks, given their ability to self-optimize to a specific problem set, are an ideal framework for the creation of automated tasking algorithms.

Transfer learning, with respect to deep neural networks, is the ability to adapt a previously trained network to a new or secondary task. For example, pretrained image classification networks can be taught to identify altogether new objects within an image by simply re-initializing their FC layers and retraining with new datasets. This secondary training is far quicker than training the full network from scratch thanks to the information pretrained into the network's convolutional layers. The principle of transfer learning is not unique to classification networks, however, and can be utilized within a reinforcement construct as well [9].

First, it is important to note that the design of our networks, while based upon the design of the environment's 10 x 10 grid, are not specifically designed to the characteristics of our modeled SSA system. In fact, the specifics of our physical system, if encoded anywhere, only appear in the reward functions that our deep reinforcement networks are trained against. As previously mentioned, DQNs primarily learn the identifying features of states through the training of their convolutional layers, and they learn to calculate Q values from a reward function through the training of their FC layers. If we have done a sufficient job of exposing our network algorithms to a wide array of state configurations, the convolutional layers of the tasking, search, and reacquisition networks should provide information generalizable to the generation of modified or altogether different reward functions. Therefore, should we wish to refine our optimization, or even change the parameters of our system or reward mechanics, we should find transfer learning to be an apt mechanism.

The ability for these algorithms to re-optimize given new parameters is significant. A single set of neural networks could be trained to task a wide array of different SSA systems with little to no code modifications – that is, a significant reduction in man-hours compared to traditional hard-coded algorithms. Additionally, an operational system could be run with a high degree of optimality, even through its natural process of degradation. While future work is necessary to fully understand the limitations and capabilities of transfer learning as it applies to tasking algorithms trained through reinforcement, the concept of transfer learning is well established, and speaks to the generalizability and adaptability of a neural SSA tasking architecture.

5. CONCLUSIONS

This study explores the ability of a modular neural network architecture to perform a basic SSA-like mission within a MATLAB simulation. Using a modular architecture, the design and training of the individual search, reacquisition, and tasking networks is simplified. Yet, despite the lack of complexity, this network-of-networks approach to deep

learning results in a set of functions that are optimizable, and adaptable to changing environment and system parameters. While this study is limited in its scope and simple in its interpretation of the SSA problem, it offers a path forward to future research on the automated tasking of SSA systems on the ground and in space using neural algorithms. The adaptability of neural algorithms to changing system parameters and mission performance metrics makes them an ideal candidate for the tasking of a wide array of future SSA systems, alleviating the need for more expensive human resources following initial network development. Future work to investigate the application of modular neural networks into an SSA tasking engine include the development and execution within a more mission-like simulation environment, verification of algorithmic performance against real world mission objectives, and a more substantive investigation into the adaptability and scalability of these network architectures to various SSA system configurations. It is the hope of the author that an initial investigation, however cursory, will drive more interest into the use of modular neural networks for the tasking of SSA systems and similar systems.

6. REFERENCES

1. Johnson, N., Orbital Debris: the Growing Threat to Space Operations, in *Proc. the 33rd Annual Guidance and Control Conference*, Breckenridge, CO, 2010
2. Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*. MIT Press, Cambridge, MA, 2017
3. Hao Peng and Xiaoli Bai. Improving Orbit Prediction Accuracy through Supervised Machine Learning. Advances in Space Research, 2017.
4. Happel, B., and Murre, J., Design and evolution of modular neural network architectures, Leiden University, The Netherlands, 1994.
5. Kandel, E., Schwartz, J., and Jessell, T., *Principles of Neural Science*, 4th Edition, McGraw-Hill, New York, NY, 2000.
6. Watkins, C., Learning from delayed rewards. PhD Thesis, University of Cambridge, England, 1989.
7. Darley, J., Glucksberg, S., Kamin, L., and Kinchla, R., *Psychology*, 2nd Edition, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984
8. Gu, S., Lillicrap, T., Sutskever., and Levine, S., Continuous Deep Q-Learning with Model-based Acceleration, in *Proc. of Machine Learning Research*, Vol. 48, New York, NY, 2016.
9. Taylor, M., Stone, P., Transfer Learning for Reinforcement Learning Domains: A Survey, Journal of Machine Learning Research, Vol. 10, 2009.