Deep Learning for an Intelligent Intruder

Alex Bott
Supervised by Edmund Hunt & James Ward

March 5, 2024

1 Introduction

In recent years there has been an increasing demand in reliable and effective surveillance solutions. As a result of this, multi-robot patrolling systems have gained significant traction. This is a result of its inherent advantages over its single-robot counterparts in the surveillance of diverse domains (e.g.: warehouse security [9]) - providing enhanced coverage, adaptability and resilience to failures. In short, multi-robot patrolling is able to utilise its collective capabilities to boast a superior monitoring performance.

At its core, the field of multi-robot patrolling is concerned with effectively monitoring an environment for security or observation purposes, potentially in the presence of adversaries that seek to gain undetected access to the environment. These adversaries could range from human intruders to sophisticated attack-robots, each posing unique challenges and requiring tailored defense mechanisms. This is of interest to governmental or private organisations concerned with infrastructure security and conservation or research work involving environmental monitoring.

Following James Ward and Edmund Hunt's work on empirical methods for benchmarking multi-robot patrolling strategies [15], this project aims to build an improved intelligent adversary model that, given realistic constraints, incorporates machine learning (ML) and reinforcement learning (RL) methods to effectively attack patrol systems without prior knowledge. The development of an adversary in this manner will provide a rigorous framework for assessing the robustness of multi-robot patrolling strategies against diverse threats - the more intelligent and realistic the attacker model, the more useful our analysis.

Therefore, the goal for this project is to optimise the success rate at which an outside entity can invade any given domain undetected against a variety of patrolling techniques incorporated by a system of agents. Ultimately, this will provide a baseline test for the critical evaluation of any/all patrol methods and in turn determine an optimal strategy for the respective environment.

2 Literature review

2.1 Background

The work done by James and Edmund (as previously mentioned [15]) sets a solid foundation for this problem. It notes that in the field of multi-robot patrolling there is a lot of work associated with distributed strategies for robot teams, but very little benchmarking of performance in an 'adversarial setting'. The concept of adversarial patrolling introduces an element of strategic competition between the patrolling robots and the adversary; this necessitates the development of intelligent adversary models that can learn from past interactions and adapt their attack strategies accordingly.

The environments in multi-robot patrolling are often represented as graphs, where nodes correspond to points of interest and edges denote the paths between these points. This graph-structured representation simplifies the problem by transforming the continuous space into discrete nodes and paths, making the application of various algorithmic solutions possible. Machado et al [10] introduces this and one of the widely used performance metrics in this field: 'idleness', defined as the time elapsed since a node was last visited. The objective of most patrolling algorithms is to minimise idleness, as lower idleness implies higher frequency of visits and, consequently, better coverage [6]. James and Edmund introduce 'vulnerability' as a target metric, defined as the time until a node is visited again. An attack is considered successful if the vulnerability exceeds the attack length, i.e., the time taken by the adversary to complete an attack.
Patrolling strategies can be broadly categorised into distributed and centralised strategies. In distributed strategies, each robot makes its own decisions based on local information, while in centralised strategies, a central controller makes decisions for all robots based on global information (e.g.: SEBS as a distributed strategy [12]). The choice between these strategies depends on the specific requirements of the application. In adversarial settings, the patrolling system's performance is measured not only by its ability to minimise idleness but also by its competence to anticipate and counter the adversary's strategies. This necessitates the development of models that can intelligently adapt to the adversary's strategies, adding another layer of complexity to the problem [2]. Noa Agmon's 'full knowledge opponent' is a good example regarding the shortcomings of the adversary used in a lot of papers, in which a lot of assumptions are made (in this case full knowledge of the patrol algorithm) which don't reflect a dynamic attacker or realistic scenario.

To go about solving the assignment, time series data from James and Edmund's paper can be sampled for different systems as an initial ML dataset. For the training of a model, in the context of having no prior knowledge, understanding the different patrol algorithms and environments, although helpful for initial visualisation, is not required. By utilising metrics such as idleness, distance, and velocity within a ML architecture, it is feasible to construct deep learning systems for an adversary in a multi-robot patrol domain.

2.2 Deep Learning

Machine Learning (ML) is a sub-field of artificial intelligence (AI) that focuses on enabling machines to learn from data without being explicitly programmed. The Hundred page Machine learning book by Andriy Burkov [5] is a good preface for the basics of ML and its potential applications. It covers a wide range of algorithms and techniques that allow computers to improve their performance on a specific task by analysing data and identifying patterns. Of the available ML tools, Deep Neural Networks (DNN) are one of the most powerful and versatile.

In 'Deep Learning Essentials' by Wei Di, Anurag Bhardwaj and Jianing Wei [4] it describes the advantages that deep learning has over traditional 'shallow' methods. A subset of this: DNNs are a type of artificial neural network that consists of multiple layers of interconnected neurons which pass information to each other through weighted connections. The weights of the connections are learned during the training process, and they determine how much influence one neuron has on another.

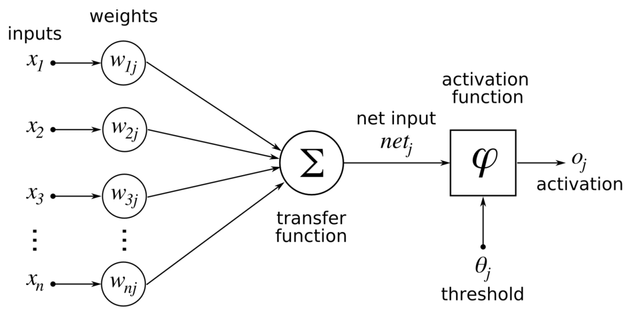

Figure 1: *A mathematically formulated artificial Neuron [16]*

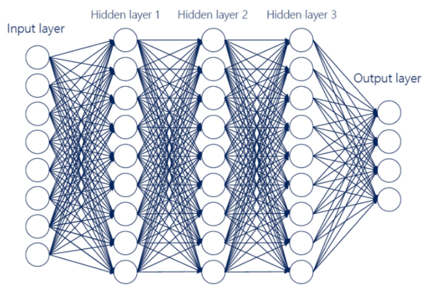

Figure 2: *An example DNN structure [11]*

These layers allow DNNs to extract features from data at different levels of abstraction, meaning, with the addition of an activation function, they are capable of learning complex/non-linear patterns and have been successfully applied in a wide range of tasks, including image recognition, natural language processing, and machine translation.

A critical component in the training of any ML model is the optimiser. This is an algorithm responsible for adjusting the parameters of the model in a way that minimizes the loss function. Yixiang Wang and their associates' work [14] provides a good background and comparison for 4 general optimsers: Stochastic Gradient Descent (SGD), RMSprop, Adadelta and Adaptive Moment Estimation (Adam). They conclude that for structured datasets, Adam produces the highest quality results, and for unstructured, Adadelta is generally better. For the adversary, given that the data and classification is well defined: Adam with a binary cross entropy loss is a good starting point.

Recurrent Neural Networks

When designing a DNN an understanding of the different layers that can be used is crucial, and in the context of the problem space, the model needs to be able to handle sequential data. Recurrent Neural Networks (RNN) are a type of layer that achieve this by using feedback loops and maintaining an internal memory. However, RNNs can suffer from vanishing and exploding gradients (see [3]), which is a problem for long range dependencies.

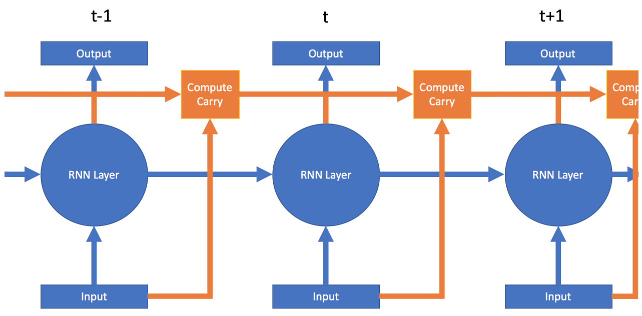

Figure 3: *simple architecture of an LSTM layer [8]*

A Long Term Short Memory (LSTM) layer 3 is an RNN that can account for this with an added carry function. Jannes Klaas [8] breaks the compute carry into 3 parts, with the new carry being computed as:

$$c_{t+1} = c_t * f_t + i_t * k_t \tag{1}$$

With the variables i_t and k_t determining what should be added from the input and state, whilst f_t determines what should be forgotten from both.

Jannes also states “While the standard theory claims that the LSTM layer learns what to add and what to forget, in practice nobody knows what really happens inside an LSTM. However, they have been shown to be quite effective at learning long term memory”.
Therefore LSTM’s are ideal for the adversary dataset, particularly for cases where the attack length is drawn out for extended periods of time

Graph Neural Networks

Graph neural networks (GNN) are a type of neural network that is specifically designed to operate on graphs. Graphs are a powerful way to represent relationships between objects, and GNNs are able to learn from these relationships to make predictions or decisions. The Google Research team give an informative introduction to GNNs [13]: how they work, what types of problems they are applicable to and how they are structured with respect to data.
From Edmund and James’s work, the Environments are based on graphic data already, which implies that any target domain by an adversary can be modelled as such. In this case the edges and edge weightings between nodes can be sampled without additional computation:

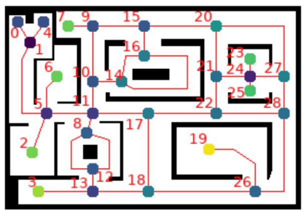

Figure 4: *Graphic visualisation of example environment*

One of the key characteristics of GNNs this research lists, is the message passing between nodes and neighbourhood aggregation, allowing them to capture the local context of each node and learn patterns with respect to the graph structure. In theory, by applying this to the sampled dataset, with adequate reshaping/structure it would allow the DNN to train on metrics from neighbouring nodes as well as its own, improving the accuracy of the model.

2.3 Python Integration

TensorFlow [1] is a free and open-source software library in Python for numerical computation and large-scale machine learning. TensorFlow is designed to be flexible and powerful, and it can be used to build a wide variety of machine learning models, including deep neural networks, recurrent neural networks, graph neural networks and more.

Keras [7] is a high level API for Tensorflow. It provides a more accessible interface for a user to build and train ML models, whilst retaining all the benefits of Tensorflow, and is therefore an adequate choice to use in the construction of the adversary model.

3 Project plan

Table 1: Table showing work progression of the project (primarily bi-weekly). At time of submission, work up to week 10 will have been completed.

ACTION	JUSTIFICATION	PERIOD
Read Through the short paper published by James and Edmund - also recap Keras API syntax and setup github repo	To Familiarise oneself with the environment space and what the data represents with respect to machine learning	Weeks 3-4
Reshape and visualise current/past data provided by James to use as an initial input into simple DNN model using Keras API	Allows for initial setup of machine learning approach without needing to simulate new data.	Week 5
Implement an LSTM layer into DNN and reshape input data appropriately with varying callbacks	This incorporates time series into the fitting of the model	Week 6
Using current model vary sigmoid binary output thresholds and plot visuals, also vary data inputs: different data, maps and algorithms	Initial performance data, directly comparable to the work done by the intelligent adversary in James and Edmund's Work	Weeks 7-8
research GNN's and optimsers and start to implement a GCN layer into model, also complete intital and final draft of interim report	hopes to improve accuracy of model with varied learning rate and utilising node neighbours and weighting data	Weeks 9-10
Finalise implementation of GCN and optimise hybrid model parameters	Finalise the machine learning approach so that a reinforcement approach can be taken after the break	Weeks 11-12
Continue to do some reading when opportunities present itself	help to build on and retain what I've learnt so far	**Christmas Break**
Read through textbook on Reinforcement learning and setup initial environment	Setup for Reinforcement learning approach	Weeks 13-14
Prototype reinforcement learning model. Start structuring technical project layout	To begin integrating reinforcement learning techniques with the previously developed neural network model	Weeks 15-16
Fine-tune reinforcement learning parameters and perform initial testing. Do some writing for final dissertation	To optimize the performance of the reinforcement learning model and ensure stability in various scenarios	Weeks 17-18
Evaluate the model using new datasets and compare against baseline models. Compile results and draft project report	To validate the effectiveness of the model and identify areas for improvement	Weeks 19-20
Finalise all work and finish the final report	Conclude all work that has been done and what has been achieved	Weeks 21-22

4 Progress

4.1 Preface - data, frameworks and environment setup

'Intelligent-Intruder' - a private github repository - is setup as a source control tool for this project.

All of the computation is done in python using Pandas for datastructures, Tensorflow and Keras for ML and Matplotlib for visualisation.

James provided time series training and classification metrics for 3 different patrol methods: CLBS, DTAP and DTAG in 3 different environments: Cumberland, example and grid. Each instance had multiple iterations for the same and different numbers of patrolling agents.

4.2 Model

4.2.1 Initial choice and fitting of data

Initially I chose to work with '4-agents DTAG grid' simulation data, which is argued as one of the more predictable maps and patrol methods.

The data itself needed cleaning. This was not a lengthy process and only required the removal of negative 'Idleness' and positive infinite 'Vulnerability' values with appropriate dataframe cropping. These would occur only at the beginning and end of the dataset and therefore had no affects on the stepsize/chronology.

To format the data I took a time series forecasting approach. By reshaping the data into a 3d array with a callback value set to be equal to the attack length I was able to create a dataset that would be appropriate for a LSTM layer in my initial DNN

4.2.2 DNN setup with LSTM

Following the reshaping of the data, the initial model was developed using Keras API. Using a sequential model (common architecture for ML models, in which the layers are in series), the initial structure is as follows:

- *LSTM layer*
  Neurons: 4 - a reasonable starting point that avoids overfitting or inefficiency in computation.
- *Output Dense layer*
  activation function: sigmoid - a robust and computationally efficient function that introduces non linearity to binary classifications.
  Neurons: 25 - means that a sigmoid value for each node is produced at each timestep as the ouput.

For the compiling and fitting:

- *Compilation*
  Optimiser: Adam - as discussed in lit review, generally one of the better optimisers when training on structured data.
  loss-function: binary cross entropy - as the classification is binary, this addresses the nature of the problem better than Adam's default cross entropy.
- *Fit*
  epochs: 10 - reasonable choice for training without overfitting.
  batches: 1 - time series forecast is a large dataset, so this helps to reduce memory usage.

Setting the attack length to 50 seconds and a sigmoid binary output threshold of 0.5, the Accuracy and F1 score metrics for the results from fitting the DNN are as follows:

	DTAG		DTAP		CLBS	
Data split	Accuracy	F1	Accuracy	F1	Accuracy	F1
Training	79.3%	45.9%	74.2%	26.1%	65.1%	27.6%
Testing	79.2 %	45.7%	73.8%	18.5%	59.5%	22.9%

Aside from the CLBS algorithm (which is inherently very unpredictable), surprisingly the first iterations of the model weren't terrible, but it obviously suffered from sample efficiency on a few nodes where it missed all available attacks. In an attempt to fix this, the learning rate was changed from the default exponential decay to static. which resulted in improved results notably F1 score:

	DTAG		DTAP		CLBS	
Data split	Accuracy	F1	Accuracy	F1	Accuracy	F1
Training	81.3%	56.9%	77.0%	33.9%	66.0%	28.2%
Testing	80.9 %	55.5%	75.2%	28.3%	65.1%	25.3%

In the context of an adversarial model, the aim is not to maximize the number of successful attacks but rather to ensure that when an attack is made, it has a high probability of success. We can do this by changing the desired output by varying the binary threshold for the Sigmoid activation function. This threshold can be adjusted to reflect different levels of confidence required before making a decision. i.e.: If the threshold is set high, the adversary will only attack when it is highly confident in success

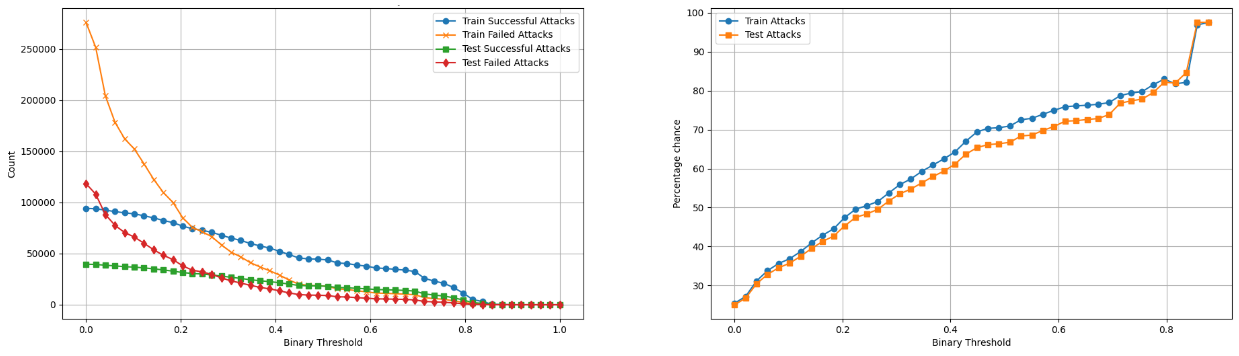

Figure 5: *Graphs showing the counts of successful and failed attacks (left) and the chance an attack is successful (right) against the binary sigmoid output threshold for DTAG patrol algorithm.*

As can be seen from the above figures 5 the chance of successful attacks has an approximately linear relation with threshold boundary and a dynamic relation with failed attacks.

4.2.3 Adding a GCN layer

With hopes of improving the current model, a custom GCN layer was added in sequence before the LSTM. To get the layer to function it required the extra input of environmental Graphic data as an adjacency matrix. In order to fit this with the sequential data for the LSTM, the timeseries data needed to input as a 4d tensor, so that the adjacency matrix had the same size dimension for nodes as this. Because of this, the following output was not the 3d tensor required for the LSTM, and needed reshaping with another custom layer.

As This GCN is still in early stages of implementation into the model, the results from initial testing for accuracy and F1 score are the same as without. This suggests the GCN wasn't performing as intended, which was expected given the complex nature of building custom layers with Tensorflow API. Given more time, I am confident i can get it to function correctly.

4.3 Future work

Given that the current model takes approximately 20 minutes to train each time, there is still a lot of testing on different environments to be done. Based on this, for the ML model, the GCN custom layer, model parameters and data dimensions will be adjusted and optimised.

As outlined in the Project plan, A Reinforcement approach will be researched and implemented into the current problem domain, in hopes that that it will enhance the model's ability to adapt and optimise its strategies in dynamic, complex environments, ultimately achieving higher levels of performance in real-time decision-making.

References

[1] Martín Abadi et al. Tensorflow. `https://www.tensorflow.org/guide/intro_to_modules`, 2023. Online; accessed 25-November-2023.

[2] Noa Agmon, Gal A Kaminka, and Sarit Kraus. Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, 42:887–916, 2011.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Anurag Bhardwaj, Wei Di, and Jianing Wei. *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling.* Packt Publishing Ltd, 2018.

[5] Andriy Burkov. *The hundred-page machine learning book*, volume 1. Andriy Burkov Quebec City, QC, Canada, 2019.

[6] Yann Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.(IAT 2004).*, pages 302–308. IEEE, 2004.

[7] François Chollet et al. Keras. `https://keras.io/api/models/`, 2015. Online; accessed 25-November-2023.

[8] Jannes Klaas. 19 - lstm for email classification. https://www.kaggle.com/code/jannesklaas/19-lstm-for-email-classification, 2023. Online; accessed 25-November-2023.

[9] Malrey Lee, Mahmoud Tarokh, and Matthew Cross. Fuzzy logic decision making for multi-robot security systems. *Artificial Intelligence Review*, 34:177–194, 2010.

[10] Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *International workshop on multi-agent systems and agent-based simulation*, pages 155–170. Springer, 2002.

[11] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. Edge machine learning for ai-enabled iot devices: A review. *Sensors*, 20:2533, 04 2020.

[12] David Portugal and Rui P Rocha. Distributed multi-robot patrol: A scalable and fault-tolerant framework. *Robotics and Autonomous Systems*, 61(12):1572–1587, 2013.

[13] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9):e33, 2021.

[14] Yixiang Wang, Jiqiang Liu, Jelena Mišić, Vojislav B Mišić, Shaohua Lv, and Xiaolin Chang. Assessing optimizer impact on DNN model sensitivity to adversarial examples. *IEEE Access*, 7:152766–152776, 2019.

[15] James C Ward and Edmund R Hunt. An empirical method for benchmarking multi-robot patrol strategies in adversarial environments. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 787–790, 2023.

[16] Wikimedia Commons. Artificial neuron model english. `https://en.m.wikipedia.org/wiki/File:ArtificialNeuronModel_english.png`, 2023. Online; accessed 20-April-2023.