Towards a Deep Reinforcement Learning based approach for real-time decision making and resource allocation for Prognostics and Health Management applications

Submitted in Partial Fulfillment of the Requirements for the Degree Master of Engineering (Mechanical Engineering)

Ricardo Pedro João Ludeke

under the supervision of
Prof. P.S. Heyns

Centre for Asset Integrity Management (C-AIM)
Department of Mechanical and Aeronautical Engineering
University of Pretoria

December 2020

Abstract

Industrial operational environments are stochastic and can have complex system dynamics which introduce multiple levels of uncertainty. This uncertainty leads to sub-optimal decision making and resource allocation. Digitalisation and automation of production equipment and the maintenance environment enable predictive maintenance, meaning that equipment can be stopped for maintenance at the optimal time. Resource constraints in maintenance capacity could however result in further undesired downtime if maintenance cannot be performed when scheduled.

In this dissertation, the applicability of using a Multi-Agent Deep Reinforcement Learning based approach for decision making is investigated to determine the optimal maintenance scheduling policy in a fleet of assets where there are maintenance resource constraints. By considering the underlying system dynamics of maintenance capacity, as well as the health state of individual assets, a near-optimal decision making policy is found that increase equipment availability while also maximising maintenance capacity.

The implemented solution is compared to a run-to-failure corrective maintenance strategy, a constant interval preventive maintenance strategy and a condition based predictive maintenance strategy. The proposed approach outperformed traditional maintenance strategies across several asset and operational maintenance performance metrics. It is concluded that Deep Reinforcement Learning based decision making for asset health management and resource allocation is more effective than human based decision making.

Keywords: Deep Reinforcement Learning, Multi-Agent Reinforcement Learning, Maintenance Policy Optimisation.

Author: Ricardo Pedro João Ludeke
Student Number: 10177303
Supervisor: Prof. P.S. Heyns

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would like to thank my supervisor Prof PS Heyns for his guidance, patience and continued support in completing this work. I appreciate the opportunity you have given me.

I would also like to thank my wife for her relentless support and understanding throughout all my endeavours, not just in writing this dissertation, but for everything else along this journey. Life is an adventure.

Lastly, I would like to thank my parents for always believing in me, for moving the world to enable me to achieve more than I ever imagined and for their endless support.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Abbreviations

A2C	Asynchronous Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ACM	Automated Contingency Management
AI	Artificial Intelligence
C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
CBM	Condition Based Maintenance
CNN	Convolutional Neural Network
CPUT	Cost Per Unit Time
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSS	Decision Support System
HMM	Hidden Markov Model
I2A	Imagination Augmented-Agents
KL	Kullback-Leibler
LSTM	Long Short Term Memory
MARL	Multi-Agent Reinforcement Learning
MBMF	Model-Based reinforcement learning with Model-Free Fine-Tuning
MBVE	Model Based Value Expansion
MCTS	Monte Carlo tree search
MDP	Markov Decision Process
MPC	Model Predictive Control
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
NE	Nash Equilibrium
PHM	Prognositcs and Health Management
PMP	Planned Maintenance Percentage
POMDP	Partially Observable Markov Decision Processes
PPO	Proximal Policy Optimisation
RCM	Reliability Centred Maintenance
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROA	Real Option Analysis
RUL	Remaining Useful Life
TRPO	Trust Region Policy Optimisation
VPG	Vanilla Policy Gradient

English Letters and symbols

$\mathbf{A}$	Action set
$\mathbf{N}$	Agent set
$\mathbf{O}$	Observation set
$\mathbf{S}$	State set
$\mathcal{D}$	Trajectory set
O	Observation function
A	Advantage function
a	Action
b	Belief
C_P	Preventive maintenance cost
c_t	Maintenance capacity at time t
C_U	Corrective maintenance cost
c_{sp}	Maintenance capacity set-point
F	Probability of failure function (Weibull Analysis)
F	Reward shaping function (Reinforcement Learning)
g	Estimated gradient
H	Hessian
J	Objective function
k	Dimensions, or iteration
L	Surrogate objective function
n	Number of steps per episode
o	Observation
P	Probability function
Q	Action-value function
R	Reliability function (Weibull Analysis)
R	Reward function (Reinforcement Learning)
r	Probability ratio (PPO)
r	Reward (Reinforcement Learning)
s	State
T	End time
t	Time step
V	Value function

Greek Symbols

α	Backtracking coefficient (TRPO algorithm)
α	Scale parameter (Weibull distribution)
β	Shape parameter (Weibull distribution)
δ	KL-divergence limit
ϵ	Clipping parameter
γ	Reward discount factor
μ	Deterministic policy function
μ	Mean (Gaussian distribution)
μ_θ	Parameterised deterministic policy function
Ω	Finite set of observations
ϕ	Approximation deep neural network parameters (Weights and Biases)
π	Stochastic policy function (Reinforcement Learning)

π^i	Individual agent policy function
π^*	Joint policy
π^{-1}	All other agent policies
π_θ	Parameterised stochastic policy function
ρ	Start-state distribution
Σ	Diagonal covariance (Gaussian distribution)
σ	Standard deviation (Gaussian distribution)
τ	Trajectory
θ	Deep neural network parameters (Weights and Biases)

Chapter 1

Introduction

1.1 Background

The fourth industrial revolution, also known as Industry 4.0, is currently underway. Industry 4.0 is a move towards digitalisation, where physical and digital systems are combined to improve system performance. These advancements in performance are made possible through the use of data and analytics. From a Prognostics and Health Management perspective, maintenance strategies are advancing with greater adoption of predictive maintenance practices. Through the use of data and advanced analytic techniques, such as machine learning and deep learning, it has become possible to identify meaningful patterns in vast amounts of data and generate practical new insights for improving asset availability.

The increasing complexity and dynamics within production systems make it difficult for humans to determine the best possible decision making and resource allocation policies. Thus, the use of predictive maintenance alone is not enough and any upstream or downstream process inefficiencies can still lead to undesired downtime or lost production. From this perspective, it is important to consider all system dynamics when making decisions or allocating resources. Reinforcement learning algorithms are well suited for determining optimal decision making policies in dynamic environments.

1.2 Problem

Industrial operational environments are stochastic and can have complex system dynamics which introduce multiple levels of uncertainty. This uncertainty leads to sub-optimal decision making and resource allocation. Digitalisation and automation of production equipment and the maintenance environment enable predictive maintenance, meaning that equipment can be stopped for maintenance at the optimal time. Resource constraints in maintenance capacity could however result in further undesired downtime if maintenance cannot be performed when scheduled.

This work will look at the applicability of using Deep Reinforcement Learning to determine the optimal maintenance scheduling policy in a fleet of assets where there are maintenance resource constraints. By considering the underlying system dynamics of maintenance capacity, a near-optimal decision making policy can be found to increase equipment availability which ultimately results in increased revenue.

1.3 Literature Review

The Prognostics and Health Management (PHM) discipline embodies the maintenance engineering practices that enable real-time health assessment of a system under current operating conditions, as well as the prediction of its future state based on historical information and the process of decision making to maintain the system health (Atamuradov et al. 2017). PHM incorporates various engineering disciplines including sensing technologies, failure physics, machine learning, statistics, and reliability engineering (Pecht 2008). It enables engineers to process operational data in order to generate information on the current system health state and using expert knowledge to form a strategy to take action and maintain the system.

The main tasks of PHM include data acquisition and processing, detection, diagnostics, prognostics, and health management (Nam-Ho et al. 2017). The first task is data acquisition and processing, which is to collect measurement data from sensors and process the data to extract useful features for detection, diagnosis and prognostics. The second task is detection, in which anomalous behaviour is identified by comparing the actual and expected behaviour of the system. The third task is diagnostics, in which the fault is detected and isolated to determine which component is failing and to quantify the fault severity. The fourth step is the prognostics that predicts how long it will take until failure under the current operating conditions. The last step is health management which comprises decision making policies for optimal maintenance scheduling and logistics support. For an effective PHM system, the synergy of all tasks is important in ensuring the reliability of a system in order to mitigate system-level risks while extending its useful life.

Over time, maintenance strategies have evolved from corrective (unscheduled, passive) or preventative (scheduled, active) maintenance strategies to condition-based (predictive, proactive) strategies (Gouriveau et al. 2016). The benefits of employing a proactive strategy over a corrective or preventative strategy comes down to being able to minimise the production time lost due to poor planning and sub-optimal operation which results in reduced operating costs, improved logistic support, increased system safety and ultimately increased revenue (Nam-Ho et al. 2017).

Recent advances in PHM and the adoption of condition-based maintenance strategies (Custeau 2017) have been made possible by the development of digital technologies that allow organisations to transform their operations with improved sensing, monitoring and control capabilities. This shift in the adoption of digital technologies is seen as the 4th industrial revolution, with major enabling technologies such as (ODonovan et al. 2015, Preuveneers & Ilie-Zudor 2017, PwC 2016): overall system integration, big data, cloud-based computation and the use of Artificial Intelligence (AI), leading to cyber-physical autonomous systems allowing for improvements in operational efficiency and productivity.

In order to achieve complete autonomy, systems need to be able to make decisions and take actions without excessive human intervention. Roychoudhury et al. (2017) compiled a report for NASA, assessing state-of-the-art system-wide safety and assurance technologies, considering decision making using prognostics for condition-based maintenance and automated contingency management. The findings are discussed in the following paragraphs looking at the research of Haddad et al. (2011), Iyer et al. (2006) and Balaban et al. (2012).

Haddad et al. (2011) proposed using Real Options theory, which provides an economic basis to manage decision making flexibility, for condition-based maintenance enabled by PHM. Through the use of a Real Option Analysis (ROA), each possible action in a decision-making policy is given a value. The quantifications of the possible options, or actions, will eventually lead to means of choosing the best management decisions for the system. In order to quantify the value of actions in a given maintenance policy, the authors proposed a hybrid solution making use of decision trees to analyse technical risks

in the system. After which Monte Carlo simulations are used to determine the uncertainty of market related risks. A considerable limitation to the use of ROA is that every option for every new application needs to be defined individually as parameters are not the same across applications.

Iyer et al. (2006) proposed a post-prognostic decision support system (DSS) for condition-based maintenance. The DSS allows the operator to make optimal decisions based on interactive expressions of user preferences. The DSS further makes use of the estimated prognostic health state of a system and other variables and constraints related to system maintenance, logistics, and operations. The proposed DSS uses an Evolutionary Multi-objective Optimisation algorithm to generate alternative near-optimal solutions that aid decision-makers to make good decisions. These suggested solutions reduce the risk of human users making sub-optimal decisions. The authors made use of a brute force approach to consider all possible solutions. This approach will not scale past solving small toy problems, but indicated that future research could include the use of a Genetic Algorithm.

Automated Contingency Management (ACM) is the capability to reconfigure control actions and mission re-planning using diagnostic and prognostic information of the system. Balaban et al. (2012) presented an approach to ACM in the aerospace domain using methods from mathematical optimisation, multidisciplinary design optimisation, and game theory. Partially Observable Markov Decision Processes (POMDPs) are used to formulate the mission re-planning problem and a Probability Collectives-based technique is adopted for generating a decision-making policy. The authors' proposed solution methods can also be extended for use in decision making for condition-based maintenance. The proposed Probability Collectives-based technique scaled better than brute force optimisation, but was only tested on a small problem with 25 components and would not be able to deliver real-time decision making in larger systems.

In PHM, deep learning has contributed to significant advances in diagnostic and prognostic capabilities (Khan & Yairi 2018, Li et al. 2018*a*) by utilising the flexibility of deep learning techniques to automatically learn representations from high-dimensional raw data without limitation of human insight or imparting human bias that is often introduced by manual feature selection or feature engineering. From a health management perspective, post-prognostic decision making for maintenance scheduling and logistics support is mostly still reliant on hand-crafted business rules or expert human domain knowledge (Haddad et al. 2011, Iyer et al. 2006, Balaban et al. 2012, Chebel-Morello et al. 2018) for traditional optimisation and the use of AI solutions are not the status quo. Although business rules can be fine tuned to improve efficiency and productivity, it is believed that the same human limitations that restricted advancement in diagnostic and prognostic capabilities in the past are also restricting current health management capabilities.

Deep learning is a subset of AI and Machine Learning that uses multi-layered artificial neural networks to deliver state-of-the-art accuracy in tasks such as object detection, speech recognition, language translation and is actively being applied to solve more problems (*Deep Learning* 2018). Machine learning algorithms allow machines to sense, comprehend and learn from data. More specifically, machines learn from (hand-crafted) examples, rather than from hand-coded rules. Deep learning differs from traditional machine learning techniques in that deep learning techniques can automatically learn representations from data such as images, video, text, or industrial asset operational data without introducing hand-coded rules or human domain knowledge (Deng & Yu 2014). Deep learning architectures are highly flexible and can learn directly from high-dimensional raw data with increased predictive accuracy when sufficient data is made available. Deep learning models also have the ability to generalise to new, unseen, data.

Deep learning is also responsible for many of the recent breakthroughs in AI such as Google DeepMind's AlphaGo (*The story of AlphaGo so far* 2018), Tesla's self-driving cars (Tesla 2018), intelligent

voice assistants (Leviathan 2018) and notably OpenAI's recent milestone to exceed human capabilities in a complex video game like Dota 2 (OpenAI 2018).

Specifically, in the case of Dota 2 there has been tremendous work done in Multi-agent cooperative reinforcement learning. Dota 2 is a real-time strategy game played between two teams of five players, with each player controlling a character called a hero. A Dota-playing AI must master the following (OpenAI 2018):

Long time horizons: Dota games run at 30 frames per second for an average of 45 minutes, resulting in 80000 time steps per game. Most actions (like ordering a hero to move to a location) have minor impact individually, but some individual actions can affect the game strategically; some strategies can play out over an entire game.

Partially-observed state: Units and buildings can only see the area around them. The rest of the map is covered in a fog hiding enemies and their strategies. Strong play requires making inferences based on incomplete data, as well as modelling what one's opponent might be up to.

High-dimensional, continuous action space: In Dota, each hero can take numerous actions, and many actions target either another unit or a position on the ground. OpenAI discretised the space into 170,000 possible actions per hero, with an average of 1,000 valid actions in each time step.

High-dimensional, continuous observation space: Dota is played on a large continuous map containing ten heroes, several buildings, non-player-controlled units, and a long tail of game features such as runes, trees, and wards. OpenAI's model observes the state of a Dota game as 20 000 (mostly floating-point) numbers representing all information a human is allowed to access.

Regardless of the tremendous complexity of Dota 2, OpenAI was able to train a team of five AI agents capable of beating a professional team of five humans. OpenAI's achievement would not have been possible if hand-coded rules or human domain knowledge were used, yet most real world decision making or resource allocation problems are solved with hand-coded business rules or guided by expert human domain knowledge. Although Dota 2 is a game, the tremendous complexity is comparable to real world environments, e.g. optimising the maintenance policy and scheduling of a fleet of mine haul trucks.

To expand on the example of managing a fleet of mine haul trucks the following is considered. In an ideal solution, all systems and business entities that have an input on the operation of the fleet (e.g. parts availability, maintenance scheduling, upstream production, route planning, obstacle avoidance) or rely on the resulting output of the fleet (e.g. overall mine production targets, logistics planning, operator safety) need to be considered. Simply put, in order to achieve the highest possible production yield, all systems need to be synchronised and running optimally. This represents a high-dimensional, partially observable continuous observation and action space where numerous linear or nonlinear system interactions need to be considered over time to achieve the optimal management strategy.

In order to achieve the ideal of utilising a holistic management solution specifically for PHM, a Deep Reinforcement Learning decision making approach is proposed. Reinforcement learning is an area of machine learning focused on how AI agents take actions in an environment in order to maximise a certain reward (Sutton & Barto 1998). Consider the AI agents as the human operators that need to maintain the health of the fleet of haul trucks in a mine operation environment. The human operators need to consider the diagnostic and prognostic results of some health event, after which some decisions need to be made while considering how these decisions impact other systems. Some of these decisions could include: determining the required remedy, determining which parts are required and available or would need to be ordered and when the optimal time would be to schedule the actual physical maintenance, given the existing schedule. The ultimate goal is to minimise production loss while

achieving maximum availability and life-cycle value of the haul truck. Once a fleet of haul trucks are considered as a system within systems, it becomes apparent that achieving this goal is not trivial. This problem and proposed solution can be generalised to the management of any system of critical industrial assets with any real-time decision making and resource allocation requirements.

Deep Reinforcement Learning offers some benefits to the identified limitations of current state-of-the-art decision making methods. The first major breakthrough in combining deep learning and reinforcement learning was made by Mnih et al. (2015) who set out to create a single algorithm that would be able to develop a wide range of competencies on a varied range of challenging tasks. To achieve this they developed a deep Q-network (DQN) which combined reinforcement learning with deep neural networks. The algorithm was tested on 49 Atari games (Bellemare, Naddaf & Veness 2013) using only pixels and the game score as inputs. The same network architecture and hyperparameters were trained to perform well on many different tasks and managed to outperform previous algorithms and even performing comparably to a human professional tester.

The breakthrough performance of DQN on various arcade games challenged Silver et al. (2016) to achieve superhuman performance in more challenging domains, such as the game of Go (Duch & Mandziuk 2007) that requires precise and sophisticated look-ahead and planning. Silver et al. (2016) developed AlphaGo using two deep neural networks: a policy network that outputs move probabilities and a value network that outputs a position evaluation. The policy network was trained initially by supervised learning to accurately predict human expert moves, and was subsequently refined by policy-gradient reinforcement learning. The value network was trained to predict the winner of games played by the policy network against itself. Once trained, these networks were combined with a Monte Carlo tree search (MCTS) (Browne et al. 2012) to provide a look-ahead search, using the policy network to narrow down the search to high-probability moves, and using the value network to evaluate positions in the tree. AlphaGo was able to beat professional human Go players, but required supervised training using human expert moves. AlphaGo Zero (Silver et al. 2017) improved on AlphaGo and was trained only by self-play reinforcement learning, starting from random play, without any supervision or use of human data. The concept of self-play reinforcement learning allows the AI agent to explore its environment learning new action sequences not limited by human insight or bias.

The advances of DQN and AlphaGo Zero formed the foundation for discovering new approaches to existing problems. The use of Deep Reinforcement Learning has thus made its way into business and engineering applications. Reinforcement learning is used in operations research (Powell 2011) e.g., supply chain, inventory management, resource management, for smart cities, healthcare, intelligent transportation system and smart grids. Deep Reinforcement Learning will be an enabling technology for Industry 4.0 in areas such as predictive maintenance, real-time diagnostics, and management of manufacturing activities and processes (ODonovan et al. 2015, Preuveneers & Ilie-Zudor 2017).

Real-world applications usually consist of systems with multiple components, which represent multi-agent scenarios where multiple agents need to work together in order to achieve a common task (Iqbal & Sha 2018). In order to learn effectively in multi-agent environments, agents must not only learn the dynamics of their environment, but also those of the other agents. Buoniu et al. (2010) modelled all agents as a single agent, whose action space is the joint action space of all agents. This approach allows coordination between agents, but due to the action space size increasing exponentially with the number of agents, it is not feasible in large systems due to computational resource constraints.

Lowe et al. (2017) and Foerster et al. (2017) proposed similar solutions where information is shared and learned between all agents. This approach circumvents the exponential growth of the action space, but still requires considerable computational resources to scale as the size of the system increases. Iqbal & Sha (2018) builds on this previous work to learn a single central critic with an attention

mechanism. The attention mechanism dynamically selects which agents to attend to at each time point, improving performance when multiple agents interact in complex domains. The proposed approach increases linearly with respect to the number of agents, as opposed to the quadratic increase in previous approaches (Lowe et al. 2017).

1.4 Related Work

In recent years Deep Reinforcement Learning (DRL) has been used to determine optimal maintenance decision making policies for health management across several industries. The complexity of the implementations vary greatly and are discussed further in this section. Several implementations focus only on the decision making aspect of maintenance scheduling for health management, with little to no research being done to also incorporate predictive maintenance approaches for prognostic capabilities. Research into the field of using Deep Learning for prognostics is however very active and can provide good insights on how to further combine prognostics and health management capabilities.

Wei et al. (2020) developed a DRL framework for determining structural maintenance policies. The authors consider both simple and complex bridge structures and show that their solution is efficient at finding optimal policies for maintenance tasks. The authors use inspection reports of structural conditions to generate a one-hot encoding of damage severity of individual components over time. The one-hot encodings are passed through a Convolutional Neural Network to generate a feature map of the bridge component states that is then passed to the Q-learning network that outputs the maintenance actions for each component. The proposed solution does not predict the damage state of components from sensor measurements and only focuses on choosing the maintenance actions of individual components. The authors use an efficient encoding of the state of multiple components and a single agent to predict the actions of all respective components. The authors did not consider any possible maintenance resource constraints in their approach. This work differs from the work of Andriotis & Papakonstantinou (2019) who took a multi-agent approach to learn a joint policy for maintaining multiple components.

Andriotis & Papakonstantinou (2019) developed a DRL framework that provides life-cycle maintenance and inspection policies for multi-component systems. The authors evaluate their solution on a multi-component truss bridge system. The algorithm was developed as an off-policy actor-critic multi-agent solution, in which each agent learns a policy for each of the subsystems or components. The authors use system states or observations obtained from inspection results of the multi-component system that indicated damage states of increasing severity. Inspections corresponded to measurement-based observations on structural health, which were used to drive posterior state distribution updates through forward filtering of the underlying Bayesian network. The algorithm is called Deep Centralised Multi-Agent Actor Critic and uses a centralised value function for the entire system, while allowing agents to learn a joint policy for decision making on individual components. The authors did not consider any possible maintenance resource constraints in their approach, however in subsequent work by Andriotis & Papakonstantinou (2020) the authors explicitly focused on how risk and budget constraints could be incorporated into their approach to achieve state of the art performance for inspection and maintenance planning.

Liu et al. (2020) developed a DRL based dynamic selective maintenance optimisation strategy for multi-component systems. The authors' approach considers the state of multiple components at the end of a mission as either operational or failed. The solution aims to execute a sequence of consecutive missions while taking optimal maintenance actions between missions to achieve optimal uptime and the lowest cost across all missions. The implementation is evaluated on a large-scale multi-component coal transportation system with constraints on the maintenance time and budget resource capacity.

The authors do not directly compare their proposed solution to alternative maintenance strategies, however show that their DRL based solution was capable of generating a policy for a system that would be intractable to solve without approximation due to the curse of dimensionality. The authors' work is significant in showing that DRL can be useful in generating decision making policies in large multi-component systems under resource constraints. The authors focus on maintenance strategy optimisation and no diagnostic or prognostic capabilities are considered.

Kuhnle et al. (2019) present a multi-agent reinforcement learning approach to predict optimal maintenance times for independent machines in parallel production systems with the aim of minimising maintenance costs, maintaining production capacities and maximising the production output. The authors use condition monitoring of machine parameters to determine system states. The implementation however uses basic statistical deviations in machine parameters to determine when a failure might occur and does not rely on the algorithm itself, or the individual agents, to learn the state transition function directly from condition monitoring measurements. The authors use a reward function that rewards agents for maintaining the lowest possible buffer volume before failure. The authors state that agents are not directly rewarded for stopping a machine for maintenance as close to failure as possible and that this behaviour is learnt implicitly. The authors show that compared to reactive and time-based maintenance strategies, their approach results in less machine downtime and lower maintenance costs. The authors explain that a single maintenance worker maintains all machines and that upon a repair or failure event maintenance is performed over a constant interval. The authors assume that the maintenance interval for a repair event takes 30% less time than a failure event. After maintenance is performed the machine returns to operation and the maintenance worker can return to perform maintenance on other machines. This suggests that the authors did implement a system with a constrained maintenance resource capacity however no discussion was provided on the significance of this constraint. The authors did not implement an agent reward that considered maintenance capacity and did not show any results comparing maintenance capacity to other strategies. This suggests that the direct effect of maintenance capacity was not a focus for the authors.

Huang et al. (2020) considers a similar serial production line problem to Kuhnle et al. (2019), however chose to model the system as a single agent that controls the entire production line using a more explicit reward. The agent reward is a combination of the corrective maintenance cost, preventive maintenance cost and the profit loss due to lost production. The authors state that the implemented solution was able to learn group maintenance and opportunistic maintenance strategies without being explicitly shown any of these strategies. This suggests that the specific reward implementation can aid the agent in learning the desired behaviour. The implementation also outperformed reactive and time-based maintenance strategies. The authors concluded that a multi-agent approach should be used to scale this solution to problems with a larger state and action space. The authors did not consider the effect of decision making in a complex system of several components. The authors did not consider any possible maintenance resource constraints in their approach.

Wei et al. (2020), Andriotis & Papakonstantinou (2019), Liu et al. (2020), Kuhnle et al. (2019) and Huang et al. (2020) use DRL for determining the optimal maintenance policy of multi-component systems, however they do not all state clearly whether they directly make use of sensor measurements from components for prognostic health estimation. The proposed solution in this dissertation does make use of sensor measurements of individual turbofan engines, in a fleet of engines, to make prognostic health state estimations of the individual engines. The health state estimations are made internally and further used to determine the optimal maintenance policy.

Skordilis & Moghaddass (2020) developed a DRL decision making framework that uses Bayesian filtering to infer the state of the system and an RL agent that chooses the maintenance actions based on the determined state. The implementation is sensor-driven, meaning that the states are determined

from real-time condition monitoring data and then provided to the RL agent in order to determine the appropriate action. The authors evaluated their implementation on the C-MAPSS turbofan engine dataset (discussed in more detail in Section 4.2). This is the same dataset used to evaluate the proposed solution of this dissertation. The authors implemented a reward that allows the agent to learn how to generate early warnings or late warnings with respect to a user-defined ideal threshold to failure. This reward enabled the agent to learn the trade-off between the cost of downtime for repair and the greater cost of downtime due to failure. The authors compared the replacement cost, time and failure rate of their implemented solution to a time-based maintenance strategy and a corrective maintenance strategy. They showed that the implemented DRL approach outperformed both benchmarks across all metrics. The authors focused on the prognostic ability of their solution for a single component, or engine, and did not consider the effect of joint decision making in a complex system of several components, such as a fleet of engines. The authors did not consider any possible maintenance resource constraints in their approach.

The work of Skordilis & Moghaddass (2020) follows a two-step approach in which a Bayesian filtering model is first trained and used to infer the health state of a turbofan engine and then passed on to an RL agent for decision making. It is not proposed that this approach offers any clear benefit or drawback when compared to training a model or agent end-to-end for inferring both the health state as well as making an appropriate maintenance decision. The work in this dissertation will however focus on training a single model end-to-end that is capable of determining both the health state of individual engines as well as deciding the appropriate maintenance action.

Other approaches in prognostics that also make use of the C-MAPSS dataset could be considered comparable to the first step of the approach proposed by Skordilis & Moghaddass (2020). Kopuru et al. (2019) reviewed the most recent approaches for prognostics and health management, specifically considering the use of the C-MAPSS turbofan engine dataset. The authors describe over fifty deep learning methods in which the C-MAPSS dataset is used for prognostic applications which mainly focus on using the following types of methods:

- Convolutional Neural Networks (CNN). CNNs apply convolutions to input data over local receptive fields and further uses spatial or temporal sub-sampling to produce a feature map of the original input that can be used for tasks such as classification or regression (LeCun et al. 1995). Li et al. (2018*b*) used a Deep Convolutional Neural Network with a time window approach for improved feature extraction to predict remaining useful life (RUL) with high accuracy.

- Recurrent Neural Networks (RNN). RNNs can process sequential data one element at a time while having the ability to selectively pass information across sequence steps. RNNs can also model sequential and time dependencies on multiple scales (Lipton et al. 2015). There are several variations of RNNs that try to improve some of the underlying problems of training an RNN, such as vanishing or exploding gradients. Long Short Term Memory (LSTM) networks is one variation that uses multiple gate functions and a memory block to prevent the problems of vanishing or exploding gradients (Hochreiter & Schmidhuber 1997). Nguyen & Medjaher (2019), Hsu & Jiang (2018), TV et al. (2019), Miao et al. (2019) and Wu et al. (2018) used LSTM networks to model degradation and predict RUL. Bi-directional LSTM networks interpret the dependencies in the sequence of input sensor data forwards and backwards in time, thus encoding more feature rich information about sensor dependencies (Schuster & Paliwal 1997). Bi-directional LSTM networks are used by Huang et al. (2019) and Wang et al. (2018) and is shown to outperform regular LSTM and RNN approaches in predicting RUL on the C-MAPSS data.

- Hybrid variations of CNN and RNN. There are several different hybrid approaches where the strengths of CNNs and RNNs have been combined to predict RUL (Al-Dulaimi et al. 2019, Ellefsen et al. 2019, Jayasinghe et al. 2019). These approaches typically use a CNN to encode a feature map of the input sequence of sensor data, which is then passed on to a LSTM network to model sequence dependencies, which is then used to predict RUL.
- Hidden Markov Models (HMM). HMMs are probabilistic models that can be used to model a sequence of events where some part of the observed state is unknown or hidden (Stratonovich 1965). HMMs are assumed to be Markov Processes, which are closely related to Markov Decision Processes used in Reinforcement Learning. Markov Decision Processes are discussed in detail in Section 3.2. Delmas et al. (2018) proposes a prognostic RUL prediction method using HMMs. The authors propose an approach of using three HMMs to predict the RUL along with upper and lower confidence bounds.

The prognostic predictions from the reviewed methods can be incorporated into health management applications for determining optimal maintenance policies. The review presented by Kopuru et al. (2019) does not include any reinforcement learning based solutions that have been implemented using the C-MAPSS dataset. The review does however indicate that the C-MAPSS dataset is very popular and used by several researchers to develop prognostic algorithms. The work by Skordilis & Moghaddass (2020) shows that the C-MAPSS dataset can in fact be used in Deep Reinforcement Learning problems by combining prognostics and health management capabilities.

1.5 Research Scope and Contribution

The aim of this research is to study the applicability of Deep Reinforcement Learning (DRL) for real-time decision making and resource allocation in a Prognostics and Health Management (PHM) setting. The PHM setting investigated will comprise of a fleet of equipment that needs to be maintained with a constrained maintenance resource capacity in place. The fleet of equipment represents a multi-component system, with each respective piece of equipment representing a sub-component. There is a finite maintenance resource capacity for the entire system.

Each individual piece of equipment will be controlled by an individual DRL agent. Each agent considers the current health state of its respective piece of equipment, as well as the available maintenance capacity, and decides whether to stop the equipment for planned maintenance, or to continue running the equipment. If an agent fails to stop the equipment before failure, corrective maintenance is performed. Agents cooperate in a multi-agent environment to learn a joint decision making policy that not only considers the health of individual pieces of equipment, but also considers the available maintenance capacity of the system. The decisions of individual agents can impact the maintenance capacity of the entire system, which can effectively also impact the decisions of other agents. The joint policy learnt by agents should thus account for the system interactions and enable optimal decision making across the fleet.

It is hypothesised that Deep Reinforcement Learning based decision making for asset health management and resource allocation is more effective than human based decision making. The objectives of this research include: decomposing a real-world problem, of maintaining a fleet of equipment with a constrained maintenance resource capacity, into a multi-agent reinforcement learning problem as well as comparing suitable methodologies and determining their effectiveness or possible limitations in a relevant domain.

The novel contribution of this work is to develop a DRL framework for joint decision making in a multi-component environment with resource constraints, as well as using a single model that is trained

end-to-end using equipment telemetry for both prognostic decision making and resource allocation for maintenance scheduling. Compared to previous work, as discussed in 1.4, research is primarily focused on using DRL for maintenance scheduling, without the use of equipment telemetry for prognostic decision making, or only focused on using equipment telemetry for prognostic decision making while not considering maintenance scheduling. Where previous work has however considered both prognostic decision making and maintenance scheduling a simplified system was considered. These simplified systems either evaluate maintaining a single piece of equipment, where no multi-component interactions are taken into account, or no resource constraints are considered.

1.6 Dissertation Overview

The chapters of this work have the following layout: the second chapter provides background on different maintenance strategies and introduces several maintenance performance metrics. The third chapter describes the key concepts of Reinforcement Learning and provides an overview of different reinforcement learning algorithms. Chapter four describes the problem definition and presents the data that will be used to simulate the problem environment. Chapter five details the methodology used to simulate the environment and the Deep Reinforcement Learning agent implementation. Chapter six presents the results of the proposed Deep Reinforcement Learning based maintenance decision making solution compared to traditional maintenance strategies. The final chapter discusses the conclusions of the implementation and recommendations for future work.

Chapter 2

Maintenance Strategies and Performance Measures

In this chapter an overview is given of different maintenance strategies, comparing unplanned and planned maintenance methods. The idea of analysing failure data is introduced showing how such analysis can be used to determine corrective actions or for selecting optimal maintenance intervals. Lastly, several asset and operational maintenance performance metrics are introduced.

2.1 Maintenance Strategies

Maintenance strategies can be broadly classified as reactive or proactive, as shown in Figure 2.1, and discussed in the following sections.

2.1.1 Reactive or Unplanned maintenance

Reactive and Unplanned Maintenance is considered a legacy practice where maintenance is only applied after a defect is detected or breakdown occurs. This practice is still used when the failure rate is minimal and failure does not have severe safety consequences or financial implications. Reactive or unplanned maintenance can further be broken down as corrective maintenance or emergency maintenance.

1. Corrective maintenance is carried out after a failure occurred to restore equipment to a condition that it can perform its required operation. Corrective maintenance is also referred to as run-to-failure maintenance, which can lead to long periods of machine downtime and maintenance resulting in lost production time and high costs because of unexpected failure (Williams et al. 1994, Sheut & Krajewski 1994, Blanchard et al. 1995).

2. Emergency maintenance is performed immediately, once a failure is detected without any preemptive action or planning, to prevent serious failure or loss of production if equipment fails or breaks down unexpectedly. Without proper planning, there is still however critical production time lost due to inefficient maintenance and prolonged downtime

2.1.2 Proactive or Planned maintenance

The alternative to waiting for a breakdown to occur before performing maintenance is to preempt any failures by taking proactive actions such as scheduling regular maintenance or predicting the possible state of deterioration in order to schedule maintenance appropriately. Thus proactive or planned

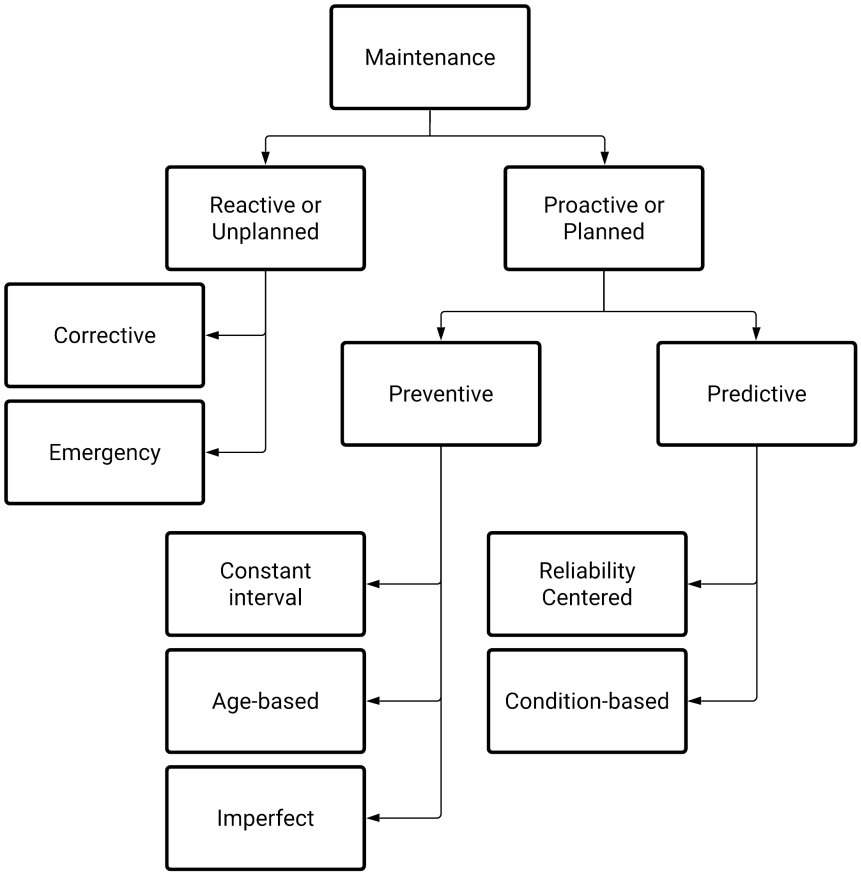


Figure 2.1: Taxonomy of maintenance strategies.

maintenance is performed before equipment fails to ensure better resource utilisation. Preventive maintenance is a strategy that is applied to perform maintenance at predetermined intervals to reduce the likelihood of failures. Preventive maintenance can also be called scheduled maintenance and can further be broken down into constant interval, age-based, or imperfect maintenance.

1. Constant interval maintenance is done at fixed intervals, along with any maintenance due to failure or breakdown during normal operation. Intervals are selected to balance a high risk of failure with long intervals and high preventive maintenance costs with short intervals (Jardine & Tsang 2013).

2. Age-based maintenance is a preventive maintenance strategy where maintenance is performed at constant intervals of a predetermined time. Time can be calendar time or operating time. If the system fails before the specified age, maintenance action is taken and the next maintenance iteration is scheduled once the determined age is reached again. By deferring initiation, this strategy reduces the number of maintenance intervals compared to constant interval maintenance (Jardine & Tsang 2013).

3. Imperfect maintenance considers the uncertainty of the condition of a system while scheduling future maintenance activities. Under preventive maintenance, it is assumed that equipment is restored to original condition after maintenance however it may be that the condition of the equipment is between original (good) and failure (bad). The uncertainty of the current state of equipment is considered while scheduling future maintenance, which is the premise of imperfect maintenance (De Carlo & Arleo 2017).

The predetermined intervals used for preventive maintenance are estimated from the failure rate distribution obtained from historical data or prescribed by the original equipment manufacturer or supplier of individual components in the system (Rao 1992).

Predictive maintenance differs from scheduled preventive maintenance in that maintenance is not performed at fixed intervals, but rather determined adaptively by considering the state of the system. In predictive maintenance, historical operational data is used to model specific fault modes or time to failure. By collecting real-time operational data, the trained predictive models can be used to predict impending failures and be used to plan and schedule maintenance appropriately. Predictive maintenance can be broken down into Reliability Centred or Condition Based maintenance.

1. Condition Based Maintenance (CBM) relies on continuously monitoring the system and its components. The decision to schedule maintenance is made by observing the condition obtained from the predicted model that was trained on historical operational data to predict the state of the system or underlying components. CBM offers several advantages, such as early warning of impending failure, the ability to diagnose the cause of failures, or the ability to predict the time to failure, which can be used to optimise planning and scheduling. It is however expensive to install and use monitoring equipment and develop the condition monitoring models and supporting decision-making strategy (Jardine et al. 2006).

2. Reliability Centred Maintenance (RCM) follows a systematic evaluation to estimate the reliability of the system and balances cost-effectiveness with safety and availability to achieve the goal of minimising costs and downtime while ensuring there is no chance of failure (Moss et al. 1985). RCM can be summarised in seven steps from the SAE JA1011 (1999) standard as follows:

   (a) Describe the operational context, functions and associated desired standards of performance for the asset

   (b) Determine how an asset can fail to fulfil its functions

   (c) Define the causes of each functional failure

   (d) Describe what happens when each failure occurs

   (e) Classify the consequences of failure

   (f) Determine tasks to be performed to predict or prevent each failure

   (g) Decide whether other failure management strategies may be more effective

   Following the steps described in the SAE JA1011 (1999) standard, the appropriate actions and schedule is determined to perform preventive maintenance. Classical RCM maintenance intervals are determined similarly to planned or scheduled maintenance, but with the increasing use of condition monitoring techniques RCM is now more of a predictive maintenance technique where the optimal maintenance interval is prescribed. To determine the required maintenance intervals for both RCM, using a preventive maintenance strategy, and normal preventive maintenance, a Weibull analysis can be performed.

2.2 Failure Data Analysis

In many maintenance strategies, historical failure data can be analysed to determine a failure probability plot. From this, the reliability or probability of failure can be determined at any operating time. This can be particularly useful for determining specific failure patterns, planning for the right corrective

actions as well as calculating the time for constant-interval tasks (Sifonte & Reyes-Picknell 2017).

2.2.1 Weibull Analysis

The basic Weibull analysis consists of fitting a Weibull distribution to failure data. The 3-parameter Weibull distribution probability density function is given by:

$$f(t) = \frac{\beta}{\alpha}\left(\frac{t-\gamma}{\alpha}\right)^{\beta-1} e^{-\left(\frac{t-\gamma}{\alpha}\right)^{\beta}} \tag{2.1}$$

where:

$$\begin{aligned} f(t) &\geq 0 \\ \beta &> 0 \\ \alpha &> 0 \\ -\infty < \gamma &< \infty \end{aligned} \tag{2.2}$$

and:

$$\begin{aligned} \beta &= \text{shape parameter} \\ \alpha &= \text{scale parameter} \\ \gamma &= \text{location parameter} \end{aligned} \tag{2.3}$$

Once the Weibull distribution parameters are fit to the failure data, it is possible to determine the reliability and probability of failure for an asset at a given time. The reliability function, shown in equation 2.4, gives the probability that an asset survives to any given age. Letting T represent the time to failure and t the operating time, $R(t)$ is the probability that the failure does not occur in the interval 0 to t. The reliability of an asset is assumed to be 100% at the beginning of its operating life and continues to decrease until it reaches 0% reliability.

$$R(t) = e^{-\left(\frac{t-\gamma}{\alpha}\right)^{\beta}} \tag{2.4}$$

The probability of failure F(t), shown in equation 2.5, represents the probability of failure at or before operating time t. The probability of failure is assumed to be 0% at the beginning of its operating life and continues to increase until it reaches a 100% probability of failure.

$$F(t) = 1 - e^{-\left(\frac{t-\gamma}{\alpha}\right)^{\beta}} \tag{2.5}$$

The β parameter offers an insight into the failure characteristics of a population of assets. Populations with $\beta < 1$ exhibit a failure rate that decreases with time, populations with $\beta = 1$ have a constant failure rate, and populations with $\beta > 1$ have a failure rate that increases with time.

The location parameter γ, provides an estimate of the earliest time of failure. The period from $t = 0$ to γ is a failure-free period.

2.2.2 Maintenance Frequency

Preventive maintenance strategies rely on scheduling maintenance before failures occur. Using equation 2.6 it is possible to calculate the optimum maintenance interval for a population of assets.

Equation 2.6 represents the cost per unit time at any given time. The reliability and failure probability functions are used, along with the preventive and corrective maintenance costs, to determine the optimal time when the cost of maintenance would be the lowest (Makis & Jardine 1992).

$$\begin{aligned} CPUT(t) &= \frac{\text{Total Expected Replacement Cost per Cycle}}{\text{Expected Cycle Length}} \\ &= \frac{C_P R(t) + C_U\,(1 - R(t))}{\int_0^t R(t)dt} \end{aligned} \tag{2.6}$$

where:

$$\begin{aligned} C_P &= \text{preventive maintenance cost} \\ C_U &= \text{corrective maintenance cost} \end{aligned} \tag{2.7}$$

To use equation 2.6, the following assumptions need to be met:

1. The asset has an increasing failure rate, $\beta > 1$.
2. The cost of preventive maintenance is less than corrective maintenance.

Preventive maintenance typically costs 3 to 5 times less than corrective maintenance (Stenström et al. 2016). Several factors can be attributed to maintenance costs, such as repair time cost, preparation time cost, spares and logistic time cost. Corrective maintenance could result in more severe failures that take longer to repair and require more parts, compared to preventive maintenance.

2.3 Maintenance Metrics

Maintenance Metrics are measurements that provide insights into how assets and maintenance operations are performing. These metrics can be used to track and influence maintenance performance by comparing different actions and strategies to ultimately select actions or strategies that improve maintenance effectiveness.

2.3.1 Asset Performance Metrics

Asset performance metrics consider the asset specific measures that influence maintenance performance.

Uptime

Uptime is defined as the time an asset is available to operate over the period of time it is scheduled to operate. This represents the production hours that an asset is operating without any downtime due to maintenance.

$$\text{UPTIME} = \frac{\text{Asset Operating Hours}}{\text{Asset Scheduled Operating Hours}} \tag{2.8}$$

Mean Time Between Failure

Mean time between failure is a metric that measures the average time between asset failures or breakdowns.

$$\text{MTBF} = \frac{\text{Uptime}}{\text{Number of breakdowns}} \tag{2.9}$$

Mean Time To Repair

Mean time to repair is a metric that measures the average time required to troubleshoot and repair failed equipment.

$$\text{MTTR} = \frac{\text{Total maintenance time}}{\text{Number of repairs}} \tag{2.10}$$

Availability

Availability is defined as the ability of an asset to perform its required function at a stated instant of time or over a scheduled period of time (Rausand et al. 2020). Time losses that influence availability include planned idle time, or times where there are no production, as well as any downtime due to maintenance.

$$\text{AVAILABILITY} = \frac{\text{Uptime}}{\text{Asset Scheduled Operating Hours}} \tag{2.11}$$

Availability can also be calculated as a function of MTBF and MTTR:

$$\text{AVAILABILITY} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \tag{2.12}$$

Utilisation

Asset Utilisation is a measure of how effectively the asset was utilised over its lifetime. Utilisation represents the total scheduled production hours over the total possible hours that the asset could have worked in a period, or over its lifetime.

$$\text{UTILISATION} = \frac{\text{Asset Operating Hours}}{\text{Total Asset Life Time}} \tag{2.13}$$

2.3.2 Operational Metrics

Operational Metrics consider the operational aspects of maintenance performance.

Planned Maintenance Percentage

Planned Maintenance Percentage measures the number of planned maintenance tasks in comparison to all maintenance tasks.

$$\text{PMP} = \frac{\text{Planned maintenance hours}}{\text{Total maintenance hours}} \tag{2.14}$$

Chapter 3

Key concepts of Reinforcement Learning

In this chapter, the key concepts of Reinforcement Learning (RL) are introduced, starting with the main components of RL, the agent and the environment. The Markov Decision Process formulation of Reinforcement Learning is defined, which is the theoretical foundation of reinforcement learning.

The formal formulation is used to introduce the functions that an agent can learn, namely a policy, value functions or advantage functions.

An overview is given of multi-agent reinforcement learning theory and its associated challenges.

Furthermore different types of reinforcement learning algorithms and approaches are introduced starting with model free methods comparing policy optimisation and Q-learning algorithms. Thereafter some model based methods are discussed where the agent either learns a model of the environment or uses a given model to plan ahead. Finally, the pseudo-code for the most prominent policy gradient algorithms are discussed.

3.1 Reinforcement Learning

Reinforcement Learning (RL) is about solving sequential decision making problems, with the goal of maximising some reward. When considering a real-world problem, such as playing a game or optimising a maintenance strategy, one can easily frame the problem in one's mind. During such gameplay, a sequence of actions needs to be taken to progress towards a winning state. The same thinking can be applied to the optimisation of a maintenance strategy, in which asset availability needs to be increased whilst reducing cost. Humans solve these type of problems by taking advantage of available information and choosing an action that can be deduced as being the most beneficial. RL has great potential in solving such problems, as it mimics this same human decision making process.

The main components of RL, as shown in Figure 3.1, are the agent and the environment. The environment is a representation of the observable world which the agent sees and interacts with. As the environment changes at each interaction step, the agent observes the changes in state s_t and selects an action a_t to take in the environment, which can in turn cause the environment to change. The agent receives a reward r_t signal from the environment, that reinforces how good or bad an action is given the current state of the environment. The goal for the agent is to learn a decision making policy that maximises its cumulative future reward. RL methods allow agents to learn these optimal decision making policies in order to achieve this goal (Graesser & Keng 2019).

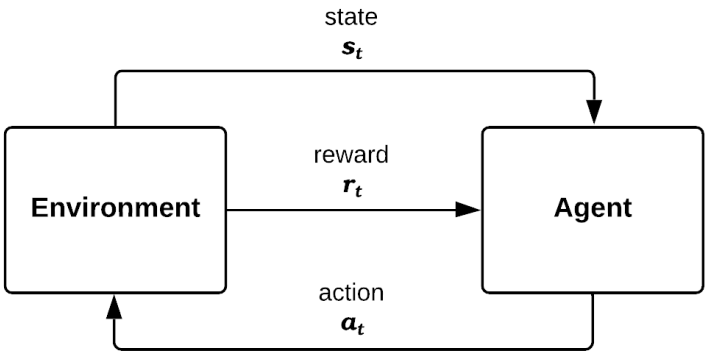


Figure 3.1: The agent-environment interaction loop.

3.2 Markov Decision Processes

To formalise how the environment transitions from one state to the next, a Markov Decision Process (MDP) can be used to formulate a mathematical framework that models the interaction between sequential decision making and state transitions. A MDP is defined by a 4-tuple $(\mathbf{S}, \mathbf{A}, P, R)$:

- $\mathbf{S}$: a set of states
- $\mathbf{A}$: a set of actions
- P: transition probabilities, which define the probability distribution over next states at time $t+1$ given the current state and current action at time t

$$P\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right) \tag{3.1}$$

- R: a reward function, mapping states to real numbers $\mathbb{R}$

$$R : \mathbf{s} \rightarrow \mathbb{R} \tag{3.2}$$

MDPs follows the Markov Property (Markov 1954), that transitions only depend on the most recent state and action, and no prior history. In RL, the transition probability function P and reward function R is unknown to the agent, and the agent only has the ability to act and observe the resulting states and rewards. In order for the agent to learn to take the best action by simply observing the states and rewards, a policy function π is learnt

$$\pi : \mathbf{s} \rightarrow \mathbf{a} \tag{3.3}$$

The following sections describe how the MDP formulation is used to learn the optimal decision making policy π.

3.2.1 States and Observations

If no environment information is hidden, the environment is fully observed and a state $\mathbf{s}$ is used as a complete description of the state of the environment. If some information about the environment is hidden from the state, the environment is partially observed and an observation $\mathbf{o}$ is used as a partial description of the state of the environment.

States and observations are usually represented by a real-valued vector, matrix or higher-order tensors.

$$\mathbf{s} \in \mathbb{R} \tag{3.4}$$

$$\mathbf{o} \in \mathbb{R} \tag{3.5}$$

As an example of the state representations in a game environment, the state can be represented as an RGB matrix of pixel values for the current frame at the current time step (Bellemare, Naddaf, Veness & Bowling 2013). An example of a real-world state representation for a maintenance strategy could be a real-valued vector of the sensor measurements of an asset, such as the temperature and oil pressure of an engine, being monitored at a specific instance in time.

3.2.2 Action Spaces

Depending on the environment, actions can either be discrete or continuous. An environment has a set of valid actions that is called the action space. In a discrete action space, only a finite number of actions are available to the agent, for example in chess an agent can only make a move for a specific piece that is allowed by the rules. Actions in a discrete action space can thus be natural numbers $\mathbb{N}$.

$$\mathbf{a} \in \mathbb{N} \tag{3.6}$$

In a continuous action space, actions can be real-valued vectors, for example in an environment where an agent controls the steering angle, throttle position and brakes of a vehicle the actions can be continuous vectors. Actions in a continuous action space can thus be continuous real numbers $\mathbb{R}$.

$$\mathbf{a} \in \mathbb{R} \tag{3.7}$$

Some Deep Reinforcement Learning algorithms are only suited to discrete action spaces, such as Deep Q-Learning (DQN) (Mnih et al. 2015), or continuous action spaces, such as Soft Actor Critic (SAC) (Haarnoja et al. 2018). The consequences of these limitations need to be considered when designing the environment action space or selecting a learning algorithm.

3.2.3 Policies

The set of decision making rules used by an agent to decide what action a_t to take, given the state s_t at current time is known as a policy. Policies can be stochastic, denoted by π:

$$a_t \sim \pi\left(\cdot|s_t\right) \tag{3.8}$$

or policies can be deterministic and denoted by μ:

$$a_t = \mu(s_t) \tag{3.9}$$

In Deep Reinforcement Learning, policies are computed as functions that are parameterised on the weights and biases of the neural networks used to approximate the functions. The neural network parameters can be updated to modify the behaviour of the policy. To denote the parameterised policy functions, subscript θ is used:

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{3.10}$$

$$a_t = \mu_\theta(s_t) \tag{3.11}$$

Stochastic Policies

Stochastic policies are used to sample actions from a distribution parameterised by the policy. Different distributions are used depending on the action space. For discrete action spaces, a categorical distribution can be used, while for a continuous action space a diagonal Gaussian distribution is used.

A categorical distribution can be used to determine the probability $P_\theta(s)$ of selecting a discrete action from a vector with a discrete number of entries, that is the same size as the number of actions. After sampling from the policy distribution, the log-likelihood for an action a can be computed as

$$\log \pi_\theta\left(a|s\right) = \log\left[P_\theta\left(s\right)\right]_a \tag{3.12}$$

A diagonal Gaussian distribution has a covariance matrix Σ that only has entries on the diagonal. As a result, it is possible to estimate the mean and the variance in each dimension separately and describe the multivariate density function in terms of a product of univariate Gaussians that can be represented as a vector (Zhao et al. 2012).

Given the mean $\mu_\theta(s)$ and standard deviations $\sigma_\theta(s)$, and a vector z of noise from a spherical Gaussian ($z \sim \mathcal{N}(0, I)$), a k-dimensional action vector is sampled with

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot z \tag{3.13}$$

The log-likelihood of the k-dimensional action vector a, for the diagonal Gaussian with mean $\mu = \mu_\theta(s)$ and standard deviation $\sigma = \sigma_\theta(s)$, is given by

$$\log \pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2\log\sigma_i\right) + k\log 2\pi\right) \tag{3.14}$$

In the case of Deep Reinforcement Learning, the weights and biases of the deep neural network used to parameterise the policy distribution represents a function that can map state observations to the mean $\mu_\theta(s)$ and covariance matrix, represented as a single vector $\sigma_\theta(s)$.

Deterministic Policies

Deterministic policies can compute a clearly defined action for every state. In the case of Deep Reinforcement Learning, the deterministic policy represented by the weights and biases of the deep neural network, can directly compute an action from a given input state (Silver et al. 2014).

3.2.4 Trajectories

A sequence of states and actions from the environment is called a trajectory τ

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{3.15}$$

State transitions between time t and time $t + 1$, as described in the MDP formulation in Section 3.2, depend on the environment and can be either stochastic,

$$s_{t+1} \sim P(\cdot|s_t, a_t) \tag{3.16}$$

or deterministic,

$$s_{t+1} = f(s_t, a_t) \tag{3.17}$$

If a trajectory contains all states and actions from the first to the last step, it is also called an episode. In RL, episodes are often referred to when talking about algorithm performance metrics, while trajectories are referred to while talking about algorithm training steps, or the process of training on collected data which might not necessarily include data from the start to end.

3.2.5 Reward and Return

The reward function R depends on the current state of the world, the action taken and the next state of the world

$$r_t = R(s_t, a_t, s_{t+1}) \tag{3.18}$$

The intuition for this is that the reward r_t should inform the agent whether the action a_t taken in state s_t at time t improved the future state the agent will be in given state s_{t+1}. It is also possible to simplify the reward function to only depend on the current state s_t

$$r_t = R(s_t) \tag{3.19}$$

or state-action pair

$$r_t = R(s_t, a_t) \tag{3.20}$$

Rewards can also be referred to as returns. The goal of an agent is to maximise the cumulative future reward over a trajectory $R(\tau)$. As mentioned in Section 3.2.4, trajectories can either contain all states and actions from the start to the end (an episode), or only a subset of states and actions.

The finite-horizon undiscounted return is the sum of rewards obtained by the agent over a fixed number of steps

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{3.21}$$

The infinite-horizon discounted return is the sum of all rewards obtained by the agent over an episode, discounted by a factor $\gamma \in (0, 1)$, that emphasises rewards obtained in the near future over rewards obtained later.

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \tag{3.22}$$

Typically γ is selected to be smaller than 1, in order to prioritise short term rewards which help reinforcement learning algorithms converge. The smaller γ is, the less weight is given to rewards in future time steps, making the reward short term. The closer γ is to 1, the more weight is given to rewards in the future.

3.2.6 Partially Observable Markov Decision Process

A partially observable Markov decision process (POMDP) is an MDP where the agent does not know the real state of the process and instead the agent can only access a partial or noisy observation of the state.

A POMDP is defined as $(\mathbf{S}, \mathbf{A}, P, R, \Omega, O)$, where:

- $\mathbf{S}, \mathbf{A}, P, R$: are the same as in the MDP
- Ω: is a finite set of observations

$$o \in \Omega \tag{3.23}$$

- O: is an observation function giving a probability distribution over all observations

$$o \sim O(s) \tag{3.24}$$

The optimal policy π^* for POMDPs can be defined as a function of belief $\mathbf{b}$. In the MDP formulation, there is an action $\mathbf{a}$ for every state $\mathbf{s}$. In the POMDP formulation, there is an action $\mathbf{a}$ for every belief $\mathbf{b}$. The POMDP formulation thus reduces to a belief-MDP formulation when state estimation is used to map beliefs to states (Zhu et al. 2018). The belief probability mass function over states is denoted as:

$$\mathbf{b} = (b(s_1), b(s_2), ...b(s_{|S|})) \tag{3.25}$$

where

$$s_i \in \mathbf{S} \tag{3.26}$$

$$b(s_i) \geq 0 \tag{3.27}$$

$$\sum_{s_i \in \mathbf{S}} b(s_i) = 1 \tag{3.28}$$

Given the current belief b_t, performing action a_t and using the next observation o_{t+1}, then the next belief $b_{t+1} = SE(b_t, a_t, o_{t+1})$ is estimated as:

$$b_{t+1}(s_j) = \frac{O(s_j, a_t, o_{t+1}) \sum_{s_i \in \mathbf{S}} P(s_i, a_t, s_j) b_t(s_i)}{P(o_{t+1}|a_t, b_t)} \tag{3.29}$$

$$P(o_{t+1}|a_t, b_t) = \sum_{s_j \in \mathbf{S}} O(s_j, a_t, o_{t+1}) \sum_{s_i \in \mathbf{S}} P(s_i, a_t, s_j) b_t(s_i) \tag{3.30}$$

The expected immediate reward for an agent performing action a at the belief state b is computed as:

$$\rho(b, a) = \sum_{s_i \in \mathbf{S}} b(s_i) R(s_i, a) \tag{3.31}$$

The transition function among beliefs becomes:

$$\tau(b, a, b') = \sum_{o \in \Omega} p(b'|b, a, o) P(o|b, a) \tag{3.32}$$

where

$$p(b'|b, a, o) = \begin{cases} 1 & \text{if } b' = SE(b, a, o) \\ 0 & \text{otherwise} \end{cases} \tag{3.33}$$

An optimal policy π^* can be computed by value iteration:

$$V(b) = \max_a \left[\rho(b, a) + \gamma \sum_{b'} \tau(b, a, b') V(b') \right] \tag{3.34}$$

where γ is a discount factor for the past history. Solving equation 3.34 can be intractable using normal value or policy iteration and therefor approximate solutions are often used.

Deep reinforcement learning is often used to solve POMDP problems by incorporating Recurrent Neural Network (RNN) elements to better estimate the current state from an arbitrarily long history of observations. Hausknecht & Stone (2015*a*) make use of a Long Short Term Memory (LSTM) layer to maintain an internal state h_t that describes all input history up to time t and is capable of learning effective POMDP policies. The advantages and disadvantages of using RNNs or LSTMs are discussed in Section 1.4.

3.3 The Reinforcement Learning Optimisation Problem

The goal of Reinforcement Learning is to select a policy that maximises the expected return when the agent acts according to the policy. Since the agent does not know what the transition probability function or reward function looks like, the agent can only interact with the environment by observing states and taking actions, followed by receiving a reward (Sutton & Barto 1998).

For a stochastic environment and policy, the probability distribution over a T-step trajectory is

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{3.35}$$

with $\rho_0(s_0)$ representing the start-state distribution. The agent interacts with the environment to learn a model of transition probability function, as well as the reward function and maximise the expected return $J(\pi)$

$$J(\pi) = \sum_{\tau} P(\tau|\pi)R(\tau) \tag{3.36}$$

$$J(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] \tag{3.37}$$

The expectation accounts for stochasticity in the environment. The objective simplifies to only rely on the expectation of returns over trajectories. Maximising the objective $J(\pi)$ is thus the same as maximising the return. The optimal policy π^* the agent is trying to optimise for is thus

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3.38}$$

3.4 Value Functions

Some RL algorithms make use of a value function to compute the value of a state or state-action pair. In this sense, value refers to the expected return if the agent were to start in the specific state or state-action pair, and act according to a particular policy for the rest of the episode.

On-Policy Value Function

The on-policy value function $V^\pi(s)$ gives the expected return if the agent were to start in state s and always select actions from policy π

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{3.39}$$

On-Policy Action-Value Function

The on-policy action-value function $Q^\pi(s, a)$ gives the expected return if the agent were to start in state s and take an arbitrary action a and afterwards always select actions from the policy π

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{3.40}$$

Optimal Value Function

The optimal value function $V^*(s)$ gives the expected return if the agent were to start in state s and always take actions from the optimal policy

$$V^*(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{3.41}$$

Optimal Action-Value Function

The optimal action-value function $Q^*(s, a)$ gives the expected return if the agent were to start in state s, take an arbitrary action a and afterwards always take actions from the optimal policy

$$Q^*(s, a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{3.42}$$

3.5 Bellman Equations

The Bellman equations are used to express the value of an unknown state in terms of a known state. Using the Bellman equations it becomes possible to iteratively solve the optimal policy, since the value of the current state can be defined recursively in terms of the value of future states.

The Bellman equations for the on-policy value- and action-value functions are

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi \\ s_{t+1}\sim P}} [r(s,a) + \gamma V^{\pi}(s_{t+1})] \tag{3.43}$$

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s_{t+1}\sim P} \left[r(s,a) + \gamma \mathop{\mathbb{E}}_{a_{t+1}\sim\pi} [Q^{\pi}(s_{t+1}, a_{t+1})] \right] \tag{3.44}$$

The Bellman equations for the optimal value- and action-value functions are

$$V^{*}(s) = \max_{a} \mathop{\mathbb{E}}_{s_{t+1}\sim P} [r(s,a) + \gamma V^{*}(s_{t+1})] \tag{3.45}$$

$$Q^{*}(s,a) = \mathop{\mathbb{E}}_{s_{t+1}\sim P} \left[r(s,a) + \gamma \max_{a_{t+1}} [Q^{*}(s_{t+1}, a_{t+1})] \right] \tag{3.46}$$

The optimal Bellman equations reflect that an agent has to pick the action that leads to the highest value in order to act according to the optimal policy. Therefore, without knowing the value of the current state, the agent can simply compare the value of all the possible outcomes from actions in the current state and select the best one (Sutton & Barto 1998).

3.6 Advantage Functions

In policy gradient methods in particular, advantage functions are used to describe the relative advantage of a specific action over other actions on average. The advantage function $A^{\pi}(s,a)$ for policy π describes how much better it is to take a specific action a in state s, instead of randomly sampling an action according to $\pi(\cdot|s)$ if the agent were to act according to π for the rest of the episode. The advantage function is defined as

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{3.47}$$

3.7 Curriculum Learning

Curriculum-based learning enables agents to learn complex tasks by progressively increasing task complexity throughout the learning process. The goal is to design and choose a sequence of tasks for an agent to train on, such that the learning speed or performance on the target task is improved (Bengio et al. 2009, Narvekar et al. 2017).

3.8 Reward Shaping

Reward shaping consists of supplying additional rewards to a learning agent to guide the learning process (Ng et al. 1999). Reward shaping techniques are applied by using a shaping function F, which augments the original reward function R, by making use of prior knowledge to lead the agent towards good overall performance. This aids convergence to an acceptable policy. Reward shaping can be applied directly to reinforcement learning by making the shaping function the native reward function (Laud 2004).

3.9 Multi-Agent Reinforcement Learning

Multi-agent Reinforcement Learning (MARL) is used for sequential decision making problems where multiple agents operate in a common environment, as shown in Figure 3.2. Each agent aims to optimise its own reward by interacting with the environment and the other agents (Busoniu et al. 2008).

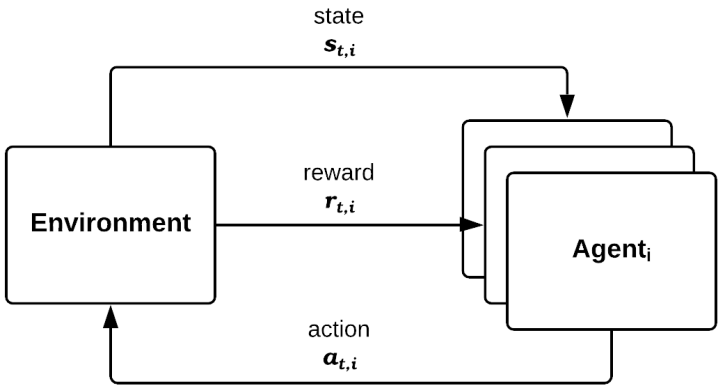


Figure 3.2: The multi-agent agent-environment interaction loop.

MARL algorithms can be placed into three groups, fully cooperative, fully competitive or a mix of the two. Depending on the problem being solved, agents can either work together in a cooperative manner to achieve their goal, or compete against each other to find the best strategy. In some instances, such as team based games, teams of agents can cooperate while competing against other teams. Across all settings, there are several challenges in MARL of which the following sections will go into more detail.

3.9.1 Multi-Agent RL Framework

In a multi-agent RL environment, the actions of a single agent do not just influence the state of the environment and the reward for the individual agent, but jointly affects the state and reward for all other agents. This means that the policy of each individual agent becomes a function of the policies of all other agents. Markov Games represent a mathematical framework for this problem.

Markov Games

A Markov Game is defined by a tuple $\left(\mathbf{N}, \mathbf{S}, \left\{\mathbf{A}^i\right\}_{i \in \mathbf{N}}, P, R\right)$:

- $\mathbf{N}$: a set of agents
- $\mathbf{S}$: the state space observed by all agents
- $\mathbf{A}_i$: the actions space of agent i
- P: transition probabilities, which define the probability distribution over next states given the current state and any join action

$$P\left(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t\right) \tag{3.48}$$

- R: a reward function for the reward received by agent i

$$R^i : \mathbf{s} \rightarrow \mathbb{R} \tag{3.49}$$

At time t, each agent $i \in \mathbf{N}$ executes an action a_t^i, according to the environment state s_t. The

environment transitions to state s_{t+1} and each agent i is rewarded $R^i(s_t, a_t, s_{t+1})$. The goal of each agent i is to optimise its own reward by finding the optimal policy:

$$\pi^i : \mathbf{s} \rightarrow \mathbf{a} \tag{3.50}$$

The value-function of agent i becomes a function of the joint policy of all agents.

$$V^i_{\pi^i, \pi^{-i}}(s) = \mathbb{E}\left[R^i(\tau) | a^i_t \sim \pi^i(\cdot|s_t), s_0 = s\right] \tag{3.51}$$

where $-i$ represents all other agents in $\mathbf{N}$ except agent i. From this it is seen that the solution of the Markov Game is different from the Markov Decision Process. The optimal policy of each agent does not just depend on its own policy, but also on the policy of other agents in the environment.

Başar & Olsder (1998) defined the Nash equilibrium (NE) solution concept to the Markov Game. Nash Equilibrium of the Markov Game is achieved by a joint policy $\pi^* = \left(\pi^{1,*}, ..., \pi^{N,*}\right)$, such that for any state $s \in \mathbf{S}$ and agent $i \in \mathbf{N}$:

$$V^i_{\pi^{i,*}, \pi^{-i,*}}(s) \geq V^i_{\pi^i, \pi^{-i,*}}(s), \quad \text{for any} \quad \pi^i \tag{3.52}$$

The joint policy π^* represents an equilibrium point from which none of the agents benefit, or has any incentive, to deviate from.

3.9.2 Multi-Agent RL Challenges

Multi-Agent Reinforcement Learning has some unique challenges that are described in more detail below:

Reward or Goal definition

Defining the right reward is necessary to achieve the desired goal. Especially in multi-task objectives it is required to design sub-rewards that allow agents to learn the sub-tasks. It can however be a complex and involved process where fine tuning is required (Hausknecht & Stone 2015*b*, Li et al. 2016, Diddigi et al. 2017).

Tampuu et al. (2017) showed that in the same MARL environment, cooperative or competitive behaviour could emerge depending on the designed reward. It is thus a challenge to design the right reward that enables agents to learn the correct policy that results in the desired behaviour.

Non-Stationarity

One of the key challenges of MARL is non-stationarity caused by multiple agents learning concurrently in the same environment. The actions taken by one agent affects the reward and change in state of the environment as observed by other agents in the environment. This is apparent when looking at Equations 3.50 and 3.51. This means that agents effectively also need to learn the joint behaviour of other agents (Busoniu et al. 2008, Tuyls & Weiss 2012).

Scalability

To account for the non-stationarity, each individual agent may also need to account for the joint action or observation space. This means that the action or observation space increases exponentially with the number of agents (Hernandez-Leal et al. 2019).

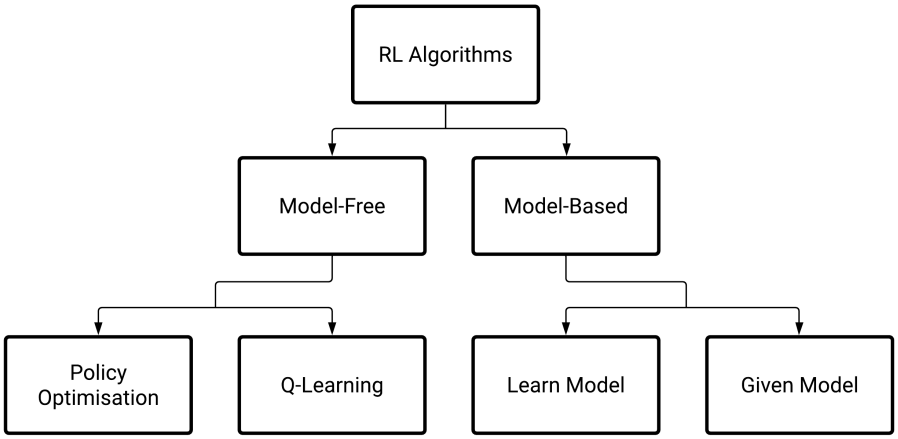


Figure 3.3: A taxonomy of different types of RL algorithms.

3.10 Taxonomy of Reinforcement Learning Algorithms

This section looks at the different types of reinforcement learning algorithms, as shown in Figure 3.3, starting with model free methods comparing policy optimisation and Q-learning algorithms. Thereafter some model based methods are discussed where the agent either learns a model of the environment or uses a given model to plan ahead. Finally, the pseudo-code for the most prominent policy gradient algorithms are discussed.

3.10.1 Model Free RL

In model free RL, the agent does not have access to, or does not learn, a model of the environment. The agent either learns the optimal policy or an approximate function for the optimal action-value function.

Policy Optimisation

Policy optimisation methods are used to optimise the parameters θ of a policy $\pi_\theta(a|s)$. The parameters can either be optimised directly on the performance objective $J(\pi_\theta)$, by using gradient ascent, or indirectly by maximising local approximations of $J(\pi_\theta)$.

Policy optimisation methods are usually on-policy, which means that parameter updates are only performed using data collected while acting according to the most recent version of the policy. Policy optimisation methods can also be used to learn an approximate value function $V_\phi(s)$ for the on-policy value function $V^\pi(s)$. The approximate value function is used in updating the policy.

Example of policy optimisation algorithms are:

- Asynchronous actor-critic / asynchronous advantage actor-critic (A2C/A3C), which uses gradient ascent to directly maximise $J(\pi_\theta)$ (Mnih et al. 2016).
- Proximal Policy Optimisation (PPO), which indirectly maximises $J(\pi_\theta)$, by using a surrogate objective function that estimates how $J(\pi_\theta)$ will change as a result of an update (Schulman et al. 2017).

Q-learning

Q-learning methods are used to learn an approximate function $Q_\pi(s, a)$ for the optimal action-value function $Q^*(s, a)$. Optimisation of these functions are performed off-policy, meaning that parameter updates can be made using data collected at any point during training, regardless of the current policy. The actions taken by Q-learning agents are given by:

$$a(s) = \arg\max_a Q_\pi(s, a) \tag{3.53}$$

Example Q-learning methods are:

- Deep Q-learning Network (DQN), which was one of the first breakthrough deep learning model approaches to learn control policies from raw pixels by learning a value function estimate using Q-learning (Mnih et al. 2015).
- Categorical 51-atom DQN (C51), is a variant of DQN that learns a distribution over the return with an expectation equal to Q^* (Bellemare et al. 2017).

Trade-offs Between Policy Optimisation and Q-learning

Policy optimisation methods are on-policy, which means that parameter updates are only made using the current policy. The policy is being optimised directly, which makes the optimisation more stable and reliable, since any changes that caused worse performance could be reversed in the next iteration. The limitation to policy optimisation methods, that are on-policy, is that sample efficiency is less than off-policy Q-learning methods since only data collected from the current policy can be used to update parameters. Q-learning methods can however have several failure modes and can be less stable to train (Tsitsiklis & Van Roy 1997).

3.10.2 Model Based RL

In model based RL the agent has access to the state transition function and reward function, or learns a model of these functions. If the agent has a model of the environment, the agent can think ahead to determine what would happen if a range of different actions were taken and explicitly decide between different actions to plan ahead. Without a ground-truth model of the environment, the agent needs to learn a model from experience. With the ability to plan ahead, model based methods have a much higher sample efficiency over model free methods.

Pure Planning

Pure planning techniques like model-predictive control (MPC) can be used to select actions. Using MPC, for each observation the agent computes an optimal plan with respect to the agent's model of the environment. The agent executes the first action of the plan, and computes a new plan with each interaction with the environment. Model-Based RL with Model-Free Fine-Tuning (MBMF) explores pure planning with MPC for model based RL (Nagabandi et al. 2018).

Expert Iteration

Instead of pure-planning, an explicit representation of the policy $\pi_\theta(a|s)$ is learnt. The representation of the policy is then used with a planning algorithm, like Monte Carlo Tree Search (MCTS), to sample actions and generate a plan with actions that is better than the policy alone would have produced. The policy is updated to produce actions more like the planning algorithm produced. AlphaZero is an example of a model based method that uses MCTS to generate an expert iteration plan (Silver et al. 2016).

Data Augmentation for Model-Free Methods

In data augmentation for model-free methods, a model-free algorithm is used to train a policy or Q-function, but agent updates are only made using augmented or fictitious experiences from a model of the environment.

Examples of data augmentation methods are:

- Model-Based Value Expansion (MBVE) augments real environment experiences with fictitious experiences to update the agent (Feinberg et al. 2018).
- World Models, learn a representation of the environment and only use fictitious experiences to update the agent, which the authors call "learning inside their own dreams" (Ha & Schmidhuber 2018).

Embedding Planning Loops into Policies

With these methods, a model-free algorithm can be used to train a policy that includes a model-based representation of the environment used to "imagine trajectories" that is used as a planning procedure. The policy can learn to choose how and when to use the plans. In the case that the model-free policy can select an action that would perform better than the planned action in the given state, the plan is ignored. Imagination Augmented-Agents (I2A) learns an environment representation that is used to plan ahead and adds the planning loops into model-free methods. I2A is more robust than other model-based methods, as it can fall back to the strengths of model-free methods where the representation of the environment model under certain states is not learnt yet (Racanière et al. 2017).

3.10.3 Policy Gradient Algorithms

The central premise of policy gradient algorithms are to push up the probabilities of actions that lead to higher returns and to push down the probabilities of actions that lead to lower returns, until a near optimal policy is achieved. Vanilla Policy Gradient is the simplest algorithm, but the least efficient and prone to getting stuck in local optima. Trust Region Policy Optimisation introduces a surrogate loss to prevent policy updates from getting too large and ensures monotonically improving performance, however can be complex to implement and is not suited for all deep learning architectures. Proximal Policy Optimisation improves on previous methods by introducing a simple surrogate loss that also aims to prevent policy updates from getting too large while remaining efficient, easy to implement and suited for any deep learning architecture.

Vanilla Policy Gradient

The Vanilla Policy Gradient (VPG) algorithm, introduced in Algorithm 1, is an on-policy algorithm that estimates the gradient of the expected return (Williams 1992, Sutton et al. 2000). It is considered to be the most basic policy gradient algorithm, however has some limitations. It can be hard to choose a single appropriate step-size, α, for optimisation which results in large changes in policy distribution with a single step. As training progresses the policy becomes progressively less random, as the policy update rule favours exploiting known rewards, which causes the policy to converge to local optima resulting in sub-optimal behaviour (Schulman 2016). By keeping the step size small this unwanted behaviour is prevented, however training is slow and poor sample efficiency is achieved.

Algorithm 1 Vanilla Policy Gradient Algorithm (Williams 1992)

1: Input: Initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards $\hat{R}_t$.
5: Compute advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_k}.
6: Estimate policy gradient $\hat{g}_k$ as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\sigma \log \pi_\theta \left(a_t | s_t\right) |_{\theta_k} \hat{A}_t .$$

7: Compute policy update, using gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k .$$

8: Fit value function

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

9: **end for**

Trust Region Policy Optimisation

The Trust Region Policy Optimisation (TRPO) algorithm, introduced in Algorithm 2, is an on-policy algorithm that estimates the gradient of the expected return using a surrogate loss which updates the policy by taking the largest possible step to improve performance, while ensuring that the Kullback-Leibler (KL) divergence between the new and old policies is not too large (Schulman et al. 2015).

Compared to VPG, TRPO improves sampling efficiency by taking the biggest possible optimisation step size. TRPO is a second order method which makes it complicated to implement and not compatible with deep learning architectures that include parameter sharing between the policy and value functions (Schulman et al. 2017). Fortunately other policy optimisation algorithms, such as Proximal Policy Optimisation, can overcome these deficiencies.

Algorithm 2 Trust Region Policy Optimisation Algorithm (Schulman et al. 2015)

1: Input: Initial policy parameters θ_0, initial value function parameters ϕ_0
2: Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5: Compute rewards $\hat{R}_t$.
6: Compute advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_k}.
7: Estimate policy gradient $\hat{g}_k$ as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\sigma \log \pi_\theta \left(a_t|s_t\right)|_{\theta_k} \hat{A}_t .$$

8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.
9: Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k$$

10: Fit value function

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

11: **end for**

Proximal Policy Optimisation

The Proximal Policy Optimisation (PPO) algorithm, introduced in Algorithm 3, is an on-policy algorithm that estimates the gradient of the expected return using a surrogate loss, similar to TRPO, which updates the policy by taking the largest possible step to improve performance without falling into a local optimum (Schulman et al. 2017). Contrary to TRPO that uses a complex second-order method, PPO uses a simpler first-order method to ensure that update steps can be sufficiently large such that in each update step the difference between new and old policies are not too large.

PPO can be implemented in two variants, either using an approximate KL-divergence penalty approach, similar to TRPO, or a specialised clipping objective that prevents the new policy from deviating too far from the old. Schulman et al. (2017) state that the PPO algorithm is simpler to implement than TRPO, works with deep learning architectures that include parameter sharing between policy and value functions and found that the clipped PPO algorithm achieves better performance when compared to other policy gradient methods.

The clipped PPO algorithm maintains parameters for two policy networks, firstly the current policy to be optimised $\pi_\theta(a_t|s_t)$ and the old policy $\pi_{\theta k}(a_t|s_t)$ that is used to collect samples. By comparing the probability ratio between the current and old policies, it is possible to maximise the surrogate

objective, as done with TRPO:

$$L(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k}(a_t|s_t)} \hat{A}_t \right] \tag{3.54}$$

with the probability ratio denoted as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k}(a_t|s_t)} \tag{3.55}$$

The probability ratio measures how different the two policies are and $r_t(\theta) = 1$ when the new and old policies are the same. The goal of the clipped PPO algorithm is thus to penalise changes to the policy that move $r_t(\theta)$ away from 1. This is achieved by using the clipped surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{3.56}$$

where ϵ is a hyperparameter, typically $\epsilon = 0.2$. The first term in the L^{CLIP} objective is the same as the TRPO surrogate objective. The second term, $\text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t$, modifies the surrogate objective by clipping the probability ratio in the interval $[1-\epsilon, 1+\epsilon]$. By taking the minimum between the first and second terms, a lower bound on the unclipped objective is obtained.

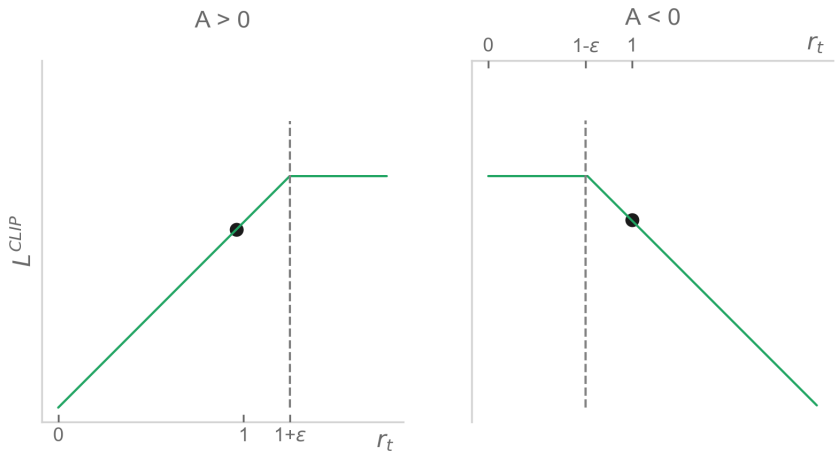


Figure 3.4: Clipped PPO surrogate objective function for a positive and negative advantage (Schulman et al. 2017).

Figure 3.4 shows a single time step in L^{CLIP}. The probability ratio $r_t(\theta)$ is clipped at $1-\epsilon$ when the advantage is negative, while it is clipped at $1+\epsilon$ when the advantage is positive. The black circle marker on each plot shows the starting point for optimisation when $r_t(\theta) = 1$. As shown, once the probability is outside the interval $[1-\epsilon, 1+\epsilon]$ the advantage function is clipped to prevent large policy updates that could result in sub-optimal performance.

Algorithm 3 Clipped Proximal Policy Optimisation Algorithm (Schulman et al. 2017)

Input: Initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards $\hat{R}_t$.
  Compute advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_k}.
  Update the policy by maximising the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k}(a_t|s_t)} A^{\pi_{\theta k}}(s_t, a_t), g\left(\epsilon, A^{\pi_{\theta k}}(s_t, a_t)\right) \right)$$

  Fit value function

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

end for

Chapter 4

Problem and Data Definition

In this chapter, the problem is described. Furthermore, the data is defined that will be used to simulate a real-world operational environment.

4.1 Problem Description

The simulated problem that will be investigated, is a maintenance strategy optimisation problem that will be designed to resemble a real-world operational system as closely as possible.

An air freight company with a fleet of identical aircraft using identical turbofan engines delivers cargo 24/7/365. For every hour that an engine is down for maintenance, it is assumed that an aircraft is grounded, resulting in lost income. There is a maintenance team with limited resource capacity. If too many engines are taken down for maintenance, or if too many engines fail at the same time, the maintenance team will start backing up work resulting in further lost income. It is assumed that if an engine is stopped for maintenance before failure the time spent on maintenance is much less than if a failure needs to be repaired. The company has instrumented all engines with sensors and can monitor the condition of an engine at any point in time. Using the sensory information from engines in the fleet as well as the planned maintenance schedule, the company needs to decide when to schedule and stop engines for maintenance. The company needs to determine the optimal maintenance strategy in order to ensure maximum availability of engines across the fleet.

A Deep Reinforcement Learning solution will be used to find the optimal decision making and resource allocation policy for when to stop a specific engine for maintenance, considering the health of other engines in the fleet as well as the current and future maintenance resource availability.

4.2 Data Description

The Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) is a tool for simulation of large commercial turbofan engines developed by NASA (Parker & Guo 2003). C-MAPSS simulates a turbofan engine model of the 400kN thrust class, with an atmospheric model capable of simulating operations at altitudes ranging from sea level to 12.192 km, Mach numbers from 0 to 0.90, and sea-level temperatures from −51.1 to 39.4 °C. Figure 4.1 shows a simplified diagram of the different components in the simulated turbofan engine.

Using the simulation model, Saxena et al. (2008) created five data sets under varying conditions, known as the PHM08 Challenge Data set. Data sets consist of multiple multivariate time series. Each time series is from a different turbofan engine and the data can be considered to be from a

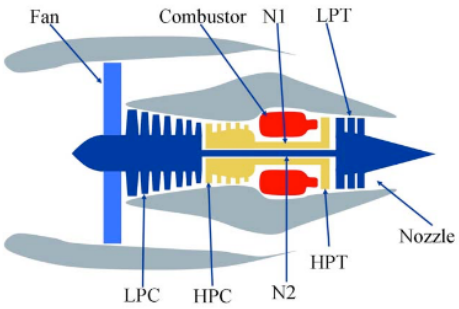


Figure 4.1: Simplified diagram of the engines simulated by C-MAPSS. (Saxena et al. 2008)

fleet of engines of the same type. Throughout the simulations, wear parameters were varied to simulate continuous degradation trends. Each engine starts with different degrees of initial wear and manufacturing variation which is unknown. Data was captured from various sensors on parts of the system, recording the effects of degradation on sensor measurements as time progressed until the engine fails. The data is also contaminated with sensor noise. There are a total of 709 unique training trajectories in the data set.

Table 4.1: Description of sensors included in the C-MAPSS data set. (Saxena et al. 2008)

Sensor	**Description**	**Units**
S1	Altitude	ft
S2	Mach Number	Ma
S3	Throttle Resolver Angle	deg
S4	Total temperature at fan inlet	°R
S5	Total temperature at Low-Pressure Compressor outlet	°R
S6	Total temerature at High-Pressure Compressor outlet	°R
S7	Total temperature at Low-Pressure Turbo outlet	°R
S8	Pressure at fan inlet	psia
S9	Total pressure in bypass-duct	psia
S10	Total pressure at High-Pressure Compressor outlet	psia
S11	Physical fan speed	rpm
S12	Physical core speed	rpm
S13	Engine pressure ratio	–
S14	Static pressure at High-Pressure Compressor outlet	psia
S15	Ratio of fuel flow to pressure at High-Pressure Compressor outlet	pps/psi
S16	Corrected fan speed	rpm
S17	Corrected core speed	rpm
S18	Bypass ratio	–
S19	Burner fuel-air ratio	–
S20	Bleed enthalpy	–
S21	Demanded fan speed	rpm
S22	Demanded corrected fan speen	rpm
S23	High-Pressure Turbo coolant bleed	lbm/s
S24	Low-Pressure Turbo coolant bleed	lbm/s

Figure 4.2 shows a single sampled trajectory from the C-MAPSS data set. Each row represents a single sensor over time. Most sensor values start nominally and end up changing significantly over time, until failure. The health index time series for each individual turbofan engine example is not

directly included in the sensor measurements. Failure for each turbofan engine is when its censored health index time series decreased to zero. The fixed time intervals between the sensor measurements and the time of failure for the turbofan engines were arbitrarily measured in cycles. Figure 4.3 shows the distribution of different cycles, or sequence lengths, in the data set.

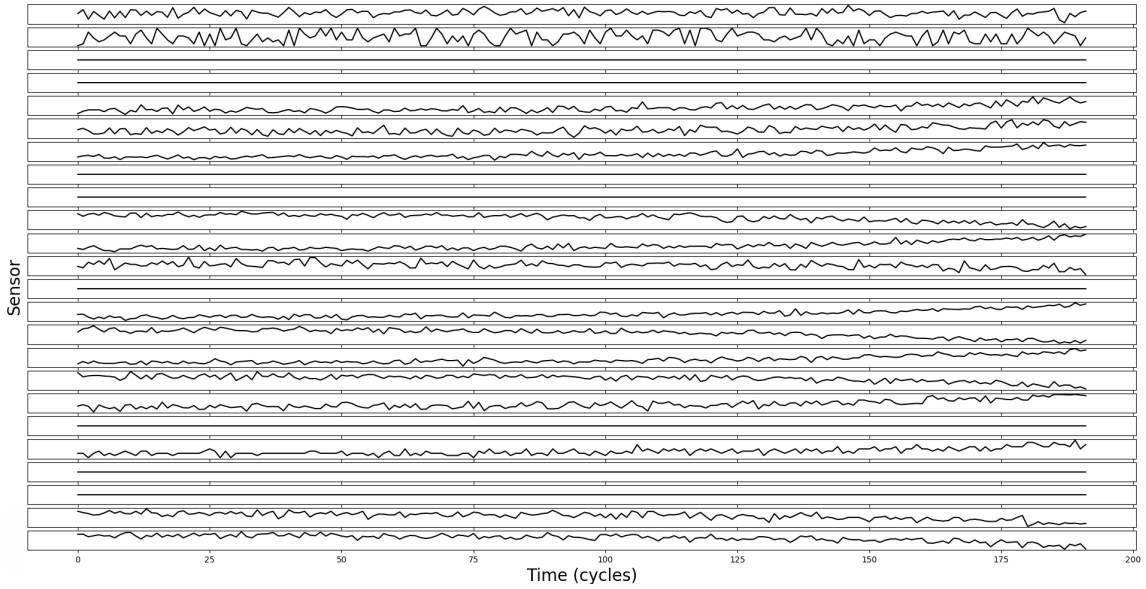


Figure 4.2: A sample of sensor values over time for a single engine. Sensor values start nominally at $t = 0$ and start to degrade over time, until failure at $t = 190$.

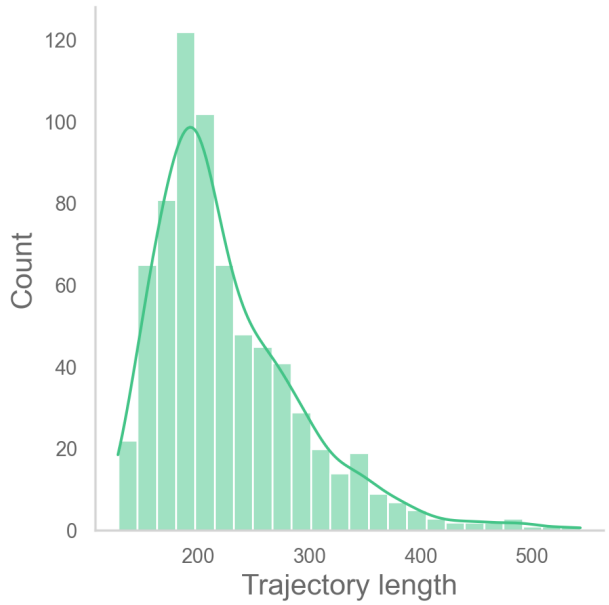


Figure 4.3: Histogram of C-MAPSS PHM08 data set run-to-failure trajectory sequence lengths.

The data sets possess unique characteristics that make them very useful and suitable for developing prognostic algorithms:

1. Data represent a multi-dimensional response from a complex non-linear system from a high fidelity simulation that very closely models a real system.

2. These simulations incorporate high levels of noise introduced at various stages to accommodate

the nature of variability generally encountered.

3. The effects of wear are masked due to operational conditions, which is yet another common trait of most operational systems.

4. The data sets are ideal for data-driven approaches where very little or no system information is available, similar to real-world operational systems.

As discussed in Section 1.4, the characteristics of the C-MAPSS data set has made it very popular for research on developing prognostic algorithms. It is also shown that the C-MAPSS data set can be used for developing health management solutions for maintenance scheduling (Skordilis & Moghaddass 2020). In a typical reinforcement learning environment, a direct simulation or model of the environment is available for agents to interact with. As explained in detail in Section 5.1, in this work a simulation environment is developed that samples from the static C-MAPSS data set in order to simulate the degradation of engines rather than using the underlying high-fidelity simulation of the engines that was developed by NASA.

The significance of this approach is that the developed simulation environment could be used with any other static data set that is significantly large enough and representative of a real operational environment to train a new policy for that environment. The C-MAPSS data set is unique, when compared to several other publicly available data sets that are used for prognostic and health management research because it was generated using a high-fidelity simulation over the entire life-cycle of several engines, rather than measured from small scale experiments that are focused only on a small portion of equipment life. This means that the size of the data set, as well as the range of parameters simulated is significant enough to represent the operation of a turbofan engine over its entire life-cycle. Even though a single data set is used to evaluate the implemented approach, it is noted that the goal is not to achieve state of the art performance on prognostic decision making or maintenance scheduling specifically, and is however to demonstrate that a DRL approach can be used for optimal decision making in a multi-component system with resource constraints.

Chapter 5

Methodology

In this chapter, the methodology used to simulate the environment as well as the agent implementation is explained. The simulated environment relates to the problem and data definitions described in Chapter 4. The Deep Reinforcement Learning agent is compared to traditional maintenance methodologies, as further discussed.

5.1 Simulated Environment

Typical reinforcement learning research environments are games, such as Atari, Go or Dota (Mnih et al. 2015, Silver et al. 2017, OpenAI et al. 2019). In game environments the agent, acting as the player, can observe the game environment or state, take some action and receive an associated reward. Games naturally have a score, or an end goal, that can implicitly be used as a reward, or to end an episode, when the game has been won or lost. These properties make game environments naturally suited for developing reinforcement learning algorithms. In reality these game environments are not too different from real-world operational environments.

In order to simulate a real-world operational environment for training a reinforcement learning algorithm, the following components are required:

1. a simulated environment of the operational system that can be stepped through in time, such as the sensor readings from the C-MAPSS turbofan engines.
2. a task or (possibly a set of) actions, that an agent needs to perform to reach some end state, such as letting the engine continue to run, or to stop for maintenance before failure.
3. a measure of performance, or score, that can be used as a reward or penalty to tell the agent how well it is performing. The reward is used to improve the agent's decision making policy, such as rewarding the agent for stopping the engine for maintenance before it breaks, or penalising it for failing to stop the engine.

The objective is to create a simulated real-world operational environment where multiple agents can interact and coordinate their decision making and resource allocation actions in order to optimise the up-time of the controlled assets in the operational environment.

The simulated environment was developed as a custom multi-agent environment using the RLlib reinforcement learning library. RLlib is an open-source library for reinforcement learning that offers both high scalability and a unified API for a variety of applications (Liang et al. 2018). The following sub-sections will describe the developed environment's components in more detail.

5.1.1 State and Observation Space

Figure 5.1 shows a flowchart of the agent-environment interaction loop and is described in more detail below.

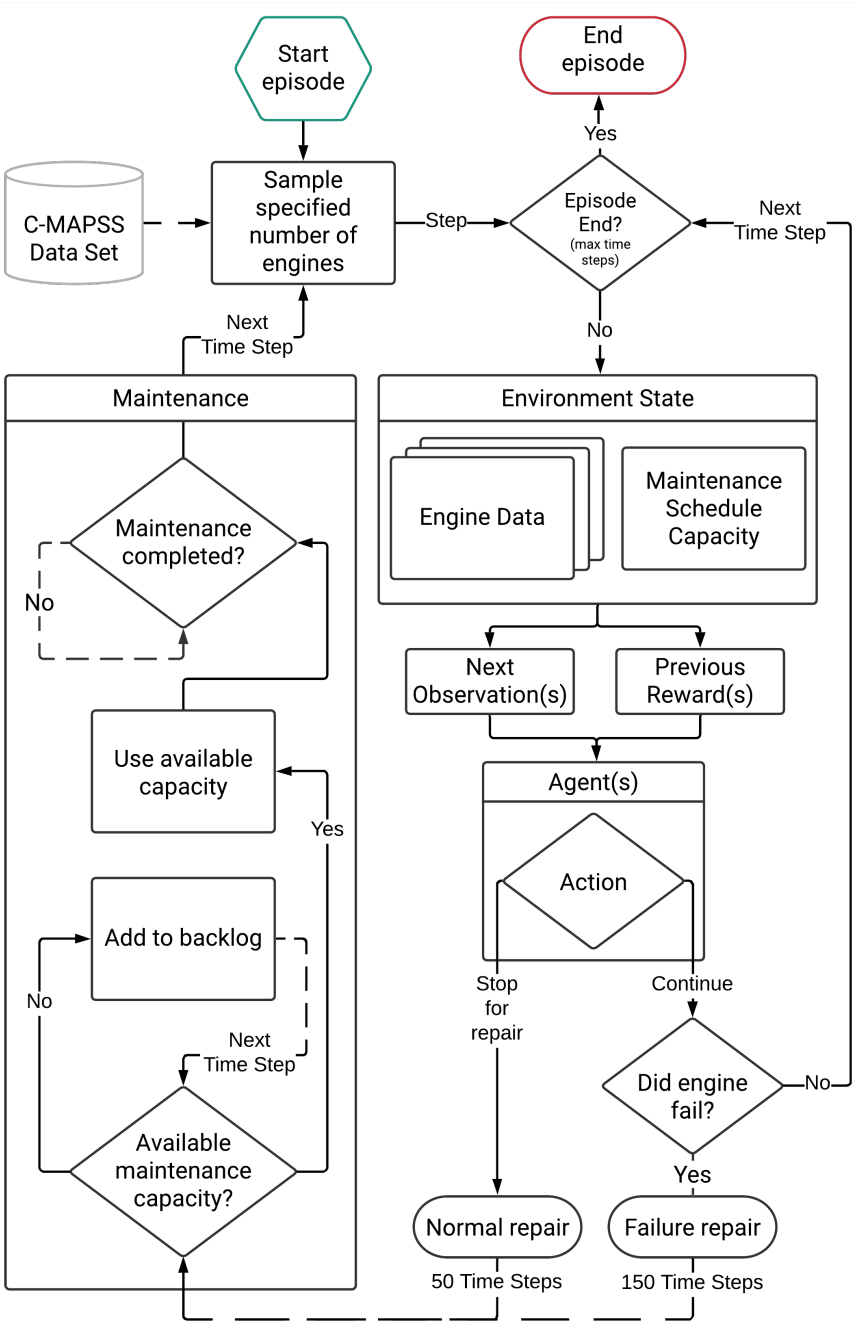


Figure 5.1: Flowchart of the agent-environment interaction loop for the simulated maintenance problem.

At the start of each episode, the number of turbofan engines to be controlled is specified. The specified number of engine run-to-failure trajectories are then randomly sampled from the C-MAPSS data set. In the C-MAPSS data set there are over 700 unique run-to-failure time series. Each engine is controlled by a single agent, so for example if a fleet of four engines are selected then four agents will also be created. One agent to control each individual engine in the fleet.

At each time step the sensor readings from each engine are used as the observations for the respective controlling agent. To ensure greater variability, a small amount of noise is also added to each sensor. Each agent also observes the current available maintenance capacity at each time step. Once an engine is stopped for maintenance, or because of failure, a new engine trajectory is sampled from the data set – resembling an engine returning to full health after maintenance. Since each engine's trajectory is unique and fails at different time steps, or because agents can take independent actions to stop an engine for maintenance, the system is stochastic and resembles a real-world operational environment. In a real-world environment, the health of individual assets naturally deteriorates differently over time, depending on variations in manufacturing tolerances, variations in operating conditions or even operator abuse.

In a real-world operational environment, there can be several objectives and constraints associated with a maintenance strategy. In this simplified maintenance strategy, the maintenance team only has a finite resource capacity. In the simulated environment any maintenance that is scheduled, but exceeds the current capacity, is delayed until capacity frees up. This adds a team dynamic to the environment where each agent needs to consider the actions of the other agents, as well as the available current and future maintenance capacity. To simulate the resource capacity, if an engine is stopped for preventative maintenance before failure, a fixed number of steps (for example 50 time steps, or 50 hours) are allocated. If an engine fails before being stopped, corrective maintenance usually takes longer to complete and a larger number of steps (for example 150 time steps, or 150 hours) are allocated.

The simulated environment continues to run for a fixed time period. Agents continue to interact with the environment over this period, with the objective of maximising the uptime of the fleet of engines under control, while also ensuring maintenance capacity is available. At the end of the time period, the environment reaches a terminal state and resets. Each sub-sequence of agent-environment interactions between the initial and terminal state represents an episode.

Table 5.1: List of environment variables.

Parameter	Experimental value	Description
Number of agents	4	agents, or engines, in the fleet
Number of steps per episode	2500	steps over which performance is measured
Maintenance capacity	50	allocatable steps for maintenance
Time to repair, before failure	50	steps to repair if stopped for maintenance
Time to repair, after failure	150	steps to repair if failed

The variables in Table 5.1 show some of the parameters that can be adjusted in the simulated environment. For the experiments performed, the values are fixed and compared across all approaches. The same random seed is used to ensure that the same engine trajectories are sampled when comparing the implemented solution to the traditional maintenance strategies. Figure 5.1 shows a flow diagram of the agent-environment interaction loop.

5.1.2 Action Space

The goal for each agent is to prevent engine failure, but also not to stop an engine for maintenance too early. The agent can take one of two discrete actions at each time step: stop the engine for maintenance, or let the engine continue to run.

5.1.3 Reward

In order for the agent to learn the different objectives of the simulated environment, there needs to be different rewards associated with each objective. This is referred to as the credit assignment problem. The agent learns to associate actions, or a combination of actions over time, with an eventual reward.

The first objective for an agent is to prevent engine failure. Using reward shaping, the reward grows exponentially as the engine reaches the end of its life. This encourages the agent to accumulate the maximum reward by stopping the engine close to its end of life. If the agent however fails to stop the engine before failure, a large negative reward is given to discourage agents from failing to prevent failure.

$$r_{t_i} = \begin{cases} 1 - \left(\left(t_{\text{failure}} - t_i\right) / t_{\text{failure}}\right)^{0.35} & \text{if } t < t_{\text{failure}} \\ -t_{\text{failure}}/2 & \text{if } t = t_{\text{failure}} \end{cases} \tag{5.1}$$

The second objective of all agents is to maximise the fleet's collective uptime over an episode, by ensuring that there is always sufficient maintenance capacity. Agents need to take actions and allocate resources effectively in order to ensure that the collective uptime over an episode is optimal. This could mean taking actions that reduce an agent's individual reward, by scheduling individual maintenance early in order to free up maintenance schedule capacity for other engines. The group reward is given as a fraction of the current maintenance capacity over the capacity set-point per step.

$$r_{t_i} = \frac{c_t}{c_{sp}} \times \frac{1}{n} \tag{5.2}$$

where:

$$\begin{aligned} c_t &= \text{maintenance capacity at time } t \\ c_{sp} &= \text{maintenance capacity set-point} \\ n &= \text{number of steps per episode} \end{aligned} \tag{5.3}$$

The implemented reward functions implicitly capture the underlying system dynamics described by the asset performance metrics and operational performance metrics described in Section 2.3, however the rewards were constructed heuristically and perhaps not objectively by not explicitly using the metrics in the reward formulation. By explicitly incorporating asset performance metrics or operational metrics as part of the reward function the process of tuning or shaping the reward functions should become less user defined and more objective. It is also noted that as more constraints are introduced, the process of tuning or shaping the reward function might become even more complex, further increasing the necessity for an objective and pragmatic approach. It is noted that the goal of this work is not to achieve state of the art performance on prognostic decision making or maintenance scheduling specifically, and is however to demonstrate that a DRL approach can be used for optimal decision making in a multi-component system with resource constraints. The implemented reward functions did thus serve this purpose.

5.2 Agent Implementation

The Deep Reinforcement Learning agents learn the optimal maintenance strategy under constrained maintenance resource capacity. The agents observe the sensor readings from their individual engines, as well as the current maintenance capacity, and decides whether to let the engine continue running or to stop the engine for maintenance. If an agent decides to let an engine continue to run and the

engine has failed, it is rewarded negatively to prevent this action from being taken in the given state. If the engine is stopped for maintenance shortly before failure the agent is rewarded positively to ensure this action is taken when in the same state. Agents are also rewarded for ensuring the maintenance capacity is not exceeded and thus learn to take actions that ensure that optimal scheduling is achieved.

The agent was implemented using an RLlib TensorFlow model and configured to interact with the custom multi-agent environment. The implemented model uses the clipped PPO policy gradient algorithm, as explained in Section 3.10.3. The model network weights are trained using stochastic gradient descent.

The implemented model uses a fully connected neural network to learn feature representations of the environment's observations. The feature representations are further processed by a recurrent neural network, specifically using Long Short Term Memory (LSTM) cells, in order to capture any time dependencies over consecutive observations, such as a deviation in engine health over consecutive time steps. The output of the model is an action distribution that is evaluated in order to determine the next action.

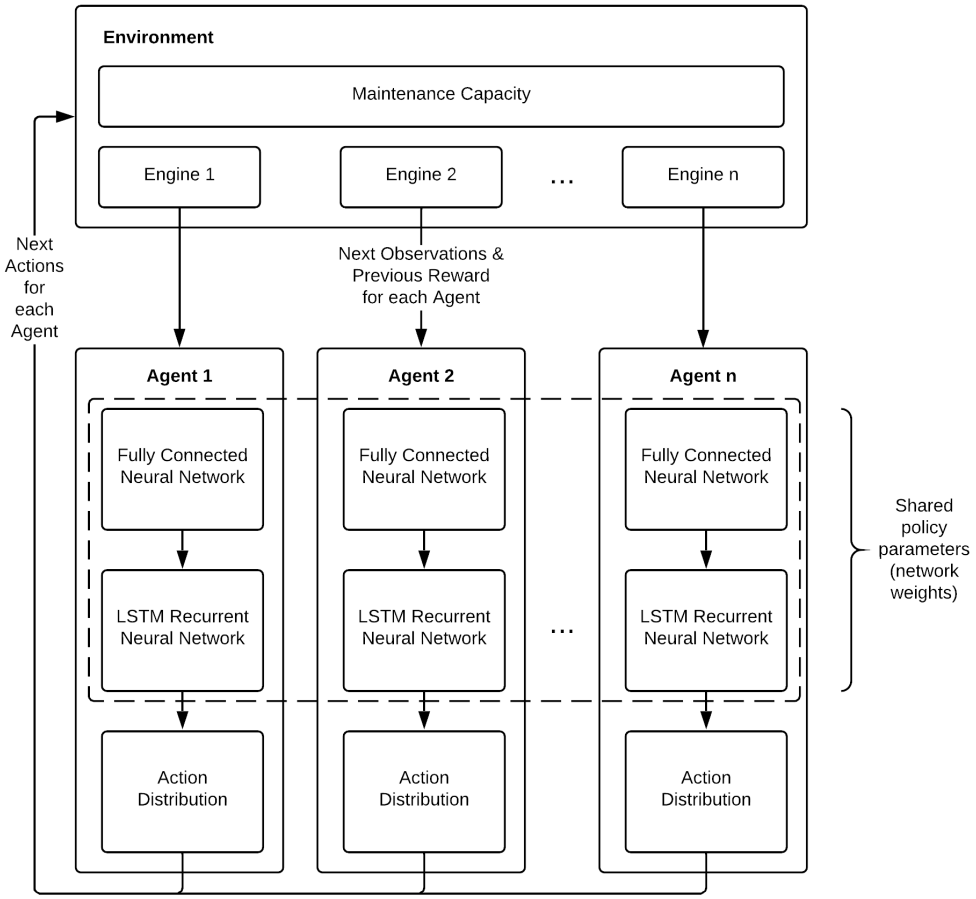


Figure 5.2: Flow diagram of the multi-agent deep learning architecture implementation.

In the multi-agent environment, agents were trained on individual observations, but the underlying network or policy weights and parameters were shared in order to learn a single policy that can be used by any agent in the fleet. Figure 5.2 shows a diagram of the implemented multi-agent architecture. In each time step, each agent receives an observation of its respective engine from the environment, the current maintenance capacity as well as a reward for the previous action after which each agent returns the next predicted action.

The implemented deep learning architecture and parameters are shown in detail in Appendix A.

5.3 Traditional Maintenance Strategy Implementations

In this section three alternative reactive and proactive maintenance strategy implementations, namely run-to-failure corrective maintenance, constant interval scheduled maintenance and condition based predictive maintenance are discussed.

5.3.1 Corrective Maintenance

Corrective maintenance is a reactive maintenance strategy, as explained in section 2.1.1. The implemented corrective maintenance strategy allows all engines to continue running until failure. This strategy does not consider the current health of any of the engines or the maintenance capacity and no repair interventions are made before failure.

5.3.2 Constant Interval Scheduled Maintenance

Constant interval maintenance is a proactive preventive maintenance strategy, as explained in section 2.1.2. In this strategy, maintenance is performed on constant intervals without considering the current health of any of the engines or the maintenance capacity. All historical failure trajectories are however used in order to perform a Weibull analysis, as described in section 2.2, that is further used to determine the optimum preventive maintenance time. Engines are scheduled for maintenance using this calculated optimum preventive maintenance time.

Weibull Analysis

The Weibull distribution, shown in equation 2.1, was fitted to the failure trajectories of the C-MAPSS engine data. Figure 5.3 shows the probability plot with the failure trajectories and the resulting parameters $\alpha = 110.6$, $\beta = 1.57$ and $\gamma = 127.12$. The Lilliefors test, a normality test based on the Kolmogorov-Smirnov test (Lilliefors 1967), was used to test the goodness of fit for the obtained parameters and achieved a P-value of 0.018, which suggest that test was significant and the fit is acceptable.Using the fitted parameters, the reliability function can be calculated using equation 2.4, as shown in equation 5.4.

$$R(t) = e^{-\left(\frac{t-\gamma}{\alpha}\right)^{\beta}} = e^{-\left(\frac{t-127.12}{110.6}\right)^{1.57}} \tag{5.4}$$

Optimal Preventive Maintenance Time

The optimal preventive maintenance time was calculated using equation 2.6, as shown in equation 5.5 and Figure 5.4. With the fitted $\beta > 1$, the condition for using the CPUT formula to calculate the optimal replacement time is satisfied. Furthermore, the following assumptions were made regarding maintenance costs:

1. Preventive maintenance cost $C_P = 50$
2. Corrective maintenance cost $C_U = 150$

$$\begin{aligned} CPUT(t) &= \frac{C_P R(t) + C_U\,(1 - R(t))}{\int_0^t R(t)dt} \\ &= \frac{50R(t) + 150\,(1 - R(t))}{\int_0^t R(t)dt} \end{aligned} \tag{5.5}$$

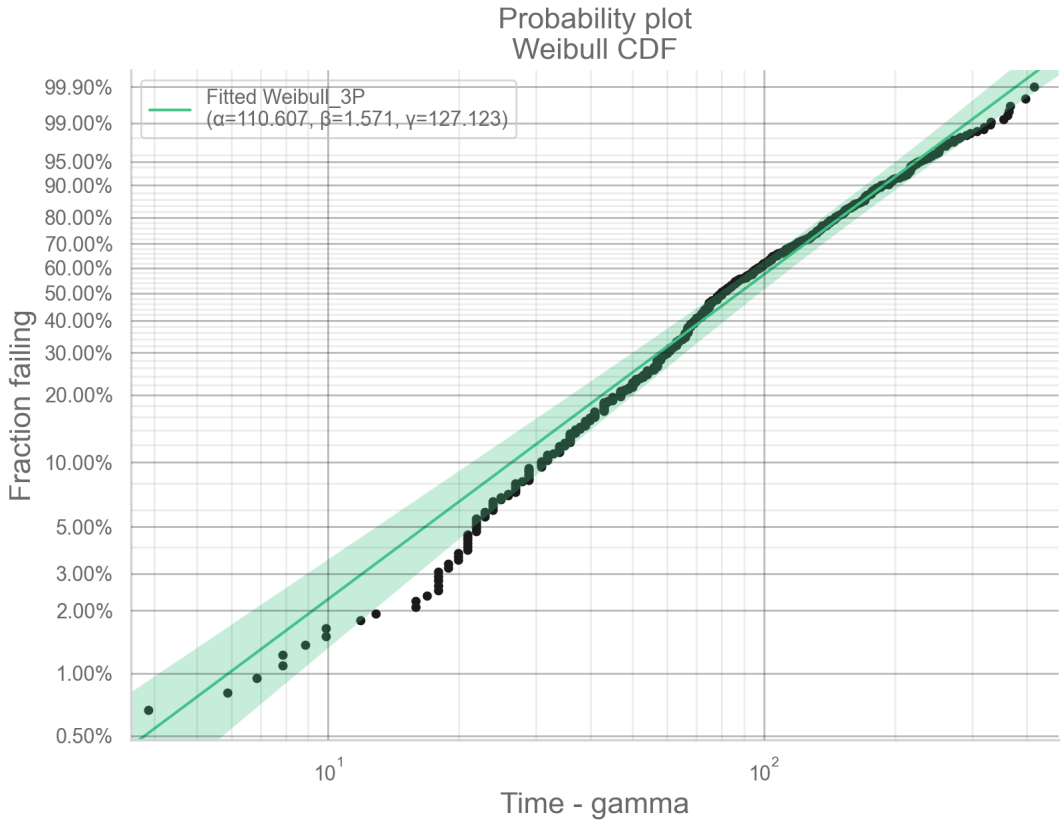


Figure 5.3: Fitted Weibull distribution probability plot.

The optimal preventive maintenance time was calculated as 143.94 time steps and a value of 144 time steps was used as the scheduled constant interval for the preventive maintenance strategy.

Table 5.2 shows the optimal replacement time for different corrective maintenance cost values. The range investigated represents a corrective maintenance cost that is 1.5, 2, 3 and 4 times greater than the preventive maintenance cost. A sensitivity analysis is performed to evaluate how the ratio of preventive maintenance cost to corrective maintenance cost influences the performance of the different strategies. For all other comparisons, the corrective maintenance cost of 150 is used.

Table 5.2: Optimal replacement times for different corrective maintenance cost values.

Capacity	C_P	C_U	Replacement Time
50	50	75	249
50	50	100	175
50	50	150	144
50	50	200	136

5.3.3 Condition Based Maintenance

Condition based maintenance is a proactive predictive maintenance strategy, as explained in section 2.1.2. In this strategy, maintenance decisions are made using equipment sensor telemetry to predict the real-time condition of the turbofan engines. An engine is stopped for maintenance once its condition deteriorates below a specified threshold.

The related prognostic algorithms, discussed in the related work Section 1.4, were used to decide what type of predictive model to implement for condition based maintenance. It was decided to

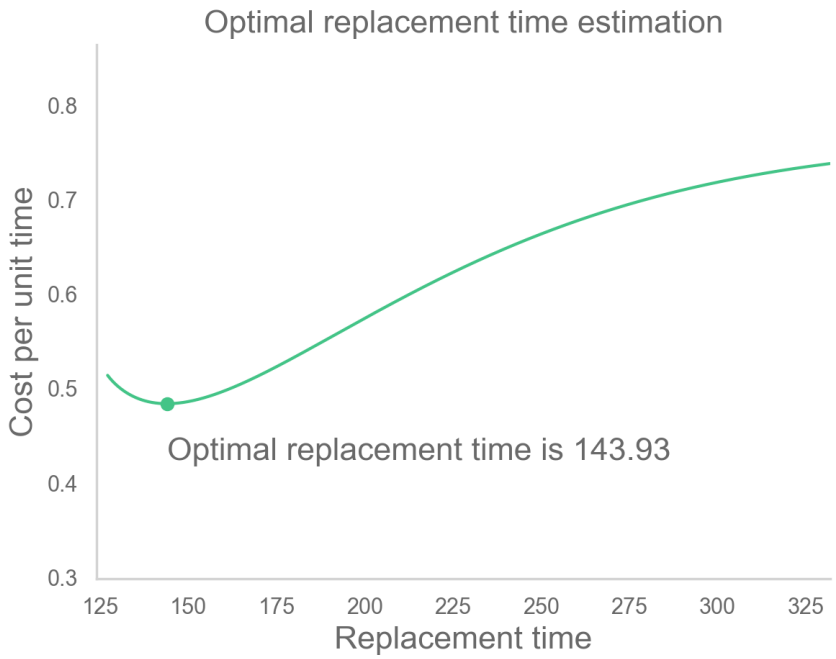


Figure 5.4: Optimal replacement time estimate using fitted Weibull parameters.

use a recurrent neural network approach, however instead of using a specific implementation an automated machine learning (AutoML) framework was used to perform a neural architecture search and architecture optimisation. AutoML is designed to reduce the demand for data scientists and enable domain experts to build high-quality custom machine learning or deep learning applications without significant statistical and machine learning or deep learning knowledge (He et al. 2019). The specific AutoML framework that was used is called AutoKeras. AutoKeras is an AutoML framework that uses the Keras deep learning framework to automatically build optimal deep learning models (Jin et al. 2019).

From the Weibull distribution that was fit in Section 5.3.2, it was determined that the mean time to failure (MTTF) for the C-MAPSS data set was 226 steps. Using the MTTF a stopping threshold of 20 steps from failure (8.85% mean remaining useful life) was selected. This value was selected to try and achieve a good asset utilisation. The neural architecture search and architecture optimisation process was performed to train a two class classification model to predict class 0 to continue running the engine (while the mean remaining useful life $>$ 8.85%) or class 1 to stop the engine for maintenance (once the mean remaining useful life $\leq$ 8.85%). This resembles the same decision making that the DRL agents would perform. A sequence of consecutive sensor measurements are passed to the model at each time step, after which the model predicts the next action to take. The resulting neural architecture and parameters that were used for the final model is shown in Appendix A.

The implemented condition based maintenance predictive model considers the real-time health of engines to stop engines for maintenance before failure, contrary to the corrective maintenance and constant interval maintenance strategies. The model does not consider the current available maintenance capacity, contrary to the implemented DRL agent.

Chapter 6

Results

This chapter discusses the applicability of the simulated production environment as well as the results and performance of the implemented Deep Reinforcement Learning solution compared to traditional corrective, preventive and predictive maintenance strategies.

6.1 Simulated Environment Applicability

It is required that the environment provides a true representation of the underlying system dynamics. The feedback any agent receives from the environment directly affects the ability of the agent to learn the desired behaviour. If the agents are thus able to learn the desired behaviour, it can be concluded that the environment is suitable for the simulated experimental purposes.

The following subsections discuss the techniques applied to improve learning and aid the convergence of the trained policies.

6.1.1 Curriculum Learning

Curriculum learning was used to assist agents in learning the complex task of stopping engines for maintenance as close as possible to failure while also considering the available maintenance capacity. The curriculum was broken into two phases, with the first phase allowing the agents to focus only on predicting the correct time of failure. The second phase considered both predicting the time of failure as well as the available maintenance capacity.

Figure 6.1 compares the training progress of the Deep Reinforcement Learning agents in an environment both using and not using curriculum learning respectively, by looking at the reward obtained by agents. During the first phase of curriculum learning, the agent learns to stop the engine as close to failure as possible. The reward quickly increases suggesting that the agent is constantly improving at the simple task. Once the second phase is initiated, the reward declines steeply, since the more complicated task now needs to be learnt. After some time, the reward starts to increase again and eventually converges to a stable value near 0.5.

Without using curriculum learning, agents need to learn the complex task from the start. It is seen that the reward slowly increases over time and then diverges, suggesting that the non-stationarity in the environment makes it difficult for agents to learn the underlying dynamics in relation to reward assignment. With sufficient exploration, the policy reward is able to increase again however continues to struggle to maintain the same level of reward per episode.

It is seen that curriculum learning greatly improves stability and convergence which allows agents to

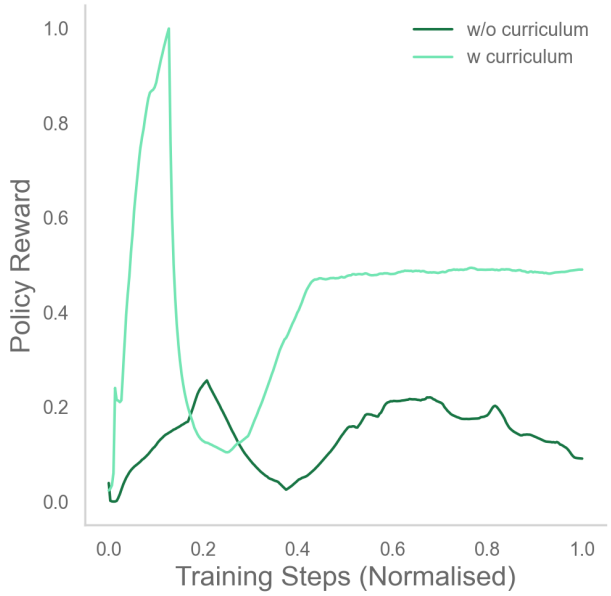


Figure 6.1: Comparison between achieved policy reward with and without curriculum-based learning.

learn the desired behaviour much faster. Without using curriculum learning, the learned policy can get stuck in a local maximum or even diverge resulting in sub-optimal or poor performance.

6.1.2 Reward Shaping

Reward shaping was used to guide agents toward stopping engines for maintenance as close to failure as possible. The goal of this is to lead agents towards choosing actions that result in better performance and easier training convergence.

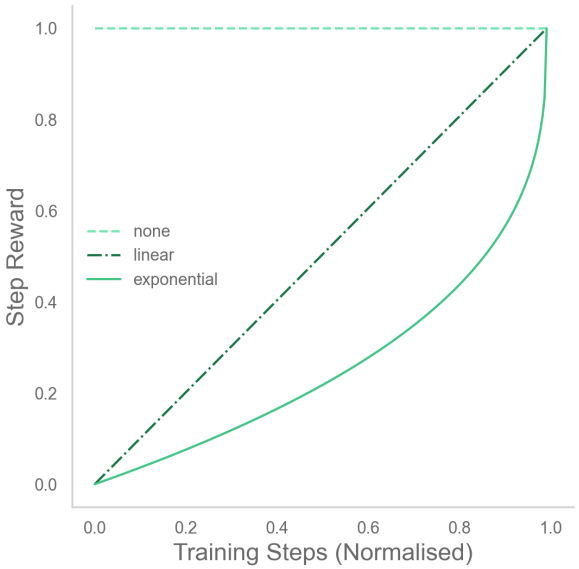
Figure 6.2a shows the individual reward function introduced in 5.1. The shaped reward function emphasises rewards obtained closer to the time of failure exponentially. A linear reward shaping function was also compared. The unshaped reward is simply $y = 1$ for each training step. In all comparisons, the agents receive a single negative reward upon failure.

Figure 6.2b compares the policy reward obtained during training using the linear, exponential and no reward shaping functions. In all experiments, curriculum learning is used to ensure that the same stable convergence behaviour is achieved. During the first phase of curriculum learning, it is shown that all reward functions achieve some maximum reward before switching to the second phase. During the second phase of curriculum learning, the exponentially shaped reward is able to converge to a higher relative reward compared to the linearly shaped reward. By not using reward shaping the agents do not learn fast enough and do not converge in the same number of training steps as the agents trained using reward shaping.

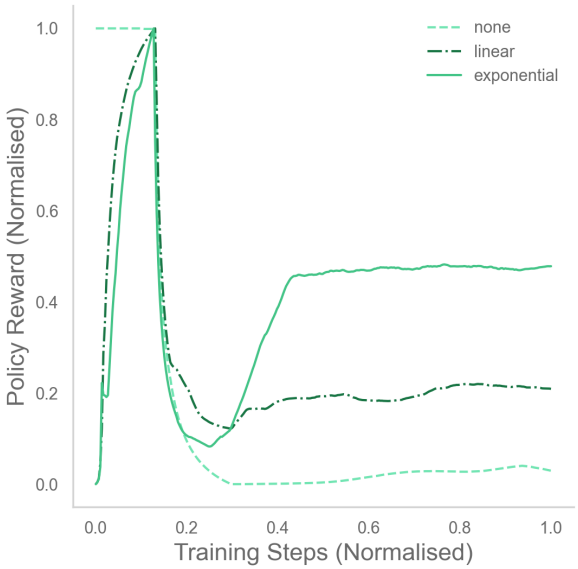
The sub-optimal performance of not using reward shaping is further highlighted in Figures 6.2c and 6.2d. Figure 6.2c shows that by using no reward shaping the agents struggle to learn the underlying dynamics of the system or fails to significantly improve uptime. Figure 6.2d shows that the agents are also unable to learn how to improve maintenance capacity without using reward shaping.

6.2 Maintenance Strategy Performance

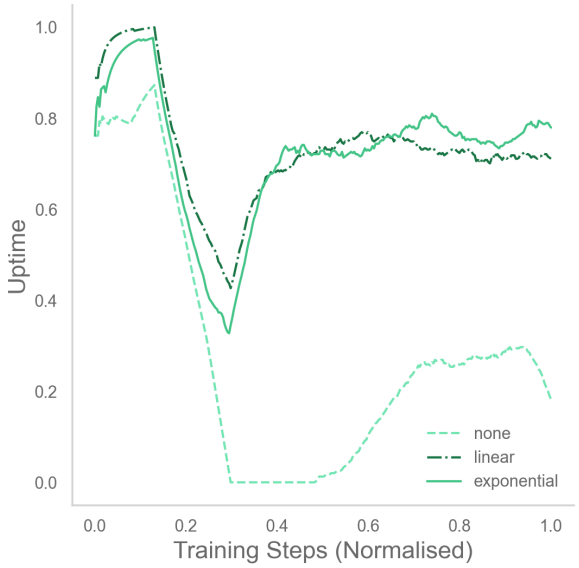
After model training converged, the different maintenance strategies were compared on the same randomly sampled trajectories over a single episode using the same environment parameters that were



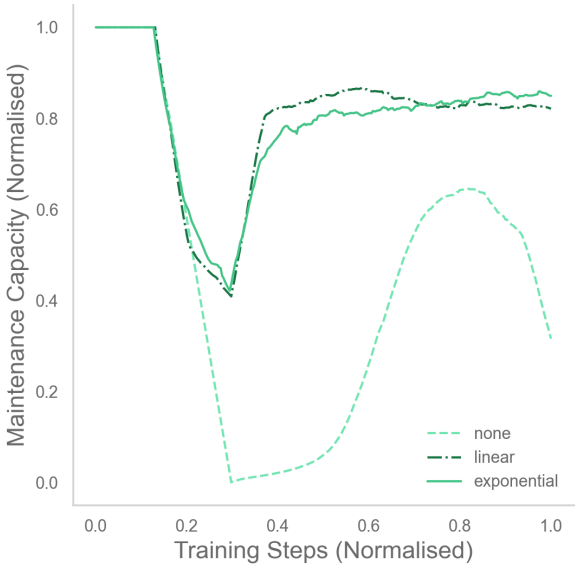
(a) Step rewards for different reward shaping functions over training.



(b) Policy reward during training using different reward shaping functions.

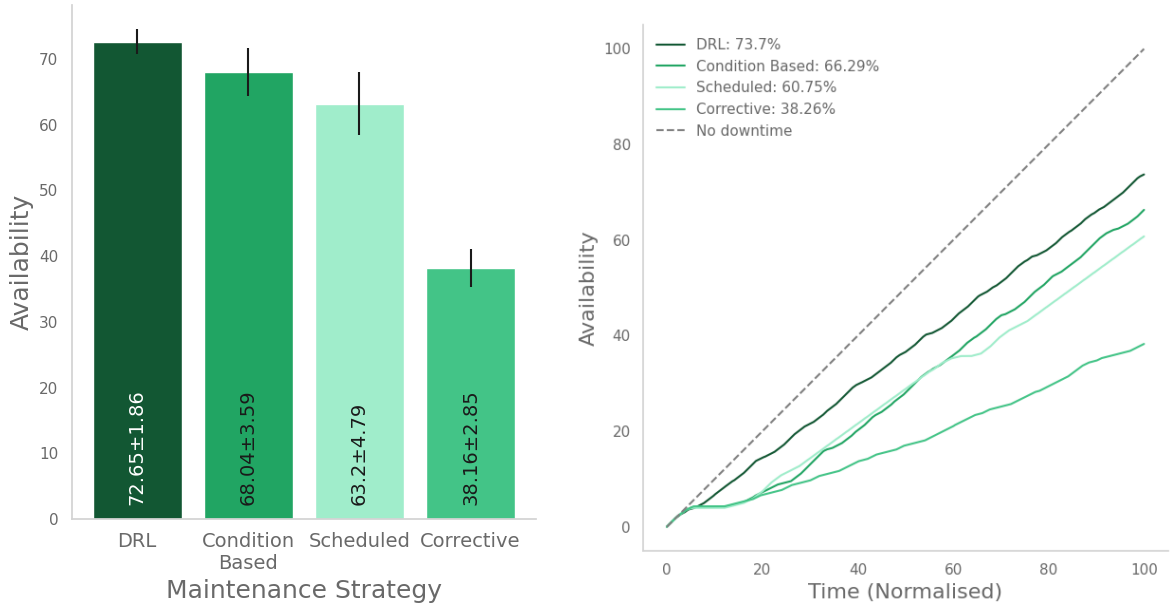


(c) Uptime during training using different reward shaping functions.



(d) Maintenance capacity during training using different reward shaping functions.

Figure 6.2: Comparison between using different reward shaping functions and no reward shaping.



(a) Availability over multiple episodes.

(b) Availability over a single episode.

Figure 6.3: Comparison of availability between different maintenance strategies over multiple episodes (a) and a single episode (b).

used during training. These experiments were repeated a thousand times to determine the mean and standard deviations for the reported metrics.

6.2.1 Availability and Uptime

Availability is a maintenance performance metric that has a direct impact on the production capability of a system and thus significantly influences revenue. The implemented environment ensured that all engines were scheduled to operate over the entire episode, meaning that the availability and uptime metrics are equal.

Figure 6.3a compares the availability achieved using the different maintenance strategies over several episodes. As expected, it is seen that the run-to-failure corrective maintenance strategy achieved the worst availability of 38.16 ± 2.85%. The scheduled preventive maintenance strategy achieved significantly better availability of 63.2 ± 4.79%. The condition based predictive maintenance model improved on the performance of the scheduled maintenance approach, with a mean availability of 68.04 ± 3.59%, however performed slightly worse than the Deep Reinforcement Learning based approach. The implemented Deep Reinforcement Learning based predictive maintenance strategy achieved the highest availability of 72.65 ± 1.86%.

From a downtime perspective, the corrective maintenance strategy results in more time being spent doing maintenance than in production. On average over an entire episode, the fleet was only operational for 38.16% of the time or effectively that 61.84% of the time over the episode was downtime for failure repairs.

The scheduled maintenance strategy achieved an average uptime of 63.2%, which means that only 36.8% of the time over the episode was spent on scheduled maintenance. This is an improvement of 40.49% over the run-to-failure corrective maintenance strategy.

The condition based maintenance strategy achieved an average uptime of 68.04%, which means that only 31.96% of the time over the episode was spent on scheduled maintenance. This is an

improvement of 48.32% and 13.15% respectively over the run-to-failure corrective maintenance and scheduled maintenance strategies.

The implemented Deep Reinforcement Learning based predictive maintenance strategy achieved an average uptime of 72.65% over the entire episode, which means that only 27.35% of the time was spent on scheduled repairs. This represents a 55.77%, 25.68% and 14.42% improvement in uptime over the corrective, scheduled and condition based maintenance strategies respectively.

Figure 6.3b shows availability over a single episode. It is seen that significant periods of downtime appear in the scheduled maintenance strategy, where the availability remains horizontal for a longer period (around a normalised time of 60) compared to the condition based maintenance and DRL based approaches. This could indicate that the maintenance capacity was reached and maintenance was moved into the maintenance backlog, resulting in periods with less productivity.

6.2.2 Planned Maintenance Percentage

Planned maintenance percentage is an operational maintenance performance metric that considers how much maintenance is planned or unplanned. From Figure 6.4 the planned maintenance percentage (PMP) for the different strategies can be interpreted. Run-to-failure corrective maintenance achieved 0% PMP, as can be expected, seeing as the intention for this strategy is to let engines run to failure with no planned maintenance. Scheduled preventive maintenance aims to achieve 100% PMP, however on average 1.98 engines fail before planned maintenance can be performed resulting in a PMP of 95.8%. The condition based predictive maintenance strategy achieved a mean PMP of 92.97%.

The implemented Deep Reinforcement Learning based predictive maintenance strategy achieved a mean PMP of 97.3% with less than one engine failing on average before planned maintenance is scheduled. In comparison to the scheduled maintenance and condition based maintenance strategies, the Deep Reinforcement Learning based maintenance strategy resulted in a relatively small improvement in PMP of 1.5% and 4.33% respectively. The limitation to regular scheduled maintenance is that the actual health of assets are not considered to adapt the planned schedule, however the condition based maintenance and DRL strategies do consider the actual asset health. The benefit of considering the actual health only becomes apparent when also comparing asset utilisation.

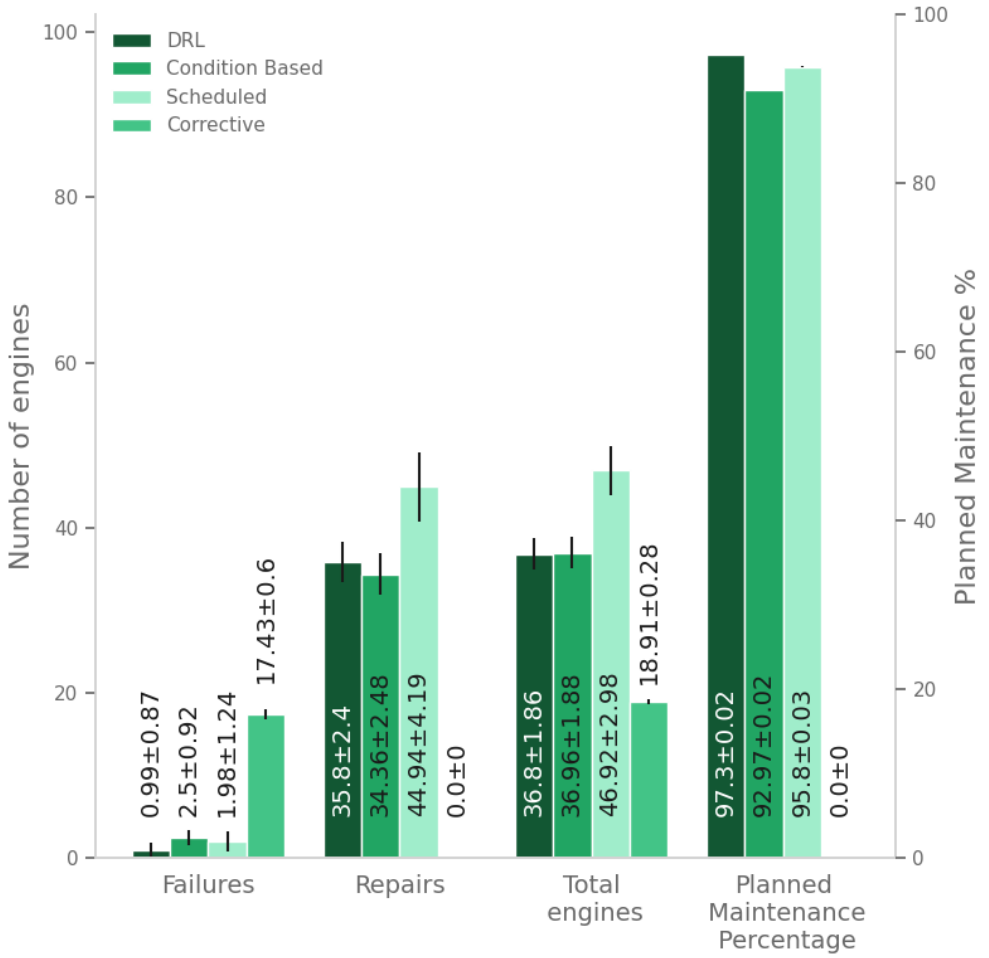


Figure 6.4: Planned Maintenance Percentage achieved between different maintenance strategies.

6.2.3 Utilisation

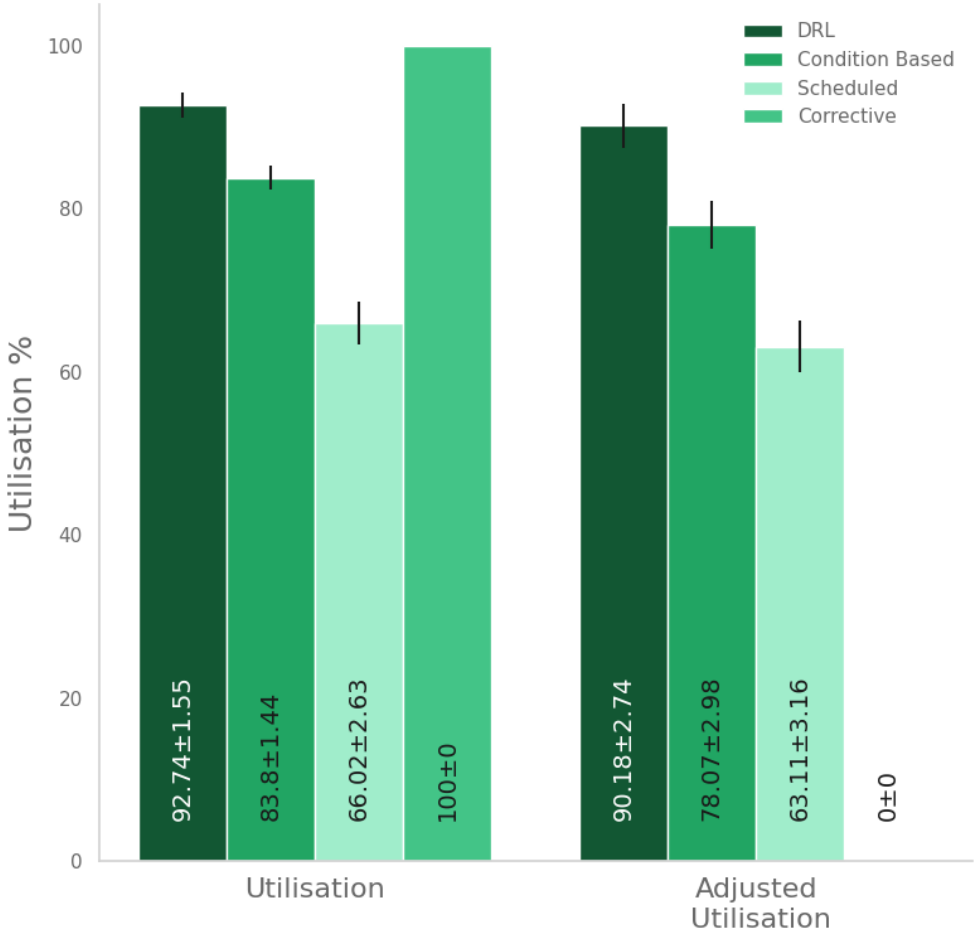


Figure 6.5: Utilisation and PMP Adjusted Utilisation achieved between different maintenance strategies.

Figure 6.5 shows utilisation and utilisation adjusted by PMP. Run-to-failure corrective maintenance achieves 100% utilisation, which is a trade-off with several other factors such as increased cost and time lost due to increased maintenance time required to repair severe failures. The PMP adjusted utilisation is 0% as this strategy ensures that all assets run to failure.

The scheduled preventive maintenance strategy achieves a mean utilisation of 66.02 ± 2.63% and a PMP adjusted utilisation of 63.11±3.26%. This is a direct result of the optimal preventive maintenance interval calculated in Section 5.3.2.

The condition based predictive maintenance strategy achieves a mean utilisation of 83.8 ± 1.44% and a PMP adjusted utilisation of 78.07 ± 2.98%. The improvement over scheduled maintenance is understandable, as the actual asset health is taken into account to determine when to stop engines for maintenance.

The implemented Deep Reinforcement Learning based predictive maintenance strategy achieved a mean utilisation and PMP adjusted utilisation of 92.74 ± 1.55% and 90.18 ± 2.74% respectively. When compared to the scheduled preventive maintenance and condition based predictive maintenance strategies, this represents a 27.07% and 12.11% respective increase in utilisation which ultimately contributes more to the asset life cycle cost. By considering the asset health the DRL strategy is capable of extracting more life out of each engine without causing serious failure.

The increased utilisation of the DRL approach compared to the condition based predictive maintenance strategy can be directly attributed to the predetermined stopping threshold selected to train the condition based predictive model. The DRL agents were able to learn a decision making policy that was not constrained by a predefined threshold, which suggests that this result is not significant, as the condition based predictive model can be fine-tuned to achieve the same, or better, utilisation. The intention is not to achieve state of the art performance, but rather to evaluate whether the DRL approach is capable of performing both prognostic decision making that results in good utilisation as well as perform maintenance scheduling which ensure optimal capacity and resource allocation ability.

6.2.4 Maintenance Capacity

Figure 6.6 shows the mean maintenance capacity over an episode for all maintenance strategies. With

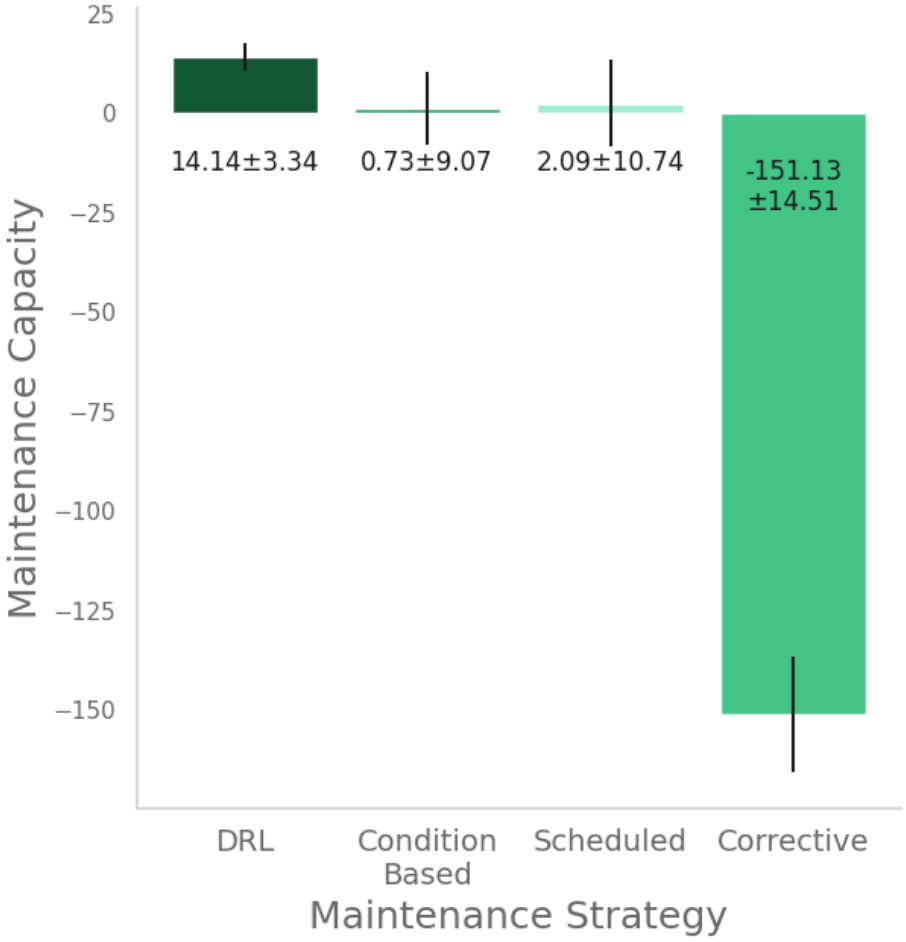


Figure 6.6: Average Maintenance Capacity achieved between different maintenance strategies.

a maintenance capacity set-point of 50 steps (using $C_P = 50$ and $C_U = 150$), the implemented Deep Reinforcement Learning based strategy is the only strategy that maintained a significant positive maintenance capacity over all experiments, with a mean and minimum capacity of 14.14 (28.28%) and 10.8 (21.6%) respectively.

The condition based predictive maintenance strategy achieved a mean and minimum capacity of 0.73 (1.46%) and −8.34 (−16.68%) respectively. The scheduled maintenance strategy achieved a mean and minimum capacity of 2.09 (4.18%) and −8.65 (−17.3%) respectively. The corrective maintenance strategy was completely overwhelmed and achieved a mean and minimum capacity of −151.13 (−319.56%) and −165.64 (−331.3%) respectively.

This result suggests that the Deep Reinforcement Learning based solution is able to learn how to schedule maintenance such that the maintenance capacity is considered and prioritised during decision making in order for capacity to remain positive. When considering both maintenance capacity and utilisation it is shown that the DRL approach is in fact capable of performing optimal decision making and resource allocation in a multi-component system with maintenance resource constraints, compared to condition based predictive maintenance, preventive scheduled maintenance and corrective maintenance.

6.2.5 Sensitivity Analysis

Table 6.1 shows the resulting availability, PMP, utilisation and capacity for different maintenance strategies for various corrective maintenance costs C_U.

Table 6.1: Sensitivity analysis of corrective maintenance cost.

	Capacity	C_P	C_U	DRL	Condition Based	Scheduled	Corrective
Availability	50	50	75	67.458	73.827	66.189	64.247
	50	50	100	71.068	72.188	60.565	52.916
	50	50	150	72.649	68.039	63.202	38.160
	50	50	200	70.617	64.540	61.072	29.207
PMP	50	50	75	91.601	94.112	34.629	0
	50	50	100	99.959	94.229	76.661	0
	50	50	150	97.302	93.236	95.168	0
	50	50	200	97.086	93.249	96.858	0
Utilisation	50	50	75	79.080	84.147	74.128	100.00
	50	50	100	66.948	84.059	72.914	100.00
	50	50	150	92.74	83.800	66.023	100.00
	50	50	200	83.880	83.943	62.536	100.00
Capacity	50	50	75	7.2231	15.072	-0.7550	-7.191
	50	50	100	13.698	11.808	-11.478	-47.826
	50	50	150	14.144	0.737	2.094	-151.13
	50	50	200	7.7043	-10.570	-1.274	-262.36

When considering availability there is not a significant performance difference between the DRL, condition based and scheduled maintenance strategies. It is shown that as the corrective maintenance cost increases, the availability of the corrective maintenance strategy decreases. This result is expected, seeing that as the corrective maintenance cost increases more time is spent in maintenance and less time is available for production.

From a planned maintenance percentage perspective, there is not a significant difference between the performance of the DRL and condition based maintenance strategies. The PMP performance of the scheduled maintenance strategy is however heavily affected by the corrective maintenance cost. This is directly due to the optimal replacement time which is a function of corrective maintenance cost. As the corrective maintenance cost increases, the optimal replacement time reduces, which means that engines are stopped earlier resulting in less failures and effectively a higher PMP.

Considering utilisation shows that the condition based predictive maintenance strategy has very little variation. This shows that the predetermined threshold directly affects the utilisation that can be

achieved. By changing the threshold, it should possible to increase or decrease the utilisation. This is seen in the utilisation achieved by the DRL approach. As the corrective maintenance cost increases, the DRL agents learn how to balance utilisation and maintenance capacity. The corrective maintenance strategy is intended to run to failure, which represents 100% utilisation, however in this case directly impacts maintenance cost and availability negatively.

Capacity is directly affected by increasing corrective maintenance cost. It is shown that as the corrective maintenance cost increases, the achieved capacity for the condition based predictive maintenance and corrective maintenance strategies decrease. This is understandable, seeing as the available capacity will reduce as each failure maintenance event now takes up more time. The scheduled maintenance strategy attempts to schedule more repairs than failure maintenance events which means that the capacity is only affected if a significant number of failures occur in a single episode. Overall the DRL approach finds a good balance between maintaining a high maintenance capacity and asset utilisation.

6.3 Discussion

By implementing curriculum-based learning and using reward shaping it was possible to create a simulation environment in which agents could be trained to optimise maintenance scheduling under constrained maintenance capacity. It was shown that by applying these techniques training was able to converge and that agents achieved the desired behaviour much faster than without using either curriculum learning or reward shaping. From this result, it is concluded that the implemented simulation environment is suitable for training multiple agents in a constrained maintenance resource setting.

From the asset and operational maintenance performance metrics it is shown that the implemented DRL based maintenance solution achieves better performance in both asset performance and operational performance. The DRL based solution considers real-time asset health and manages to extract the maximum life out of the engine without resulting in failure. The DRL based solution also considers current maintenance capacity and shows an improved ability to schedule maintenance, such that over an episode downtime is minimised while ensuring sufficient maintenance capacity remains available.

The implemented DRL based strategy outperforms the condition based predictive maintenance, scheduled maintenance and corrective maintenance strategies. This result suggests that Deep Reinforcement Learning based decision making for asset health management and resource allocation in a multi-component system with maintenance resource constraints is more effective than human based decision making in traditional maintenance strategies.

Chapter 7

Conclusion and Recommendations

Real-world industrial operational environments are stochastic, with multiple sub-systems or components and can have complex system dynamics resulting in uncertainty which makes decision making and resource allocation difficult. In a production environment with constrained maintenance resource capacity, it is important to consider the available maintenance capacity when scheduling maintenance. Without doing so, maintenance could be scheduled without available capacity, leading to unwanted downtime. With greater adoption of predictive maintenance practices, it has become easier to predict the optimal time to schedule maintenance before failure occurs, however these approaches do not necessarily consider system constraints which might affect scheduling.

This work considers the use of Deep Reinforcement Learning to determine the optimal maintenance scheduling policy for a fleet of assets with a limited maintenance capacity. The implemented technique did not just consider the decision making aspect of when to schedule maintenance before failure, but also considered maintenance capacity. The implemented approach was also compared to condition based predictive maintenance, constant-interval scheduled maintenance and run-to-failure corrective maintenance.

The simulated environment made use of the C-MAPSS PHM08 data set to simulate a fleet of turbofan engines. Engines could be stopped for maintenance before failure which resulted in less time required for maintenance. If engines were run until failure, maintenance performed would take three times longer. In a typical reinforcement learning environment, a direct simulation or model of the environment is available for agents to interact with. As explained in detail in Section 5.1, in this work a simulation environment was developed that could sample from the static C-MAPSS data set in order to simulate the degradation of engines rather than using the underlying high-fidelity simulation of the engines originally developed by NASA.

The significance of the developed simulation environment is that the environment could be used with any other static data set that is significantly large enough and representative of a real operational environment to train a new policy for that environment. The C-MAPSS data set is unique, when compared to several other publicly available data sets that are used for prognostic and health management research because it was generated using a high-fidelity simulation over the entire life-cycle of several engines, rather than measured from small scale experiments that are focused only on a small portion of equipment life. For this reason the size of the data set, as well as the range of parameters simulated is significant enough to represent the operation of a turbofan engine over its entire life-cycle. Even though a single data set was used to evaluate the implemented approach, it is noted that the goal was not to achieve state of the art performance on prognostic decision making or maintenance scheduling specifically, and was however to demonstrate that a DRL approach can be used for optimal decision

making in a multi-component system with resource constraints.

The Deep Reinforcement Learning based agent implementation made use of the Proximal Policy Optimisation algorithm in order to learn the optimal decision making policy in a model free on-policy based approach. A fully connected neural network was used to learn feature representations of the environment observations and further processed by a recurrent neural network using Long Short Term Memory cells to capture any time dependencies over consecutive observations. By using a model based approach, agents could use planning techniques to find the true optimal decision making policy.

Multiple agents were trained in the same environment in order to introduce a team dynamic where agents had to cooperate in order to determine the optimal decision making policy that maximised asset utilisation and maintenance capacity. Agents can take one of two actions, either stopping their individual engine for maintenance or continue to run the engine. Agents are rewarded for their individual ability to stop an engine for maintenance before failure occurs, as close to failure as possible. Agents also receive a team reward for optimising maintenance capacity over an episode. Curriculum learning as well as reward shaping was used to improve training convergence and allowed agents to learn the desired behaviour in the simulated environment. It was shown that without using these techniques, training did not converge or sub-optimal performance was achieved.

In reality, there are several resource constraints that need to be considered in a production environment, with maintenance capacity only representing a single type of constraint. This work builds towards a complete solution where DRL could be used to find the optimal decision making policy under several constraints. In this work, state augmentation was used used to incorporate the maintenance capacity into states which effectively limited certain actions at certain time steps. Other resource constraints can be incorporated in a similar way. Stochastic constraints such as risk-based or chance constraints are not considered in this work.

The novel contribution of this work was to develop a DRL framework for joint decision making in a multi-component environment with resource constraints, as well as using a single model that was trained end-to-end using equipment telemetry for both prognostic decision making and resource allocation for maintenance scheduling. Compared to previous work, as discussed in Section 1.4, research is primarily focused on using DRL for maintenance scheduling, without the use of equipment telemetry for prognostic decision making, or only focused on using equipment telemetry for prognostic decision making while not considering maintenance scheduling. Where previous work has however considered both prognostic decision making and maintenance scheduling a simplified system was considered. These simplified systems either evaluate maintaining a single piece of equipment, where no multi-component interactions are taken into account, or no resource constraints are considered.

One advantage of the implemented DRL framework is that it enables decision making in multi-component environments where there are resource constraints. This can be extremely helpful in operational environments where uncertainty makes decision making and resource allocation difficult. The second advantage is using a single model that is trained end-to-end using equipment telemetry and maintenance capacity for both prognostic decision making and resource allocation for maintenance scheduling. This approach reduces the complexity needed to implement a model into a real-world environment, where different data streams or models might need to be connected or coordinated, which represents its own challenges.

The implemented Deep Reinforcement Learning based approach was capable of finding a near-optimal decision making policy that considered individual asset health as well as overall maintenance capacity in a multi-component environment with maintenance resource constraints. The DRL framework outperformed traditional maintenance strategies across several maintenance performance metrics. It is concluded that Deep Reinforcement Learning based decision making for asset health management

and resource allocation is more effective than human based decision making. The proposed approach can be extended further by following the recommendations discussed below.

7.1 Recommendations

The following recommendations are made for future work:

1. Consider other resource constraints that could have an impact on maintenance scheduling including replacement part availability and part lead time. The geographical location of assets relative to the maintenance facility. Production targets could also be a constraint which needs to be balanced.

2. Stochastic constraints should also be considered in order to develop a framework that can deliver optimal decision making policies under uncertainty where constraints are not simply resource related. Lagrangian relaxation is successfully used by Andriotis & Papakonstantinou (2020) to incorporate stochastic constraints into a DRL framework for inspection and maintenance planning under incomplete information.

3. Consider equipment or data sets with several failure modes. In the simplified environment, there is only one type of failure. In reality, assets could have several failure modes. Each type of failure has a unique mean time between failure, unique mean time to repair and even unique replacement part requirements and lead times.

4. Use model based reinforcement learning with planning techniques, such as Monte Carlo Tree Search, to enable agents to plan ahead. Planning techniques could be greatly beneficial to solving scheduling problems as agents could consider all possible future scenarios before making a decision.

5. Combine unsupervised learning techniques with the implemented Deep Reinforcement Learning based approach to use health states that have been learnt from the underlying data and not from prior known failure trajectories. World Models (Ha & Schmidhuber 2018) could be a potential approach to (4) and (5).

6. Implement further agent communication through the use of a centralised critic. By sharing individual agent value function predictions with all other agents at each step, greater performance is expected as agents can make decisions with more explicit knowledge of the state of other agents. Sharing individual value function predictions should scale better than sharing all individual agent observations.

7. Further evaluation is needed for the construction and shaping of reward functions to explicitly include constraints or other metrics in the reward formulation. As the number of constraints applied to a system increase, the process of constructing and shaping reward functions will become more complex requiring a pragmatic and objective approach to succeed.

Bibliography

Al-Dulaimi, A., Zabihi, S., Asif, A. & Mohammadi, A. (2019), 'A multimodal and hybrid deep neural network model for remaining useful life estimation', *Computers in Industry* **108**, 186–196.

Andriotis, C. & Papakonstantinou, K. (2019), 'Managing engineering systems with large state and action spaces through deep reinforcement learning', *Reliability Engineering & System Safety* **191**, 106483.

Andriotis, C. & Papakonstantinou, K. (2020), 'Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints', *arXiv preprint arXiv:2007.01380* .

Atamuradov, V., Medjaher, K., Dersin, P., Lamoureux, B. & Zerhouni, N. (2017), 'Prognostics and health management for maintenance practitioners-review, implementation and tools evaluation', *International Journal of Prognostics and Health Management* **8**(060), 1–31.

Balaban, E., Alonso, J. & Goebel, K. (2012), An approach to prognostic decision making in the aerospace domain, *in* 'Annual Conference of the Prognostics and Health Management Society'.

Başar, T. & Olsder, G. J. (1998), *Dynamic noncooperative game theory*, SIAM.

Bellemare, M. G., Dabney, W. & Munos, R. (2017), A distributional perspective on reinforcement learning, *in* 'Proceedings of the 34th International Conference on Machine Learning-Volume 70', JMLR. org, pp. 449–458.

Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. (2013), 'The arcade learning environment: An evaluation platform for general agents', *Journal of Artificial Intelligence Research* **47**, 253–279.

Bellemare, M., Naddaf, Y. & Veness, J. (2013), 'The arcade learning environment: An evaluation platform for general agents', *Journal of Artificial Intelligence Research* **47**, 253279.

Bengio, Y., Louradour, J., Collobert, R. & Weston, J. (2009), Curriculum learning, *in* 'Proceedings of the 26th annual international conference on machine learning', pp. 41–48.

Blanchard, B. S., Verma, D. C. & Peterson, E. L. (1995), *Maintainability: a key to effective serviceability and maintenance management*, Vol. 13, John Wiley & Sons.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012), 'A survey of monte carlo tree search methods', *IEEE Transactions on Computational Intelligence and AI in Games* **4**(1), 143.

Busoniu, L., Babuska, R. & De Schutter, B. (2008), 'A comprehensive survey of multiagent reinforcement learning', *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **38**(2), 156–172.

Buoniu, L., Babuska, R. & Schutter, B. D. (2010), Multi-agent reinforcement learning: An overview,

in 'Innovations in multi-agent systems and applications-1', Springer, p. 183221.

Chebel-Morello, B., Nicod, J. & Varnier, C. (2018), *From Prognostics and Health Systems Management to Predictive Maintenance 2 - Knowledge, Traceability and Decision*, 7th ed edn, ISTE Ltd, London.

Custeau, K. (2017), *Asset Performance Management 4.0 and Beyond with Risk-Based Maintenance*, Schneider Electric.

De Carlo, F. & Arleo, M. A. (2017), 'Imperfect maintenance models, from theory to practice', *System Reliability* p. 335.

Deep Learning (2018), [online] Nvidia Developer.
URL: *https://developer.nvidia.com/deep-learning*

Delmas, A., Sallak, M., Schön, W. & Zhao, L. (2018), 'Remaining useful life estimation methods for predictive maintenance models: defining intervals and strategies for incomplete data', *in Industrial Maintenance and Reliability Manchester, UK 12-15 June, 2018* p. 48.

Deng, L. & Yu, D. (2014), 'Deep learning: Methods and applications'.

Diddigi, R. B., Reddy, D. & Bhatnagar, S. (2017), 'Multi-agent q-learning for minimizing demand-supply power deficit in microgrids', *arXiv preprint arXiv:1708.07732* .

Duch, W. & Mandziuk, J. (2007), *Challenges for Computational Intelligence*, Springer, Berlin.

Ellefsen, A. L., Ushakov, S., Æsøy, V. & Zhang, H. (2019), 'Validation of data-driven labeling approaches using a novel deep network structure for remaining useful life predictions', *IEEE Access* **7**, 71563–71575.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E. & Levine, S. (2018), 'Model-based value estimation for efficient model-free reinforcement learning', *arXiv preprint arXiv:1803.00101* .

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N. & Whiteson, S. (2017), 'Counterfactual multi-agent policy gradients.'.

Gouriveau, R., Medjaher, K. & Zerhouni, N. (2016), *From prognostics and health systems management to predictive maintenance 1: Monitoring and prognostics*, John Wiley & Sons.

Graesser, L. & Keng, W. L. (2019), *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*, Addison-Wesley Professional.

Ha, D. & Schmidhuber, J. (2018), 'World models', *arXiv preprint arXiv:1803.10122* .

Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. (2018), 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor', *arXiv preprint arXiv:1801.01290* .

Haddad, G., Sandborn, P. & Pecht, M. (2011), Using real options to manage condition-based maintenance enabled by phm, *in* 'IEEE Conference on Prognostics and Health Management'.

Hausknecht, M. & Stone, P. (2015*a*), 'Deep recurrent q-learning for partially observable mdps', *arXiv preprint arXiv:1507.06527* .

Hausknecht, M. & Stone, P. (2015*b*), 'Deep reinforcement learning in parameterized action space', *arXiv preprint arXiv:1511.04143* .

He, X., Zhao, K. & Chu, X. (2019), 'Automl: A survey of the state-of-the-art', *arXiv preprint*

arXiv:1908.00709 .

Hernandez-Leal, P., Kartal, B. & Taylor, M. E. (2019), 'A survey and critique of multiagent deep reinforcement learning', *Autonomous Agents and Multi-Agent Systems* **33**(6), 750–797.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Hsu, C.-S. & Jiang, J.-R. (2018), Remaining useful life estimation using long short-term memory deep learning, *in* '2018 IEEE International Conference on Applied System Invention (ICASI)', IEEE, pp. 58–61.

Huang, C.-G., Huang, H.-Z. & Li, Y.-F. (2019), 'A bidirectional lstm prognostics method under multiple operational conditions', *IEEE Transactions on Industrial Electronics* **66**(11), 8792–8802.

Huang, J., Chang, Q. & Arinez, J. (2020), 'Deep reinforcement learning based preventive maintenance policy for serial production lines', *Expert Systems with Applications* **160**, 113701.

Iqbal, S. & Sha, F. (2018), 'Actor-attention-critic for multi-agent reinforcement learning'.

Iyer, N., Goebel, K. & Bonissone, P. (2006), Framework for post-prognostic decision support, *in* '2006 IEEE Aerospace Conference', IEEE, pp. 10–pp.

JA1011, S. (1999), 'Evaluation criteria for reliability-centered maintenance (rcm) processes', *Society for Automotive Engineers* .

Jardine, A. K., Lin, D. & Banjevic, D. (2006), 'A review on machinery diagnostics and prognostics implementing condition-based maintenance', *Mechanical systems and signal processing* **20**(7), 1483–1510.

Jardine, A. K. & Tsang, A. H. (2013), *Maintenance, replacement, and reliability: theory and applications*, CRC press.

Jayasinghe, L., Samarasinghe, T., Yuenv, C., Low, J. C. N. & Ge, S. S. (2019), Temporal convolutional memory networks for remaining useful life estimation of industrial machinery, *in* '2019 IEEE International Conference on Industrial Technology (ICIT)', IEEE, pp. 915–920.

Jin, H., Song, Q. & Hu, X. (2019), Auto-keras: An efficient neural architecture search system, *in* 'Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining', ACM, pp. 1946–1956.

Khan, S. & Yairi, T. (2018), 'A review on the application of deep learning in system health management', *Mechanical Systems and Signal Processing* **107**, 241265.

Kopuru, M. S. K., Rahimi, S. & Baghaei, K. (2019), Recent approaches in prognostics: State of the art, *in* 'Proceedings on the International Conference on Artificial Intelligence (ICAI)', The Steering Committee of The World Congress in Computer Science, Computer , pp. 358–365.

Kuhnle, A., Jakubik, J. & Lanza, G. (2019), 'Reinforcement learning for opportunistic maintenance optimization', *Production Engineering* **13**(1), 33–41.

Laud, A. D. (2004), Theory and application of reward shaping in reinforcement learning, Technical report.

LeCun, Y., Bengio, Y. et al. (1995), 'Convolutional networks for images, speech, and time series', *The handbook of brain theory and neural networks* **3361**(10), 1995.

Leviathan, Y. (2018), *Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone*, Blog] Google AI Blog.
URL: *https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html*

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J. & Jurafsky, D. (2016), 'Deep reinforcement learning for dialogue generation', *arXiv preprint arXiv:1606.01541* .

Li, X., Ding, Q. & Sun, J. (2018*a*), 'Remaining useful life estimation in prognostics using deep convolution neural networks', *Reliability Engineering and System Safety* **172**, 111.

Li, X., Ding, Q. & Sun, J.-Q. (2018*b*), 'Remaining useful life estimation in prognostics using deep convolution neural networks', *Reliability Engineering & System Safety* **172**, 1–11.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M. & Stoica, I. (2018), Rllib: Abstractions for distributed reinforcement learning, *in* 'International Conference on Machine Learning', pp. 3053–3062.

Lilliefors, H. W. (1967), 'On the kolmogorov-smirnov test for normality with mean and variance unknown', *Journal of the American Statistical Association* **62**(318), 399–402.

Lipton, Z. C., Berkowitz, J. & Elkan, C. (2015), 'A critical review of recurrent neural networks for sequence learning', *arXiv preprint arXiv:1506.00019* .

Liu, Y., Chen, Y. & Jiang, T. (2020), 'Dynamic selective maintenance optimization for multi-state systems over a finite horizon: A deep reinforcement learning approach', *European Journal of Operational Research* **283**(1), 166–181.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P. & Mordatch, I. (2017), Multi-agent actor-critic for mixed cooperative-competitive environments, *in* 'Advances in Neural Information Processing Systems', p. 63826393.

Makis, V. & Jardine, A. K. (1992), 'Optimal replacement policy for a general model with imperfect repair', *Journal of the Operational Research Society* **43**(2), 111–120.

Markov, A. A. (1954), 'The theory of algorithms', *Trudy Matematicheskogo Instituta Imeni VA Steklova* **42**, 3–375.

Miao, H., Li, B., Sun, C. & Liu, J. (2019), 'Joint learning of degradation assessment and rul prediction for aeroengines via dual-task deep lstm networks', *IEEE Transactions on Industrial Informatics* **15**(9), 5023–5032.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. & Kavukcuoglu, K. (2016), Asynchronous methods for deep reinforcement learning, *in* 'International conference on machine learning', pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**, 529533.
URL: *http://dx.doi.org/10.1038/nature14236*

Moss, M. A. et al. (1985), *Designing for minimal maintenance expense: the practical application of reliability and maintainability*, Vol. 1, CRC Press.

Nagabandi, A., Kahn, G., Fearing, R. S. & Levine, S. (2018), Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, *in* '2018 IEEE International Con-

ference on Robotics and Automation (ICRA)', IEEE, pp. 7559–7566.

Nam-Ho, K., Dawn, A. & Joo-Ho, C. (2017), *Prognostics and Health Management of Engineering Systems An Introduction*, Springer International Publishing, Switzerland.

Narvekar, S., Sinapov, J. & Stone, P. (2017), Autonomous task sequencing for customized curriculum design in reinforcement learning., *in* 'IJCAI', pp. 2536–2542.

Ng, A. Y., Harada, D. & Russell, S. (1999), Policy invariance under reward transformations: Theory and application to reward shaping, *in* 'ICML', Vol. 99, pp. 278–287.

Nguyen, K. T. & Medjaher, K. (2019), 'A new dynamic predictive maintenance framework using deep learning for failure prognostics', *Reliability Engineering & System Safety* **188**, 251–262.

OpenAI (2018), OpenAI Five. [online].
URL: *https://blog.openai.com/openai-five/*

OpenAI, C. B., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C. et al. (2019), 'Dota 2 with large scale deep reinforcement learning', *arXiv preprint arXiv:1912.06680* .

ODonovan, P., Leahy, K., Bruton, K. & OSullivan, D. (2015), 'Big data in manufacturing: a systematic mapping study', *Journal of Big Data* **2**(20).

Parker, K. I. & Guo, T.-H. (2003), 'Development of a turbofan engine simulation in a graphical simulation environment'.

Pecht, M. (2008), *Prognostics and Health Management of Electronics*, John Wiley and Sons, Inc, Hoboken, New Jersey.

Powell, W. (2011), *Approximate Dynamic Programming: Solving the curses of dimensionality*, 2 edn, John Wiley and Sons.

Preuveneers, D. & Ilie-Zudor, E. (2017), 'The intelligent industry of the future: A survey on emerging trends, research challenges and opportunities in industry 4.0', *Journal of Ambient Intelligence and Smart Environments* **9**(3), 287298.

PwC (2016), *Industry 4.0: Building the digital enterprise South Africa highlights. 2016 Global Industry 4.0 Survey*, online.
URL: *https://www.pwc.co.za/en/assets/pdf/south-africa-industry-4.0-report.pdf*

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y. et al. (2017), Imagination-augmented agents for deep reinforcement learning, *in* 'Advances in neural information processing systems', pp. 5690–5701.

Rao, S. S. (1992), *Reliability-based design*, McGraw-Hill Companies.

Rausand, M., Barros, A. & Hoyland, A. (2020), *System reliability theory: models, statistical methods, and applications*, John Wiley & Sons.

Roychoudhury, I., Reveley, M., Phojanamongkolkij, N. & Leone, K. (2017), *Assessment of the State-of-the-Art of System-Wide Safety and Assurance Technologies*, online.
URL: *https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20170011193.pdf*

Saxena, A., Goebel, K., Simon, D. & Eklund, N. (2008), Damage propagation modeling for aircraft engine run-to-failure simulation, *in* '2008 international conference on prognostics and health management', IEEE, pp. 1–9.

Schulman, J. (2016), Optimizing expectations: From deep reinforcement learning to stochastic computation graphs, PhD thesis, UC Berkeley.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. (2015), Trust region policy optimization, *in* 'International conference on machine learning', pp. 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017), 'Proximal policy optimization algorithms', *arXiv preprint arXiv:1707.06347* .

Schuster, M. & Paliwal, K. K. (1997), 'Bidirectional recurrent neural networks', *IEEE transactions on Signal Processing* **45**(11), 2673–2681.

Sheut, C. & Krajewski, L. (1994), 'A decision model for corrective maintenance management', *The International Journal of Production Research* **32**(6), 1365–1382.

Sifonte, J. R. & Reyes-Picknell, J. V. (2017), *Reliability Centered Maintenance–Reengineered: Practical Optimization of the RCM Process with RCM-R®*, CRC Press.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016), 'Mastering the game of go with deep neural networks and tree search', *Nature* **529**(7587), 484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. & Riedmiller, M. (2014), Deterministic policy gradient algorithms.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. & Hassabis, D. (2017), 'Mastering the game of go without human knowledge', *Nature* **550**(7676), 354359.

Skordilis, E. & Moghaddass, R. (2020), 'A deep reinforcement learning approach for real-time sensor-driven decision making and predictive analytics', *Computers & Industrial Engineering* **147**, 106600.

Stenström, C., Norrbin, P., Parida, A. & Kumar, U. (2016), 'Preventive and corrective maintenance–cost comparison and cost–benefit analysis', *Structure and Infrastructure Engineering* **12**(5), 603–617.

Stratonovich, R. L. (1965), Conditional markov processes, *in* 'Non-linear transformations of stochastic processes', Elsevier, pp. 427–453.

Sutton, R. & Barto, A. (1998), *Reinforcement Learning: An Introduction*, MIT Press, Cambridge.

Sutton, R. S., McAllester, D. A., Singh, S. P. & Mansour, Y. (2000), Policy gradient methods for reinforcement learning with function approximation, *in* 'Advances in neural information processing systems', pp. 1057–1063.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J. & Vicente, R. (2017), 'Multiagent cooperation and competition with deep reinforcement learning', *PloS one* **12**(4), e0172395.

Tesla (2018), *Autopilot*, [online] Tesla.
URL: *https://www.tesla.com/AUTOPILOT*

The story of AlphaGo so far (2018), DeepMind Technologies Limited.
URL: *https://deepmind.com/research/alphago/*

Tsitsiklis, J. N. & Van Roy, B. (1997), Analysis of temporal-diffference learning with function approx-

imation, *in* 'Advances in neural information processing systems', pp. 1075–1081.

Tuyls, K. & Weiss, G. (2012), 'Multiagent learning: Basics, challenges, and prospects', *Ai Magazine* **33**(3), 41–41.

TV, V., Malhotra, P., Vig, L., Shroff, G. et al. (2019), 'Data-driven prognostics with predictive uncertainty estimation using ensemble of deep ordinal regression models', *arXiv preprint arXiv:1903.09795* .

Wang, J., Wen, G., Yang, S. & Liu, Y. (2018), Remaining useful life estimation in prognostics using deep bidirectional lstm neural network, *in* '2018 Prognostics and System Health Management Conference (PHM-Chongqing)', IEEE, pp. 1037–1042.

Wei, S., Bao, Y. & Li, H. (2020), 'Optimal policy for structure maintenance: A deep reinforcement learning framework', *Structural Safety* **83**, 101906.

Williams, J. H., Davies, A. & Drake, P. R. (1994), *Condition-based maintenance and machine diagnostics*, Springer Science & Business Media.

Williams, R. J. (1992), 'Simple statistical gradient-following algorithms for connectionist reinforcement learning', *Machine learning* **8**(3-4), 229–256.

Wu, Y., Yuan, M., Dong, S., Lin, L. & Liu, Y. (2018), 'Remaining useful life estimation of engineered systems using vanilla lstm neural networks', *Neurocomputing* **275**, 167–179.

Zhao, T., Hachiya, H., Niu, G. & Sugiyama, M. (2012), 'Analysis and improvement of policy gradient estimation', *Neural Networks* **26**, 118–129.

Zhu, P., Li, X., Poupart, P. & Miao, G. (2018), 'On improving deep reinforcement learning for pomdps', *arXiv preprint arXiv:1804.06309* .

Appendix A

Deep Learning Model Architectures and Parameters

A.1 Deep Reinforcement Learning Agent Architecture and Parameters

Table A.1 details the model architecture used for the DRL agents. The model built up an internal input window over 10 time steps consisting of the 24 input sensor values as well as the current maintenance capacity at each time step. The effective size of a single input sample was 25 dimensions. The 25 dimensions were encoded through the dense feed forward network to 16 units, which was then combined with the previous action and reward, resulting in an input for the LSTM of 18 units. The LSTM cell had 64 units and predicted the action and value of the next time step. The *none* size in the first dimension of the output shape represents the variable batch size of the input.

Table A.1: Deep Reinforcement Learning Agent Model Architecture.

Layer	**Activation function**	**Output Shape**
dense input layer	none	(*none*, 1, 25)
dense hidden layer 1	tanh	(*none*, 1, 64)
dense hidden layer 2	tanh	(*none*, 1, 32)
dense hidden layer 3	tanh	(*none*, 1, 16)
previous action and reward	linear	(*none*, 1, 2)
LSTM input later	none	(*none*, 1, 18)
LSTM	tanh	(*none*, 10, 64)
logits (action)	linear	(*none*, 1, 2)
value	sigmoid	(*none*, 1, 1)

A.2 Condition Based Predictive Maintenance Model Architecture and Parameters

Table A.2 shows the model architecture and parameters obtained after neural architecture and parameter search was performed using AutoKeras. A Recurrent Neural Network model was optimised using an input window of 10 time steps over the 24 input sensor values. The *none* size in the first dimension of the output shape represents the variable batch size of the input. The model was trained with a batch size of 512. Inference was done on a single sample, with an effective batch size of 1. AutoKeras determined that a single bi-directional LSTM cell with 48 units using a Sigmoid activation function

was the optimal architecture.

Table A.2: Condition Based Predictive Maintenance AutoKeras Model Architecture.

Layer	**Activation Function**	**Output Shape**
input layer	none	(*none*, 10, 24)
bi-directional LSTM	sigmoid	(*none*, 48)
dense	linear	(*none*, 1)
classification head	sigmoid	(*none*, 1)