YAŞAR UNIVERSITY

GRADUATE SCHOOL

MASTER’S THESIS

A DEEP REINFORCEMENT LEARNING MODELLING APPROACH FOR *s*, *S* INVENTORY CONTROL PROBLEM

GÜRAY KILINÇ

THESIS ADVISOR: ASSOC. PROF. DR. BANU YETKIN EKREN

INDUSTRIAL ENGINEERING

PRESENTATION DATE: 24.08.2020

BORNOVA / İZMİR
AUGUST 2020

ABSTRACT

A DEEP REINFORCEMENT LEARNING MODELLING APPROACH FOR (*s*, *S*) INVENTORY CONTROL PROBLEM

Kılınç, Güray
MSc, Industrial Engineering
Advisor: Assist. Prof. Dr. Banu Yetkin Ekren
August 2020

In this thesis, deep reinforcement learning (DRL) is applied to an inventory control optimization problem in a single-echelon supply chain network. In the DRL approach, intelligent agents determining how much to order in each time step to maximize the total profit of the network is determined. Also, a static model is developed in which the optimal reorder points (*s*) and the optimal order-up-to levels (*S*) are calculated by a mathematical model. Later, an agent with deep *Q*-Networks (DQN) by a DRL algorithm is trained to decide those levels in a learning environment. The two approaches are compared under different scenarios and the results show that the DQN agent outperforms the static (*s*, *S*) policy under a stochastic environment.

Key Words: reinforcement learning, single-echelon inventory, deep reinforcement learning, inventory optimization, machine learning

ÖZ

ENVANTER YÖNETİMİ İÇİN DERİN TAKVİYELİ ÖĞRENME YAKLAŞIMI

Güray, KILINÇ
Yüksek Lisans, Endüstri Mühendisliği
Danışman: Doç. Dr. Banu Yetkin EKREN
Ağustos 2020

Bu tezde, tek kademeli tedarik zinciri ağında envanter optimizasyonunu yöneten derin takviyeli öğrenme (DRL) ajanlarının performansı incelenmiştir. DRL'de, akıllı ajan, bu ağın toplam kârını en üst düzeye çıkarmak için her bir zaman adımında ne kadar sipariş verileceğini belirlenir. Ayrıca, yeniden sipariş noktası (*s*) ve en yüksek envanter düzeyi (*S*) bir algoritma yardımı ile bulunarak statik bir model geliştirilmiştir. Ardından, bir öğrenme ortamında bu seviyelere karar vermek için bir DRL algoritması olan derin Q-Networks (DQN) kullanan bir ajan eğitilmiştir. İki yaklaşım, farklı senaryolar altında karşılaştırılmış ve sonuçlar DRL yaklaşımının statik (*s, S*) politikasından daha iyi performans gösterdiği gözlemlenmiştir.

Anahtar Kelimeler: takviyeli öğrenme, tek kademeli, derin takviyeli öğrenme, envanter optimizasyonu, makine öğrenmesi

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Banu Y. EKREN for his guidance and patience during this study.

I would like to express my enduring love to my parents, who are always supportive, loving and caring to me in every possible way in my life.

Güray KILINÇ

İzmir, 2020

TEXT OF OATH

I declare and honestly confirm that my study, titled “A Deep Reinforcement Learning Approach for Inventory Management” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Güray KILINÇ

Signature

…………………………………..

August 23, 2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS:

RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DQN	Deep Q Networks
MDP	Markov Decision Process
DNN	Deep Neural Network

SYMBOLS:

I_i	Inventory level of the Actor i
O_i	Ordering Size of the Actor i
D_t	Demand
P_i	Profit of the Actor i
r	Reward
S	Order up to Quantity
s	Reorder Point
μ	Demand Mean
a	Action
h	Holding Cost
P	Unit Price
p	Penalty Cost
γ	Discount rate
Co	Fixed Ordering Cost
μ	The Average of Poisson Distributed One-Period Demand

CHAPTER 1
INTRODUCTION

A supply chain involves all activities related to the production and information flow from raw materials to the final products (Tan & Kannan, 1998). Supply chain management involves efficient management starting from the design of product or service until its final delivery to customers (Houlihan, 1985). The supply chain may result in waste, cost loss, inefficiency, and consumer dissatisfaction due to a lack of effective inventory management (Jones & Riley, 1985). Therefore, inventory optimization is an important problem in the supply chain. The key decision here is how much to order from the factory and how to keep the stock level in warehouses. While small businesses do not use a management system or just maintain manually, big companies use non-dynamic policies such as (*s*, *S*) inventory policies (Gavirneni, 2005). However, after the Covid-19, many large to small size companies experienced profit loss due to sudden demand changes, due to not able to predict the number of demand arrivals.

This thesis proposes a deep reinforcement learning (Mnih et al., 2013) approach for inventory optimization where the order amounts are adjusted based on real-time demand data. The deep reinforcement learning (DRL) applications such as AlphaGo (D. Silver et al., 2016), DQN Atari (Mnih et al., 2013), and OpenAi (Brockman et al., 2016) break new ground in artificial intelligence. Inspired by these successful DRL implementations, it is decided to use DRL for an inventory optimization problem in this thesis. The aim of the thesis is to design intelligent agent-based modelling determining what amount and when to order by tracing the inventory problem changing environment (i.e., demand). The system is simulated by the Python programming language (Van Rossum & Drake, 2009) and utilize the machine learning libraries in Python to facilitate the coding.

1.1 Problem Definition and Assumptions

In this thesis, a single-echelon inventory problem consisting of a single warehouse is modeled and a single product type is ordered from a higher echelon which is the factory as shown in Figure 1.1. It is assumed that the factory has no capacity constraint and can provide an infinite number of products. The delivery time from the factory to the warehouse is assumed to be zero. The objective function is determined to be the maximization of profit in which is considered fixed ordering cost and holding cost. If demand arriving at the factory is not satisfied, the agent will bear the penalty cost and profit will fall. But on the other hand, if the agent puts too much product in inventory, profit will fall again. For this dilemma, the agent aims to reduce the holding inventory cost as low as possible while meeting the demands. Demand is modeled by Poisson distribution with mean *(λ)* per time-period for a more realistic environment in the experiment.

In the literature, basic reinforcement learning algorithms used in inventory optimization problems and heuristic policies are compared. These are given below, respectively.

- Smart algorithm and (*s*, *S*) policy (Giannoccaro & Pontrandolfo, 2002)
- *Q*-learning, Sarsa algorithm and (*r*, *Q*) policy (Kemmer & Read, 2018)
- DQN and (*r*, *Q*) policy (Oroojlooy, 2019)
- Deep deterministic policy gradient (DDPG) and (*r*, *Q*) policy (Hutse, Victor, Verleysen, A., & Wyffels, 2019)

In this thesis, the DQN algorithm is used to train the agent. Later, the trained model is compared with a static (*s*, *S*) policy. In addition, different from the other papers, By dynamically changing the value of λ, models are tested and compared under highly variable demand conditions.

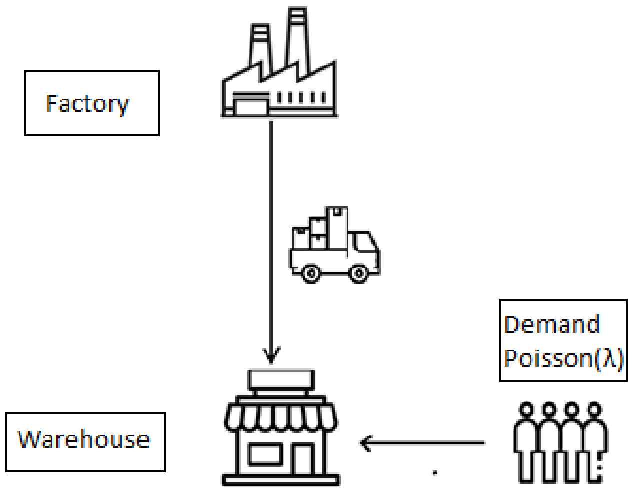


Figure 1.1. The Structure of the Supply Chain

As mentioned the purpose of the agent is to maximize the total profit throughout the supply chain. Different demand scenarios (as defined in section 3.3) are applied to test the robustness of the model. The DRL results are compared to a static inventory model where (*s*, *S*) levels are pre-defined based on the calculation of an analytical model and fixed throughout the run time which is 5 years.

The remainder of the thesis is structured as follows:

- Chapter 2 introduces RL, Deep Neural Networks, DQN by explaining the mathematical equations associated with them. The chapter discusses off-policy, model-free, value-based methods. Different components of the DNN are presented and a regularization technique is discussed. Also, inventory optimization is defined.
- Chapter 3 presents how the DRL algorithm is trained and it is compared with the static (*s*, *S*) policy in different environments.
- Chapter 4 provides a summary and conclusion of the thesis and also providing guidelines for future work.

CHAPTER 2

LITERATURE REVIEW

In this chapter is explained reinforcement learning, studies on deep neural networks as well as basic concepts. Also, methods such as regularization required for the DNN are discussed. The working principle of deep *Q* learning and (*s*, *S)* are also given in this section.

2.1 Reinforcement Learning

This section begins by introducing the main concepts of reinforcement learning. Reinforcement Learning (RL) is one of the branches of machine learning where an agent learns a specific state depending on the actions (Sutton & Barto, 2018). Other branches of machine learning such as supervised learning comparing to labeled inputs data then mapping outputs data. Unsupervised learning, on the other hand, uses clustering to interpret data based on unlabeled inputs data.

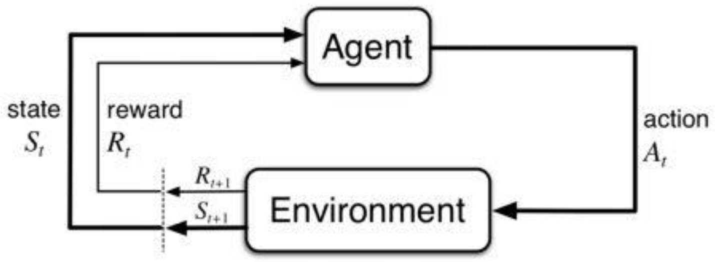


Figure 2.1. Interaction of agent and environment in MDP (Sutton & Barto, 2018)

RL uses the concept of Markov Decision Process (MDP) and it consists of agent, environment, action, state, and reward (as shown in Figure 2.1). At every timestep, the agent receives state and reward signals as feedback from the environment depending on each action. The agent is expected to predict possible reward value from current action and define an optimal approach to maximize the value of the overall reward (Sutton & Barto, 2018). An (MDP) is defined by a set of states, s ∈ S, where S denotes the state space, a set of actions, a ∈ A, where A denotes the action space,

- The transition probability function is the probability of going from state s to state s′ when the agents perform action given by the a shown by (2.1) (Sutton & Barto, 2018). T: S × A ∈ [0, 1] :

$$p(s^j \mid s,a) = Pr(s_{t+1} = s^j \mid s_t = s, a_t = a) \tag{2.1}$$

- The reward function represents the reward given by the environment when agent takes the action (a) that results in a transition the state s to state s′, shown by (2.2). (Sutton & Barto, 2018), $R: S \times S \times A \rightarrow \mathbb{R}$

$$R(s,a,s^j) = \mathbb{E}(r_t \mid s_t = s, a_t = a, s_{t+1} = s^j) \tag{2.2}$$

- The policy is mapping from states to actions and could be deterministic $\pi(s)$ or stochastic $\pi(a \mid s)$.

2.1.1 Value Function

Value function evaluates expected sum of reward policy when agent is at state (s) (Kaelbling et al., 1996) and represented ($V\pi(s)$), which defined as:

$$V\pi(s) = R(s,\ \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V\pi(s') \tag{2.3}$$

The optimal value function selects the maximum value at state *s* between all $V\pi(s)$ and the optimum policy can be defined as:

$$\pi^*(s) = \left(arg\,\underset{a}{max}\ R(s,\ \pi(s)) + \ \gamma\ X\ \sum T(s,\ \pi(s),\ s'\ )V_\pi^*(s') \right) \tag{2.4}$$

2.1.2 *Q* – Learning

In 1992, Watkins introduced the *Q*-learning algorithm (Watkins & Dayan, 1992) and it became the most popular RL algorithm due to its ease of implementation. *Q*-learning uses the policy π for any *s* ∈ *S* and *a* = *π*(*s*) also known as Bellman equation:

$$Q(s,\ a)\ =\ E\ [r_t\ +\ \gamma r_{t+1}\ +\ \gamma^2 r_{t+2}\ +\ \cdots\ |\ \mathrm{s}_t\ =\ s,\ a_t\ =\ a;\ \pi] \tag{2.5}$$

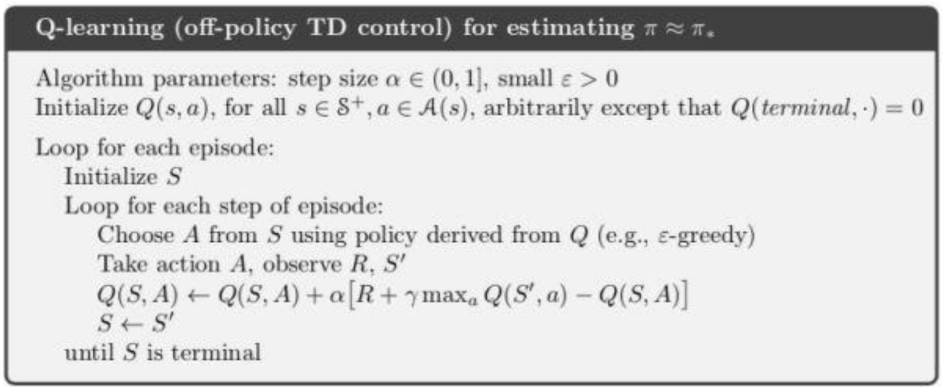
In the equation below, *s* means the current state, a represents the action the agent takes from the current state. s'represents the resulting state after the action taken by agents. a'represents the action from the next state. *r* is the reward value, *γ* is the discount factor and finally *α* is the learning rate. The discount factor *γ* ∈ (0, 1] determines how important you are going to give to future rewards. As the value increases, the importance given to the future reward increases (Sutton & Barto, 2018).

Then *q*-value is updated after observation based on the iterative formula:

$$Q(s_t, a_t) = (1\ -\ a_t)Q(s_t, a_t)\ +\ a_t\left(r_{t+1} + \gamma \max_a Q(s_{t+1},\ a)\right)\ ,\ \forall t\ =\ 1,\ 2,\ \ldots, \tag{2.6}$$

The *Q*-Learning algorithm is composed of three main methods which are off-policy, model-free, and value-based (Watkins & Dayan, 1992). The model-free method is divided into two main approaches. One of them is the value function based method which the agent tries to maximize a value function and understand the optimal policy.

Another method, policy search, agent determines optimal policy by searching among policy parameters (Sutton & Barto, 2018).



```
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until $S$ is terminal
```

Figure 2.2. The Q-learning Algorithm (Sutton & Barto, 2018)

Policy-off means that the agent takes the best action (a′) using the *ε*- greedy policy in state s′ and tries to maximize the cumulative return (Watkins & Dayan, 1992). This is the biggest difference between policy-on and policy-off. The values of the reward functions are written on the matrix with dimensions number of states by the number of actions at the initial zero value (Sutton & Barto, 2018) . The *q*-value of each state-action is saved in *q*-table. The agent selects action based on the *q*-tables with a probability of 1-*ε* with the highest cumulative action value i.e. $a_{t+1} = \arg\max_a Q(s_{t+1}, a)$ or explores the environment by taking random actions. Namely, with a probability of *ε* it takes random action. These methods are called exploitation and exploration (Osband et al., 2016).

2.2 Deep Neural Networks

Deep Neural Networks (DNN) is one of the main factors in the development of artificial intelligence (AI) applications in recent years, which are autonomous car (Bojarski et al., 2016) and robots (Schulman et al., 2016) (Mnih et al., 2015) large scale image recognition (Simonyan & Zisserman, 2015), speaker and language recognition (Richardson et al., 2015) and recommender systems (Zhang et al., 2019).

Neural networks work is inspired by a biological neural network which is a complex interconnected neuron system and information can be extracted from the given inputs and produce a corresponding output (Haykin, 1994). Likewise, DNN consists of a series of interconnect layers where neurons receive a pattern and determine valuable output. That's the way it recognizes and predicts patterns such as images, sounds, texts, or time series. Also, these patterns must be represented with the numeric value (Goodfellow et al., 2016).

In this chapter, artificial neurons and activation functions are mentioned to give in-depth knowledge of the working principle of artificial neural networks. Besides, The normalization methods, which make to learn artificial neural networks faster and simplifies the creation of deeper networks (Goodfellow et al., 2016), are also introduced in this section.

2.2.1 Artificial Neuron

Neurons are the simplest component of DNN and consists of bias, weights, and activation functions. Figure 2.3. shows, a neuron takes inputs $x_{i,}$ determines the weighted sum of the inputs, and is represented by w_i. Then, this variable added with the value called bias denoted by b, which is previously determined. Subsequently, the neuron gets an output y, after passing through the activation function f.

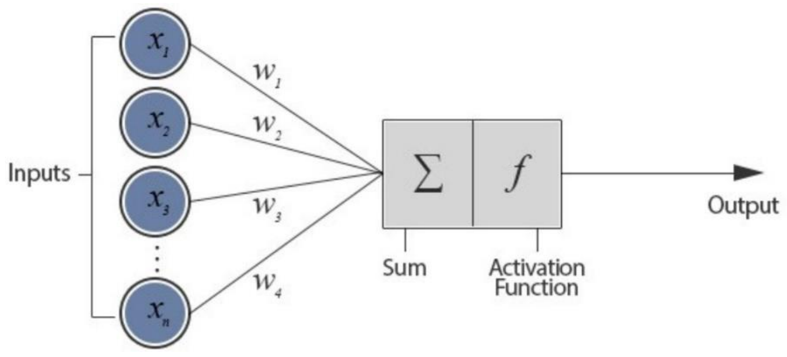


Figure 2.3. The Artificial Neuron – Basic Concepts (Agatonovic-Kustrin & Beresford, 2000)

The exact equation for a neuron can be written as:

$$y = f\left(\sum_{\mathrm{i}} w_i x_i + b \right) \tag{2.7}$$

As the input w_i gets closer to the model, the value of output *(y)* increases with it (Patterson & Gibson, 2017).

2.2.2 Activation Function

Neural networks are used in the approximation of nonlinear functions and different patterns can be satisfied through the activation function (Sutton & Barto, 2018). Because real-world problems are usually non-linear transformation (Goodfellow et al., 2016). Relu activation function is used in this thesis.

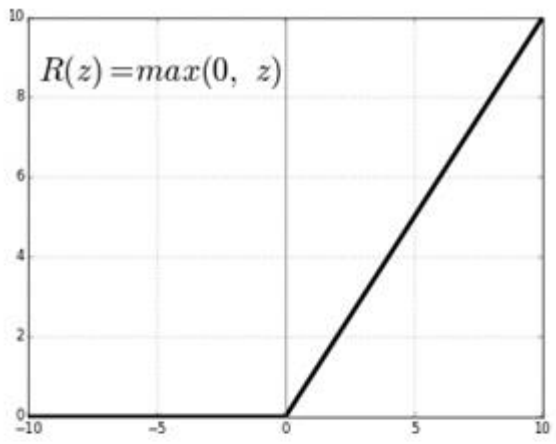


Figure 2.4. RELU Graph

• Rectified Linear Unit (ReLu):

$$f(x) = max(0, x) \tag{2.8}$$

Relu has proved to be one of the best activation functions and has a wide range of uses. It prevents neurons from converging their weights zero (Glorot et al., 2011).

2.2.3. Training the Network

The most important task of the DNN is to adjust the weights and bias and try to minimize the cost function *J(θ)* during training using the *backpropagation* technique (Larochelle et al., 2009). This technique starts with the weights by giving a random value for each input and then finds an error value by subtracting the calculated output value from the actual output (Patterson & Gibson, 2017). *(θ)* represents all DNN parameters and the cost function shows how to change parameters to minimize the error value. Thus, weights are updated usually using the gradient descent algorithm (Goodfellow et al., 2016).

This derivative called gradients and for each parameter shown as:

$$\frac{dC}{d\theta} \tag{2.9}$$

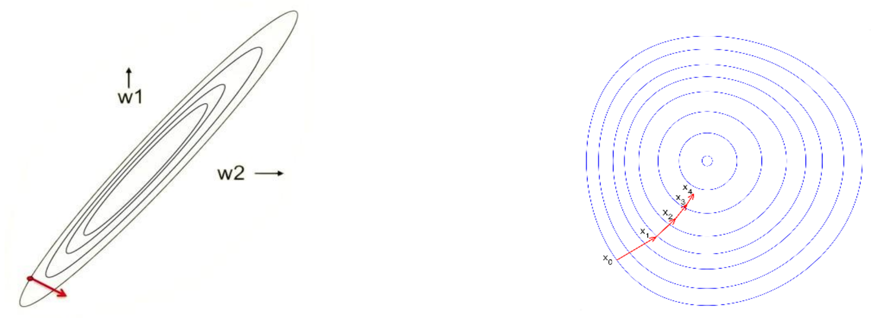
How to found the minimum cost is, the weights are updated with a small number of gradients each time, starting to take small steps depending on the learning rate towards the minimum direction which is meaning the opposite of the gradient direction (Goodfellow et al., 2016). That’s why the name "Gradient Descent". This process shows with this equation (2.10) and the learning rate donated by *a*:

$$\theta_{new} \leftarrow \theta_{old} - \alpha \frac{dC}{d\theta} \tag{2.10}$$

Stochastic gradient descent is the most popular algorithm in deep learning such as *Adam* (Kingma & Ba, 2015) and *AdaGrad* (Duchi et al., 2011). Because weights converge faster by the optimal learning rates based on the distribution of the training data. These algorithms provide how much the values of the weights can be increased or decreased. Thus find the local or global minimum more correctly (Kingma & Ba, 2015).

2.2.4. Normalization

In real-world data, variance and average scales between the columns of the data can significantly vary greatly (Srivastava et al., 2014). This problem makes it difficult to choose the optimal learning rate and other hyperparameters and consequently, misdirection of the gradients occurs. Batch normalization (Ioffe & Szegedy, 2015) solves this difficulty which takes a sample from the dataset as mini-batch and normalizes their mean equal to zero and variances as a unit (Ioffe & Szegedy, 2015). Thus gradient finds the optimum value.



Unnormalized data surface and the gradient is indicated by the red arrow.

Normalized data surface and the gradient find optimum value faster than unnormalized data.

Figure 2.5. Illustration of the Benefit of Normalization (Ioffe & Szegedy, 2015)

2.2.5 Dropout and Early Stopping

Dropout and early stopping methods are used to prevent overfitting in neural networks. Mainly, the dropout method (Srivastava et al., 2014) means to eliminate the neurons with a pre-specified probability value. Because when all weights get updated together with each training iteration however some of them have become better predictive capability than the other neurons and it continues until the end of this training (Goodfellow et al., 2016). Thus, this causes some neurons to decrease their predictive capacity. For this reason, the dropout technique allows us to make deeper and wider neural networks, by disabling the weight of some neurons at each training step and trying to equalize their predictive capacity (Goodfellow et al., 2016).

The early stopping method means that the training stops if the total number of awards received by the model does not increase within a predefined episode. So, this method reduces the model's dependence on training data (Srivastava et al., 2014).

2.3 Deep Reinforcement Learning

Deep Reinforced Learning (DRL) has attracted a lot of attention from the AI committee. In 2017, an artificial intelligence application won a game against Go champion, which is one of the most complex board game in the world and it shows the world how intelligence learning models can be (D. Silver et al., 2016).

The biggest difference between DRL and RL is that DRL can work in high-density data and continuous action problems due to DNN as a function approximate policy search or value function method (Goodfellow et al., 2016). This chapter introduces the *Deep Q-Network* (DQN) method which is used for this thesis due to the state size of our inventory and the number of actions the agent can be taken (Sutton & Barto, 2018).

2.3.1 Deep Q-Network (DQN)

As discussed earlier, *Q*-learning is the most popular model-free method in RL. In the first instance, it is unsteady when using DNN. Thus, most applications of it are limited to tasks with small-scale data. For the First Time, DQN (Mnih et al., 2013) has been developed that can play Atari 2600 games more successfully than humans.

The loss function is minimized by using stochastic gradient descent and *ε*-greedy policy is applied to provide exploration and also it is a value-based algorithm like Q-learning (Mnih et al., 2015). However, DQN using DNN to estimate the best Q^* value function, $Q(s,\ a;\ \theta)$ which weight represented by θ. In this case of DQN, the loss function $L(\theta_t)$ is defined by:

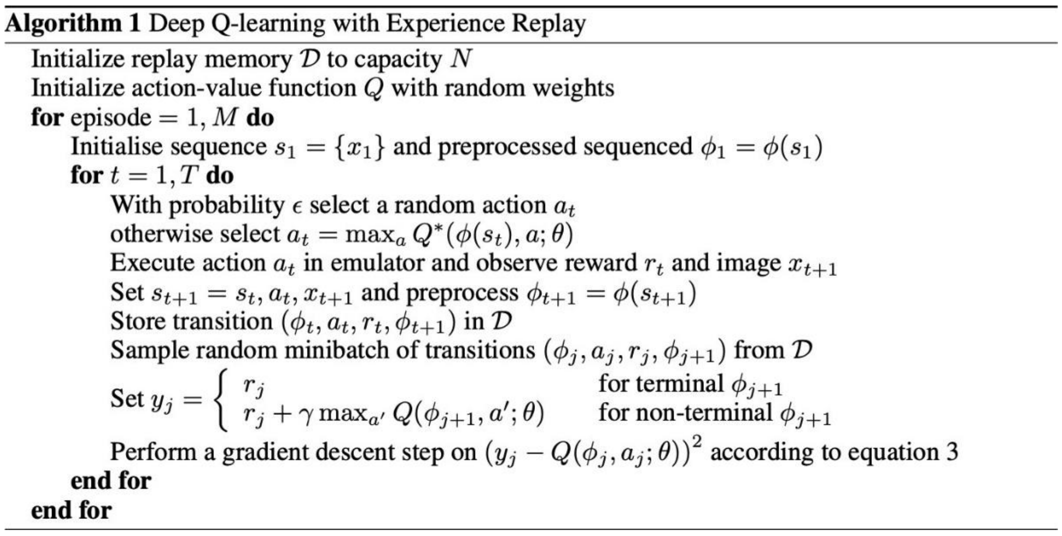
$$L(\theta_j) = \mathrm{E}_{s,a}\left[\left(Q_{target} - Q_{predict}\right)^2\right] \tag{2.11}$$

$$= \left(r_j + \gamma \max_a Q(s_{j+1}\ ,\ a\ \mid \theta') - \max_a\ Q(s_{j,a} \mid \theta')\right)^2$$

The most important problem for DQN is high variance because it causes overfitting or weights that may not let the system learn very well (Goodfellow et al., 2016). For this reason, two main methods are developed to overcome this problem. The first one is the *target Q-network* method which is used as a loss function. As given in equation (2.11) loss function equation contains the same θ' parameters causing increase of variance and decrease of stability of learning. Because of these reasons, target *Q*-network method updates θ' the parameter of *Q*-target after hundreds or thousands of training steps. (Mnih et al., 2015)

$$\left(r_j + \gamma \max_a Q(s_{j+1}\ ,\ a\ |\theta^{-})\ -\ \max_a\ Q(s_{j,a}\,|\,\theta')\right)^2 \tag{2.12}$$

The *experience replay memory* is the second method. Unlike the *Q*-learning, this method stores the previous experiences of the agent instead of action-state values and it is shown as (s_j, a_j, r_j, s_{j+1}) (Mnih et al., 2013). Thus, the experience replay method uses with mini-batch in the training steps and they are playing a big role in decreasing the variance of large-scale, break the correlation between patterns and finding efficiently loss value at every training iteration in the future. The DQN algorithm is shown below.



```
Algorithm 1 Deep Q-learning with Experience Replay
  Initialize replay memory $\mathcal{D}$ to capacity $N$
  Initialize action-value function $Q$ with random weights
  for episode $= 1, M$ do
      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
      for $t = 1, T$ do
          With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
          Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
          Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
      end for
  end for
```

Figure 2.6. Deep Q-Network (DQN) (Mnih et al., 2013)

2.4 Inventory Optimization

Inventory refers to materials and goods purchased for the profit of a company. One of the most important steps in the supply chain management is the inventory management policy that is the tracking of inventory from producers to storage and from these facilities to sales stores (E. A. Silver, 1981). In other words, inventory management requires knowing when to order, how much to order, and where to store goods (Jones & Riley, 1985). Accordingly, it is expected to follow a policy that gives appropriate orders of unknown demand fulfillment and minimize the holding cost.

Many methods have been developed to find suitable policies. The (*s*, *S*) inventory control policy is introduced.

2.4.1 (*s*, *S*) Inventory Policy

In an (*s*, *S*) inventory policy, review period of current inventory levels could be either periodic or continuous (Zheng & Federgruen, 1992). When inventory level drops to the reorder point *(s)*, the order quantity (*Q*) equals to the current stock level is subtracted from optimal order-up-to level (*S*) (Veinott & Wagner, 1965). Therefore, *Q* does not take a constant value as in other policies like (*r*, *Q*). Considering the example in Figure 2.7, it is seen that the blue line is the inventory level, optimal order-up-to level *(S)* is equal to 50 and the optimal reorder point *(s)* is equal to 10. When the inventory level drops below the reorder point, an order is placed with the inventory level equal to *S*.

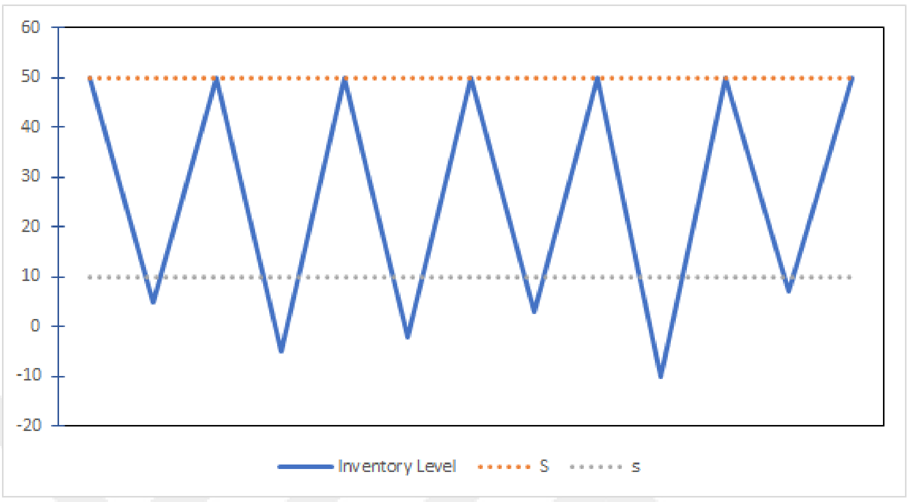
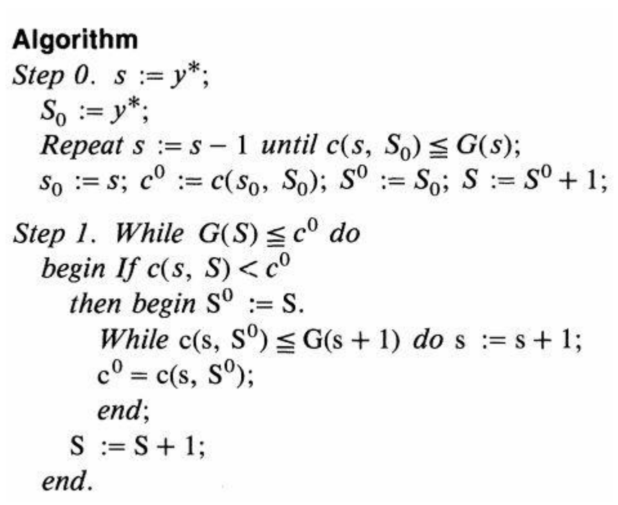


Figure 2.7. (*s*, *S*) Inventory Policy

In this thesis, the DRL technique is compared with a static model whose optimal (s, S) levels are calculated by using Zheng and Federgruen's algorithm (1992) given in Figure 2.8. The algorithm is simple to comprehend and implement. Let,

- D = The one-period demand
- $p_j = Pr\{D = j\}, \quad j = 0,1,2,...;$
- y = The beginning of the inventory position, integer y.

First of all, step *0* is calculated the base stock level indicated as y^* with the Poisson percent point function using penalty cost *(p)* and fixed setup cost *(K)*.



```
Algorithm
Step 0. $s := y^*$;
  $S_0 := y^*$;
  Repeat $s := s - 1$ until $c(s, S_0) \leqq G(s)$;
  $s_0 := s$; $c^0 := c(s_0, S_0)$; $S^0 := S_0$; $S := S^0 + 1$;

Step 1. While $G(S) \leqq c^0$ do
  begin If $c(s, S) < c^0$
    then begin $S^0 := S$.
      While $c(s, S^0) \leqq G(s+1)$ do $s := s + 1$;
      $c^0 = c(s, S^0)$;
      end;
    $S := S + 1$;
  end.
```

Figure 2.8. Finding Optimal (*s*, *S*) Policy Algorithm (Zheng & Federgruen, 1992)

The optimum *s* value is calculated for *S* with its lower limit fixed at S_0. The reorder point *s* value is decremented one unit until the period when the expected cost, *G*(*s*), is less than the long-run average cost *c*(*s*, S_0) and it is defined as s_0. The cost function value is calculated with expected total costs showing as *M*(*j*), (*k* (*s*, *y*)) in the equation.

$$c(s, S) = k(s, S)/M(S - s) \tag{2.13}$$

When beginning inventory is $s + j\ (y)$ units, $j > 1\ (y > s)$, $M(j)\ (k(s, y))$ be the expected total costs until the next order arrives.

$$m(0) = (1 - p_0)^{-1},\ M(0) = 0 \tag{2.14}$$

$$M(j) = M(j-1) + m(j-1), \qquad j = 1, 2,\ ....$$

$$k(s, y) = K + \sum_{j=0}^{y-s-1} m(j) G(y - j) \qquad y > s \tag{2.15}$$

M(*) and *k*(*s*, *y*) clearly define the discrete renewal equation. Finally, the long-run average cost c (*s, S*) is calculated using the following equation.

$$c(s, S) = k(s, S)/M(S - s) \tag{2.16}$$

In step 1, *c* (*s*, *S*) function is recalculated by using s_0 and *S* which equal to S_0 +1. The maximum inventory level point *S* is increased one unit until the one-period expected to cost *G(S)* value is higher than *c*(*s*, *S*) function. However, when S_0 is equal to *S*, reorder level *s* is incremented one unit until $c(s, S_0) > G(s + 1)$ $c(s, S_0)$ is updated with new reorder level *s* afterwards. At the end of the algorithm, it returns optimal reorder value *s* and maximum inventory level point *S* as S_0.

For numerical example, The example model at Veinott and Wagner is chosen, with linear holding and holding costs, zero lead time, and Poisson distributed one-period demands (Veinott & Wagner, 1965).The following parameters are common to all 5 problems and identical to those used in Veinott and Wagner (Veinott & Wagner, 1965):

- fixed setup cost, *K* = 60;
- holding cost rate, *h* = 1;
- penalty cost, *p* = 9.

The mean one-period demand (μ) is displayed from 10 to 25, increasing by 5.

Table 2.1. Performance of Algorithm

µ	**Reorder Point (*s*)**	**Order-Up-to Level (*S*)**
10	6	40
15	10	49
20	14	62
25	19	56

Finally, the applications of (*s*, *S*) policy in the literature are examined. In the study of Fries (1975), the model is examined for the perishable product when the lifetime is two periods or more. Considering the defined cost function, it has shown that the policy (*s*, *S*) is not optimal when the product life is 2 periods or more and the planning period is at least the product life (Fries, 1975). A paper of Brunaud et al (2019) compared using the (*r*, *Q*) policy and (*s*, *S*) policy on the gas storage problem. A continuous review of the (*r*, *Q*) policy is adapted to a discrete-time easing the need of having the inventory level exactly at the reorder point to trigger replenishment. Thus, although the efficiency of these policies is the same, the (*r*, *Q*) policy has become a better policy by responding to demand faster as expected (Brunaud et al., 2019). Also, Chen et al (2001) try to explore the impact of variations in service level constraints on attained service levels. The experiment is performed for a continuous (*s*, *S*) review policy and it proves that inventory policy with minimal service level models performs better than a policy with mean service level constraints (Chen & Krass, 2001).

CHAPTER 3

INVENTORY OPTIMIZATION EXPERİMENT with DQN AGENT

In this thesis, the Deep *Q*-Network algorithm is proposed in section 1.1. is developed in Python language by using the Keras library developed by Google engineers (Chollet & others, 2015). The reasons for using the Keras library are:

- Easy to use but also supported advance TensorFlow functions
- GPU support
- Can be work with NumPy (S van der Walt et al., 2011) and Pandas (McKinney, 2010)
- Support other machine learning frameworks too

This section describes how the agent is trained and tested. In addition, the design and development phase of the environment is discussed.

3.1. Environment

The environment state is considered to be the current inventory level in the system. The actions are defined to be the amount of order that is given from the warehouse. The variables and notations used are defined in Table 3.1.

Table 3.1. Supply Chain Variables

$I_i(t)$, *(i = RL, SS)* represents the inventory level of the actor *i* at time step *t*.
$O_i(t)$, *(i = RL, SS)* denotes the ordering size of the actor *i* at time step *t*.
$D_t(\mu)$, random variable, denotes the demands coming with the mean (μ) of the Poisson distribution at time step *t*
$P_i(t)$, *(i = RL, SS)* represents the profit of the actor *i* at time step *t*

In the basic supply chain process, a decision-maker, or is called it an agent, is responsible for 3 stages: distribution, order and demand fulfillment. In this experiment, the order and demand fulfillment stages are considered. After each periodic review, the agent decides whether or not to place an order to satisfy the demand with a non-lead time. If the agent decides to order, then it bears the fixed ordering cost as well as the holding cost. For the unsatisfied demand, it incurs penalty cost as the unsatisfied demand amount. In this thesis, a static (*s*, *S*) inventory policy is compared with an RL agent-based policy. In Table 3.2, the values of variables in the supply chain are set.

Table 3.2. Supply Chain Constant Variables

Unit Price, *P*	100
Fixed Ordering Cost, *Co*	80
Periodic Time Review (time-step)	1
Unit Holding Cost (per periodic-review), *h*	1
Unit Penalty Cost (per periodic-review), *p*	80
Warehouse Capacity	100
Lead time	0

3.1.1. The Profit Function

Equation (3.1) shows how to calculate the total profit for a single period:

$$P_i(t) = (D_t(\mu) \times P) - (Dp \times p) - (Nc \times Co) - (h \times I_i(t)) \qquad (3.1)$$

According to (3.1) ordered amount is multiplied by unit selling price and the penalty cost for unfulfilled demand, holding cost as well as fixed ordering cost are decremented from that amount. *Nc* is a binary variable and is equal to 1 if the agent made an order, otherwise 0. Total profit is seen as the performance measure of the supply chain network.

3.1.2. State and Action Space

Once again, the states and the actions refer to the inventory level and the order size, respectively. The state space (inventory level) is considered to be the levels between

-20 and 100 in training stage. The backlog is not allowed after the inventory level equals -20 Hence, the state space contains 120 dimensions. On the other hand, the action space includes the integer values between 0 and 6. Hence, actions can occur in 7 possible discrete values. It is indicated by $a(t)$ and the equation is defined below.

$$a(t) = k*10 \qquad k = [0,1,2,3,4,5,6] \tag{3.2}$$

3.1.3. The Reward Function

The RL agent selects the most proper actions according to the reward function. Here, the reward function is considered to be profit. Profit is undoubtedly one of the most important variables in the supply chain and therefore is considered it as a reward function as (3.3) shows by:

$$r = \sum_{t=0}^{m} P_i(t) \tag{3.3}$$

In (3.3), *m* represents training time and as can be seen in the equation above, the model receives the reward only once in one episode. This function is designed for give freedom to the agent in the environment and prevent over-compliance with training data.

3.1.4. Determining the Reorder Point and Order up to Quantity for the Static Model

As mentioned in chapter 2, the RL-based solution is compared to a static model. The utilized analytical model for the static inventory problem is developed by Veinott and Wagner (1965). It is used to find the reorder point (*s*) and the order up to levels (*S*). As in the RL model, the performance metric is considered same in the static model as the RL model. Namely, total profit is calculated as in (3.1) by assuming the same assumptions.

3.2. Training of the RL Agent

As discussed in the problem description, the problem consists of a supply chain network with a single-echelon level. There is a single type of product and zero lead time from the factory. The DQN algorithm is implemented for the RL agent. The parameters and their values considered in the training are shown in Table (3.3).

Table 3.3. Hyperparameters of DQN

Replay Memory Size	7000
Batch Size	512
Discount factor γ	0.98
Greedy- ε^0	0.05
Greedy- ε^1(after 50 episode)	0.01
Greedy- ε^2 (after 100 episode)	0.005
Loss Function	MSE
Optimizer	ADAM
Dropout	0.1
Kernel Initializer	He-uniform
Activation F.	ReLu, Softmax
Early Stopping (Episode)	50

In the training process, the random.poisson function of the NumPy library (S van der Walt et al., 2011) is used for demand distribution and the average of demands (μ) is taken as 10. This thesis aims to observe how a trained agent behaves under changing environmental conditions, in other words, is it would overfit the training data set. The values in our action dimensions are a great value that it can exceed the maximum capacity with a single random action. When the agent exceeds the maximum capacity or falls below the minimum capacity, the episode ends and the and then the training continues with the next episode. Thus, depending on the reward function, the agent learns the minimum and maximum capacities. All are shown to the agent with an observe state. This state consists of the current state, reward value, binary game over value which refers to whether it exceeds maximum capacity or minimum capacity, respectively.

The mean square error (MSE) is used as the loss function in the DNN and the task of the agent is to reduce the loss value as much as possible. *ADAM* is preferred for the learning rate optimization algorithm and its parameters are determined according to the study (Kingma & Ba, 2015). Adam is an algorithm that updates neuron weights more efficiently depending on the learning rate. It has developed use the advantages of two algorithms, *AdaGrad* and *RMSProp*. AdaGrad is used to preverse the learning rate of parameters with sparse gradients and mostly use in computer vision problems. Also, RMSProp preverse learning rate of parameters but it adapted learning rate on average of the gradients for the weights. Adam algorithm make use mean first moment and second moment of uncentered variance.

All models in this thesis have an input layer, two hidden layers and an output layer. These layers use the *ReLu* and *Softmax* activation functions, excluding the input layer, respectively.

The number of hidden neurons and learning rate is determined as the most important parameters in this experiment. Because, the learning rate is the frequency of updating the weights. Hidden layers divide the input data into specific patterns, and each hidden layer function focus only one pattern. For example, a hidden layer functions that are used to identify bird's beak, wing and eyes used in combination by subsequent layers to identify bird species in images. Depending on the number of variables in the problem, those layer values: [32,16], [64,32], [128,64] are selected. The learning rates are considered to be: 0.01, 0.001, and 0.0001. Table 3.4 show the simulation results based on the RL-based simulation solution. In the simulation, each timestep equals one day and training time is considered to be 800 days.

Table 3.4. Settings Hyperparameters of DQN

Layer Size	**32 - 16**			**64 - 32**			**128 - 64**		
Model no	1	2	3	4	5	6	7	8	9
Learning Rate	0.01	0.001	$1x10^{-4}$	0.01	0.001	$1x10^{-4}$	0.01	0.001	$1x10^{-4}$
Best Reward	2522	3102	405	7131	959146	10258	512212	64651	35214

Model 5 produces the best result compared to others. When looking at the 10th (Figure 3.1), 100th (Figure 3.2) and 255th (Figure 3.3) training episodes of this model, the policy can be understood correctly.. In each Figure 3.1, 3.2 and 3.3, from top to down, inventory level, $I_i(t)$, order amount, $O_i(t)$, and cumulative profit, r, can be observed, respectively.

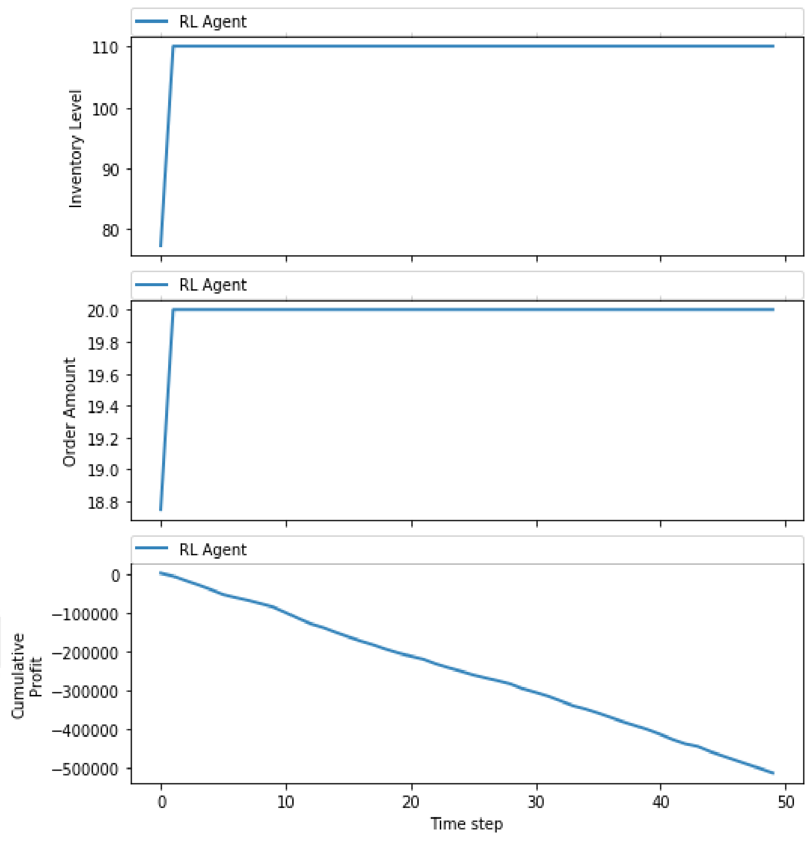


Figure 3.1. The Agent Trained Until the 20th Episode

In Figure 3.1., It has been observed that the agent has not learned the situation-action pair yet. The order amount stuck around 20 and it exceeds the maximum inventory capacity level. Therefore, the model is exposed to the cost of the penalty.

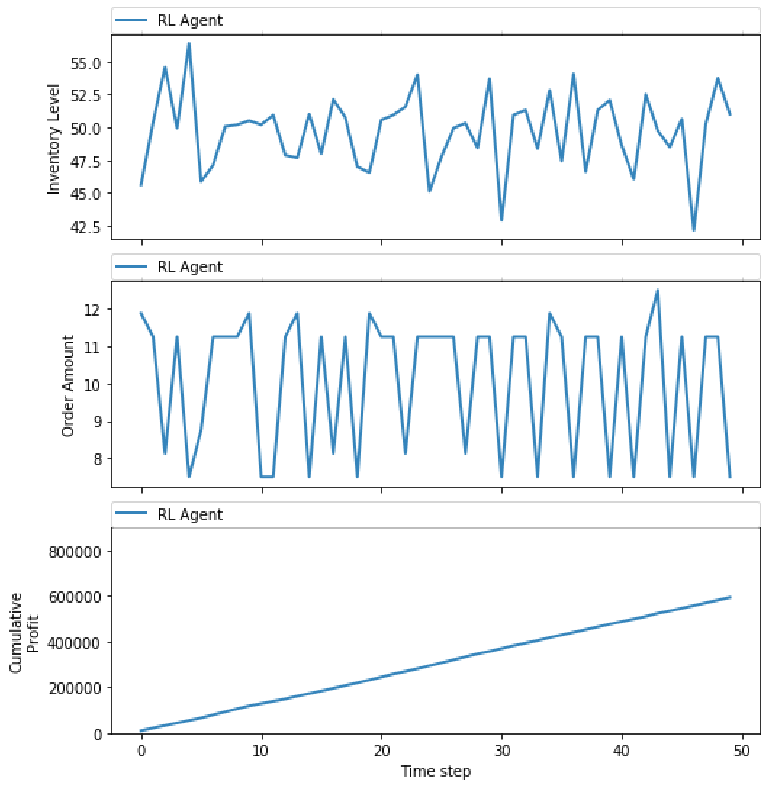


Figure 3.2. The Agent Trained Until the 100th Episode

It is observed that the agent started to learn the state-reward relationship about better in episode 100 and by exploring new action policies. Hence, the system starts to make a profit.

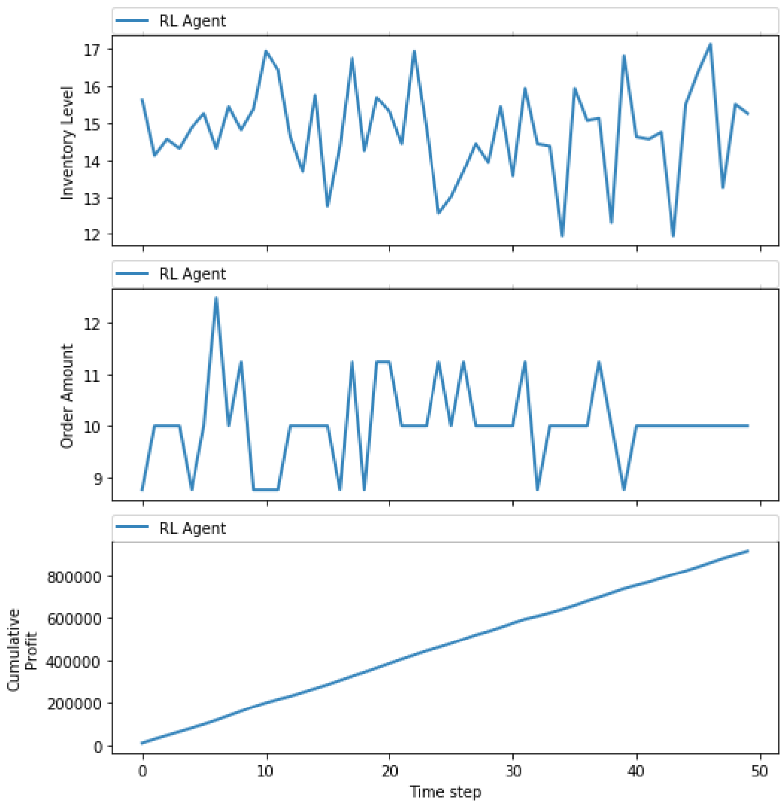


Figure 3.3. The Agent Trained Until the 250th Episode

In Figure 3.3, the agent completes training with the early stopping method in the 255th episode. Compared to the 100th episode agent, it can be observed that the inventory level is reduced from an average of 50 to 15 level. Thus, cumulative profit increased from $6*10^5$ to $1*10^6$.

3.3. Comparison of RL Solution with the Static Approach

This section is shown how the intelligent trained agent can adapt to the changing environment by comparing it with a static (*s, S)* inventory policy. There are four different (s, S) value update scenarios and different numbers of experiments depending on the value of the average of Poisson distributed one-period demand (μ) in each scenario. For the static model, the (*s*, *S*) values are calculated by analytically. Training time is 1800 days and the average of every 16 days is calculated to make the graphics easier to understand. The percentage of Profit (RL-SS) shows how much profit DRL agent makes than (*s, S*) policy in percentage. From top to down the Figures 3.4 and 3.5 show inventory level in the warehouse, $I_i\,(t)$, amount of order given to the factory, $O_i\,(t)$, and the cumulative profit of the network, r, respectively.

Three experiments is defined based on average demand amounts arriving at the warehouse. These are 5, 10 and 20 shown in Table 3.5. In the static model, (*s, S*) levels are calculated according to the mean demand arrival and the model is run accordingly. Once again three experiments are defined as follows:

- First experiment $\mu = 5$
- Second experiment $\mu = 10$
- Third experiment $\mu = 20$

Also, scenarios based on the dynamic nature of demand arrivals are defined. In the first scenario mean demand is static and does not change in time. In addition, scenarios, where mean demand arrivals change dynamically in time, are also considered. Table 3.5 shows the results of the experiments for the first scenario for both static and DRL model. It is observed that the DRL results are always better than the static results.

Table 3.5. Results of the First Scenario

μ	5		10		20	
Agent	RL	(*s, S*) Static	RL	(*s, S*) Static	RL	(*s, S*) Static
Total Orders	8920	8890	17980	16548	35120	33000
Total Profit	753745	715260	2106240	1518215	3911355	3175420
Percent of Profit (RL-SS)	5,38%		38,73%		23,17%	

Since the model is trained in an environment where μ equal to 10, it is expected that it will result with the best profit in Table 3.5. The result is summarized in Figure 3.4.

In Figure 3.4, although the DRL agent orders more amount than the static model (*s*, *S* agent), it keeps the average inventory level lower. The static (*s*, *S*) agent keeps the inventory level at around 20, while the RL agent keeps it at around 15.

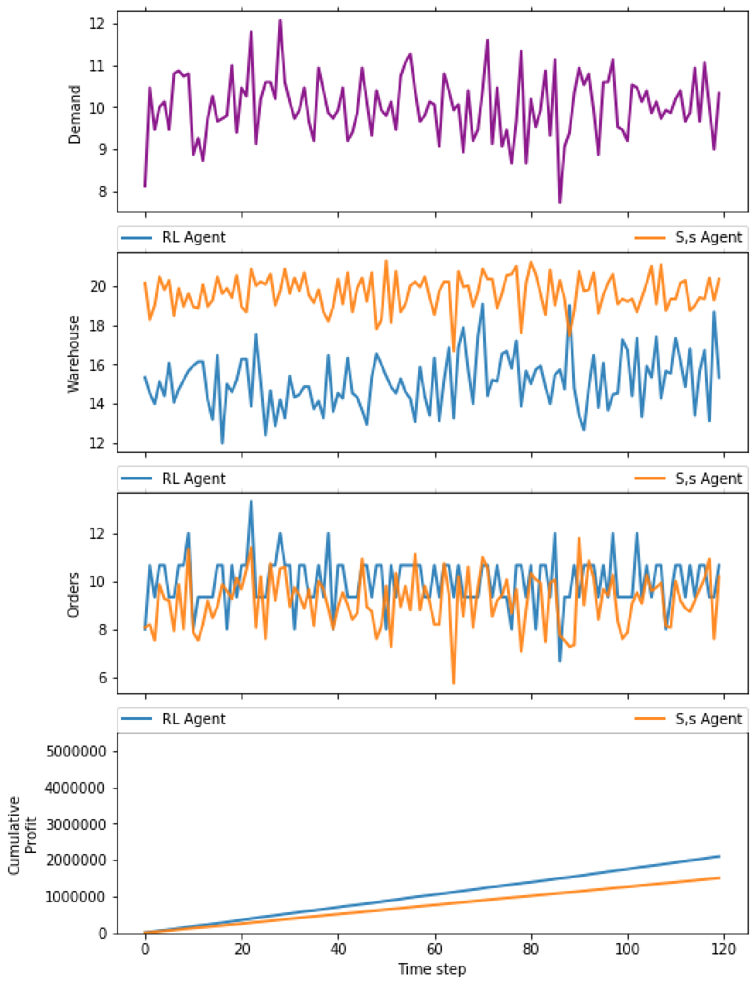


Figure 3.4. 2nd Experiment Chart of the First Scenario

In Figure 3.5. first experiment's results are shown. In that figure, even when μ is equal to 5, the RL agent manages to keep the inventory level lower than the static (*s*, *S*) agent. Moreover, the number of total orders of the RL agent is higher than the static agent.

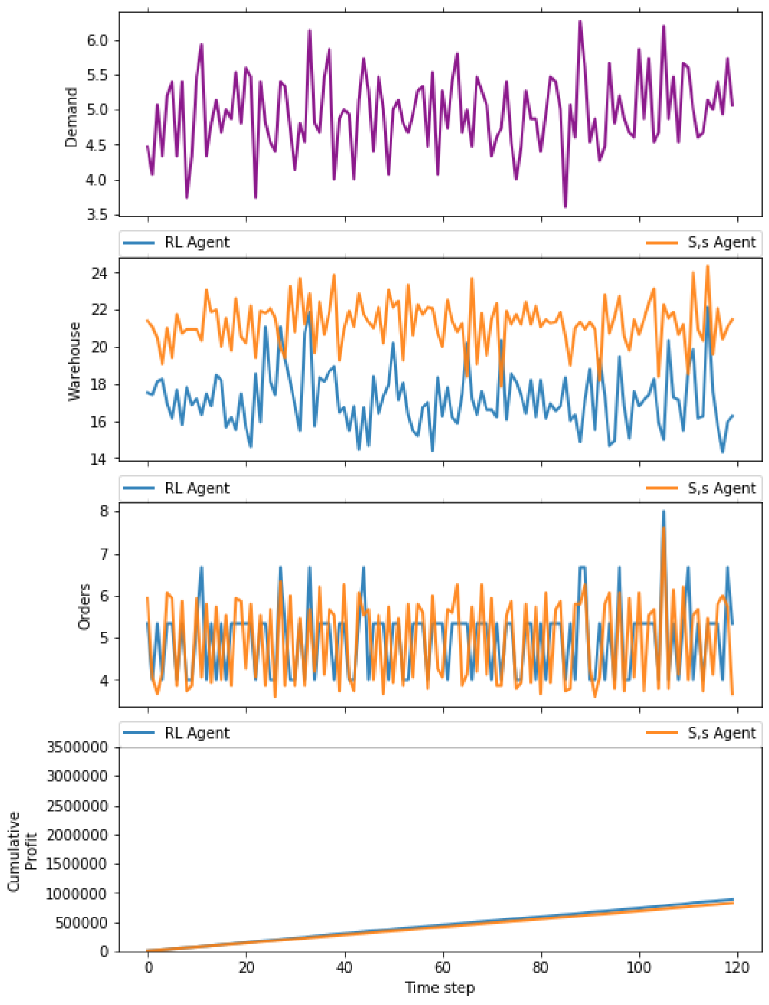


Figure 3.5. 1st Experiment Chart of the 1st Scenario

In the second scenario, μ changes every 300 days and (*s*, *S*) values are updated after 150 days depending on μ.

- μ value is chosen with equal probability for values of 30,10,5,25 and 7 with using the *random.choice* function in the NumPy library. Thus, μ changes every 300 days.

Table 3.6. Results of the Second Scenario

μ	(30,10,5,25,7)	
Probability	0.2	
Update rules	μ is changed every 300 days, *(s, S)* is updated after 150 days	
Agent	RL	*(s, S) Static*
Total Orders	24360	16923
Total Profit	2735819	1669308
Percent of Profit (RL-SS)	63,88%	

Table 3.6 summarizes the results of the second scenario. As seen in Figure 3.6., the adaptation of the RL agent to a changing environment is faster than the static agent (*s*, *S*), especially the subject about the fulfillment of demand.

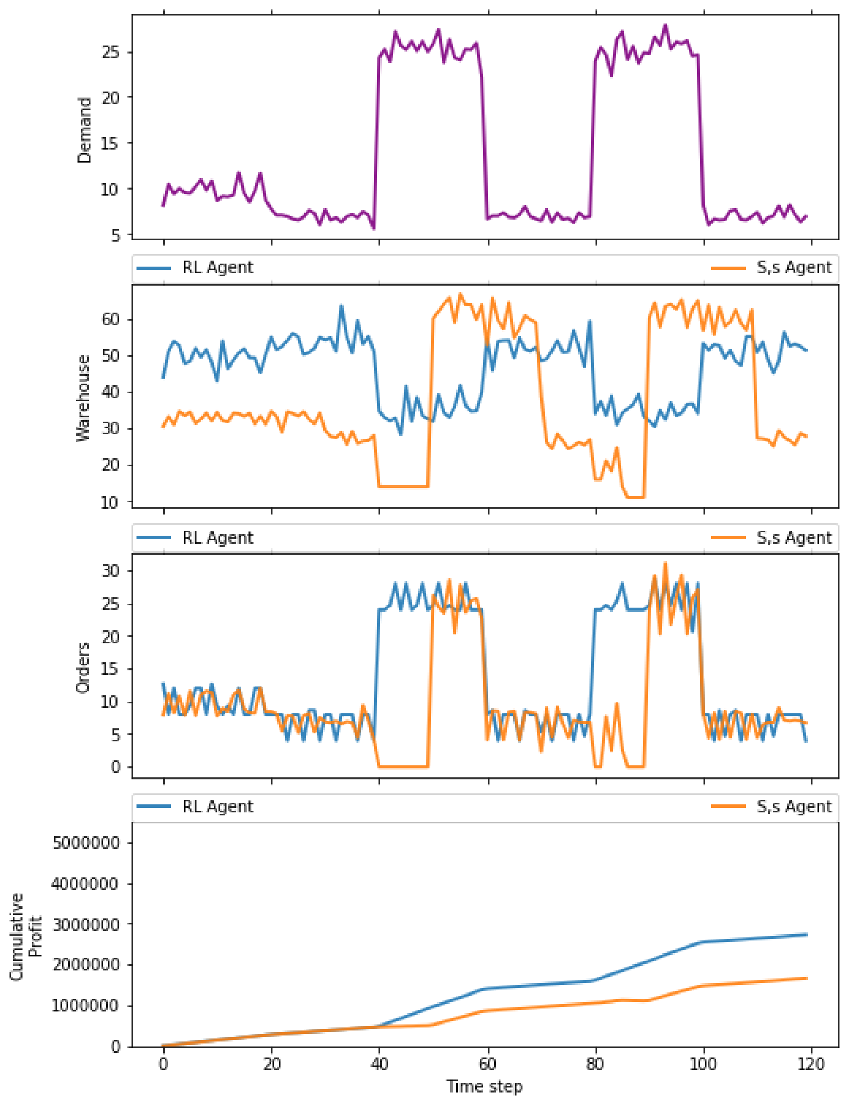


Figure 3.6. Experiment Chart of the Second Scenario

In the third scenario, μ changes every month. The average demand is estimated by considering the last three month's demand. Two experiments are considered in the third scenario.

- In the first experiment, μ takes the values of 10, 30, 5, 25, 7, 18, 4, 13, 20 and 27 with equal probabilities.

- In the second experiment, μ takes the values of 18, 5, 25 and 10 with probabilities of 0.3, 0.2, 0.2 and 0.3, respectively.

Table 3.7 summarizes the results for the first experiment explained in the first bullet above.

Table 3.7. Results of the 1st Experiment of the Third Scenario

μ	30,10,5,25,7,18,4,13,20,27	
Probability	0.1	
Update rules	μ is changed every month, *(s, S)* takes the average of the μ in the past 3 months	
Agent	RL	*(s, S) Static*
Total Orders	29580	25992
Total Profit	3322037	2635682
Percent of Profit (RL-SS)	26,04%	

In Figure 3.7., although the RL agent keeps higher inventory level, it makes more profit by satisfying more demand than the static agent (*s*, *S*). Again DRL outperforms even in changing environment.

.

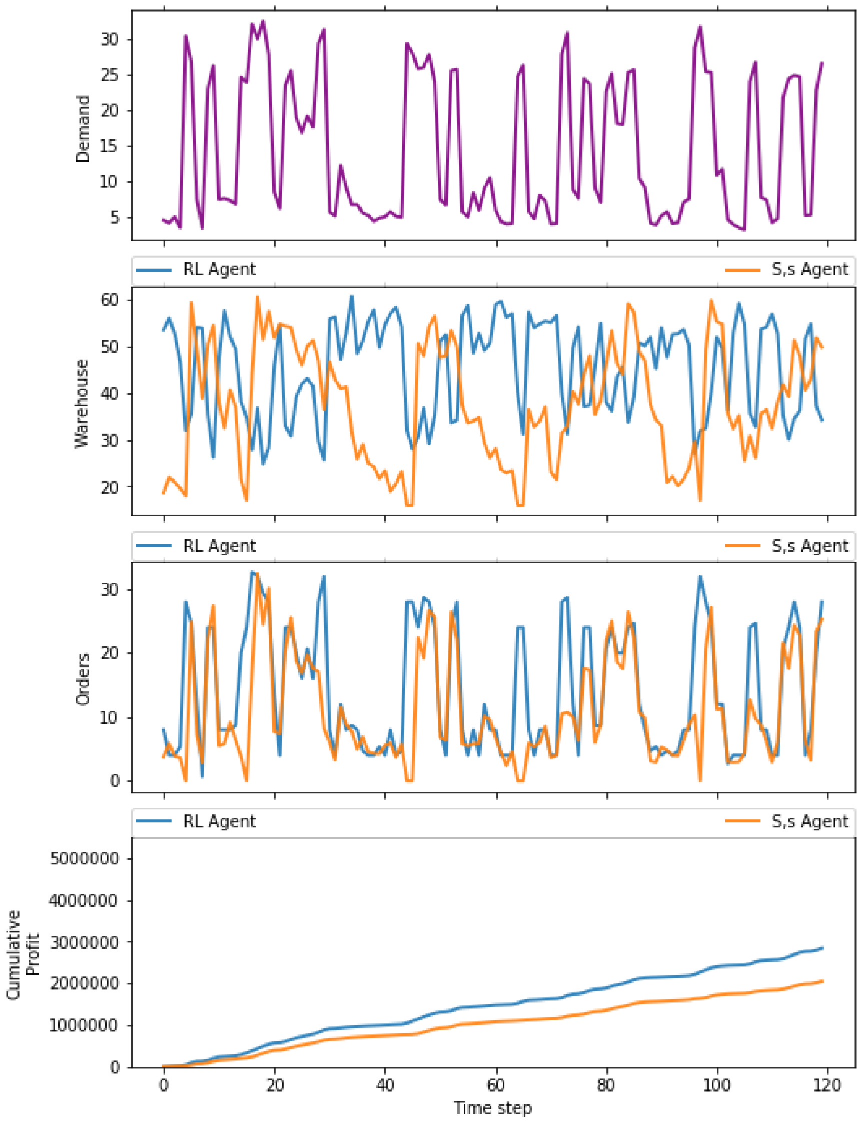


Figure 3.7. 1st Experiment Chart of the Third Scenario

Table 3.8. Results of the 2nd Experiment of the Third Scenario

μ	10,5,25,18	
Probability	0.3, 0.2, 0.2, 0.3	
Update rules	μ is changed every month, *(s, S)* takes the average of the μ in the past 3 months	
Agent	RL	*(s, S) Static*
Total Orders	25460	22683
Total Profit	2806395	2322296
Percent of Profit (RL-SS)	20,84%	

This experiment shows that the RL agent's inventory level is higher, but it appears to predict demands better.

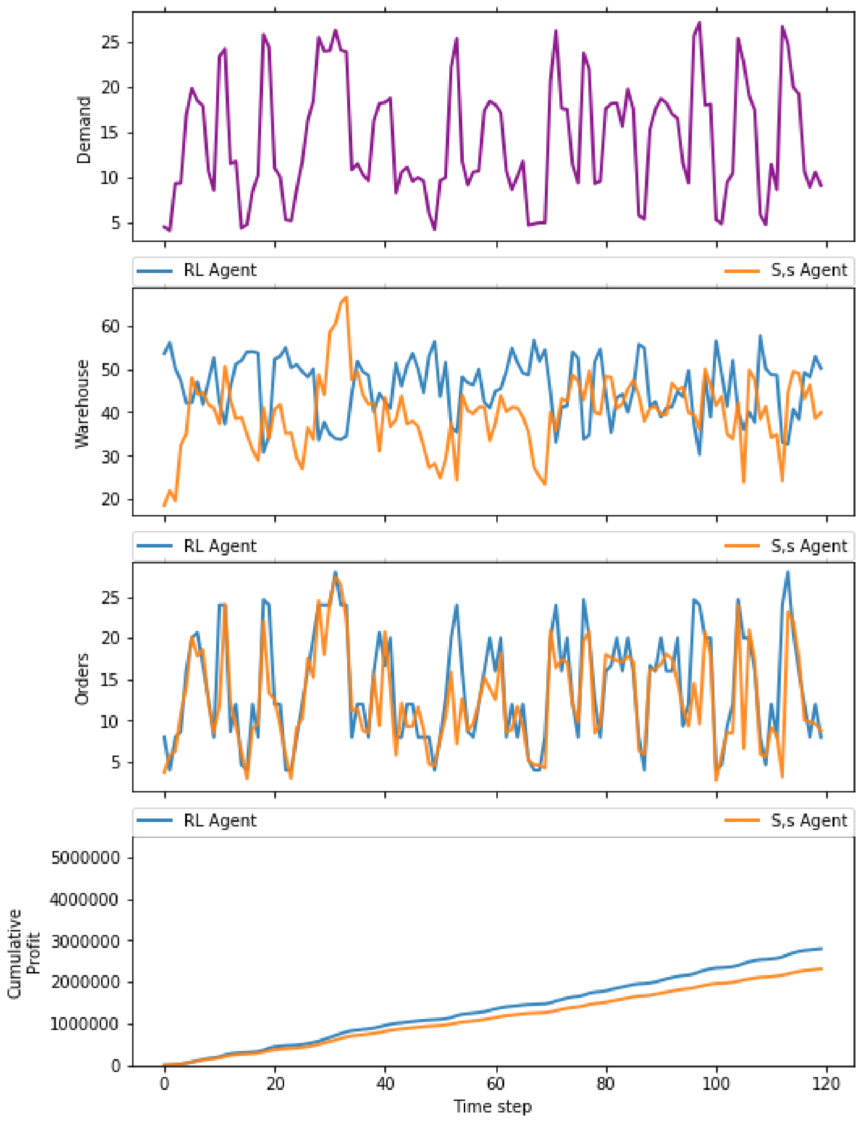


Figure 3.8. 2nd Experiment Chart of the Second Scenario

Finally, in our last scenario (the fourth one), the static agent (*s*, *S*) takes the average of the μ in the last months for update s and S values inally, in our last scenario, the agent (*s*, *S*) takes the average of the μ in the last months for update *s* and *S* values.

- μ values and probabilities of choice are used in the same values as in the previous scenario for each expirement.

Table 3.9. Results of the 1st Experiment of the Fourth Scenario

μ	10,5,25,18	
Probability	0.3, 0.2, 0.2, 0.3	
Update rules	μ is changed every month, *(s, S)* takes the average of the μ in the last months	
Agent	RL	*(s, S)* *Static*
Total Orders	24730	21030
Total Profit	2777101	2195809
Percent of Profit (RL-SS)	26,47%	

Table 3.9 summarizes the fourth scenario explained above. Although the orders are similar, As can be seen in Figure 3.9 that the RL agent is holding more inventory than the static one. But RL agent has managed to make higher profits in this experiment. This shows that RL agent adapts to the environment better than the static agent.

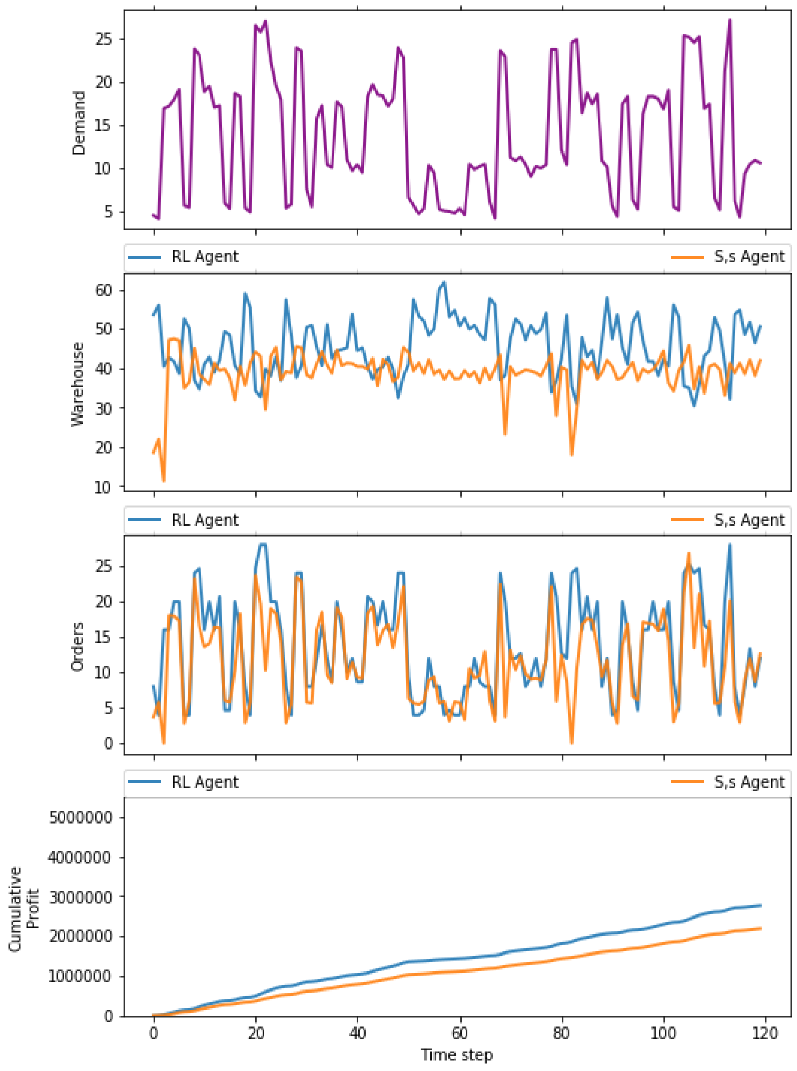


Figure 3.9. 1st Experiment Chart of the Fourth Scenario

Table 3.10. Results of the 2nd Experiment of the Fourth Scenario

μ	30,10,5,25,7,18,4,13,20,27	
Probability	0.1	
Update rules	μ is changed every month, *(s, S)* takes the average of the μ in the last months	
Agent	RL	*(s, S) Static*
Total Orders	32450	28952
Total Profit	3686268	2982561
Percent of Profit (RL-SS)	23,59%	

Table 3.10 and Figure 3.10 summarize the results for second experiment for the fourth scenario. In Figure 3.10, RL agent keeps inventory level lower than the static policy although it orders more products. Therefore, the RL agent makes more profit than the other policy.

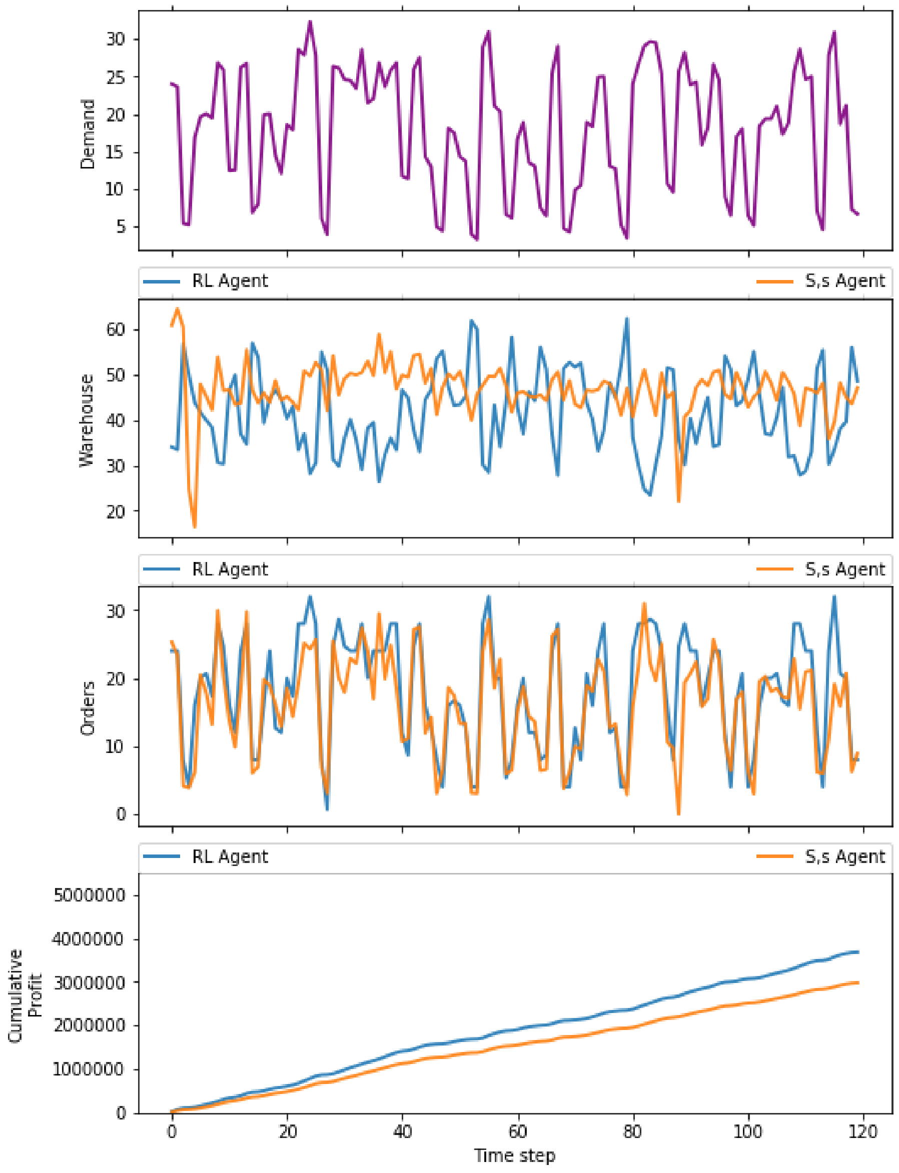


Figure 3.10. 2nd Experiment Chart of the Fourth Scenario

Since μ comes with a certain probability in the experiments, each experiment is run 5 times to make the results more precise. The results are then averaged over all 5 runs and they are summarized in Table 3.11.

Table 3.11. The Result of All Experiments Running 5 Times

		Total Profit		Percent of Profit (RL-SS)
Agent		RL	SS	
1.Scenario				
	1.1.Experiment	751545	715160	5,08%
	1.2.Experiment	2076248	1519215	36,66%
	1.3.experiment	3890214	3175420	22,51%
2.scenario				
	2.1. Experiment	2574529	1799458	43,07%
3. scenario				
	3.1. Experiment	3254532	2655702	22,54%
	3.2. Experiment	2816395	2327296	21,01%
4.scenario				
	4.1. Experiment	2736241	2175623	25,76%
	4.2. Experiment	3886268	2998561	29,60 %

At the end of all these experiments, the RL agent keeps the inventory level lower when periods of average demand decrease, otherwise it keeps the inventory level higher. Besides this, it appears imitating the pattern of demands when ordering. Finally, the RL agent makes more profit than the (*s, S*) agent.

CHAPTER 4

Conclusions and Future Work

The inventory optimization problem can be defined as finding the optimal control policy order to satisfy the uncertain demand and ensure that the product is in the right place at the right time. The aim of this study is to investigate how reinforcement learning deals with uncertain demand and how its performance outperforms a static policy. For this purpose, the environment of a single echelon inventory system is developed in a python language and an agent using the deep reinforcement learning algorithm is trained and tested in this environment. Thus, simulation helped to make it possible to observe what kind of policy reinforcement learning will be developed and allowed us to test policy in different environments.

A Deep *Q*-learning algorithm has been proposed to solve the proposed problem. According to that, and the defined agent could react the future demand by learning the process and follows an order policy accordingly. Besides, the model doesn't need any static input like (*s*, *S*) heuristic policy because the model trains itself according to the reward function, not the incoming demands.

In summary, the reinforcement learning application has proven that it can compete with current static methods to solve the proposed (*s*, *S)* inventory control problem. This thesis suggests that using reinforcement learning integrated with systems such as Warehouse Management Software (WMS) can increase the profitably of the system. If companies can make integration of inference with data stream, decisions allow faster and better execution.

As future works, two Deep Q-Network agents can be studied to increase the value of reward function by training each other. Besides, the Deep Deterministic Policy Gradient (DDPG), which is an RL algorithm that has continuous action spaces, can also be used in inventory optimization problems.

REFERENCES

Agatonovic-Kustrin, S., & Beresford, R. (2000). Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis*, *22*(5), 717–727.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). *End to End Learning for Self-Driving Cars*. 1–9.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. 1–4. arXiv preprint arXiv:1606.01540.

Brunaud, B., Laínez-Aguirre, J. M., Pinto, J. M., & Grossmann, I. E. (2019). Inventory policies and safety stock optimization for supply chain planning. *AIChE Journal*, *65*(1), 99–112.

Chen, F. Y., & Krass, D. (2001). Inventory models with minimal service level constraints. *European Journal of Operational Research*, *134*(1), 120–140.

Chollet, F., & others. (2015). *Keras*. GitHub. Retrieved from https://github.com/fchollet/keras

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Fries, B. E. (1975). Optimal Ordering Policy for a Perishable Commodity With Fixed Lifetime. *Operations Research*, *23*(1), 46–61.

Gavirneni, S. (2005). Information Centric Optimization of Inventories in Capacitated Supply Chains: Three Illustrative Examples. In *Supply Chain Optimization* (pp. 1–49). Springer-Verlag.

Giannoccaro, I., & Pontrandolfo, P. (2002). Inventory management in supply chains:

A reinforcement learning approach. *International Journal of Production Economics*, *78*(2), 153–161.

Glorot, X., Bordes, A., & Bengio, Y. (2011). *Deep Sparse Rectifier Neural Networks* (G. Gordon, D. Dunson, & M. Dudík (eds.); Vol. 15, pp. 315–323). JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Haykin, S. (1994). Neural Networks, A Comprehensive Foundation. In *International Journal of Neural Systems* (Vol. 05, Issue 04).

Houlihan, J. B. (1985). International Supply Chain Management. *International Journal of Physical Distribution & Materials Management*, *15*(1), 22–38.

Hutse, Victor, Verleysen, A., & Wyffels, F. (2019). *Reinforcement Learning for Inventory Optimisation in multi-echelon supply chains*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, *1*, 448–456.

Jones, T. C., & Riley, D. W. (1985). Using Inventory for Competitive Advantage through Supply Chain Management. *International Journal of Physical Distribution and Materials Management*, *15*(5), 16–26.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kemmer, L., & Read, J. (2018). Reinforcement learning for supply chain optimization. *The 14th European Workshop on Reinforcement Learning, EWRL 2018*, *14*.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15.

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, *10*, 1–40.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt & J. Millman (Eds.), *Data Structures for Statistical Computing in Python* (pp. 56–61).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. 1–9. arXiv preprint arXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., 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*(7540), 529–533.

Oroojlooy. (2019). Applications of Machine Learning in Supply Chains. In *Theses and Dissertations. 4364.* Lehigh University.

Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). Deep exploration via bootstrapped DQN. *Advances in Neural Information Processing Systems*, 4033–4041.

Patterson, J., & Gibson, A. (2017). *Deep Learning: A Practitioner's Approach* (1st ed.). O'Reilly Media, Inc.

Richardson, F., Reynolds, D., & Dehak, N. (2015). A unified deep neural network for speaker and language recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, *2015-Janua*, 1146–1150.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *4th International Conference on Learning Representations, ICLR 2016 - Conference*

Track Proceedings, 1–14.

Silver, D., Schrittwieser, J., Simonyan, K., Nature, I. A.-, & 2017, U. (2016). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354.

Silver, E. A. (1981). Operations Research in Inventory Management: a Review and Critique. *Operations Research*, *29*(4), 628–645.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–14.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book.

Tan, K. C., & Kannan, V. R. (1998). Supply Chain Management: Supplier Performance and Firm Performance. *International Journal of Purchasing & Materials Management*, *34*(3), 2–9.

van der Walt, S, Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, *13*(2), 22–30.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Veinott, A. F., & Wagner, H. M. (1965). Computing Optimal (s, S) Inventory Policies. *Management Science*, *11*(5), 525–552.

Watkins, C. J. C. H., & Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, *8*(3), 279–292.

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys*, *52*(1), 1–35.

Zheng, Y.-S., & Federgruen, A. (1992). Corrections to “Finding Optimal (s , S) Policies is About as Simple as Evaluating a Single Policy.” *Operations Research*, *40*(1), 192–192.