Sharif University Of Technology

Industrial Engineering

Bsc Thesis

A Deep Q-Network for the Beer Game:
Deep Reinforcement Learning for Inventory Optimization

Mohaddeseh Hosseinzadeh

sahar.hosseinzadeh.1999@gmail.com

Table of contents

1. Introduction

The Bullwhip effect refers to the phenomenon where small fluctuations in consumer demand can cause progressively larger fluctuations in demand as one moves upstream in the supply chain. The Bullwhip effect can be caused by several factors, including order batching, price fluctuations, and lack of information sharing among supply chain partners.It can result in excess inventory, stockouts, and poor customer service.

The beer game is a decentralized, multi-agent, cooperative problem that is widely used in supply chain management classes to demonstrate the bullwhip effect and the importance of supply chain coordination.

The serial supply chain network consists of various stakeholders, including agents, retailers, warehouses, distributors, and manufacturers. These entities make replenishment decisions independently based on limited information. The goal is to minimize the overall cost of the network by selecting optimal order quantities. Each agent only has access to local information until the game concludes.

In the beer game, it is a fundamental rule that agents are not allowed to communicate or share any local inventory statistics or cost information with each other until the game concludes. At that point, all agents are informed of the overall system-wide cost.

The retailer receives varied demand from its customers, while the manufacturer has an abundant supply. The transportation of products between different stages is bound by predetermined lead times, but the actual duration can be unpredictable due to upstream stockouts.; In addition, there are predetermined durations for the transfer of information from downstream to upstream in terms of replenishment orders. The beer game assumes that agents experience costs associated with holding inventory and stockouts but does not account for fixed ordering expenses.

In each phase of the game, every agent selects a quantity q to send to its preceding supplier in an effort to minimize overall system costs over time.

$$\sum_{t=1}^{T}\sum_{i=1}^{4} c_h^i (IL_t^i)^+ + c_p^i (IL_t^i)^-$$

T: The time horizon of the game, representing the number of periods over which the game is played.

t: The index of time periods, ranging from 1 to T.

i: The index of agents (retailer and manufacturer)

c_h^i: The holding cost coefficient of agent i (This cost represents the expenses incurred for holding inventory in a given period)

C_p^i: The stockout costs coefficient of agent i.)This cost represents the penalties incurred for unmet demands (backorders) owed to customers in a given period(

IL_t^i: The inventory level of agent i in period t (If IL_t^i > 0, the agent has inventory on-hand; if IL_t^i< 0, the agent has backorders)

Base-stock policy is a replenishment strategy used in supply chain management. It involves maintaining a fixed inventory level, known as the base stock level, and ordering additional units whenever the stock falls below this level.

The base stock policy is commonly viewed as the best approach in certain situations, particularly when there is a consistent demand process and stationary costs, and backorder costs are only incurred during specific stages of the supply chain. One method for determining the optimal base stock levels is through algorithms like the one introduced by Clark and Scarf. Under this approach, each stage places an order for a quantity that will bring its inventory position (on-hand + on-order inventory - backorders) to match a predetermined number called its base stock level.

There is currently no established method for determining the most efficient base-stock levels in cases where stockout costs can vary. Additionally, when dealing with decentralized supply chains, the behavior of individual agents may be irrational or unpredictable. When some agents do not abide by a specific policy, it becomes difficult to predict the optimal approach for the remaining players. In studies such as the beer game, it has been observed that relying solely on a base-stock policy can prove suboptimal when other agents utilize more realistic models based on human ordering behavior.

The problem defined in this paper is figuring out an optimal strategy to address this issue without requiring knowledge of the demand probability distribution and uses only historical data.

The SRDQN algorithm is a deep reinforcement learning algorithm that utilizes deep Q-networks designed for addressing this problem.The algorithm is called shaped-reward DQN (SRDQN) due to its emphasis on reward shaping. It utilizes deep reinforcement learning (RL) to optimize inventory decisions in the game. The algorithm does not make any assumptions about costs or other settings and can be trained to execute in real-time. It has been shown to perform near-optimally when playing with agents who follow a base-stock policy and significantly

outperforms a base-stock policy when other agents use a more realistic model of human ordering behavior.

Transfer learning is applied to adapt the training performed for one agent quickly to other agents and settings, reducing the training time. To speed up the training process for new agents with different cost coefficients or action spaces, a transfer learning method is introduced, which reduces the need to tune hyperparameters and also reduces the number of trainable variables.

Although there are no theoretical guarantees of reaching global optimality with this non-linear approximation, it has been proved to be effective, as it results in near-optimal game costs and speeds up the time it takes to train new agents.The proposed algorithm can be used to improve decision-making in supply chain networks, especially when supply chain partners act irrationally or unpredictably.

2. Literature Survey

2.1 current state of art

This study aims to explore the optimization of agent behavior in the Beer Game, a widely recognized simulation used for illustrating complexities in supply chain management.. The base-stock policy is a rule for managing inventory that stems from the Beer Game's supply chain model. According to the document, under conditions dictated by the game, a base-stock policy is optimal at each stage. If the demand process and costs are stationary, optimal base-stock levels also become stationary. This means that in each period (except the first one), each stage orders from its supplier exactly the amount that was demanded from it. When customer demands are independently and identically distributed (i.i.d.) random variables, and if backorder costs, which refers to the context of inventory management and, by extension, the Beer Game, refer to the costs incurred when a company cannot immediately fill an order with available inventory and must instead extend the delivery timeline, the optimal base-stock levels can be determined using an algorithm(clarck 1960) by Clark and Scarf (1960) :

Its main highlight is the computation of optimal base-stock levels in a multi-stage, multi-period inventory system under specific conditions. The algorithm processes from downstream to upstream stages, solving smaller deterministic dynamic programming subproblems at each stage. It takes advantage of the so-called ‘echelon-stock’ concept, leading to a significant reduction in the complexity of the problem. Let's look at the Clark and Scarf decomposition for

a four-stage inventory system. The principle remains the same, but the number of stages increases the complexity.

The actual calculation of the optimal base-stock levels requires knowledge of the demand distribution and other system parameters. However, the above framework provides a structured approach to decompose the problem and find the optimal solution.

It's important to note that the complexity increases as more stages are added due to the backward recursion. This can make it computationally challenging for very large systems, but the principle of the method remains consistent.

There is substantial literature on the beer game and the bullwhip effect. We review some of that literature here, considering both independent learners (ILs) and joint action learners (JALs), (Claus and Boutilier 1998). ILs and JALs refer to Independent Learners and Joint Action Learners, respectively.

1. Independent Learners (ILs) - These are agents who have no information about the actions of other agents. They make decisions independently based on their own observations and experiences, without considering the implications of their actions on the overall system or the actions of the other agents.

2. Joint Action Learners (JALs) - These are agents who may share some information and make collective decisions. Unlike ILs, JALs consider the actions of other agents and try to coordinate their actions with the rest. These classifications might be rooted in reinforcement learning theory, where agents can either learn independently or in a coordinated fashion. (Claus and Boutilier 1998).

In the beer game, the choice between acting as an IL or a JAL significantly impacts how effectively an agent can respond to changes in the environment (e.g., fluctuating demands, backorders, etc.) and optimize its objective, such as minimizing costs or maximizing efficiency in the supply chain.

The Sterman formula, as proposed by Sterman in 1989, is a mechanism used to determine the order quantity in a supply chain setting(like the beer game).

Sterman formula parameters:

The computational experiments that use Strm agents calculate the order quantity using $qti = max\{0, AOt + 1i - 1 + \alpha i(ILti - ai) + \beta i(OOti - bi)\}$, where α , a , β , and b are the parameters corresponding to the inventory level and on-order quantity. The idea is that the agent sets the order quantity equal to the demand forecast plus two terms that represent adjustments that the agent makes based on the deviations between its current inventory level (resp., on-order quantity) and a target value a i (resp., b i). We set ai = μd, where μd is the average demand; bi =μd(lifi + litr); αi = −0.5; and βi = −0.2 for all agents i = 1,

2, 3, 4. The negative α and β mean that the player over-orders when the inventory level or on-order quantity fall below the target value a or b .

This formula bases the order quantity on factors like the current backlog of orders, available inventory, incoming and outgoing shipments, incoming orders, and expected demand. Sterman's formula is designed to model typical human behavior in response to observed situations in a supply chain, such as shortages or excess inventory. It tries to capture how human players might overreact or underreact to these circumstances. Notably, the Sterman formula does not seek to optimize order quantities but rather reflects the human players' typical decision-making.

There are some flaws and difficulties that arise when agents don't behave optimally, for example players do not tend to follow such a policy, or any policy, often they behave quite irrational. there isn't much literature or theoretical proof for policy optimality within these scenarios. The different approaches taken in the literature for both JALs and ILs are touched upon as well.

Kimbrough et al. (2002) proposed a model in the context of the Beer Game, which is based on Joint Action Learners (JALs). In their approach, they employed what's referred to as the d + x rule. In this model, an agent notes the received demand/order in a particular period, denoted as dit. They then choose an additional quantity, xit. The agent thus places an order of size ait which equals the sum of dit and xit. Basically, this model emphasizes the role of shared information in agent decision-making and how the collective decisions of agents can influence the outcome of the Beer Game.

In the literature on multi-echelon inventory problems, there are only a few papers that use RL. Among them, Jiang and Sheng (2009) proposed a reinforcement learning (RL) model for a two-echelon serial system in multi-echelon inventory problems. In this system, the retailer has to optimize either "r, Q" (reorder point, order quantity) or "T, S" (time period, order-up-to level) policy parameters. However, the model primarily considers the decision of retailers and as such, it isn't genuinely multi-echelon. The model defines the state as a combination of both the independent learner (IL) and the demand component. To avoid the issue known as "the curse of dimensionality", where the complexity of a problem exponentially increases with each additional dimension, they resort to discretizing and truncating the state and action values, effective techniques to manage high-dimensional state-action spaces in RL by quantizing them into a manageable number of categories or limit their possible range.

Another tabular RL algorithm which is more comprehensive is Giannoccaro and Pontrandolfo (2002).Giannoccaro and Pontrandolfo (2002) proposed a reinforcement learning solution for the beer game with three agents and stochastic factors. Their model defines the state as three inventory positions, discretized into ten intervals, with actions ranging from 0 to 30. The

model assumes real-time information sharing and centralized decision-making, which diverge from the usual independent nature of decisions in the beer game.

The main critique of Giannoccaro and Pontrandolfo's algorithm is its oversimplification. It incorrectly presumes that information is shared and decisions are made centrally, while in reality, agents usually function independently. Additionally, by discretizing inventory positions, it disregards intricate details, potentially discarding valuable data that could lead to more accurate solutions.

In general, prior research has suggested the use of traditional tabular RL algorithms with simplified states and actions or proposed a centralized decision-making framework, which is not in line with the usual Beer Game scenario. Despite their contributions, these methods tend to oversimplify the state space or lose important state information, affecting the precision of the solution. Moreover, most Beer Game literature assumes that all agents have access to shared information (JALs), but in real-world situations, the independent learner (IL) model is more common, where each agent only has local information. This discrepancy in literature introduces a more complex and challenging situation for Reinforcement Learning (RL) algorithms.

Chaharsooghi et al. (2008) examined the same game and solution method, but with four agents and a set game length of 35 periods. In their RL proposal, the state variable consists of the four inventory positions, each of which is discretized into nine intervals. Furthermore, their RL algorithm employs the d + x rule to calculate the order quantity, with x limited to values in {0, 1, 2, 3}; But it still had this problem.

While traditional supervised machine learning algorithms could be considered to address this issue, they are not directly applicable to the beer game due to the absence of historical data in the form of "correct" input/output pairs. Consequently, standalone support vector machines or deep neural networks cannot be trained with a dataset to learn the optimal action, as done by Oroojlooyjadid et al. (2017, 2020), Ban and Rudin (2019), Ban et al. (2019) to solve simpler supply chain problems like newsvendor and dynamic procurement. There is a significant gap between the current algorithms' capabilities and the effective solution of the beer game problem, as per our literature understanding. To bridge this gap, we suggest a variant of the DQN algorithm, SRDQN, for selecting order quantities in the beer game.

2.2. Reinforcement Learning

Reinforcement Learning (RL), as defined by Sutton and Barto (1998),is a type of machine learning where an agent learns how to behave in an environment by performing certain actions and receiving rewards in return to maximize a cumulative reward.

The Algorithm:

At each step t :

1. The agent observes the state s_t

2. Chooses an action a_t

3. Receives a reward r_t

4. The system randomly transitions to a new state s_{t+1}

5. The agent updates its knowledge.

This sequence is termed a Markov Decision Process (MDP).

The transition probability matrix $P_a\left(s, s^{'}\right)$ and the reward matrix $R_a\left(s, s^{'}\right)$ is the likelihood of transitioning to another state and the rewards to be received for such a transition respectively.

The ultimate goal in RL is finding optimal policy that maximizes expected discounted sum of the rewards, $G_t = E\left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}\left(s_t, s_{t+1}\right)\right]$ when the systems runs for an infinite horizon.

γ ($0 \leq \gamma \leq 1$) is the discount factor which determines the value of future rewards.

Policy (π) is a strategy that the agent follows while interacting with the environment. It is basically a mapping that determines what action the agent should choose given a particular state.

Optimal policy (π^*) is best possible policy that the agent can follow to interact with the environment.

In each state, the optimal policy dictates the action that the agent should take to achieve maximum reward return.

In the process of learning, an agent may start with a random or specified policy, then through exploration and exploitation – and based on feedback from the environment in the form of rewards or punishment – the agent adjusts its policy.

For given $P_a\left(s, s^{'}\right)$ and $R_a\left(s, s^{'}\right)$, the optimal policy can be obtained through dynamic programming or linear programming (Sutton and Barto 1998).

Q-learning is a value-based (meaning it estimates the value (Quality) of each action in each state), model-free (meaning allows the agent to learn the optimal policy without requiring a model of the environment), reinforcement learning algorithm that aims to discover an optimal policy π* for maximizing cumulative rewards.

It seeks to find the best action to take by learning an action-value function for a given finite Markov decision process ,i.e. guides the agent a way to estimate the expected utility of taking an action in a given state.

It's utilized useful when the reward function is uncertain or the environment is subject to change.

Q-Learning approaches the problem by establishing a table of Q-values for each possible state-action pair, which serve as approximations of their expected future rewards.This table, often referred to as the Q-table, guides the agent to the best action to take given a specific state.

In this algorithm the agent learns the optimal policy regardless of the policy being followed.

The Algorithm:

1.Initialize a Q-value table for all possible state-action pair (s, a) has a designated Q-value, typically initialized to 0 but can be arbitrary.

2. For a finite number of episodes, the agent perceives the state s,chooses an action a based on ε -greedy approach (This might be exploration or exploitation), receives an immediate reward r, and arrives in a new state s_{t+1},then updates the Q-value for the taken action using the update rule iteratively:

$$Q\left(s_t, a_t\right) = (1 - \alpha)Q\left(s_t, a_t\right) + \alpha\left[r_t + \gamma Q\left(s_{t+1}, a\right)\right]$$

Which:

α: learning rate (how much weight to give the new information)

γ: discount factor.It determines the importance of future rewards compared to immediate rewards.

r_t: the observed reward

$Q\left(s_{t+1}, a\right)$: the maximum reward that can be obtained from state s_{t+1}

3. Return Optimal Policy: The optimal policy derived from the final Q-table is the action with the highest Q-value at every state. The agent following this policy is expected to achieve the most long-term reward. By repeating actions in each state and updating Q-values based on rewards, Q-learning allows the agent to learn the value of each action in each state, leading to an optimal policy.

Q-learning is start to converge to the optimal Q-values denoted Q*(s,a). Meaning it will eventually learn to make the best possible decisions in each state.

The optimal policy π*(s), is obtained by selecting the action that yields the highest Q-value for each state s:

$$\pi^{*}(s) = \arg arg\, Q^{*}(s, a)$$

ε**-greedy algorithm** guarantee the convergence to the optimal policy.

The principle of the approach is to balance between exploration, where an agent takes random actions to discover potentially optimal choices, and exploitation, where the agent makes decisions based on existing knowledge to maximize its rewards over the time.

The ratio of exploration versus exploitation is influenced by ε ,a value between 0 and 1.

An ε value closer to 1 implies that the agent will prioritize exploration, while a value closer to 0 means it's more likely to exploit its current knowledge.

The ε value starts high to encourage the agent learns more about the environment. The agent will choose an random action from the set of possible actions.It gives the agent a chance to explore all states sufficiently to correctly estimate the rewards of all state-action pairs.

Decreasing ε over time in later stages of learning lets the agent use its learned knowledge as the agent has a good understanding of the environment to take most estimated rewarding actions more frequently.

When ε is very small, the agent has learned the environment well and constantly chooses the action with the highest Q-value in each state. If you continue this process of decreasing ε while allowing the agent to learn from its experiences, the agent will eventually learn the optimal policy ($\pi*$).

Dynamic programming and Q-learning are useful to obtain optimal policies solving Markov Decision Processes (MDPs) .These methods struggle to effectively address MDPs featuring large number of possible states or decisions/actions. In the Beer game the full state variable cannot be observed by the decision-maker. This is known as partially observed MDP (POMDP), causing the problems to be much more complex.As the number of possible actions grows, the Q-table becomes massive. This is where the curse of dimensionality comes into play. With each new action added, the size of the table grows exponentially, making it increasingly difficult and computationally expensive to update and maintain.

To address this issue, Q-learning employs approximation of Q-values. While linear regression is a common method for this approximation, it does not provide sufficient accuracy.

Non-linear functions and neural networks can provide more accurate approximations of Q-values.However these methods are known for their potential instability or divergence due to non-stationarity and correlations in the sequence of observations. Instability refers to the

output values varying too much, while divergence refers to the Q-values progressively drifting away from the optimal solution.

Mnih et al(2015) addressed these challenges by proposing a deep Q-network algorithm, which employs a deep neural network to approximate the Q-function and trains it using iterations of the Q-learning algorithm.DQN uses target networks and experience replay memory.

A Deep Q-Network (DQN) is a combination of reinforcement learning (Q-Learning) and deep learning (Neural Networks). The goal of a DQN is to estimate the Q-values, which represent the expected future rewards for a particular action taken in a particular state, in order to guide policy decision in a reinforcement learning problem. A DQN with target networks and experience replay memory enhances the stability of the original DQN algorithm.

The Algorithm:

The DQN algorithm uses a neural network to approximate the best action within a given state. This algorithm essentially learns to predict the expected maximum future reward of a current action, given a state.

Setup: The setup phase involves creating an experience replay buffer (which has the agent's previous game states, actions, rewards, and new states), an acting neural network (which predicts the future reward for each possible action given the current state), and a target network (a duplicate of the acting network updated less frequently to improve stability).

1. Experience Replay Buffer: This is a data structure that stores the agent's past experiences. Each experience, or episode, is a sequence of states, actions, and rewards. The first state is where an action is taken, a reward is received, and a new state is entered. This set is stored in the buffer, allowing the agent to later replay and learn from these experiences. This method breaks the correlation of sequential actions, thereby stabilizing learning by providing a diverse range of experiences for training.
2. Acting Neural Network: This is the main network used by the agent during interaction with the environment. It takes the current state as input and outputs the expected future reward for each possible action the agent can take. By choosing the action with the highest predicted future reward, the acting network effectively guides the agent's behavior.
3. Target Network: This is a duplicate of the acting network, but its update frequency is less. The target network mitigates the risk of drastic shifts in values that could occur if you constantly update a single network. By keeping its weights frozen for a certain number of steps ,it provides a kind of moving target for training the acting network. After the defined steps, the weights of the target network are updated with the weights

from the acting network. This is done to maintain stability and improve convergence performance during learning

In essence, the setup phase of a DQN creates the necessary infrastructure for the agent interact with the environment and learn from these interactions.

Acting Phase: Here, the agent interacts with the environment. It passes the current state through the neural network, selects an action (based on an epsilon-greedy approach for exploration and exploitation balance), receives a reward and a new state from the environment, and stores this data in the experience replay buffer.

1. An observation, or the current state of the environment at time t, is passed as an input to the acting neural network.
2. The neural network then produces an expected future reward for each possible action the agent can take based on the current state.
3. Then, an action is selected. The process for this selection is an epsilon-greedy exploration strategy, which aids in maintaining a balance between exploration (trying out new, possibly beneficial actions) and exploitation (choosing the action that presently seems best). With a certain probability epsilon, the agent selects a random action. The rest of the time, it selects the action with the highest expected future reward as predicted by the acting neural network.
4. The chosen action is performed, and the agent receives a reward and the new state (t+1) from the environment.
5. Finally, the data containing the initial observation at time t, the action taken, the reward received, whether the episode has terminated or not, and the new observation at time t+1 are all stored in the experience replay buffer. This data will later serve in the learning phase, where the agent uses its past experiences to improve its strategy or 'policy.'

Learning Phase: in learning phase the agent learns and improves its strategy using data stored in the experience replay buffer.The network learns by randomly sampling batches from the experience replay buffer. It computes the difference between the predicted future reward and the actual reward obtained. The Q-network parameters are updated to minimize this difference using gradient descent. Every C steps, the target network updates its parameters to match the Q-network's.Throughout this process, the aim is to minimize prediction error (the difference between the estimated reward and the actual reward) to improve the accuracy of future predictions and hence the quality of the chosen action. The prime goal of the learning phase is to minimize the prediction error. By doing so, the agent grows competent at estimating the future rewards for an action more accurately, thus improving its decision-making in the game.

1. A random sample of items (typically 32 instances) are selected from the replay buffer. This step allows the model to learn from a variety of past experiences, reducing the risk of overfitting or concentrating too heavily on recent experiences.
2. Then, the learning target for each instance in the sample is set as the sum of the actual reward obtained and the predicted future reward, estimated by the target network using the new observation at time t+1. The target network's slower update rate provides a more stable and less biased target.
3. The agent then calculates its prediction error, i.e., the difference between the learning target (actual plus estimated future rewards) and its own estimation of total reward for the chosen action at time t.
4. This error, quantifies as the 'loss function', is then minimized by updating the Q-network parameters using a process called 'gradient descent'. This process adjusts the parameters incrementally to reduce the error.

Every C steps ,the parameters of the target network are updated to match those of the Q-network. This helps maintain stability throughout the learning process.

.

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 ϵ 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 $y_j = \begin{cases} r_j & \text{for terminal } \varphi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

While DQN demonstrates practical success, there is no theoretical proof of convergence due to the non-convex nature of deep neural networks and the use of the non-smooth argmax

function in DQN. Furthermore, there is currently limited understanding regarding how target networks and replay buffer memory impact sample distribution and gradients during training.

Also,The Beer Game displays multiple unique characteristics unlike traditional settings of DQN applications. DQN is designed for single- agent, competitive, zero-sum games and usually assumes that the agent fully observes the state of the environment at any time step t of the game while the Beer game is a multi-agent, decentralized,cooperative, non-zero-sum game and the agents are playing independently and do not have any information from other agents until the game ends and the total cost is revealed without sharing information about rewards. DQN provides actions for an agent that interacts with an environment in a competitive setting but the Beer game involves cooperative actions from multiple decentralized agents in the sense that all of the players aim to achieve a single objective which is minimizing the total cost of the system in a random number of periods. This decision-making framework is known as Decentralized Partially Observable Markov Decision Process (Dec-POMDP).

The partial observability and the non-stationarity of the agent's observations require high complexity in computational logic.Applying DQN in a straightforward way to a given agent would result in a policy that optimizes that agent locally but will not result in an optimal solution for the supply chain as a whole.

The SRDQN algorithm proposed in this research addresses the limitations of directly applying DQN to the beer game.

3. Methodology

The general structure of SRDQN is based on the DQN algorithm (Mnih et al. 2015) which applies Q-learning to optimize the beer game using Deep Neural Networks for approximating the Q-function but an adaptation has been developed.

In the SRDQN approach ,the agents still play independently from one another, but in the training phase, they use reward shaping via a feedback scheme so that the SRDQN agent learns the total cost for the whole network and can, over time,learn to minimize it. Thus, the SRDQN agent in their model plays smartly in all periods of the game to get a near-optimal cumulative cost for any random horizon length.

Each agent operates based on its local information and treats other agents as part of its environment. This means that the RL agent has no knowledge about static parameters such as costs and lead times and dynamic state variables like inventory levels.

The goal is to learn an effective policy that minimizes the overall cost of the game by implementing a feedback scheme to guide the RL agent in prioritizing the minimization of system-wide costs rather than individual local costs.

Currently, they have focused their efforts on designing and testing their approach for a single SRDQN agent whose teammates consist of simple formulas or human players But the framework has the potential to be applied to multiple SRDQN agents playing the beer game as a team.Enhancing the algorithm to train multiple SRDQN agents simultaneously and cooperatively is an area of ongoing research.Solving multi-agent RL problems is significantly more challenging than single-agent RL. Note that multi-agent RL is significantly harder to solve than single-agent RL problems. This is due to the fact that the policy of each agent changes throughout the training, and as a result, the environment becomes non-stationary from the viewpoint of each individual agent. Therefore, specialized algorithms and techniques are required to address this issue effectively.

It is possible to achieve optimal performance by tuning and training a new network but this approach can be time-consuming. The transfer learning approach is employed to address this challenge. This involves transferring the knowledge acquired by one agent under specific game parameters to another agent with different game parameters. As a result, the training time for new agents was significantly reduced.

In summary, SRDQN is a variant of the DQN algorithm that effectively selects actions in the beer game. To achieve near-optimal cooperative solutions, we have developed a feedback scheme as our communication framework. Lastly, for simplified training of agents with new settings, we leverage transfer learning to efficiently incorporate previously learned knowledge from trained agents

3.1. Shaped-Reward Deep Q-Network

The Shaped-reward Deep Q-Network (SR-DQN) algorithm extends the DQN algorithm by incorporating additional information about the environment through a shaping reward function to improve the learning process's efficiency, leading to a quicker convergence and more optimal policy.

The Algorithm:

1. Initialization: Initialize the Deep Neural Network (DNN) with random weights and set the shaping reward function based on prior knowledge about the environment.

The shaping reward function injects prior knowledge about the environment into the reward signal. It is designed to guide the agent to areas in the state-action space that are more likely to yield higher rewards, resulting in more efficient learning.

2. Observation: Observe the current state 's' and choose an action 'a' using an ε-greedy strategy based on the Q-value (expected future reward) predicted by the DNN.

3. Execution and Reward Observation: Execute the selected action 'a' in the environment, observe the immediate reward, and the shaped reward .Instead of simply taking this immediate reward, in SR-DQN, the agent applies the shaping reward function to it. This function augments the immediate reward from the environment based on additional criteria or information to create a "shaped reward". The shaped reward gives the agent extra information about the desirability of its actions, guiding the learning process more effectively.

4. Transition: Transition to the new state after action is executed.

5. Q-value Calculation: Calculate the Q-value for the new state-action pair using the reward 'rs' plus the maximum predicted Q-value of the new state 's' discounted by the factor γ, where a' is the next action, and θ represents the current weights of the DNN. 6. Network Updating: Use backpropagation to update the DNN weights, minimizing the difference between the target Q-value 'y' and the predicted Q-value.

7. Repeat: Repeat steps 2-6 for each new state until the episode ends.

8. Target Network Update: Every C episodes, update the DNN used for calculating the target Q-value y by copying the weights from the DNN being trained. The network's weights are updated regularly using backpropagation, reducing the difference between the predicted Q-value and the observed reward plus the discounted estimated future reward of the next state.

9. Iteration: Repeat steps 2-8 for a specified number of episodes or until the Q-function converges. In summary, the SR-DQN algorithm works similarly to a standard DQN but incorporates an additional shaping reward function to effectively guide the learning process, hence leading to a quicker convergence and more optimal policy.

Algorithm 1 SRDQN for Beer Game

procedure SRDQN
  Initialize Experience Replay Memory $E_i = [\,]$, $\forall i$
  for $Episode = 1 : n$ **do**
    Reset IL, OO, d, AO, and AS for each agent
    for $t = 1 : T$ **do**
      for $i = 1 : 4$ **do**
        With probability ϵ take random action a_t,
        otherwise set $a_t = \underset{a}{\operatorname{argmin}}\, Q(s_t, a; \theta)$
        Execute action a_t, observe reward r_t and state s_{t+1}
        Add $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ into E_i
        Get a mini-batch of experiences (s_j, a_j, r_j, s_{j+1}) from E_i
        Set $y_j = \begin{cases} r_j & \text{if it is the terminal state} \\ r_j + \min\limits_{a} Q(s_{j+1}, a; \theta^-) & \text{otherwise} \end{cases}$
        Run forward and backward step on the DNN with loss function $(y_j - Q(s_j, a_j; \theta))^2$
        Every C iterations, set $\theta^- = \theta$
      end for
    end for
    Run feedback scheme, update experience replay of each agent
  end for
end procedure

3.2. Transfer Learning

Transfer learning has emerged as a successful field of research in machine learning, particularly in image processing. It involves utilizing a pre-trained neural network that has been trained on a source dataset S to perform specific tasks such as classification, regression, or decision-making through reinforcement learning. Training such networks can be time-consuming, spanning days or even weeks. However, for similar or slightly different target datasets T, it is possible to leverage the existing knowledge by customizing the pretrained network instead of training a new one from scratch with minimal additional training. This approach not only reduces the overall training time but also effectively utilizes the learned knowledge from dataset S.

To leverage transfer learning in the beer game, we can utilize a source agent i∈{1,2,3,4} that possesses a trained network S_i with fixed-sized parameters $P_1^i = \left\{ \left| A_1^j \right|, c_{p_1}^j, c_{h_1}^j \right\}$, observed

demand distribution D_1 , and co-player policy π_1. We will employ the weight matrix W_i which contains learned weights denoted as W_i^q between layers q and q+5 of the neural network. Here $q \in \{0, \ldots nh\}$, where nh represents the number of hidden layers. The objective is to train a neural network S_j for target agent $j \in \{1, 2, 3, 4, 5\}$, j≠i using this approach. We adopt a similar structure for network S_j as that of S_i and initialize W_j with W_i. The initial layers are set to be non-trainable, while the subsequent layers are trained using a small learning rate. It is worth noting that as we approach the final layer, which provides Q-values, the weights become less similar to those of agent i and more specific to each individual agent. Therefore, the knowledge acquired in the earlier hidden layers of agent i's neural network is transferred to agent j through this procedure. The value of k determines how many initial hidden layers are transferred and can be adjusted accordingly. By following this approach, we evaluate transfer learning in six different scenarios where we transfer learned knowledge from source agent i to target agent j:

$j \neq i$ in the same game.

[1] Our implementation uses the version of the Clark-Scarf algorithm presented by Chen and Zheng (1994), Gallego and Zipkin (1999), but we refer to it as the "Clark-Scarf" algorithm throughout. 2. $\left\{\left|A_1^j\right|, c_{p_2}^j, c_{h_2}^j\right\}$, i.e., the same action space but different cost coefficients.

$\left\{\left|A_2^j\right|, c_{p_1}^j, c_{h_1}^j\right\}$, i.e., the same cost coefficients but different action space.

$\left\{\left|A_2^j\right|, c_{p_2}^j, c_{h_2}^j\right\}$, i.e., different action space and cost coefficients.

$\left\{\left|A_2^j\right|, c_{p_2}^j, c_{h_2}^j\right\}$, i.e., different action space and cost coefficients, as well as a different demand distribution D_2.

$\left\{\left|A_2^j\right|, c_{p_2}^j, c_{h_2}^j\right\}$, i.e., different action space and cost coefficients, as well as a different demand distribution D_2 and co-player policy π_2.

In addition to the aforementioned benefits, transfer learning can also be applied when other aspects of the problem change, such as lead times and state representation. This helps avoid the need for tuning neural network parameters for each new problem, resulting in significant reduction in training time. However, it is important to determine which agent can serve as a base agent for transferring learned knowledge and decide on the number of trainable layers.

Overall, this approach offers computational efficiency compared to building networks from scratch with their associated hyperparameters.

4. Results

In order to validate our algorithm, we compare the results of SRDQN to those obtained using the optimal base-stock levels (when possible) by Clark and Scarf (1960),1 as well as models of human beer-game behavior by Sterman (1989)

The SRDQN (state-based Deep Q-Network) was successful in handling the problem posed by the beer game.

- When playing with teammates who follow a base-stock policy, the algorithm achieves near-optimal order quantities.
- It performs much better than a base-stock policy when other agents use a more realistic model of human ordering behavior.
- The model's training might be slow, but execution is very fast.
- Demand probability distribution knowledge isn't required - it uses only historical data.

Transfer Learning method is employed to reduce the computation time needed to train new agents with different cost coefficients or action spaces. This method is faster because it bypasses hyper-parameter tuning and has fewer trainable variables. Transfer learning remains effective even when these aspects differ between agents, demonstrating agility in adapting to new environments.

- It reduces computation time required for training new agents with different cost coefficients or action spaces.
- It avoids the need to tune hyper-parameters and has fewer trainable variables, speeding up the training process.
- It obtains beer game costs close to, or better than, those of a base-stock policy.
- In different cost and action space settings, transfer learning is still effective—reserving a 12.58% gap compared to a base-stock policy result.

Sensitivity analysis indicates that a trained model is robust to changes in the cost coefficients. Additionally, applying transfer learning significantly reduces the training time.

In summary, the suggested method showcases impressive results when adapting and training the SRDQN for a specific agent and varying game parameters. Our analysis demonstrates that even with alterations to these parameters, a trained model remains resilient and performs admirably in slightly different scenarios.

The use of base-stock policies can lead to comparable performance as the optimal policy when utilizing the SRDQN agent. This achievement is made possible without relying on information about other players' strategies or cost coefficients, which are assumptions in algorithms like Clark and Scarf. Instead, the SRDQN agent learns from its own local information and through experimentation. Similar results are observed with random co-players, indicating that a base-stock policy may be optimal even if we do not know the exact details of the optimal policy in this particular scenario.

The SRDQN agent, when used with Sterman co-players, demonstrates superior performance compared to a base-stock policy, even when the base-stock level is optimized. Existing literature has not explored the optimal policy with Sterman co-players comprehensively. Based on our findings, it appears that a base-stock policy is suboptimal in this scenario..

inventory level (IL),on-order quantity (OO), order quantity (a), reward (r),order up to level (OUTL)

*plots are saved in a pdf.

References

Oroojlooyjadid, A., Nazari, M., Snyder, L., & Takáč, M. (2017). A Deep Q-Network for the Beer Game: A Deep Reinforcement Learning algorithm to Solve Inventory Optimization Problems (Version 4)

Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning (Version 1)

G.-Y. Ban and C. Rudin. The big data newsvendor: Practical insights from machine learning. Operations Research, 67(1):90–108, 2019.

G.-Y. Ban, J. Gallien, and A. J. Mersereau. Dynamic procurement of new products with covariate infor- mation: The residual tree method. Manufacturing & Service Operations Management, 21(4):798–815, 2019.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. Mathematics of operations research, 27(4):819–840, 2002.

S. K. Chaharsooghi, J. Heydari, and S. H. Zegordi. A reinforcement learning model for supply chain ordering management: An application to the beer game. Decision Support Systems, 45(4):949–959, 2008.

D. Chan. The AI that has nothing to learn from humans. The Atlantic, October 20 2017.
F. Chen and R. Samroengraja. The stationary beer game. Production and Operations Management, 9(1): 19, 2000.

F. Chen and Y. Zheng. Lower bounds for multi-echelon stochastic inventory systems. Management Science, 40:1426–1443, 1994.

A. J. Clark and H. Scarf. Optimal policies for a multi-echelon inventory problem. Management science, 6 (4):475–490, 1960.

C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. AAAI/IAAI, 1998:746–752, 1998.

R. Croson and K. Donohue. Behavioral causes of the bullwhip effect and the observed value of inventory information. Management science, 52(3):323–336, 2006.

G. Gallego and P. Zipkin. Stock positioning and performance estimation in serial production-transportation systems. Manufacturing & Service Operations Management, 1:77–88, 1999.

S. Geary, S. M. Disney, and D. R. Towill. On bullwhip in supply chains—historical review, present practice and expected future impact. International Journal of Production Economics, 101(1):2–18, 2006.

I. Giannoccaro and P. Pontrandolfo. Inventory management in supply chains: A reinforcement learning approach. International Journal of Production Economics, 78(2):153 – 161, 2002. ISSN 0925-5273.

J. Gijsbrechts, R. N. Boute, J. A. Van Mieghem, and D. Zhang. Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. Available at SSRN: https://ssrn.com/abstract=3302881, 2019.

S. C. Graves. A multi-echelon inventory model for a repairable item with one-for-one replenishment. Man- agement Science, 31(10):1247–1256, 1985.

C. Jiang and Z. Sheng. Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. Expert Systems with Applications, 36(3):6520–6526, 2009.

Kaggle.com. Store item demand forecasting challenge. https://www.kaggle.com/c/demand-forecasting-kernels-only/overview, 2018. Accessed: 2019-08-26.

S. O. Kimbrough, D.-J. Wu, and F. Zhong. Computers play the beer game: Can artificial agents manage supply chains? Decision support systems, 33(3):323–333, 2002.

H. L. Lee, V. Padmanabhan, and S. Whang. Information distortion in a supply chain: The bullwhip effect. Management Science, 43(4):546–558, 1997.

H. L. Lee, V. Padmanabhan, and S. Whang. Comments on “Information distortion in a supply chain: The bullwhip effect”. Management Science, 50(12S):1887–1893, 2004.

Y. Li. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274, 2017.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine

Learning, 8(3-4):293–321, 1992.

I. J. Martinez-Moyano, J. Rahn, and R. Spencer. The Beer Game: Its History and Rule Changes. Technical

report, University at Albany, 2014.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asyn- chronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937, 2016.

A. OroojlooyJadid and D. Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. arXiv preprint arXiv:1908.03963, 2019.

A. Oroojlooyjadid, L. Snyder, and M. Taka ́ˇc. Stock-out prediction in multi-echelon networks. arXiv preprint arXiv:1709.06922, 2017.

A. Oroojlooyjadid, L. V. Snyder, and M. Tak ́aˇc. Applying deep learning to the newsvendor problem. IISE Transactions, pages 444–463, 2020.

S. J. Pan and Q. Yang. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2010.

Pentaho. Foodmart’s database tables. http://pentaho.dlpage.phi-integration.com/mondrian/mysql-foodmart-database, 2008. Accessed: 2015-09-30.

P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225, 2017.

L. V. Snyder. Multi-echelon base-stock optimization with upstream stockout costs. Technical report, Lehigh University, 2018.

L. V. Snyder and Z.-J. M. Shen. Fundamentals of Supply Chain Theory. John Wiley & Sons, 2nd edition, 2019.

J. D. Sterman. Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment. Management Science, 35(3):321–339, 1989.

F. Strozzi, J. Bosch, and J. Zaldivar. Beer game order policy optimization under changing customer demand. Decision Support Systems, 42(4):2153–2163, 2007.

R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. MIT Press, Cambridge, 1998.

Z. Yang, Y. Xie, and Z. Wang. A theoretical analysis of deep q-learning. arXiv preprint arXiv:1901.00137,

2019.

K. Zhang, Z. Yang, and T. Ba ̧sar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. arXiv preprint arXiv:1911.10635, 2019.