TD-GAMMON REVISITED: INTEGRATING INVALID ACTIONS AND DICE FACTOR IN CONTINUOUS ACTION AND OBSERVATION SPACE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ENGİN DENİZ USTA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2018

Approval of the thesis:

TD-GAMMON REVISITED: INTEGRATING INVALID ACTIONS AND DICE FACTOR IN CONTINUOUS ACTION AND OBSERVATION SPACE

submitted by **ENGİN DENİZ USTA** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halit Oğuztüzün _____
Head of Department, **Computer Engineering**

Prof. Dr. Ferda Nur Alpaslan _____
Supervisor, **Computer Engineering, METU**

Examining Committee Members:

Prof. Dr. Pınar Karagöz
Computer Engineering, METU _____

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering, METU _____

Assist. Prof. Dr. Kasım Öztoprak
Computer Engineering, Karatay University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Engin Deniz Usta

Signature :

ABSTRACT

TD-GAMMON REVISITED: INTEGRATING INVALID ACTIONS AND DICE FACTOR IN CONTINUOUS ACTION AND OBSERVATION SPACE

Usta, Engin Deniz
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Ferda Nur Alpaslan

September 2018, 39 pages

After TD-Gammon's success in 1991, the interest in game-playing agents has risen significantly. With the developments in Deep Learning and emulations for older games have been created, human-level control for Atari games has been achieved and Deep Reinforcement Learning has proven itself to be a success. However, the ancestor of DRL, TD-Gammon, and its game Backgammon got out of sight, because of the fact that Backgammon's actions are much more complex than other games (most of the Atari games has 2 or 4 different actions), the huge action space has much invalid actions, and there is a dice factor which involves stochasticity. Last but not least, the professional level in Backgammon has been achieved a long time ago. In this thesis, the latest methods in DRL will be tested against its ancestor game, Backgammon, while trying to teach how to select valid moves and considering the dice factor.

Keywords: TDGammon, Deep Deterministic Policy Gradients, OpenAI-GYM, GYM-

Backgammon, Continuous Action Space

ÖZ

TD-GAMMON'A YENİDEN BAKIŞ: TAVLA'DA SÜREKLİ AKSİYON VE GÖZLEM ALANI İÇİNE GEÇERSİZ HAMLELERİ VE ZAR FAKTÖRÜNÜ DAHİL ETMEK

Usta, Engin Deniz
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ferda Nur Alpaslan

Eylül 2018 , 39 sayfa

TD-Gammon'un 1991'deki başarısından sonra, oyun oynayabilen etmenlere olan ilgi bir hayli artmış durumda. Derin Öğrenme ve eski oyunların emülatorlerindeki gelişmelerden sonra, Atari oyunları için insan seviyesinde oynayabilen etmenler ortaya çıktı, ve Derin Takviyeli Öğrenme kendi başarısını kanıtladı. Ancak, Derin Takviyeli Öğrenme'nin atası olan TD-Gammon, ve oyunu Tavla, arka planda kaldı. Bunun sebepleri ise, Tavla'nın aksiyonlarının diğer Atari oyunlarına gore çok daha kompleks olması (genelde çoğu Atari oyununda 2 veya 4 farklı aksiyon alınabilir), aksiyon alanında çok fazla geçersiz aksiyon olması, ve zar faktörünün getirdiği rastgelelik olarak gorülüyor. Son sebep olarak ise, Tavla'da uzun sure once profesyonel seviyede oynayabilen etmenlerin varlığı olduğunu söyleyebiliriz. Bu tezde, son çıkan Derin Takviyeli Öğrenme yöntemleri onların atası olan oyuna, Tavla'ya karşı test edilecektir. Bu sırada ek olarak, etmenlerimiz zar faktörünü de hesaba katarak geçerli hamleleri bulmaya çalışacaktır.

Anahtar Kelimeler: TD-Tavla, Derin Deterministik Poliçe Eğimi, OpenAI-GYM, GYM-Tavla, Sürekli Aksiyon Alanı

To my dearest family and my lovely Müge...

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor, Prof. Dr. Ferda Nur Alpaslan who always supported and kept my motivation high. Her extensive knowledge and intuitions greatly helped me through hardest times. It has been a pleasure being guided by her, and I hope I will have a chance to work with her again.

I would also like to thank Özgür Alan for his extensive data and motivation support.

Last but not least, I would like to thank Ömer Baykal for providing his knowledge about his prior experiences on NeuroGammon.

This work is supported and funded by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. 2228-A, 1059B281500849.

TABLE OF CONTENTS

LIST OF TABLES

.

TABLES

LIST OF FIGURES

.

FIGURES

LIST OF ABBREVIATIONS

RL	Reinforcement Learning
TD	Temporal Difference
TDGammon	Temporal Difference Backgammon
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
DDPG	Deep Deterministic Policy Gradients
DDPG-Gammon	Deep Deterministic Policy Gradients - Backgammon Agent
RNG	Random Number Generator
CRAS	Custom Reward and Action System

CHAPTER 1

INTRODUCTION

1.1 Reinforcement Learning in Continuous State and Action Spaces

Previous reinforcement-learning algorithms are created for small discrete action and observation spaces. On the other hand, most of the real-world problems have continuous action and/or observation spaces, which requires to find good decision policies within more complex situations [28].

Although Q-Learning is proven to be able to integrate with continuous action/observation spaces, Actor-Critic learning has an edge over Continuous-Q-Learning [4], because of its ability to respond fast-changing observations with fast-changing actions.

In order to tackle with continuous action/observation spaces, there are many algorithms proposed, such as Continuous Actor-Critic Learning Automaton [29], Gradient Temporal Difference Networks [18], Trust Region Policy Optimization [17], Deep Deterministic Policy Gradients [8].

Among all these methods, this thesis will focus on Deep Deterministic Policy Gradients (DDPG) [8], which is constructed upon Policy-Gradient and Actor-Critic algorithms.

1.2 Revisiting TD-Gammon: Factoring Invalid Actions and Dice

Although TD-Gammon's success is proven in 1992 World Cup Backgammon tournament [24], there are still room for improvement in TD-Gammon. Since TD-Gammon does not require any information regarding valid/invalid actions [25], we may ask the question: "Can an agent learn which action is valid/invalid in Backgammon?".

In order to be able to integrate every move in Backgammon, the dice factor must be integrated into the game knowledge, too. Without any filtering of information, the proposed agent, DDPG-Gammon, will be trained to separate the valid moves from invalid moves.

1.3 Contributions

This thesis proposes and focuses on a major novelty: GYM-Backgammon, suited from its ancestor, TD-Gammon. The most prominent feature of GYM-Backgammon is that it is not providing suitable actions from a set of actions. Instead, GYM-Backgammon is providing both valid and invalid actions, with the addition of dice factor. Additionally, there are 2 different GYM-Backgammon environments created, GYM-Backgammon-Discretized and GYM-Backgammon-Continuous, respectively.

To summarize, GYM-Backgammon-Discretized is converting the network outputs into backgammon actions directly. On the other hand, GYM-Backgammon-Continuous will combine all the previous network outputs in an episode, and the combined action is converted into backgammon actions. The details are explained in Chapter 3.

The proposed novelty is an environment for network agents, which is aimed to provide a comparison between the first DRL agent, the ancestor; TD-Gammon, and the latest DRL agents. With this new environment, any agent suited for GYM-Backgammon will be able to select valid actions from a huge set of actions, with added knowledge of dice, which brings stochasticity, to increase the challenge and the complexity.

1.4 Outline

The outline of this thesis is as follows:

- Chapter 2 provides the background information and related work for this thesis.
- Chapter 3 provides the environment (GYM-Backgammon) and the tested networks for DDPG-Gammon, the experiments, and the results.
- Finally, in Chapter 4, concludes this work and propose the next steps for both the future of DDPG-Gammon and DRL researches.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Reinforcement Learning

Reinforcement Learning is an online learning method, where the agent tries to find the optimal policy for a given task without prior model knowledge, via basic feedback mechanism. The agent's actions will either be rewarded or punished accordingly. Hence, RL differs from supervised learning methods since the agent needs to find the optimal policy on its own by direct feedback from the environment [20].

2.1.1 Temporal Difference

Temporal Difference (TD) learning [23] is composed upon the idea of learning the value function $V(s)$ directly from experiences with TD error. The update commences for a state-action-reward as in Equation 2.1 [23]:

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)], \tag{2.1}$$

where α is learning rate, and $r + \gamma V(s') - V(s)$ is TD error.

2.1.2 Q-Learning

Q-Learning is a model-free reinforcement learning, which aims to create a learning process over state-action pairs with experiencing negative/positive rewards [30]. Given an experience $< s_t, a_t, s_{t+1}, r >$, the update commences for a state-action pair in time $t, Q(s_t, a_t)$ as in Equation 2.2 [30]:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{2.2}$$

where α is learning rate, and γ is the discount factor.

2.1.3 Deep Q Network (DQN)

DQN uses a deep neural network as a function approximator to estimate the optimal Q-value function for individudal actions, which conforms to the Bellman optimality equation in 2.3 [12]:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{(a' \in A)} Q(s_{t+1}, a'). \tag{2.3}$$

The policy in DQN is defined implicitly by Q as $\pi(s_t) = argmax_{a' \in A} Q(s_t, a')$. The Q-network approximates a Q-value for each action and it is updated by using off-policy data from an experience replay buffer.

2.1.4 Deep Deterministic Policy Gradients (DDPG)

DDPG is an actor-critic algorithm compatible with continuous action and observation spaces. Similar to DQN, in 3.3.2.1, the Q-value is estimated by critic, using an off-policy data and the recursive Bellman equation in 2.4 [19]:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi_\theta(s_{t+1})), \tag{2.4}$$

where π_θ is the actor (policy). The actor is trained to maximize the critic's estimated

Q-values by back-propagating through both networks. The corresponding algorithm can be found in 2.1.5.4.

2.1.5 Continuous Control with Deep Reinforcement Learning

While DQN is capable of solving problems with high-dimensional observation spaces, it is only able to handle discrete and low-dimensional action spaces [8]. In order to deal with continuous action space, Deep Deterministic Policy Gradient (DDPG) algorithm is presented, a model-free, off-policy actor critic algorithm, that is capable of learn policies in high-dimensional, continuous action spaces [16].

2.1.5.1 Policy-Gradient Method

Since the standard approach of approximating a value function to find a policy is infeasible, an alternative approach is constructed upon the idea of representing the value function by a function approximator, which is proven to be convergent. [21]

For any Markov Decision Process (MDP), the policy gradient can be summarized as in Equation 2.5 [21]:

$$\nabla_\theta J = \int_X d^\pi(x) \int_U \nabla_\theta \pi(x,u) Q^\pi(x,u) du dx, \tag{2.5}$$

where $d^\pi(x)$ is the state distribution, and $\pi(x,u)$ is action distribution over action space, U.

2.1.5.2 Actor-Critic Concept

The Actor-Critic algorithm is used to separate the value function from policy function. The policy function is represented by Actor, and the value function is represented by Critic. The Actor produces an action from given observation, and the Critic produces a Temporal Difference (TD) error for given observation and the reward. Both Actor

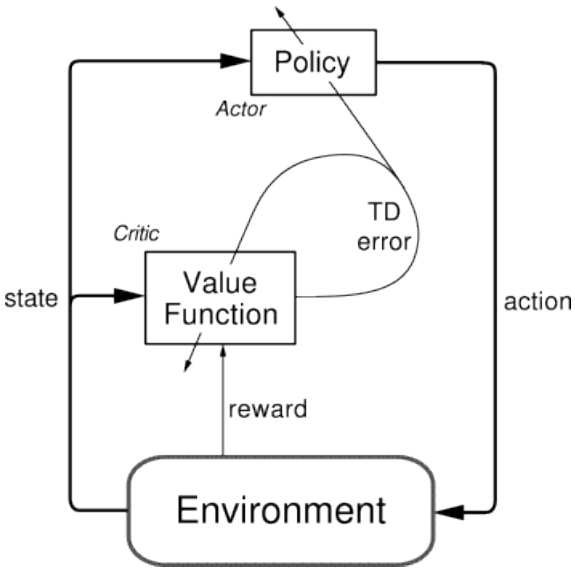


Figure 2.1: The summary of Actor-Critic Methodology. [7]

and Critic functions are trained from the output error of Critic, as shown in Figure 2.1.

2.1.5.3 Batch Normalization

While training deep networks, the distribution and the parameters of the layers change constantly, which is called *internal covariate shift* [6]. In order to prevent saturation in the networks, the learning rate must be reduced, which causes longer training times. In order to address this problem, normalizing the layer inputs is proposed and proven to be successful [6].

Batch Normalization process will normalize each scalar feature apart from each other, by making it have the mean of 0 and the variance of 1. For a layer with d-dimension input $x = (x^{(1)}...x^{(d)})$, the corresponding equation can be found in Equation 2.6 [6].

$$x^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}. \tag{2.6}$$

2.1.5.4 Algorithm

Algorithm 1 DDPG Algorithm [8]

Randomly initialize critic network $Q(s, a|\theta^Q$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weigths $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$.
Initialize replay buffer R
for episode = 1, M **do**
    Initialize a random process N for action exploration
    Receive initial observation state s_1
    for t = 1, T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i} \tag{2.7}$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{2.8}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{2.9}$$

    end for
end for

2.2 TD-Gammon: The Ancestor of Deep Reinforcement Learning

2.2.1 Information and Background Info

TD-Gammon [26] is one of the first self-teaching neural network. Although TD-Gammon's first task was not about creating a winner Backgammon agent, it has surpasses even the best human players, granted master-level play [24].

With its success story, it opened the way for many current reinforcement learning methods to be created, since it has proven that neural networks can act as a non-linear function approximators. Combining this with the advancements in the computational power and the neural networks (such as Convolutional Neural Network (CNN)), TD-Gammon paved the way for Deep Reinforcement Learning.

However, it is important to note that TD-Gammon did not approximate approximated the action-value function $Q(s, a)$. Instead, it has approximated the state value function $V(s)$, and learnt on-policy directly from the self-play games [11].

2.2.2 Continuous Action and Observation Space In Backgammon

Although the action space is discretized in Backgammon, in order to tackle with large space action, the action set is converted into a continuous action space. This helps make networks to be able to represent the actions. Afterwards, these actions are converted back into discrete action space.

The observation space is consisted of the checker counts in every possible board position, 'off' checkers that the player managed to take away from the board, 'bar' checkers that the player needs to put back into the game, and the current dice value.

2.2.2.1 Mapping Discrete Action Space to Continuous Action Space

The actions are constructed in Backgammon as such:

There are 24 different board position that a checker can be on. Generally, a player needs to move 2 checkers within a round, although it may be 4 if the dice is double or 0 if there are no available positions left. Additionally, for a player, a checkers' positions increase, while the opposing player's checkers' positions decrease. For the sake of brewity, this thesis will focus on the initial board position, 2 checker movements while calculating the action space, and one player that its checkers positions decrease.

In the Backgammon engine, for every dice value, there are $(24 - diceValue)$ different action pair, which results in a list: $[23, 22, 21, 20, 19, 18]$

When combined the dice values (from 1 to 6, no doubles, no difference between 1-6 and 6-1), it results in $2300 + 1716 + 1197 + 740 + 342 = 6295$ possible actions at the initial state of Backgammon. If one also includes the observation changes, there are $34 * 10^{21}$ [16] unique state-action pairs in Backgammon.

Hence, traditional reinforcement learning methods, such as Q-Learning, are not a good fit for this problem. With the developments in Deep Reinforcement Learning, some of the agents (such as DDPG) are proven to be trained/tested for continuous action spaces [8], which made Backgammon possible to be trained/tested with newest methods.

The constructed network will output floating point(s), between [0-1], which will be later mapped into Backgammon actions. The details are provided in Chapter 3.

2.2.2.2 Continous Observation Space

There are 2 different observation spaces for Backgammon.

First observation space is the NeuroGammon's [22] observation space, which includes both the player's and the rival's on-off checkers. Although there are 24 different board positions, every place in a board position is coded as a separate input. However, since TD-Gammon is capable of self-learning [26], the observation space also included the current player's type, which is removed. Additionally, the dice value is added to the observation space.

Second observation space is the RGB values of the Backgammon in the corresponding state. With providing this space, any agent that relies on the pixels while training will be able to run Backgammon seamlessly. An example visual can be found in Figure 2.2.

In this thesis, first observation space is used for training and testing the networks.

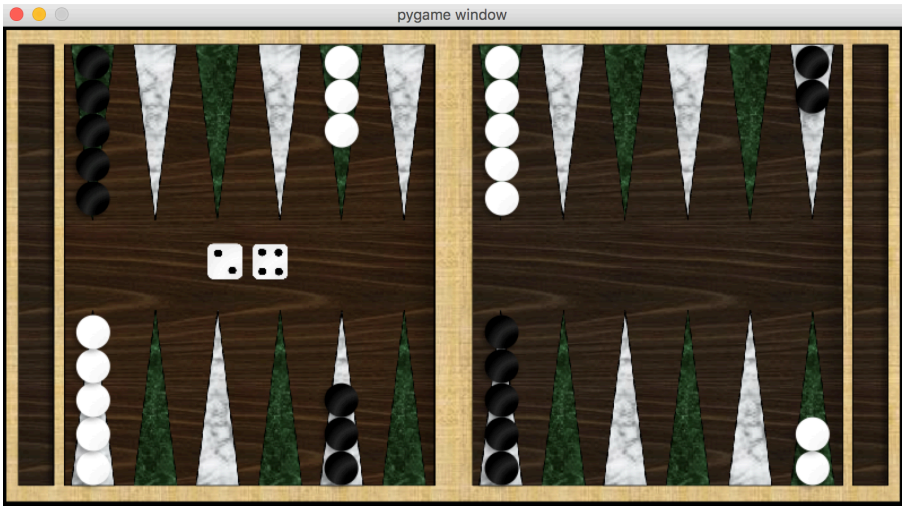


Figure 2.2: An example GUI from Backgammon [5]

2.2.3 Diversity of Actions and Dice Factor

Although the board position is limited to 24, any checker place alteration will change the actions greatly. For example, in Figure 2.3, the observation for the initial game is shown.

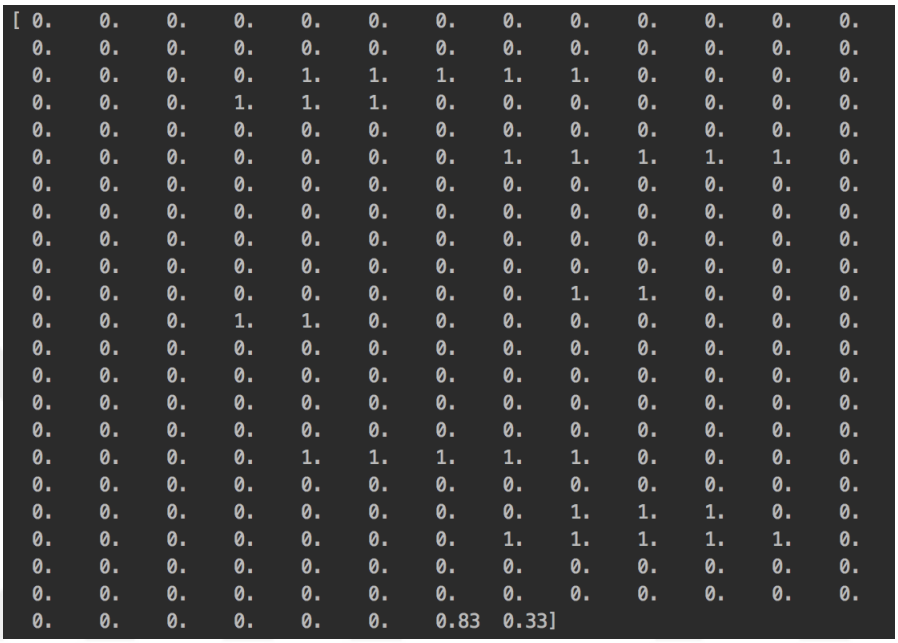


Figure 2.3: Initial observation space from Backgammon.

In this given observation in Figure 2.3, the valid actions are as listed:

[((12, 7), (5, 3)), ((12, 10), (12, 7)), ((23, 21), (7, 2)), ((12, 10), (7, 2)), ((12, 7), (23, 21)), ((23, 21), (12, 7)), ((12, 7), (7, 5)), ((12, 10), (10, 5)), ((12, 7), (12, 10)), ((7, 2), (23, 21)), ((7, 2), (5, 3)), ((7, 5), (7, 2)), ((7, 2), (7, 5)), ((7, 2), (12, 10)), ((7, 5), (12, 7)), ((5, 3), (7, 2)), ((5, 3), (12, 7))]

After a random play from the player and the random opposing agent, the observation space changes as seen in Figure 2.4:

And the valid actions for the observation in Figure 2.4 becomes:

[((7, 6), (12, 8)), ((5, 1), (7, 6)), ((5, 4), (9, 5)), ((5, 4), (4, 0)), ((7, 6), (5, 1)), ((23, 19), (7, 6)), ((9, 5), (5, 4)), ((9, 8), (5, 1)), ((7, 6), (23, 19)), ((23, 19), (9, 8)), ((7, 3),

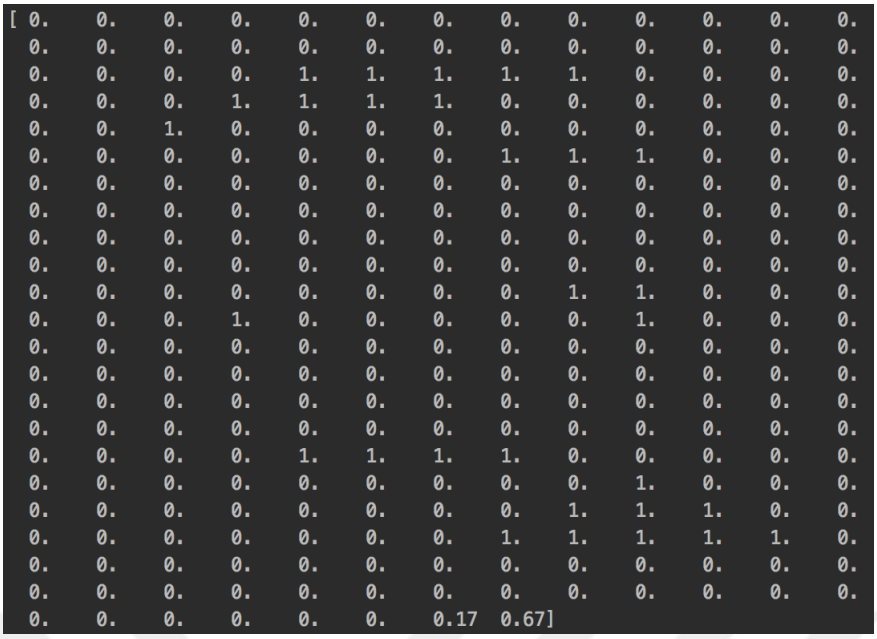


Figure 2.4: New observation space after one turn.

(7, 6)), ((9, 8), (8, 4)), ((23, 19), (23, 22)), ((5, 4), (12, 8)), ((7, 6), (9, 5)), ((5, 1), (23, 22)), ((9, 8), (23, 19)), ((23, 19), (5, 4)), ((5, 4), (5, 1)), ((5, 1), (5, 4)), ((12, 8), (23, 22)), ((7, 3), (3, 2)), ((12, 8), (9, 8)), ((23, 22), (5, 1)), ((23, 22), (12, 8)), ((23, 22), (23, 19)), ((12, 8), (5, 4)), ((5, 4), (7, 3)), ((7, 3), (9, 8)), ((12, 8), (8, 7)), ((23, 22), (9, 5)), ((9, 8), (7, 3)), ((9, 8), (12, 8)), ((12, 8), (7, 6)), ((5, 1), (9, 8)), ((23, 22), (7, 3)), ((7, 3), (23, 22)), ((7, 6), (6, 2)), ((5, 4), (23, 19)), ((5, 1), (1, 0)), ((9, 5), (7, 6)), ((7, 6), (7, 3)), ((9, 5), (23, 22)), ((7, 3), (5, 4))]

So, even with one action taken, the observation space and the valid action space changes majorly. This rule also applies for the dice values, even greater. Although dice values just occupy the last 2 inputs in the observation space, the action space changes significantly. For example, in Figure 2.4, the dice value is $(1, 4)$. If the dice value is set to $(2, 5)$, which accumulates to a change of $(0.17, 0.17)$ in the observation space, the valid actions becomes:

[((12, 7), (5, 3)), ((9, 4), (12, 10)), ((5, 0), (23, 21)), ((7, 2), (2, 0)), ((23, 21), (5, 0)), ((7, 2), (23, 21)), ((23, 21), (12, 7)), ((7, 2), (12, 10)), ((5, 0), (9, 7)), ((23, 21), (7, 2)), ((12, 7), (9, 7)), ((9, 4), (4, 2)), ((9, 4), (5, 3)), ((12, 7), (23, 21)), ((5, 3), (9, 4)), ((9,

4), (23, 21)), ((23, 21), (9, 4)), ((9, 7), (7, 2)), ((7, 5), (7, 2)), ((12, 10), (12, 7)), ((12, 10), (7, 2)), ((12, 7), (7, 5)), ((12, 7), (12, 10)), ((9, 4), (7, 5)), ((7, 2), (7, 5)), ((5, 3), (5, 0)), ((7, 2), (5, 3)), ((7, 5), (9, 4)), ((5, 3), (7, 2)), ((12, 10), (9, 4)), ((7, 5), (12, 7)), ((5, 3), (12, 7)), ((5, 0), (12, 10)), ((12, 10), (5, 0)), ((5, 0), (7, 5)), ((9, 7), (5, 0)), ((7, 2), (9, 7)), ((9, 7), (12, 7)), ((7, 5), (5, 0)), ((12, 10), (10, 5)), ((5, 0), (5, 3))]

To summarize, any observation space change can create massive changes in the valid action space. The agents/networks in the training process must be robust enough to handle these changes in order to be proven successful, which explains the complexity of the constructed task.

2.3 OpenAI-GYM

OpenAI-GYM is a toolkit for developing and comparing reinforcement learning algorithms [1]. The concept of the GYM is to create a unified library which provides a common structure for any kind of reinforcement-based problems/games. The problems/games are called "environments", and since these environments have a shared interface, one can create a general algorithm which can be applied to every environment in the GYM.

Some example environments in OpenAI-GYM are Pendulum [14] and CartPole [13]. Both enviroments have continuous observation space, but Pendulum has continuous action space, whereas CartPole has discrete action space. These environments' visualizations can be seen in Figure 2.5.

In this thesis, there will be two new environments, namely GYM-Backgammon-Continuous and GYM-Backgammon-Discretized, which are inspired from these environments. Their details can be found in Chapter 3.

Although these are basic problems compared to GYM-Backgammon, since Pendulum

(a) Pendulum (b) CartPole

Figure 2.5: Example OpenAI-GYM Environment Visuals

is proven to be solved by DDPG [3] and CartPole is proven to be solved by DQN [9], these environments constructed the framework of this thesis work.

2.4 Related Work

2.4.1 Deep Reinforcement Learning Compared with Q-Table Learning Applied to Backgammon

In this work, the author has provided a comparison between the traditional Q-Learning method and the DRL methods in the Backgammon, which encouraged this thesis to focus on DRL methods, since their results [15] show that Q-Learning is still a better option than DRL in Backgammon.

However, in this work, Backgammon is modified in a major way to utilize Q-Table. For example, the checker count is reduced from 30 to 12, the board position count is reduced from 26 to 9, and last but not least, they have used a 3-sided dice instead of 6-sided dice. With this information, we may conclude that this setup will not be enough to challenge the state-of-the-art Backgammon agent, TD-Gammon.

In this thesis, there will not be any modification(s) regarding the game play. The only constraint introduced is that doubled-actions (such as 6-6, 5-5) are disabled for the sake of simplicity. Nevertheless, the networks are capable of handling double-actions

just by doubling the action count (outputs) for the network(s).

CHAPTER 3

DDPG-GAMMON

3.1 Preparing Backgammon For Networks

The networks are built upon having 4 different floating point outputs, which represent the board position selected to play. These action outputs will be mapped into discrete actions by interpolating $[0-1]$ into $[0-25]$ for Backgammon engine, in order to convert the values into board positions.

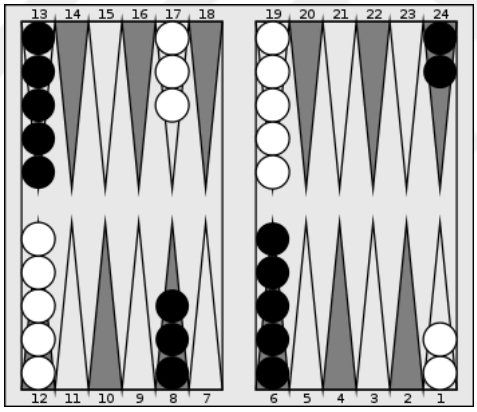


Figure 3.1: Backgammon Board Notation [10]

These numbers represent the board positions, shown in Figure 3.1. Number 25 is 'off' position, which represents an action that involves removing checker off the table. Number 24 is 'on' position, which represents an action that involves putting an 'off' checker on the table.

Each GYM-Backgammon environments are responsible for constructing Backgammon engine pairs for each step, which looks like $((3,9),(7,1))$.

After pairing, the constructed pair is checked whether they are valid or not in the current gameplay. If the pair is valid, the backgammon engine plays the given action, and returns the new observation with a new value of dice. If the pair is invalid, the observation and the dice value will stay as is.

3.2 GYM-Backgammon

In order to test various DRL agents, Backgammon is suited into an environment in GYM. It is published and available on GitHub [27].

With this novelty, any algorithm that supports Continous Action Space/Observation Space will be able to run/train/test GYM-Backgammon without further ado. And also, since the foundation of GYM-Backgammon is the same with TDGammon, the trained agents can be tested againts the state-of-the-art Backgammon agent(s).

There are 2 main branches of GYM-Backgammon, Discretized and Continous. They have been tailored to create different scenarios for Backgammon, which will be tested head-to-head.

3.2.1 GYM-Backgammon-Discretized

In this environment, every output of the networks is singular, within range of $[0 - 1]$. The conversion algorithm can be found in Algorithm 2.

As an example, the current action $A = [0.12, 0.36, 0.28, 0.04]$ is converted into $[3, 9, 7, 1] = ((3, 9), (7, 1))$, and the previous action is discarded.

Algorithm 2 Discrete Action Calculation

for each episode **do**
  Initialize current action A with $[0, 0, 0, 0]$
  for each step **do**
    required: Selected action in current step = A'
    $A = A'$
    Convert current action A into Backgammon action A''
    $A'' = A * 25$
    Construct Backgammon action pair $P = ((A''[0], A''[1]), (A''[2], A''[3]))$
    Feed P into Backgammon engine
  end for
end for

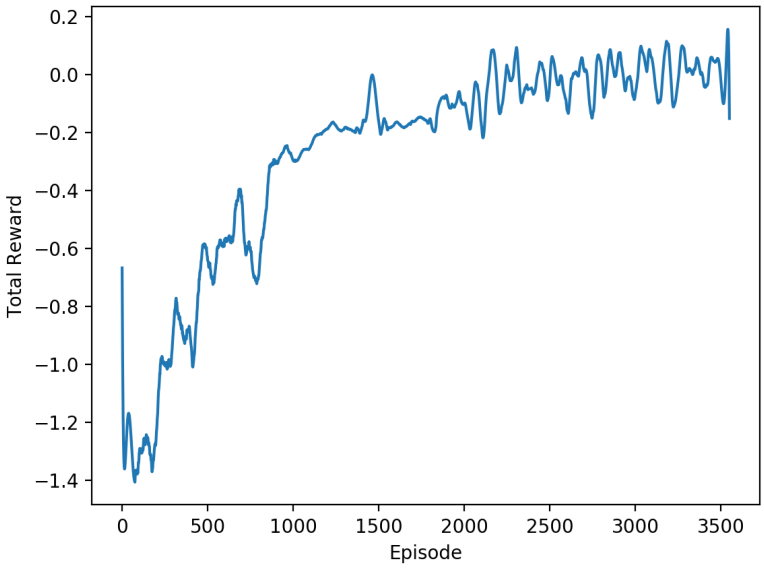


Figure 3.2: Test Of Discretized Backgammon

3.2.1.1 Test Of Concept

As a test of concept, DDPG-Gammon-Discretized is trained on, aimed to find only single action, $((12, 7), (5, 3))$, without batch normalization. The results can be seen in Figure 3.2.

In this figure, after 3500 episodes, the average reward exceeds 0, which means that the network has successfully find the action provided. After that episode, the network is converged into the found action and in the next episodes it continues to find it with a decreased period of steps.

Hence, it is conclusive that DDPG-Gammon-Discretized is capable of converging with provided network and reward system.

3.2.2 GYM-Backgammon-Continuous

In this environment, the network outputs is combined with each step, within range of $[0 - 0.1]$. The conversion algorithm can be found in Algorithm 3.

Algorithm 3 Continuous Action Calculation

for each episode **do**
  Initialize current action A with $[0, 0, 0, 0]$
  for each step **do**
    requried: Selected action in current step = A'
    $A = A + A'$
    Convert current action A into Backgammon action A''
    $A'' = A * 25$
    Construct Backgammon action pair $P = ((A''[0], A''[1]), (A''[2], A''[3]))$
    Feed P into Backgammon engine
  end for
end for

As an example, in Step 1 and Step 2, the actions are as provided:
$[0.01, 0.03, 0.02, 0.04]$
$[0.03, 0.04, 0.01, 0.02]$

The step outputs are summed up to $[0.04, 0.07, 0.03, 0.06]$, which is converted as $[1, 2, 1, 1] = ((1, 2), (1, 1))$ and to Backgammon action space. After various steps, the constructed action will increase or decrease according to the previous step selections.

This environment is constructed in order to find out whether Backgammon is suitable to be converted into a real continuous control.

3.2.2.1 Test Of Concept

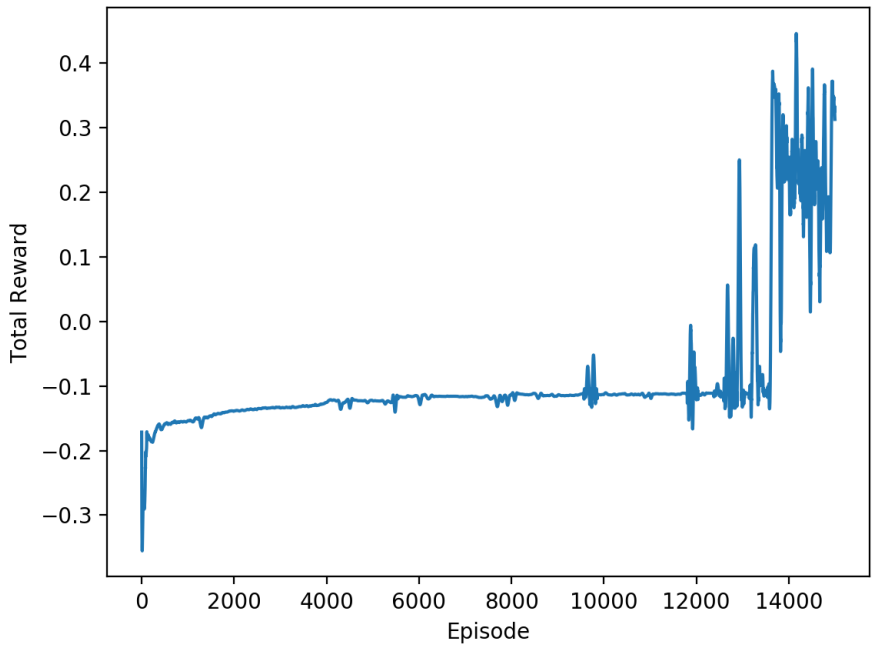


Figure 3.3: Test of Continuous Backgammon

As a test of concept, DDPG-Gammon-Continous is trained on, aimed to find only single action, $((12, 7), (5, 3))$, without batch normalization. The results can be seen in Figure 3.3.

In this figure, after 12K steps, the network manages to find the given action, gaining a positive reward. After that, the network also succesfully finds the next actions, increasing the total reward to above 0.2.

Hence, it is conclusive that DDPG-Gammon-Continuous is capable of converging and

increase its cumulative reward with provided network and reward system.

3.3 Experiments

With the information of tests in 3.2.1.1 and 3.2.2.1, the experiments are constructed to push DDPG-Gammon to its boundaries to find out appropriate setups for better and faster converging rate. There are 4 different setups for DDPG-Gammon-Discretized, and 4 different setups for DDPG-Gammon-Continuous, which is detailed in Section 3.3.2.3.

3.3.1 Network Layouts

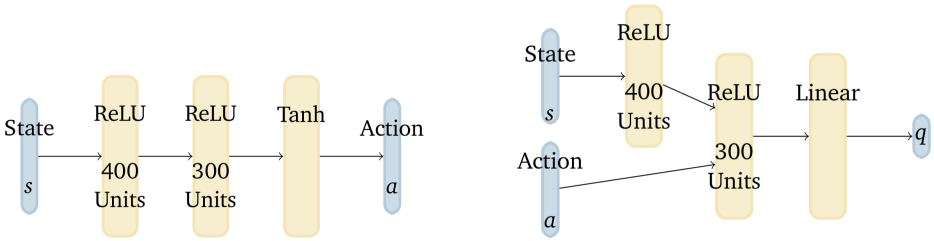
There are 2 networks, **Actor** and **Critic**. **Actor** network is fully connected, and has approximately $141,120,000$ trainable parameters, whereas **Critic** network has approximately $35,281,200$ trainable parameters.

The **Actor** network is constructed upon 4 fully connected layers with sizes of:

- Input Layer (Observations): 294
- First Hidden Layer: 400
- Second Hidden Layer: 300
- Activation Layer: $Tanh$
- Output Layer (Actions): 4
- Learning Rate: $1e^{-}4$

The **Critic** network is constructed upon 4 fully connected layers, with the addition of another input layer consisted of Actions, with sizes of:

- Input Layer (Observations): 294



(a) Actor Network (b) Critic Network

Figure 3.4: Network Layout [16]

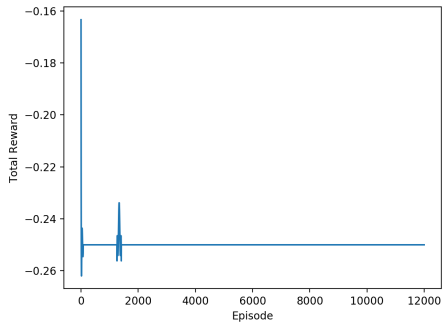
- Secondary Intput Layer (Actions): 4
- First Hidden Layer: 400
- Second Hidden Layer: 300
- Activation Layer: $Linear$
- Output Layer (Q-Value): 1
- Learning Rate: $1e^{-}3$

The layouts can be seen in Figure 3.4.

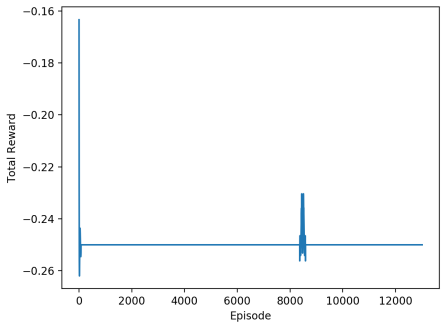
3.3.2 Setups

3.3.2.1 Random Seed

Since the dice values are assigned randomly, the dice values series will different for each gameplay, since these values are bound to a random number generator (RNG) [2]. However, if that RNG is seeded, it is possible to have the same dice value series for the next gameplays.



(a) Discrete Network - Without CRAS



(b) Continuous Network - Without CRAS

Figure 3.5: Networks without CRAS

3.3.2.2 Custom Reward and Action System (CRAS)

With a negative static reward for invalid actions, the networks cannot converge, as can be seen in Figure 3.5. There are only few changes in the reward, which is not sustained. Hence, the reward system is modified such that the networks are expected to find the first action from given valid actions.

The reward for an invalid action is calculated according to the absolute distance between the network's selected action and the target action, as shown in Algorithm 4.

Algorithm 4 Example CRAS

1: **required:** Negative Reward Coefficient: $c : -0.1$
2: **for** each episode **do**
3: **required:** Target Action: $T = ((12, 7), (5, 2)) = ((0.48, 0.28), (0.2, 0.08))$
4: **for** each step **do**
5: **required:** Selected action in current step: $A = ((11, 6), (8, 2)) = ((0.44, 0.24), (0.32, 0.08))$
6: Calculate Distance: $d = |T - A| = [0.04, 0.04, 0.12, 0.00] = 0.2$
7: Calculate Total Reward: $r = d * c = 0.2 * -0.1 = -0.02$
8: **end for**
9: **end for**

3.3.2.3 Configurations

In the setups for DDPG-Gammon-Discretized, if the random seed is enabled, the step size for an episode is **64**. If the random seed is not enabled, the step for an episode is **1000**.

In the setups for DDPG-Gammon-Continuous, the step size is either **25** or **100**.

The Discretized setups are created with two changing options, Batch Normalization and Random Seed. Continuous setups are created with two changing options, Step Size and Random Seed.

3.4 Results

3.4.1 Setup 1

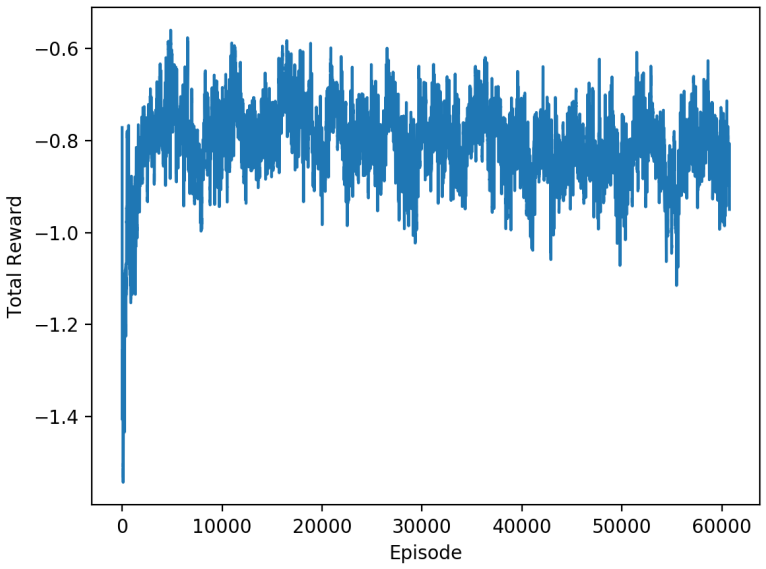


Figure 3.6: Results of Setup 1

Without batch normalization and without random seed: In Figure 3.6, the total

reward for an episode fluctuates highly, and there is no sign of positive reward at all. The overall total reward is near to -1.0, which needs to be converged into -0.1 in order to find a valid action.

3.4.2 Setup 2

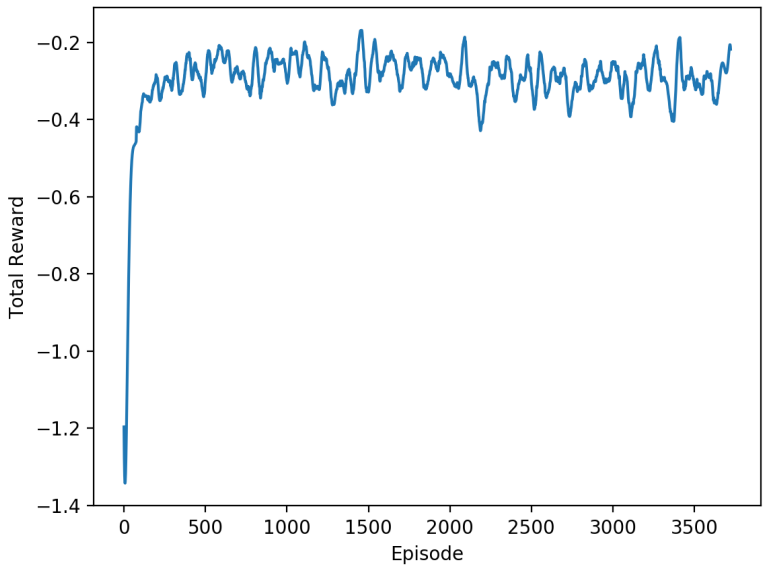


Figure 3.7: Results of Setup 2

Without batch normalization and with random seed: In Figure 3.7, the total reward for an episode is more stable than Setup 3.4.1, and there are multiple picked valid actions. Since the network continues to train after finding a valid action, the negative rewards after the observation state changes outweights the positive reward from the valid action. Nevertheless, this setup is the most promising one amongs the others.

3.4.3 Setup 3

With batch normalization and without random seed: In Figure 3.8, the fluctuation in the total reward is similiar with Setup 3.4.1. However, the overall average reward is near to -0.6, which concludes that batch normalization is a better choice for this

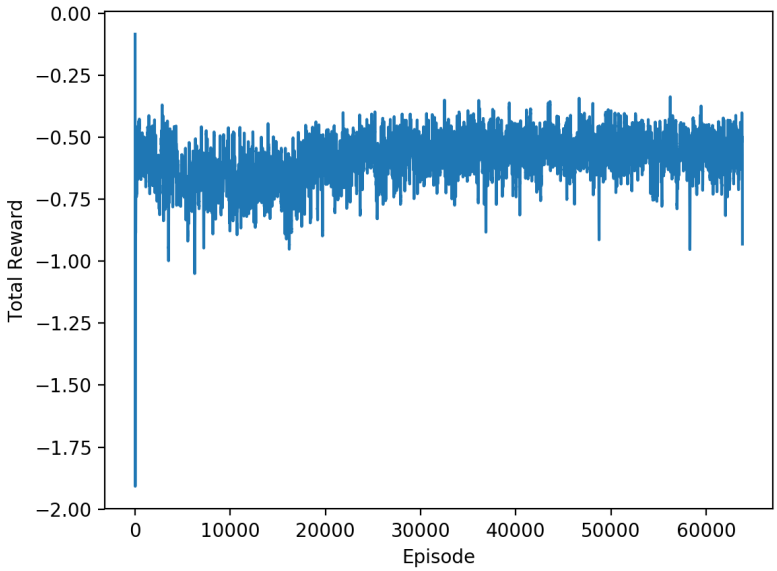


Figure 3.8: Results of Setup 3

environment. Also, in few different episodes, the setup manages to find a valid action.

3.4.4 Setup 4

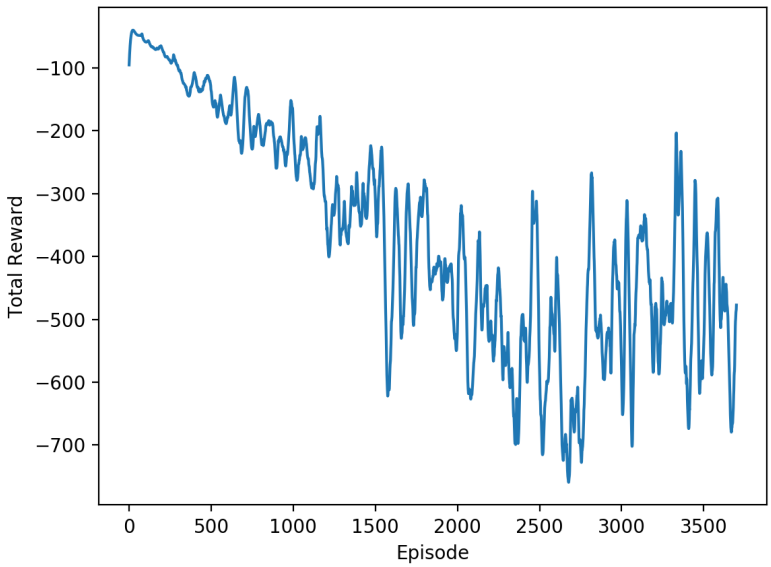


Figure 3.9: Results of Setup 4

With batch normalization and with random seed: Since the random seed is enabled, the network inputs are similar to each other within episodes. Since the batch normalization regulates the network layers to reduce noise, it requires different inputs in order to be effective. As it can be observed in Figure 3.9, the network is diverging, which concludes that batch normalization is not effective when the random seed is enabled. Also, to compare with Setup 3.4.3, the total reward drops down hundreds, which is expected to be within a range of $[-1, 0+]$.

3.4.5 Setup 5

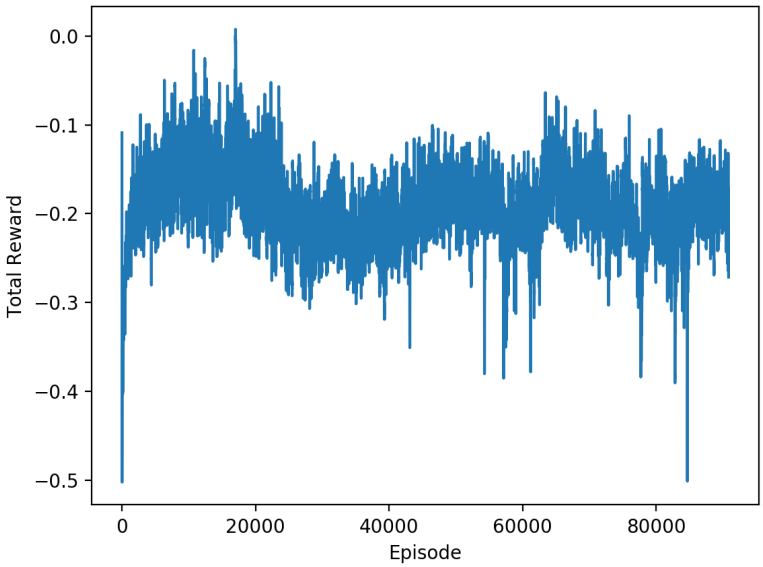


Figure 3.10: Results of Setup 5

With 25 step size and without random seed: In Figure 3.10, it can be observed that although the network has found few valid actions, it did not converge into overall positive reward.

3.4.6 Setup 6

With 25 step and with random seed: As seen in Figure 3.11, although there are oscillations to negative reward area, the network has successfully converged into

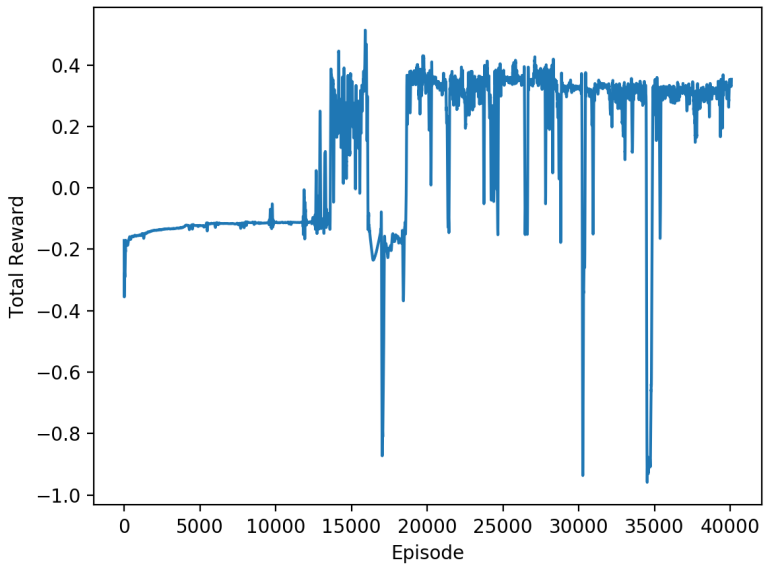


Figure 3.11: Results of Setup 6

positive reward area, with multiple actions found. However, it is limited to 3 consecutive actions, which is caused by small step size. Without enough step size, the network does not have enough time to explore additional actions.

3.4.7 Setup 7

With 125 step size and without random seed: In Figure 3.12, the similarity with its counterpart Setup 3.4.5, can be seen as expected. However, with increased step size, the overall reward is lower than its counterpart setup, because of the fact that the actions are cumulative. With increased action cumulation and without having stable observation series, the network oscillates on -1 total reward.

3.4.8 Setup 8

With 125 step size and with random seed: In Figure 3.13, it can be concluded that increasing the step size creates more volatility, which even returned the network to its initial state. Nevertheless, the network handles the abrupt change, and returns back to the positive reward area. With increased step size, this setup manages to find

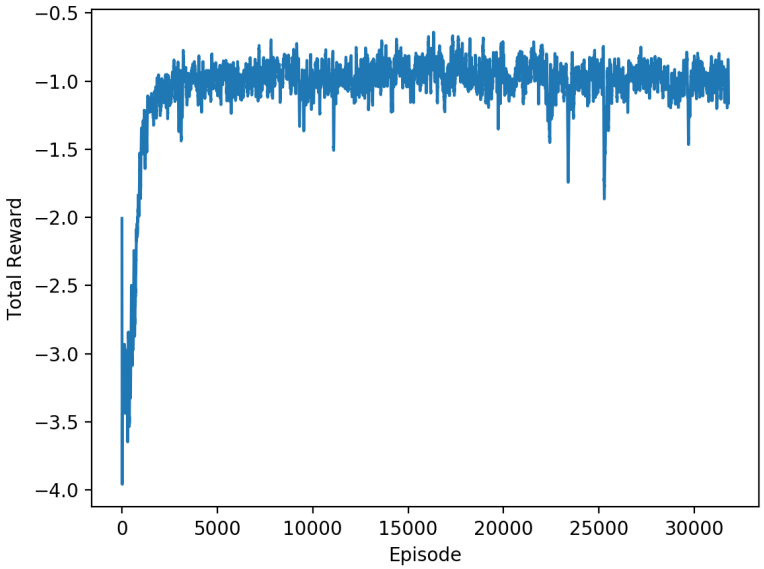


Figure 3.12: Results of Setup 7

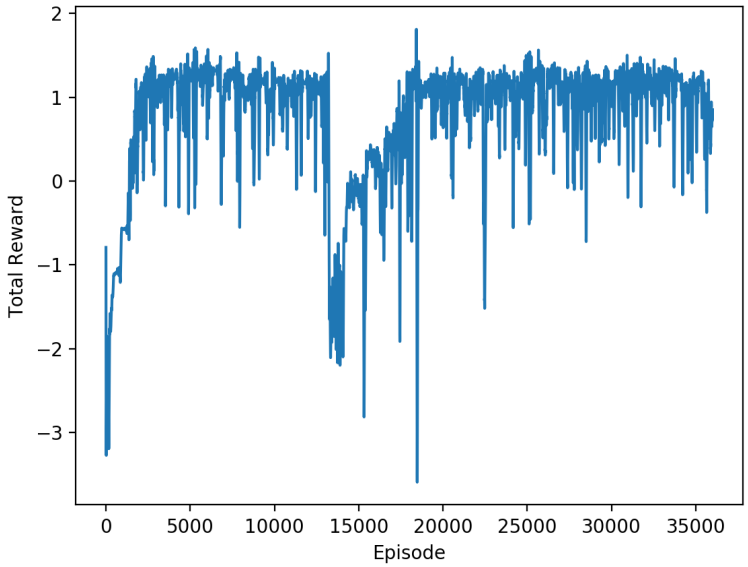


Figure 3.13: Results of Setup 8

more consecutive actions from its counterpart Setup 3.4.6, since the network has more opportunity to explore additional actions.

3.4.9 Summary

In conclusion, DDPG-Gammon-Discretized manages to find multiple valid actions, but with the distruption in the valid action space, the network setups are not able to handle the changes, hence it is not able to continue finding other actions. Since DDPG concept relies on the gradient changes from multiple state-actions to train Actor-Critic networks [8], these abrupt changes keeps the network state fluctuating within episodes.

On the other hand, DDPG-Gammon-Continuous manages to find consecutive actions, because of the fact that the cumulative actions creates an environment that smooths abrupt changes in the action space.

As a summary; Backgammon, a game with huge action space and the stochasticity of dice factor, is suited into a playground such that the newest DRL methods can be applied/trained/tested without any additional changes required. One of these methods (DDPG) has been tested with minor modifications, which proved that any agent/network that supports continuous action space/observation space will be able to play Backgammon.

However, as the results show, trying to pinpoint the actions from action space is infeasible. Hence, the Backgammon engine must be modified to be suitable for supporting continuous action space.

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

This thesis proposes a new step for the future of Deep Reinforcement Learning, from its very roots, with Backgammon. By converting an enormous discrete action space into continuous action space, without any further modification, Backgammon can be again the milestone of the future for DRL.

Creating a Backgammon agent is a very difficult task to handle. Due to huge number of possible observation space, table look-up is not a working solution. Also, due to high branching ratio from the stochasticity that dice brings, brute-force is also not working [23]. In this thesis, two different environments are proposed to tackle with this problem, and to create a "real" Backgammon agent that does not rely on selecting from the possible actions and involves the dice factor.

DDPG-Gammon is tested for its robustness and its success, with multiple network options and different GYM-Backgammon setups. As the results show, Backgammon's observation space and the volatility of its valid action space makes it a complex challenge to take upon. Nevertheless, few of the setups managed to find valid actions consecutively, which shows that it is eligible to be used as a train/test environment.

With custom reward/action system, and its Discretized/Continuous setups, GYM-Backgammon is ready for the next generation of DRL agents and its multiple variations.

4.2 Future Work

This thesis focused on being able to teach an agent how to *play* Backgammon, not *select* an action from given sets, so the agent isn't motivated with winning or losing the episode. Hence, the future work involves creating an agent which can *play* Backgammon from start to end, and wins in its encounters with its state-of-the-art ancestor, **TD-Gammon**. Nevertheless, DDPG-Gammon setup is suitable for endgame rewards, which enables DDPG-Gammon to learn the winning moves, too.

Another work will include pure discretization of Backgammon, which will also enable DDPG-Gammon to be compared with agents that works on discrete action space, such as DQN. The actions will be defined similar to DDPG-Gammon-Continuous, except discrete actions will increase/decrease the selected action pair values by a selected integer coefficient.

Last but not least, DDPG-Gammon will be tested and compared with Trust Region Policy Optimization [17], as TRPO-Gammon.

REFERENCES

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[2] P. Documentation. random. `https://docs.python.org/3/library/random.html`, 2018.

[3] H. Farkhoor. ddpg-pendulum-v0. `https://gist.github.com/heerad/1983d50c6657a55298b67e69a2ceeb44#file-ddpg-pendulum-v0-py`, 2017.

[4] C. Gaskett, D. Wettergreen, and A. Zelinsky. Q-learning in continuous state and action spaces. In N. Foo, editor, *Advanced Topics in Artificial Intelligence*, pages 417–428, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[5] A. Hannun. Backgammon. `https://github.com/awni/backgammon`, 2017.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[7] V. Konda. *Actor-critic Algorithms*. PhD thesis, Cambridge, MA, USA, 2002. AAI0804543.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[9] H. S. Lodha. Deep-q-network. `https://github.com/harshitandro/Deep-Q-Network`, 2018.

[10] P. Magriel. *Backgammon*. Times Book, 1976.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529 EP –, 02 2015.

[13] OpenAI-GYM. Cartpole v0. `https://github.com/openai/gym/wiki/CartPole-v0`, 2018.

[14] OpenAI-GYM. Pendulum v0. `https://github.com/openai/gym/wiki/Pendulum-v0`, 2018.

[15] H. C. K. Peter, Winberg, and M. May. Deep reinforcement learning compared with q-table learning applied to backgammon. 2016.

[16] S. Ramstedt. *Deep Reinforcement Learning with Continuous Actions*. PhD thesis, 2016.

[17] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[18] D. Silver. Gradient temporal difference networks. In M. P. Deisenroth, C. Szepesvári, and J. Peters, editors, *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 117–130, Edinburgh, Scotland, 30 Jun–01 Jul 2013. PMLR.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395. JMLR.org, 2014.

[20] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[21] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[22] G. Tesauro. Neurogammon: a neural-network backgammon program. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 33–39 vol.3, June 1990.

[23] G. Tesauro. Temporal difference learning and td-gammon. `http://www.bkgm.com/articles/tesauro/tdl.html`, 1992.

[24] G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[25] G. Tesauro. Programming backgammon using self-teaching neural nets. *Artif. Intell.*, 134(1-2):181–199, Jan. 2002.

[26] G. Tesauro. Programming backgammon using self-teaching neural nets. 134:181–199, 01 2002.

[27] E. D. Usta. Gym-backgammon. `https://github.com/edusta/gym-backgammon`, 2018.

[28] H. van Hasselt and M. A. Wiering. Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, April 2007.

[29] H. van Hasselt and M. A. Wiering. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*, pages 1149–1156, June 2009.

[30] C. J. C. H. Watkins and P. Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.