ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

AUTOMATED CURRICULUM DESIGN FOR REINFORCEMENT LEARNING WITH GRAPH THEORY AND EVALUATION HEURISTICS

M.Sc. THESIS

Anıl ÖZTÜRK

Department of Computer Engineering

Computer Engineering Programme

JUNE 2021

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

AUTOMATED CURRICULUM DESIGN FOR REINFORCEMENT LEARNING WITH GRAPH THEORY AND EVALUATION HEURISTICS

M.Sc. THESIS

Anıl ÖZTÜRK
(504181504)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Nazım Kemal Üre

JUNE 2021

<u>**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ**</u>

ÇİZGE KURAMI VE DEĞERLENDİRME BAZLI SEZGİSEL YÖNTEMLER İLE PEKİŞTİRMELİ ÖĞRENME İÇİN OTOMATİK MÜFREDAT TASARIMI

YÜKSEK LİSANS TEZİ

Anıl ÖZTÜRK
(504181504)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Nazım Kemal Üre

HAZİRAN 2021

Anıl ÖZTÜRK, a M.Sc. student of ITU Graduate School student ID 504181504, successfully defended the thesis entitled “AUTOMATED CURRICULUM DESIGN FOR REINFORCEMENT LEARNING WITH GRAPH THEORY AND EVALUATION HEURISTICS”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Nazım Kemal Üre** Istanbul Technical University	....
Jury Members :	**Assoc. Prof. Dr. Uğur Behçet Töreyin** Istanbul Technical University	....
	Dr. Gökhan Budan Eatron Technologies	....

Date of Submission : 11 June 2021
Date of Defense : 29 June 2021

To my family and friends who strongly support me,

FOREWORD

I would like to express sincere gratitude to thank Asst. Prof. N. Kemal Üre for his help in discovering the problems in the domain we are working on and for supervising the whole process by not avoiding any help that will encourage my creativity while solving the identified problem.

I am also grateful to all the educators who contributed to me many things that I used and did not use in my thesis on behalf of computer sciences during my graduate education, and Istanbul Technical University for giving me this opportunity.

Finally, I would like to thank all my friends who have contributed greatly to the relief of my stress due to the COVID pandemic, common life problems and academic responsibilities by sending me quality memes through any social media platform, and Lo-Fi internet radios, which helped me to keep my focus at a high level during my thesis and my research during my graduate education.

This research is supported by Istanbul Technical University BAP Grant NO: MOA-2019-42321 and Eatron Technologies.

June 2021 Anıl ÖZTÜRK

TABLE OF CONTENTS

ABBREVIATIONS

AI	: Artificial Intelligence
ACL	: Automated Curriculum Learning
ANN	: Artificial Neural Network
BCL	: Balanced Curriculum Learning
CL	: Curriculum Learning
DL	: Deep Learning
GAN	: Generative Adversarial Deep Neural Network
ML	: Machine Learning
MDP	: Markov Decision Process
PCL	: Progressive Curriculum Learning
RL	: Reinforcement Learning
SPCL	: Self-Paced Curriculum Learning
SPL	: Self-Paced Learning
App	: Appendix

SYMBOLS

S	**:** Set of markov states
A	**:** Action space
P	**:** State transition probability matrix
R	**:** Reward function
γ	**:** Discount factor
π	**:** Policy
Q	**:** Q-value function
∇_θ	**:** Gradient of θ

LIST OF TABLES

LIST OF FIGURES

AUTOMATED CURRICULUM DESIGN FOR REINFORCEMENT LEARNING WITH GRAPH THEORY AND EVALUATION HEURISTICS

SUMMARY

In reinforcement learning, models try to find the environment dynamics and the policy that will take them to the target by exploring the environment. For this purpose, environment-centric or model-centric auxiliary methods can be used to teach the target task to the models in a faster way. Model-centric solutions mostly serve to change the architecture and update sensitivity of the model, while environment-centric solutions allow to adjust the environment dynamics and the difficulty of the target task. In reinforcement learning problems, in order for the model to analyze the environment thoroughly, it must first discover most critical and ground situations, and then learn (exploit) all possible values of all important situations. But the more a model is inclined to explore, the further away it is from exploitation, and the more inclined it is to exploitation, the further away from discovery. Both of the mentioned approaches offer various suggestions to solve the exploration-exploitation dilemma, which is one of the intensive research topics in the reinforcement learning field.

It may be slow or impossible for the model to learn a relatively difficult environment or task in one try. The model may not be able to learn conditions that require long-term planning or immediate action due to the large number of variables and combination spaces they contain. The curriculum learning structure allows the model to go through phases with distinctive difficulty differences throughout the training process. In this context, methods have been proposed that enable the model to see the examples it collects from the environment with a certain frequency, present the pre-designed discrete environment designs to the model with increasing difficulty, teach a more difficult task and enable it to achieve more success in an easier task.

With automated curriculum learning methods, the need for domain expertise in the curriculum learning structures and the effort spent on model optimization processes are tried to be minimized. Student-teacher neural networks working mutually with each other, predefined methods that can instantly reduce or increase the difficulty according to the learning outcomes of the model, and dynamically changing the conditions, rewards and goals of the environment are the strategies frequently used in automated curriculum learning algorithms.

During the design process of the proposed algorithm, two preliminary studies were carried out and the effects, advantages, disadvantages and types of curriculum learning on the basic learning process were investigated and their effects were observed. In the first study, the adaptability of an autonomous traffic vehicle model to changing traffic situations was tested. In order to increase the adaptation of the model, the model was

trained in variable traffic environments during the process. It has been observed that the vehicle model trained in a difficult, complex and highly random environment is more successful in simpler traffic scenarios than a model trained in a simple traffic scenario from the very beginning. In the second study, the change in the learning ability of an autonomous vehicle driving model against variable road type and weather conditions was observed in a realistic physics simulation. By comparing the standard trainings of the vehicle model in different environment parameters, a route from easy to difficult was created for the model. It has been observed that the models that have undergone a phased training process by complying with this route information are significantly more successful in the target (the most difficult) environment compared to other models.

In this study, an algorithm-based strategy is proposed for the model to learn the aforementioned difficult problems. Within the scope of the strategy, graph theory and reward metrics were used as references. The environments used for reinforcement learning have been made editable with variables, and an algorithm has been designed that determines the values and orders of the variables in order for the model to have a more stable training process. Within the scope of defined variables, all combinations of variables that environments can have are modeled as separate environments. These discrete environments were subjected to a difficulty ranking by comparing the difference in variables among themselves. The rewards obtained by the model for the possible environment change in each combination are compared, provided that the order of learned difficulty (only going from easy to hard) is followed. Changes in rewards are determined as the weights of edges in the generated curriculum graph. The magnitude of the change in reward is indicative of the difference between the model's initial reward and the reward it received in the new environment. By running the shortest-path algorithm on the generated curriculum graph, a route is searched for by experiencing the least possible total reward change from a starting environment to the target environment. The route that includes the least amount of reward change is determined as the curriculum learning route, and the created model learns the environment combinations in this route. At the end of the process, the model learns to perform its task in the target environment.

In order to test the proposed algorithm, virtual game environments known in the field were used by making them parametrized-changable with custom variables. It has been tried not to choose environments that are known to be easy to learn. Since one of the hypotheses put forward by the algorithm is to shorten the training time, relatively difficult problems are chosen. The algorithm was run 10 times for each environment independently and with fixed randomization seed to ensure reproducibility of the results. The results are reported in such a way that the outputs of all 10 trials can be interpreted as common.

PPO was chosen as the type of model used throughout the experiments, and deep learning architectures and algorithms were used. Python programming language and dedicated process servers with Ubuntu operating system were used in order to keep the runtime as short as possible.

The test outputs show that the proposed algorithm gives the results of the normal training process kx times and it contributes +k% to the results at the end of the process. While the proposed method gives advantageous results over the standard method in most of the test environments; for some environments, it showed break-even results with the standard method - no improvement in output. Since the parameters of the models created during the training and the initial conditions of the environments can be evaluated in the context of randomness, a seed variable was assigned to all the randoms that could be seeded, and an average result was obtained over 10 trials in order to obtain a more stable result from those that could not be seeded.

The results indicate that the curriculum outcomes of the algorithm have obvious advantages over standard training. It is understood from this study that data (environment)-centric innovations are as important as model-centric developments and inventions. During the experiments, it was observed that a variable thought to be important had no effect at all, and the variables thought to have a positive effect had a negative effect. Thanks to the proposed curriculum learning strategy, the most suitable curriculum can be created for the artificial intelligence model without having an expert knowledge about the environment. This situation is an opportunity to discover the importance of a variable that is new about the environment or whose impact is suspected as well as creating an efficient curriculum route. The study can be improved to generate more adaptive variable values with Gaussian-based randomized sampling methods. The results would be expected to be similar to the contributions of hyper-parameter optimization to other machine learning techniques.

ÇİZGE KURAMI VE DEĞERLENDİRME BAZLI SEZGİSEL YÖNTEMLER İLE PEKİŞTİRMELİ ÖĞRENME İÇİN OTOMATİK MÜFREDAT TASARIMI

ÖZET

Pekiştirmeli öğrenmede, modeller ortam dinamiklerini ve onları hedefe götürecek poliçeyi ortama keşfederek bulmaya çalışırlar. Bu amaç doğrultusunda modelleri hedefe hızlı yoldan ulaştırabilmek için ortam bazlı veya model bazlı yardımcı metotlardan faydalınabilmektedir. Model tabanlı çözümler daha çok modelin mimarisini ve güncelleme hassasiyetini değiştirmeye yararken, ortam tabanlı çözümler ortam dinamiklerini ve hedef görevin zorluğunu ayarlamayı sağlamaktadır. Pekiştirmeli öğrenme problemlerinde modelin ortamı iyice analiz edebilmesi için öncelikle çoğu kritik ve temel durumu keşfetmesi, ardından bütün önemli durumların tüm olası değerlerini öğrenmesi (sömürmesi) gerekir. Fakat bir model ne kadar keşfetmeye meyilli olursa sömürüden o kadar uzaklaşır, aynı şekilde sömürüye ne kadar meyilli olursa da keşfetmekten o kadar uzaklaşır. Bahsedilen iki yaklaşım da pekiştirmeli öğrenme alanındaki yoğun araştırma konularından biri olan keşif-sömürü ikilemini çözmek adına çeşitli öneriler sunmaktadır.

Görece zor bir ortam veya görev karşısında, modelin hedef görevi tek seferde öğrenmesi yavaşlayabilir veya imkansızlaşabilir. Model, uzun süreli planlama veya ani aksiyonları gerektiren koşulları barındırdıkları çok sayıda değişken ve kombinasyon uzayının genişliğinden ötürü öğrenemeyebilir. Müfredat öğrenme yapısı, modelin eğitim süreci boyunca birbirinden ayırt edici zorluk farkları olan fazlardan geçmesini sağlar. Bu bağlamda modelin öğrenmek için ortamdan topladığı örnekleri belli bir sıklıkla görmesini sağlayan, önceden tasarlanmış ayrık ortam tasarımlarını modele gitgide zorlaştırarak sunan, daha zor bir görevi öğretip daha kolay bir görevde daha fazla başarıya ulaşmasını sağlayan metotlar önerilmiştir.

Otomatik müfredat öğrenme metotları ile, bahsi geçen müfredat öğrenme yapılarındaki alan uzmanlığı gereksinimi ve model optimizasyon süreçlerine sarfedilen efor en aza indirgenmeye çalışılmaktadır. Birbiriyle müşterek bir şekilde çalışan öğrenci-öğretmen sinir ağları, modelin öğrenme çıktılarına göre zorluğu anında düşürüp artırabilen öntanımlı metotlar ve ortama ait durumları, ödülleri ve hedefleri dinamik şekilde değiştirmek otomatikleştirilmiş müfredat öğrenme algoritmaları içerisinde sıklıkla başvurulan stratejilerdendir.

Önerilmiş algoritmanın tasarımı süreci boyunca iki adet ön çalışma yapılmış olup müfredat öğrenmenin temel öğrenme sürecine olan etkileri, avantajları, dezavantajları, çeşitleri araştırılmış ve etkileri gözlenmiştir. İlk çalışmada otonom bir trafik aracı modelinin değişen trafik durumlarına gösterdiği adaptasyon becerisi sınanmıştır. Modelin adaptasyonunun artırılabilmesi amacıyla model, süreç içerisinde değişken

trafik ortamlarında eğitilmiştir. Zor, karmaşık ve rastlantısallığı yüksek ortamda eğitilen araç modelinin daha basit trafik senaryolarında en başından beri basit trafik senaryosunda eğitilen bir modele kıyasla daha başarılı olduğu gözlemlenmiştir. İkinci çalışmada ise gerçekçi bir fizik simülasyonunda bir otonom araç sürüş modelinin değişken yol tipi ve hava durumu karşısındaki öğrenme becerisinin değişimi gözlemlenmiştir. Araç modelinin değişik ortam parametrelerindeki standart eğitimleri kıyaslanarak model için kolaydan zora giden bir rota oluşturulmuştur. Bu rota bilgisine uyarak parçalı bir eğitim sürecinden geçen modellerin hedef (en zor) ortamda diğer modellere kıyasla belirgin derecede daha başarılı olduğu gözlemlenmiştir.

Bu çalışmada, bahsi geçen zor problemleri modelin öğrenebilmesi için algoritma tabanlı bir strateji önerilmiştir. Strateji kapsamında, çizge kuramı ve ödül metrikleri referans olarak kullanılmıştır. Pekiştirmeli öğrenme için kullanılan ortamlar değişkenler ile düzenlenebilir hale getirilmiş, ilgili modelin daha kararlı bir eğitim süreci yaşaması adına değişkenlerin değer ve sıralarını belirleyen bir algoritma tasarlanmıştır. Tanımlı değişkenler kapsamında, ortamların sahip olabileceği bütün değişken kombinasyonları ayrı birer ortam olarak modellenmiştir. Sözkonusu ayrık ortamlar kendi aralarında değişkenlerdeki farklılık mukayese edilerek bir zorluk sıralamasına tabii tutulmuştur. Öğrenilmiş zorluk sıralamasına uymak (sadece kolaydan zora gitmek) şartıyla, her kombinasyondaki olası ortam değişimi için modelin elde ettiği ödüller kıyaslanır. Ödüllerdeki değişimler oluşturulmuş müfredat çizgesindeki bağlantıların ağırlıkları olarak belirlenir. Ödüldeki değişimin büyüklüğü, modelin en başta aldığı ödül ile yeni ortamda aldığı ödülün arasındaki farkın göstergesidir. Oluşturulmuş müfredat çizgesi üzerinde en-kısa-yol algoritması çalıştırılarak bir başlangıç ortamından hedef ortama olabilecek en az toplam ödül değişimini yaşayarak gidilebilecek bir rota aranır. Olabilecek en az miktarda ödül değişimi içeren rota müfredat eğitim rotası olarak belirlenir ve oluşturulmuş model bu rotadaki ortam kombinasyonlarını sırasıyla öğrenerek sürecin sonunda hedef ortamdaki görevini gerçekleştirmeyi öğrenmiş olur.

Önerilmiş algoritmanın denenmesi amacıyla alanda bilinen sanal oyun ortamları değişkenler ile değiştirilebilir hale getirilerek kullanılmıştır. Kolay öğrenildiği bilinen ortamlar olabildiğince seçilmemeye çalışılmıştır. Algoritmanın öne sürdüğü hipotezlerden birisi de eğitim süresini kısaltmak olduğundan göreli zor problemlere odaklanılmıştır. Algoritma her ortam için 10 kez bağımsız ve sonuçların yeniden üretilebilir olması adına sabit rastlantısallık ile çalıştırılmıştır. Sonuçlar 10 denemenin de çıktılarının ortak olarak yorumlanabileceği şekilde raporlanmıştır.

Deneyler boyunca kullanılan modelin tipi olarak PPO seçilmiş, derin öğrenme mimarisi ve algoritmaları kullanılmıştır. Süreç boyunca Python programlama dili kullanılmış ve çalıştırma zamanının olabildiğince kısa olması adına işe tahsis edilmiş, Ubuntu işletim sistemine sahip özel işlem sunucuları kullanılmıştır.

Test çıktıları önerilen algoritmanın normal eğitim sürecinin verdiği sonuçları ortalama kx kat daha hızlı verdiğini, sürecin sonundaki sonuçlarda ise +k% bir katkı sağladığını göstermektedir. Önerilen metot test ortamlarının çoğunda standart metoda göre avantajlı sonuçlar vermekteyken; bazı ortamlar için standart metot ile başabaş sonuçlar göstermiş, çıktılarda herhangi bir iyileşme alınamamıştır. Eğitim esnasında oluşturulan modellerin parametreleri, ortamların başlangıç koşulları gibi durumlar rastlantısallık bağlamında değerlendirilebileceği için, sabitlenebilecek bütün rastlantısallara bir tohum değişken atanmış, sabitlenemeyenlerden daha istikrarlı bir sonuç çıkarmak adına ise 10 deneme üzerinden ortalama bir sonuç çıkarılmıştır.

Sonuçlar, algoritmanın müfredat çıktılarının standart eğitime göre bariz avantajlar barındırdığını belirtmektedir. Veri (ortam) odaklı yeniliklerin de model odaklı geliştirmeler ve buluşlar kadar önemli olduğu bu çalışmayla da anlaşılmaktadır. Yapılan deneyler esnasında önemli olduğu düşünülen bir değişkenin hiç etkisinin olmadığı, değişiminin pozitif bir etki yaratacağı düşünülen değişkenlerin negatif etki yarattığı durumlar da görüldü. Önerilen müfredat öğrenme stratejisi sayesinde, ortam hakkında uzmanlık seviyesinde bir bilgi birikimine sahip olmadan da yapay zeka modeli için öğrenmeye en elverişli müfredat oluşturulabilmektedir. Bu durum, verimli müfredat rotasını oluşturmanın yanısıra; ortam hakkında yeni veya etkisi hakkında şüphe duyulan bir değişkenin önemini keşfetmek için de bir fırsat niteliği taşımaktadır. Çalışma, Gaussian tipi rastlantısal örnekleme metotları ile daha adaptif değişken değerleri sunacak şekilde geliştirilebilir. Sonuçların hiper-parametre optimizasyonunun diğer makine öğrenmesi tekniklerine katkılarıyla benzer olması beklenecektir.

1. INTRODUCTION

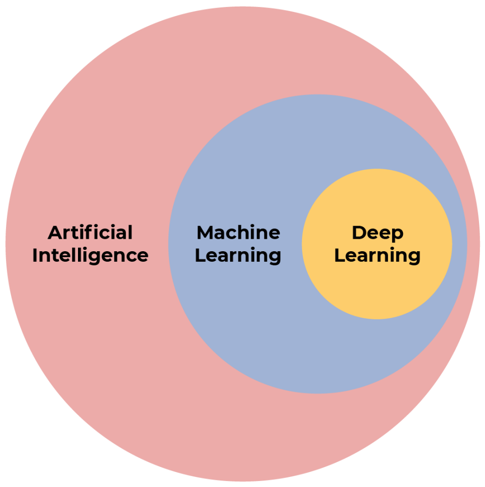


Figure 1.1 : Hierarchical representation of AI, ML and DL [1]

Machine learning (ML) is a subfield of artificial intelligence (AI). It aims to create an interpretation suitable for the data format, to understand the data and to find an intermediate function that produces successful results for the data. Most ML methods draw their strength from essential statistical theories and approaches. They perform the desired tasks by creating a canonical structures according to the distribution of variables. Throughout its development until today, ML has been used to solve supervised problems such as regression [2], classification [3–6] and unsupervised problems such as clustering [7, 8] and anomaly detection [9, 10].

Traditional ML methods perform good in many applications work with structured-tabular data. But they might struggle when it comes to the unstructured data like image, text or video. For this type of data, deep learning (DL) techniques are preferred. While most traditional ML techniques evaluate variables within boundaries, DL models take these variables as parameters. They pass the parameters through a linear model, then apply a non-linear activation on them at the output, thus creating an irreversible complex representation for the input. This way, an encoding process for unstructured data that traditional ML methods cannot perform is performed. Like ML, regression and classification tasks can be performed on unstructured data with DL.

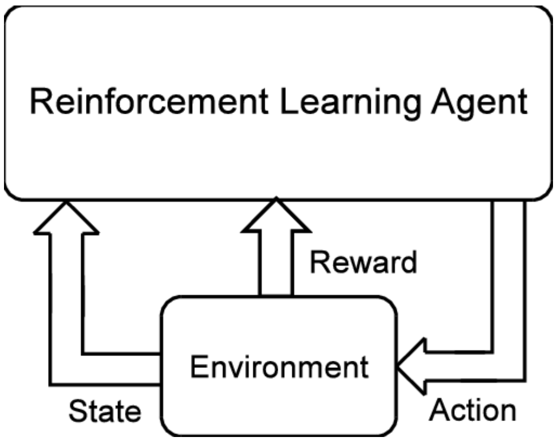


Figure 1.2 : Working principle of RL models in their basic form [11]

In addition to extracting insights from structured or unstructured data, algorithms were designed also for decision-making in real life or simulation scenarios. This requirement has led to the development of algorithms that will enable machines to reach the desired target with maximum efficiency in any iterative scenario. Reinforcement learning (RL) covers all problems and solutions that can be evaluated within this domain. In a standard RL problem, the model can be defined as "an agent that perceives and acts in an environment" [12]. The essential parts of a RL structure are model, policy, reward and value function [13, 14]. The agent takes actions with respect to its policy π. The reward indicates if the agent performs good or bad with its taken action in the environment. The value function is an estimation of the future cumulative reward. It is useful for evaluating the states and action selection.

In its most basic form, any RL problem can be thought of as a markov decision process (MDP). (S,A,P,R,γ) tuple can define a MDP. In that definition; S is set of Markov states, A is the action space of the RL agent, P is the state transition probability matrix, R is the reward function and γ is the discount factor. γ is set between $0 < \gamma \leq 1$. The next state of the agent is extracted by the probabilistic transition Eq. 2.1 determined by the current state and action.

$$P(s_{t+1} = s' | s_t = s, a_t = a) \qquad \textbf{(1.1)}$$

In this context, the aim of an RL model is to produce an optimal action scheme (policy) $\pi : S \rightarrow A$ by taking the maximum total reward within the provided environment. The reward function Eq. 2.2 produces a probabilistic expected value output according to the state transition rewards.

$$E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t, s_{t+1})\right] \tag{1.2}$$

The MDP-typed definitions mentioned above are also widely used approaches in control theory. Most of the models created with this understanding are based on the work of Richard Bellman. Bellman showed in his studies [15, 16] that if the problems that can be modeled as MDP are solved with dynamic programming, the processing load is reduced considerably. An optimization function prepared with an approach suitable to Bellman's methodology using value function and state transitions is used to find an sub-optimal policy for RL agents. This optimality equation is non-linear, and generally there is no closed form solution. However, the iterative algorithms such as value iteration, policy iteration, Q-Learning [17] and SARSA [18] are proposed as solutions for this problem.

In addition to the reinforcement algorithms based on iterative structured flows, there are also approaches that the deep neural networks are used. This fusion improves performance by taking advantage of the above-mentioned DL architectures' ability to derive meaningful encodings from unstructed data. One of the biggest examples of this fusion approach is the success of the Deep Q-Network architecture on Atari [19, 20].

The above mentioned developments in the RL area have helped RL agents solve most problems with ease. However, RL agents have always had difficulty while learning problems that involve too many tasks or have too difficult tasks. Curriculum learning [21] is teaching a machine learning model how to solve a difficult task. Curricula are methodologies that people benefit greatly in their education in any domain. It is a highly effective method that is used not only in human tasks but also in animal training [22, 23]. The methodology reduces the difficulty of education process and increase its quality by splitting it into phases. The most difficult problems or tasks are not given to the student at the first stage of their education. The student must first learn and go through the essentials of the problem. That way, each mini-problem the student learns will help to solve the next problem. Most of the time, the RL agent will be having difficulties when it's trying to learn a difficult task from-scratch due to the task's complexity. The curriculum approach encourages increasing the difficulty level gradually. This way; the agent is always being challenged with the new configurations, but not as much as in the training from-scratch scenario. In the first study in which this method was proposed [21], it was seen that the reward convergence was happened faster and towards higher levels. Many curriculum learning strategies have been proposed and tested by researchers. These will be discussed in Section 1.2.

After the definitions about the curriculum, an automated version can be mentioned. Automated curriculum learning (ACL) is the creation of curricula automatically in a certain optimization context. Due to [24], it can be defined as:

$$Obj : \max_{D} \int_{T \sim T_{target}} P_T^N \, dT \tag{1.3}$$

where P_T^N represents the cumulative reward for the agent at the task T at the training iteration N. With this pseudo formula, maximum cumulative reward at minimum training iterations can be searched. Since it can be defined as sort of a meta-learning process, D is the whole training pipeline (task sequence or scenario) for the agent.

Automated curriculum learning eliminates the necessary domain expertise in curriculum learning by leaving the environment exploration to statistical methods. In this way, it can provide faster insight and better performance in newly discovered or waiting to be discovered domains. The studies carried out on behalf of automated curriculum learning and their intended use will also be discussed in Section 1.2.

1.1 Purpose of Thesis

The aim-objective of this thesis is to propose a method for creating sub-optimal curriculum scenarios for RL agents.

The proposed system aims for any RL agent to achieve maximum success in the target setting for environment by training in the optimum order and time in given environment setting space. During this process, the environment settings were represented as a graph and a resource constrained shortest path algorithm was used. The edge weights are determined with a custom heuristic that uses the differences of episodic rewards that the agent get in different environment settings.

As a future work, this study would define environment parameters by sampling in a certain range based on probability distribution functions instead of defining the environment parameter setting space manually.

1.2 Literature Review

After Bengio's work [21], many researchers began experimenting with curriculum methodology in all areas where ML could be applied. Studies with the curriculum method have achieved significant success in neural machine translation [25,26], object [27,28] and weakly supervised object localization [29,30] tasks. These studies have shown that dynamically adjusting the complexity and density of samples is better than sampling with a fixed stochastic manner.

Curriculum methods can also be divided according to their working style. In a method, the difficulty level is estimated at the learning stage and the probabilistic distributions of the samples are updated instantly. This approach is called self-paced learning (SPL). This approach was applied in one study [31] by updating the sample likelihoods, and as a result, improvements were observed. A study [32] also approached the subject as the class imbalance problem in a classification task, defending that the samples should be sufficiently diverse. In order to do this, they have designed prerequisites to ensure that samples are constantly taken from different parts of the pictures. They called it balanced curriculum learning (BCL). With the self-paced curriculum learning (SPCL) proposed by a study [33], these two methods are being used jointly. As a result, pre-defined conditions and difficulty detection processes are carried out together. Another approach suggests modifying task descriptions rather than sorting samples in the context of a challenge. Thus, as the model performs better, the target task gets harder or more detailed. In a study [34], it is encouraged to reduce the defined dropout rate within the network. It may also help to solve the exploration-exploitation dilemma in reinforcement-learning. This can be called Progressive Curriculum Learning (PCL). In the teacher-student approach, while the student network learns to solve the network task, the teacher network tries to learn the optimal parameters required for the student network. This approach [35] was first proposed for reinforcement-learning problems. Then it began being used in other domains. In addition to the curriculum understanding that goes from easy to difficult, hard-example mining [36] and anti-curriculum [37] approaches that prioritize difficult examples and high-noise sounds in speech recognition tasks have also been tried.

ACL approaches can be used to change the problem in more than one direction. Prioritized experience replay [38] can be used in RL problems with strict task definitions to improve sample efficiency. In scenarios where job description is too difficult or rewards are too sparse, it is very difficult for the agent to learn from scratch on its own. To prevent this, teacher-student architectures can be used to gradually change task difficulties. In one study [39], the agent was trained with a scenario that progressed from easy mazes to difficult mazes, and better results were obtained than training from scratch. In a study [40], it has been observed that starting the robot actuator close to the terminal state gives good results in robotic tasks. Thus, the proximity to the target parameter was determined as the axis of difficulty. The agent learns the basics of the mission by starting at positions close to the target. In the later stages, it is gradually moving away from the target and converges to more difficult tasks faster. This approach can also be called reverse-curriculum. In problem definitions that have excessive sparse rewards, there are also studies [41–43] that define intrinsic rewards and make the problem more learnable for the agent.

ACL methods can be used to modify the initial state, environment, reward, task definitions, and opponents for multi-agent settings. Within the scope of my work, only the work done within the scope of environmental change and transition modification will be mentioned.

It has been shown that teacher-student networks increase the performance of Minecraft maze solvers [39] and Sonic the Hedgehog agents [44]. At the same time, the Procedural Content Generation algorithm [45] has been proposed to create wider combinations of tasks. With this algorithm, successful results were obtained [46] on a robot hand that solves rubik's cube. In multi-target environments, Hindsight Experience Replay (HER) [47] recommends recreating trajectories collected with a specific task-goal to a different task. With this algorithm, the specific target state is replaced with a state on the path to the target. In this way, the agent is enabled to learn the problem piece by piece by determining the parts of the solution as the sub-targets.

1.2.1 Research Gap

The methods described in the aforementioned studies and algorithms are not repeatable, interpretable and simply applicable. All of these features cannot be achieved with a single study. Some are designed for just a single observation type, some for a single environment, and some for a single task type. A method that works with a common parameterized implementation format is not proposed.

1.3 Hypothesis

A curriculum can include many parameters and processes in the context of reinforcement learning. In recent years, many successful algorithms and techniques built upon representing the problems as a graph structure. We modeled the curriculum process as a graph, then set the edge weights as the reward difference of the same agent in these nodes (environments) that the given edge is connected. Considering this structure, if the shortest path algorithm is run on the potential paths the agent would go, it will select the path where the agent will collect the smallest (biggest in negative) reward difference. This means that the agent will be getting more and more challenged in our context. We built a hypothesis with keeping these in mind.

Hypothesis 1: The automated curriculum algorithm will reveal previously unnoticed levels of difficulty and transitions. The scenarios when one parameter makes the environment harder while the other makes is easier will be an important learning step for the reinforcement learning agent.

Hypothesis 2: The dynamic and independent parameter changes that brought by our proposed algorithm will give better results than a random parth or a curriculum process that supervised manually.

2. BACKGROUND

2.1 Reinforcement Learning

Traditional reinforcement learning process is mostly based on Markov Decision Processes (MDP).

A MDP is defined by the tuple (S, A, P, R, γ), where γ is the discount factor, R is the reward function, S is set of Markov states, A is the action space of the agent and P is the state transition probability matrix. The discount factor γ is set between $0 < \gamma \leq 1$. The next state of the agent is governed by the probabilistic transition determines by the current state and current action (see Eq. 2.1) The main purpose of the RL agent is to get maximum total reward in a long term (Eq. 2.2) by interacting with the environment by choosing an optimal policy $\pi : S \rightarrow A$.

$$P(s_{t+1} = s' | s_t = s, a_t = a) \tag{2.1}$$

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, s_{t+1})\right] \tag{2.2}$$

The main objective of Q-Learning is to compute the value function $Q(s,a)$, which determines the long term total reward of taking action a at state s. Hence if the optimal value function is determined, optimal policy π can be obtained by taking the action $a = argmax_a(Q(s,a))$ at state s. Classic Q-learning algorithm iteratively updates the the value function estimate by applying the update in Eq. 2.3, where α_k is the learning rate.

Whether or not if taking an action in a state is good or bad is evaluated by Q-learning through action value function $Q(s,a)$. $Q(s,a)$ for all potential actions and states are stored in memory table $Q[s,a]$. New states and the reward $R(s)$ for that particular action have been collected such that, the succeeding action will be the highest $Q(s', a')$ valued action in $Q[s,a]$. The action value function can be calculated as shown in Equation (2.3) where γ is the discount factor and α_k is the learning rate.

$$Q(s,a) \leftarrow (1-\alpha_k)Q(s,a) + \alpha_k(R(s) + \gamma \sum_{s'} \max_{a'} Q(s',a')) \qquad \textbf{(2.3)}$$

There might be situations such that the continuous action states or actions-spaces can take too much space on memory, this will increase the load on computation of Q.

2.1.1 Policy-wise types of reinforcement learning

In reinforcement learning, agents take actions in an environment to maximize their cumulative rewards in an episode that consists of finite number of steps. Agents tune their policies with the penalties or rewards they get. There are two kinds of reinforcement learning algorithm in terms of policies; on-policy and off-policy.

2.1.1.1 On-policy reinforcement learning

In on-policy reinforcement learning, the policy used for updating the target model and the policy used for selecting the actions (behavior model) are the same. In off-policy reinforcement learning, the policy used for updating the target model and the policy used for selecting the actions (behavior model) can be different. Updating the model only requires specific inputs like state, action, next state and reward.

2.1.1.2 Off-policy reinforcement learning

In off-policy reinforcement learning, the target policy may be deterministic, while the behavior policy can act on the sampled states from all possible past scenarios. This flexibility enables training the algorithm using previous experiences. In this context, all collected iteration data are stored in one place. In each training iteration, a certain amount of data is sampled from this collected data. This set where we sample the experience is called the buffer, and the sampled data is called batch.

2.1.2 Policy gradient reinforcement learning

In policy gradient, policy is usually defined through a parameterized function, like a neural network. The generalized reward function for a policy gradient learner is defined as:

$$J(\theta) = \sum_{s \in S} d^{\beta}(s) \sum_{a \in A} \pi_{\theta}(a|s) Q^{\pi}(s,a) \quad \textbf{(2.4)}$$

In Eq. 2.4, $d^{\beta}(s)$ is the distribution of the behavior policy β. Without using the behavior policy, the value function Q^{π} can be calculated only by the target policy. The difference at using π or π_{θ} at decision-making stage is what makes an algorithm on-policy or off-policy.

As mentioned before, there is a rapidly increasing complexity in the definitions of continuous space state and action due to the expansion of the probability space. This complexity can be handled much better with policy-based methods. By using gradient ascent, θ parameter can be updated in the direction of the gradient $\nabla_{\theta} J(\theta)$ and converge towards the policy that brings the highest reward.

2.1.2.1 Execution of policy gradient

$\nabla_{\theta} J(\theta)$ is complicated to calculate because it depends on both the choice of action and the stationary distribution of states. Given that the environment is not generally fully known, it is difficult to estimate the impact on the state distribution by a policy update.

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_{\theta}(a|s) Q^{\pi}(s,a) \quad \textbf{(2.5)}$$

The original equation (equation 2.5) for a policy gradient execution includes taking derivative for state distribution d.

$$\nabla_{\theta} J(\theta) \propto \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s,a) \quad \textbf{(2.6)}$$

By using policy gradient theory, we can get rid of the gradient computation cost for state distribution. The result of equation 2.6 will be proportional with equation 2.5. The full proof can be seen in Sutton's work [48].

2.1.3 Trust region policy optimization (TRPO)

Applying very large differences during the update of the policy may destabilize the process. Trust Region Policy Optimization (TRPO) [49] tries to solve this problem by applying KL divergence [50] constraints on the size of policy update. While doing off-policy update, the behavior policy β is different from optimized target policy π. The objective function estimates the cumulative advantage over state distribution and actions. Meanwhile, importance sampling is compensating the error between ground-truth data state distribution and the training data state distribution. Normally, the function is like in equation 2.7:

$$J(\theta) = \sum_{s \in S} \rho^{\pi_{\theta_{old}}}(s) \sum_{a \in A} \pi_\theta(a|s) \hat{A}_{\theta_{old}}(s,a) \tag{2.7}$$

When importance sampling is applied, it forms as in equation 2.8:

$$J(\theta) = \sum_{s \in S} \rho^{\pi_{\theta_{old}}}(s) \sum_{a \in A} \beta(a|s) \frac{\pi_\theta(a|s)}{\beta(a|s)} \hat{A}_{\theta_{old}}(s,a) \tag{2.8}$$

When it's written in expectation format, it forms as in equation 2.10:

$$J(\theta) = \mathbb{E}_{s \sim p^{\pi_{\theta_{old}}}, a \sim \beta} \frac{\pi_\theta(a|s)}{\beta(a|s)} \hat{A}_{\theta_{old}}(s,a) \tag{2.9}$$

Since the true rewards are not known, the reward (advantage) estimation is written as $\hat{A}$. For the definition in equation 2.10, the behavior policy β and parameters of the target policy just before the update $\pi_{\theta_{old}}$ are different because of the off-policy routine. If it's going to run in on-policy manner, the policies should be same. But many rollout workers and optimizators running in parallel can cause the behavior policy to be outdated. To solve that problem, TRPO labels the behavior policy.

$$J(\theta) = \mathbb{E}_{s \sim p^{\pi_{\theta_{old}}}, a \sim \pi_{\theta_{old}}} \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\theta_{old}}(s,a) \tag{2.10}$$

After the behavior policy β denoted as $\pi_{\theta_{old}}$, the TRPO can control the magnitude of the ratio of the new and the old policies. This brings a trust region on the change amount of model's policy. TRPO enforces the distance between the old and the new policies to be small enough (δ) with KL-divergence as in equation 2.11:

$$\mathbb{E}_{s \sim p^{\pi_{\theta_{old}}}} [D_{KL}(p^{\pi_{\theta_{old}}}(.|s) || p^{\pi_\theta}(.|s)] \leq \delta \tag{2.11}$$

Thus, by limiting the size of policy updates, policy changes are prevented from diverging beyond a certain threshold.

2.1.4 Proximal policy optimization (PPO)

Proximal Policy Optimization (PPO) [51] is an easing application on one of its predecessors, Trust Region Policy Optimization (TRPO). The probabilistic ratio in TRPO equation 2.10 can be written as:

$$r_p(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \tag{2.12}$$

After the new functionized variable, the objective function of TRPO in equation 2.10 can be written as:

$$J^{TRPO}(\theta) = \mathbb{E}_{s\sim p^{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}} r_p(\theta)\hat{A}_{\theta_{old}}(s,a) \tag{2.13}$$

If there is no limitation applied on the distance between θ and θ_{old}, maximizing J^{TRPO} would cause instability due to large optimization steps. PPO tries to force $r(\theta)$ to be in a determined interval $[1-\varepsilon, 1+\varepsilon]$. It clips the ratio $r(\theta)$ for the values out of the determined range.

If the architecture shares the parameters between actor and critic networks, the objective function should be changed with the addition of error and entropy terms as in equation 2.14:

$$J^{CLIP'}(\theta) = \mathbb{E}[J^{CLIP}(\theta) - c_1(V_\theta(s) - V_{target})^2 + c_2 H(s, \pi_\theta(.))] \tag{2.14}$$

where c_1 and c_2 are both hyperparameters for the PPO algorithm.

2.2 Deep Learning

Deep learning is a sub-field of machine learning based on the working principle of the human brain. Deep learning algorithms operate on a principle similar to the way neurons in the nervous system communicate with each other.

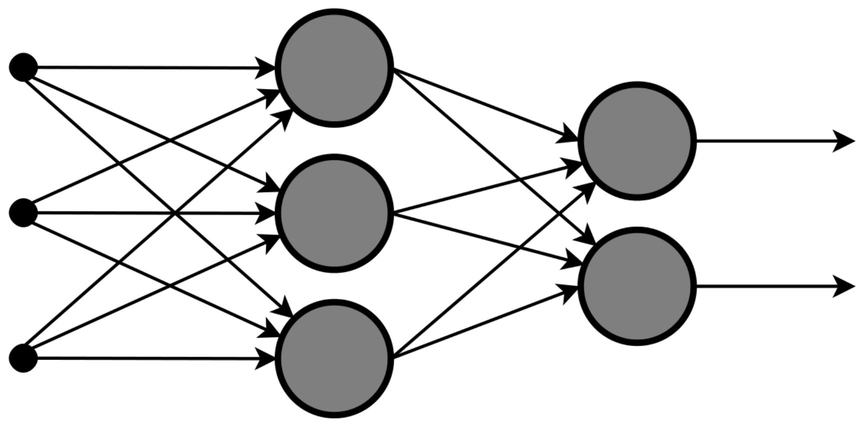


Figure 2.1 : An exemplary deep learning model architecture [52]

As can be seen in the Figure 2.1, deep learning models consist of layers and discrete processing units in each layer called nodes. Nodes in each layer interact separately with all nodes in the next layer. At the end of each layer, a non-linear activation function is applied. Thus, a complex representation is obtained in which each value in the input and all the values generated from these values have separate importances called weights. The most important feature that distinguishes deep learning models from traditional ML methods is this automatic feature-extraction process.

2.2.1 Gradient Descent

Gradient descent is an algorithm that approximates the values that the parameters of a particular function must take in order to obtain the minimum value in an error metric. The parameters of deep learning models are updated with gradient descent. When used, it makes the difference the most when needed parameters cannot be calculated analytically. A gradient descent formula which updates the θ parameter with the learning rate α according to an error metric named E, can be defined as follows:

$$\theta = \theta - \alpha \frac{\delta}{\delta \theta} E \tag{2.15}$$

2.3 Graph Theory

In essence, graph theory is the study of the properties and applications of graphs-networks. A graph consists of vertices (also called nodes) connected by graph edges (also called links). Graphs are one of the main subjects of discrete mathematics research.

2.3.1 Undirected graph

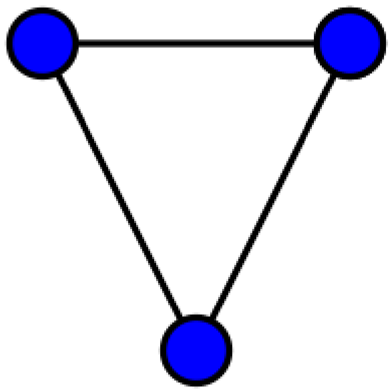


Figure 2.2 : Undirected graph with three nodes and three edges [53]

Undirected graphs like in Figure 2.2 includes the information about the connectivity of each node pair. Operating on undirected graphs generally depends on whether there is an edge between two nodes or not.

2.3.2 Directed graph

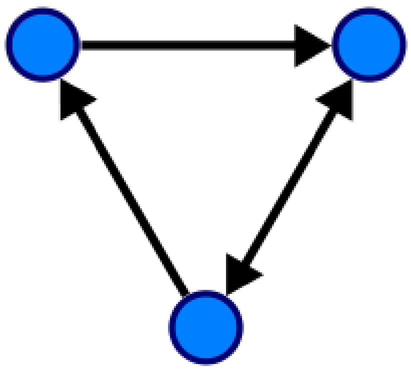


Figure 2.3 : Directed graph with three nodes and four edges [54]

Directed graphs like in Figure 2.3 represent the connection between two nodes, including direction information. In this context, the bi-directional edge in the graph in Figure 2.3 counted as two separate edges. By looking at this kind of graphs, the direction of the information flow can be determined, therefore more customized and realistic systems can be designed.

2.3.3 Weighted graph

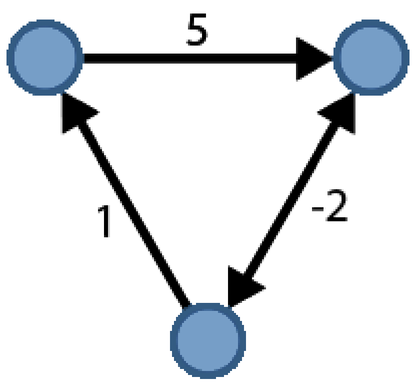


Figure 2.4 : Weighted graph with three nodes and four edges

In a graph, the connection between each pair of nodes may not have the same importance or impact. In such cases, specific links or directions must be able to be weighted differently. In this way, more cost-efficient and optimal results can be obtained in processes such as route selection and planning.

2.3.4 Dijkstra's algorithm

The Dijkstra's algorithm is an algorithm that finds the shortest path between nodes in a graph. This was conceived in 1956 by Edgar W. Dijkstra [55]. The algorithm uses the weighted graph structure to find the shortest path. It iteratively discovers the least resource-consuming (edge weight) route from the starting point to the destination point by using dynamic programming. The algorithm is widely used in route-planning tasks.

The original version of the algorithm finds the shortest path between two given nodes. But modern versions finds the shortest paths between a fixed node and all the other nodes. Dijkstra's algorithm uses optimized loops with classical data structures. This algorithm is different from well-known minimum spanning tree algorithms. Because the shortest path between two nodes may not include all nodes in the graph.

The pseudo-code of Dijkstra's algorithm is given in Algorithm 1:

Algorithm 1: Dijkstra's Algorithm

```
G = graph
s = vertex
for $each\ vertex\ v \in V_G$ do
    $dist[v] \leftarrow \infty$
    $parent[v] \leftarrow UNDEFINED$
end
$dist[s] \leftarrow 0$
$Q \leftarrow V_G$
while $Q \neq \emptyset$ do
    $u \leftarrow EXTRACT - MIN(Q)$
    for $each\ edge\ e = (u,v)$ do
        if $[v] > dist[u] + weight[e]$ then
            $dist[v] \leftarrow dist[u] + weight[e]$
            $parent[v] \leftarrow u$
        end
    end
end
$H \leftarrow (V_G, \emptyset)$
for $each\ vertex\ v \in V_G,\ v \neq s$ do
    $E_H \leftarrow E_H \cup \{(parent[v], v)\}$
end
return $H, dist$
```

3. PRELIMINARY STUDIES

In the preliminary studies to be mentioned, custom PyGame Environment and SimStar simulators developed by Eatron Technologies were used; which are research and product oriented, respectively. Related simulators will be mentioned in related studies.

3.1 Development of a Stochastic Traffic Environment with Generative Time-Series Models for Improving Generalization Capabilities of Autonomous Driving Agents

The main contribution in this work is the development of a data-driven traffic simulator, where I simulate trajectories of the surrounding traffic by using a generative adversarial deep neural network (GAN) trained on real traffic data. Details about the test environment are given in Appendice I.

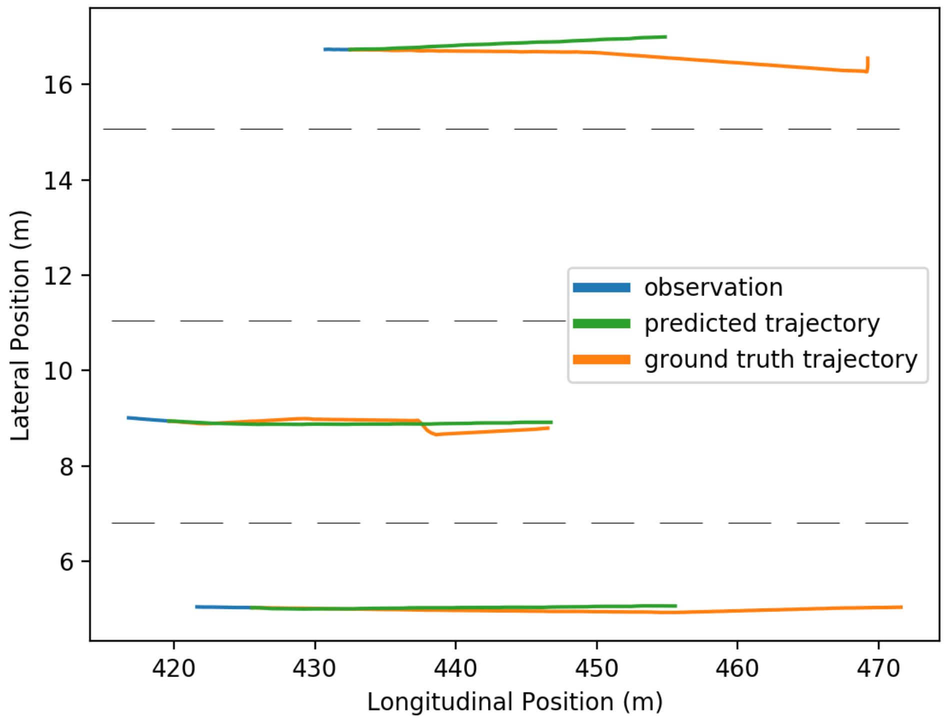


Figure 3.1 : Performance check of the trajectory generator (with few vehicles)

In order to make the work experienceable, a 2D linear traffic environment was created through the PyGame [56] library. While the default vehicles are driven by MOBIL [57] and IDM [58] algorithms, the ego-vehicle was managed by the reinforcement learning model designed within the scope of the study.

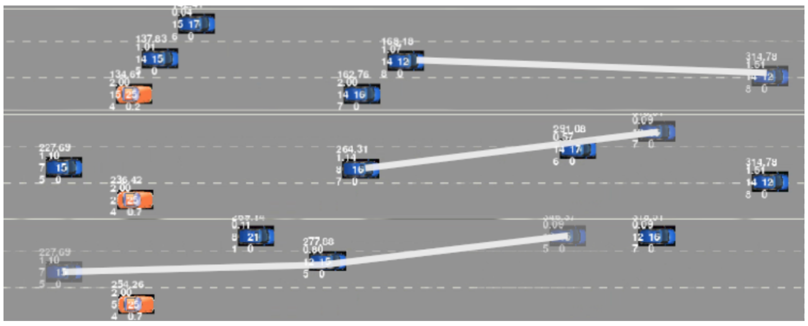


Figure 3.2 : Generated Trajectories in the Simulation

The generated randomized trajectories in Figure 3.2 resemble real life scenarios and thus the developed simulator provides a much richer and realistic environment for training RL agents.

An RL agent has been developed for automated lane changing that is suitable for training on both GAN based and rule-based simulators. To train an agent in environment led by Social-GAN (SGAN) [59]; The training process was carried out on IDM traffic first and then SGAN traffic. A simple and manual curriculum process was provided, allowing the agent to adapt to a more realistic and complex environment faster than it would normally be.

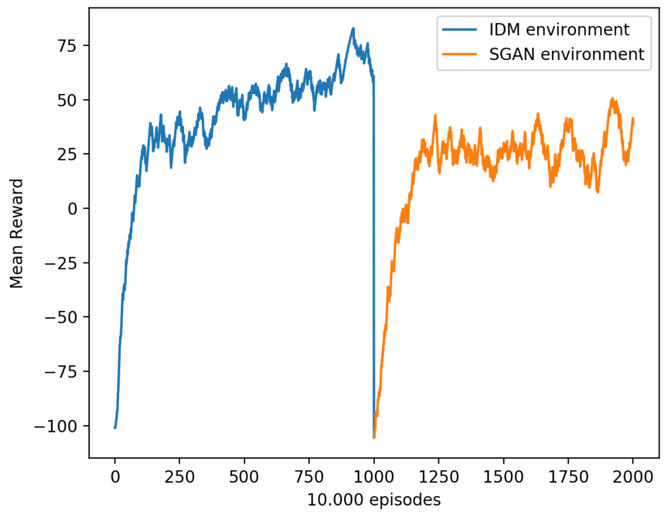


Figure 3.3 : Comparison of reward in two different training phases

As can be seen in Figure 3.3, as a sign of curriculum, the agent experiences a catastrophic reward decrease in the first environment change, then quickly begins to adapt to the new environment settings.

After initialization of the simulation environment, two types of RL agents named Agent$_{\text{IDM}}$ and Agent$_{\text{GAN}}$ have been trained on deterministic traffic scenarios that have been led by MOBIL and IDM algorithms and uncertain traffic scenarios that have been generated by the trajectory generator network, respectively. Agent$_{\text{IDM}}$ has been trained for $10,000,000$ iterations. After that, Agent$_{\text{GAN}}$ has been trained for $3,000,000$ iterations in the uncertain traffic environment with using transfer learning with curriculum manner by taking the Agent$_{\text{IDM}}$'s weights in the initialization step.

3.1.1 Results

Table 3.1 : Number of crashes on Traffic$_{\text{IDM}}$

Models	Hard Crash	Soft Crash
Agent$_{\text{IDM}}$	19	2
Agent$_{\text{GAN}}$	9	0

Table 3.2 : Performance of agents on Traffic$_{\text{GAN}}$

Models	Normalized (% MOBIL)	Mean Reward
Agent$_{\text{IDM}}$	5.21%	-22.33 ± 100.66
Agent$_{\text{GAN}}$	114.82%	33.62 ± 95.19

The results show that RL agents trained on GAN-based traffic have significantly better generalization capabilities compared to agents trained on rule-based traffic simulators.

To have a fair comparison between the trained agents, The agents have been tested in 2 different types of environments together as shown in Tables 3.1 and 3.2. The first environment, Traffic$_{\text{IDM}}$, is a static environment where the other actors in the environment do not make complex decisions such as changing their lanes. The second environment Traffic$_{\text{GAN}}$, is based on the non-deterministic trajectory generator network where other agents acts in a similar way with real traffic scenarios, which can cause them to make unnecessary decisions. Two agents have been compared at the same time with the MOBIL in the Traffic$_{\text{GAN}}$ in order to have a fair comparison between two agents. The results in Table 3.2 have been obtained after 1000 sample simulations.

According to Table 3.1; in a relatively certain and non-complex traffic, even though Agent$_{\text{IDM}}$ has been tested in the environment that it has been trained, it stays behind of the Agent$_{\text{GAN}}$. Also Agent$_{\text{GAN}}$ does relatively good considering it hasn't been trained on the same environment. According to Table 3.2; in a complex and uncertain traffic, Agent$_{\text{IDM}}$ obtains less rewards than the MOBIL algorithm since it hasn't been tested in the environment that it has been trained. Agent$_{\text{GAN}}$ does better than MOBIL and Agent$_{\text{IDM}}$ since it has observed the uncertain and faulty behavior situations during its training phase.

From the Tables 3.1 and 3.2, It can be claimed that an RL agent that has been trained in a static non-complex environment can not learn the underlying dynamics and can not adapt to uncertainty of the real-world applications where surrounding vehicles make complex or faulty decisions such as lane changing, instant-acceleration or instant-slowing. In these type of environments, an agent that has been trained on complex and uncertain scenarios can avoid crashes and make safer decisions.

3.1.2 Conclusion

With this study, an agent was trained with the curriculum strategy with a partial change of difficulty. The training process has been taken to environments with higher difficulties compared to the target environment. Thus, the usefulness of the concepts of achieving faster convergence with an increase in forward difficulty (curriculum) has been proven, as well as the fact that the agent trained in more difficult tasks than the target will be more successful in the target task (reverse-curriculum).

3.2 Investigating Value of Curriculum Reinforcement Learning in Autonomous Driving Under Diverse Road and Weather Conditions

The main contribution of this work is an in-depth study on impact of different curricula on deep reinforcement learning for autonomous driving in a realistic driving simulator. We develop a structured environment, where the adversity of weather conditions and road complexity can be tuned independent from each other. We setup several different curricula, where the training starts from simple weather conditions and road geometries, and then ramps up to more complex road and weather conditions. Evaluation results show that agents trained using curriculum reaches superior performance using much lesser samples, compared to agents that are directly trained in complex environments.

All training and evaluation procedures applied in this work are implemented in "SimStar" simulation environment. In the environment, custom roads with various traffic, track, and weather conditions can be generated easily for more realistic-comprehensive training and evaluation of RL agents. This engine is responsible for 3D visualization of the environment and creating accurate vehicle dynamics. Different types of vehicles (sedan, SUV, truck etc.) can be added to the environment. Road conditions such as tar, dirt, damage can be implemented on the tracks to accurately resemble real-world counterparts. Details about the environment are given in Appendice J.

We describe the details regarding curriculum implementation in this section. All agents are trained with the Soft Actor Critic (SAC) [60] algorithm in different scenarios and conditions. In this context, the agent is firstly trained exclusively in different weather conditions, which are: "clear", "rainy" and "snowy".

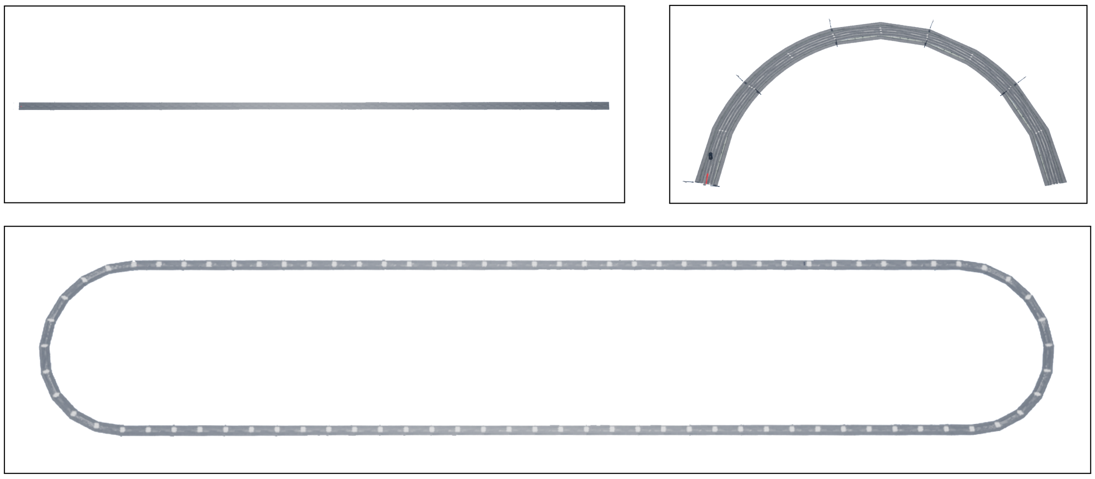


Figure 3.4 : Road geometries used in simulation setup.

Another curriculum parameter is the geometry complexity of the track. The track that has the most complex geometry is the hardest track to be learned by the agent. All roads in the simulation are 10 meters wide and consist of 2 different lanes. Three tracks in total are used. As it can be seen on Figure 3.4, they are;

- Straight Road
- U-Turn Road
- Complete Circuit Race Track

To increase the variety of the curriculum learning, weather conditions are added onto these tracks for more realistic settings. The agents which are trained on different weather conditions (such as rain and snow) would learn unique driving capabilities. Note that clear, rainy, and snowy weather conditions are chosen in their maximum value for noteworthy atmospheric ambiances and their effects on the tracks. As a consequence, the curriculum reinforcement learning methodology is implemented on "Weather Condition" and "Track Type" combinations.

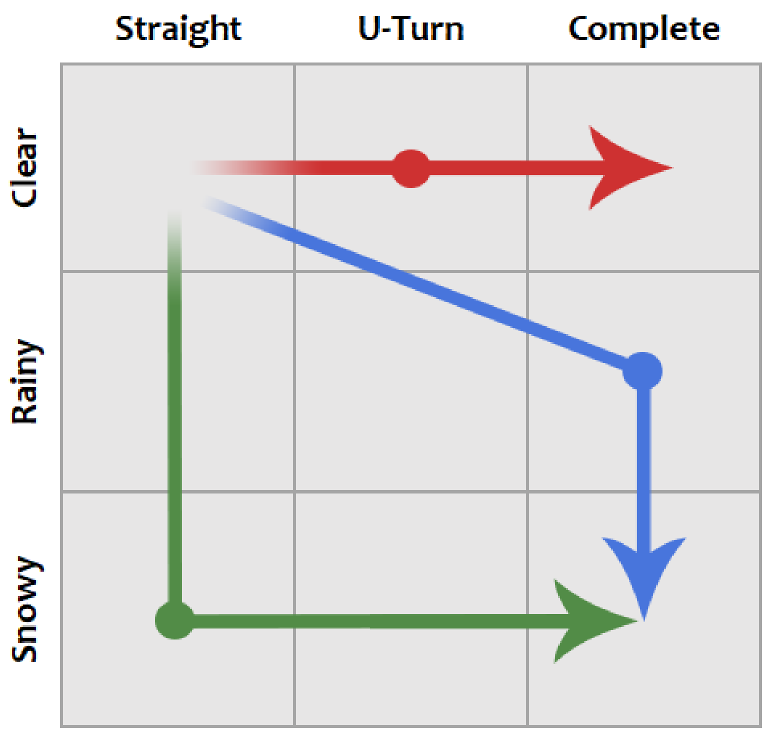


Figure 3.5 : Three example hand-made curriculum scenarios.

Each variable set of curriculum training scenarios can be evaluated as separate dimensions in a space. In this study, it can be concluded that there will be 2 dimensions in the curriculum space. The transition process after each curriculum training phase will take place in the form of a transition from one point in the space to another. Figure 3.5 can be examined as a representation of the mentioned space and transitions.

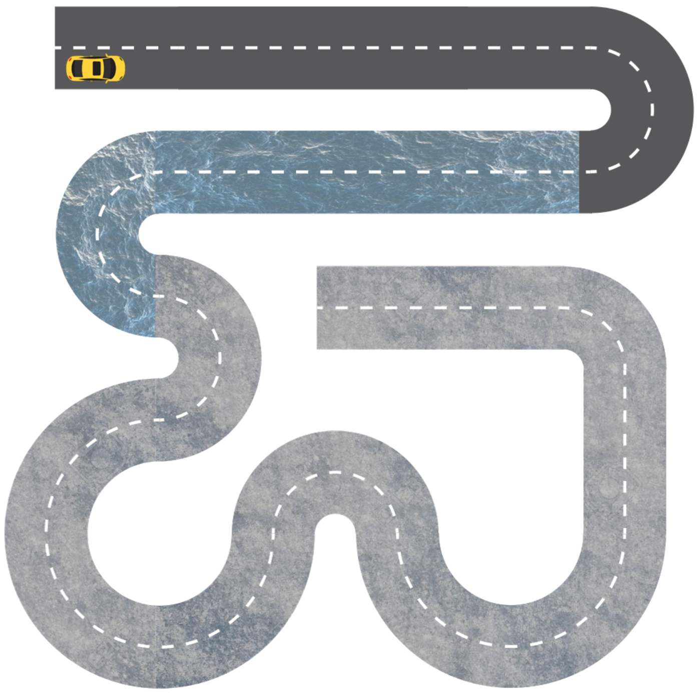


Figure 3.6 : Representation of a full curriculum training scenario.

An example and a complete curriculum scenario can be seen on Figure 3.6. The agent observes the phases (Straight, Clean), (U-turn, Clean), (Straight, Rainy), (U-turn, Rainy), (Complete, Snowy) respectively. In the example design, the agent is trained only 1 iteration on each phase and transferred to the other phase immediately. In the variations to be presented in the context of the proposed methodology, these iteration numbers will be in thousands. Agents can be thought of as spawning back to the beginning of the current phase at the end of each iteration. When the iteration number of each phase is exceeded, the transition to the next phase will be executed.

Table 3.3 : Training Combinations

Curriculum Scenarios	Phase 1	Phase 2	Phase 3
Scenario 1	Straight (C)	U-Turn (C)	Race Track (C)
Scenario 2	U-Turn (R)	Race Track (R)	Race Track (C)
Scenario 3	Straight (R)	Race Track (R)	Race Track (S)
Scenario 4	Straight (C)	Race Track (R)	Race Track (S)
Scenario 5	Straight (C)	Straight (S)	Race Track (S)

The complete information about training scenarios can be found in Table 3.3. A total of 5 different curriculum scenarios are implemented. Letters C, R and S corresponds to clear, rainy and snowy weather conditions respectively. These route orders are determined by examining the training results without curriculum learning. Most of the cases, the routes are chosen to be ordered from easiest scenarios to hardest. The order of curriculum learning is crucial as the agent has to acquire useful experience in the consecutive tasks [61]. It is seen that, the hardest scenario for the RL agent to solve is complete track scenario with snowy weather condition. So, the goal is to train the agent starting from the easiest track to be learned to the hardest track. To do so, specific training orders in Table 3.3 are determined where agent starts from the simple environment and finishes at more complex one. By doing so, the agent will gather convenient experiences throughout the ordered training tracks.

The curriculum training order for each route is illustrated with ascending enumeration. As stated in [61], the quality of the curriculum learning task is directly dependent on the training order quality of the tracks.

3.2.1 Results

At first, the non-curriculum training procedures are carried out on 9 base tracks (3 weather scenarios for 3 road types) as illustrated in Table 3.3. In baseline trainings, the iteration limits are set according to track difficulty levels. The straight road scenarios are trained for 75000 iterations, U-Turn scenarios are trained for 50000 iterations and the Circuit Race Track scenarios are trained for 200000 iterations. After the baseline non-curriculum trainings, a 200000 iteration limit is decided for the training of curriculum scenarios. In each of 5 different curriculum learning scenarios, 3 different road and weather combinations are used. The first step of all of these curriculum scenarios is one of the baseline trainings that are conducted.

In the Figure 3.7, it is possible to see the complete picture of the curriculum learning results. There are 5 rows in the figure and each one of them represents a separate curriculum scenario. Each training phase executed with 3 different pre-defined seeds for the sake of stability. The left side of the figure shows the ending point of the curriculum scenario and right side shows the baseline and curriculum training results. The red lines in the right side shows the baseline training results for that particular curriculum scenario and the black one shows the curriculum training results. Each weather condition is represented with a different color. The clear weather is shown with green, the rainy weather is shown with blue and the snowy weather is shown with

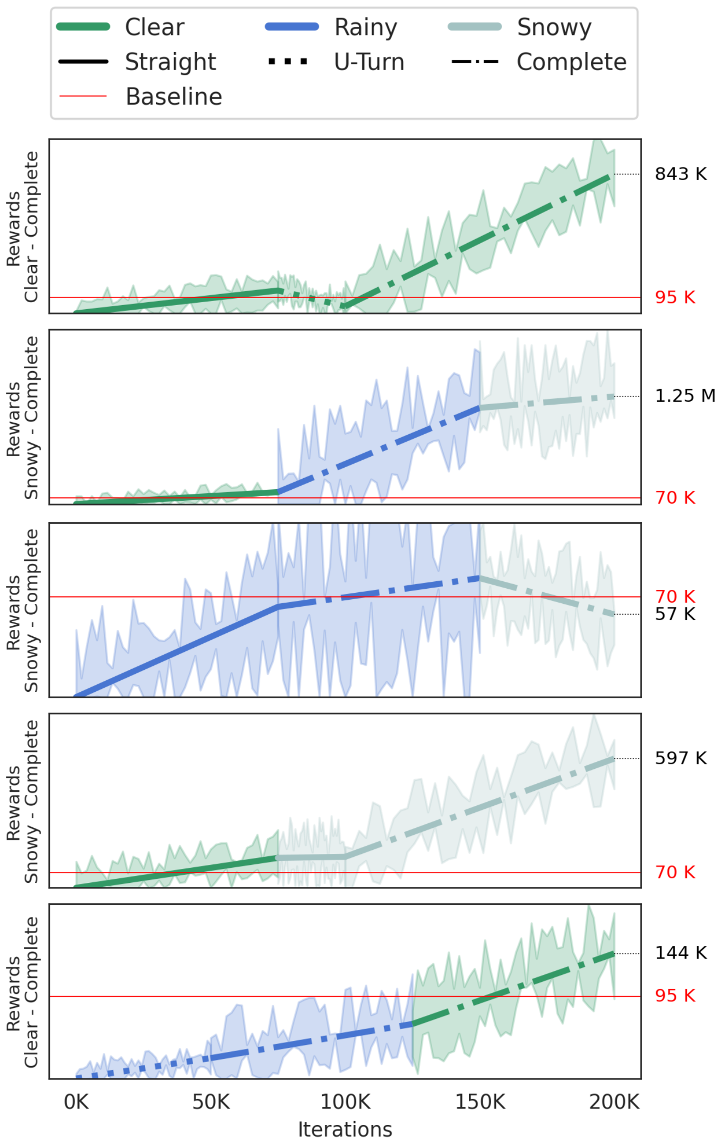


Figure 3.7 : Comparative curriculum learning results.

celeste color. In the same fashion, all road types are shown with different markers and they can be seen in the upper part of the Figure 3.7.

Up until this point, it is shown that the curriculum learning significantly boosts the performance of the trained RL agents and aids them to learn the underlying dynamics of the different weather and road combinations. In addition to this advantage, it is observed that curriculum learning also decreases the training times significantly.

The best results are achieved in a curriculum scenario where the best performing non-curriculum agent used directly in the highest complexity environment. In nearly all of the results, the curriculum learning yielded greater results compared with the non-curriculum results except one case. Using a bad performing non-curriculum agent on a curriculum learning scenario severely hurt the training process and decreased the reward compared with the non-curriculum scenario. This shows that it is crucial to use a good baseline agent in all curricula to achieve better results. The results are affected more by the weather changes than the road complexity.

3.2.2 Conclusion

In this work, it is showed that training a deep reinforcement learning agent with curriculum learning strategy increases the performance and decreases the overall training time for an autonomous driving agent trained on different weather and road conditions. These results are concluded by developing a structured reinforcement learning system with different road types and weather conditions. The curriculum scenarios on this work consist of different road geometries and weather combinations. It is illustrated that training an RL agent on the relatively simple environment then continuing the the training process of this agent in a more complex environment resulted in a performance boost. During the training process, all parameters kept constant in order to make sure the validity of the experiments. The experimental results demonstrated that an RL agent trained with a curriculum learning structure performed significantly better than an RL agent trained from scratch without a curriculum approach.

4. PROPOSED ALGORITHM

The proposed algorithm aims to automate the creation phase of curriculum learning scenarios. By training the agent in defined environment states separately, it makes a relative difficulty difference estimation between states and creates the curriculum-route where the reinforcement model can learn the target with minimal effort. The algorithm runs an iterative optimization process using the directed-graph structure and Dijkstra's algorithm.

4.1 Step-by-Step Explanation of Algorithm

4.1.1 Prerequisites for the algorithm

In order for the algorithm to work, parameters that significantly change the operation of the target environment must first be created. Then, a probability set should be defined for these parameters. For the sake of exemplary explanation; A parametrized environment named "DummyEnv" will be considered. It will be assumed that DummyEnv has 2 environment variables named "A" and "B". These variables must affect parameters that greatly affect the model's ease of completing tasks in the environment. The parameters are given in a mixed order in order to indicate the sorting feature of the proposed algorithm.

Table 4.1 : Parameter combinations for DummyEnv

A	B
30	8
30	5
30	10
50	8
50	5
50	10
40	8
40	5
40	10

4.1.2 Determining the direction of difficulty

The algorithm tries to understand whether the difficulty of the environment increases or decreases according to the change of the value for each variable separately. In order to do this, a user-specified length of training (preferably short enough to understand the solvability of the environment) is performed in all parameter combinations. It will be assumed that training takes place on the parameter combinations in Table 4.1 and the resultant rewards in Table 4.2 are received. Rewards are the average of the episodic rewards received as a result of the last *n* episode in the training process.

Table 4.2 : Training results of each parameter combination for DummyEnv

A	B	Reward
30	8	80
30	5	65
30	10	100
50	8	100
50	5	75
50	10	120
40	8	70
40	5	70
40	10	105

After the individual reward results for each combination are collected, the algorithm locks each parameter separately.

Table 4.3 : Mean rewards for parameter 'A'

A	Mean Reward
30	81.67
50	98.34
40	83.33

Table 4.4 : Mean rewards for parameter 'B'

A	Mean Reward
8	85
5	66.67
10	108.33

After the mean rewards are calculated, the relationship between the parameter values in the context of difficulty is derived at a basic level. The algorithm orders the parameter values according to their mean rewards from the highest to the lowest (from easiest to hardest). The ordered states of the parameters in the given example would be as follows:

- A $\rightarrow$ [50 40 30]
- B $\rightarrow$ [10 8 5]

4.1.3 Differential evaluation

It will be assumed that the target task is to solve the DummyEnv on "the most difficult" parameters. Target states can be specified in the algorithm, this is valid for the given example. Each parameter will change its values from easy to difficult with their index numbers. This also provides convenience in terms of representation and data structure management. The new forms of values will be displayed as follows:

- A $\rightarrow$ [50 40 30] $\rightarrow$ [0 1 2]
- B $\rightarrow$ [10 8 5] $\rightarrow$ [0 1 2]

From now on, each combination will be shown with its index numbers (e.g. combination $[A:50, B:5]$ will be shown as $[0,2]$). The next step is to model the combinations with difficulty information as a graph in order to create curriculum scenarios. For ease of representation, two values of each parameter will be considered.

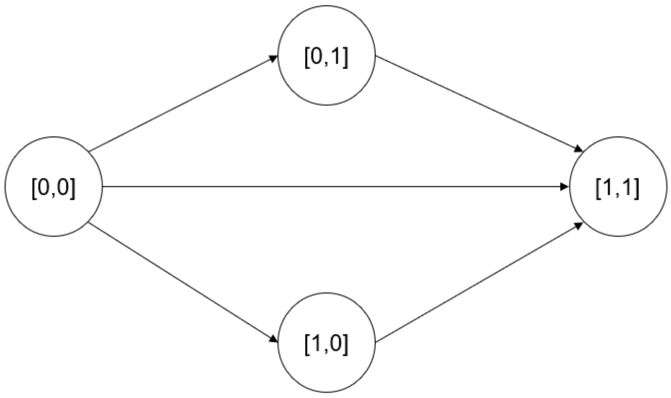


Figure 4.1 : Curriculum graph for DummyEnv

Curriculum graphs (as in Figure 4.1) are being created with some limitations, as Dijkstra's algorithm can cause problems with graphs containing cycles. These limitations are as follows:

- Paths leading from the current node to the more difficult node are ignored
- In order to avoid a cycle, a scenario flow can only be from easy to difficult

As it can be seen in Figure 4.1, a probabilistic flow from easy to difficult combinations has been established according to the mentioned limitations. Let's assume that for the parameter combinations shown in Figure 4.1, k evaluation episodes with a fixed seed is run. The values shown in Table x will be assumed to be the run evaluation results for the mentioned combinations.

Table 4.5 : Mean evaluation rewards for parameter combinations

Combination	Evaluation Reward
$[0,0]$	100
$[0,1]$	80
$[1,0]$	75
$[1,1]$	40

After this phase; the model trained in each combination will be evaluated in the combinations they can reach with the edges from their state node. The weight of the edge between these two combination nodes should represent the difference in difficulty between the two combinations. Let's assume that the function named $Eval(M,x)$ returns the evaluation reward of the specified reinforcement learning model in the specified combination, given the reinforcement learning model is called M and the environment parameter combination is called x. Assume that the model is trained and evaluated in environment a, and in one of the environments it can go to in the curriculum scenario is environment b. The negative change in evaluation reward will be an indirect metric indicating how much the model is being challenged. The algorithm sets the weight of the edge between nodes representing combinations of a and b as in equation 4.1:

$$W_{a \to b} = Eval(M,a) - Eval(M,b) \tag{4.1}$$

Suppose a situation like:

- Model trained in combination $[0,0]$.
- Model evaluated in combination $[0,0]$, mean evaluation reward is 100
- Model evaluated in combination $[0,1]$, mean evaluation reward is 65

If the above cases are known, the edge weight can be calculated according to the equation 4.1 and the relevant curriculum graph can be filled with the calculated value:

$$W_{[0,0]\rightarrow[0,1]} = 100 - 65 = 35 \quad \textbf{(4.2)}$$

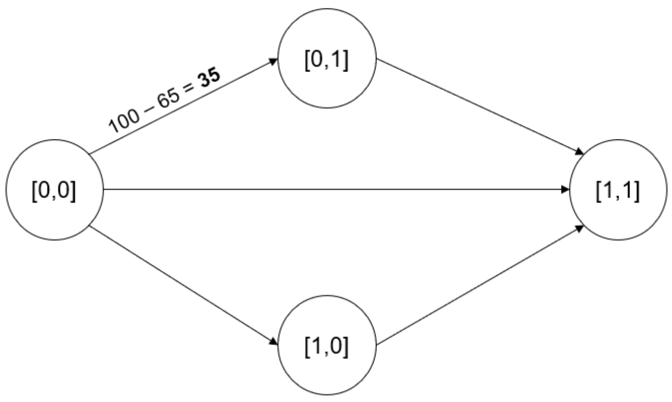


Figure 4.2 : Placement of edge weight for the curriculum graph of DummyEnv

When equation 4.1 is applied for all edges, it will be seen that the result in Figure 4.3 is obtained:

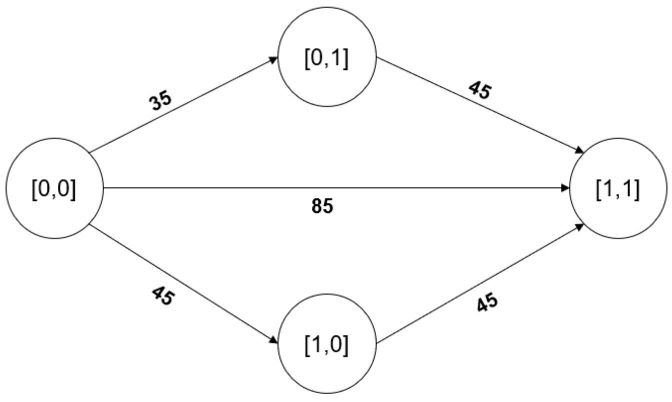


Figure 4.3 : All edge weights of the curriculum graph of DummyEnv

4.1.4 Setting up the curriculum path

The task of finding the optimal path in the graph is approached as a shortest path problem. In fact, the edge weights between nodes indicate how much the model is being challenged when moving in the specified direction between each pair of nodes. Therefore, the model that goes from the specific combination to the target combination with the least cumulative edge weight, theoretically learns the target environment with the least challenge.

If we express a curriculum learning scenario that includes points a, b, and c respectively as $C_{a \to b \to c}$; computing the cumulative challenge of all possible curriculum scenarios based on the edge weights in Figure 4.2 yields the following results:

$$C_{[0,0] \to [0,1] \to [1,1]} = 35 + 45 = 80 \tag{4.3}$$

$$C_{[0,0] \to [1,1]} = 85 \tag{4.4}$$

$$C_{[0,0] \to [1,0] \to [1,1]} = 45 + 45 = 90 \tag{4.5}$$

The same results will be obtained when the paths calculated with the naive method as an example are calculated with Dijkstra's algorithm and it will be seen that the shortest path is $C_{[0,0] \to [0,1] \to [1,1]}$:

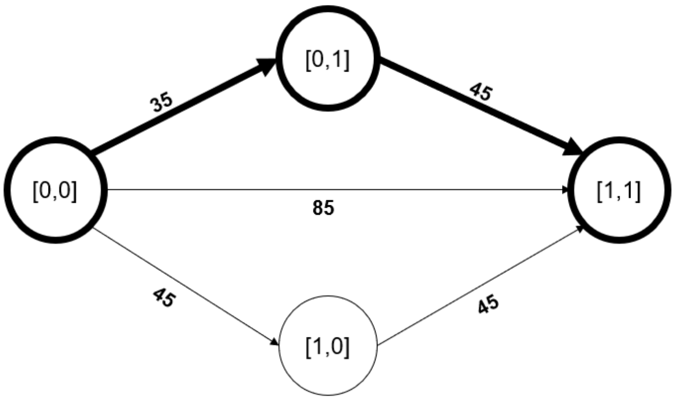


Figure 4.4 : The shortest path for the curriculum graph of DummyEnv

4.1.5 Executing the training on the specified curriculum path

A naive and bilateral conditioning process has been determined for the education on the determined curriculum path. If the vanilla training processes of the model in the first stage take a maximum of k iterations and the selected environment reaches R reward means that the environment has been learned by the model; The trigger to move to the next phase in curriculum training is defined as follows:

- If the current curriculum phase already took k iterations
- If the model reached R mean reward in the current environment parameter combination

4.1.6 Comparing the methodology with vanilla training

Since k training iterations can be expected for each curriculum stage in the worst conditions, a training up to nk iteration length can be encountered in an n-phase curriculum training. In such a case, the existing k iteration-trained vanilla models will not work for a sound comparison. Let's assume that the curriculum learning process takes t iterations and t is defined as $0 < t < nk$. If a separate model is trained from-scratch throughout t iterations in the environment with target parameter combination, it can be compared with the curriculum learning model.

At the end of the process, the two models can be compared with the mean rewards of the evaluation rounds started with the same seed and the instant training reward differences with each other on a specific training iteration.

4.2 Complexity of the Proposed Algorithm

We can think of the algorithm in two different ways, with training and evaluation steps in terms of the operations it performs. Assume that there are n environment parameters fed to the algorithm and each parameter can have k different values. Considering the training stages, the algorithm includes:

- For determining the direction of difficulty: k^n trainings
- For evaluating each model on other parameter combinations: $(k^n) \times (k^n - 1)$ evaluations

Thus, the exact complexity for the proposed algorithm would be:

$$O(k^n) \text{ trainings} + O(k^{2n}) \text{ evaluations} \quad \textbf{(4.6)}$$

Since training phase is known to take much longer than evaluation phase, the complexity of the algorithm can be described in its simplest form as $O(k^n)$. The complexity for the execution of Dijkstra's algorithm is ignored as it represents much lesser execution-time when it's compared to the training and evaluation phases.

4.3 Pseudo Code of the Proposed Algorithm

Algorithm 2: Proposed Algorithm

```
$P$ = Available environment parameters
$S_c$ = Parameter combination space
for each parameter combination $c \in S_c$ do
    calculate vanilla training reward $R_c$ for $c$
end
for each parameter $p \in P$ do
    Calculate mean vanilla training reward $M_p$ for $p$
end
Sort each $p$ in $P$ according to their respective mean rewards
Create a graph $G$
Add each $c \in S_c$ as nodes to $G$
for each parameter $c \in S_c$ do
    for each parameter $c_2 \in S_c$ do
        if $c_2$ occurs later than $c$ in the sorted reward array then
            $R_c \leftarrow$ Evaluation reward of the agent trained in $c$ at $c$
            $R_{c_2} \leftarrow$ Evaluation reward of the agent trained in $c$ at $c_2$
            Add edge $c \rightarrow c_2$ to $G$ with weight $R_c - R_{c_2}$
        end
    end
end
Find the shortest path from the easiest parameter combination to the target
  combination $T_s$ in $G$
return $T_s$
```

5. TESTING ENVIRONMENT

The hardware and software details of the environment in which the algorithm results are tested are given in Appendix C1 and Appendix C2, respectively.

5.1 Infrastructure

The tests of the proposed algorithm were performed in various gaming and physics simulation environments created with OpenAI's Gym [62] environment manager. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It contains the known problems in the field and the most popular titles of the gaming world, as seen in Figure 5.2.

5.1.1 Functions

Environments created in the Gym are expressed as a class object. Each object has the following functions.

5.1.1.1 Render

It is the command to get instant visual or comprehensible output from the environment. It can be modified, turned on and off as desired, and it is very useful for debugging, especially in environments with physics simulation.

5.1.1.2 Step

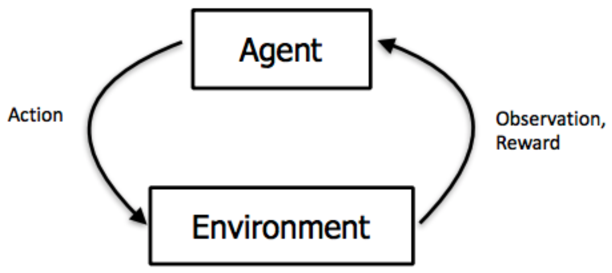


Figure 5.1 : The agent-environment loop in Gym [63]

It is the command that is used to take the action specified as input in the environment and to request post-action returns. It does not have a customizability, it progresses the environment simulation by a time iteration each time it is called. The amount of progress can be shaped and changed according to the structure of the code written by the user.

The function has a design which basically fits the default interaction model (in Figure 5.1) in reinforcement learning. The function transmits the action from the model to the environment, and as a result, the values returned from the environment are being transferred to the model. The model uses these returned values to determine its next action.

5.1.1.3 Reset

This is the command used to reinitialize the environment and return all environment variables to their defaults. It can be triggered manually or at the termination phase of the episode.

5.1.2 Observations

The values returned from "step" function have a tuple format. This tuple object contains more than one information about the environment.

5.1.2.1 Observation

An environment-specific object that represents your observation of the environment. Sensor data for environments using sensors can be a sample frame for environments using cameras.

5.1.2.2 Reward

The reward or penalty given to the model by the environment in response to the action taken just before in the environment. Value ranges may vary from environment to environment, but the model should always tend to get more rewards.

5.1.2.3 Done

It is a variable that gives information about whether the environment has terminated after the action just taken in the environment. According to this variable, the environment is restarted or the agent routine can be terminated. Since the terminal states of some environments are not defined, done may never get the value "True".

5.1.2.4 Info

It is a variable that has a lot of functionality when debugging is desired on the environment. Additional information that does not affect the decision process of the environment can be transferred and logged over this variable. However, since it will be considered as a data leak, it is recommended not to use the values here during the training process.

5.1.3 Spaces

Values defined in Gym environments can be expressed in two different types with their intervals.

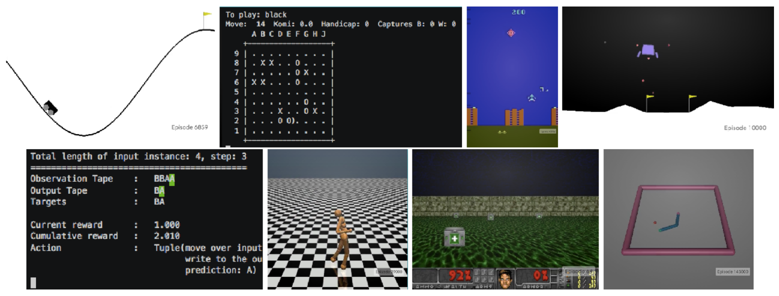
5.1.3.1 Discrete

For this value type, values are expressed with non-negative numbers. Thus, if there are 2 different actions that can be taken in the environment, the values of the action space will be 0 or 1 within the scope of this definition.

5.1.3.2 Box

In this value type, the value space is considered in an n-dimensional box. Thus, each value in the box can be defined separately with values between $-inf$ and inf. It is generally preferred for defining observation spaces. But it is also used to define the action set in continuous-action situations where more than one action can be taken at the same time in the environment.

5.2 Environments



```
To play: black
Move:  14  Komi: 0.0  Handicap: 0  Captures B: 0 W: 0
    A B C D E F G H J
  +-------------------+
9 | . . . . . . . . . |
8 | . X X . . O . . . |
7 | . . . . . O X . . |
6 | X X . . . O . . . |
5 | . . . . . . . . . |
4 | . . . . . . O . . |
3 | . . . X . . O X . |
2 | . . . O O). . . . |
1 | . . . . . . . . . |
  +-------------------+
```

```
Total length of input instance: 4, step: 3
==========================================
Observation Tape    :   BBAA
Output Tape         :   BA
Targets             :   BA

Current reward      :   1.000
Cumulative reward   :   2.010
Action              :   Tuple(move over input
                              write to the ou
                              prediction: A)
```

Figure 5.2 : Example environments in Gym. [62]

Gym provides a diverse set of environments, from simple to difficult, involving many types of data. Environments aimed at completing small goals and testing the basic principles of reinforcement learning, environments aiming to learn algorithm-based planning processes only from examples, Atari game environments with visual observation format, robotic environments with 2D and 3D dimensional physics simulation characteristics are all available within the Gym package.

The environments used to test the algorithm within the scope of this study are as follows:

- Pendulum-v0
- LunarLander-v2
- LunarLander-v2-Continuous
- BipedalWalker-v3
- Acrobot-v1
- Breakout-v0

5.2.1 Pendulum-v0

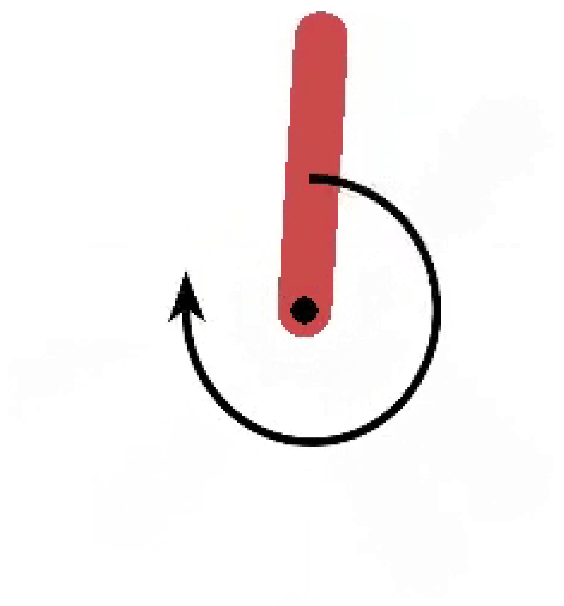


Figure 5.3 : Pendulum-v0 Environment

In this environment, the inverted pendulum problem, which is a well-known problem in the control literature is modeled. At the beginning of the environment, the pendulum starts in a random position. The purpose of the environment is to bring the pendulum to a position that will face upward and remain upright within the specified iteration limit.

Due to the variables such as the gravity of the environment, the weight of the pendulum and the magnitude of the applied force, it is not possible to reach the target with a single movement in the environment configuration with the default values. Therefore, the model needs to learn to plan to create the sling effect by accelerating the rod in opposite directions.

5.2.1.1 Rewards

The reward decreases in proportional with the increase in the angle value away from 90 degrees, the rod speed and the applied force. Positive reward is not defined, so the model will try to get the highest negative reward possible. The goal of the reward is to prioritize keeping the pendulum upright at 90 degrees with minimal force and acceleration.

5.2.1.2 Observation Space

Assuming that the angle the pendulum makes with the plane parallel to the ground is θ, the observation variables are as follows:

- $cos(\theta)$
- $sin(\theta)$
- $\dot{\theta}$

5.2.1.3 Action Space

As an action, a continuous torque value is taken and clipped by the environment at a certain interval. The sign of the torque value determines the direction in which the torque will be applied.

5.2.1.4 Parametrized Variables

In order to test the proposed algorithm, the following variables of the environment have been parameterized and made changeable:

- g: The gravitational acceleration of the environment
- m: The mass of the pendulum

5.2.2 LunarLander-v2

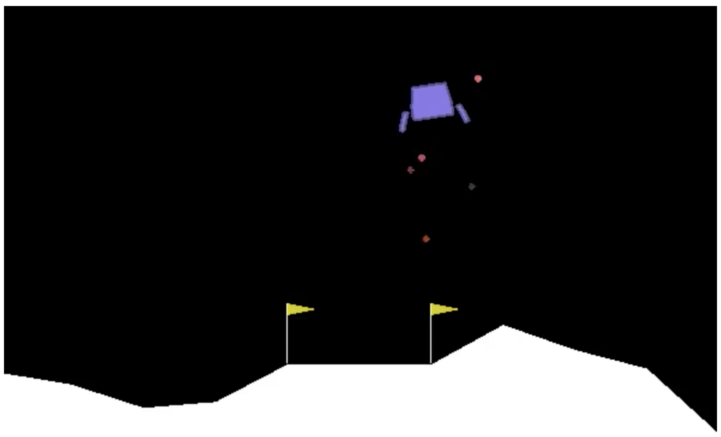


Figure 5.4 : LunarLander-v2 Environment

In this environment, the problem of a space shuttle landing on the lunar surface is modeled. At the beginning of the environment, the space shuttle is initialized at a constant height with acceleration in random direction and magnitude. The purpose of the environment is to land the shuttle at the landing zone, which is the area indicated by the flags. Smooth landings outside the landing area are also considered successful, but they reward the agent less.

The problem in the environment is how to land in the target area by running which engine and how much power at various angles. To support solving this exploration problem, the agent is launched into the environment with random acceleration and rotation.

5.2.2.1 Rewards

- Zeroing the speed after being properly placed in the landing area from the starting point: $[100-140]$
- Accident situation: -100
- Rest situation: $+100$
- Each leg ground contact: $+10$
- Firing main engine: -0.3 per iteration
- Solved: $+200$

5.2.2.2 Observation Space

The observation variables are as follows:

- Normalized position at x
- Normalized position at y
- Normalized velocity at x
- Normalized velocity at y
- Angle θ of the shuttle
- Angular velocity ω of the shuttle
- Contact status of each leg of the shuttle

5.2.2.3 Action Space

The shuttle cannot run more than one engine at the same time. The agent can take action to run only one of the right engine, left engine and bottom thrust main engine at full power at a time.

5.2.2.4 Parametrized Variables

In order to test the proposed algorithm, the following variables of the environment have been parameterized and made changeable:

- *g*: The gravitational acceleration of the environment
- INITIAL_RANDOM: Magnitude of the initial random acceleration

5.2.3 LunarLander-v2-Continuous

This environment is the continuous-action type of the environment in Section 5.2.2.

5.2.3.1 Action Space

The shuttle can run more than one engine. The agent can take action to run the right engine, left engine and bottom thrust main engine at different desired power levels at the same time.

5.2.3.2 Parametrized Variables

As in Section 5.2.2, in order to test the proposed algorithm, the following variables of the environment have been parameterized and made changeable:

- *g*: The gravitational acceleration of the environment
- INITIAL_RANDOM: Magnitude of the initial random acceleration

5.2.4 BipedalWalker-v3

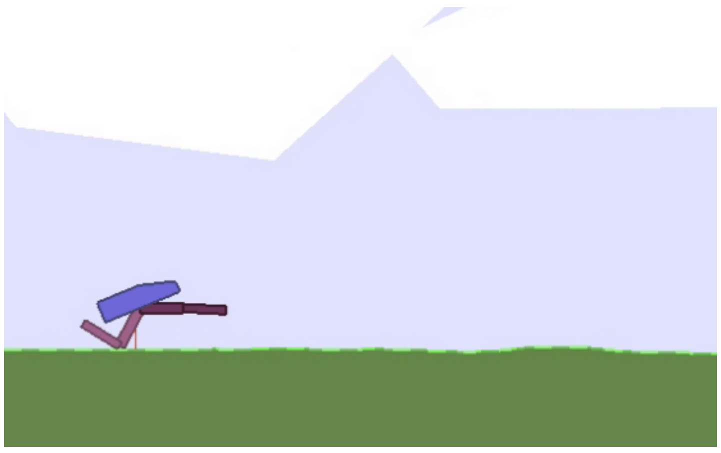


Figure 5.5 : BipedalWalker Environment

In this environment, a robot with 2 separate legs with 2 degrees of freedom is tried to reach the target. At the start of the environment, the robot spawns at the starting point with a random acceleration. By managing the foot actuators, it is expected to learn to walk the target distance in a certain iteration. It can be considered as a difficult problem to create a policy, as penalties or rewards are given only according to the distance traveled and the accident situation.

5.2.4.1 Rewards

- Walking up to the far end: $300+$
- Accident situation: -100
- Applied torque for each motor: -0.00035 per iteration

5.2.4.2 Observation Space

The observation variables are as follows:

- Hull angle θ
- Angular velocity ω of the hull
- Angles of each joint in each leg
- Contact status of each leg of the hull

5.2.4.3 Action Space

The environment has a continuous type of action scheme. Torque values to be supplied to all 4 motors at the same time are produced by the model.

5.2.4.4 Parametrized Variables

In order to test the proposed algorithm, the following variables of the environment have been parameterized and made changeable:

- g: The gravitational acceleration of the environment
- INITIAL_RANDOM: Magnitude of the initial random acceleration

5.2.5 Acrobot-v1

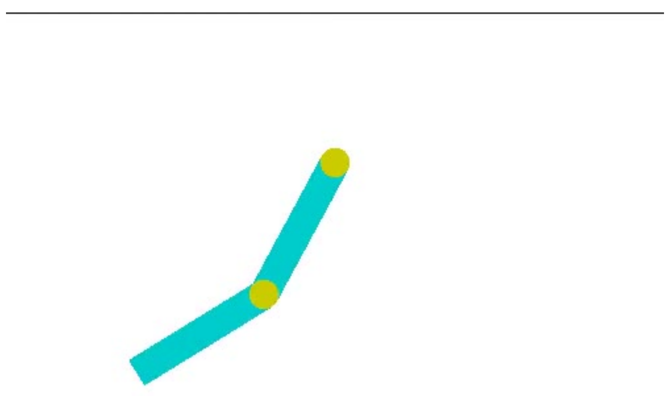


Figure 5.6 : Acrobot Environment

In this environment, there is an acrobot system with two joints and two links. The purpose of the environment is to raise the end link up from a certain height. At the start of the environment, both links point directly to the ground. Action is taken by applying torque to the first joint.

5.2.5.1 Rewards

- If the environment is not terminated: -1
- If the environment is terminated: 0

5.2.5.2 Observation Space

The observation variables are as follows:

- *sin* and *cos* of angle θ_1 of the joint 1
- *sin* and *cos* of angle θ_2 of the joint 2
- Angular velocity ω_1 and ω_2 of the joints

5.2.5.3 Action Space

The environment has a continuous type of action scheme. The amount of torque to be applied to the first joint can be specified as negative and positive magnitudes for both directions. It will be clipped for that the torque magnitude remains within predetermined limits.

5.2.5.4 Parametrized Variables

In order to test the proposed algorithm, the following variables of the environment have been parameterized and made changeable:

- t_{mult}: Torque multiplier
- m: The mass of the links

5.2.6 Breakout-v0

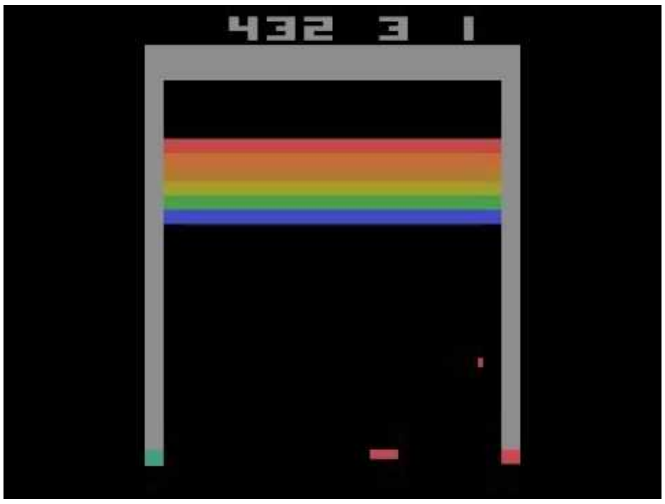


Figure 5.7 : Breakout Environment

This environment is a copied and adapted version of Breakout, an Atari 2600 game. The aim of the environment is to break all existing bricks by bouncing the ball without missing the ball. In cases where the ball is at equal speed or slower than the paddle, keeping the paddle in line with the ball may be the simplest heuristic solution. However, in order to make the problem more complex, the environment does not provide the ball and paddle position as a regular output.

5.2.6.1 Rewards

- For each brick destroyed from the top row to the bottom row: $[6,5,4,3,2,1]$

5.2.6.2 Observation Space

The observation variables are as follows:

- Snapshot of the game environment in $84x84$ grayscale format

5.2.6.3 Action Space

The environment has discrete action space. The paddle can only be moved to the right or left. No adjustment can be made regarding the speed.

5.2.6.4 Parametrized Variables

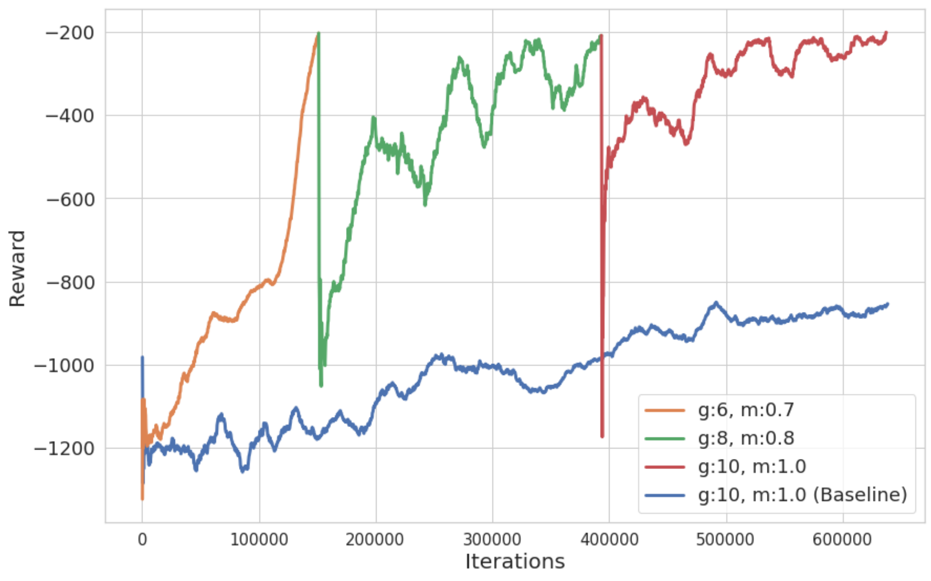
In order to test the proposed algorithm, the following variables of the environment have been parameterized and made changeable:

- Paddle Width
- Paddle Speed

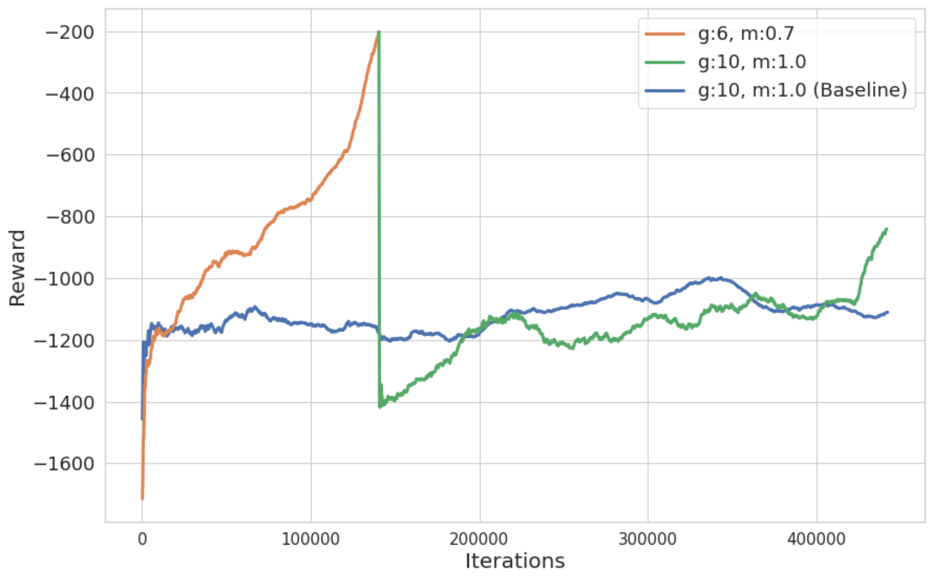
6. RESULTS

In this section, the results obtained by running the proposed algorithm on Gym environments introduced in Section 5.2 will be reported.

6.1 Pendulum-v0



(a) The best training result on Pendulum-v0.



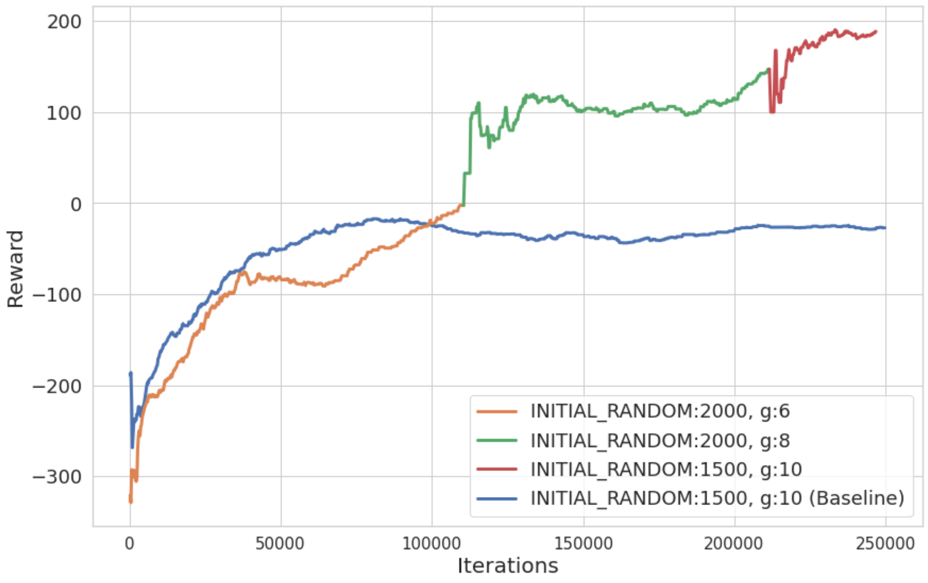
(b) The best training result on Pendulum-v0.

Figure 6.1 : The best (a) and the worst (b) samples from Pendulum-v0 results.

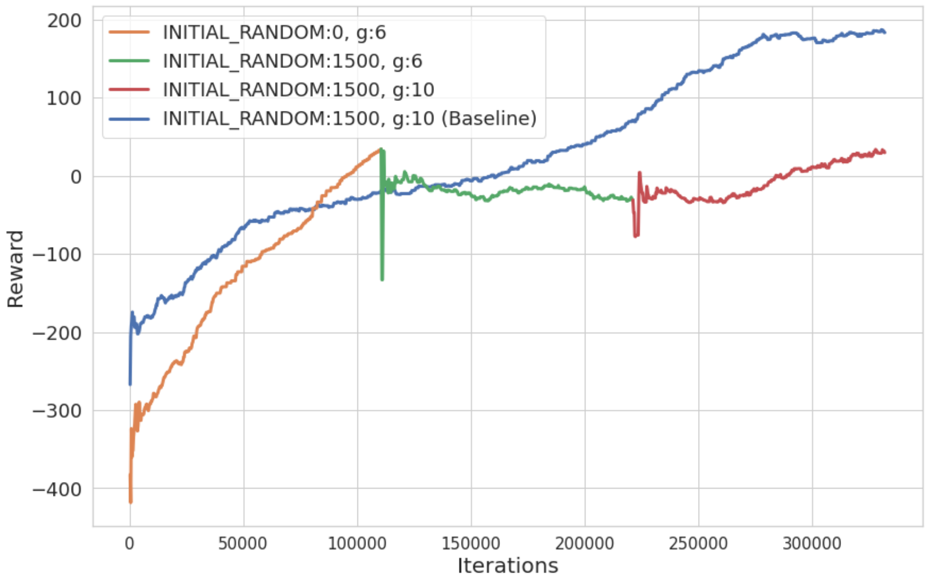
Table 6.1 : Average results for Pendulum-v0 environment

Property	Value
Target Reward	-200
Maximum Iterations	793948.65
After-Curriculum Reward	**-325.236**
After-Vanilla Reward	-1092.029

6.2 LunarLander-v2



(a) The best training result on LunarLander-v2.



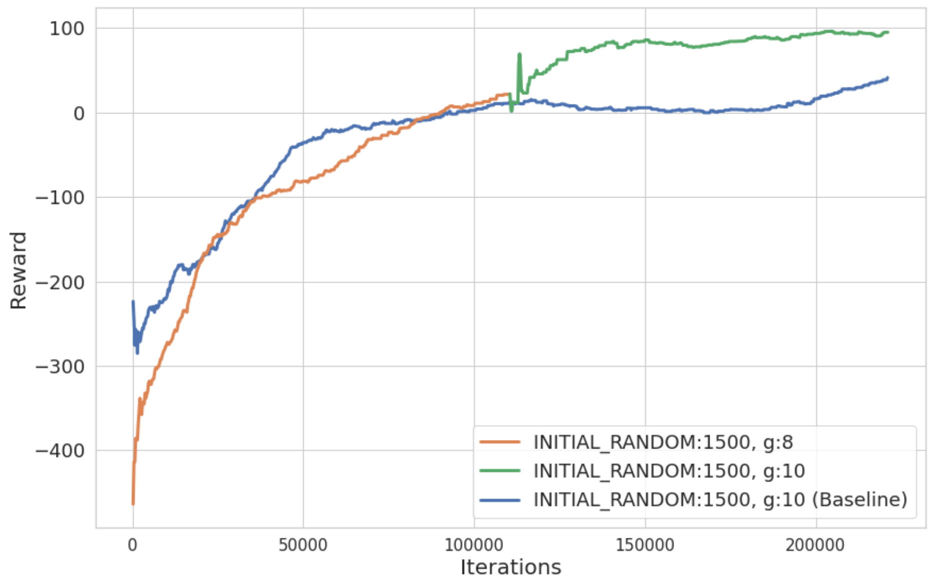
(b) The worst training result on LunarLander-v2.

Figure 6.2 : The best (a) and the worst (b) samples from LunarLander-v2 results.

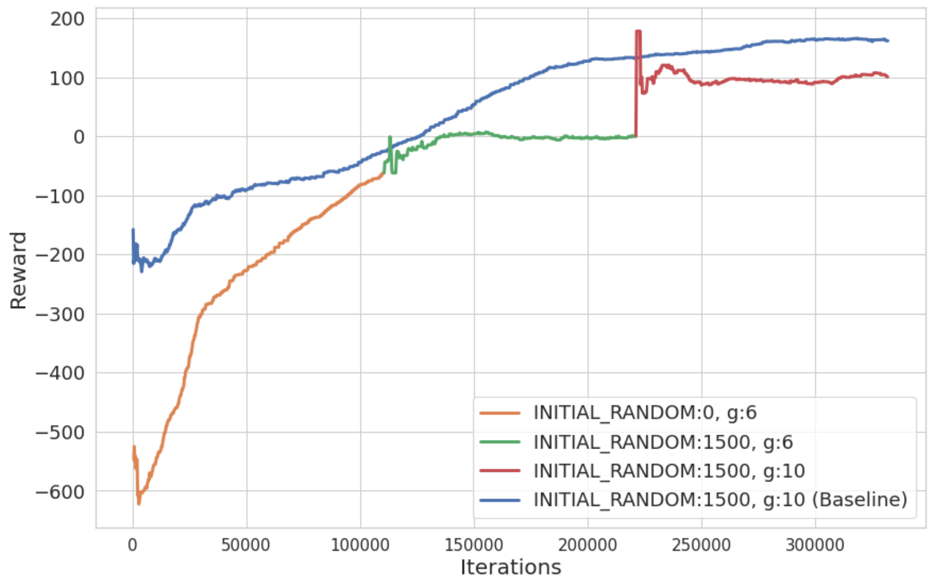
Table 6.2 : Average results for LunarLander-v2 environment

Property	Value
Target Reward	150
Maximum Iterations	353894.4
After-Curriculum Reward	**93.519**
After-Vanilla Reward	84.641

6.3 LunarLander-v2-Continuous



(a) The best training result on LunarLander-v2-C.

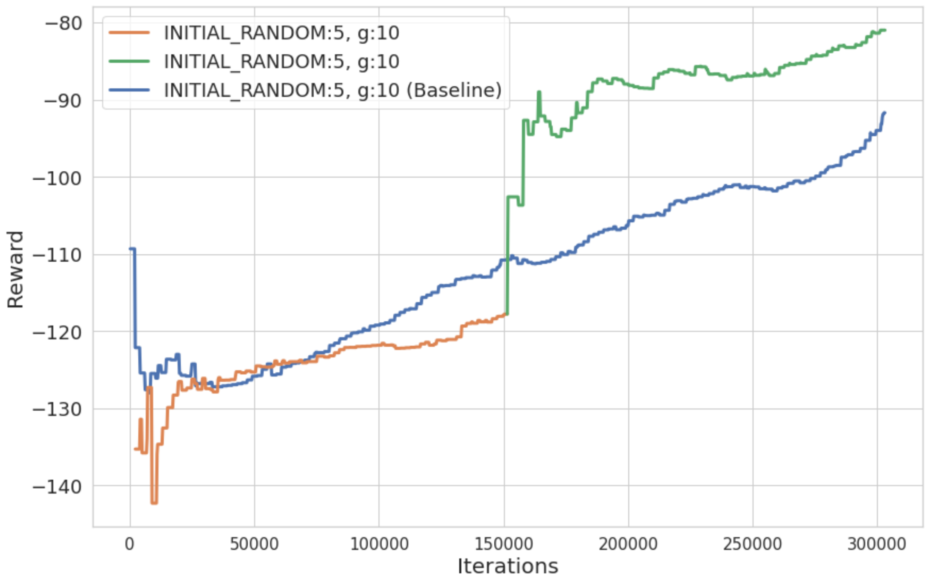


(b) The worst training result on LunarLander-v2-C.
Figure 6.3 : The best (a) and the worst (b) samples from LunarLander-v2-C results.

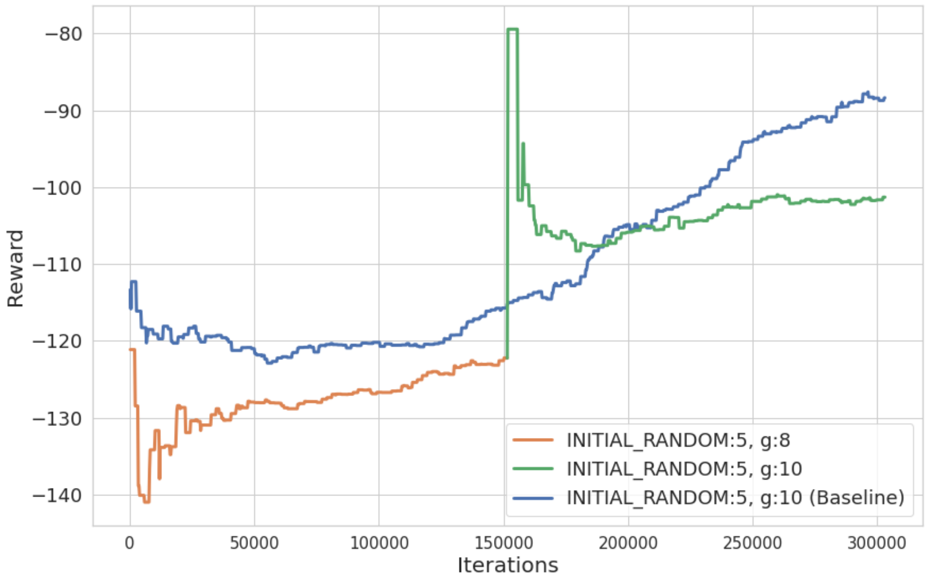
Table 6.3 : Average results for LunarLander-v2-Continuous environment

Property	Value
Target Reward	150
Maximum Iterations	309657.6
After-Curriculum Reward	**126.885**
After-Vanilla Reward	119.805

6.4 BipedalWalker-v3



(a) The best training result on BipedalWalker-v3.



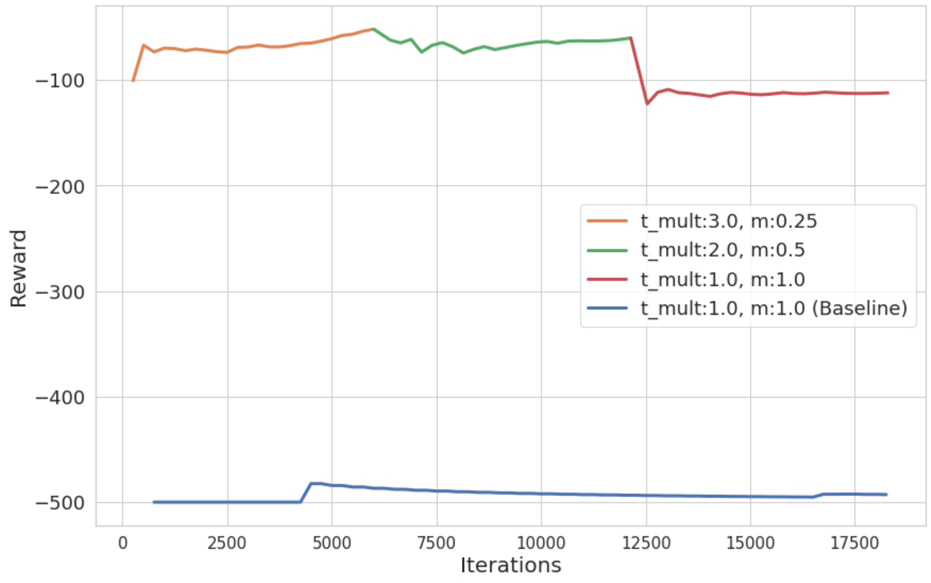
(b) The worst training result on BipedalWalker-v3.

Figure 6.4 : The best (a) and the worst (b) samples from BipedalWalker-v3 results.

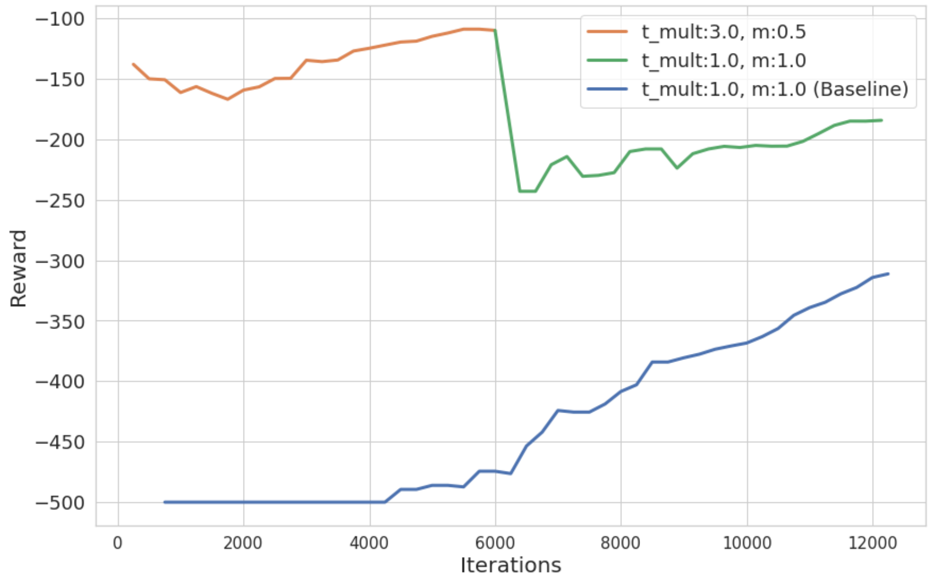
Table 6.4 : Average results for BipedalWalker environment

Property	Value
Target Reward	0
Maximum Iterations	346214.4
After-Curriculum Reward	-68.975
After-Vanilla Reward	**-50.408**

6.5 Acrobot-v1



(a) The best training result on Acrobot.



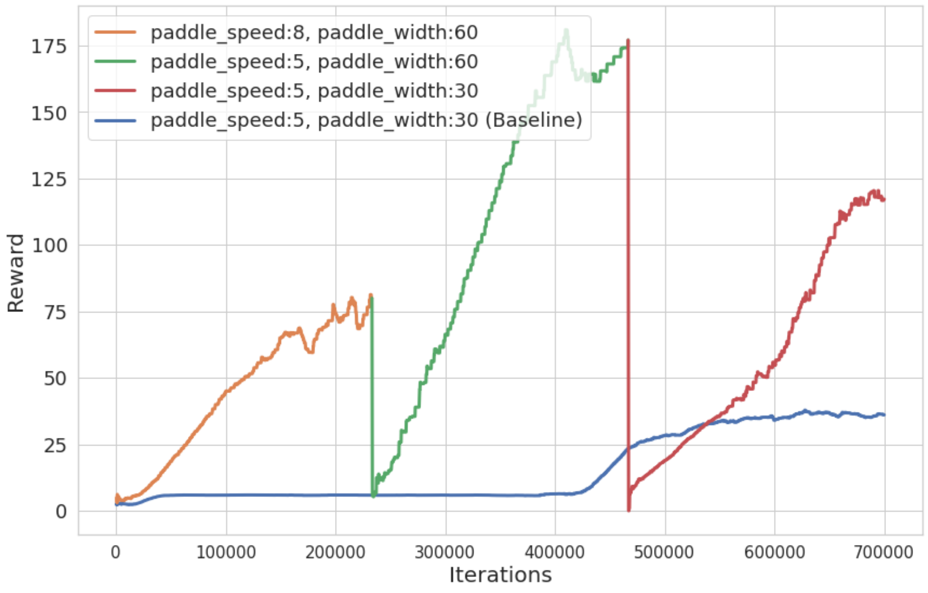
(b) The worst training result on Acrobot.

Figure 6.5 : The best (a) and the worst (b) samples from Acrobot results.

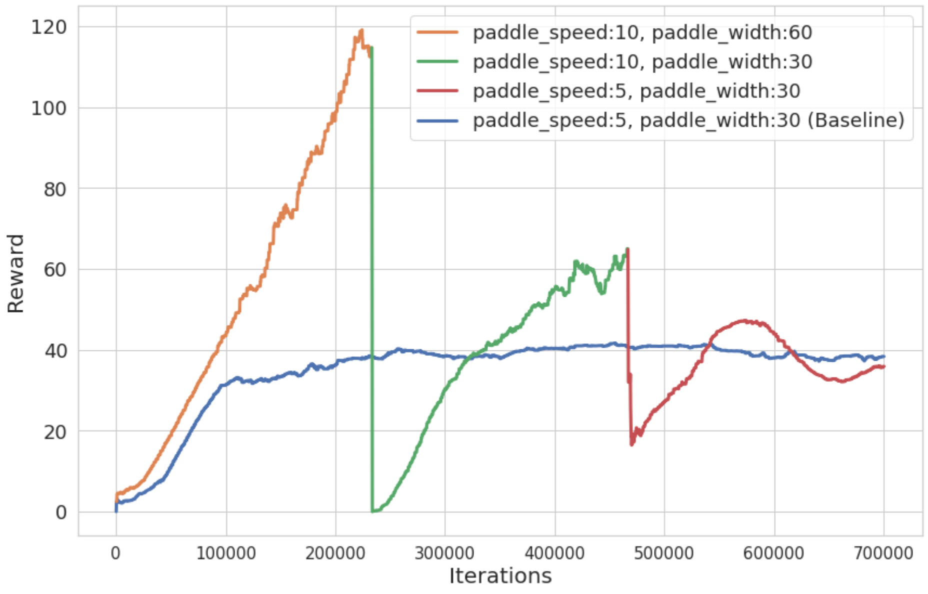
Table 6.5 : Average results for Acrobot environment

Property	Value
Target Reward	-50
Maximum Iterations	17203.2
After-Curriculum Reward	**-126.96**
After-Vanilla Reward	-414.2

6.6 Breakout-v0



(a) The best training result on Breakout.



(b) The worst training result on Breakout.

Figure 6.6 : The best (a) and the worst (b) samples from Breakout results.

Table 6.6 : Average results for Breakout environment

Property	Value
Target Reward	200
Maximum Iterations	645120.0
After-Curriculum Reward	**68.9**
After-Vanilla Reward	27.5

6.7 Overall Results

Table 6.7 : Overall results of the proposed algorithm

Environment	Vanilla Training Reward	Curriculum Training Reward	Target Reward	Improvement (I)
Pendulum-v0	-1092.029	**-325.236**	-200	+85.96%
LunarLander-v2	84.641	**93.519**	150	+13.58%
LunarLander-v2-C	119.805	**126.885**	150	+23.45%
BipedalWalker-v3	**-50.408**	-68.975	0	−36.82%
Acrobot-v1	-414.2	**-126.96**	-50	+78.87%
Breakout-v0	27.5	**68.9**	200	+24%

After the results were normalized according to the distances to the target reward, the relative improvement rate brought by the algorithm was calculated. The growth rate equation used for each experiment can be expressed as in Equation 6.1:

- δ_{VT}: Reward difference between target and vanilla rewards
- δ_{CT}: Reward difference between target and curriculum rewards

$$I = \frac{\delta_{VT} - \delta_{CT}}{\delta_{VT}} \tag{6.1}$$

6.7.1 Discussion

It is seen that the proposed algorithm promises up to 85% improvement. Since the algorithm is based on step-by-step statistical methods, it is interpretable and allows to make reasonable observations about the experiment. Although the algorithm contributes to the learning process in most cases, in some scenarios it can give results that are no different or worse than vanilla training. The reason for this may be the complexity of the environment dynamics, the relevance of the selected curriculum environment parameters to the learning process or the working principle of the algorithm. The algorithm trains all parameter combinations one by one. At the same time, it evaluates the models trained in all combinations separately on all combinations. This workflow causes great complexity. The algorithm can be improved with a heuristic to remove this drawback.

6.7.1.1 Future Work

- In order to get clearer and more precise results from the algorithm, instead of making each experiment separately and aggregating the results, it can be ensured that more efficient statistical models can be obtained by processing for all parallel experiments at each step and aggregating the intermediate results step by step.
- Instead of experimenting with a grid-search type method on the given values as in the current state, more clear parameter values can be found with a randomized-search-based method by making probabilistic sampling in a given value range.

7. CONCLUSION

The curriculum outcomes of the algorithm have obvious advantages over standard training. Since it generates the statistics according to the environment with ten different seeds, its robustness is high and its reproducibility on other devices has been partially proven. During the experiments, it was found that a variable thought to be important was not important, and the effect that was thought to be created was the opposite. Thanks to the proposed curriculum learning algorithm, improvements can be seen with certain parameter options without the need for expertise in the field. This is a helpful factor in clarifying the less obvious features related to the structure and dynamics of the environment. The study can be made more stochastic with Gaussian type random sampling methods, at the same time, performing the aggregation process in the intermediate phases rather than at the end can increase the robustness and quality of the results.

REFERENCES

[1] **Hadri W.**, (2021), Machine Learning, Data Science and Artificial Intelligence, `https://link.medium.com/s5xaTySadib`, [Online; accessed July 26, 2021].

[2] **Hocking, R.R.** (1983). Developments in Linear Regression Methodology: 1959-1982, *Technometrics*, *25*(3), 219–230.

[3] **Pelillo, M.** (2014). Alhazen and the nearest neighbor rule, *Pattern Recognition Letters*, *38*, 34–37.

[4] **Hearst, M.A.**, **Dumais, S.T.**, **Osuna, E.**, **Platt, J. and Scholkopf, B.** (1998). Support vector machines, *IEEE Intelligent Systems and their applications*, *13*(4), 18–28.

[5] **Hosmer Jr, D.W.**, **Lemeshow, S. and Sturdivant, R.X.** (2013). *Applied logistic regression*, volume398, John Wiley & Sons.

[6] **Ke, G.**, **Meng, Q.**, **Finley, T.**, **Wang, T.**, **Chen, W.**, **Ma, W.**, **Ye, Q. and Liu, T.Y.** (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree, *Advances in Neural Information Processing Systems*, volume 30, Curran Associates, Inc.

[7] **Patel, S.**, **Sihmar, S. and Jatain, A.** (2015). A study of hierarchical clustering algorithms, *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp.537–541.

[8] **Lloyd, S.P.** (1982). Least squares quantization in PCM, *IEEE Trans. Inf. Theory*, *28*, 129–136.

[9] **Breunig, M.**, **Kriegel, H.P.**, **Ng, R. and Sander, J.** (2000). LOF: Identifying Density-Based Local Outliers., volume 29, pp.93–104.

[10] **Liu, F.T.**, **Ting, K. and Zhou, Z.H.** (2009). Isolation Forest, pp.413 – 422.

[11] **Regueiro C.V.**, (2021), Basic diagram of a RL agent, `https://www.researchgate.net/figure/Basic-diagram-of-a-RL-agent_fig2_248278861`, [Online; accessed July 26, 2021].

[12] **Russell, S.J. and Norvig, P.** (2009). *Artificial Intelligence: a modern approach*, Pearson, 3 edition.

[13] **Sutton, R.S. and Barto, A.G.** (1998). *Reinforcement Learning: An Introduction.*, MIT Press, Cambridge, MA.

[14] **Silver, D.** (2016). Deep Reinforcement Learning, International Conference on Machine Learning (ICML).

[15] **Bellman, R.** (1952). On the Theory of Dynamic Programming, *Proceedings of the National Academy of Sciences*, *38*(8), 716–719, `https://www.pnas.org/content/38/8/716`, `https://www.pnas.org/content/38/8/716.full.pdf`.

[16] **Bellman, R.E. and Dreyfus, S.E.** (1962). *Applied Dynamic Programming*, Princetown University Press.

[17] **Watkins, C.J.C.H. and Dayan, P.** (1992). Q-learning, *Machine Learning*, *8*(3), 279–292.

[18] **Rummery, G. and Niranjan, M.** (1994). On-Line Q-Learning Using Connectionist Systems, *Technical Report CUED/F-INFENG/TR 166*.

[19] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Graves, A.**, **Antonoglou, I.**, **Wierstra, D. and Riedmiller, M.** (2013). Playing Atari with Deep Reinforcement Learning.

[20] **Mnih, V.**, **Kavukcuoglu, K.**, **Silver, D.**, **Rusu, A.A.**, **Veness, J.**, **Bellemare, M.G.**, **Graves, A.**, **Riedmiller, M.**, **Fidjeland, A.K.**, **Ostrovski, G.**, **Petersen, S.**, **Beattie, C.**, **Sadik, A.**, **Antonoglou, I.**, **King, H.**, **Kumaran, D.**, **Wierstra, D.**, **Legg, S. and Hassabis, D.** (2015). Human-level control through deep reinforcement learning, *Nature*, *518*(7540), 529–533.

[21] **Bengio, Y.**, **Louradour, J.**, **Collobert, R. and Weston, J.** (2009). Curriculum Learning, ICML '09, Association for Computing Machinery, New York, NY, USA, p.41–48.

[22] **Skinner, B.** (1958). Reinforcement today, *American Psychologist*, *13*, 94–99.

[23] **Peterson, G.B.** (2004). A DAY OF GREAT ILLUMINATION: B. F. SKINNER'S DISCOVERY OF SHAPING, *Journal of the Experimental Analysis of Behavior*, *82*(3), 317–328.

[24] **Portelas, R.**, **Colas, C.**, **Weng, L.**, **Hofmann, K. and Oudeyer, P.Y.** (2020). Automatic Curriculum Learning For Deep RL: A Short Survey, *ArXiv*, *abs/2003.04664*.

[25] **Zhang, X.**, **Kumar, M.**, **Khayrallah, H.**, **Murray, K.**, **Gwinnup, J.**, **Martindale, M.J.**, **McNamee, P.**, **Duh, K. and Carpuat, M.** (2018). An Empirical Exploration of Curriculum Learning for Neural Machine Translation, *ArXiv*, *abs/1811.00739*.

[26] **Wang, W.**, **Caswell, I. and Chelba, C.** (2019). Dynamically Composing Domain-Data Selection with Clean-Data Selection by "Co-Curricular Learning" for Neural Machine Translation, *ACL*.

[27] **Siyang Li, Xiangxin Zhu, Q.H.H.X. and Kuo, C.C.J.** (2017). Multiple Instance Curriculum Learning for Weakly Supervised Object Detection, ***G.B. Tae-Kyun Kim Stefanos Zafeiriou and K. Mikolajczyk**, editors, Proceedings of the British Machine Vision Conference (BMVC)*, BMVA Press, pp.29.1–29.14, `https://dx.doi.org/10.5244/C.31.29`.

[28] **Wang, J.**, **Wang, X. and Liu, W.** (2018). Weakly- and Semi-supervised Faster R-CNN with Curriculum Learning, *2018 24th International Conference on Pattern Recognition (ICPR)*, pp.2416–2421.

[29] **Shi, M.**, **Caesar, H. and Ferrari, V.** (2017). Weakly Supervised Object Localization Using Things and Stuff Transfer, *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, IEEE Computer Society, pp.3401–3410, `https://doi.org/10.1109/ICCV.2017.366`.

[30] **Tang, Y.**, **Wang, X.**, **Harrison, A.P.**, **Lu, L.**, **Xiao, J. and Summers, R.** (2018). Attention-Guided Curriculum Learning for Weakly Supervised Classification and Localization of Thoracic Diseases on Chest Radiographs, *MLMI@MICCAI*.

[31] **Kumar, M.**, **Packer, B. and Koller, D.** (2010). Self-Paced Learning for Latent Variable Models, *NIPS*.

[32] **Zhang, D.**, **Meng, D.**, **Li, C.**, **Jiang, L.**, **Zhao, Q. and Han, J.** (2015). A Self-Paced Multiple-Instance Learning Framework for Co-Saliency Detection, *2015 IEEE International Conference on Computer Vision (ICCV)*, 594–602.

[33] **Jiang, L.**, **Meng, D.**, **Zhao, Q.**, **Shan, S. and Hauptmann, A.** (2015). Self-Paced Curriculum Learning, *AAAI*.

[34] **Morerio, P.**, **Cavazza, J.**, **Volpi, R.**, **Vidal, R. and Murino, V.** (2017). Curriculum Dropout, *2017 IEEE International Conference on Computer Vision (ICCV)*, 3564–3572.

[35] **Kim, T.H. and Choi, J.** (2018). ScreenerNet: Learning Self-Paced Curriculum for Deep Neural Networks, *arXiv: Computer Vision and Pattern Recognition*.

[36] **Shrivastava, A.**, **Gupta, A. and Girshick, R.B.** (2016). Training Region-Based Object Detectors with Online Hard Example Mining, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 761–769.

[37] **Braun, S.**, **Neil, D. and Liu, S.C.** (2017). A curriculum learning method for improved noise robustness in automatic speech recognition, *2017 25th European Signal Processing Conference (EUSIPCO)*, 548–552.

[38] **Schaul, T.**, **Quan, J.**, **Antonoglou, I. and Silver, D.** (2016). Prioritized Experience Replay, *CoRR*, *abs/1511.05952*.

[39] **Matiisen, T.**, **Oliver, A.**, **Cohen, T. and Schulman, J.** (2020). Teacher–Student Curriculum Learning, *IEEE Transactions on Neural Networks and Learning Systems*, *31*, 3732–3740.

[40] **Florensa, C.**, **Held, D.**, **Wulfmeier, M.**, **Zhang, M. and Abbeel, P.** (2017). Reverse Curriculum Generation for Reinforcement Learning, *ArXiv*, *abs/1707.05300*.

[41] **Bellemare, M.G.**, **Srinivasan, S.**, **Ostrovski, G.**, **Schaul, T.**, **Saxton, D. and Munos, R.** (2016). Unifying Count-Based Exploration and Intrinsic Motivation, *NIPS*.

[42] **Pathak, D.**, **Gandhi, D. and Gupta, A.** (2019). Self-Supervised Exploration via Disagreement, ***K. Chaudhuri and R. Salakhutdinov**, editors, Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, PMLR, pp.5062–5071, `http://proceedings.mlr.press/v97/pathak19a.html`.

[43] **Shyam, P.**, **Jaśkowski, W. and Gomez, F.** (2019). Model-Based Active Exploration, ***K. Chaudhuri and R. Salakhutdinov**, editors, Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, PMLR, pp.5779–5788, `http://proceedings.mlr.press/v97/shyam19a.html`.

[44] **Mysore, S.** (2019). Reward-guided Curriculum for Robust Reinforcement Learning.

[45] **Risi, S. and Togelius, J.** (2019). Procedural Content Generation: From Automatically Generating Game Levels to Increasing Generality in Machine Learning, *ArXiv*, *abs/1911.13071*.

[46] **OpenAI**, **Akkaya, I.**, **Andrychowicz, M.**, **Chociej, M.**, **Litwin, M.**, **McGrew, B.**, **Petron, A.**, **Paino, A.**, **Plappert, M.**, **Powell, G.**, **Ribas, R.**, **Schneider, J.**, **Tezak, N.**, **Tworek, J.**, **Welinder, P.**, **Weng, L.**, **Yuan, Q.**, **Zaremba, W. and Zhang, L.** (2019). Solving Rubik's Cube with a Robot Hand, *ArXiv*, *abs/1910.07113*.

[47] **Andrychowicz, M.**, **Crow, D.**, **Ray, A.**, **Schneider, J.**, **Fong, R.**, **Welinder, P.**, **McGrew, B.**, **Tobin, J.**, **Abbeel, P. and Zaremba, W.** (2017). Hindsight Experience Replay, *NIPS*.

[48] **Sutton, R.S.**, **McAllester, D.**, **Singh, S. and Mansour, Y.** (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation, ***S. Solla, T. Leen and K. Müller**, editors, Advances in Neural Information Processing Systems*, volume 12, MIT Press, `https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf`.

[49] **Schulman, J.**, **Levine, S.**, **Abbeel, P.**, **Jordan, M. and Moritz, P.** (2015). Trust Region Policy Optimization, ***F. Bach and D. Blei**, editors, Proceedings of the 32nd International Conference on Machine*

Learning, volume 37 of *Proceedings of Machine Learning Research*, PMLR, Lille, France, pp.1889–1897, `http://proceedings.mlr.press/v37/schulman15.html`.

[50] **Joyce, J.M.**, (2011). Kullback-Leibler Divergence, Springer Berlin Heidelberg, Berlin, Heidelberg, pp.720–722, `https://doi.org/10.1007/978-3-642-04898-2_327`.

[51] **Schulman, J.**, **Wolski, F.**, **Dhariwal, P.**, **Radford, A. and Klimov, O.** (2017). Proximal Policy Optimization Algorithms, *ArXiv*, *abs/1707.06347*.

[52] **Wikimedia Commons**, (2015), Multi-Layer Neural Network, `https://commons.wikimedia.org/wiki/File:Multi-Layer_Neural_Network-Vector.svg`, [Online; accessed July 26, 2021].

[53] **Wikimedia Commons**, (2007), Undirected graph., `https://commons.wikimedia.org/wiki/File:Undirected.svg`, [Online; accessed July 26, 2021].

[54] **Wikimedia Commons**, (2007), Directed graph., `https://commons.wikimedia.org/wiki/File:Directed.svg`, [Online; accessed July 26, 2021].

[55] **Dijkstra, E.W.** (1959). A note on two problems in connexion with graphs, *Numerische mathematik*, *1*(1), 269–271.

[56] **Team, P.D.**, (2011), PyGame, `http://pygame.org/`.

[57] **Kesting, A.**, **Treiber, M. and Helbing, D.** (2007). General lane-changing model MOBIL for car-following models, *Transportation Research Record*, *1999*(1), 86–94.

[58] **Treiber, M.**, **Hennecke, A. and Helbing, D.** (2000). Congested traffic states in empirical observations and microscopic simulations, *Physical review E*, *62*(2), 1805.

[59] **Gupta, A.**, **Johnson, J.**, **Fei-Fei, L.**, **Savarese, S. and Alahi, A.** (2018). Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks, *CoRR*, *abs/1803.10892*, `http://arxiv.org/abs/1803.10892`, `1803.10892`.

[60] **Haarnoja, T.**, **Zhou, A.**, **Abbeel, P. and Levine, S.** (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, ***J. Dy and A. Krause**, editors, Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, PMLR, Stockholmsmässan, Stockholm Sweden, pp.1861–1870, `http://proceedings.mlr.press/v80/haarnoja18b.html`.

[61] **Narvekar, S.**, **Peng, B.**, **Leonetti, M.**, **Sinapov, J.**, **Taylor, M.E. and Stone, P.** (2020). Curriculum learning for reinforcement learning domains: A framework and survey, *Journal of Machine Learning Research*, *21*(181), 1–50.

[62] **Brockman, G.**, **Cheung, V.**, **Pettersson, L.**, **Schneider, J.**, **Schulman, J.**, **Tang, J. and Zaremba, W.**, (2016), OpenAI Gym, `http://arxiv.org/abs/1606.01540`, cite arxiv:1606.01540.

[63] **Mayank M.**, (2018), Reinforcement learning — Agent's action and environemet's reply, `https://itnext.io/reinforcement-learning-with-q-tables-5f11168862c8`, [Online; accessed July 26, 2021].

[64] **Hoel, C.J.**, **Wolff, K. and Laine, L.** (2018). Automated speed and lane change decision making using deep reinforcement learning, *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, pp.2148–2155.

[65] **Salvucci, D.D. and Gray, R.** (2004). A Two-Point Visual Control Model of Steering, *Perception*, *33*(10), 1233–1248, pMID: 15693668.

[66] Vehicles — NVIDIA PhysX SDK 3.4.0 Documentation, `https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Vehicles.html`.

[67] **Pacejka, H.B.** Chapter 4 - Semi-Empirical Tire Models, **H.B. Pacejka**, editor, Tire and Vehicle Dynamics (Third Edition), Butterworth-Heinemann, pp.149–209, `https://www.sciencedirect.com/science/article/pii/B9780080970165000048`.

[68] **Kordani, A.A.**, **Rahmani, O.**, **Nasiri, A. and Boroomandrad, S.M.** Effect of Adverse Weather Conditions on Vehicle Braking Distance of Highways.

APPENDICES

APPENDIX A

A system with NVIDIA Tesla V100 GPU and 128GB RAM has been used in training.

Initialization Phase

The highway environment for training the RL agent is controlled by several parameters. The highway is initialized with n number of lanes. Next, m agents are placed in the environment, following certain rules. Each of the agents has a dimension of 4.5×2.5 meters.

The initial longitudinal (x_0) and lateral (y_0) positions of the vehicles were determined, provided that the maximum initial vehicle spread of the vehicles did not exceed the maximum distance d_{long} and not fall below the minimum inter-vehicle distance $d_{\triangle}$.

The agent in the middle was chosen as ego-vehicle when agents were sorted according to their longitudinal positions. The agents in front of the ego-vehicle have relatively lower initial speed v_0 within the range of $[v_{min}^{front}, v_{max}^{front}]$. The agents behind the ego-vehicle have relatively higher initial speed v_0 within the range of $[v_{min}^{rear}, v_{max}^{rear}]$. The same range layout $[v_{min}^{ego}, v_{max}^{ego}]$ also applies for the ego vehicle. Also, desired speeds are defined for each agent included ego-vehicle in the range of $[v_{min}^{d}, v_{max}^{d}]$ and v_{ego}^{d}. A distance limit d_{max} has been set to finish each episode. These parameters have been determined by taking reference values from [64]. Table A.1 shows the parameters.

Table A.1 : Simulation parameters

Minimum inter-vehicle distance, $d_{\triangle}$	25 m
Maximum initial vehicle spread , d_{long}	200 m
Desired speed for ego vehicle, v_{ego}^{d}	25 m/s
Episode length, d_{max}	5000 m
Desired speed range for other vehicles, $[v_{min}^{d}, v_{max}^{d}]$	$[18, 26]$ m/s
Rear vehicles initial speed range, $[v_{min}^{rear}, v_{max}^{rear}]$	$[15, 25]$ m/s
Front vehicles initial speed range, $[v_{min}^{front}, v_{max}^{front}]$	$[10, 12]$ m/s
Initial speed range for ego vehicle, $[v_{min}^{ego}, v_{max}^{ego}]$	$[10, 15]$ m/s
Number of vehicles, m	9
Number of lanes, n	3

Vehicle and Steering Control Model

To simulate the dynamics of vehicles, non-linear kinematic bicycle model is used. Steering angle δ_f and the acceleration value a have been set to be control inputs. To calculate δ_f and a, two-point visual control model of steering [65] and the IDM [58] is used, respectively. Steering angle δ_f with two key-points from the rear and front of the vehicle is estimated by a calculation method called two-point visual control model.

Observation states and action spaces

The ego vehicle has the capability to observe the entire environment. The table A.2 shows the observable states which were described such that, it can adapt to different number of vehicles which besiege the ego vehicle. [64].

Table A.2 : Observation states of the ego vehicle

s_1,	Normalized ego vehicle speed v_{ego}/v^d_{ego}
s_2,	ego vehicle $\begin{cases} 1, & \text{if there is a lane to the leftt} \\ 0, & \text{otherwise} \end{cases}$
s_3,	ego vehicle $\begin{cases} 1, & \text{if there is a lane to the right} \\ 0, & \text{otherwise} \end{cases}$
s_{3i+1},	Normalized relative position of vehicle i, $\Delta s_i/\Delta s_{max}$
s_{3i+2},	Normalized relative velocity of vehicle i, $\Delta v_i/v_{max}$
s_{3i+3},	$\begin{cases} -1, & \text{if vehicle i is two lanes to the right of ego vehicle} \\ -0.5, & \text{if vehicle i is one lanes to the right of ego vehicle} \\ 0, & \text{if vehicle i is in the same lane as the ego vehicle} \\ 0.5, & \text{if vehicle i is one lanes to the left of ego vehicle} \\ 1, & \text{if vehicle i is two lanes to the left of ego vehicle} \end{cases}$

where the maximum allowed speed for all vehicles is v_{max}, maximum relative position between ego vehicle and vehicle i is Δs_{max} and the maximum allowed speed for ego vehicle is v^d_{ego}. The are three action spaces for the vehicle. a_1 for no lane change, a_2 for left lane change and a_3 for right lane change .

Table A.3 : MOBIL Hyper-Parameters

Changing threshold, a_{th}	0.1 m/s2
Politeness factor for rear vehicles, q	0.5
Politeness factor for side vehicles, p	1
Maximum safe deceleration, b_{safe}	4 m^2

Table A.4 : IDM Hyper-Parameters

Minimum gap, d_0	2 m
Safe time headway, T	1.6 s
Desired deceleration, b	1.7 m/s^2
Maximum gap for empty lane, d_{max}	10000 m
Minimum deceleration, a_{min}	-20 m/s^2
Maximum acceleration, a_{max}	0.7 m/s^2
Acceleration exponent, δ	4

The environment-specific parameters of IDM and MOBIL algorithms used for lane change and speed control are as in Table A.3 and Table A.4.

Reward function

The objective of this work is to train an agent that can adapt in various environments and drive safely without violating the safety of the road. The parameters used in the reward function are shown below.

$$r(s,a,s') = \begin{cases} \text{Speed Reward: } (v_{cur} - 15)/(v_{des} - 15) \\ \text{Low Acc Reward: } -\text{Speed Reward} \\ \text{Lane Change Penalty: } -1 \\ \text{Out of Road Penalty: } -100 \\ \text{Hard Crash: } -100 \\ \text{Soft Crash: } -10 \\ \text{Goal: } +100 \end{cases} \tag{A.1}$$

There are two different crashes defined in the reward function. Hard crash is the direct collusion with the other vehicle whereas the soft crash is the dangerous approach to the other vehicle.

APPENDIX B

Vehicle Dynamics

SimStar makes use of NVIDIA PhysX [66] as the primary physics engine. It is a highly sophisticated physics engine, and the realism in vehicle dynamics validated through various real-life experiments. Additionally, the vehicle properties are adjusted to match a real autonomous vehicle. They are also validated through real-life road tests.

Weather Effects on Physics

Accurate simulation of weather effects depends on modelling of the interaction between the tire and the road. Pajecka Tire Model [67] is used as the tire model. The parameters regarding road friction and tire friction are calculated to match a real world study on the topic. The work by Kordani et. al [68] calculates the road friction coefficient at different weather conditions for different type of vehicles.

Table B.1 : Braking Distance of Vehicles on Adverse Conditions

	Dry	Rainy	Snowy
Coefficient of Friction	0.8	0.4	0.28
Sedan (m)	105	114	133
Bus (m)	115	116	169

The results for the difference in braking differences on adverse weather conditions is used in this work's simulations to generate realistic behavior. Braking distance of a vehicle going at $80kph$ can be seen at Table B.1 at each adverse condition. Since the vehicles used in the simulator are different than the vehicles in the reference paper, only modeling the adverse weather effect solutions are opted in proportion to the original study.

The breaking distance of carefully modeled *sedan* vehicle on dry weather is 80.5 meters. Then, the rest of the adverse weather road models are adapted to create the correct proportional effect on braking distances. The final results can be seen on the Table B.2.

Table B.2 : Braking Distance of Ego Vehicle on SimStar

	Dry	Rainy	Snowy
Sedan (m)	80.5	84.1	91.0

Observation and Control

All vehicles get several information about the road and the vehicle itself as well as information about other vehicles at every control step. SimStar provides several sensors on every vehicle in order to supply the information demands of the control algorithms. The road deviation sensor gives these information about the vehicle:

- The vehicle's angular deviation from the road's central axis in radians.
- The vehicle's distance deviation from the road's central axis in meters.

These two values are scalars and are included in the observation at every control step.

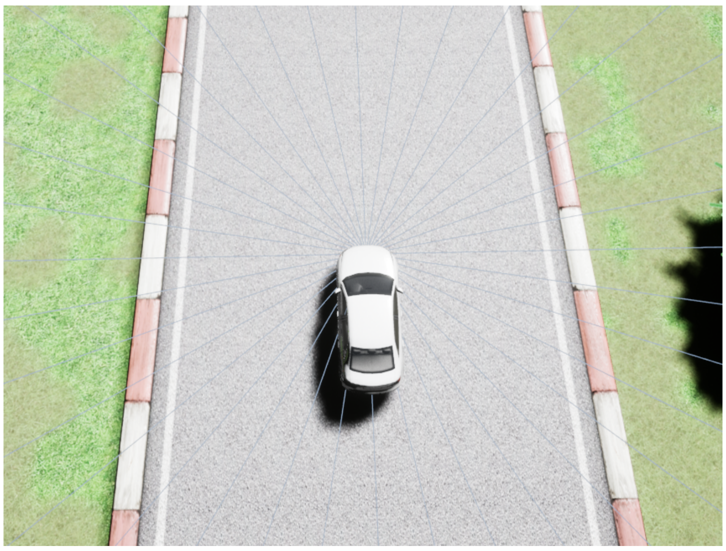


Figure B.1 : The sensor returns the distance of each ray to road boundaries.

The track sensor in Figure B.1, receives information about the vehicle's location on the road. This sensor is used to identify borders and edges of the road with respect to the vehicle's central body. The sensor gives a vector of 19 scalar values. These values are being created by scanning the front-half of the surrounding area with 10° splits. Thus, there are 19 distance sensor lines. These values are also included in the observation state in the environment at every action step.

State and Action Space Of The Agent

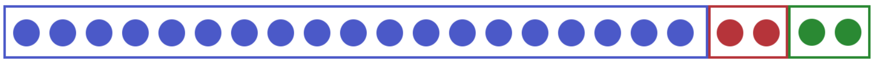


Figure B.2 : Visualization of the observation states.

The state vector consist of 23 different inputs. 19 of these inputs come from the laser sensors in Figure B.1. The other inputs are related to vehicle position on the track and the current velocity of the vehicle in both X and Y directions. The complete representation of the state vector is shown in Figure B.2.

There are 2 different actions that can be taken by the agent in the simulation. One of the actions is throttle/break and the other one is steering action. They are represented in Figure B.3.

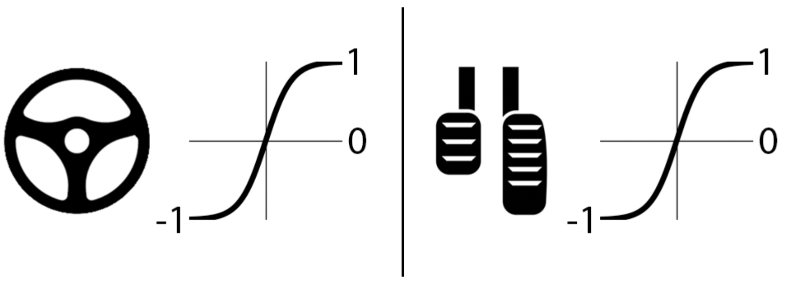


Figure B.3 : Visualization of the action space (steering and throttle).

Reward Structure

A custom reward calculation is implemented with a predetermined penalty. The reward function for the training procedures is given in Eq. B.1. The reward is calculated at every control step, when an action is done by the agent.

$$R = sp \times (cos(\beta) - |sin(\beta)| - tr_{pos}) \quad \textbf{(B.1)}$$

sp is the speed of the vehicle in km/h obtained from the resultant value of lateral and longitudinal axes speeds. β is the angle in radians and it shows the angular deviation of the vehicle heading direction and road central axis. tr_{pos} is the lateral deviation of the vehicle from the road center in meters.

Eq. B.1 states that if the speed of a vehicle is high, resultant reward gets higher; but the agent would have to minimize the result of $|sin(\beta)|$ and tr_{pos} to get a positive reward. In the reward function, the speed (sp) is multiplied with a difference of the road deviation angle's cosine and sine components to attain members of vehicle's speed in lateral and longitudinal motion. This means the agent will get negative reward with ratio of its speed in lateral axis. Thus, the agent would learn to minimize its speed in a lateral axis. Another variable in Eq. B.1 is tr_{pos} which is the distance of the agent from the central axis in meters. Minus sign of tr_{pos} multiplied with resultant speed of the vehicle makes the negative reward component of total reward calculation. The agent should learn to stay at the center of the road as much as possible because of this negative reward.

-20 penalty is given to the agent if it crashes, gets damaged, gets out of the road or if its speed is too low. In all these situations, the agent gets a constant penalty, which makes the penalization process equal for all terminations.

APPENDIX C.1

The training, testing and analyzing processes of the proposed algorithm were carried out on a workstation with the following hardware:

- NVIDIA RTX3090
- 128GB of RAM
- Intel® Xeon® Gold 5220R Processor
- 14TB HDD

APPENDIX C.2

The software, libraries and related versions used throughout the study in the thesis are given in table C.2.1:

Table C.2.1 : Software information

Software	Version
Python	3.7.10
GCC	7.3.0
numpy	1.19.2
gym	0.18.0
stable_baselines3	1.0
scipy	1.6.1
pandas	1.2.3
networkx	2.5

CURRICULUM VITAE

Name Surname	**: Anıl Öztürk**
Place and Date of Birth	**: 30.01.1996 Bursa**
E-Mail	**: anilozturk96@gmail.com**

EDUCATION	**:**
• **B.Sc.**	**:** 2018, Yildiz Technical University, Faculty of Mechanical Engineering, Department of Mechatronics Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2021-Present, Machine Learning Engineer at Eatron Technologies

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Öztürk, A., Günel, M. B., Dal, M., Yavaş, U., Üre, N. K.** 2020. Development of A Stochastic Traffic Environment with Generative Time-Series Models for Improving Generalization Capabilities of Autonomous Driving Agents, *IEEE Symposium on Intelligent Vehicle,* 1343-1348.

- **Öztürk, A., Günel, M.B., Dagdanov, R., Vural, M.E., Yurdakul, F., Dal, M., Ure, N.K.** 2021. Investigating Value of Curriculum Reinforcement Learning in Autonomous Driving Under Diverse Road and Weather Conditions, *IEEE Symposium on Intelligent Vehicle*