POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Predicting Deep Reinforcement Learning agents learning time for video game playing: a data-driven approach

Supervisors
prof. Paolo Giaccone
prof. Andrea Bianco

Candidate
Alessandro Lovaldi

Academic Year 2020-2021

Acknowledgements

Vorrei ringraziare con tutto il mio cuore la mia famiglia che mi ha sempre supportato durante la mia carriera scolastica. In particolare voglio ringraziare la mia mamma e il mio papà, Elena e Roberto, senza i quali non avrei avuto al fortuna di poter spendere la mia gioventù solamente preoccupandomi dello studio.
Voglio anche ringraziare i miei amici che evito di elencare per evitare di dimenticare qualcuno. Gli ringrazio per avermi sempre spinto a dare del mio meglio.
Infine voglio ringraziare coloro che mi hanno aiutato a completare questo lavoro; i mei supervisori Paolo Giaccone, Andrea Bianco e German Sviridov. Loro mi hanno seguito passo dopo passo nei mesi necessari per la relalizzazione di questa tesi, aiutandomi con le loro idee e consigli a potarla a termine nei miglior modo possibile.

Abstract

In the last few decades, machine learning has made massive progress. This progress has made machine learning useful in a wide range of studies. One of the flourishing research filed is the one that applies machine learning to gaming. Countless reinforcement learning models have been created for a wide range of game genres.
Many studies and applications make use of AI agents trained with one of those models. As an example, the work the inspired this thesis proposes to use an AI agent to assess the Quality of Experience in cloud gaming services. Training an agent from zero has some inconveniences. One of the problems is the high variance of the duration of the training phase. This variance is due to many factors. The main ones are the reinforcement learning model selected, the hardware used to run the training, and the complexity of the game.
The goal of this thesis is to identify which characteristics make a game complex to be learned by an AI agent and how this complexity affects the time of learning. More precisely, we have studied if it is possible to predict how long it takes for a selected model to learn a game. This prediction is based solely on the game features. For our research, the games selected are Atari games and the model selected is Double DQN. DDQN is a deep reinforcement learning algorithm able to play at a superhuman level Atari games.
We have achieved our goal by modifying an existing DDQN model to gathered data from tens of Atari games during the training phase. The data collected describe two main aspects of the game: the shape of the reward signals and the visual component. The shape of the rewards is a key aspect of reinforcement learning. Reward frequency and magnitude can heavily influence the model performance. The visual component is considered because the DDQN uses as input the frame's pixels. We then used unsupervised machine learning techniques, like linear regression analysis, to research the correlation between the game characteristics and the training duration.

Contents

1 Introduction 4
- 1.1 Motivation . . . 4
  - 1.1.1 Application of artificial bots in gaming industry . . . 4
  - 1.1.2 Thesis's problem tackled and idea . . . 5
- 1.2 Outline of Thesis . . . 6

2 Preliminaries/Background 7
- 2.1 Machine Learning . . . 7
  - 2.1.1 Reinforcement learning . . . 7
  - 2.1.2 Q-Learning . . . 9
  - 2.1.3 Limits of classical reinforcement learning . . . 10
- 2.2 Deep Reinforcement Learning . . . 10
  - 2.2.1 Deep Learning . . . 10
  - 2.2.2 DQN . . . 11
  - 2.2.3 DDQN . . . 12
  - 2.2.4 Other models . . . 13
- 2.3 Deep Reinforcement Learning for video game playing . . . 14
  - 2.3.1 Mapping DQN on games . . . 14
  - 2.3.2 Atari dataset . . . 15
  - 2.3.3 Other examples of games . . . 15
- 2.4 Correlation coefficients . . . 17
  - 2.4.1 Pearson correlation coefficient . . . 17
  - 2.4.2 Spearman's rank correlation coefficient . . . 17
- 2.5 Regression analysis . . . 18
  - 2.5.1 Linear Regression . . . 18

3 Estimating game difficulty from synthetic data 23
- 3.1 Methodology . . . 23
  - 3.1.1 General description . . . 23
- 3.2 Which data can be gathered? . . . 25
  - 3.2.1 Actions . . . 25
  - 3.2.2 Reward . . . 25

3.2.3 MSE and what it means . . . 26
3.2.4 SSIM and what it means . . . 26
3.3 Data Elaboration . . . 27
3.3.1 VAR SD and CV and what they mean . . . 27
3.3.2 Entropy what it means . . . 28
3.4 Data gathering . . . 28
3.4.1 Games considered . . . 28
3.4.2 Data gathering procedure . . . 29
3.4.3 Running on HPC . . . 30
3.5 Procedure polishment . . . 30
3.5.1 Refine the data . . . 32
3.6 Data analysis . . . 32

4 Results 35
4.1 Result analysis . . . 36
4.1.1 Correlation Coefficients results . . . 36
4.1.2 Regression Results . . . 38
4.2 Result discussion . . . 41
4.3 Model . . . 42

5 Discussion 43
5.1 Future work . . . 43
5.1.1 Game clustering based on obtained data . . . 43
5.1.2 Analyse more complex games . . . 43
5.1.3 More advanced data analysis . . . 44
5.2 Conclusion . . . 44
5.2.1 Summary . . . 44

Bibliography 45

Chapter 1

Introduction

1.1 Motivation

1.1.1 Application of artificial bots in gaming industry

For the last decade, thanks to the advent of the internet, the videogame industry has boomed. The popularity of the videogames is increased thanks to the global success of different multiplayer games and mobile games.
In the same years, machine learning has been and still is a hot topic in computer science. Machine learning algorithms have been applied in many fields of everyday life. Gaming is one of the fields in which machine learning has been found useful.
One of the uses of machine learning in videogames is the creation of non-player characters, also known as NPC. In single-player games, the character controlled by the players has to face CPU-controlled enemies. In multiplayer games, instead is a common feature to be able to add NPCs in a game as a way for a player to train his skills. In both cases, the best CPU-controlled bots are the one the behaves like humans. NPCs not skilled enough makes the game not challenging and soon boring for the players. An extremely skilled AI agent, on the other hand, makes the game too hard and too frustrating to be enjoyed.
By studying the gameplay of human players, machine learning algorithms can be used to create bots that have human-like behaviors. A better game AI could highly enhance the gameplay quality.
Procedural content generation (PCG) is the branch of computer science that studies the application of AI technologies for the creation of content. Multiple studies apply PCG to the computer generation of game stages. One of these studies uses information gathered from players' questionnaires in order to generate Super Mario levels [1]. PCG techniques that predict map entertainment values have been used to automatically generate maps for the real-time strategy game Starcraft [2]. Another study uses PCG to create tracks for racing games the better suit the player's driving style [3]. These are just a few examples in which machine learning algorithms have

been applied in the production of videogame content.
During the last years a new way to play videogames is born; cloud gaming. Instead of buying the game and make it run on the local driver, the user can pay a subscription to play games on the server of the provider. The player's device sends to the server the inputs and the server sends back the output video. This kind of service is very dependent on the network performances. Various games need different latencies to reach a satisfying quality of experience. A study has been made to compute the maximal acceptable latency by each game by using AI agents [4]. By measuring the decrease in performance of an AI agent in correlation with the increase of the latency it is possible to classify games based on their latency requirement. The service provider could then prioritized the internet packets of the more demanding games to ensure the best possible QoE for all its customers.
These are just few examples in which machine learning technologies can be applied to gaming.

1.1.2 Thesis's problem tackled and idea

Even though there are many ways in which it is possible to apply machine learning to videogames its use also has some inconveniences.
One of the biggest problems with machine learning is that, if not already present online, you have to train your agent from scratch. The training of an agent is often long and very computationally expensive. Computers with dedicated GPUs are needed to complete the training in a reasonable time. If at least one resource between time or computational power is limited it would be useful to have a priori estimation of the time needed for the training in order to evaluate if the use of an AI agent is worth the resources investment.
Right now there are no tools able to predict the time required to train an agent to play the desired game. The duration of the training doesn't only depend on the used hardware. The chosen model and game have also an impact on the time needed. Most of the models use the video output of the game as the input for the agent training. it is reasonable to assume that some of the characteristics of the game probably influence the ability of the model to learn the game quickly. The intuition that gave birth to this work is that games in which many frames are very similar to each other are the ones more difficult to learn for machine learning models. The reasoning behind this idea is that with a lesser variation in the frames the model has less information to use for improving the agent's policy.
If this assumption is true, it may be possible to create a framework able to predict the duration of the training based on the game features. Gathering game statistics, such as information regarding the game video output, is often an easier and faster process than the agent training procedure. In this work, we study the behavior of the training of multiple Atari games using a Double DQN model in order to find, if present, the correlation between the time of learning of a game and some of its

characteristics. In case a correlation is found we will use that relationship to create a theoretical prediction model for the training duration of a game.

1.2 Outline of Thesis

In chapter 2, we discuss the background knowledge needed to better understand the work done. More precisely, in the first part of chapter 2 we talk about the theory behind machine learning and some of the machine learning algorithms used to train agents able to play videogames. In the later part of the chapter, we instead illustrate statistical notions regarding correlation coefficients and linear regression analysis that will be helpful to understand the meaningfulness of our results.
In chapter 3, we describe which game statistics we have decided to collect and why. We also explain the procedure followed to gather such data and how we have elaborated them to obtain the needed information.
In chapter 4, we present and explain the results obtained.
In the last chapter, chapter 5, we offer some ideas on possible future works on the topic dealt addressed in this thesis. Finally, we conclude with a summary that recaps the most important points of the paper.

Chapter 2

Preliminaries/Background

2.1 Machine Learning

Machine learning is the field of computer science that studies computer algorithms that recognize and exploit information present in data without the direct intervention of human help. Machine learning algorithms can be divided into general families depending on which learning techniques the model exploits. The great majority of the algorithm can be divided between supervised learning, unsupervised learning, and reinforcement learning.
Supervised learning algorithms work utilizing labeled data. Supervised learning is more commonly utilized in classification problems. The model takes as input a set of label data, called the training set. The goal of the model is to exploit the knowledge present in the training set to form a policy able to classify unlabeled data. The model improves its policy by minimizing the loss function. The loss function is a function that quantifies how much the current classification differs from the ground truth. For supervised learning to work well often is needed a huge quantity of label data and that is not always the case.
Unsupervised learning tries to discover hidden patterns in the input data. More specifically unsupervised learning is applied, for example, in clustering problems where the goal is to discover if the data can be grouped in clusters or in regression problems where the model tries to use the present information to decipher the correlation between the data and so be able to predict future values based on some of their features.
Reinforcement learning takes a completely different approach. It tries to learn a task through experience by interacting with its environment.

2.1.1 Reinforcement learning

The reinforcement learning approach is very similar in the way humans learn things. An agent interacts with the environment receiving different stimuli depending on

whether the action computed is positive or negative in order to reach the goal. In the same way, our brain releases substances like dopamine when we engage in productive behaviors.
To understand how reinforcement learning works, it is important to understand the key concepts that define it. A reinforcement learning task is composed of an environment. The representation of the environment at time i is called state s_i. The set of all possible environment states is called S. The agent interacts with the environment through a defined set of actions $a_0, a_1, ..., a_n \in A$. When the agent, at step i, performs the action a_i this action has an effect on the state s_i that transits to state $s_i + 1$. [5]
To infer how good an action a_i is in state s_i the agent receives a signal, called a reward r. The reward can be positive in case the action makes the agent reach one of the task's goals, negative in case of failure, and 0 if the action has not an immediate effect on the state of the task. The triggering factors and the magnitude of the rewards are highly task-dependent.
The function that selects which action to select for the current state is called policy π. More precisely the policy π is a mapping from each state, $s_i \in S$, and action, $a_i \in A$, to the probability $\pi(s_i, a_i)$ of taking action a_i when in state s_i. The model goal is to create the optimal policy π^*. The better policy is the one that maximizes the cumulative reward R obtained during the task. [5]

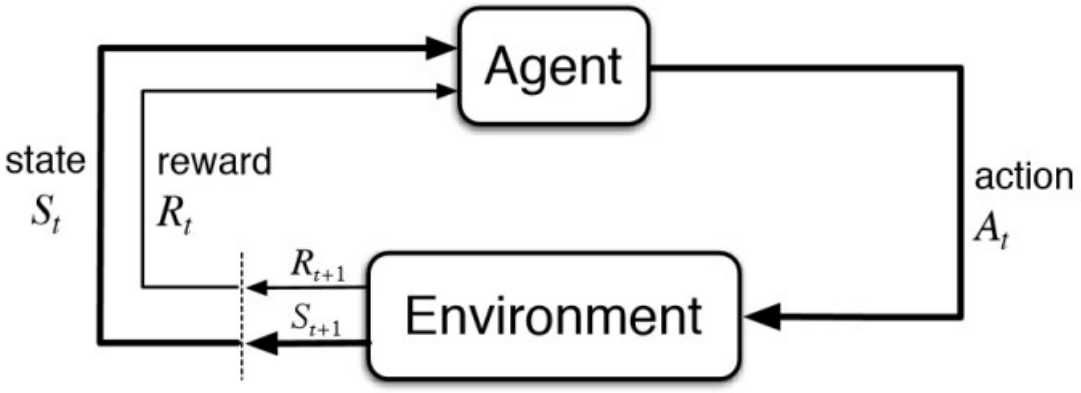

Figure 2.1. A scheme of the the basic reinforcement learning mechanism. At each iteration the agent selects an actions to interact with the environment. The selected action makes the environment transit to state $s + 1$ and produce the reward $r + 1$.

The simplest policy is the greedy one; in which the agent selects the action which gives the highest immediate reward. This strategy is few times the optimal one. A better way to produce a strategy is to compute the value of each state. If the state representation can fully describe the environment it is possible to forecast all the

future actions chosen by policy π starting from state s_i. Therefore it is possible to compute the expected cumulative reward R for state s_i. Given a police π and a state s_i we can compute the state value function $V^\pi(s_i)$ as the expected sum of the rewards starting from state s_i and following policy π. [5]

$$V^\pi(s_i) = E_\pi\{R|s = s_i\}$$

A possible strategy is therefore select the action that leads to the state with the highest value. Similarly to the state value function, that returns the value of the state, it is possible to compute the action-value function $Q^\pi(s_i, a_i)$ that evaluates the expected reward by selecting the action a_i in the state s_i and then following the policy π in all the following states. [5]
The action-state function is defined as:

$$Q^\pi(s_i, a_i) = E_\pi\{R|s = s_i, a_t = a_i\}$$

2.1.2 Q-Learning

Different reinforcement learning algorithms use different ways to update their policy. In this work, we focus on Q-learning. In Q-learning, the goal is to compute the best possible approximation of the Q value for all actions and in all states. The action-state value of action a_i in state s_i is updated with the formula:

$$Q(s_i, a_i) = Q(s_i, a_i) + \alpha[r_{i+1} + \gamma max_a Q(s_i + 1, a) - Q(s_i, a_i)]$$

Where $Q(s_i, a_i)$ is the current values associated to the pair s_i, a_i. α is the learning rate, i.e. a value between 0 and 1 that regulates how big is each adjustment of the Q-value. r_{i+1} is the reward received when a_i is selected and γ is the discount factor. The discount factor is a value between 0 and 1. Its purpose is to regulate how much the expected future reward affects the updating of the current Q-value. Furthermore, a discount factor less the 1 guarantees the convergence of the action-state value. $max_a Q(s_i + 1, a)$ indicates that the current Q-value is updated based on the maximum expected Q-value of the following state [5].
In Q-learning, the policy is not considered in the update of the state-action value. The policy utilized the majority of the time is the greedy one. The action selected in state s_i is the one with the greater action-state value. A greedy policy works only if all the approximated Q-values tend to their real ones. To guarantee that the agent explores all the actions and updates the Q-values of each of them the algorithm selects ϵ percent of the time a random action. ϵ represents the exploration factor. The policy that takes the greedy action $1 - \epsilon$ percentage of the time is called ϵ-greedy policy. With infinite exploration time, It is proven that Q-learning converges to the optimal action-selection policy [6].

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Figure 2.2. The Q-learning algorithm [5]

2.1.3 Limits of classical reinforcement learning

Reinforcement learning works particularly well when the task tackled has a limited set of states and the state representational can fully describe the environment. The problem is the vast majority of real problems do not have these characteristics. For example, create an agent able to play Atari2600's games, as we do this work, is not feasible with a classical reinforcement learning algorithm. Since the state representation utilized are the pixels of each frame the states are too many and too complex.
To solve more complex tasks, reinforcement learning is combined whit deep learning giving birth to deep reinforcement learning.

2.2 Deep Reinforcement Learning

2.2.1 Deep Learning

Deep learning can solve a vast gamma of problems, including ones where a high dimensional representation is required. It is common practice, for example, to use deep learning in image-related tasks.
Deep learning models utilize networks composed of a series of layers. Each of these layers applies a non-linear function to the input data. This type of network is called Convolutional Neural Network (CNN).
The most common type of layer is the convolutional layer. A convolutional layer is composed of parametrized filters, also called kernels, which have a small receptive field. These filters compute convolutional operations on the input data and then pass the results to the next filter. Usually in a convolutional neural network are present multiple convolutional layers. Each layer has the duty of learning a specific

feature of the input image. Starting from the figures' edges and arriving at complex subjects. [7] [8]
Poling layers are used for non-linear down-sampling operations. Down-sampling is utilized in machine learning to reduce the dimensionality of the internal representation of the data. The most common polling layer implements max pooling. Max pooling is an operation done on data which have a matrix representation. The input matrix is dived into regular sub-regions. For each of these sub-regions, only the higher value is kept. In this way, a smaller matrix is created. The new matrix, even if smaller, retains most of the information of the original one. [9] Reduce the dimensions of data it is useful to model since it removes the noisy unhelpful portion of the data and helps to speed up to subsequent computations. This type of layer is very commonly used in image related models.
The last portion of CCN is often formed of a series of fully connected layers. The fully connected layers are the ones that extract the information computed by the previous layers and produce the final results. [9] For example in a reinforcement learning setting the last layer, called the output layer, produces the probability to be picked for each action. Instead in the supervised learning case, the output layer produces the probability of correctness of each label.

2.2.2 DQN

Combing the Deep learning approach with Q-learning theory has given birth to the deep Q-network (DQN) [10]. DQN uses a deep network to estimate the Q-Value of reinforcing learning problems. It has been tested on 49 Atari games using as input raw pixels, proving its ability to tackle a gamma of different tasks.
DQN is formed of 3 convolutional layers followed by 2 fully connected layers that produce the estimated Q-value for each valid action.
Previous attempts of using a non-linear approximator to approximate the action-value function suffered from high instability. This is the case of a neural network approximator. The instability is due to many causes. Some of them are the correlations present in the sequence of observations and the fact that variation of the Q-values makes the policy change causing a variation of the data distribution.
To improve stability in the DQN model experience replay has been introduced. The experience replay mechanism works by storing the agent experience. Each experience value e_i a step i is formed of the state representation s_i, the action selected a_i, the reward obtained r_i, and the resulting state s_{i+1}. After a fixed number of steps, a random experience value is extracted by a uniform distribution and it is used to update the loss function. The presence of experience reply helps with the instability because it randomizes the sequence of the samples analyzed by the system, lowering sampling correlation which reduces the variance of the updates.
As briefly mentioned in section 2.1 in machine learning, the loss function measures the distance between the present performance of the model and the optimal performance.

The loss function has as parameters the parameters of the neuronal network layers. During training, the model updates its parameters in an attempt to minimize the loss function. The loss function's minimum corresponds to the best parameters' value since they are the values that produce the least distance to the optimal solution. The search of the minimum is done by stochastic gradient descent, i.e. the parameters are modified in the direction in which the loss function is the steepest by an amount controlled by the learning rate. The loss function utilized in DQN is the following:

$$L_i = E(s, a, r, s')[(r + \gamma max'_a Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2]$$

where γ is the discount factor, θ_i are the parameters of the Q-network at time i, and θ_i^- are the target parameters at iteration i.
To reduce instability, every C steps, the network is duplicate. The copied network is the target network, i.e. the network that computes the target values for the following C steps. The target values are obtained by approximating the optimal target values $r + \gamma max'_a Q^*(s', a')$ with $r + \gamma max'_a Q^*(s', a'; \theta_i^-)$ where θ_i^- are the parameters of the current target network.This modification improves stability because in this way a modification of the Q-value by the main network does not influence the target value, reducing oscillation.

2.2.3 DDQN

One of the problems of the DQN model is its tendency to overestimate actions. This problem is due to the fact that the DQN model uses the same value to both select and evaluate actions. The Double DQN (DDQN) [11] uncouples the selection and evaluation procedure. To do so, DDQN modifies the target approximation function utilized by DQN. The target used by DQN can be written as:

$$Y = r + \gamma Q(s', max'_a Q^*(s', a'; \theta_i); \theta_i)$$

The modified target utilized in DDQN is instead:

$$Y = r + \gamma Q(s', max'_a Q^*(s', a'; \theta_i); \theta'_i)$$

To least modify the original DQN code the implementation of DDQN uses the parameters θ of the online network to select the action following the ϵ-greedy policy. The estimation is done by utilizing the parameters θ^- of the target network instead. Tests have shown that DDQN performs better than the standard DQN model on the 49 Atari games considered in the original DQN paper.

2.2.4 Other models

Besides the DQN and the DDQN model exist many other models that can play arcade games at the same, if not better, level as humans [12]. All these models use as input raw pixels.
Some of these models are, like the DDQN model, modifications of DQN. This is the case with Deep Recurrent Q-Learning that adds to the DQN network a recurrent layer [13]. The General Reinforcmentent Learning Architecture, called Gorilla architecture, is a distributed version of DQN. Gorilla uses parallel agents which experience is collected in a distributed replay memory. Gorilla outperforms DQN in 41 of the 49 Atari games used as a test [14]. Another modification of DQN is Dueling DQN. This model further develops the DDQN idea of splitting the action selection and action evaluation process. After the convolutional layers, the network split into two branches one selects the actions, and the other evaluates them. [15]
Another family of models implements the Actor-Critic method (AC). The AC method combines the stochastic gradient descent approach with temporal difference learning. In TD learning the value function, the function that computes the expected cumulative reward of a state, is computed as

$$V(s) = V(s) + \alpha(R(s) + \gamma V(s') - V(s))$$

where $V(s)$ is the current value of state s, $V(s')$ is the value of the next state s', $R(s)$ is the current reward, and α is the learning rate.
Updating the state value in this way, the future value of the future states backpropagates to the previous ones. Some of the models that implement an AC method are the A3C, A2C, ACER, and UNREAL. [12]
A3C stands for Asynchronous Advantage Actor-Critic. It uses parallel agents that asynchronously update the actor-critic network [16]. The synchronous version of the A3C model is called A2C [12]. The actor-critic experience replay model (ACER) is an actor-critic model that, as the name suggests, implements the concept of experience replay [17]. UNREAL stands for UNsupervised REinforcement and Auxiliary Learning. The UNREAL algorithm is a modification of A3C which makes use of experience replay. The replay memory technique is used to learn auxiliary tasks. Auxiliary learning is a branch of machine learning. In auxiliary learning, the model learns to solve one or more secondary tasks concurrently with the main task. The information learned for the secondary tasks is used by the model to improve the main task performance. UNREAL has a similar performance to A3C on arcade games, but it has been proven better in other domains. [18]
The ones describe are just a few of the models existent. There are many other models able to have humanlike or even better performance in playing arcade games. Even more if broad the spectrum to other game genres like first-person shooters, fighting, and racing games.

2.3 Deep Reinforcement Learning for video game playing

2.3.1 Mapping DQN on games

Key aspects that make a model performing are the task's environment representation and an effective way for the agent to interact whit it.
In the case of DQN [10], the task is to play any Atari games with similar performance that an expert human can achieve. Each state representation is composed of the pixels of the frame and the current score. In the case in which the game contemplates multiple lives also this number is part of the state.
Each Atari frame is a 210 x 160 pixels image. Each pixel can be one of the possible 128 colors. Due to its extremely limited computational power, the Atari2600 can only handle few objects on the screen in the same frame. To overcome this restriction, when the game required many simultaneous entities the game developers used to make half of the objects appear on screen in the odd frames and the other half during the even frames. Atari games run at 60fps. The refresh rate is high enough to let the player sees both sets of objects on the screen simultaneously. This stratagem could heavily impact the performance of the model if not handled. The model does not perceive the frames as a sequence like the human eyes, but it just analyzes one frame at a time. This means that the agent could only see half of the object at every given instant if the frame were taken as they are. In the implementation of the DQN model to resolve this issue, each frame is combined with the previous one. The encoding takes the highest value for each pixel color of the two frames. In this way on each frame both sets of objects are present. A further preprocessing step is applied to each frame. To reduce the dimensionality each image gets rescaled to 84 x 84 pixels format. Furthermore, a function extracts the Y channel, the luminance, of each image. This procedure makes them black and white, reducing the dimension needed to represent each frame.
Since the agent's goal is to maximize the score, the simplest choice is to map the reward with the game's point. Each time an agent's action scores points in the game, the model generates a reward signal of equal magnitude.
The agent interacts with the environment emulating all the possible combinations of buttons allowed in the game. When the game file is loaded in ALE (Arcade Learning Environment), the emulator reads from it all the action a player can execute. The actions are then mapped to integer values. An action can not only correspond to the pressing of a button but also to a combination of them. An Atari game can have from 4 to 18 actions.
To improve the model's performance a technique called Frame Skipping is used [19]. Frame skipping consists of making the agent select an action every k frame instead of at all steps. In the frames in which the agent doesn't select a new action the last action chosen is repeated. Frame skipping makes the model run k times faster

since selecting an action is way more computationally expensive than making the environment advance of one step. A selection of a large value for k lets the model play more games in a fixed interval, but the agent can select an action less frequently making it less performing. The preselected value of k is 4.

2.3.2 Atari dataset

The games utilized in the DQN study are 49 games from the Atari2600 dataset. The Atari2600, released in 1977, is one of the first home consoles ever created. Due to its old age, the computational power of the hardware is quite limited. The games run on an 8-bit CPU alongside a RAM of just 128 bytes. Even though the computational power is quite limited the Atari2600 games are varied and represent a wild range of different tasks. Starting from the simple Pong where the goal is just to rebound a dot to the other half to the screen. Passing through a more complex game like Ms. Pacman in which the model as to learn how to navigate the maze of each stage avoiding colliding with the chasing ghosts. Arriving at the very difficult task to learn as Montezuma's Revenge in which the player controls a character the has to move in different rooms filled with enemies and trap and find keys to open doors to collect treasures. All these different game's type is tackle by the same algorithm and solved more or less well by only analyzing the state composed of the frame and the score and without having any other previous knowledge on any of them.

2.3.3 Other examples of games

The reinforcement learning models have been applied, as well as arcade games, to many other game genres [12].
Racing games are one of the challenges in which researchers apply machine learning models. In a racing game, an agent has to learn how to navigate the track as fast as possible managing the accelerator and the brake pedal in continuous time. In more realistic games the agent has to learn how to change gear and avoid the other cars. A popular platform used for this kind of study is TORCS. TORCS is a fully customizable 3D racing environment that, as DQN, uses pixels as input. AI models developed on racing games can be useful in the real world as self-driving technologies.
Another popular game genre in which deep learning is applied to videogames is the first-person shooter. In FPS games the player is put in the shoes of a soldier who has to shoot at some kind of enemy. In this type of game, the agent has to learn to recognize and quickly aim at the opponents. Furthermore, it has to learn how to navigate the 3D world. A common game used as a benchmark is Doom (id Software, 1993–2017). Between the models applied to this genre, there are also various DQN versions.
Open-world games offer a completely different challenge. These videogames don't

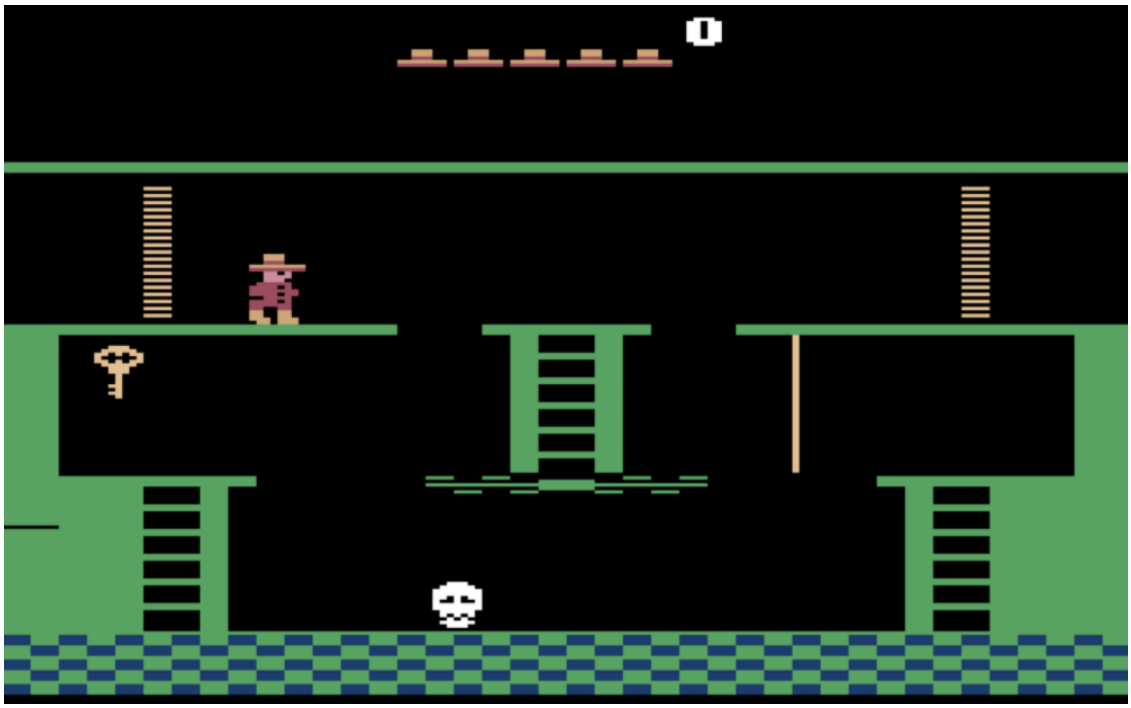

Figure 2.3. A screenshot of Montezuma's Revenge for Atari2600

have a clear goal. The player is let free to roam a vast 3D world and do whatever he likes. All the reinforcement learning algorithms are based on the concept of reward. Open-world games present a big challenge for reinforcement learning since the lack of an objective makes reward shaping a difficult task. Reward shaping consists of adding artificial rewards to encourage positive behaviors that are not rewarded by the game itself.
Lastly, real-time strategies games (RTS) have also been subject to AI studies. In this kind of game, the player has to control many units at a time. Often the goal is to manage the map resources to build buildings and train soldiers to destroy the enemy base. Each of the units created has to be managed by the player, that has to quickly strategy how to overwhelm the opponent. The environment often used is the one offered by StarCraft (Blizzard Entertainment, 1998–2017). Blizzard Entertainment has developed an API to enable software interaction with its game. This API is the result of a collaboration between the game developers and the DeepMind project of Google. [20]

2.4 Correlation coefficients

In statistics, the necessity to compute the correlation between two variables is a common problem. A possible way to resolve it is to compute a correlation coefficient. A correlation coefficient is a number that represents how strong is the statistical relationship of two features. There are various techniques to compute such correlation. In our work, we have utilized the Pearson correlation coefficient and Spearman's rank correlation coefficient.

2.4.1 Pearson correlation coefficient

The Pearson correlation coefficient (PCC), also called Pearson's r or ρ, is a measure of the linear correlation present between two variables. Given two random variables X and Y, the PCC is computed by dividing the covariance of the two variables by the product of their standard deviations.

$$\rho_{XY} = \frac{cov(X,Y)}{\sigma_x \sigma_y}$$

Where $cov(X,Y)$ is the covariance of X and Y. σ_x and σ_y are instead the respective standard deviation of X and Y. For a series of sample Pearson's r can be computed as:

$$r_x y = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where n is the number of samples and $\bar{x}$ is the series average. The PCC value is bounded between -1 and 1. A higher magnitude of the coefficient corresponds to a higher correlation between the two variables. The sign indicates if the correlation is a direct correlation or an inverse correlation. A ρ of 0 means that the two variables are not correlated linearly.

2.4.2 Spearman's rank correlation coefficient

The Spearman's rank correlation coefficient, also called Spearman's ρ or r_s, measures the rank correlation between two variables. It is computed exactly as the Pearson correlation coefficient with the difference that the coefficient is computed using the ranks of the values and not the values themselves. The lowest value is mapped to rank 1, the second-lowest to 2, and so forth. After every value of X has been transformed to its rank value rg_{X_i} and every value of Y to rg_{Y_i} we compute ρ as:

$$\rho = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X} \sigma_{rg_Y}}$$

The formula is the same as Pearson's coefficient but applied to the value of the ranks. The benefit of Spearman's rank correlation coefficient is that it doesn't assume the data belong to a normal distribution. Furthermore, Spearman's coefficient better evaluates non-linear correlation with respect to Pearson's coefficient as long as the correlation is monotonic.

2.5 Regression analysis

Correlation coefficients are useful to understand if a correlation is present between two variables. The use of regression analysis makes it possible to discover an approximation of the model that predicts the value of the independent variable Y. Furthermore, regression analysis allows us to study the combined effect of multiple variables on the behavior of the dependent variable Y.

2.5.1 Linear Regression

The simplest form of regression is linear regression. In 2 dimensional space, the goal of linear regression is to find the best fit line that better represents the relationship between a dependent variable y and the independent variable x. The concept of linear regression can be easily applied also in spaces with more than just 2 dimensions. For example, the correlation between y and a set of two variables x_1 and x_2 is described by a 3-dimensional plane. More formally, the dependent variable y is the linear combination of $k$independent variables or regressors $x_1, x_2, ...x_n$.

$$y' = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_n x_n + \epsilon_i$$

Where $\beta_0, \beta_1, ...\beta_n$ are the unknown coefficient that we are looking for and ϵ_i is the error in the values measurement. i ranges from 1 to n, where n denotes number of data points. Going forward we will talk about the 2-dimensional case for simplicity, but all the concepts are also valid in higher dimensions spaces. We make use of the linear regression analysis to precisely study the cases with more than 2 dimensions since in such instances the correlation coefficients explained before are not applicable. In the 2D settings, we have multiple measurements belonging to only one feature x. Each value of x corresponds to one value of the dependent variable y. We define the error term e_i of the point (x_i, y_i) as the difference between the measured value y_i and the estimated value y'_i of the best fit line. The best fit line is the line the minimizes the sum of the squared errors. The errors are squared because if not the sum of the negative errors and the positive ones will cancel out to 0. There are infinite lines that make the sum of the errors to be 0. Utilizing the square of the errors we transform all the errors values into positive ones resolving the problem of them adding up to 0. There is only one line that minimizes the sum of squared errors.

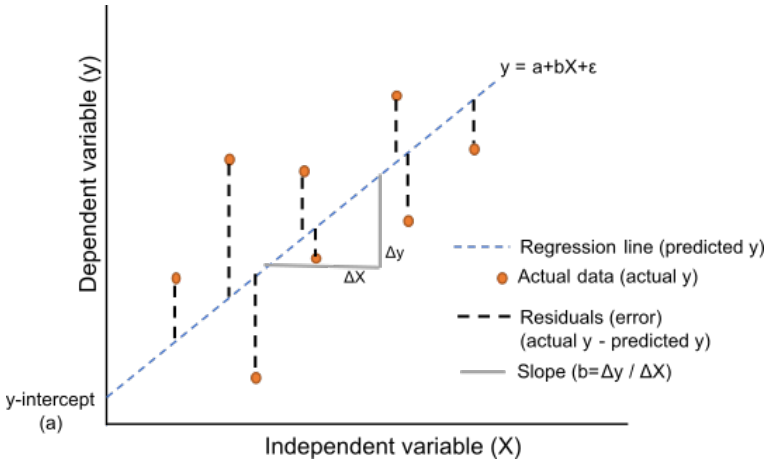

Figure 2.4. Example of regression line

We can use the sum of the squared errors to infer how good is the best fit line to explain the correlation between the independent variable and the dependent one. We consider the sum of the squared differences between the values predicted by the regression and the mean values of y, $\bar{y}$, as the amount of variance explained by the regression. This value is called SSR, Sum of Squares due to Regression.

$$SSR = \sum_{i=1}^{n} (y'_i - \bar{y})^2$$

y'_i indicates the predicted value of dependent variable Y by the best fit line for the value x_i. The sum of the squared error is called SSE and it indicates the amount of variance that our model is not able to explain.

$$SSE = \sum_{i=1}^{n} (y_i - y'_i)^2$$

The sum of SSR and SSE gives SST, Sum of Squares Total. SST indicates the total variation of the points from the mean. SST can also be computed as

$$SST = SSR + SSE = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

We can compute the proportion of variance explained by our model by computing R^2. R^2 is defined as the SSR divided by the SST.

$$R^2 = 1 - \frac{SSE}{SST} = \frac{SSR}{SST}$$

R^2 varies from 1 to 0. Smaller is the SSE, hence the points are closer to the line, higher is R^2. A regression line that passes through all the data points has an R^2 of

1. On the contrary, if many points a far from the regression line R^2 will be closer to 0.
The R^2 value can sometimes be misleading. This is especially the case when our model is described by more than one feature. With a rising number of features the R^2 value always increases. The improvement of the coefficient happens even if the added feature is useless in order to describe the correlation. The reason for this phenomenon is that every time a new independent variable is added the degrees of freedom(df) of the model decrease and, on the other hand, the number of constraints increases. We can compute the degrees of freedom of the model as

$$df = n - k - 1$$

where n is the number of samples and k is the number of independent variables. In the extreme case in which df equals 0 R^2 is always 1, no matter the distribution of the points. This concept is easy to understand in the 2-D case. With only one x we have 0 df when we have only 2 measurements. No matter where the 2 points are, there will be always a line the passes through both of them. The selection of β_0 and β_1 is therefore fixed.
A better representation of the explanatory power of our model is given by the $adjustedR^2$. The $adjustedR^2$ takes into consideration the model's df. It is computed through the formula

$$adj.R^2 = 1 - (\frac{SSE}{SST})\frac{n-1}{n-k-1} = 1 - (1-R^2)\frac{n-1}{n-k-1}$$

Adding a useless feature to the model will decrease the $adjustedR^2$, but a useful one increases it. The $adjustedR^2$ is not bounded between 0 and 1.
Besides the $adjustedR^2$, we have other statistical tools to understand how good is the regression line is in explaining the correlation under examination. One of these tools is called F-test.
The F-test is useful to compute with which level of significance we can reject the null hypothesis H_0. The null hypothesis states that all the coefficients of the regression are equal to 0, i.e. there is no relationship between the dependent variable and the independent ones. The F-test takes this name because it assumes that the test statistics have an F-distribution. The F-value is computed as:

$$F_{(k,df)} = \frac{\frac{SSR}{k}}{\frac{SSE}{df}}$$

Where k is the number of independent variables of the model. Computing the area between the absolute value of the computed F-value and $+\infty$ under the F-distribution we obtain the probability that the model performance is due to random chance. This probability is called the p-value. In statistics, it is common to consider the 95% significance level. If our p-value is less than 0.05 we can reject the null

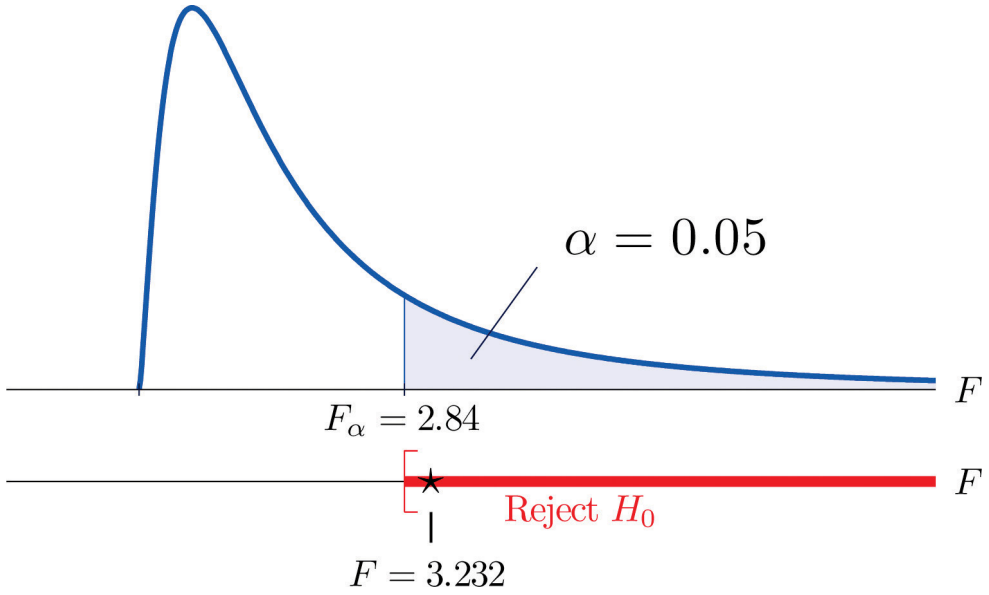

Figure 2.5. Example F-test in which the F-value is 3.232. Since it is higher than 2.84 we can reject the null hypothesis H_0 with more than 95% confidence

hypothesis H_0 with a confidence of 95% or more.

The use of the linear regression analysis is useful in our study because the relationship that we want to find could be formed from the contribution of multiple independent variables. When multiple features are used in the regression it is convenient to understand which of them is actually helpful to the model. To gain this knowledge the t-statistic is used. The t-statistic of the feature x_i is computed as

$$t_i = \frac{\beta_i}{SE(\beta_i)}$$

Where β_i is the coefficient of variable x_i in the regression and $SE(\beta_i)$ indicates the standard error of β_i. Assuming a t-distribution of the data, if the null hypothesis were true the t-statistics would be 0 since β_i would be equal to 0. As with the F-value, we compute the area under the t-distribution in the interval $(|t_i|, +\infty)$ to quantify the probability of getting an equal or higher value of β_i due to pure chance. The lower is this probability higher is the statistical significance that the independent variable has. Furthermore, we can compute the 95% confidence interval for each independent variable. If 0 does not belong to the interval we are 95% confident that that feature correlates with the dependent variable. In other words, we are 95% sure that the value of β_i is not 0, therefore, having an impact on the regression model.

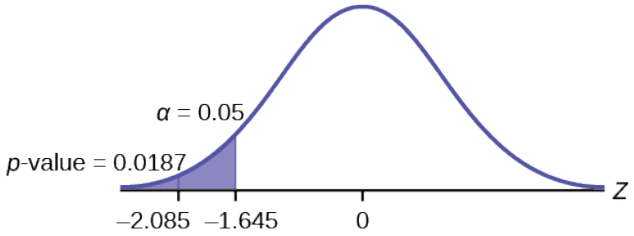

Figure 2.6. Example T-test in which the T-value is -2.085. Since it is lower than -1.645 the probability of getting such value by chance it is less than 5%. The feature is inversely correlated to the y since the t-value is negative.

Chapter 3

Estimating game difficulty from synthetic data

In this chapter it is described in detail how we studied the correlation between the training time and the intrinsic characteristics of the Atari2600 games. The goal is to be able to predict the time of training of a random game with the DDQN model.

3.1 Methodology

3.1.1 General description

The procedure followed consists of training, using the DDQN model, multiple Atari 2600 games. Each training period is composed of 500 epochs. An epoch can be considered as a training session in which the model tries to improve its policy. As already explained in the reinforcement learning section the policy is the function utilized by the system to select the best action for the current state of the game. Each epoch is formed of 25000 steps, where each step is the elaboration of one frame.
Since it is of our interest to study how a game is learned and not to achieve the best performing agent, we have to consider not only the more trained and therefore better performing model as often occurs in projects that use AI. We also need to study all the intermediate less skilled models. To do so we sample with a delta of 4 epochs the agent model during the training period. Each of these models is utilized to gather data on how the features of each game impact the training process. The data gathering process, from here onward referred to as the evaluation phase, is done by making each model play 50 games during which a series of key statistics are collected. These data are then analyzed through statistical and unsupervised machine learning techniques to find, if present, a correlation between the time of learning of a game and its characteristics.

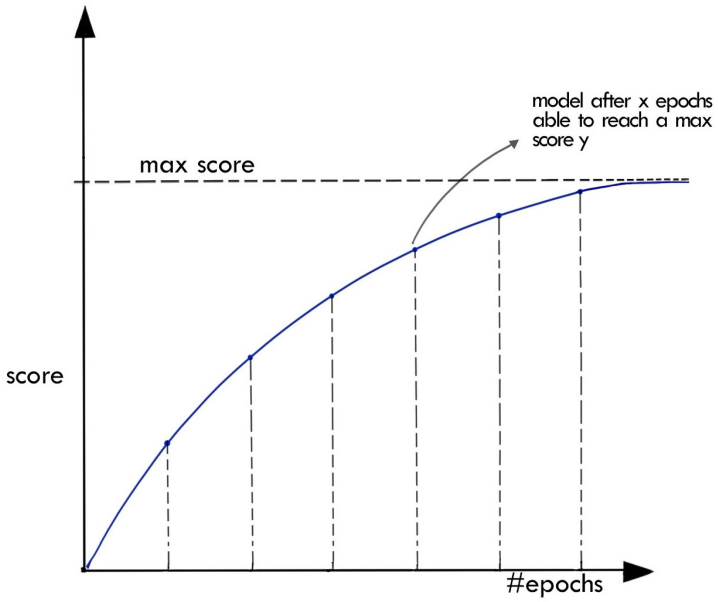

Figure 3.1. Example of a learning curve of a game reaching peak performance. During the training process the model is sampled with constant delta

Time of learning

It is fundamental to define a measurement that can represent the time of learning of a game since it is the cornerstone measure of this work. Our goal is in fact to predict the time of learning of each game.
The model improves its performance with each epoch of training until it reaches its peak performance. When we refer to the concept of peak performance we do not refer to the best performance possible on the game, but just the best performance reachable by the model on that specific task.
We consider a game to be "learned" by the agent when it reaches 90% of the score of the peak performance. We consider as the time of learning (ToL) the number of epochs that are needed to learn the game. Taking this metric as an indicator of how quickly a game is learned has some positive features. First of all, it is very simple to compute. Furthermore, the choice of considering the number of epochs instead of the effective time needed makes the measurement machine-independent and game-independent. The time needed to complete an epoch depends on the hardware used to run the training and the game played, but its computational

amount for a fixed game is constant and set to be equal to the one needed to elaborate 25000 frames. This makes the conversion between the number of epochs to a time unit of measurement trivial. It is just needed to run few epochs of training of the target game on the target machine and keep track of how long does it takes in average to complete one epoch and then multiple that measurement to the number of epoch indicated by the ToL to have an estimate on how long is going to be the training. This is the exact goal of this work, predict the time of learning a game without discovering it a posteriori after days of training.

3.2 Which data can be gathered?

Given that the DQN model learns is policy-based only on the video frames and the score it is important to analyze which data can be gathered in these domains that can impact the time of learning of each game.

3.2.1 Actions

The number of actions is a very important aspect that characterizes the complexity of a game. In the DQN based model, the agent needs to explore all the possible actions at every possible state to correctly assign at each of them a Q-value. Intuitively, it is easy to understand that, the more actions are present, the larger is the space of possibilities that in each state have to be evaluated to found the optimal one. So it is logical to think that with a bigger number of actions higher is the complexity of the game. The number of actions used by a game is easily obtained by the Tensorpack DDQN implementation at the moment the game environment is loaded.
A possibility to be considered is that just a subset of the total number of actions is utilized during the playing time with the rest rarely used by the model. It is then useful to keep track of the percentage in which each action is selected to explore these possibilities.

3.2.2 Reward

A hypothesis worth studying is the one that links the ToL with qualities of the reward signal. The concept of reward is a key aspect of reinforcement learning. The DDQN agent receives a reward signal each time it scores some points. Therefore the magnitude and the frequency of the rewards are solely dependent on the game. It is shown in many studies that reward shaping can increase the learning performance of an agent. For example, in the Visual Doom AI Competition 2016 the winning agent was trained using an A3C using reward shaping to overcome the problem of sparse rewards present in Doom. [21] Since the frequency and the magnitude of the rewards are independent characteristics of each game the hypothesis is that games

with more frequent rewards or/and with bigger rewards are more easily learnable. To study this hypothesis we have to keep track of all the rewards received by the model during the game both in terms of magnitude and also in terms of relative frames distance to have their temporal behavior.

3.2.3 MSE and what it means

The DQN model base is learning using the video information instead of reading the state of the game straight from the RAM. The information contained in each frame can be summarized as the difference between itself and a previous frame. If the majority of the frames are very similar to each other the amount of "information" that the model can utilize to improve the policy is very low and this can be reflected in the learnability of the game. To represent this amount of information we utilize the Mean Square Error MSE. The MSE is computed through the formula:

$$\frac{1}{n}\sum_{i=1}^{n}(X^2 - Y^2)$$

The MSE is therefore the average of the squared differences of X and Y, where usually Y is the predicted value and X is the result of an experiment. In our case, X and Y represent the luminosity value of a pixel of two adjacent greyscaled frames. Since all the differences are squared the MSE can only be a positive value. If the MSE is 0 that means that the two images exactly equal, on the contrary, larger is the MSE larger is the difference between the two frames. The magnitude of the MSE can be then seen as the information present in the transition between frame Y to frame X.

3.2.4 SSIM and what it means

The MSE is a useful means able to describe through a number the similarity of two frames, but it doesn't tell all the story. If we take for example two black and white figures, A and B. Figure A depicts a black square in a white background, instead, figure B depicts the same square of figure A but white in a black background. Even though the figure represents the same subject, and for a human their similarity is obvious, if we compute the MSE between the two figures the result would be a very high value. The cause is that the luminosity of every pixel is the largest possible. The two images have a structural similarity that is not reflected in the difference between each pixel value.
To study the information present in the difference between pixels' structure we make use of the SSIM, structural similarity index. The SSIM is a full reference metric in which values are included between 0 and 1. A SSIM of 0 means that the two images are completely different and instead if the SSIM is 1 they are two structural equal images. The SSIM index is calculated on various windows of an image. The

measure between two windows x and y of common size N $\times$ N is computed with the formula:

$$\frac{(2\mu_x\mu_y + c_1)(2\sigma_x y + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Where μ_x and μ_y are respectively the average of x and y, σ_x^2 and σ_y^2 the variance of x and y and σ_{xy} is the covariance of x and y. The two variable c_1 and c_2 are used to stabilize the division with weak denominator and their are computed as $c_1 = (k_1 L)^2, c_2 = (k_2 L)^2$, where L is the dynamic range of the pixel values and $k_1 = 0.01$ $k_2 = 0.03$ by default.
The SSIM is usually used to compute the similarity between a distorted image and its uncompressed original. In our case, as with the MSE, we utilize the SSIM to compute the structural similarity between the present frame and a frame in the past as a means to identify the amount of variation and therefore information present in the visual component of the game.

3.3 Data Elaboration

To further investigate the possible underlying correlation between how easy the model to learn a game and the data that we gather it is possible to extract from the later some useful characteristics such as the coefficient of variance and the entropy.

3.3.1 VAR SD and CV and what they mean

One of the data's characteristics that is useful to consider is their variance (VAR), i.e. how much the data are sparse with respect to their average value. In statistics the variance of a data distribution X is computed as follow

$$Var(X) = E[(X - \mu)^2]$$

where $E(x)$ is the expected value of x and μ is the mean of the distribution. The Variance is often represented by σ^2, this is due to the fact σ is the symbol chosen to represent the standard deviation (SD). The standard deviation is, therefore, the square root of the variance and is the most common way to compute how much the distribution's data are close to the distribution's mean. As more data are near the mean of the distribution more the SD value is low. The SD has the same unit of measure of the data of the distribution. The Coefficient of Variation(CV) is defined as the division of the SD by the mean μ

$$CV = \frac{\sigma}{|\mu|}$$

This measure is dimensionless.
We will compute the CV of the reward, MSE, and SSIM measurements to study if there is a correlation between more variation of the data and a faster ToL.

3.3.2 Entropy what it means

With the word entropy in this work, we do not refer to the scientific concept, but the measurement of information present in a distribution. The two concepts share the same name because both describe the amount of randomness and uncertainty that, as we have already explained in the case of two adjacent frames, can be correlated with the amount of information present in the distribution. The entropy applied in information theory has been theorized by Claude Shannon in 1948 and that's why it is also called Shannon entropy [22]. The entropy of a random variable X is computed as

$$H(X) = -\sum_{i=1}^{n} P(x_i) log(P(x_i))$$

where x_1, x_2, .. x_n are the possible outcome of X and $P(x_i)$ is the probability of i^{th} outcome. For a better interpretation and ease the comparison of different entropy value it is used to normalize the entropy through the formula

$$\eta = \frac{H}{H_{max}} = -\sum_{i=1}^{n} \frac{P(x_i) log(P(x_i))}{log(n)}$$

The normalized entropy is also called efficiency and is represented by the letter η. As whit the CV we are going to study the correlation between the entropy present in the reward, MSE, and SSIM data to further explore the hypothesis that more information in this data corresponds to a faster ToL.

3.4 Data gathering

In this section, we now describe how we were able to gather all the data described in the sections before.

3.4.1 Games considered

The games considered in our work are a subset of the Atari dataset. We have chosen these games for two reasons.
First, the Atari games represent a standard benchmark for multipurpose deep reinforcement learning models since they contain a vast range of different tasks. They have been used also as the benchmark for the DDQN model that we have selected. This allows us to study which are the characteristics of a game that influence the ToL on a fixed model.
Second, the learning of Atari games is less computational intensive relative to the other type of games. This allows us to train agents at more games in a shorter period on our limited hardware resources.

3.4.2 Data gathering procedure

The procedure starts with the training of an agent using the DDQN model to play Atari 2600 games. The training lasts for 500 epochs. Every 4 epochs the agent model is saved. Every sampled model can achieve a score in correlation with the number of epochs its training lasted. Each of these models goes through an evaluation procedure that consists of playing 50 games. The code has been modified such that during each of these games the information regarding action, reward, MSE e SSIM are saved on a tab-separated values (TSV) file. A TSV file is a simple text file where each data column is separated by a tab character. To be more precise the reward information is saved on file with name *reward_log_n*, where n is a number between 1 and 50 that correspond to the number of games played. In the Tensorpack DDQN implementation one episode, i.e. one game, is executed in the function play_one_episode in the file common.py. This function has the task to apply the current policy to select the action for the current state (the current frame and score). When the action is chosen it is used as a parameter of the environment function step that returns the information to the next state and the reward returned by the selected action. This reward is intercepted by a custom function called step_eval present in the file atari.py. If the reward is different from 0 the information regarding the frame in which the reward is received, that action that has triggered that reward, and the magnitude of the reward are saved on the TSV file. Furthermore, at the start of the evaluation task, an array of length 18 has been initialized with 0s. Each cell of the array corresponds to a counter of possible action. Every time an action is taken the corresponding counter is increased by one. For example, if the chosen action is mapped to the value 4 then the 5th cell(since the first cell is cell 0) of the array is increased by one.
At the end of the evaluation process, the action array is appended to a file called *name_action.txt*, where the name is the name of the game evaluated. This file it is utilized to study which action are used and which are not by the model playing each game.
Also, the frame information is computed in the custom function eval_step in atari.py. Since the delta between frame can be a factor in the result we compute the MSE and the SSIM between frames distant from each other 30 frames, 60 frames, and 120 frames that correspond to a distance of 0.5 s, 1 s, and 2 s in real-time since an Atari game runs at 60 frames per second. For simplicity, the frames considered are greyscaled. As for the reward information the data are saved in a TSV file called *image_log_n* where n is the number of games played. At every delta, the current frame is compared with the previous frame that has been stored in a specific variable. In each comparison, the MSE and SSIM are computed and saved on file then and the value of the variable containing the past frame is substituted with the current frame.
At the end of the evaluation phase of one game 125 folders (one for each model) are

produced. Each folder stores 50 files (one for each evaluation game) containing all the information needed regarding the reward, and another50 files containing the MSE and SSIM computation. A final file is created that stores the total count of use of each action.

3.4.3 Running on HPC

The training phase and the subsequent evaluation phase are, as often is with machine learning, very computationally expensive and take a very long time with a system without a dedicated GPU. To execute the computation we have utilized the hardware of the LEGION cluster offered by the initiative HPC@POLITO managed by the DAUIN (Dipartimento di Automatica e Informatica del Politecnico di Torino). [23] The cluster LEGION has installed 4 Nvidia Tesla V100 SXM2, one of which can complete on average the 500 epochs of training in 2 days and complete the evaluation phase in 1 day.
To make run the DDQN implementation on the cluster some preliminary steps are needed. There is a problem of compatibility between the CUDA version present on the cluster and the one utilized by the code that makes that the software doesn't recognize the GPUs present on the cluster. One of the possible solutions to the problem, and then utilized by us, is to create a Singularity container in which are installed the right version of the CUDA driver and the requirement needed i.e. TensorFlow-GPU, gym[atari], and the Tensorpack library. The training python file can then be run in the Singularity container using the Singularity module present on the cluster resolving the compatibility issue.
Each job on the cluster is queued to the scheduler of the cluster through the *sbatch* command. The sbacth command takes as an argument a sbatch file in which one has to specify the resources needed, the modules to load, and the commands to execute. The jobs queue is managed through the open-source Slurm workload manager. [24]

3.5 Procedure polishment

The procedure described above was run and refined multiple times until reaching its final form. We now discuss the most important improvements made.

Turning off dynamic learning rate/exploration

After the first runs utilizing the method described above, we have noticed a peculiar behavior in the scores of the game analyzed. At epoch 400 there was a visible jump in performance as can be seen in figure 3.3.

This behavior can be easily traceable to the setting of the learning rate and exploration parameters.
The learning rate parameter is a common feature of machine learning algorithms,

Figure 3.2. Ms. Pacman score with changing hyper-parameters at epoch 400

it controls how big is the step the model takes toward the better solution at every iteration. A too big learning rate worsens the performance because the model has a lack of precision that makes it is easier for the best solution to be overshot. A too small learning rate is also harmful because the model becomes very slow in reaching the optimal solution and it runs into the problem of getting stuck in a local minimum or maximum. The learning rate in the stock version of the DDQN model is set to change during the training to enhance the performance of the model. In the first 60 epochs is set to 0.001, between epoch 60 and epoch 400 it is set to 0.0005, and finally from epoch 400 onward to 0.0001.
The exploration rate instead is unique to some reinforcement learning algorithms. It balances the ratio between exploitation and exploration done by the agent. The DQN algorithm is based on making a greedy decision most of the time, i.e. selecting the action with the highest value, exploiting its knowledge. However, it is important that a percentage of time it selects a random action from the subset no containing the greedy one. This is necessary so that the agent explores all the other possible actions, refining their values and converging little by little to the real value of each action ensuring that the action with the max value is indeed the best one. The percentage of time in which a random action is selected is defined by the exploration rate.
Like the learning rate also the exploration rate is set to change during training in the stock version of the model. It is set to be 1 (that means that every action is chosen randomly) in the first 10 epochs, 0.1 between epoch 10 and epoch 400 and 0.01 from epoch 400 onward.
It is easy to understand that a lower learning rate and especially a lower exploration factor makes that after epoch 400 the performance improves drastically. Since our

goal is not performance but to study what makes a game difficult to learn is better to set this parameter constant during all the training. The values selected are 0.005 for the learning rate and 0.1 for the exploration.

3.5.1 Refine the data

We have defined the ToL as the number of epochs needed for the model to reach 90% of the peak performance, this definition is not straightforward to apply to the real data. This is because in practice the score-epoch graph is not as smooth as one can imagine due to the random component of the exploration and the seed of the game that introduces disturbance on the performance of the agents during the evaluation phase. The result of these disturbances can be seen in the score graph where the average of the score increases in correlation with the number of epochs of training but this growth is characterized by a heavy seesaw behavior. This seesaw behavior makes the concept 90% of top performance and of the top performance itself volatile since they are heavily affected by random peaks of performance. To counteract this phenomenon suitable precautions have to be taken.
First of all, is necessary to tackle the intrinsic random nature of the training phase since 10% of the time the agent selects a random action to explore the space of possibilities. This generates somewhat different learning curves even though they all converge to the same policy and so to the same performance given enough epochs of training. To minimize this effect we have chose to run the all procedure 5 times on the same game and consider as a learning curve the average of the 5 runs. To further get rid of the random factor that affects the data we also apply a rolling average on the learning curve, this will help us smooth the peaks allowing more precise measurement of the ToL.
To contrasts the randomness present in the Atari games, a fixed seed has been selected for all the games played in the evaluation procedure.

3.6 Data analysis

After collecting all the data regarding a game this data have to be analyzed. Each of the 5 training-evaluation procedures run for a game creates a folder each of them containing 125 additional folders, one for each model saved during the 500 epochs training (only one model every four is saved). Inside each of these 125 folders are present 100 files, 50 files containing the reward information of the 50 evaluation games played and 50 containing the information collected regarding the frames. The data analysis of this huge amount of files is done thanks to the python library pandas.
Pandas allow us to elaborate each of these files in few lines of codes instead of reading each of the lines by line. The elaboration of the file storing the rewards

information and the frame information is for the most part identical. Each file contains the information regarding one epoch that is loaded in a pandas dataframe. These data are elaborated to extract the metrics discussed before that are in case of the reward files the total score, the mean reward, the reward frequency, the entropy of the rewards, and the reward's coefficient of variation. All this data, that represents in a few number an epoch, are taken and added as a row to another dataframe that can be viewed as a table of a database where each row contains the information of a specific epoch of a specific training-evaluation run. Pandas allow us to collapse the 5 lines corresponding to the same epoch to just one line by effectuating a group by operation on the epoch field of the table. The resulting line is the mean of each field of the five rows with the same epoch number. We have now a 125 rows dataframe storing the information of the average training behavior of a specific game. We compute the rolling average (with window 10 and minimum period 1) of the score series and we obtain the final learning curve of the game. This is the curve utilized to compute the game's time of training.

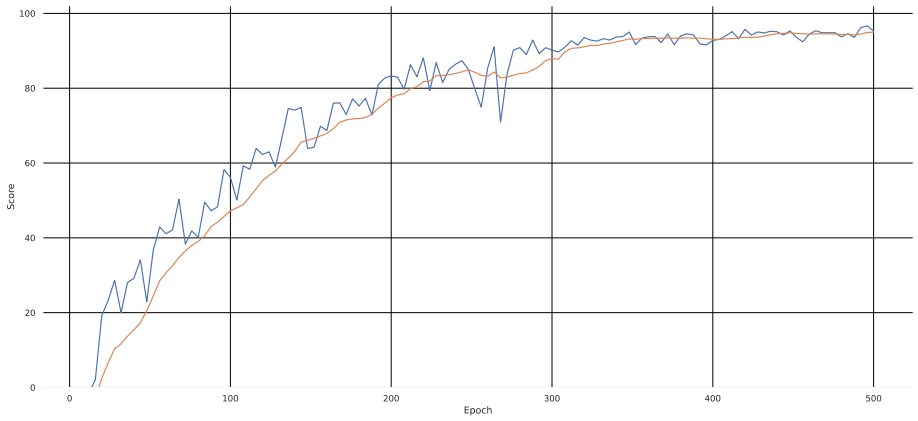

Figure 3.3. The learning curve of the game Boxing. In blue the curve before the rolling average and in orange after

To summarize all the game characteristics in few numbers we compress the 125-row table in just one line computed as the average value of each column. Each row presents the information regarding the mean reward value, the reward frequency, the reward CV, the reward efficiency for the reward aspect and the MSE mean, MSE CV, MSE efficiency, the SSIM mean, the SSIM CV, and the SSIM entropy for the frame analysis aspect. For each measurement regarding the MSE and SSIM 3 values are stored, one for each delta between frames considered.
These values with the addition of the number of actions, computed for each game tested, are utilized in the last phase of the process where we use simple non supervised

machine learning techniques to look for the correlation between them and the ToL.

Chapter 4

Results

At the end of the evaluation phase, we obtain numerous statistics for each game. More specifically, we have got the average ToL, obtained from 5 different training times, and other 14 features. These 14 features correspond to the number of actions, 8 reward features, and 6 frame-related features. For each of the frame-related features, we have recorded 3 values to prove if the delta between frames affects the result. Our goal is to find the correlation between the mentioned ToL value with the other game statistics.

Game	ToL	Actions	Score	Frequency	Number	Delta	Reward_M	Reward_C	Reward_E	MSE_M	MSE_CoV	MSE_E	SSIM_M	SSIM_CoV	SSIM_E
chopper_command	0,000	1,000	0,006	0,277	0,000	0,004	0,116	0,751	0,252	0,021	0,364	0,438	0,951	0,026	0,189
air_raid	0,091	0,143	0,057	0,073	0,278	0,055	0,052	0,770	0,599	0,299	0,773	0,534	0,680	0,278	0,824
breakout	0,136	0,000	0,002	0,141	0,148	0,002	0,004	0,796	0,455	0,027	0,273	0,581	0,982	0,018	0,666
atlantis	0,330	0,000	1,000	0,107	0,547	1,000	0,500	0,920	0,356	0,882	1,000	0,479	0,838	0,485	1,000
crazy_climber	0,364	0,357	0,589	0,024	1,000	0,363	0,150	0,746	0,289	0,077	0,264	0,483	0,673	0,445	0,906
boxing	0,386	1,000	0,001	0,011	0,168	0,000	0,001	0,808	0,066	0,127	0,043	0,266	0,852	0,026	0,397
assault	0,409	0,214	0,012	0,058	0,164	0,011	0,018	0,714	0,108	0,256	0,331	0,496	0,814	0,254	0,657
battle_zone	0,409	1,000	0,111	0,461	0,016	0,107	1,000	0,751	0,712	0,186	0,634	0,243	0,609	0,569	0,693
carnival	0,500	0,143	0,028	0,003	0,049	0,023	0,118	0,905	0,000	0,199	0,176	0,446	0,811	0,131	0,000
bank_heist	0,511	1,000	0,003	0,055	0,091	0,003	0,009	0,700	0,832	1,000	0,198	0,432	0,309	0,636	0,492
beam_rider	0,545	0,357	0,042	0,196	0,171	0,041	0,058	0,829	0,286	0,907	0,586	0,264	0,000	1,000	0,949
fishing_derby	0,557	1,000	0,000	0,027	0,264	0,000	0,000	0,000	0,097	0,129	0,000	1,000	0,732	0,031	0,674
ms_pacman	0,625	0,357	0,012	0,000	0,264	0,000	0,013	0,967	0,028	0,057	0,066	0,363	0,927	0,024	0,477
berzerk	0,773	1,000	0,004	0,314	0,237	0,004	0,043	0,704	1,000	0,162	0,266	0,667	0,826	0,232	0,371
demon_attack	0,773	0,143	0,048	0,123	0,008	0,002	0,037	0,797	0,033	0,152	0,672	0,000	0,886	0,179	0,991
alien	0,864	1,000	0,009	0,013	0,302	0,053	0,012	1,000	0,474	0,054	0,128	0,251	0,904	0,083	0,507
asteroids	0,875	0,714	0,006	0,188	0,180	0,007	0,049	0,784	0,113	0,251	0,292	0,736	0,839	0,093	0,421
bowling	0,875	0,143	0,000	1,000	0,020	0,003	0,003	0,734	0,765	0,000	0,158	0,747	1,000	0,000	0,602
amidar	0,886	0,429	0,002	0,061	0,000	0,000	0,004	0,943	0,238	0,067	0,082	0,608	0,956	0,011	0,573
double_dunk	0,920	1,000	0,000	0,420	0,000	0,000	0,000	0,325	0,282	0,100	0,156	0,608	0,929	0,042	0,710
asterix	1,000	0,357	0,039	0,041	0,090	0,002	0,062	0,753	0,216	0,140	0,038	0,517	0,865	0,024	0,562
seaquest	1,000	1,000	0,022	0,059	0,046	0,000	0,022	0,724	0,000	0,072	0,023	0,713	0,852	0,026	0,727

Figure 4.1. Table storing part of the gathered data. The data are normalized and color-coded. The colors go from the dark green for the lowest value to red for the highest value.

4.1 Result analysis

We have computed Pearson's and Spearman's correlation coefficient and applied the regression analysis between the game data that we have collected during the game evaluation phase and their corresponding time of learning. All the computations have been executed on the normalized data since the high difference in magnitude can affect the results. All the data have been normalized following the formula

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$$

This guarantees that all the data are in the interval [0, 1].

4.1.1 Correlation Coefficients results

We compute both the Pearson correlation coefficient and Spearman's rank correlation coefficient for all the games' features gathered. The results are displayed in table 4.1.
We can notice that the feature with the highest correlation is the coefficient of variation of the mean square error computed on frames taken every delta frames. The average coefficient value is -0.5, which is a strong indication of the presence of a relationship between the two variables. The correlation is inverse since both Pearson's and Spearman's coefficients are negative. The results show that higher is the variation in the frame differences lower is the ToL of the game.
Another consideration that can be extracted is that the delta used to compare the frames has almost no impact on the results. All the features have very similar coefficient values even if they have been computed with different deltas. Since multiple features with different delta are redundant it is optimal to select only one of them for future calculations. We have selected the features computed with a delta of 60 frames since it is a good trade-off between the precision of the measurement and the amount of data that have to be stored and analyzed.
The magnitude of Pearson's coefficient of few other features is higher than 0.25. One of those features is the coefficient of variation of the SSIM. Even its Spearman coefficient values are close to -0.4. The correlation between the SSIM's CV and the ToL should be the same that links the MSE's CV and the ToL since the two measurements quantify the difference between frames. We deduce that the MSE is the better measurement to study the frames' differences in this study.
The other features that have both coefficients with a magnitude above 0.25 are the score and mean reward. The two statistics are logically connected. A high average reward means an average high score. It is especially interesting to study the correlation between the average reward and the ToL since it is a quantity easy to compute.
All other features have correlation coefficients with a magnitude lower than 0.25.

Too low to convincingly sustain the presence of any correlation between one of them and the ToL.

	Pearson	Spearman
#actions	0,2103	0,2772
Score	-0,2678	-0,2934
Delta Score	-0,2463	-0,2849
REWARDS		
Frequency	0,1644	0,0560
Number	-0,2350	-0,2159
Reward_M	-0,2620	-0,3448
Reward_CV	-0,0599	-0,1012
Reward_E	-0,0873	-0,1577
MSE		
MSE_30_M	-0,1955	-0,1668
MSE_60_M	-0,1880	-0,1899
MSE_120_M	-0,1735	-0,1815
MSE_30_CV	-0,4119	-0,4884
MSE_60_CV	-0,4693	-0,5732
MSE_120_CV	-0,5039	-0,5461
MSE_30_E	0,2497	0,3624
MSE_60_E	0,1872	0,3522
MSE_120_E	0,1227	0,2804
SSIM		
SSIM_30_M	0,1833	0,3363
SSIM_60_M	0,1778	0,3058
SSIM_120_M	0,1640	0,2103
SSIM_30_CV	-0,2643	-0,2826
SSIM_60_CV	-0,2749	-0,3557
SSIM_120_CV	-0,2655	-0,3793
SSIM_30_E	0,0192	-0,0848
SSIM_60_E	0,0007	-0,0599
SSIM_120_E	-0,0530	-0,0690

Table 4.1: Table containing all the correlation coefficients results. The final letter after the underscore of some features indicates that the feature is the mean in the case of M, the coefficient of variation in the case of CV, or the entropy in the case of E. For example, Reward_M indicates the reward's mean and SSIM_E is the entropy of the frames' SSIM. In the case of the MSE and SSIM the number indicates the frames delta.

4.1.2 Regression Results

To further analyze the possible existing correlation, we have also analyzed the data with a linear regression model. As well as study the correlation between just one variable and the ToL, we have also studied the correlation between the time of learning and every pair of the other features. The pair order does not influence the results.

First of all, we start by analyzing the R^2 and the $adjusted R^2$ of all the regressions computed. The results are displayed in table 4.2. We can notice how the best fit planes with the higher R^2 are the ones in which one of the two features is the CV of the MSE. All these R^2 are between 0.307 and 0.22. Adding a new independent variable to the model always increases the R^2. If we compare the R^2 of the model with only the MSE's CV as variable and the ones in which is paired with other features we can notice that in many cases there is a very marginal increase. We can therefore claim that the extra feature paired with the CV of the MSE is not very effective. The paired variable that better improves the R^2 values is the SSIM's entropy. The pair CV of MSE and SSIM entropy is the only one with a R^2 higher than 0.3. If we analyze the $adjusted R^2$ values we can notice that the best performing model remains the one with the MSE's CV and SSIM entropy. The model with only the MSE's CV is the third-best model $adjusted R^2$ wise. This result confirms what the correlation coefficients indicate.

All the models containing the CV of MSE have a p-value associated with the F-test lower of 0.05. We can therefore be more than 95% confident that these results are not due to chance. If we analyze the t-values of the top-performing pair we discover that we cannot reject the null hypothesis for the SSIM entropy. Its p-value associated with the t-test, when paired with the CV of MSE, is 0.139. Too high for a 95% confidence level. On the other hand, the p-value of the CV of MSE is less than 0.009 in that regression model. The linear regression analysis concludes that the only correlation present to be statistically significant is the one between the ToL and the MSE's CV.

The good correlation coefficient values for the average reward do not correspond with the regression analysis results. Both the R^2 and $adjusted R^2$ are lower than 0.1. Furthermore, the statistical tests show a probability of 23.9 % that the regressor value is due to random chance. In conclusion, the correlation between the ToL and the mean reward is too weak to be considered for the predictive model.

Features	R^2	**Adj.**R^2	**F-stat**	**P>|F|**	t_1	$P > \|t_1\|$	t_2	$P > \|t_2\|$
MSE_CV SSIM_E	0,307	0,234	4,208	0,0307	-2,901	0,009	1,542	0,139
Frequency MSE_CV	0,263	0,186	3,398	0,0548	1,056	0,304	-2,47	0,023
MSE_CV	0,22	0,181	5,648	0,0276	-2,376	0,028		
Reward_E MSE_CV	0,238	0,158	2,974	0,0752	0,674	0,509	2,399	0,027
Number MSE_CV	0,236	0,155	2,928	0,0779	-0,619	0,543	-2,118	0,048
MSE_M MSE_CV	0,223	0,142	2,733	0,0905	0,28	0,783	-2,145	0,045

Reward_Cov MSE_CV	0,223	0,142	2,731	0,0907	0,275	0,786	-2,318	0,032
Delta MSE_CV	0,222	0,141	2,718	0,0916	0,235	0,817	-1,988	0,061
MSE_CV SSIM_M	0,222	0,14	2,711	0,0921	-2,156	0,044	0,209	0,837
Actions MSE_CV	0,221	0,139	2,689	0,0937	0,096	0,925	-2,073	0,052
Reward_M MSE_CV	0,22	0,138	2,684	0,0941	-0,044	0,965	-1,922	0,07
Score MSE_CV	0,22	0,138	2,684	0,0941	-0,041	0,968	-1,903	0,072
MSE_CV MSE_E	0,22	0,138	2,683	0,0941	-2,124	0,047	0,009	0,993
MSE_CV SSIM_CV	0,22	0,138	2,683	0,0941	-1,877	0,076	-0,013	0,99
SSIM_CV	0,076	0,029	1,635	0,216	-1,279	0,216		
Actions Reward_M	0,119	0,026	1,282	0,3	1,041	0,311	-1,269	0,22
Frequency Reward_M	0,119	0,026	1,277	0,302	1,037	0,313	-1,404	0,176
Score	0,072	0,025	1,545	0,228	-1,243	0,228		
Reward_M	0,069	0,022	1,474	0,239	-1,214	0,239		
Number Reward_M	0,115	0,022	1,24	0,312	-1,003	0,329	-1,137	0,27
Actions SSIM_CV	0,111	0,018	1,189	0,326	0,873	0,393	-1,197	0,246
Score SSIM_CV	0,108	0,015	1,156	0,336	-0,837	0,413	-0,885	0,387
Delta	0,061	0,014	1,292	0,269	-1,136	0,269		
Number SSIM_CV	0,105	0,011	1,114	0,349	-0,79	0,439	-1,028	0,317
Number	0,055	0,008	1,169	0,293	-1,081	0,293		
Delta SSIM_CV	0,102	0,007	1,078	0,36	-0,746	0,465	-0,934	0,362
Reward_M SSIM_CV	0,102	0,007	1,075	0,361	-0,743	0,466	-0,836	0,413
SSIM_M SSIM_CV	0,101	0,007	1,07	0,363	-0,736	0,471	-1,213	0,24
Frequency SSIM_CV	0,101	0,006	1,062	0,365	0,726	0,477	-1,246	0,228
Score MSE_E	0,1	0,005	1,056	0,368	-1,171	0,256	0,773	0,449
Score SSIM_M	0,099	0,004	1,045	0,371	-1,193	0,248	0,759	0,457
Score Reward_M	0,097	0,002	1,02	0,38	-0,771	0,45	-0,728	0,475
SSIM_CV SSIM_E	0,094	-0,002	0,9809	0,393	-1,401	0,177	0,615	0,546
Score SSIM_E	0,093	-0,002	0,9756	0,395	-1,397	0,179	0,67	0,511
Actions	0,044	-0,004	0,9258	0,347	0,962	0,347		
Delta SSIM_M	0,09	-0,006	0,9401	0,408	-1,105	0,283	0,783	0,443
Score Frequency	0,09	-0,006	0,9374	0,409	-1,145	0,266	0,614	0,546
Delta MSE_E	0,089	-0,006	0,9334	0,411	-1,066	0,3	0,775	0,448
Reward_M MSE_M	0,089	-0,007	0,9311	0,411	-1,061	0,302	-0,656	0,52
Delta Reward_M	0,088	-0,008	0,9167	0,417	-0,635	0,533	-0,755	0,46
Reward_M SSIM_M	0,088	-0,008	0,9137	0,418	-1,081	0,293	0,631	0,536
Actions Score	0,087	-0,009	0,9109	0,419	0,573	0,573	-0,949	0,355
MSE_E SSIM_CV	0,084	-0,012	0,8706	0,435	0,417	0,681	-1,007	0,327
Reward_M MSE_E	0,083	-0,013	0,8624	0,438	-0,999	0,33	0,55	0,589
MSE_M	0,035	-0,013	0,7325	0,402	-0,856	0,402		
MSE_E	0,035	-0,013	0,7264	0,404	0,852	0,404		
Actions SSIM_M	0,083	-0,014	0,8596	0,439	1,032	0,315	0,896	0,382
Number SSIM_E	0,082	-0,014	0,853	0,442	-1,306	0,207	0,75	0,462

Number MSE_E	0,082	-0,014	0,8518	0,442	-0,989	0,335	0,749	0,463
Number MSE_M	0,082	-0,015	0,8464	0,444	-0,981	0,339	-0,742	0,467
Frequency Delta	0,081	-0,016	0,8338	0,45	0,643	0,528	-1,053	0,305
Score MSE_M	0,08	-0,017	0,8281	0,452	-0,962	0,348	-0,418	0,681
SSIM_M	0,032	-0,017	0,653	0,429	0,808	0,429		
MSE_M SSIM_CV	0,079	-0,018	0,8135	0,458	0,261	0,797	-0,948	0,355
Score Delta	0,079	-0,018	0,8118	0,459	-0,61	0,549	0,38	0,708
Number SSIM_M	0,079	-0,018	0,811	0,459	-0,985	0,337	0,695	0,495
Reward_M Reward_E	0,079	-0,018	0,8101	0,46	-1,21	0,241	-0,453	0,656
Actions Delta	0,078	-0,019	0,8082	0,46	0,605	0,552	-0,839	0,412
Delta SSIM_E	0,078	-0,019	0,8008	0,464	-1,266	0,221	0,593	0,56
Reward_E SSIM_CV	0,077	-0,02	0,7945	0,466	-0,182	0,858	-1,197	0,246
Reward_CV SSIM_CV	0,076	-0,021	0,7836	0,471	-0,113	0,911	-1,222	0,237
Actions Number	0,076	-0,022	0,7776	0,474	0,648	0,524	-0,804	0,432
Frequency	0,027	-0,022	0,5554	0,465	0,745	0,465		
Score Number	0,075	-0,023	0,7689	0,477	-0,636	0,533	-0,255	0,802
Score Reward_E	0,073	-0,024	0,7502	0,486	-1,159	0,261	-0,173	0,864
Score Reward_CV	0,072	-0,026	0,7357	0,492	-1,182	0,252	-0,056	0,956
Actions MSE_M	0,072	-0,026	0,7341	0,493	0,863	0,399	-0,75	0,462
Number Delta	0,072	-0,026	0,7329	0,494	-0,474	0,641	-0,58	0,569
Reward_M SSIM_E	0,071	-0,026	0,7291	0,495	-1,208	0,242	0,232	0,819
Actions Frequency	0,07	-0,027	0,7202	0,499	0,942	0,358	0,732	0,473
Actions MSE_E	0,07	-0,027	0,72	0,5	0,851	0,406	0,732	0,473
Reward_M Reward_CV	0,069	-0,029	0,7089	0,505	-1,16	0,261	-0,128	0,9
Delta MSE_M	0,068	-0,03	0,6979	0,51	-0,822	0,421	-0,398	0,695
Delta Reward_E	0,062	-0,037	0,6293	0,544	-1,051	0,307	-0,172	0,865
Frequency Number	0,062	-0,037	0,6246	0,546	0,362	0,721	-0,838	0,413
Delta Reward_CV	0,061	-0,038	0,6155	0,551	-1,076	0,295	-0,061	0,952
MSE_M MSE_E	0,059	-0,04	0,5945	0,562	-0,694	0,496	0,69	0,499
Number Reward_CV	0,058	-0,041	0,5818	0,569	-1,045	0,309	-0,224	0,825
Number Reward_E	0,057	-0,042	0,5737	0,573	-0,997	0,331	-0,187	0,853
Reward_E	0,008	-0,042	0,1537	0,699	-0,392	0,699		
MSE_E SSIM_M	0,054	-0,045	0,5442	0,589	0,673	0,509	0,62	0,543
Frequency MSE_M	0,054	-0,045	0,5436	0,589	0,614	0,546	-0,738	0,47
Reward_CV	0,004	-0,046	0,07205	0,791	-0,268	0,791		
Frequency MSE_E	0,052	-0,048	0,5223	0,601	0,585	0,566	0,709	0,487
Frequency SSIM_M	0,052	-0,048	0,5201	0,603	0,638	0,531	0,706	0,489
Actions SSIM_E	0,05	-0,05	0,4969	0,616	0,997	0,331	0,33	0,745
SSIM_E	0	-0,05	1,06E-05	0,997	0,003	0,997		
Actions Reward_E	0,045	-0,055	0,452	0,643	0,867	0,397	0,153	0,88
Actions Reward_CV	0,045	-0,056	0,4459	0,647	0,906	0,376	0,109	0,915
MSE_M SSIM_E	0,039	-0,062	0,3866	0,685	-0,879	0,39	0,273	0,788

Reward_CV MSE_E	0,038	-0,063	0,3798	0,689	0,259	0,798	0,83	0,417
MSE_M SSIM_M	0,038	-0,063	0,3773	0,691	-0,361	0,722	0,238	0,814
Reward_E MSE_M	0,038	-0,063	0,3765	0,691	-0,235	0,817	-0,776	0,447
Reward_CV MSE_M	0,037	-0,064	0,3653	0,699	-0,183	0,857	-0,812	0,427
Reward_E MSE_E	0,037	-0,065	0,362	0,701	-0,181	0,858	0,757	0,458
Reward_CV SSIM_M	0,036	-0,065	0,3597	0,703	-0,31	0,76	0,805	0,431
MSE_E SSIM_E	0,036	-0,066	0,3523	0,708	0,839	0,412	0,118	0,907
Reward_E SSIM_M	0,035	-0,066	0,3495	0,709	-0,276	0,786	0,741	0,468
SSIM_M SSIM_E	0,035	-0,067	0,3404	0,716	0,825	0,42	0,242	0,812
Frequency Reward_E	0,03	-0,072	0,2911	0,751	0,658	0,519	-0,23	0,82
Frequency Reward_CV	0,028	-0,074	0,2769	0,761	0,695	0,495	-0,16	0,875
Frequency SSIM_E	0,027	-0,075	0,2639	0,771	0,727	0,476	-0,015	0,988
Reward_CV Reward_E	0,009	-0,095	0,09082	0,914	-0,188	0,853	-0,336	0,741
Reward_E SSIM_E	0,009	-0,096	0,0819	0,922	-0,405	0,69	0,133	0,896
Reward_CV SSIM_E	0,004	-0,101	0,03464	0,966	-0,263	0,795	-0,029	0,977

Table 4.2: The table contains F-value, t-values, and corresponding p-value for single feature and pair of features. The features with the nomenclature MSE and SSIM are the ones computed on MSE and SSIM of frames taken 60 frames apart.

4.2 Result discussion

Our original hypothesis was that a big difference between adjacent frames would make a game easier to learn for the model. This idea was based on the fact that the DDQN model uses as states representation the game's output frames. A game in which the frame change often should be easier to learn because the model receives more information with respect to a game in which the frames a very similar between each other. We also suspect that some of the characteristics of the reward signal could affect the ToL. The only correlation that we have found is instead with the coefficient of variation of the mean square error of the adjacent frames. This means that the ToL is not affected if the adjacent frames are very dissimilar from each other. We have demonstrated this statement by proving that there is no correlation between the average MSE and the ToL. What instead affects the ToL is the variation of the variation of the frames. The games that are learned faster are the ones in which the CV of the MSE is high.
What we also discovered is that there is not any evident relationship between the frequency, magnitude, or number of rewards received by the model during a game and its ToL. The lack of correlation with the reward magnitude can be explained by the fact that the algorithm, for stability reasons, clips the reward between -1 and 1. It was instead surprising to find that the frequency and number of the rewards do

not correlate with the difficulty with which the model learns the game. More and more frequent rewards should help the system to sooner reach the correct estimation of each action Q-value. Our data say that that is not true. The correlation between the number of rewards and ToL may be present, but more sophisticated statistical tools are needed to highlight it.

4.3 Model

The correlation analysts showed the presence of a relationship between the mean Coefficient of Variation computed between frame 60 frames apart. We can therefore exploit this relationship to create a model the can predict the ToL based only on the characteristic of the game. Since the CV of the MSE is a feature strictly related to the game itself it is relatively easy to compute its average in a small amount of time. Analyzing the data gathered from us on the CV of the MSE it is possible to notice that such measure is quite stable during all the training phase. This quality allows us to have a good estimation by training an agent only for few epochs. Usually, no more than 50 epochs are needed to obtain a valid estimation of the average CV of the MSE. Thanks to the estimation of the CV and the linear regression model we can therefore forecast the number of epochs needed for the agent to learn the game. The epochs utilized for the estimation of the CV are also useful to estimate how long does it takes to complete an epoch on the currently used hardware. Multiplying the estimated number of epochs and the time needed to finish an epoch it is possible to predict the time needed to train an agent.
If we call ECV the estimated CV of the MSE and ET the estimated time for the computation of one epoch it is possible to predict the game ToL by applying the formula

$$ToL = (380.62 - 69.19ECV)ET$$

Where 393.39 and -71.56 are the $beta_0$ and $beta_1$ obtained through the regression analysis.

Chapter 5

Discussion

5.1 Future work

The work described in this thesis can be expanded and improved. We discuss in this section some ways in which that can be done.

5.1.1 Game clustering based on obtained data

The learning curves produced by DQN training of Atari games have already been studied in other research works. In particular, after performing hierarchical clustering on the learning curves, three clusters of games have been found [25]. It is interesting to study if it is possible to perform clustering starting from the data we collected to reproduce the same clusters. If achieved, this will prove that the form of the learning curves is correlated with some of the game's characteristics. Furthermore, by studying what makes games belong to the same cluster, it may be possible to bring to light new insights on which game's qualities affect the ToL.

5.1.2 Analyse more complex games

Even though the Atari dataset is a commonly used benchmark for the deep reinforcement learning model, an agent able to play Atari games is not very useful to the solution of many real-world problems. Most of the modern games are vastly more complex with respect to the Atari ones. For the prediction tools theorized in this work to be more useful more complex and modern games have to be studied. Due to the limited computational power at our disposal we have focused our research on Atari games. The DQN model and the models derived from it have been proven effective also on more complex games like Doom. Gathering and analyzing data from more recent games could bring the creation of a more useful and precise prediction tool.

5.1.3 More advanced data analysis

Our data analysis has been focused to find a linear relationship between the ToL and the games' features. A linear correlation is the easiest to find and study. A more detailed data analysis can be performed using more complex unsupervised machine learning algorithms.
It is possible the relationship between the ToL and a subset of the games' features is not governed by a linear model. Studying the data using polynomial regression models may discover a correlation that our linear analysis could not found. Therefore a more complete data analysis may increase the precision of the model.

5.2 Conclusion

5.2.1 Summary

In this thesis, we tackled the problem of the high variance in the time needed to train AI agents to play different videogames.
The goal is to identify which characteristics make a game complex to be learned by an AI agent and how this complexity affects the time of learning. More precisely, we have studied if it is possible to predict how long it takes for a selected model to learn a game. This prediction is based solely on the game features.
For our research, the games selected are Atari games and the model selected is Double DQN. DDQN is a deep reinforcement learning algorithm able to play at a superhuman level Atari games.
We have achieved our goal by modifying an existing DDQN model to gathered data from tens of Atari games during the training phase. The data collected describe two main aspects of the game: the shape of the reward signals and the visual component. The shape of the rewards is a key aspect of reinforcement learning. Reward frequency and magnitude can heavily influence the model performance. The visual component is considered because the DDQN uses as input the frame's pixels. We then used unsupervised machine learning techniques, like linear regression analysis, to research the correlation between the game characteristics and the training duration. The data analysis brought to light the presence of an inverse correlation between the ToL and the coefficient of variation of the mean square error taken between frames 60 frames apart. The correlation found differs from our original hypothesis. Our idea was that the more the consecutive frames are different from each other more easily the model would be able to learn the game. We instead found that the average MSE is not a good indicator to forecast the ToL. What matters is the variation of the variation of the frames. If the MSE of adjacent frames is constant during the game, even if high, the model has a harder time learning the game with respect to a game in which the MSE between adjacent frames is variable.

Bibliography

[1] N. Shaker, G. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 5, no. 1, 2010.

[2] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 2010, pp. 265–272.

[3] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *2007 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2007, pp. 252–259.

[4] G. Sviridov, C. Beliard, A. Bianco, P. Giaccone, and D. Rossi, "Removing human players from the loop: Ai-assisted assessment of gaming qoe," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1160–1165.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] F. S. Melo, "Convergence of q-learning: A simple proof," *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[7] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *arXiv preprint arXiv:1811.12560*, 2018.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] "Convolutional neural network," https:/en.wikipedia.org/wiki/Convolutional{_}neural{_}network, accessed: 2021-02-15.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*,

vol. 30, no. 1, 2016.

[12] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, vol. 12, no. 1, pp. 1–20, 2019.

[13] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *arXiv preprint arXiv:1507.06527*, 2015.

[14] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[15] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[17] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[18] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[19] M. Bellemare, J. Veness, and M. Bowling, "Investigating contingency awareness using atari 2600 games," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012.

[20] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[21] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," 2016.

[22] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[23] "Hcp@polito," https://hpc.polito.it/, accessed: 2021-03-15.

[24] "Slurm worklod manager," https://slurm.schedmd.com/documentation.html, accessed: 2021-03-15.

[25] E. Emekligil and E. Alpaydın, "What's in a game? the effect of game complexity on deep reinforcement learning," in *Workshop on Computer Games*. Springer, 2018, pp. 147–163.