Atlantis Studies in Uncertainty Modelling, volume 3
Joint Proceedings of the 19th World Congress of the International Fuzzy Systems Association (IFSA), the 12th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT), and the 11th International Summer School on Aggregation Operators (AGOP)

Machine-imitative Learning by Using Computational Perceptions as Labeled Data-sets: A first Empirical Approximation

*Clemente Rubio-Manzano [a,b] and **Tomás Lermanda** [b] and **Claudia Martinez** [c] and **Alejandra Segura** [b] and **Christian Vidal** [b]
[a] Department of Mathematics, University of Cadiz, clemente.rubio@uca.es
[b] Department of Information Systems, University of the Bío-Bío clrubio@ubiobio.cl
[c] Computer Science Department, Universidad Catolica de la Santisima Concepcion

Abstract

In this paper, we show how computational perceptions can be employed as labeled datasets to train agents in computer games. The idea is to automatically create a correspondence between perceptions and movements by using computational perception networks, next this knowledge is learned by the agents by using a decision tree. The result is a machine-imitative learning model able to mimic the human players. This approach is formally presented, a problem formulation based on the combination of linguistic descriptions of phenomena and classification is carried out. Additionally, we present a software architecture and module is explained. Finally, this architecture has been implemented and tested.

Keywords: Computational Theory of Perceptions, Imitative Learning, Machine Learning, Intelligent Agents, Computer Games.

1 Introduction

Artificial Intelligence (AI) aims to get computational systems with the capability of performing intelligent tasks. One of the most important tasks in AI is the ability to imitate. The idea of learning/teaching by imitation is not a new one, however, it is gaining attention due to advances in computing, sensing and big data [2, 3, 14, 16]. In [6] there were pointed out two important advantages: i) it facilitates teaching complex tasks with minimal expert knowledge of the tasks; ii) reduce the problem of teaching a task to that of providing demonstrations without the need for explicit programming or designing reward functions specific to the task.

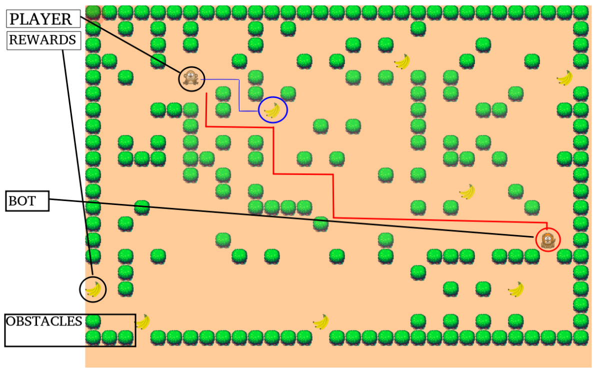

Figure 1: Computer game employed to implement machine-imitative learning model

More specifically, imitation learning techniques aim to mimic human behavior in a given task. An agent is trained to perform a task from demonstrations by learning a mapping between perceptions and actions [14]. This suggests that the Computational Theory of Perceptions (CTP) [15], and therefore, the management of imprecision plays an important role in achieving good models of machine-imitative learning.

The design of a computational system based on CTP consists of defining a hierarchy of levels of abstraction [12]: Domain of Experience/Language; Granular World; Computational Perceptions of order n; Computational Perceptions of first order. In our case, video games have been selected as the domain of experience because they have become an alternate, low-cost yet rich environment for assessing machine learning algorithms [4]. Additionally, some of the most important and revolutionary research works in machine learning in the last years employ computer games for testing their algorithms [1, 5, 7]. In this work, we employ our own 2D computer game based on the pac-man game created for this research. Other versions of this computer game have been previously used in [11, 10, 9, 8]. In each play session, the player navigates through a maze containing rewards and one opponent. The goal of the game is to collect all the rewards and escaping from the opponent. The opponent roams the scene and chase the player. The game ends when all rewards has been collected or when the player is captured. Therefore, the aim is to learn how human players play and creating intelligent agents that mimic human players in this kind of computer games (see Figure 1).

Although several learning algorithms have been applied in computer games, none have actually been used to dictate agent behavior directly by using CTP. More precisely, it is intended to create a general method that allows it to map computational perceptions onto moves for training a bot from the human play sessions. To do this, the behavior of the players will first be captured from a computational perception network that will be created from the observations (execution traces) of the games (play-sessions). Later, we will use the computational perception network to create a labeled dataset that will be used to generate a decision tree. Finally, the machine learning model is employed to control the movements of the bot.

The rest of the article is organized as follows. In Section 2 a problem formulation of the machine-imitative learning process is performed. In Section 3 a software architecture is presented and its modules are explained. In Section 4 the implementation of the modules is detailed and a function test is performed. Finally, conclusions and future work are presented in Section 5.

2 Machine-Imitative learning on video games

In this section, we introduce the concept of machine-imitative learning from a theoretical point of view by defining important concepts as state of a video game, observation space, precise observation vector, linguistic vector, action vector, policy and a process of imitative learning. Second, we show how this learning process can be implemented by using a combination of two computational techniques: linguistic descriptions of phenomena (what do we want to learn) and classification (how we want to do it). A complete test will be carried out in the section 4.

2.1 Problem formulation

Definition 1 *A state of a video game is defined by a vector that will contain the information about what is happening in the play session at that moment i. We call this vector the vector of the play session $\overline{s_i} \in \mathscr{S}$, with $\mathscr{S}$ being the video game space where every state can be represented.*

Definition 2 *An observation space $\mathscr{O}$ represents the observable information for an agent and is therefore a subspace of $\mathscr{S}$. We define an observation as a vector $\overline{o} \in \mathscr{O}$ with $\mathscr{O} \subset \mathscr{S}$.*

A set of observations must be selected. The selection is made considering the features that intervene in the behavior and that are believed to be key to implement the imitation process.

Definition 3 *A precise observation vector is a set $v = (v_1, v_2, \ldots, v_n)$ where $v \in \mathscr{O}$ and each v_i is is defined by using a precise feature (for example position, distance, protection, so on)*

As some features may be imprecise, we will handle it using computational perception vectors.

Definition 4 *Given a precise observation vector $v = (v_1, v_2, \ldots, v_n)$; For each v_i, there may be m perceptions $(l_1, l_2, \ldots, l_m)$ that will depend on the fuzzy linguistic definition of each v_i. A computational perception vector l_{v_i} is defined as: $L_{v_i} = (l_1^{vi}, l_2^{vi}, \ldots, l_m^{vi})$ a pair (L_{v_i}, W) which is described as follows:*

$L_{v_i} = (l_1^{v_i}, \ldots, l_m^{v_i})$ is a linguistic vector that represents the whole linguistic domain of v_i

$W = (w_1, \ldots, w_n)$ is a vector of the validity degrees $w_i \in [0,1]$ of each l_i. w_i represents the suitability of l_i to describe the current perception of a specific aspect of the monitored phenomenon.

All computational perception vectors are grouped and called CPs vector.

Definition 5 *An action vector is a set $a = (a_1, a_2, \ldots, a_m)$ where $a \in \mathscr{A}$ and $\mathscr{A} \subset \mathscr{S}$.*

Definition 6 *A policy is a function that maps a CPs vector to an action vector. It is what the agent uses to decide which action to execute*

Definition 7 *The process of imitation learning is one by which an agent in a video game is capable of learning a policy. That is, it learns a relation P: CPs $\rightarrow$ A*

2.2 Linguistic descriptions of complex phenomena

Automatic Linguistic Description of Complex Phenomena (LDCP) aims to extract and represent knowledge by using natural language sentences (report) as if they were produced by a human expert, describing the most relevant aspects of a phenomenon for certain

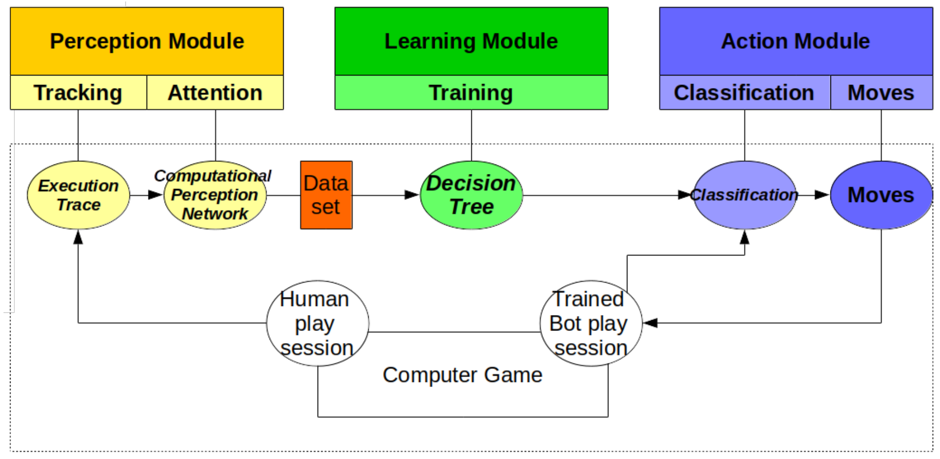

Figure 2: Software architecture for machine-imitative learning on computer games

users in specific contexts [13]. LDCP is based on the CTP.

The basic architecture is based on the concept of Computational Perception (CP). A CP takes as values the elements of a set of linguistic expressions (words or sentences in natural language) that represents all the possible values of the perception (e.g.:"The current situation is { dangerous, safe, easy, risky }").

We apply this methodology to generate labeled datasets during play sessions. The result is a method able to collect and interpret events performed by human players in the game world during play sessions, yielding knowledge about their actions and relating these with their behaviors (attitudes, situations or movements). The result of this process is a network (see Figure 3) in which each computational perception covers a specific aspect of the events occurred with certain granularity degree. Defining this network is an iterative process that starts with the definition of the top-order computational perceptions. Here, it is a matter of answering general questions about how was a particular session and which was the behavior of the players. Here we have to take into consideration several features: (i) the kind of situations produced between the player and the opponent, (ii) the kind of attitude showed by the player in a particular situation, (iii) kind of movements performed by the player. From these features we could answer one important question: which are the actions performed by the players based on their behavior.

Example 1 *A computational perception network can be defined by using a set of linguistic if-then rules (see Table 1, 2, 3). Being $CP-D_{po}$ the distance between player and opponent; $CP-D_{pc}$ the distance between player and the closest reward; $CP-D_{oc}$ the distance between the opponent and the closest reward.*

Table 1: Rules for CP of Situation

CP-Situation	CP-Protection	$CP-D_{po}$
Risky	High	Close
Dangerous	Low	Close
Safe	High	Far
Easy	Low	Far

Table 2: Rules for CP of Attitude

CP-Attitude	$CP-D_{pc}$	$CP-D_{oc}$
Wise	Close	Far
Brave	Close	Close
Cautious	Far	Close
Passive	Far	Far

Table 3: Rules for 2CP of Movements

2CP-Movement	$1CP-D_{pc}$	$1CP-D_{po}$
Good	Close	Far
Scared	Far	Far
Kamikaze	Close	Close
Bad	Far	Close

Of course, the definition of the linguistic variables and its associated fuzzy sets depend on the user (a possible definition can be found in[11, 8]). For example, for the three perceptions defined (CP Attitude, CP Movement, CP Situation) and assuming we have identified two actions (Go_Ahead, Get_Away), the following rules could be defined: R1) Go_Ahead $\leftarrow$ Wise, Good, Safe; R2) Go_Ahead $\leftarrow$ Wise, Scared, Safe; R3) Go_Ahead $\leftarrow$ Wise, Good, Easy; R4) Go_Ahead $\leftarrow$ Wise, Scared, Easy; R5) Go_Ahead $\leftarrow$ Wise, Scared, Eafe; R6) Go_Ahead $\leftarrow$ Wise, Scared, Easy; R7) Get_Away in any other case.

2.3 Classification

The learning process is performed by using a supervised model where actions will be automatically computed by the system. That is, in our approach instances are automatically labeled by the computational perception network. Then, the model is capable of predicting the appropriate action when a particular situation is presented. A classification model has been selected for our imitation learning process because these kind of approaches are usually employed when the player's actions can be categorized into discrete classes. This is suitable for applications where the actions can be viewed as a decision such as navigation and flight simulations [6].

Classification is a popular task in machine learning where observations are automatically categorized into a finite set of classes. A classifier $h(x)$ is used to predict the class y to which an independent observation x belongs; where $y \in Y$, $Y = \{y_1, \ldots, y_p\}$ is a finite set of classes, and $X = \{x_1, \ldots, x_m\}$ is a vector of m features. In supervised classification, $h(X)$ is trained using a dataset of n labeled samples $(x^{(i)}, y^{(i)})$ where $x^{(i)} \in X, y^{(i)} \in Y$ and $i = 1, \ldots, n$

3 Software architecture

In this section, we describe a software architecture for learning by imitation. This is made up of five sub modules: tracking, attention, training, classification, and movements. At the same time, these modules are grouped into three modules: perception, learning, and action. In the Figure 2, each of the modules is shown together with their dependencies: 1) Tracking - Execution Traces; 2) Attention - Computational Perception Network; 3) Training - Decision Tree; Classification - Static Tree; Moves - Bot.

The **perception module** is made up of two sub modules, tracking and attention, which are used to acquire and prepare the data so that they can be processed by a computational perception network. Tracking is responsible for extracting the information from the current state of the game . The attention sub module is responsible for selecting the relevant information from the data captured in the tracking. The selection is made considering the variables that intervene in the behaviour and that are believed to be key to implementing the imitation process. In particular, we will use fuzzy logic and computational perception networks (if-then rules with linguistic variables).

The **learning module** aims to learn a relationship of the type P: CPs $\rightarrow$ A that, based on perceptions, gives us the best move to perform given the state of the game at a given moment. The learning result (computational perception decision tree) can be seen in Figure 3.

The **action module** is formed by two sub modules: classification and movements. These are in charge of carrying out the imitation process through the execution of the movements acquired in the learning phase. More precisely, the process is as follows: a row is read from the execution trace, the data are converted to CPs, which are organized in an instance, the instance is classified and the label resulting from the classification is used as input data in a function of search whose result determines the direction in which the agent moves. Once the direction is determined, the corresponding movement is carried out.

4 Implementation and testing

For the implementation of the **tracking sub module**, we use the concept of execution trace, which is a technique used to record the relevant information that occurs during the execution of the play session. Due to the nature of video games, we use them to capture and store the data generated from the movements made by human players, agents and opponents. In this way, the traces perform the observation function (key in frameworks based on learning by imitation).

The **attention sub module** is implemented using the concept of a computational perception network. We use a modified version of the network of perceptions presented in [11]; in this case, the computational perceptions (CPs) have been reduced. Although all first-order computational perceptions are maintained (1CP), two second-order computational perceptions (2CP), CP Ability and CP Skill, are eliminated. This is because our objective is not to evaluate the performance of the player but rather to use the perceptions that best represent the actions of a player. In this way, the 2 CPs used correspond to CP Situation, CP Attitude and CP Movement. The metrics, linguistic variables and rules used have been redefined for this development.

To generate the decision trees, we use J48, which is the implementation of the C4.5 algorithm from the Weka library. The resulting trees are exported as a static classifier (Java class), which provides us with a convenient function to perform the classification. The classification function is where an instance is used to create the data in the table in real time to determine the class label, the variable that determines the final decision to be made.

Agent learning was posed as a classification problem where the CPs associated with the actions of a human player are used as training data to implement a resulting static classifier in the video game to decide the

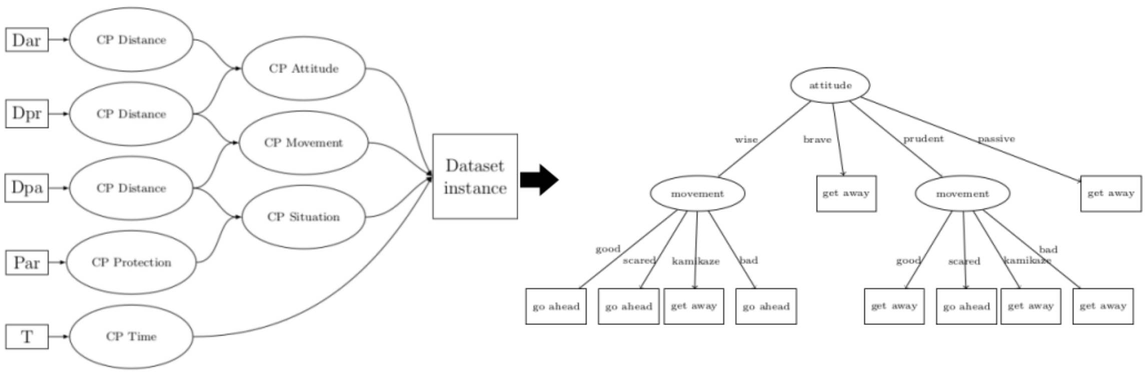

Figure 3: Computational Perception Decision Tree: a decision tree generated by using a network of computational perceptions

movements of an agent according to its own CPs. A decision tree without pruning was used as a classification method. This method is very useful in this case since the resulting classifiers can easily be expressed as a set of if-then rules. In our case, J48 provides us with static classifiers that can be easily implemented in our video game. The attributes used for the generation of the classifiers were defined from CPs, and two attributes were assigned for each perception, one for the linguistic variable and another for the degree of validity. The tags used were constant throughout the tests. The class was defined as an action: **go ahead, get away**.

The test points to the use of a minimum number of 1CP in the training data set. Three 1CPs were used, chosen based on the relevance of the metrics with which they were constructed. The 1CP used corresponded to the Protection CP, Distance CP (player-opponent), and Closeness CP. The latter (**CP Closeness**) was used exclusively for this test and it is the perception used to perceive how much closer or farther one entity is from another in relation to a third entity. Given a scenario divided into cells of equal size and considering that an entity occupies the space of a cell, the proximity is calculated according to the distance between entities, which is determined by the number of cells between them. We define Closeness C as the difference between the distances from entity a to entity b and from entity b to entity c. This can be calculated using the following equation: C = d (a, b) - d (b, c). In particular, we use this CP to perceive how much closer or farther the player is to the closest reward in relation to the opponent. We define this 1CP as follows: Z = [-N; N], where N is the maximum distance in the game scenario. A = ("much closer"; "closer"; "farther away"; "much farther away"). g: This function is constructed using four labels that are represented with trapezoidal membership functions (a,b,c,d) as follows: "much closer" (2,6, N, N), "more close" (- 2,2,2,6), "farther" (- 6, -2, -2,2), and "much farther" (- N, -N, -6, - 2).

Attribute	Description	Values
CP Protection	Number of obstacles between player and opponent.	low, medium, high
DV Protection	Degrees of validity for CP Protection	Numeric (Real)
CP Distance (player-opponent)	Distance between player and opponent.	small, medium, large
DV Distance	Degrees of validity for CP Distance	Numeric (Real)
CP Closeness	Perception used to perceive how much closer or farther one entity is from another in relation to a third entity	"much closer"; "closer"; "farther away"; "much farther away"
DV Closeness	Degrees of validity for CP Closeness.	Numeric (Real)
Action	**Action performed by the player (the class).**	**“go ahead”, “get away”**

Table 4: Test attribute list.

In this test, the class label value of each instance of the training dataset are determined by a simple rule based on the distance metric (player-opponent). The rule is that if the distance between the player and the opponent is maintained or increased, then the player is evading the opponent unless the player is ignoring the adversary. The system and a demonstration videos that show the results can be publicly accessed. The development can be found at the following URL:

`https://github.com/pp1856/Agent-Training`

The following is a demonstration video of a bot trained using LDCP and decision trees. Test was created by using 7 attributes: `https://www.youtube.com/watch?v=A8Y4by5Llng`

5 Conclusions and future work

A first empirical approximation for machine-imitative learning has been presented. Our approach is based on the theory of computational perceptions which is employed to automatically generate labeled datasets. These datasets are used to train agents in computer games by mapping the perceptions of the human player into actions. A software architecture has been presented and their main modules have main explained. Finally, aspects about the implementation of this AI system have been given and a complete test in order to check its functionality has been carried out.

As future work, much more tests using different number of attributes should be performed. A more complete evaluation about the quality of the machine-learning process and its performance need to be performed. We would like to use others learning method as neural networks. Also, we want to research how theory of computational perceptions could contribute to the design and implementation of explainable AI systems.

References

[1] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, 253-279.

[2] Bauckhage, C., Thurau, C., and Sagerer, G. (2003, September). Learning human-like opponent behavior for interactive computer games. In Joint Pattern Recognition Symposium (pp. 148-155). Springer, Berlin, Heidelberg.

[3] Bewley, T., Lawry, J., and Richards, A. (2020). Modelling agent policies with interpretable imitation learning. arXiv preprint arXiv:2006.11309.

[4] Q. Gemine, F. Safadi, R. Fonteneau, and D. Ernst. Imitative learning for real-time strategy games. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 424–429. IEEE, 2012.

[5] Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. (2014). Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. Advances in neural information processing systems, 27, 3338-3346.

[6] Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. ACM Computing Surveys (CSUR), 50(2), 1-35.

[7] Mnih, V., et al (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[8] Rubio-Manzano, C., et al (2019). Fuzzy linguistic descriptions for execution trace comprehension and their application in an introductory course in artificial intelligence. Journal of Intelligent & Fuzzy Systems, 37(6), 8397-8415.

[9] Rubio-Manzano, C., et al (2018, June). Human Players versus Computer Games Bots: A Turing test based on Linguistic Description of Complex Phenomena and Restricted Equivalence Functions. In International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (pp. 27-39). Springer, Cham.

[10] Rubio-Manzano, C., & Pereira-FariÃśa, M. (2017, July). Declarative computational perceptions networks for automatically generating excerpts in computer games by using Bousi Prolog. In 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (pp. 1-6). IEEE.

[11] Rubio-Manzano, C., & Trivino, G. (2016). Improving player experience in Computer Games by using players' behavior analysis and linguistic descriptions. International Journal of Human-Computer Studies, 95, 27-38.

[12] Trivino, G., & Sobrino, A. (2009). Human Perceptions versus Computational Perceptions in Computational Theory of Perceptions. In IFSA/EUSFLAT Conf. (pp. 327-332).

[13] Trivino, G., & Sugeno, M. (2013). Towards linguistic descriptions of phenomena. International Journal of Approximate Reasoning, 54(1), 22-34.

[14] Thurau, C., Bauckhage, C., & Sagerer, G. (2004, November). Imitation learning at all levels of game-ai. In Proceedings of the international conference on computer games, artificial intelligence, design and education (Vol. 5).

[15] Zadeh, L. A. (2001). A new direction in ai: Toward a computational theory of perceptions. AI magazine, 22(1), 73-73.

[16] Zhou, Y., and Li, W. (2020, August). Discovering of Game AIsâĂŹ Characters Using a Neural Network based AI Imitator for AI Clustering. In 2020 IEEE Conference on Games (CoG) (pp. 198-205). IEEE.