Learning language skills based on implicit, explicit and active learning

Luís Miguel Santos Henriques

Thesis to obtain the Master of Science Degree in

Engineering Systems and Computer Engineering

Supervisors: Prof. Francisco António Chaves Saraiva de Melo
Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Francisco António Chaves Saraiva de Melo
Members of the Committee: Prof. David Manuel Martins de Matos

October 2018

Acknowledgments

I would first like to thank my thesis advisors Professor Luísa Coheur and Professor Francisco Melo. Both Professors were always available whenever I ran into a trouble spot or had a question about my research or writing. They consistently allowed this paper to be my own work, but steered me in the right direction whenever they thought I needed it.

I would also like to thank the users who were involved in the tests made for this research project. Without their passionate participation and input, the validation of my work could not have been successfully conducted.

Finally, I must express my very profound gratitude to my parents and to all my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

To each and every one involved, thank you for everything!

Abstract

In this work, we present a system that explores the idea of using the feedback provided by the user to help in the process of learning. We focus on the task of learning a language skill, using the processes of implicit, explicit, and active learning to accomplish this task. As proof of concept, we use the SHRDLURN game, where the system learns to map the user language (initially the system has no knowledge of this language) into actions related to the game. The user feedback consists of an utterance and either an action or a property of a game element, being this feedback used to update the system knowledge. In our work, we extend the previous implementation of the SHRDLURN game, which is based on the process of implicit learning. In this thesis, we add to this game the possibility of having the processes of explicit and active learning. After testing the system with users, we conclude that our approach brings an improvement to the system used as baseline. Results indicate that we can take advantage of bringing together the three processes of learning used in our work.

Keywords

Natural language processing; Reinforcement learning; User feedback; Language skills; Implicit learning; Explicit learning; Active learning.

Resumo

Neste trabalho, apresentamos um sistema que explora a ideia de usar o feedback fornecido pelo utilizador para ajudar no processo de aprendizagem. Focamo-nos na tarefa de aprender uma característica da língua, usando os processos de aprendizagem implícita, explícita e ativa, para realizar essa tarefa. Como prova de conceito, usamos o jogo SHRDLURN, onde o sistema aprende a mapear a linguagem do utilizador (inicialmente o sistema não tem conhecimento desta língua) em ações relacionadas com o jogo. O feedback do utilizador consiste em uma frase e uma ação ou uma propriedade de um elemento do jogo, sendo este feedback usado para atualizar o conhecimento do sistema. No nosso trabalho, estendemos a implementação anterior do jogo SHRDLURN, que é baseado no processo de aprendizagem implícita. Nesta tese, adicionámos a este jogo a possibilidade de ter os processos de aprendizagem explícita e ativa. Depois de testar o sistema com os utilizadores, concluímos que a nossa abordagem traz uma melhoria ao sistema usado como baseline. Os resultados indicam que podemos tirar vantagem em reunir os três processos de aprendizagem usados no nosso trabalho.

Palavras Chave

Processamento de língua natural; Aprendizagem por reforço; Feedback de utilizador; Características da língua; Aprendizagem implícita; Aprendizagem explícita; Aprendizagem ativa.

Contents

List of Figures

List of Tables

Acronyms

QA	Question Answering
NLP	Natural Language Processing
RNN	Recurrent Neural Networks
QBC	Query-by-Committee
EGL	Expected Gradient Length
CPS	Changes per State

1

Introduction

Contents

We can think as Natural Language Processing (NLP) as an advance in the field of interactions between computers and humans, since it allows computers to interpret text written in human language (among others) and make it analyzable in order to find relevant information.

Systems that try to understand human language, have been increasingly present in our lives (from on-line social bots to the systems present in household robots), so it is important to allow them to learn from the interaction with users.

Nowadays, one of the common ways to train intelligent systems is using reinforcement learning. This type of machine learning allows systems to improve their action policies, so the actions taken by the system are those that maximize its performance (getting the best reward possible).

One possible source of learning is the user feedback after her/his interaction with the system. There are several works that already use the idea of having a user as helper of the reinforcement learning process, where the user communicates with the system through natural language [Buck et al., 2017], [Ling and Fidler, 2017]. Nevertheless, in the current state of the art, there are few works that try to learn a skill related with the natural language. Being, to the best of our knowledge, the only works developed by [Wang et al., 2016] where the system learns a language game while playing a game with the user, and the robot system named George developed by [Skočaj et al., 2016] where the system learns characteristics of the objects in its environment while has dialogues with a human tutor.

In this work, we intend to develop a system that uses the feedback given by the user to learn a language skill. In the continuation of this section, we will specify the definitions of the different types of learning used in our system, we will also define the objectives and contributions of our work, then, we are going to talk about the proof of concept explored in the implementation of the system, and finally, we will make a overview of the rest of document.

1.1 Different types of learning

In our work, the process of learning is divided into three parts: *implicit learning*, *explicit learning*, and *active learning*. Based in psychology literature ([Anderson, 2007], [Bruneau, 2014]), implicit and explicit learning can be defined as:

- Implicit learning: The system learns new information through exposure. This type of learning consists of a passive process and does not imply conscious mechanisms. For example, a system that is learning verbs receives several regular verbs in the past form and realizes that all end with "-ed". That is, even if it has not been explicit told to the system "all regular verbs in the past simple end with -ed", it learned this linguistic characteristic. Another example would be a system that

is learning a language game. After the system receiving several utterances with the words “bloc rouge”, it realizes that utterance is mapped into the concept of a red block, even though the system has no knowledge of the language spoken by the user.

- Explicit learning: The system learns new information through explicit examples/information. This type of learning consists of an active process and implies conscious mechanisms. The information received is added to the knowledge base in a direct way. For example, an English system that is learning numbers in Portuguese, receives the information that the word “dez” is translated into “ten”. Another example would be a system that is learning a language game, and the user tells the system that when s/he inputs the utterance “bloc rouge” s/he is referring to the red block.

During the process of learning, it is normal for the system to have gaps in its knowledge. In order to try to fill these gaps, the system is able to take initiative and ask questions to the user. These questions should be related to the information that the system is less uncertain about, so the information gained is maximized. This behavior is usually called *active learning*, and following [Settles, 2010], we can define this process as:

- Active learning: The system decides which data is more valuable to learn from and asks information about that data to the user. The system also decides when it needs additional information from the user, in order to only trigger this process when it is really needed. For example, an English system that is learning numbers in Portuguese, receives the information that the word “vinte e dois” is translated into “twenty-two”, then, the system can ask what “twenty” means, given that, at the beginning, he does not know which part of the sentence is translated into what (“twenty” can be translated into “vinte”, “e”, or “dois”). Another example would be a system that is learning a language game, and it is given the information that “supprimer le bloc rouge” is mapped to the action of “delete the red block”, similarly to the last example, the system can ask what “supprimer” means.

1.2 Objectives and contributions

We intend to contribute to the current machine learning and NLP literatures with a system that not only aims to learn a target language skill using implicit, explicit, and active learning processes, but also incorporates the user feedback in these processes, taking as proof-of-concept the idea of language games.

In other words, the central problem driving our work can be formulated as: How can a system use implicit, explicit, and active learning processes in order to learn a language skill from the interaction with a user, taking advantage of the feedback received?

Given the research problem identified above, the main goals of our work are:

- Developing a system that can use the feedback provided by a user in order to learn a language skill;
- Construct the system learning component, which is composed of both implicit and explicit learning processes;
- Using the active learning process to make the system able to detect gaps in its knowledge and ask questions, to the user, in order to receive representative feedback that will fill those knowledge gaps.

Now that we have defined the different types of learning relevant to our work and identified our research problem, we are going to describe the proof of concept used in the implementation of the system.

1.3 Proof of Concept: SHRDLURN game

As a proof of concept, we explore the idea of language games, where the user and the computer do not initially speak the same language but both need to collaborate in order to achieve a goal. More specifically we use the same proof of concept as in [Wang et al., 2016], where the authors presented a language game named SHRDLURN game.

The SHRDLURN game is composed of several iterations where the user has to interact with the computer, by giving him natural language instructions, in order to achieve a goal. This game is based on a world of blocks, where there are a “start” and a “goal” states, being each one represented by a set of blocks.

In each iteration of the game, both player and computer are presented with a starting state, but only the human sees the goal state. The human transmits an utterance to the computer, with the intention of telling which action(s) the computer must perform, in order to transform the starting state into the goal state. Then, the computer constructs a ranked list of candidate actions and returns, to the human, a list of candidate states, which are the result of applying the various candidate actions to the current state. The human then chooses one of the candidates from the list (we say the computer is correct if the human chooses the first candidate in the list). The actual state is then updated, being the iteration completed when the current state is equal to the goal state.

1.4 Document organization

This document is organized as follows: Chapter 1 gives context about the thesis motivations and goals, the problem domain where our thesis is developed, and details the different types of learning used; Chapter 2 gave an overview of the state of the art, by exposing several works that use the different types of learning that we are interested in; Chapter 3 presents the architectural decisions made to our system and their implementation; Chapter 4 presents and discusses the results obtained after making user tests to the system; Chapter 5 concludes the document with an overview of what we made and which conclusions were drawn.

2

Related work

Contents

There are several works that explore the idea of using NLP to process information received from their users and learn from this information. An example is the parser used in [Scheutz et al., 2017] that produces both syntactic structure and semantic interpretation, by labeling each word in the utterance input by the user and then mapping them into parser rules. These parser rules are used to update the state of the parser, which is represented by a binary tree that, when the parser receives an unrecognized word, is updated with a new branch/leaf.

Still in the parsers domain, SEMPRE [Liang, 2015] is a toolkit developed for training semantic parsers. The toolkit uses intermediate logical forms in order to map natural language utterances (usually questions) into answers (called "denotations" by the authors). An example is when the user utters "Which college did Bill Gates go to?" this toolkit transforms the utterance into "(and (Type University) (Education BillGates))", which can be queried into a database.

Another example of systems that are using NLP to process information received from their users is the system used in [Song and Kautz, 2012], where a testbed for learning by demonstration is constructed using a combination of spoken language and sensor data. The authors extracted high-level features (such as user/object location and relative positions) through depth cameras. These features were extracted by the authors from a set of several examples, one of them consisted of 7 different individuals demonstrating how to make tea. Lastly, speech is recognized and processed using a parser, where the processed information is then used to label the previously mentioned examples. The final set of information can further be used by systems that wish to learn from activities that use vision and language in an instructive setting.

All the works presented so far do not explore the idea of including feedback in the learning process. This idea is explored in several works, for example, in the domain of Question Answering (QA) systems, we have for example systems like JUST.ASK [Curto et al., 2014], that learns how to answer questions based on previous successful interactions. When the user inputs a natural language question, the system processes this question, then based both on information found on the Internet, related with the question, and on the feedback given by the user in previous related questions, constructs the answer to be output to the user.

In the continuation of this section, we describe additional works in which, not only the interaction between the user and the system plays a central role in the learning process, but also use reinforcement learning as the way for their systems to learn from these interactions. We break these works into three broad categories: those that use explicit learning, those that use implicit learning, and those that use active learning. We start by exploring works that either use explicit or implicit learning, then we explore a merging between the two, and finally, we explore works that use active learning.

2.1 Explicit Learning

Most of the systems that apply explicit learning are related with the Learning from Demonstration technique, which consists of policy learning through examples/demonstrations presented by a teacher. This technique is usually applied in the Robotics field. For this reason, in some of the forthcoming works, we will narrow down the discussion to the essentials for our work.

A possible learning procedure to be applied when we are talking about explicit learning is the *show-and-tell* procedure. In [Alomari et al., 2017] the authors present a framework where the learning process is accomplished using such procedure. The framework presented receives video clips of a manually controlled robot arm and natural language commands describing the actions, having the objective of learning the grounding in visual semantics and the grammar of the natural language commands.

First, each video and the corresponding description are segmented, such each segment contains a single action/verb in it. Then, using a parser, the system maps each description segment (n-grams) to its visual representation segment – in the example "pick up the apple" and the segment of the video where the robot picks up an apple, "pick up" is recognized as the action, "apple" as the object being picked up, and "the" has no association).

The process of learning has two parts: 1. the system trains the parser with the different mappings between the n-grams (for example, 2-gram "pick up" and 1-gram "apple") and the visual representations – in the above example, if the system was not sure about the meaning of "apple", the validation of the word "apple" would be made in comparison with other examples that contain that word; 2. when the system maps a particular description segment and its visual representation segment, this mapping is added to the knowledge base.

The previous work presents a explicit learning process, since the system receives video-clips and, for each one of them, the natural language descriptions. Based on this conclusion, we will use the idea of *show-and-tell* on the process of explicit learning necessary for our work.

Another good example of systems that traditionally use explicit learning in their learning process, are the QA systems. Usually, QA systems are focused on learning from question-answer pairs ([Burger et al., 2017]). However, a different approach is presented in [Sachan and Xing, 2017], where the authors explore the idea of using natural language demonstrations for questions provided by a teacher in order to improve QA systems. As a specific domain, the authors chose the task of learning to solve geometry problems, where the solutions are taken from textbooks.

In the previously mentioned work, the authors mapped geometry questions into logical expressions (e.g., "the radius of the circle with center O is 4 cm" is mapped to $radius(O, 4)$) and modeled the different parts of the demonstration using a deduction model, which treats each part as a state and the way those

parts interact as transitions between states. The identified states are then used to train the deductive solver. In this process of training, the deductive solver learns to score the sequence of theorems (or part of them) that can lead to the solution of the problem.

The process of learning present in the previous work is an explicit learning process, where the system updates its parameters based on the explicit mapping between a geometry problem and its solution.

Most of the works developed in the area of problem-solving in machines are based in the approach of teaching a solution to a problem, being the system programmed with the game state description, goals, and possible action. A different approach is explored in [Kirk and Laird, 2014], where the authors aim to develop a system that learns the problem specifications through instructions given by humans, in the form of constrained natural language.

The authors developed an agent named Rosie, which can play simple spatial games and puzzles such as Tower of Hanoi, Tic-Tac-Toe, among others. The idea is to represent such games as a set of blocks and locations, where the agent performs actions in order to complete the chosen game. Rosie is incorporated in a table-top robot, composed by a robot arm (which moves the blocks) and a Kinect sensor (which detects the objects and their different properties – color, shape, location, among others).

The authors claim that the developed agent needs much less information than other common approaches. The authors also explore the idea that knowledge is not only added to the knowledge base but also passed through the different games that the agent is playing.

The agent interacts with the human before starting the game. In this interaction the agent finds which game it is about to play. If the agent does not know the game yet, it asks for the number of players and basic concepts of the game such as possible actions, the verb and parameters associated with the actions. For a better understanding of this interaction, let us consider the example represented in Table 2.1 (example taken from the original article), where the agent is about to play Tic-Tac-Toe, but since the agent does not know that game yet, it needs to ask the basic concepts of the game (those concepts are asked in the “Rosie” side of the table).

The agent is implemented in Soar [Laird, 2012], which is a general cognitive architecture that integrates reasoning, planning, learning from experience, among others. The agent passes the information gathered from the interaction with the human to the Soar semantic memory, which specifies all the information required (actions, goals, and failure conditions) to determine a possible action and find a solution. Here, we can see a process of explicit learning, where the agent gradually adds to its knowledge base object descriptions, verb knowledge for the current game, and the required spatial relationships.

The previous work assumes methods that are integrated with Soar, which is a requirement that is not present in our work. Nevertheless, the idea of receiving information about the basic concepts in the game, to acquire a better knowledge of the domain, is used in our work.

Rosie	Mentor
What is the name of the game?	The game is tic-tac-toe.
I do not know that game, how many players are there?	Two.
Please start by teaching me the name of a legal action in the game.	Place.
What are the verb and parameter arguments associated with this action?	move 1 to 2
Ok I know that verb, what kind of object is the first parameter? (block, location, or either).	Block.
Please list all constraints for this parameter (such as 'it is red' or 'it is on [parameter] 2') and then finished.	It is red.
It is not on a location.	Finished
What is the type of the next parameter? (or finished if no more)	Location.
What are the constraints?	It is not under a block.

Table 2.1: Dialog between Rosie and Mentor before starting a Tic-Tac-Toe game (taken from [Kirk and Laird, 2014]).

CogX[1] is a scientific project founded by the European Commission. The objective of this project is to develop cognitive systems, like robots, that have the ability to work in open environments and dealing with new and unpredictable information. The project takes advantage of two key elements which have taken a significant step forward in recent years. First, the available learning methods have been increasing significantly in terms of range and power, like human augmented mapping and computer vision. Second, the progress made in the field of building robotic systems that are capable of using multiple nodes of sensing and acting.

The main goal of the CogX project is to create a framework that connects self-understanding and self-extension with specific robot learning algorithms, and make that framework "convincingly instantiated and studied in robot systems". One of the systems developed in this project is the robotic system George ([Skočaj et al., 2016]). George learns interactively while having dialogues with a human tutor. The robot system, autonomously, acquires new categorical knowledge over the time. This knowledge is related to objects in the system environment and is passed to the system while it has natural language dialogues with its tutor. The system uses this information, jointly with visual information, to continually update its knowledge.

George needs to understand *what* the tutor is trying to tell with the information provided and *why* s/he is providing that particular information. To accomplish this understanding, the system generates and verifies (the verification part is discussed in more detail in Section 2.4) tutor behavior hypotheses, in concerns of communicative intentions, by using continual abduction, which is a method that allows inferring hypotheses of an event occurring based on another event.

[1] http://cogx.eu

The system performs *reference resolution*[2] to learn from a situated dialogue, by relating the tutor information with its own perceptions. Reference resolution tell us which words or phrases are assigned to other words or phrases, and how are they related.

The CogX system represents a possible approach to the explicit learning process required in our work, since the system uses explicit learning by forming beliefs (of the domain in which it is inserted) based on the linguistic information and visual perceptions received. Then, using those beliefs, updates the internal representations of objects.

In this section, we have been reviewing works that use explicit learning. All of those works have made us have a better vision of how to apply this learning process to our work. However, the show-and-tell procedure and the idea of receiving information about basic concepts, are directly used in our work.

2.2 Implicit Learning

Previous work developed in [Buck et al., 2017], aims to build an agent that serves as an intermediary in the interaction of the user and a black box QA system. For a better understanding, let us consider Figure 2.1 (taken from the original article), where the interaction between the agent with both the user and the QA system is represented. The purpose of this agent is to reformulate the question input by the user so the answer given by the QA system is the best possible. The authors propose a method that resembles active learning[3], called Active Question Answering. The main difference between the Active Question Answering and standard active learning "is that it searches in the space of natural language questions and selects the question that yields the most relevant response".

The key component of the agent architecture is the Sequence-to-Sequence model, which is intended to reformulate the question input by the user into one or more alternative questions that will have a better answer from the QA system. This model is trained using reinforcement learning, where the reward is based on the answers given by the QA system to the reformulated questions sent by the system. We can see this process as an implicit learning process, where for each question input by the user, the agent reformulates the question (based on its policy) in order to get the higher reward possible, although this type of feedback only tells to the agent if the reformulation was good/bad and not why.

In the previous work, to have an effective Active Question Answering model, the authors needed to use multilingual corpus to pre-train the system. Given the large amount of data that usually is required to train a Sequence-to-Sequence model, we decided to not use this type of model in our work. However, the previous work, represents a possible approach to the implicit learning process present in our work.

[2] https://wiki.opencog.org/w/Reference_resolution

[3] In active learning the algorithm chooses what particular data should be sent to the environment to be labeled, with the propose of collecting the most valuable information.

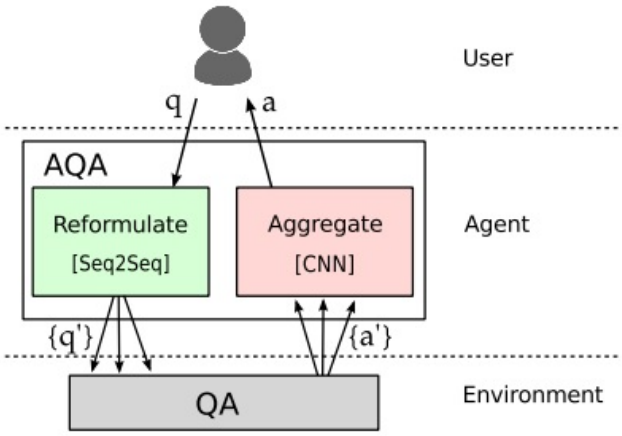

Figure 2.1: The agent-environment setup for Active Question Answering (taken from [Buck et al., 2017]).

End-to-End learning is usually related to applying Gradient-based learning to the training of learning system as a whole, instead of focusing in each component individually. In [Zhao and Eskénazi, 2016], the authors propose an End-to-End framework oriented to dialog systems, which aims to remove the problem of determining the source of the errors in typical Spoken Dialog System pipelines (found the error source on these pipelines, requires error analysis in each module). In this work, an agent interacts with the environment – user and database – by sending verbal actions to the user (receiving natural language responses and rewards) and by querying the database (receiving observations and rewards). For a better understanding, let us consider Figure 2.2, which represents the dialog framework where the agent interacts with the environment.

The authors chose to follow [Henderson et al., 2014], which learns a sequential classifier (based on a Recurrent Neural Network) that maps speech recognition results to the dialog state without using an explicit semantic decoder. They also decide to unify the dialog state tracking with the dialog policy, treating them as actions, which are used by the reinforcement learning agent. Specifically, an optimal policy is learned in order to either generating a verbal response or modifying the current estimated dialog state based on new observations.

At each turn, the agent applies an action based on its policy and then receives the observations from either the user (in form of natural language responses) or the database. Then, using a Long Short Term Memory network, a state dialog is generated so it can be used to update the current policy.

We can see the process of learning present in this system as an implicit learning process, where for each action made by the agent, it receives feedback from the environment and then uses it to update the policy. This means even the system receives feedback from the environment, this feedback does not tell exactly what the agent did wrong/right.

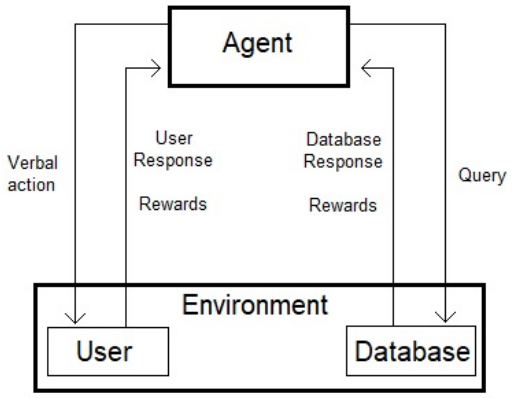

Figure 2.2: End-to-end task-oriented dialog framework (adapted from [Zhao and Eskénazi, 2016]).

The process of learning present in this work is related to ours, and even End-to-End systems are a possible approach to our problem, we prefer to treat the system as separated modules in our approach.

When we are trying to improve a dialogue policy via reinforcement learning an essential factor to take into account is the accuracy of the reward function. A possible type of reward to use in the process of reinforcement learning is the feedback provided by the user after interacting with the system. However, explicit user feedback in real-world applications is often costly to collect and affects the user experience.

A possible solution to mitigate the difficulty and disadvantages of collect user feedback is presented in [Su et al., 2016], where the authors explore the idea of first analyze the information provided by the user and then, only if necessary, ask for user feedback. The authors propose an on-line learning framework where the user interacts with the system using natural language and, if the system decides so, the user is asked to hand over feedback about the correctness of the dialog. The system framework is composed by three main components (as we can see in Figure 2.3): 1. a module that contains the dialogue policy and sends/receives dialogues to the user; 2. a dialog embedding function, which, at the end of each dialogue, receives a set of turn-level features (e.g., dialog duration) and transforms it into a fixed-dimension representation; 3. a reward model, which receives the dialog representation output from the previous component and, using a Gaussian Process, estimate the task success along with a measure of the estimate uncertainty. Based on this uncertainty, this component decides if it is necessary to ask for user feedback. If user feedback was requested, a reinforcement signal based on this feedback is used to train the dialogue policy, otherwise the dialogue policy is updated directly with the predictive success rating.

We can see the process of learning present in this system as implicit learning process, where the dialogue policy is trained with the information processed either from the feedback present in the extracted turn-level features or from the feedback given by the user feedback about the correctness of the dialogue.

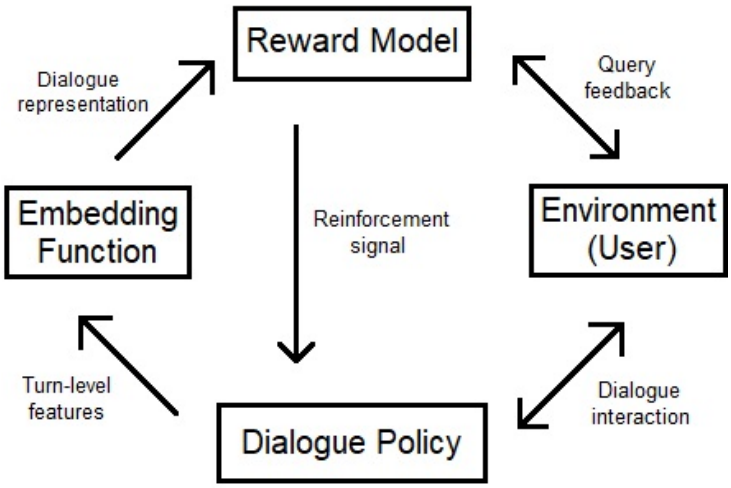

Figure 2.3: Schematic of the Active Reward system framework (adapted from [Su et al., 2016]).

The system presented applies the implicit learning process required in our work. But more importantly, this work motivated us not only for the problem of continually requesting user feedback, but also how to mitigate this problem. We will apply this acknowledgment in the active learning process, which will be reviewed later in this chapter (Section 2.4).

Another possible use of the user feedback in the process of learning is explored in [Lampinen, 2017], where the authors enter in the area of machine learning systems that try to play games with humans. The authors note that to train those machine learning systems, requires a large amount of data in comparison with the amount needed by humans (humans need much less data). The authors argue that user feedback can be related with the least amount of data necessary by humans, since humans usually receive a considerable amount of natural language instructions when they are trying to learn a task. To explore this idea, the author used Q-learning[4] to train a Neural Network, incorporating natural language instructions in this process, in order to help the network play the tic-tac-toe game.

In this work, the user interacts with the system by inputting questions that would influence the system decisions. Let us consider Figure 2.5 (taken from the original article) for a better understanding, which represents the interaction between the different components of the system during a Tic-Tac-Toe game. In this example, the user inputs the question "What's in the middle row?", the system considers the actual state of the game (lower board), answers the question ("Opponent has two, unblocked") and take an action (upper board).

The system architecture consists of the following components: 1. a visual parser, which is composed by a two layer Neural Network that receives the board state and outputs the system internal repre-

[4]https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc

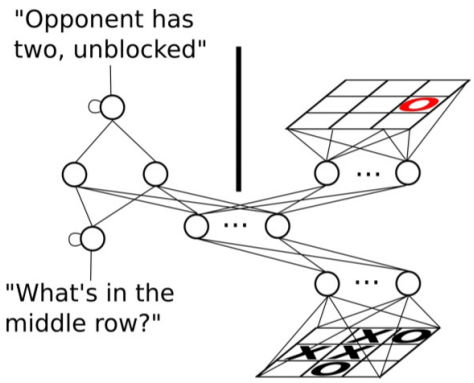

Figure 2.4: Example of an action in the tic-tac-toe game (taken from [Lampinen, 2017]).

sentation of the board state; 2. a question answerer, that receives a question, encodes it (using word embeddings - which maps words/a text into a vector of real numbers) and then use this information jointly with the output of visual parser to produce the answer to the question; 3. a Q-approximator, which is composed by a two layer Neural Network that receives the output of the visual parser and produces the Q-values for the 9 possible plays on the board. The Q-approximator component share the two layers that make up the Neural Network with the question answerer component, which causes the Q-approximator to take into account the information output by the question answerer.

The process of learning present in this work is an implicit learning process, since the user is inputting questions that influence the actions made by the system, not saying exactly which move would be the correct one. Although this learning process is related to our work, the authors are training a Neural Network to solve the problem identified. This type of network requires a considerable amount of training, which was a requirement avoided in our work.

A possible scenario to make a system to learn a language skill is using a language game. This concept was introduced by Wittgenstein [Wittgenstein, 1953], where the author argued that speaking a language is part of an activity. The work developed in [Wang et al., 2016] explores the idea of language games in a learning setting. The authors developed a system that interacts with the user in order to learn the instructions given by her/him. In this work is presented a game named SHRDLURN (Figure 2.4), a game developed in the world of blocks where the user is presented with an initial state and a final state (each state consists in a set of blocks, where each block has a color). Since the computer only has awareness of the initial state, the goal of the user is to give natural language instructions to the computer about what transformations it should do (e.g., remove a certain block) in order to transform the initial state into the final state.

When the user inputs an utterance, a semantic parser is used to process the information and mapping it into a ranked list of actions (based on the current policy), more specifically the system has a predefined

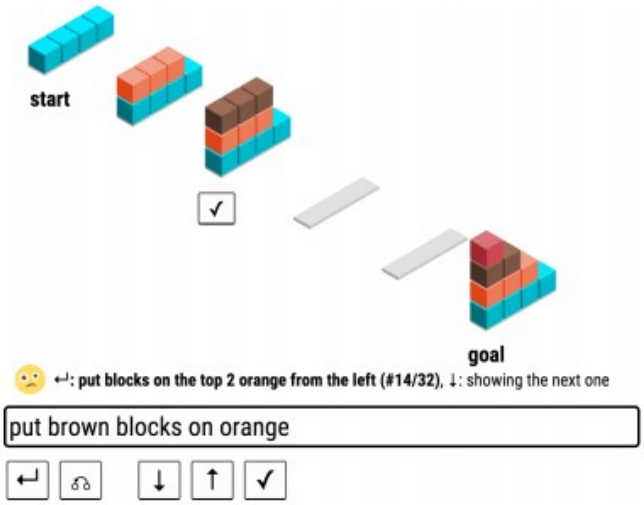

Figure 2.5: Interface of the SHRDLURN game (taken from [Wang et al., 2016]).

grammar which maps the utterances into the system language. Applying these actions to the current state gives the possible next states, which are presented (in ranked way) to the user, so s/he can choose the most correct.

The process of mapping the user utterance into the multiple rules of the predefined grammar is the core of the learning process in the system. The model used has in consideration a vector of parameters (which works as a policy), which is updated with the feedback provided by the user (when selects the most correct state for the utterance that s/he input).

Lastly, to improve the process of learning, the authors joined the concept of mutual exclusivity to the system. The concept is usually related with the human learning, where the human avoids assigning a second label to an already known object. As an example for SHRDLURN in particular, if the computer knows the word used by the user to refer the blue block, after receiving an utterance with an unknown word, the computer should first try to associate that word with other color than blue. In order to introduce this concept in the system, the authors used probabilistic models of pragmatics [Golland et al., 2010] to create an user strategy, which models the user language based on the context. This strategy is used to improve the ranked list of states presented to the user after s/he input an utterance.

The presented system uses a process of implicit learning, where the system tries to learn the language used by the user, improving its policy based on the user feedback. This feedback only tells the system which is the correct action to the utterance input, not telling which part of the utterance correspond to which part of the action.

The system presented addresses our goals in the part of implicit learning process, needing only the addition of the explicit and active learning processes required to fulfill our objectives.

In this section, we have been reviewing works that use implicit learning processes. We found a work that matches with our objectives in the part of implicit learning. However, this work does not contain the process of explicit learning required in our objectives. All the remaining works do not fully fit in our problem, nevertheless they have given us a broader view of the problem we are addressing and, possible, different ways to approach our work.

2.3 Merging Explicit and Implicit Learning

The problem of image captioning is a good example where new users can easily provide good feedback, since the user only has to evaluate the correctness of the caption. This problem is approached in [Ling and Fidler, 2017], where the authors present a system where a non-expert user can give feedback to a reinforcement learning agent using natural language. The authors focus on descriptive sentences since, as they argue, this type of sentences provide much stronger learning information than a numeric reward. In particular, the best type of feedback to this system is when the user describes only one mistake (e.g. a single wrong word).

In order to generate the image caption, the authors use two Recurrent Neural Networks (RNN). The first RNN receives a set of context features extracted from the image (e.g., there is a beach in the image, there is a woman in the image) and provides topics. Each topic is used to contextualize the second RNN, that generates sequences (one for each topic) of words that describe the image (e.g. of three sequences of words, (a woman) (is sitting) (on the beach)).

The image and the respective captioning is present to the user, then the user gives two types of feedback: 1. the user writes in natural language what is wrong in the caption (e.g., "there is a man on the beach, not a woman"); 2: the user rewrites the caption in the correct way (e.g., (a man) (is sitting) (on the beach)). The first type of feedback given by the user is divided into sentences and, using a Neural Network, is defined the correctness of the each sentence (using the last example: (a man) – correct, (a woman) – incorrect, (on the beach) – not relevant).

In conclusion, the presented system uses both implicit and explicit learning based on the user feedback. The implicit learning comes from the first type of feedback provided by the user, where the user utters what is wrong in the caption, being the interpretation of this feedback left for the system. On the other hand, the explicit learning comes from the second type of feedback provided by the user, where the user corrects directly the caption.

Although the system contains both types of learning processes necessary for our work, the authors built a solution based on multiple Neural Networks, which require a considerable amount of training and is requirement that we avoided in our work.

2.4 Active Learning

The key idea behind *active learning* is that, if the system is able to choose from which data it wants to learn (allow the "curiosity" of the system), the system will produce better results using fewer training labels ([Settles, 2010]). In other words, we are letting the system ask the user for labels to the information that the system considers more necessary, at the moment. For a better understanding let us consider the learning cycle presented in Figure 2.6, which represents the interaction between the system and the user, during the process of active learning.

This type of learning is mostly present in works where unlabeled data is easily acquired, but the labels for that data are expensive/time-consuming to acquire. Examples of high-cost label information are: *speech recognition* - label words/phrases taken from audio; *information extraction* - label entities or relations of interest in a text, like: organizations, people names, and in which organizations these people work. In problems like these, active learning attempts to choose the most relevant information to be labeled, instead of asking labels for all the information, thus, making the work of the user less tedious, while trying to get better results (considering the same amount of labeled examples obtained).

An example of a work that uses active learning is the robotic system named George (already introduced in Section 2.1) developed in [Skočaj et al., 2016]. As we mentioned, the system receives linguistic information from a tutor and integrates that information with visual perceptions. During this process, not only the tutor gives information to the system, but also the system itself takes the initiative to ask questions. To decide when is the proper time ask questions, the system needs to detect gaps in its knowledge and estimate what information would be more relevant to fill those gaps. Also, the active

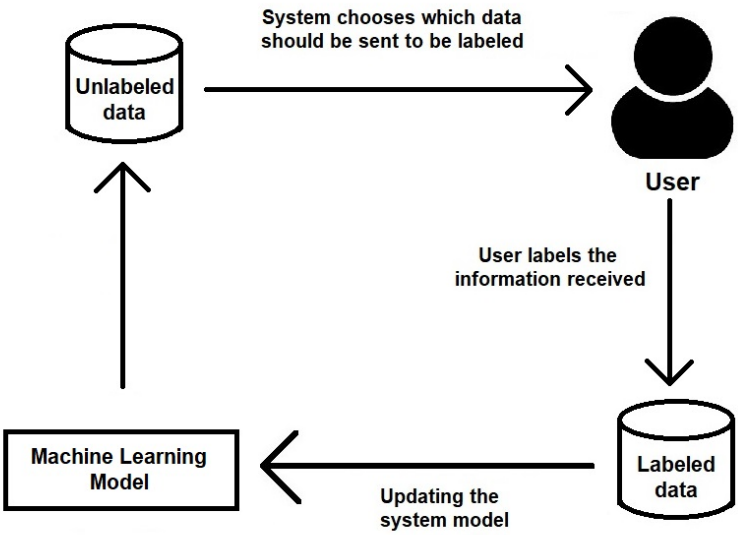

Figure 2.6: Learning cycle during active learning process.

learning component is what helps the system to verify tutor behavior hypothesis. To do this, the system uses abduction to generate partial hypotheses about the behavior of the tutor. Then, also using the abduction, represents the knowledge gaps as partial abductive proofs. Lastly, the system needs to verify or deny (by asking to the tutor) the knowledge gaps, in order to transform these partial abductive profs into real profs.

In our work, the active learning component plays a similar role to what happen in [Skočaj et al., 2016]. Since the user enters multiple utterances during the game, we use active learning to detect gaps in the system knowledge about those utterances. Then, the system asks for user feedback, that is used to fill the previously identified gaps. To perform the previous actions, the system needs to identify the information that is more uncertain about and when is the proper time to request user feedback about that information.

In the continuation of this chapter, we will follow the structure of survey presented by [Settles, 2010], dividing the literature review into *scenarios* and *query strategy frameworks*, then, we will present the approach chosen in the implementation of our work. In the following subsections, we will use several times the term "instance", which refers to a particular example of the information used by the system, that can have a label associated (for example, a corpus of English to Portuguese data could have the instance "dez" associated with the label "ten").

2.4.1 Scenarios

There are several scenarios where the system, as a learner, may be capable of asking questions to the user. The three main scenarios considered in the active learning literature are:

- Query synthesis:
  - The system generates examples from scratch to be labeled by the user. The examples generated are those that the system considers more informative about the gaps in its knowledge, being the informativeness of each example, usually, measured using query strategies (section 2.4.2). In this type of scenario, the system can select any unlabeled instance from the input space, which can be a problem when the oracle is a human annotator, since the system can generate examples that have no meaning to the human.
  - An example of the problem of generate examples that have no meaning to the human is encountered in the work presented by [Baum and Lang, 1992]. The authors used query synthesis, in collaboration with human annotators, to train a neural network to classify handwritten characters. The authors realized that many of the images generated by the system contained only symbols that had no semantic meaning for the users.

- We can think of the problem exposed by the previous authors, in the domain chosen for our work. If the system requests user feedback about information that has no meaning to the user, s/he will not be able to provide relevant information to be used by the system to fill the gaps in its knowledge. Given this problem, we chose not to use the query synthesis scenario in the implementation of the active learning process in our system.

- Selective Sampling:
  - The system gets, sequentially, unlabeled instances from the data source and, for each one, decides if it should be labeled by the user. The decision to whether asking for a label or not, to the user, can be done in several ways. One way to make this decision is by using query strategies (section 2.4.2). This type of scenario is useful in problems as the one presented by [Fujii et al., 1998], where the authors applied selective sampling in word sense disambiguation (for example, the word "charge" in the phrase "I gave him a charge", can either mean that I gave someone an electric charge or a charge related with a criminal law). Nevertheless, in this scenario is assumed that obtaining unlabeled instances has no cost, and such requirement might not always hold in real-world learning problems.
  - The selective sampling is a possible scenario to be applied in our work. However, we chose not to apply this scenario in our implementation, since we want to take into account the problem of continually requesting user feedback ([Su et al., 2016]), where the selective sampling is not a good scenario (the user is asked about the labels of several unlabeled instances).

- Pool-based Sampling:
  - From a pool of unlabeled instances, the system chooses which one is the most informative to be labeled by the user. Usually, the system uses a query strategy (section 2.4.2) to choose the most informative instance present in the pool.
  - Although the system has to evaluate the entire pool at each iteration (usually requires great computational power), this scenario is probably the most used in real-world learning problems, such as: *Speech recognition*, where the authors of [Tür et al., 2005] used active learning to select the utterances that are likely to be most informative for labeling, in order to reduce the number of human-labeled utterances needed to train a system intended to understand spoken language; *Text classification*, where the authors of [Hoi et al., 2006] used active learning to select a batch of text documents for labeling manually, instead of selecting a single document, as made in many works that applied active learning to automatic text categorization.
  - We chose the pool-based sampling to be the scenario in our implementation of the active learning component. The system decides which instance is the most informative and if is the proper time to request information, about that instance, to the user.

2.4.2 Query Strategy Frameworks

In the previous subsection, we saw three active learning scenarios that have in common the use of a strategy to decide the informativeness of unlabeled instances/examples created. In this subsection we present several ways to formulate such query strategies, by presenting the most used *query strategy frameworks* in the active learning literature:

- Uncertainty Sampling: Probably the simplest and most used framework, the uncertainty sampling is based on asking the user for the correct label of the most uncertain instance. The uncertainty of each label can, usually, be obtained using three different strategies:

  1. *Least confident*, where the system calculates the uncertainty of each instance for the most probable label, and then chooses the most uncertain one. This strategy has the disadvantage of only considering information about the most probable label;
  2. *Margin sampling*, in order to mitigate the disadvantage of the previous strategy, we can consider the margin sampling strategy, which uses the two most probable labels in the calculation of each instance. In problems with large sets of labels, by considering only the two most probable labels we are "throwing away" several information about the remaining label distribution;
  3. *Entropy*, to take into account all the possible labels we can use the entropy strategy. This strategy considers all labels in the computation of each instance and then, chooses the instance that has higher entropy (for example, if we have 4 possible labels, an instance that has a probability of 0.7 in one of the labels and 0.1 in each one of the others, will have a lower entropy than an instance that has 0.25 in each one of the labels - in other words, the more the information is concentrated, the lower the entropy is).

  Comparisons between the three strategies were made in a few papers (such as in [Schein and Ungar, 2007]) being the results mixed, which suggests that the best strategy usually depends on the application of active learning component.

- Query-by-Committee (QBC): This framework involves maintaining a committee of models, being each model trained using the current labeled set, but represents a different (competing) hypothesis. In each iteration, all the models vote on the label for each unlabeled instance. The unlabeled instance with more disagreement is selected as the more informative one. QBC aims to minimize the version space, which is the set of hypothesis that is in accordance with the labeled training set. The QBC selection algorithm has two requirements: 1.Construct a committee of models capable of representing different regions of the version space; 2.Have some measure of disagreement among committee members. To the second requirement there are two main approaches in the current literature:

1. *Vote entropy*, presented in the work of [Dagan and Engelson, 1995], can be interpreted as QBC generalization of entropy-based uncertainty sampling. Being C the number of committee members, y_i the set of all possible labels, and $V(y_i)$ the number of votes that a particular label y_i received, the vote entropy measure has the following formula:

$$x^*_{VE} = \arg\max_x -\sum_i \frac{V(y_i)}{C} log \frac{V(y_i)}{C} \tag{2.1}$$

2. *Kullback-Leibler divergence*, presented in the work of [McCallum and Nigam, 1998], selects the most informative query as the one that has the "largest average difference between the label distributions of any one committee member and the consensus".

- Expected Model Change: This strategy aims to select the instance that, if we knew its correct label, would lead to largest changes in the current model. One possible strategy used in this framework is the Expected Gradient Length (EGL), which was introduced in the work developed by [Settles et al., 2008] and is applied to learning problems that use gradient-based training. While using EGL, the system should select the instance x which, if we knew its correct label and we added it to the labeled data L, would produce the new training gradient with the largest magnitude (the largest the gradient magnitude, the more the model has changed). In order to select that instance, the system applies the following expression:

$$x^*_{EGL} = \arg\max_x \sum_i P_\theta(y_i|x)||\nabla l_\theta(L \cup \langle x, y_i \rangle)||, \tag{2.2}$$

  where y_i are the possible labels for the instance x, $l_\theta(L)$ the objective function l with respect to the model parameters θ, $\nabla l_\theta(L)$ as the gradient of this function, and $||A||$ as the Euclidean norm of A.

In conclusion, we decided that the approach that best fits our work is to use a pool-based scenario, where the system has several instances and it needs to decide which instance is the more informative one. To evaluate the informativeness of each instance, we are using the uncertainty sampling framework, where we decide to use the entropy strategy, so for each instance, the system can decide which one has the more uncertainty based on the current knowledge base. In other words, the system asks for a label to the instance that it has less certain at the moment, within its instances set.

3

Solution

Contents

The main objective of this dissertation is to build a framework that receives feedback from the user and learns a language skill from that feedback. As learning processes, we use explicit, implicit, and active learning, in order to make the system explore the different types of information that can be extracted from the user feedback.
With the development of our system, we aim to accomplish the following learning problems:

- In terms of *implicit learning*, the system receives an utterance, performs an action representative of the utterance received, and the user inputs feedback about the action performed. Finally, the system learns from the user feedback.
- In terms of *explicit learning*, the user inputs an utterance about a domain concept, and, explicitly tells to the system, which concept s/he is talking about. Then, the system learns from this feedback.
- In terms of *active learning*, the system detects gaps in its knowledge and estimate what information is more relevant to fill those gaps. When the system decides that is the proper time, the user is asked to give feedback about that information. Finally, the system learns from the user feedback.

As a proof-of-concept for our system, we decided to use the game developed in [Wang et al., 2016], named SHRDLURN. In SHRDLURN, the implicit learning process required for our work is already present, which means we will rely on this system to build our solution, adding the processes of explicit and active learning to it.

In the continuation of this chapter, we will present an overview of the SHRDLURN system, where we will talk about the general architecture and key concepts of the system, then, we will review the architectural components that make up the system, and, finally, we present the new version of this system (SHRDLURN+) that is an extension of the original system with our improvements (explicit and active learning).

3.1 SHRDLURN Overview

3.1.1 General Architecture

As mentioned before, the system receives an utterance from the user and calculates the most likely actions to that utterance. Then, it presents to the user the result of applying the different actions to a set of blocks, and, lastly, the user gives feedback to the system, by choosing the right set of blocks. This feedback is used by the system to update its knowledge base.

The SHRDLURN system has as core the *SEMPRE toolkit*[1]. In a high-level vision and focusing on

[1] https://github.com/percyliang/sempre/blob/master/TUTORIAL.md

the parts used by the SHRDLURN system, we can think of this toolkit as having two main components, the **Parser** and the **Learner**. In order to have an idea of the role played by those components in the system, let us consider Figure 3.1 (where the numbers represent the information flow order).

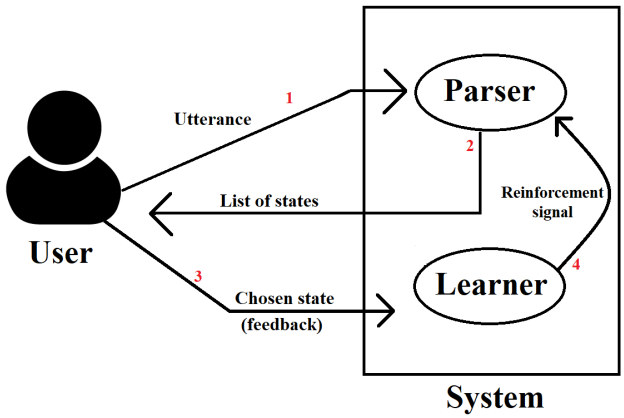

Figure 3.1: Representation of the interaction between the user and the system.

In the continuation of this section, we first describe a few fundamental concepts for our work, then we make a small review of both the *parser* and the *learner* system components. Finally, we will present the architectures implemented/developed for each one of the learning processes: *explicit*, *implicit*, and *active*.

3.1.2 Key Concepts

Grammar: is the set of rules that are recursively used to construct actions (following concept below). In the SHRDLURN system, the grammar is the following:

Rule	Semantics
Set	all()
Color	cyan | brown | red | orange
Color → Set	with(c)
Set → Set	not(s)
Set → Set	leftmost(s) | rightmost(s)
Set → Color → Act	add(s, c)
Set → Act	remove(s)

Figure 3.2: SHRDLURN grammar

In the previous table, the column "Rule" represents the possible relations between rules. For example, in the third line, "Color → Set" means that a Set will receive as argument a Color. Also in the fifth line, "Set → Color → Act" means that an Action will receive as arguments a Set and a Color. In the "Semantics" column we can see the possible values for each rule.

Actions: each one represents a change that will be applied to the current state of set of blocks, in order to obtain another set of blocks (which can be different or not). For a better understanding, let us consider the following example: the initial state has 3 blocks, “red blue blue”, if we make the action *remove(with(red))*, the next state would have 2 blocks, “blue blue”; An action can also result in the same state. For example, the initial state of the previous example, applying the action *remove(with(orange))*, would result in the same exact state as the initial one, since there are no orange blocks to be removed.

Features: each one represents a particular relation between part of an utterance and an action. In the SHRDLURN system, the authors chose to use a combination of tree-structures (for actions) and n-grams (for parts of utterances) as representation of the features. As example, let us consider that the user introduced the utterance “apagar azul”, Figure 3.3 illustrates the internal representation of part of the features generated for the action *remove(leftmost(nonEmpty))* (being each line a different feature – the full example can be seen in Figure A.1), where the components between parentheses are examples of possible actions (for example, in line three, we have the action *(removeTop (leftMost1 *))*) and the components between brackets are (part of) the utterances introduced by the user (for example, in line three, we have utterance “apagar”, which is part of the utterance “apagar azul”). Even though we can see a substantial mismatch between those actions and the ones in the previous table (referent to the system grammar), there is a direct relation between them. For example, “leftMost1 *” in Figure 3.3 correspond to “leftmost(s)” in Figure 3.2 and “removeTop *” in Figure 3.3 corresponds to “remove(s)” in Figure 3.2.

```
tree :: removeTop.1-leftMost1.1-getNonEmpty.0-[apagar, *, *]
subtree :: (removeTop (leftMost1 (getNonEmpty))) :: [apagar, *, *]
subtree :: (removeTop (leftMost1 *)) :: [apagar, *, *]
subtree :: (leftMost1 (getNonEmpty)) :: [apagar, *, *]
subtree :: (removeTop *) :: [apagar, *, *]
subtree :: (leftMost1 *) :: [apagar, *, *]
subtree :: (getNonEmpty) :: [apagar, *, *]
```

Figure 3.3: Example of features generated

Features weights: each feature has a weight, that represents how strong is the relation between the utterance and the action associated with that feature. For example, if we have the feature F1 with weight 3 that represents the relation between the utterance “apagar” and the action *remove(all())*, and the feature F2 with the weight 0.5 that represents the relation between the utterance “apagar” and the action *remove(leftmost(with(red)))*, when the system receives an utterance U that contains “apagar”, the system will prioritize the feature F1 over F2, while ranking the possible actions for U.

Parameters vector: consists of a vector that contains all features and the currently assigned weights. Each position of the vector contains a particular feature and its weight.

Actions scores: each action has a score related to a particular utterance. As an example, let us consider a particular action A that is present in multiple features. The score of A in relation to an utterance U is the sum of all weights of features that contain relations between A and U (or parts of U).

States: each state consists of a set of blocks, where that set of blocks has an internal representation in the form of $[[X_0, ..., X_m], ..., [X_n, ..., X_m]]$, where X represents a number between 0 and 3 (0 is **blue**, 1 is **brown**, 2 is **red**, and 3 is **orange**), n is the number of columns in that state, and m the number of lines in a specific column.

Let us consider some examples: [[0],[1]] represents a state where on the left there is a blue block, and on the right, there is a brown block. [[0, 2], [1], [3]] represents a state where on the left there is a blue block and on the top of this one a red block, on the middle there is a brown block, and on the right, there is an orange block.

Derivations: is the name given to the data structures used to group the following components: one action, the result of applying that action to the actual state, and the score for that action.

3.1.3 Architectural Components

3.1.3.A Parser

This component receives utterances (in this case, instructions related with the game, given by the user) and generates the possible actions, sorted by score. These **actions** are generated by accessing the system **grammar** and, for each rule, generate the possible combinations (limited to a depth from 1 to 8 – e.g. depth 1 = *(remove)*, depth 4 = *(remove (leftmost (with (blue))))*).

To sort those actions by score, the log-linear model, introduced by [Zettlemoyer and Collins, 2005], is used. This model is represented by the function:

$$p_v(a|u) \propto exp(v \cdot f(u, a)), \tag{3.1}$$

where a represents an action, u is the utterance given by the user, v is the parameters vector, and f is the function that maps the pair (u, a) to a feature vector represented by $f(u, a)$.

As mentioned before, v contains the mapping between each feature and its weight. At the end of each iteration, the weight of the features used will be updated by the *learner* component.

3.1.3.B Learner

This component receives the feedback given by the user and, based on this feedback, updates the parameters vector.

After receiving the feedback (in this case, the state chosen by the user), the system goes through all the derivations and mark as "correct" the ones that, applied to the current state, result in the state chosen by the user. The next step consists in applying the loss-function to every derivation, where the previous derivations marked as "correct" will have a higher score than those that were not. The loss function, presented by [Wang et al., 2016], is represent by the Equation 3.2 (where *ns* is the new state and *cs* is the current state).

$$
\begin{aligned}
l(v, u, ns) &= -log(p_v(ns|u, cs)) + \lambda||v||_1, \\
p_v(ns|u, cs) &= \sum_{a:||a||_{cs}=ns} p_v(a|u).
\end{aligned}
\tag{3.2}
$$

During the previous step, the derivations marked as "correct" will have a positive score, while the derivations that were not marked as "correct" will have a negative score. Having in mind that each derivation has an action A associated, and there is a set of features SF that contains different relations between A and the utterance introduced by the user, the system updates the parameters vector doing the following procedure:

- For each feature within SF, verifies if the feature is already on the parameters vector;
- If it is, adds/subtracts (if the derivation is marked as "correct" or not) the weight generated to the previous feature weight;
- If it is not, adds that feature to the parameters vector with the correspondent weight.

Lastly, a single Gradient Update is performed using AdaGrad algorithm [C. Duchi et al., 2010]. This algorithm performs two types of updates: the smaller ones intended for frequent features, and larger ones intended for infrequent features, which adapts correctly the learning rate to the different features.

Now that we reviewed the SHRDLURN game, made an overview of the architecture and all the fundamental components, it is time to explain how we added both explicit and active learning processes to the system. In the next sections, we will talk about the implementation of both learning processes, what needed to be changed, and what the two processes of learning bring to the extension of the system. We named this new system SHRDLURN+.

3.2 Explicit Learning

With the explicit learning process, we intend to extend the type of feedback that each user can provide to the system. So far, our system received user feedback related with actions performed, being those actions the result of an utterance introduced by the user.

Our goal with this new learning process is to allow the user to explicitly tell what s/he is trying to say with the feedback provided. To do this, the user must choose a domain concept and input the utterance, that is used by the user, to refer to the concept chosen.

In terms of the learning component, this process does not introduce major changes in the system. Remembering that, so far, the system received feedback in form of a new state. Then, in order to apply the loss function (Equation 3.2), the system needed to mark as "correct" the derivations that lead to the new state.

The difference in this new learning process, is that the system has to mark as "correct" the derivations that are related to the concept chosen by the user. This makes the loss function give more weight to features that represent relations between the utterance and the concept introduced by the user.

In the continuation of this section, we will explain how we implemented this learning process in the SHRDLURN+.

3.2.1 Explicit Learning in SHRDLURN+

Until now, the user could only give feedback to the system by introducing instructions (utterances) with the propose of playing the SHRDLURN game. We introduced a new way for the feedback to be given, where the user is allowed to explicitly tell the system what s/he is uttering.

We introduce this change in the system gradually, by first testing this new functionality with simple information (color and position) and, then, we present the final implementation of the explicit learning, which combines both color and position information. Let us now review the different types of feedback tested and introduced in our work.

3.2.1.A Learning Colors

We start by allowing the user to select a block and introduce an utterance that represents the color of the block selected (*blue*, *brown*, *red*, or *orange*). Figure 3.4 illustrates the menu created and an example where the user chooses the red block and introduces the utterance "rouge" (French for "red").

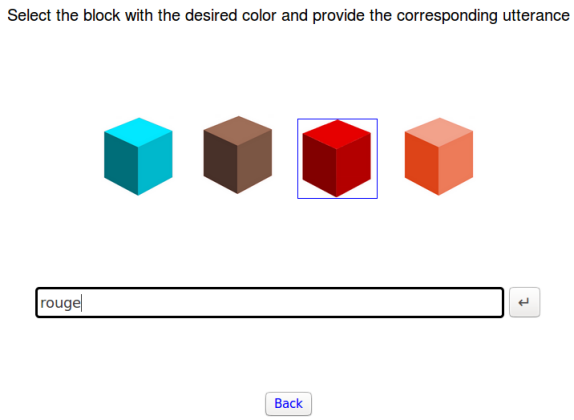

Figure 3.4: Color menu

The implementation of this functionality is done through:

1. Receiving the user feedback in the form of "X U" (being X $\in$ [0, 3] the index of the block with the desired color and U the utterance — 0 is blue, 1 is brown, 2 is red, and 3 orange);
2. Generating all the possible actions, and, using the parser component, generate also all the features, associated with each action and the utterance U;
3. Selecting, among all the actions, as "correct", the actions that contain the color of the block X. The actions related to the color have the form represented in Figure 3.5, where if the action is the type *stackOnTop* (in the Figure 3.5, on the top of block(s) with the color 3 add a block with the color 0), we consider that the action is correct if either the first or the second blocks have the same color as the block X. On the other hand, if the action is the type *removeTop* (in the Figure 3.5, remove the block with the color 3), we consider that the action is correct if the color of the block is the same as the block X;
4. Running the normal loss-function and the AdaGrad algorithm, in order to update the features with the information collected.

```
(call context:removeTop (call complement (call rightMost1 (call context:getTopColor (number 3 COLOR)))))))
(call context:stackOnTop (call complement (call context:getTopColor (number 3 COLOR))) (number 0 COLOR))))
```

Figure 3.5: Example of actions generated

Let us consider the results of a simple test, where the user is presented with the game [[2],[3],[1],[2],[2]] | [[2],[],[1],[2],[2]] (on the left, the initial state and on the right, the goal state - the goal is to delete the orange block). Figure A.2 (in appendix) shows, on the left image, the index where the correct answer is, after the user introduces the utterance "apagar laranja". On the other hand, on the right image, the same

result is shown, but in the scenario where the user first enters the explicit feedback “laranja” associated with the orange block.

The previous example shows that the correct answer goes from index 24 to index 0, if the user first introduces the explicit feedback about the block color that s/he wants to remove. Which represents an improvement for the system.

3.2.1.B Learning Position

The second type of feedback that we now allow users to introduce is related with the position of the blocks (*left* or *right*). Figure 3.6 illustrates the menu created and an example where the user chooses the position right and introduces the utterance “droite” (French for “right”).

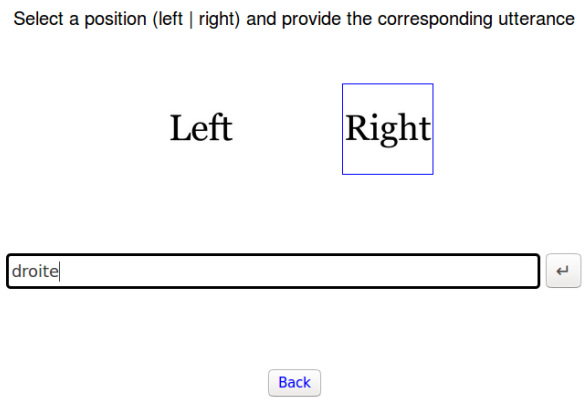

Figure 3.6: Position menu

In parallel with the last functionality, the implementation of this functionality is made by:

1. Receiving the user feedback in the form of “X U” (being X $\in$ [0, 1] the index of the position, and U the utterance — 0 is left and 1 is right);
2. Generating all the possible actions, and, using the parser component, generate also all the features, associated with each action and the utterance U;
3. Selecting, among all the actions, as “correct”, the actions that contain the position X. As an example, we can see both types of actions accepted, in Figure 3.7, being the first one correspondent to the “left” position and the second one to the “right” position;
4. Running the normal loss-function and the AdaGrad algorithm, in order to update the features with the information collected.

As in the previous functionality, let us now consider the results of a simple test, where the user is presented with the game [[0],[3],[2],[3],[3],[3],[3]] | [[],[3],[2],[3],[3],[3],[3]] (the goal is to remove the left block).

```
(call context:removeTop (call leftMost1 (call context:getTopColor (number 3 COLOR))))))
(call context:removeTop (call rightMost1 (call context:getTopColor (number 0 COLOR))))))
```

Figure 3.7: Example of actions generated

Figure 3.8 shows a ranked list of possible states and, inside the red rectangle, the index where the correct answer is, after the user introduces the utterance “apagar esquerda”. Figure 3.9 shows the same result, but in the scenario where the user first enters the explicit feedback “0 esquerda”.

The previous example shows that the correct answer goes from index 2 to index 0, if the user first introduces the explicit feedback about the block position that s/he wants to remove. Which represents an improvement for the system.

```
0 - [[],[],[],[],[],[],[]]
1 - [[0],[3],[2],[3],[3],[3],[3]]
2 - [[],[3],[2],[3],[3],[3],[3]]
3 - [[0],[3],[2],[3],[3],[3],[]]
4 - [[],[3],[2],[3],[3],[3],[3]]
5 - [[0],[3],[2],[3],[3],[3],[3]]
6 - [[0],[3],[],[3],[3],[3],[3]]
7 - [[0],[],[2],[],[],[],[]]
8 - [[0,0],[3,0],[2,0],[3,0],[3,0],[3,0],[3,0]]
9 - [[0,1],[3,1],[2,1],[3,1],[3,1],[3,1],[3,1]]
10 - [[0,2],[3,2],[2,2],[3,2],[3,2],[3,2],[3,2]]
11 - [[0,3],[3,3],[2,3],[3,3],[3,3],[3,3],[3,3]]
12 - [[0],[],[],[],[],[],[]]
13 - [[],[],[],[],[],[],[]]
14 - [[],[],[2],[],[],[],[]]
15 - [[],[3],[],[3],[3],[3],[3]]
...
```

Figure 3.8: Scenario without explicit feedback

```
0 - [[],[3],[2],[3],[3],[3],[3]]
1 - [[0,0],[3],[2],[3],[3],[3],[3]]
2 - [[0,1],[3],[2],[3],[3],[3],[3]]
3 - [[0,2],[3],[2],[3],[3],[3],[3]]
4 - [[0,3],[3],[2],[3],[3],[3],[3]]
5 - [[0],[],[],[],[],[],[]]
6 - [[],[3],[2],[3],[3],[3],[3]]
7 - [[0],[3],[2],[3],[3],[3],[3]]
8 - [[0],[3],[],[3],[3],[3],[3]]
9 - [[0],[],[2],[3],[3],[3],[3]]
10 - [[0,0],[3],[2],[3],[3],[3],[3]]
11 - [[0,1],[3],[2],[3],[3],[3],[3]]
12 - [[0,2],[3],[2],[3],[3],[3],[3]]
13 - [[0,3],[3],[2],[3],[3],[3],[3]]
14 - [[0],[3],[2],[3],[3],[3],[3]]
15 - [[0],[3],[2],[3],[3],[3],[3]]
...
```

Figure 3.9: Scenario with explicit feedback

3.2.1.C Learning Color & Position

In order to increase the complexity of our functionalities, we decided to put the two functionalities, presented so far, together in one. So, we have a type of feedback that not only exactly tells what the user is saying to the system, but also introduces the need to disambiguate this information. The ambiguity is introduced by giving the system a feedback that can either be related with a position or related to a color. Next, we illustrate, with examples, how the system handles this situation.

The idea consists in letting the user choose a block (from a random configuration of blocks) and introduce an utterance. This utterance can be related with the color of the block or with the position of the block (exclusively).

Figure 3.10 illustrates the menu created and two examples: on the top image, the user chooses the blue block and introduces the utterance “bleu” (French for “blue”); on the bottom image, the user chooses the position left and introduces the utterance “gauche” (French for “left”).

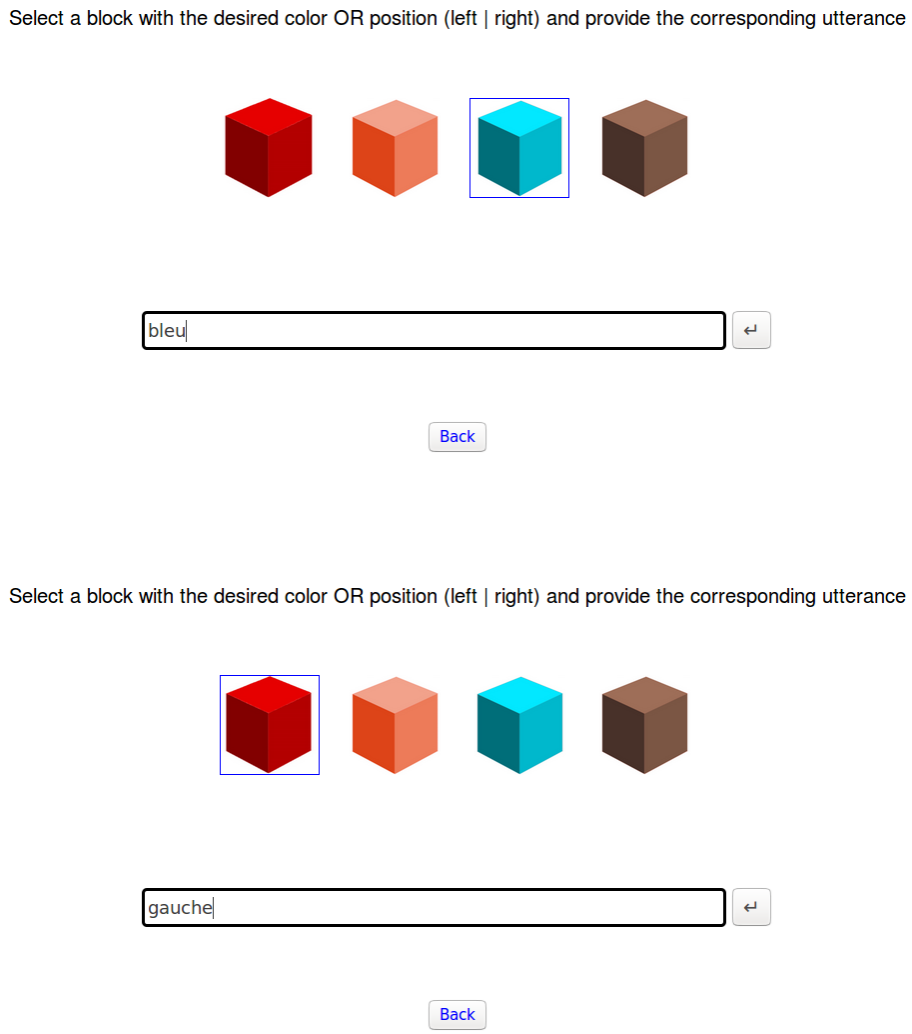

Figure 3.10: Color & Position menu

The implementation of this functionality it is a merging of the two previous ones, where the system:

1. Receives the user feedback in the form of "X U" (being X $\in$ [0, 3] the index of the block with either the desired position or color, and U the utterance);
2. Generates all the possible actions, and, using the parser component, generate also all the features, associated with each action and the utterance U;
3. Selects, among all the actions, as "correct", the actions that contain the color/position X;
4. Runs the normal loss-function and the AdaGrad algorithm, in order to update the features with the information collected.

During step 3, the system does not know if the *X* component corresponds to a color or to a position.

However, we are assuming that if the user chooses the blocks in the middle (indexes 1 or 2), s/he will be talking about a color and not a position (considering that if the user is presented with a line of 4 blocks and intends to give feedback about a position, it is more likely the user chooses one of the blocks on the extremes - left/right - than the ones in the middle). Another important consideration is that, if the user selects the brown block on the right (index 3 in figure 3.10) and, for example, introduces the utterance “droite”, the features related with both position *rightMost* and color *brown* will have the same weight. Nevertheless, as the game progresses the initial ambiguity is solved by itself, since feature related with “droite” and the position *rightMost* will be overweighted in relation to feature related with “droite” and the color *blue*.

Following the thought line of the previous functionalities, let us now consider the results of a simple test, where the user, after being presented with the configuration displayed in Figure 3.4, selects the brown block and introduces the explicit feedback “marron” (French for “brown”).

After this, the user is presented with the game [[2],[0],[0],[1],[1]] | [[2],[0],[0],[],[]] (remove the two brown blocks). Figure 3.11 shows the index where the correct answer is, after the user introduces the utterance “supprimer marron”, and Figure 3.12 shows the same result, but in this scenario where the user first enters the explicit feedback.

The previous example shows that the correct answer goes from index 18 to index 8, if the user first introduces the explicit feedback about the block position/color that s/he wants to remove. Which represents an improvement for the system.

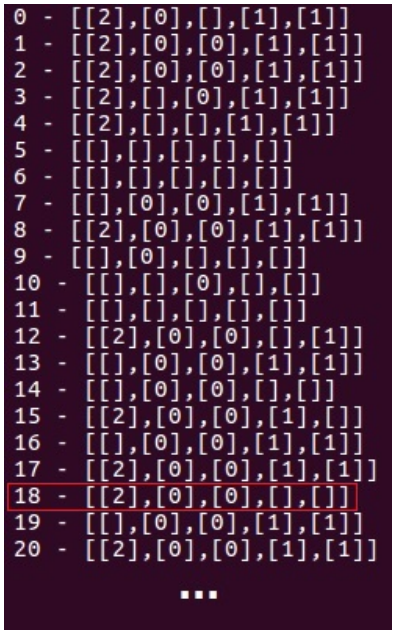

Figure 3.11: Scenario without explicit feedback

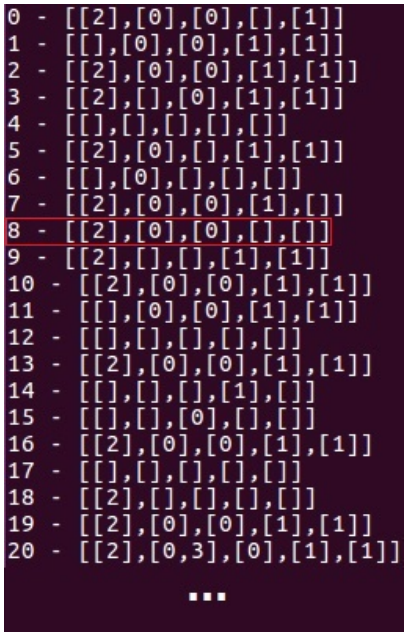

Figure 3.12: Scenario with explicit feedback

3.3 Active Learning

In order to make the system more autonomous, by letting it have the ability to decide when additional information from the user is needed, we are now introducing the concept of *active learning*. In this type of learning, the system decides which data is more valuable to learn from and asks information about that data to the user.

For our system in particular, we want it to be able to decide when is the proper time to interrupt the game and ask the user for feedback about a certain utterance. That utterance is the one, at that moment, considered the most informative.

In terms of the learning component, this process does not introduce changes in the system, since the feedback provided by the user has the same form as in the implicit learning process. While in the implicit learning the user provides the utterance and the state associated with it, the objective here is that the system chooses an utterance and the user provides a state representative of that utterance.

In the continuation of this section, we will explain how we implemented this learning process in the SHRDLURN+.

3.3.1 Active Learning in SHRDLURN+

When one wants to implement active learning in the system, there are a few decisions to take. First, we want to decide the type of *scenario* and the *query strategy* that are more suitable for the system. Next, we explain not only the decision we made for our system in particular, but also the implementation of the active learning in the SHRDLURN+ game. For a better understanding, let us consider Figure 3.13, which not only illustrates the active learning menu, but also shows an example where the user is presented with the instance “apagar vermelho” (Portuguese for “remove red”), and already scrolled through the options until reaching the correct one.

3.3.1.A Architectural decisions

Scenario: We use *pool-based sampling* (previously mentioned in Subsection 2.4.1) for the active learning functionality. The system first checks its pool of instances and selects the instance that considers the most informative one. Then, constructs a set of blocks representative of the instance selected and present it to the user. Finally, the user gives feedback by selecting a new state of blocks, which is the result of applying the instance presented to the initial set of blocks.

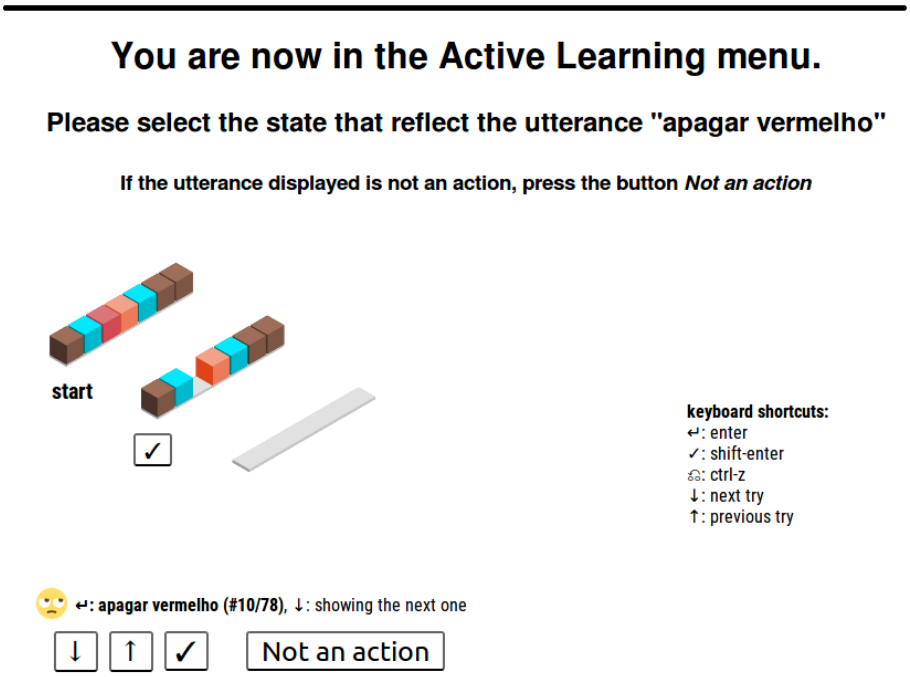

Figure 3.13: Active Learning menu

Query strategy framework: The system needs a metric to sort the information, available on its knowledge base, in order to decide which instance should be queried to the user. In our implementation, we used the *uncertainty sampling* framework (previously mentioned in Subsection 2.4.2), and more specifically, we used the entropy strategy. To do this, the system estimates the uncertainty for each instance and selects the least confident instance (the one with more uncertainty). In other words, the system asks for a label to the instance that it has less certain at the moment, within its instances set.

In the continuation of this section, we will describe how we implement the previous theoretic decisions. First, we will explain how the system selects the most informative utterances, to be presented to the user. Then, how the system constructs the representative set of blocks, so the user can give the feedback about the utterance presented.

3.3.1.B Selecting the most informative instance

Since the SHRDLURN game is already based on the user feedback, we need to take into account the negative impact, on the user experience, that a new feedback request will bring ([Su et al., 2016]). Having this in mind, we are only triggering the active learning component at every three iterations of the game (to prevent the possibility of the active learning be triggered at all iterations) and when a few conditions are verified. Those conditions have the purpose to make the system choose the instance that it is more uncertain about, and, at the same time, to take into account that some instances might not

make sense to the user (for example, if the user entered the utterance “remove blue block”, the system could ask for the instance “block”, which would not make sense to be asked, in this context, to the user). We will now describe those conditions while talking about the process of selecting the utterance. Then, we will explain *why* we select those conditions and talk about other approaches that we tested before we reach the following approach:

- **Condition 1:** At each iteration of the game, the user enters new utterances, in order to interact with the system. Those utterances are represented by the system as n-grams. We are storing the uni-grams with a count associated with each one (if the user enters “delete blue” and “delete red” we have “delete”:2, “blue”:1, “red”:2). The *first condition* is associated with those counts, where we select only the uni-grams that have a count higher or equals to 3;

- If the previous condition is verified by any uni-gram, we select the two uni-grams (or one, if there is only one uni-gram that verifies the previous condition) with the more uncertainty, let us call it UNI1 and UNI2 for convenience. To the uncertainty calculation for each uni-gram, we use the Formula 3.3 (entropy formula), where x is an utterance and z is an action from the set of all possible actions Z. Is also important to notice that in Formula 3.3, $p(x|z)$ is calculated using the *Log-linear model formula* from 3.1. After we compute the entropy for each one of the uni-grams selected in the previous condition, we select the one with higher entropy;

$$Entropy(x) = -\sum_{z \in Z} p(x|z) \log(p(x|z)) \tag{3.3}$$

- **Condition 2:** As mentioned, the system uses a representation of n-grams to the utterances, more specifically uni-grams, bi-grams, tri-grams, and skip-grams. Our *second condition* is to select the bi-grams, tri-grams, and skip-grams that contain both UNI1 *AND* UNI2. If the previous condition does not verify for any n-gram, we select the ones that contain UNI1 *OR* UNI2. For convenience, let us call SET-X the set of bi-grams, tri-grams, and skip-grams that verify the previous condition;

- If SET-X is not empty, we need to select the instance, within the SET-X, with more uncertainty. To do this, one more time, we use the Formula 3.3 and select the instance with more entropy. In other words, we are selecting the utterance that the system has less certainty of what it means.

Note: In the continuation of this section, we will refer a few times to the terms *short phrases* and *long phrases*, the first term is referent to phrases as “remove blue” or “add blue red”, while in comparison the second term is referent to phrases like “remove the blue block” or “add blue block in red”.

During the development of our solution, we tested other approaches (with changes in the previous conditions) to select the proper instance to be presented to the user, but after early tests, we decided that the

best approach is the one described above (for convenience, let us call it Final-Approach). Nevertheless, we will talk, bellow, about the other approaches tested. Presenting the results for those approaches, and for the Final-Approach, in Chapter 4.

Approach 1: At first, we tried to work with all the n-grams, but since we are in a domain that is based on instructions, we decide that the system should ask for utterances with two or more words (usually give an instruction, requires at least two words), which made us consider only bi-grams, tri-grams, and skip-grams for the construction of the utterances to be asked to the user. While testing this scenario, we realize that in short phrases the system was having a worst performance comparing with the tests made in the system without the active learning component. Also, in long phrases, the system not only had worst results, but also took a considering amount of extra time, due to the computations made in the active learning.

Approach 2: In this approach, we introduced the second condition presented in the Final-Approach, which means we are now taking account that the n-grams presented in the SET-X need to have UNI1 and/or UNI2, but UNI1 and UNI2 are computed over all the uni-grams presented in the game so far. The difference is that in the Final-Approach we are only considering the uni-grams with 3 or more occurrences, which takes into account that the user can introduce a sporadic term or mistype something, and even though those words will have a higher uncertainty (since the system does not have much information about them), they will not be considered. This approach showed a performance improvement using long phrases, however, the system continued to perform better without using active learning.

Approach 3: In this approach, we introduce the first condition of the Final-Approach, where we consider only the uni-grams with 3 or more occurrences. However, we made a change in the second condition, where the system, while choosing the SET-X, does not verify first if there are any n-grams with UNI1 and UNI2. Instead, it jumps directly to the condition of having n-grams with UNI1 **or** UNI2, which makes SET-X less restricted. This approach showed performance improvement using long phrases, even outperforming the system without active learning. Also, using short phrases, this approach had a better performance than every previously presented approach. However, the system continued to perform better without using active learning for short phrases.

All the three approaches presented were outperformed by the Final-Approach in the early tests, results that can be seen in Section 4.1 of the next chapter. It is also important to mention that the conditions introduced in the previous approaches were motivated, during early tests, due to the utterances chosen by the active learning component before having any of those conditions. We realized that the system was asking information about utterances that had no semantic meaning to the user, which was caused by the way that the information is represented to the system (in Section 4.1 we can see the utterances asked to the user in the different approaches tested). So we have a better understanding of what was causing this problem, let us consider an example: if we introduce, in the first iteration, the

utterance “remove blue block”, the two bi-grams “remove blue” and “blue block” would have the same meaning for the system and, even though the utterance “remove blue” would make sense to be asked to the user, the utterance “blue block” could make no sense to the user (we can think of Figure 3.13 with the utterance “remove blue” – which would make sense for the user – and with the utterance “blue block” – which could not make sense to the user). Another possible approach to mitigate this problem, would be by introducing semantic restrictions in the pool of instances (for example, select only among the ones that contain a verb), however, this solution cannot be implemented in our system, since the user can use any type of natural language and, then, we are unable to identify the verbs.

3.3.1.C Building a representative set of blocks

After the system chooses the utterance to be presented to the user, a state of blocks needs to be constructed. So, the user can give feedback to the system of what the utterance selected means, when applied to that state.

Since there is no score/rank associated with a state of blocks, we have no way to choose a “optimal” state to present to the user. So we do not have predefined states, we chose to apply some randomness to the creation of the state, by the following the next procedure:

- Generate 6 random states and select 10 random actions (from the set of all possible actions);
- Apply each action to each one of the six states;
- Calculate the number of Changes per State (CPS);
- Choose the state with most average CPS.

The CPS of a particular state is the sum of different blocks after applying each one of the 10 actions to the original state. For example, considering only two actions, if the first action gives a state that has 3 different blocks compared to the original state, and the second action gives a state that has 2 different blocks compared to the original state, the CPS of this state is 5.

We also consider a linear relation between the informativeness of a state of blocks and the number of different colors of blocks (for example, a state with only blue blocks is less informative than a state with blocks of all colors), so, in addition to the number of changes in blocks, we also consider the number of different colors (*ndc*), being the equation to choose the state of blocks, that should be presented the user, the following:

$$State(s) = \arg\max_{s} \left(\frac{cps(s) + ndc(s)}{10 + 1} \right) \tag{3.4}$$

As previously seen in Figure 3.13, the selected state is presented to the user jointly with the utterance previously selected. The user is also presented with a ranked list of states, from where the user must choose the state that reflects the changes when applying the presented utterance to the initial state.

As mentioned before, it is possible for the system to prompt the user for information about an utterance that has no semantic meaning. For those cases, we add an option where the user can tell the system that the presented utterance is not related to an action, which makes the system ignore this iteration.

In the end, if the user gives feedback in form of a new state, the system performs the usual update, by applying the lost function (3.2) and the AdaGrad algorithm, using the utterance selected by the system, the initial state generated, and the final state chosen by the user. In case the user selects the "Not an action" option, the system ignores this iteration and follows the normal flux of the game.

3.4 Extending the System to Other Test Domains

One possible modification to our work is the test domain used in its implementation. In our solution, the SHRDLURN game was always the test domain (proof-of-concept) used to demonstrate the applicability, to do tests, and to discuss results related with the different approaches that make up our solution. Nevertheless, at a certain point, we can ask ourselves "what if we want to implement this solution in a different domain?".

The SHRDLURN game is based on learning the mapping between the user language and the internal language of the system. The user language can be any type of natural language, as long the user intends to use it to collaborate with the system. The internal language of the system is composed of actions (add or delete blocks) that are defined compositionally using a predefined grammar, and also composed of objects characteristics (blocks colors and relative positions). Having these requirements in mind, we can think of several different domains where the system could be applied.

As a possible domain, let us think of an agent developed to be a traditional surgeon assistant. The agent interacts with a surgeon that asks for surgical instruments and the agent task is to store/pick up those instruments in/from the surgical instrument table.

A prerequisite to this system, would be a speech recognition component, transform the instructions given by the surgeon into natural language text. However, this component is out of the scope of our thesis, so we will only assume that this component exists, so that system receives as input written natural language utterances.

The first step to accomplish this task with our system, is to change the predefined grammar. Instead of the actions remove and add, the agent now has to perform the actions of grab and store. Then, in terms of object concepts, the system no longer needs to have the notion of blocks with colors. Instead, the system needs to have an internal representation for all the surgical objects.

In terms of the processes of learnings, the whole system remains the same. The only difference would be in the type of information inputted and outputted by the system:

- Implicit learning: The user inputs an utterance related with a request about storing/picking up surgical instruments. The system presents a list of candidates with the different types of actions applied to the different types of instruments. The user gives feedback to the system by choosing the correct choice.
- Explicit learning: The user chooses an instrument and inputs an utterance representative of that instrument.
- Active learning: The system evaluates the pool of utterances received and selects the most informative one (or part of it) to be asked to the user. Then, the user should give feedback by selecting the correspondent action to the question asked by the system.

4

Evaluation/Results

Contents

To evaluate the performance of our system, we use the same **evaluation metric** presented in [Wang et al., 2016], where the authors measure the score of each user by the average number of scrolls, per utterance, made in the game (for example, if the user makes 190 scrolls and introduces 10 utterances during a game, the score of the player is 19). Each scroll represents the distance from the first position on the list of possible states presented by the system after the user entered an utterance. For example, if the user plays a game where one of the lists of possible actions is the same as represented in Figure 4.1, and the correct choice is represented by the red rectangle, we say that the user made 2 scrolls.

```
0 - [[],[],[],[],[],[],[]]
1 - [[0],[3],[2],[3],[3],[3],[3]]
2 - [[],[3],[2],[3],[3],[3],[3]]
3 - [[0],[3],[2],[3],[3],[3],[]]
                ...
```

Figure 4.1: Example of a list of possible states

This evaluation metric is also used to evaluate the performance of the four main approaches tested in the development of the active learning component (as we presented in Section 3.3.1 of the previous chapter).

As we mentioned in Section 3.3.1, all the four approaches presented were motivated by the questions asked by the system after triggering the active learning component. Having this in mind, to know which approach should be used in the final implementation, we not only made tests related with the score, but also related with the quality of the questions asked by the active learning component (if those questions had a semantic meaning to the user). In the continuation of this chapter, we will talk about those tests (early tests) and present the respective results. Then, we will talk about the user tests made to the final implementation, where we explain our decisions and present the final results.

4.1 Early Tests on Active Learning

In order to simplify the presentation of the results related to the four approaches (presented in Section 3.3.1), let us identify and remember what each one consists of:

- Approach 1: Approach where we restrict the active learning component to ask utterances composed by bi-grams, tri-grams, and skip-grams;

- Approach 2: Approach where we calculate the UNI1 and UNI2 and, then, we restrict the SET-X to utterances that contain UNI1 and/or UNI2 (only verify the *or* condition if the *and* condition is not verified by any utterance);

- Approach 3: Approach where we restrict UNI1 and UNI2 to uni-grams that have occurred more than three times, but, when defining the SET-X, we restrict it to utterance that contain UN1 OR UNI2;
- Final-Approach: Approach that merges the previous ones. More specifically, restricts the calculation of UNI1 and UNI2 to uni-grams that have occurred at least three times, and restricts SET-X to utterances that contain UNI1 and/or UNI2 (only verify the *or* condition if the *and* condition is not verified by any utterance).

During the early tests, we used Portuguese language to test the active learning component. We used two types of language, one with *short phrases* and other with *long phrases*. Table 4.1 represents the Portuguese utterances used in both types of languages (C1 and C2 represent two colors, that can be the same color or different ones).

English	Portuguese Short Phrase	Portuguese Long Phrase
remove C1 block	apagar C1	apagar bloco C1
remove C1 blocks	apagar C1	apagar blocos C1
remove not C1 blocks	apagar n C1	apagar blocos nao C1
remove left/right block	apagar esquerda/direita	apagar bloco esquerda/direita
add line of C1 blocks	adicionar C1	adicionar blocos C1
add C1 block on C2 block	adicionar C1 C2	adicionar bloco C1 em C2
add C1 blocks on C2 blocks	adicionar C1 C2	adicionar blocos C1 em C2
add C1 blocks on not C2 blocks	adicionar C1 n C2	adicionar blocos C1 em nao C2
add C1 block left/right	adicionar C1 esquerda/direita	adicionar bloco C1 esquerda/direita

Table 4.1: Utterances used in early tests to the active learning component

While we were doing the early tests, we wrote down the different questions asked by the system. We can see those question in Figure 4.2. In the figure, for each approach, we have the questions asked during the tests with short phrases (represented with 1-) and with long phrases (represented with 2-). The **X** mark after the utterance, means that that question had no meaning to the user. It is important to refer that during the game, is the system that decides when it is the proper time to trigger the active learning component. Due to this fact, while testing the system with long phrases, only the first approach has six questions, while all the remaining approaches have five questions.

In Figure 4.2, we can see that on the Approach 1, while using long phrases, most of the questions asked, had no meaning to the user. This problem was mitigated with the introduction of the conditions

```
Approach 1:

1-(utterance "apagar azul")
  (utterance "adicionar direita")
  (utterance "adicionar azul")
  (utterance "adicionar laranja")
  (utterance "vermelho laranja")
  (utterance "laranja azul")

2-(utterance "apagar laranja")
  (utterance "adicionar azul")
  (utterance "adicionar em") X
  (utterance "em vermelho") X
  (utterance "blocos em") X
  (utterance "adicionar bloco em") X
```

```
Approach 2:

1-(utterance "apagar azul")
  (utterance "adicionar vermelho")
  (utterance "adicionar direita")
  (utterance "adicionar laranja")
  (utterance "adicionar azul")
  (utterance "vermelho esquerda")

2-(utterance "apagar laranja")
  (utterance "bloco em") X
  (utterance "adicionar em") X
  (utterance "adicionar direita")
  (utterance "apagar esquerda")
```

```
Approach 3:

1-(utterance "apagar azul")
  (utterance "adicionar direita")
  (utterance "adicionar azul")
  (utterance "adicionar laranja")
  (utterance "vermelho laranja")
  (utterance "apagar direita")

2-(utterance "apagar laranja")
  (utterance "bloco direita")
  (utterance "adicionar em") X
  (utterance "adicionar direita")
  (utterance "bloco em") X
```

```
Final-Approach:

1-(utterance "apagar azul")
  (utterance "adicionar laranja")
  (utterance "adicionar castanho direita")
  (utterance "adicionar laranja castanho")
  (utterance "adicionar vermelho")
  (utterance "adicionar laranja azul")

2-(utterance "apagar laranja")
  (utterance "adicionar bloco") X
  (utterance "adicionar em") X
  (utterance "adicionar bloco direita")
  (utterance "blocos vermelho laranja")
```

Figure 4.2: Active learning questions on different approaches

present in the other approaches. As result, from Approach 1 to all the other approaches, there was a decrease of questions that have no meaning to the user (dropped from 67% to 40%).

On the other hand, if we compare Approach 2, Approach 3, and the Final-Approach, that are no major differences in terms of the number of questions that had no meaning to the user. Nevertheless, if we compare those approaches in terms of the score (using the evaluation metric of our work) the results already show differences. Let us now present and discuss those results, so we can see why the Final-Approach was chosen over the other two.

In order to evaluate the different approaches in terms of score, we made two games (one with short phrases and one with long phrases) with the original game[1] and two games with each approach. The results of those games can be seen in Figure 4.3. The games with short phrases are represented by "1-" and the games with long phrases are represented by "2 -". The "vs" tag separates the score of the games with the different approaches and the score of the original game. For example, "Approach 2: 1 - 5.65 vs 4.6" means that the game with the Approach 2 and using short phrases, had a score of 5.65, while the original game using short phrases, had a score of 4.6.

[1] Original game is the game without explicit and active learning components

```
Approach 1:
1 - 5.3 vs 4.6
2 - 8.25 vs 7.3

Approach 2:
1 - 5.65 vs 4.6
2 - 7.65 vs 7.3

Approach 3:
1 - 5.28 vs 4.6
2 - 7.12 vs 7.3

Final-Approach:
1 - 4.97 vs 4.6
2 - 6.9 vs 7.3
```

Figure 4.3: Evaluation scores on different approaches vs original game

As we suspected in Section 3.3.1, the Final-Approach outperforms all the other approaches. Although the Final-Approach does not outperform the original game in short phrases, it is the approach with the best score when using this type of phrases. And more important, the Final-Approach outperforms the original game when using long phrases. These results were what made us choose the Final-Approach over all the other ones.

4.2 User Tests on the Final Implementation

To evaluate the changes that *explicit* and *active* learning components introduced to the system, we are replicating the procedure used in [Wang et al., 2016], where the authors tested the system by making user tests. Our tests are composed of two games:

- Original game: The user plays the game without explicit or active learning. Since each player can use a different language, the original game needs to be played in each test, so we have a baseline.
- Extended game: The user plays the game with explicit and active learning.

The original game is constituted of 20 iterations, divided into 6 levels. We decided to reduce the number of iterations per game, in comparison with the original paper [Wang et al., 2016] (where the authors made 50 iterations per game), so the user does not have to spend 2 hours doing the test (considering

each game took on average 1 hour to complete, according to the original paper). On the other hand, the extended game is also constituted of 20 iterations plus the explicit learning iterations (which depends on how much times the user decides to enter explicit feedback) plus the active learning iterations (which depends on how much times the system triggers the active learning component).

During the initial tests, we noticed that the users usually made inconsistencies in the instructions used (e.g., use the word “remove” and then start using the word “delete”) and, sometimes, the users provide an utterance and then realize a better option is available and chooses that option instead (e.g., an iteration where the user needs to remove both blue and orange blocks, the user enters the utterance “remove blue blocks”, but, then, sees an option where both blue and orange blocks are removed, and selects that option). Both problems cause a negative impact on user performance during the game. In the first problem, considering the previously given example, since the word “delete” is unknown to the system (so far the user always used the word “remove”), it needs to learn a new word, which causes the correct choice not to be at the top of the ranked list. This causes the number of scrolls to be higher than would be if the user had used always the same word. On the other hand, the second problem makes the system receive wrong feedback. As we saw on the given example, the system receives the utterance “remove blue blocks” with, for example, the action *remove(not(with(red)))* (which means remove all blocks that are not red).

The two previous problems happened mostly during the first game performed by the user. The reason we found for this situation is: during the first game, the user initially has to understand the game concept (even the user receives basics instruction before the game starts), and in the second game the user not only has more understanding about the game, but also knows the better strategies to solve each iteration (even if the set of blocks are constructed randomly, at the start of each game, the seed of the random number is reseted, which causes most of the iterations, of both games, to be similar). In order to mitigate this situation, we made the users start, randomly, either by the original game or by the extended game.

At the beginning of each game, a set of instructions is presented to the user (as shown in Figure 4.4). To avoid biasing the type of language used by the user, we provide no example of utterances used or any examples of interactions with the system. We tested our system with 15 users, and, since all the users were Portuguese, the languages used to perform the tests were mainly Portuguese or English.

In the continuation of this section, we will present and discuss the results of the user tests. Then, we are going to present a few strategies used by the users and discuss which ones were the most/the less effective on our system.

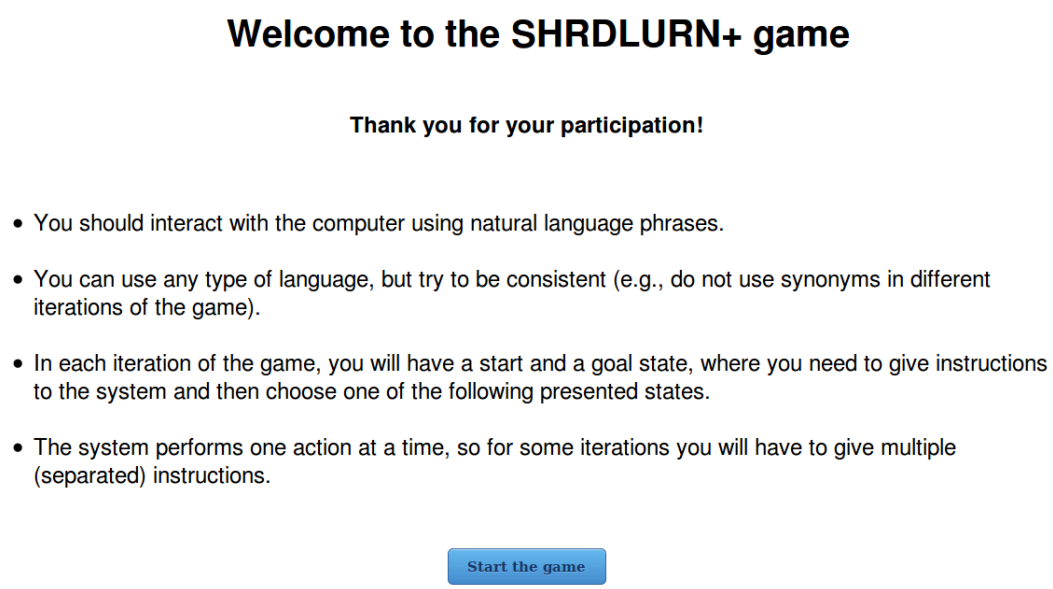

Figure 4.4: System introduction screen

4.2.1 User Tests Results

As we mentioned before, we are using the **number of scrolls** as the evaluation metric of each game. Since the system goal is to, at each iteration of the game, give the best prediction of the correct action for the utterance given by the user, the number of scrolls must be the **smallest possible**. In other words, after the user gives an utterance, a ranked list of actions is calculated by the system, and, in that list, the higher the correct action, the more accurate the system is.

Having this in mind, we can see the results of the users tests, to our system, in Figure 4.5. In the figure, the blue line represents the score of the *original game* and the red line the score of the *extended game*. We can see that in test number 1, test number 6, and test number 9, the original game had the best performance, although, in all the other 12 tests, our approach outperformed the original game.

After taking a close look to Figure 4.5, we can see that most of the first five tests have, in terms of score, a greater distance than the rest of the tests. This is caused by the previously mentioned problems, where we noticed that the users always scored better in the second game. The first test started with the extended game and all the other four started with the original game, causing the starting game to have a higher number of scrolls. To mitigate this situation, at the beginning of each test, we made the users aware of the inconsistency problem (as we can see in the second point of Figure 4.4). We also alert the users to the importance of giving the correct feedback for the utterance provided.

The two previous modifications, in the initial instructions, made the score of the tests 6 to 15 to be more close, and, in our opinion, more reliable.

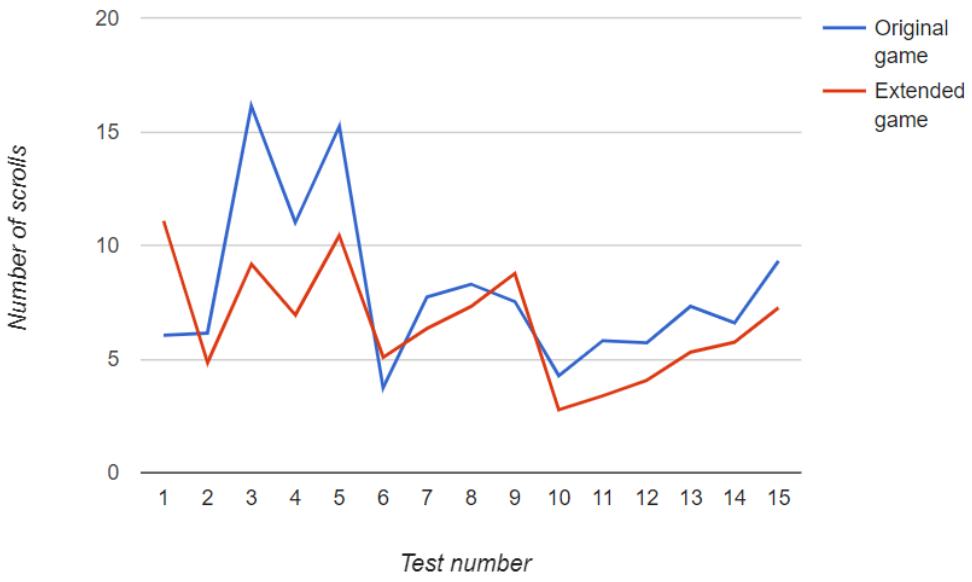

Figure 4.5: Graphic of scores on the users tests

4.2.2 User Tests Strategies

All the users performed the tests either using Portuguese or English language. Most of them used common language as, for example, “remove red block”, “add blue on left”, or “add brown squares on top of orange squares”. However, there were some strategies that stood out, both by positive and by negative ways. In terms of strategies with better performance, worst performance, and the oddest utterances used, we have:

- Better strategy in original game: The test with the lower number of scrolls per utterance, in the original game, was test number 6 (Figure 4.5). The user used utterances as “remove brown block”, “remove two red blocks”, “add brown blocks top blue”. The final score was 3.74.

- Worst strategy in original game: The test with the highest number of scrolls per utterance, in the original game (and also in the set of all tests), was the test number 3 (Figure 4.5). The user used utterances as “laranja”, “castanho”, “delete azul”. The final score was 16.12.

- Better strategy in extended game: The test with the lower number of scrolls per utterance, in the extended game (and also in the set of all tests), was the test number 10 (Figure 4.5). The user

used utterances as “apagar castanho”, “adicionar laranja azul”, “adicionar azul esquerda”. The final score was 2.77.

- Worst strategy in extended game: The test with the higher number of scrolls per utterance, in the extended game, was the test number 1 (Figure 4.5). The user used utterances as “remove brown block leftmost”, “insert orange blocks top”, “delete one bottom orange block”. The final score was 11.08.

- Oddest strategy: The test with the most odd type of utterances used, was the test number 15 (Figure 4.5). The user used commands as “b”, “r”, “br”, “del b 1”, “add r 2 3 4”. The final score was 9.31 for the original game and 7.25 for the extended game.

Through the last points, we can conclude that the best strategies have in common the conciseness and coherence in the language syntax, being, in our opinion the most important factors to perform well in our system. Nevertheless, if we are too concise (for example, in test 3, where the user did not use, most of the time, words to represent actions, but, instead, used only words to represent the colors), we may risk giving less information to the system than the necessary for it to learn the language game. In the other hand, if we are not coherent (for example, in test 1, the user used both “remove” and “delete” to teach the same action), the system will learn slower the game language, which causes the performance within the game to be worst.

The conciseness and coherence in the language used are also two important factors for explicit learning and active learning components, separately. In terms of *explicit learning*, if the user chooses to use a concise and coherent language, the system has fewer words to associate with its internal language, which causes the information given through the explicit learning, to be easily associated with the correct terms. On the other hand, in terms of *active learning*, using a concise language makes easier the construction of utterances with meaning to the user. For example, if the utterances used contain articles (“*the* orange block”) or prepositions (“add blue *on top of* red”), those words can be asked to the user in an utterance that has no meaning to her/him (situation illustrated in Figure 4.2, while comparing utterances asked by the system in the game tests with short phrases and long phrases).

5

Conclusions and Future Work

In this document, we presented a system that learns a language skill, by receiving feedback provided by a user, and incorporates that feedback in the learning process. Different from the current state of the art, we take advantage of the processes of *implicit learning*, *explicit learning*, and *active learning* to accomplish our goal. As a proof of concept, we used the idea of language games, more specifically, the game presented in [Wang et al., 2016], named SHRDLURN. Since the work developed in [Wang et al., 2016] already contains the implicit learning process necessary to our work, we reuse the authors system and add it two new processes of learning: the explicit and the active learning. We named this new system "SHRDLURN+".

The learning component in all the three processes differ in the type of feedback and in the way we get the user feedback:

- In the implicit learning, the user is presented with an initial state and a goal state, and has to input an utterance(s) that represent(s) the action(s) necessary to transform the initial state into the goal state.
- On the other hand, in the active learning, the user is presented with an initial state and with an utterance that the system is uncertain about. Here, the user has to choose a new state that represents the application of the utterance to the initial state.
- In the explicit learning process, the user gives feedback by choosing a block (from a set of multiple blocks) and entering an utterance related to that block. The information of this block can be either related with the color or with the position (since the block is chosen from a set of multiple blocks, it has a relative position in relation to the other blocks).

All the three learning processes use a loss function to update the system knowledge. This function is used after the user gives feedback to the system, where the relations between the utterance (and parts of it) and the action/block properties inputted gain more weight. After the system has applied the loss function, it also performs a single Gradient Update using AdaGrad algorithm.

In order to evaluate our system, we chose to make user tests, being those tests composed by two games: 1. original game (game with only implicit learning process); 2. extended game (game with all three learning processes). The original game is our baseline to compare with the extended game.

As the evaluation metric, we chose the average number of scrolls, per utterance introduced, made in the game. Each scroll represents the distance from the first position on the list of possible states, presented by the system.

The tests were performed by 15 users, all of them Portuguese, which makes the strategies used mostly based on the Portuguese and English languages. In terms of score, the users that used concise

and coherent languages, had the best scores. Although, one of the users used a too concise language (where s/he has omitted the actions) and got the worst score within the tests.

In terms of results, 12 out of 15 users performed better with our approach (extended game) in comparison with the original game. Based on that, we conclude that our approach, where we use the three processes of learning, bring an improvement in terms of how much information the system needs to learn the user language game.

Looking forward, we imagine that our system can be applied across a range of different domains. As we have seen, if the new domain contains the restrictions associated with our system, the extensibility of our system to that domain, is almost direct. As the next step, we would like to see how our system would behave when applied to a real-world learning problem.

In terms of active learning, we consider that the use of n-grams (from uni-grams to tri-grams) restricts the questions made by the system. A possible modification, would be the use of a context-free grammar, that would be updated throughout the multiple interactions between the system and the user. Then, using this grammar during the choice of utterances in the active learning component, the system would produce utterances with more semantic meaning for the user.

Bibliography

[Alomari et al., 2017] Alomari, M., Duckworth, P., Hogg, D., and Cohn, A. (2017). Natural language acquisition and grounding for embodied robotic systems. In *AAAI Conference on Artificial Intelligence*.

[Anderson, 2007] Anderson, D. (2007). Adult language learning, lecture notes: Li2 - language variation. University of Cambridge.

[Baum and Lang, 1992] Baum, E. B. and Lang, K. (1992). Query learning can work poorly when a human oracle is used. In *International joint conference on neural networks*, volume 8, page 8.

[Bruneau, 2014] Bruneau, E. G. (2014). Cognitive neuroscience: Implicit vs. explicit learning activity. MIT - Massachusetts Institute of Technology.

[Buck et al., 2017] Buck, C., Bulian, J., Ciaramita, M., Gesmundo, A., Houlsby, N., Gajewski, W., and Wang, W. (2017). Ask the right questions: Active question reformulation with reinforcement learning. *CoRR*, abs/1705.07830.

[Burger et al., 2017] Burger, J., Cardie, C., Chaudhri, V., Gaizauskas, R., Harabagiu, S., Israel, D., Jacquemin, C., Lin, C.-Y., Maiorano, S., Miller, G., Moldovan, D., Ogden, B., Prager, J., Riloff, E., Singhal, A., Shrihari, R., Strzalkowski, T., Voorhees, E., and Weischedel, R. (2017). Issues, tasks and program structures to roadmap research in question & answering (Q&A).

[C. Duchi et al., 2010] C. Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization. In *The Journal of Machine Learning Research*, volume 12, pages 257–269.

[Curto et al., 2014] Curto, S., Mendes, A., Curto, P., Coheur, L., and Costa, A. (2014). JUST.ASK, a QA system that learns to answer new questions from previous interactions. In *LREC 2014, Ninth International Conference on Language Resources and Evaluation, European Language Resources Association (ELRA), Iceland, May 2014*.

[Dagan and Engelson, 1995] Dagan, I. and Engelson, S. P. (1995). Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ICML'95, pages 150–157.

[Fujii et al., 1998] Fujii, A., Tokunaga, T., Inui, K., and Tanaka, H. (1998). Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 24(4):573–597.

[Golland et al., 2010] Golland, D., Liang, P., and Klein, D. (2010). A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 410–419, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Henderson et al., 2014] Henderson, M., Thomson, B., and Young, S. (2014). Word-based dialog state tracking with recurrent neural networks. *SIGDIAL*, pages 292–299.

[Hoi et al., 2006] Hoi, S. C. H., Jin, R., and Lyu, M. R. (2006). Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 633–642. ACM.

[Kirk and Laird, 2014] Kirk, J. and Laird, J. (2014). Interactive task learning for simple games. In *Advances in Cognitive Systems*, volume 3, pages 13–30.

[Laird, 2012] Laird, J. E. (2012). *The Soar Cognitive Architecture*. The MIT Press.

[Lampinen, 2017] Lampinen, A. K. (2017). 224n project: Natural language learning supports reinforcement learning.

[Liang, 2015] Liang, P. (2015). SEMPRE: Semantic parsing with execution. Standford University.

[Ling and Fidler, 2017] Ling, H. and Fidler, S. (2017). Teaching machines to describe images via natural language feedback. *CoRR*, abs/1706.00130.

[McCallum and Nigam, 1998] McCallum, A. and Nigam, K. (1998). Employing EM and pool-based active learning for text classification. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 350–358.

[Sachan and Xing, 2017] Sachan, M. and Xing, E. (2017). Learning to solve geometry problems from natural language demonstrations in textbooks. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, pages 251–261, Vancouver, Canada. Association for Computational Linguistics.

[Schein and Ungar, 2007] Schein, A. I. and Ungar, L. H. (2007). Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265.

[Scheutz et al., 2017] Scheutz, M., Krause, E., Oosterveld, B., Frasca, T., and Platt, R. (2017). Spoken instruction-based one-shot object and action learning in a cognitive robotic architecture. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1378–1386, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Settles, 2010] Settles, B. (2010). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

[Settles et al., 2008] Settles, B., Craven, M., and Ray, S. (2008). Multiple-instance active learning. In *Advances in Neural Information Processing Systems 20*, pages 1289–1296. Curran Associates, Inc.

[Skočaj et al., 2016] Skočaj, D., Vrečko, A., Mahnič, M., Janíček, M., Kruijff, G.-J. M., Hanheide, M., Hawes, N., Wyatt, J. L., Keller, T., Zhou, K., Zillich, M., and Kristan, M. (2016). An integrated system for interactive continuous learning of categorical knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(5):823–848.

[Song and Kautz, 2012] Song, Y. C. and Kautz, H. A. (2012). A testbed for learning by demonstration from natural language and RGB-depth video. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

[Su et al., 2016] Su, P., Gasic, M., Mrksic, N., Rojas-Barahona, L. M., Ultes, S., Vandyke, D., Wen, T., and Young, S. J. (2016). On-line active reward learning for policy optimisation in spoken dialogue systems. *CoRR*, abs/1605.07669.

[Tür et al., 2005] Tür, G., Hakkani-Tür, D., and Schapire, R. E. (2005). Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171 – 186.

[Wang et al., 2016] Wang, S. I., Liang, P., and Manning, C. D. (2016). Learning language games through interaction. *CoRR*, abs/1606.02447.

[Wittgenstein, 1953] Wittgenstein, L. (1953). *Philosophical Investigations*. Blackwell, Oxford.

[Zettlemoyer and Collins, 2005] Zettlemoyer, L. S. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

[Zhao and Eskénazi, 2016] Zhao, T. and Eskénazi, M. (2016). Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. *CoRR*, abs/1606.02560.

A

Appendix

```
tree :: getNonEmpty.0-[azul, *, *]
tree :: removeTop.1-getNonEmpty.0-[azul, *, *]
tree :: wallToString.1-removeTop.1-getNonEmpty.0-[azul, *, *]
subtree :: (wallToString (removeTop (getNonEmpty))) :: [azul, *, *]
subtree :: (wallToString (removeTop *)) :: [azul, *, *]
subtree :: (removeTop (getNonEmpty)) :: [azul, *, *]
subtree :: (wallToString *) :: [azul, *, *]
subtree :: (removeTop *) :: [azul, *, *]
subtree :: (getNonEmpty) :: [azul, *, *]

tree :: getNonEmpty.0-[apagar, *, *]
tree :: removeTop.1-getNonEmpty.0-[apagar, *, *]
tree :: wallToString.1-removeTop.1-getNonEmpty.0-[apagar, *, *]
subtree :: (wallToString (removeTop (getNonEmpty))) :: [apagar, *, *]
subtree :: (wallToString (removeTop *)) :: [apagar, *, *]
subtree :: (removeTop (getNonEmpty)) :: [apagar, *, *]
subtree :: (wallToString *) :: [apagar, *, *]
subtree :: (removeTop *) :: [apagar, *, *]
subtree :: (getNonEmpty) :: [apagar, *, *]

tree :: getNonEmpty.0-[apagar, azul, *]
tree :: removeTop.1-getNonEmpty.0-[apagar, azul, *]
tree :: wallToString.1-removeTop.1-getNonEmpty.0-[apagar, azul, *]
subtree :: (wallToString (removeTop (getNonEmpty))) :: [apagar, azul, *]
subtree :: (wallToString (removeTop *)) :: [apagar, azul, *]
subtree :: (removeTop (getNonEmpty)) :: [apagar, azul, *]
subtree :: (wallToString *) :: [apagar, azul, *]
subtree :: (removeTop *) :: [apagar, azul, *]
subtree :: (getNonEmpty) :: [apagar, azul, *]
```

Figure A.1: Features generated for the utterance “apagar azul” and the action *remove(leftmost(nonEmpty))*

```
0 - [[2],[3],[1],[2],[2]]
1 - [[],[3],[1],[2],[2]]
2 - [[2],[3],[1],[2],[2]]
3 - [[2],[3],[1],[2],[2]]
4 - [[],[],[],[],[]]
5 - [[2],[3],[1],[2],[2]]
6 - [[],[],[],[],[]]
7 - [[2],[3],[1],[2],[2]]
8 - [[2],[3],[1],[2],[2]]
9 - [[],[],[],[],[]]
10 - [[2,0],[3],[1],[2],[2]]
11 - [[2,0],[3,0],[1,0],[2,0],[2,0]]
12 - [[2],[],[],[],[]]
13 - [[],[],[],[],[]]
14 - [[2],[3,0],[1,0],[2,0],[2,0]]
15 - [[2],[3],[1],[2],[]]
16 - [[2],[3],[1],[2],[2]]
17 - [[2],[3],[1],[2],[2]]
18 - [[2],[3],[1],[2],[2]]
19 - [[2],[3],[1],[2],[2]]
20 - [[2],[3],[1],[2],[2]]
21 - [[2],[3],[1],[2],[2]]
22 - [[2],[3],[],[2],[2]]
23 - [[],[3],[1],[2],[2]]
24 - [[2],[],[1],[2],[2]]
25 - [[2],[3],[1],[2],[2]]
26 - [[2],[3],[1],[2],[2]]
27 - [[2],[3],[1],[2],[2]]
28 - [[2],[3],[1],[2],[2]]
29 - [[],[],[],[],[2]]
30 - [[2],[3],[],[2],[2]]
...
```

```
0 - [[2],[],[1],[2],[2]]
1 - [[2],[],[1],[2],[2]]
2 - [[2],[],[1],[2],[2]]
3 - [[2],[3],[1],[2],[2]]
4 - [[],[3],[1],[2],[2]]
5 - [[2],[3],[1],[2],[2]]
6 - [[],[],[],[],[]]
7 - [[],[],[],[],[]]
8 - [[2],[3],[1],[2],[2]]
9 - [[2],[3],[1],[2],[2]]
10 - [[2],[3],[1],[2],[2]]
11 - [[2],[3],[1],[2],[2]]
12 - [[2],[3],[1],[2],[2]]
13 - [[2],[3],[1],[2],[2]]
14 - [[2],[3],[1],[2],[2]]
15 - [[],[3],[],[],[]]
16 - [[],[],[],[],[]]
17 - [[],[3],[],[],[]]
18 - [[2],[3,0],[1],[2],[2]]
19 - [[2],[3,0],[1],[2],[2]]
20 - [[2],[3,3],[1],[2],[2]]
21 - [[],[3],[],[],[]]
22 - [[2],[3,3],[1],[2],[2]]
23 - [[2],[3,0],[1],[2],[2]]
24 - [[2],[3,3],[1],[2],[2]]
25 - [[2,0],[3],[1],[2],[2]]
26 - [[2,3],[3],[1],[2],[2]]
27 - [[2,0],[3,0],[1,0],[2,0],[2,0]]
28 - [[2,3],[3,3],[1,3],[2,3],[2,3]]
29 - [[2],[],[],[],[]]
30 - [[2,0],[3],[1,0],[2,0],[2,0]]
...
```

Figure A.2: Early tests results for the *explicit learning* considering only the *color* component.
Left: scenario without explicit feedback — Right: scenario with explicit feedback