Learning language skills based on implicit, explicit and active learning

Luís Miguel Santos Henriques
luis.santos.henriques@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal
October 2016

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.

1. Introduction

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 a human language (among others) and make it analyzable in order to find relevant information.

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 ([4], [7]). 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 [12] where the system learns a language game, and by [9] where the system learns characteristics of the objects in its environment.

In this work, we intend to develop a system that uses the feedback given by the user to learn a language skill, taking advantage of 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 ([2], [3]), implicit and explicit learning can be defined as:

- Implicit learning: The system learns new information through exposure. This type of learning consists in 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.

- 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, a sys-
tem that is learning a language game, the user tells the system that when s/he inputs the utterance "bloc rouge" s/he is referring to a 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. This behavior is usually called active learning, and following [8], 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.

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.

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 [12], 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 transform an initial state into a goal state (being each state composed of a set of blocks).

The rest of the document is organized as follows: Section 2 gives an overview of the state of the art, by exposing several works that use the different types of learning that we are interested in; Section 3 presents the architectural decisions made to our system and the implementation of them; Section 4 presents and discusses the results obtained after making user tests to the system; Section 5 presents our conclusions.

2. Related Work

Our literature review of related works relevant to our thesis is divided according to the different types of learning used in our solution.

In our work we use the show-and-tell procedure applied to the explicit learning process. An example of an application of this procedure is represented in [1], where the authors present a framework that receives video clips of a manually controlled robot arm and natural language commands describing the actions. The authors train the system with the mapping between the different parts of the video clips and the corresponding parts of the natural language commands.

In the type of information that we are learning with the explicit learning process, we chose to follow [6] and adopt the idea of receiving information about the basic concepts, in order to acquire a better knowledge of the system domain. In the mentioned previous work, the authors aim to develop a system that learns the problem specifications through natural language instructions received by humans. The authors developed an agent named Rosie, which can play simple spatial games and puzzle. 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.

In terms of the implicit learning process required for our work, we follow the approach presented by [12], where the authors explore the idea of learning a language skill using a language game (concept introduced by [13]). In this work is developed a game named SHRDLURN, which is integrated in the world of blocks. In SHRDLURN 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.

After the user inputs an instruction, the system presents a sorted list of new states (each one represents an action applied to the initial state, being the list sorted by what the system finds most likely the instruction of the user to mean).
Finally, the user gives feedback to the system by choosing the correct state within the presented list. This feedback is then used to update the system knowledge base.

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 ([8]). In our work, we allow the system to take initiative and ask questions to the user. Those questions are representative of gaps identified by the system in its knowledge. The objective of active learning in our work is to make the system learn faster the user language game, by requesting feedback about utterances (or parts of it) that the user previously entered.

We made a similar approach to the presented in the robotic system named George ([9]), where the system receives linguistic information from a tutor and integrates that information with visual perceptions about objects placed in the system domain. During this process, not only the tutor gives information to the system, but also the system itself takes the initiative to ask questions about the objects. 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.

We followed [8] to choose the active learning approach that best fit our work. In terms of scenario, we used a pool-based sampling, where from a pool of unlabeled instances, the system chooses which one is the most informative to be labeled by the user. This scenario is applied in many real-world learning problem, such as in the speech recognition area, where the authors of [11] used pool-based 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.

As a proof-of-concept for our system, we decided to use the game developed in [12], 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.

3. SHRDLURN Overview

There are a few key concepts that are fundamental to understanding the system:

**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:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>all</td>
</tr>
<tr>
<td>Color</td>
<td>true, [true, red], orange</td>
</tr>
<tr>
<td>Set → Set</td>
<td>true, true, true, true, true</td>
</tr>
<tr>
<td>Set</td>
<td>north, [true, west]</td>
</tr>
<tr>
<td>Color → Action</td>
<td>east, true, true</td>
</tr>
<tr>
<td>Set → Action</td>
<td>true, true, true, true, true</td>
</tr>
</tbody>
</table>

![Figure 1: SHRDLURN grammar](image)

**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).

**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.

3. Solution

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 [12], 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.

3.1. SHRDLURN Overview

There are a few key concepts that are fundamental to understanding the system:

**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:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>all</td>
</tr>
<tr>
<td>Color</td>
<td>true, [true, red], orange</td>
</tr>
<tr>
<td>Set → Set</td>
<td>true, true, true, true, true</td>
</tr>
<tr>
<td>Set</td>
<td>north, [true, west]</td>
</tr>
<tr>
<td>Color → Action</td>
<td>east, true, true</td>
</tr>
<tr>
<td>Set → Action</td>
<td>true, true, true, true, true</td>
</tr>
</tbody>
</table>

**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).

**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.
**Features weights:** each feature has a weight, that represents how strong is the relation between the utterance and the action associated with that feature.

**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.

**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.

**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.

In a high-level vision, we can think of the SHRDLU system as having two main components, the Parser and the Learner.

**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 is used:

\[
p_v(u|a) \propto \exp(v \cdot f(u,a)),
\]

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.

**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 [12], is represent by the equation 2 (where \(ns\) is the new state and \(cs\) is the current state).

\[
l(v,u,ns) = -\log(p_v(ns|u,cs)) + \lambda||v||_1, \quad p_v(ns|u,cs) = \sum_{a:||u||_2 = ns} p_v(a|u).
\]

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 [5]. 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.

### 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 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 with to 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.

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.

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).

The implementation of this functionality is done through:
1. Receiving the user feedback in the form of “X U” (being X ∈ [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;
4. Running the normal loss-function and the AdaGrad algorithm, in order to update the features with the information collected.

Learning positions The second type of feedback that we now allow users to introduce is related with the position of the blocks (left or right).

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 ∈ [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;
4. Running the normal loss-function and the AdaGrad algorithm, in order to update the features with the information collected.

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 to a position or related to a color.

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). 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 ∈ [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 of the line, 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, and, for example, introduces the utterance “droite”, the features related with both positions 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.

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.

3.3.1 Active Learning in SHRLURN+

When one wants to implement the 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.

Scenario: We use pool-based sampling 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.

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, and more specifically, we used the entropy strategy. To do this, the system estimates the uncertainty for each instance and selects the least confident one (the one with more uncertainty). In other words, the system asks for a label to the instance that has less certainty at the moment, within its instances set.

Selecting the most informative instance

Since the SHRLURN 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 ([10]). 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 being 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 be asked, (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 (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, $p(x|z)$ is calculated using the Log-linear model formula from 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;

$$\text{Entropy}(u) = - \sum_{a \in A} p(u|a) \log(p(u|a)) \quad (3)$$

- **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.

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 Section 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.

- **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.

- **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.

All the three approaches presented were outperformed by the Final-Approach in the early tests, results that can be seen in Section 4.

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.

### 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.

We also consider a linear relation between the informativeness of a state of blocks and the number of different colors of blocks, 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:

### 3.4. Extending the System to Other Test Domains

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.

### 4. Results & discussion

To evaluate the performance of our system, we use the same evaluation metric presented in [12], 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.

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). 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).

**Early Tests on Active Learning.** 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.

While we were doing the early tests, we wrote down the different questions asked by the system. We can see those question in Figure 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.

**Figure 2:** Active learning questions on different approaches

In Figure 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 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.

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 and two games with each approach. The results of those games can be seen in Figure 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.

**Figure 3:** Evaluation scores on different approaches vs original game

As we suspected in Section 3.3, 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.

**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 [12], 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 in 6 levels. 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 two problems that caused a negative impact in game performance. First, the users usually made in-
consistencies in the utterances used (e.g., both utterances “remove” and “delete”), which slows the learning of the system (the system has to learn more words to the same concepts). Second, sometimes, the users provide an utterance and a wrong feedback (e.g., the user wants to remove both blue and red blocks, starts to enter “remove blue blocks” but then chooses an option where both blue and red blocks were removed). The two previous problems happened mostly during the first game performed by the user. The reason we found for this situation is: in the second game the user not only has more understanding about the game, but also knows the better strategies to solve each iteration. In order to mitigate this situation, we made the users start, randomly, either by the original game or by the extended game.

User Tests Results. 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. 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.

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. 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, 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. To mitigate this situation, at the beginning of each test, we made the users aware of the inconsistency problem and alert them to the importance of giving the correct feedback for the utterance provided. Those 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: Graphic of scores on the users tests

All the users performed the tests either using Portuguese or English language. 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: Test number 6 (Figure 4). 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: Test number 3 (Figure 4). The user used utterances as “laranja”, “castanho”, “delete azul”. The final score was 16.12.
- Better strategy in extended game: Test number 10 (Figure 4). The user used utterances as “apagar castanho”, “adicionar laranja azul”, “adicionar azul esquerda”. The final score was 2.77.
- Worst strategy in extended game: Test number 1 (Figure 4). 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: Test number 15 (Figure 4). 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.

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), we may risk giving less information to the system than the necessary for it to learn the language game. The same happens if we are not coherent
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 proof of concept we used the idea of language games, more specifically, the game presented in [12], named SHRDLURN. Since the work developed in [12] 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+”.

We performed tests with 15 user, where the users that used concise an coherent languages, had the best scores.

In terms of results, 12 out of 15 users performed better with our approach in comparison with the original system. 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 seen, if the new domain contains the restrictions associated with our system, the extensibility of our system to that domain, is almost direct. As 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.

References


