# XAI on Reinforcement Learning

## Explaining Tic Tac Toe moves

MARTIN HECKEL, Hochschule für Angewandte Wissenschaften Hof, Deutschland

In the recent decades, there was much research on the models used for Artificial Intelligence (AI). As a consequence, the predictions of those models were becoming much better. However, typically there is a trade-off between the prediction quality and the explainability: The explanation of the chosen decision is the harder to explain the better the predictions of the model are.

In general, humans question decisions that they do not understand, especially when the decision was made by a mathematical model and not by another human. From a technical perspective, even complex models are self explaining: There is a set of parameters and a parameterized function that used the parameters and input variables to calculate an output value (e.g. for classification). Thereby, the explanation of the output value with the parameters and input variables leads to a reproducible result: Repeating the calculation with the parameters and input variables will result in the same decision. However, humans are not able to understand that kind of explanation.

For this reason, the ability to explain the decisions of an AI model in a way that a human can understand why the output value was calculated, is important for applied AI. In addition to the increase in trust for the decisions humans understand, the decisions have to be explainable for some applications, e.g. are insurance companies in the United States required to justify their decision if they denied coverage. That justification is not possible when the decision was made by an AI model that can not be understood. For that reason, those models can not be used for applications that require explanation.

To solve that problem, the research field of eXplainable Artificial Intelligence (XAI) has the goal to develop approaches to explain AI models that are not explainable in a way understandable for humans based on the model itself.

In this semester paper, Local Interpretable Model-Agnostic Explanations (LIME) is adjusted in a way that it can be used on a Reinforcement Learning model trained to play the game *Tic Tac Toe*. While that scenario has no relevance for practical applications, it helps to understand the approaches. Also, it makes evaluation more easy because the modified version of LIME can be tested with explainable algorithms in order to verify its results.

CCS Concepts: • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: XAI, LIME, Reinforcement Learning

## 1 INTRODUCTION

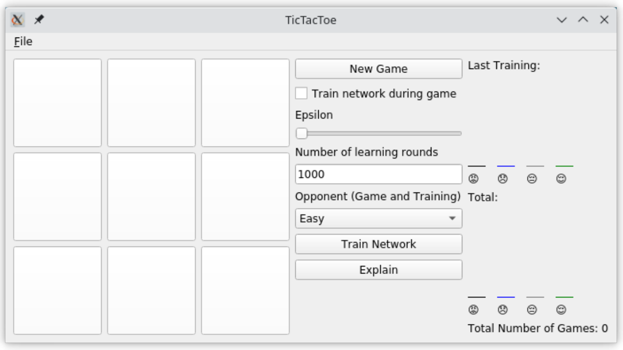
Humans trust decisions of other humans, which they can not understand, often without resistance, especially when those other humans are higher ranking. It would be possible to ask the other humans to explain their decision. However, this is often not done and the decision is just trusted.

If the decision was not made by another human but a mathematical model, there are more trust issues: Before humans trust the model they often require an explanation of the decision the model has done. If that explanation is comprehensible, they trust the decision.

For some applications it is required to explain the decisions that were made. For example, insurance companies in the United States have to justify their decision if they decline coverage [4].

In general, there is a trade-off between the performance (e.g. prediction quality) and explainability of a model: The better the predictions of the model are, the harder they are to explain in a way that humans can understand them. The goal of eXplainable Artificial Intelligence (XAI) research is to develop approaches that can be used to explain models that are hard to explain based on their internal functionality.

For this semester paper, the Local Interpretable Model-Agnostic Explanations (LIME) algorithm is adjusted in a way it can be used for Reinforcement Learning. Afterwards, it is evaluated with the game *Tic Tac Toe*. I have developed the game with some basic opponents in the scope of the module "Angewandtes maschinelles Lernen" during the bachelor program. In the scope of this module, I did some adjustments to the architecture, implemented a backtracking based optimal opponent, adjusted the network design and implemented the modified version of LIME described in this paper to evaluate the approaches discussed in the following sections. Figure 1 shows the Graphical User Interface (GUI) of the currently implemented version of the game.



Last Training:

Number of learning rounds: 1000

Opponent (Game and Training): Easy

Total:

Total Number of Games: 0

Fig. 1. GUI of the *Tic Tac Toe* implementation

An Artificial Intelligence (AI) for an opponent in a game does not require explanation or trust from humans. However, implementing XAI in the field of Reinforcement Learning for the application area of games has the benefit that, at least for simple games, the strategies are easy to understand for a human. For that reason, a human

Author's address: Martin Heckel, martin.heckel@hof-universiy.de, Hochschule für Angewandte Wissenschaften Hof, Alfons-Goppel-Platz-1, Hof, Bayern, Deutschland, 95028.

can easily decide what action he would have chosen in a given situation and explain why he would have selected the chosen and no other action.

Another benefit is the fact that model agnostic XAI approaches can be used to explain the behaviour of simple opponent implementations. During that, it can be evaluated if the model agnostic approach gave an explanation that corresponds to the known and human-understandable internal implementation. Thereby, the implementation of the XAI algorithm can be evaluated easily because the explanation of a turn is already known due to the understanding of the internal implementation.

## 2 BACKGROUND

In this section, some basics about Reinforcement Learning and XAI are described. Afterwards, the LIME algorithm is introduced briefly.

### 2.1 Reinforcement Learning

In Reinforcement Learning, there is an environment (e.g. the game field) that has a *state*. The algorithm chooses an *action* based on that state. Performing the action by interacting with the environment results in a *following state*. That procedure is repeated until the following state is final, e.g. the game is over. In order to estimate how good the action that was taken in a given state is, a *value* is calculated for that action (see Figure 2).

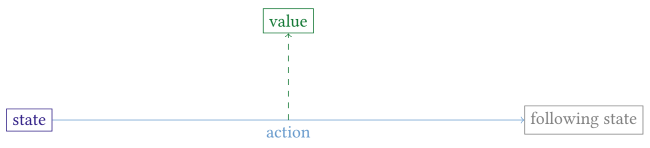


Fig. 2. Graphical representation of a basic Reinforcement Learning approach

However, often there is no possibility to calculate the value of an action while the game is still running because estimation of mid-game states can be pretty hard. If that would not be the case, one could simply perform all possible actions in a given state and calculate the value of those actions. Afterwards, the action with the best value could be chosen. The game *Tic Tac Toe*, for example, is simple enough to estimate the value of mid-game states, which is implemented as "optimal opponent" using the Minimax algorithm based on backtracking. However, *Tic Tac Toe* can be used for Reinforcement Learning as well.

Because value estimation is typically not possible in mid-game states, the value of an action can often not be determined directly. That problem is solved by storing the turn (e.g. state and action) in a buffer and continuing the game without calculating the value. When the game is over, the final value of the game is known. Depending on the game, that can be the score, the distance the player reached, or the final state (e.g. "Won", "Draw", "Lost"). Using that final value, the values of the turns stored in the buffer can be calculated afterwards. This is done by multiplying the value of the next action with a factor $\gamma$. That approach is shown in Figure 3

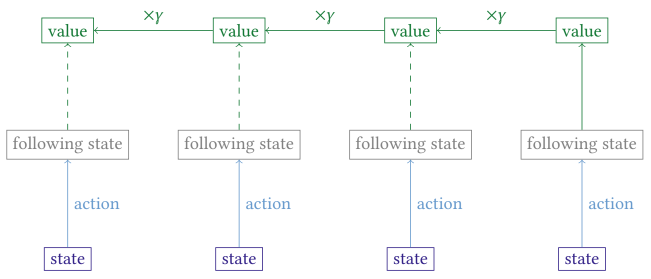


Fig. 3. Graphical representation of the buffer for Reinforcement Learning

The value should thereby converge to the *neutral* value, e.g. the value of taking a random valid turn that is neither considered to be *good* nor *bad* for winning the game. Depending on the neutral value of the model, the calculation has to be adjusted. In the model used for that study paper, the neutral value was 0.5, so the calculation had to be adjusted in a way that the first turn in the buffer would have a value of 0.5 if there would be $\infty$ turns stored in the buffer.

### 2.2 eXplainable Artificial Intelligence

The goal of XAI is to explain decisions of a model in a way that these decisions can be understood by humans. The XAI approaches can be distinguished based on the following parameters [5]:

- Time the explanation is created
  - **Post-Hoc**: The explanation is created after the model has done the prediction (e.g. the result of the prediciton is already known).
  - **Intrinsic**: The explanation is created while the model does the prediction, the result of the precition is not known at that point.
- Relation to the model
  - **Model agnostic**: The XAI algorithm is independent of the used model. That approach requires knowledge about the representation of the input and output parameters and handles the model itself as a black box by generating input parameters and measuring output parameters. The explanations are typically created based on the predictions of the model for several generated variations of the original input parameters.
  - **Model specific**: The XAI algorithm does depend on the used model. Knowledge about the inner workings of the model are required to calculate the explanation (white box). This can result in better explanations than the model agnostic approaches but requires knownledge of the model. Algorithms for one type of model can therefore typically only be used to explain models of the same kind.

- Explanation scope
    - **Global**: The global functionality of the model is explained for all possible inputs. The explanation that is generated explains how the entire model works.
    - **Local**: The functionality of a model is explained for a single input. The explainations that is generated explains why the according input resulted in the corresponding output without explaining the entire model.

## 2.3 Local Interpretable Model-Agnostic Explanations

LIME [6] was introduced by Ribeiro et al. and can be used to explain the predictions of any classifier. This was demonstrated by Ribeiro et al. with different models for text and image classification. LIME is a post-hoc, model agnostic and local approach.

For images, the identification is done by selecting *relevant* parts of the images that should be classified. This is implemented by splitting the image into multiple parts and conducting predictions for the single parts. For those predictions, the other parts are *removed.* Technically, this is done by setting those removed parts to a neutral color, e.g. gray. If the predicted probability for the class is higher for a set of parts than for the entire image, those parts are marked as *relevant* for the selection of according class In the end, the explanation is the combination of those relevant parts.

## 3 IMPLEMENTATION

In this section, the architecture of the *Tic Tac Toe* game is explained in more detail. Afterwards, the architecture of the Neuronal Network that is trained and the architecture of the software project itself are briefly described.

## 3.1 Tic Tac Toe

In the current version of the *Tic Tac Toe* implementation, the state of the game is encoded as vector of size 18. The first 9 values represent the game field from the perspective of the first player, the other 9 values represent the game field from the perspective of the second player. When a field has the value 1 is is used by the according player, if it has 0 it is not used. When a field has a value of 0 for both players, it is empty.

The actions are encoded as a vector of size 9. Each field represents the predicted value for the corresponding field in the game. In the training phase, the chosen actions (vector fields with the highest values) are adjusted in a way that the value is set to the one calculated after the game ended using the final value of the game and the factor $\gamma$ as described in Section 2.1. The other fields keep there value because they were not used as action and, therefore, no estimation of their influence is possible.

The current implementation supports several opponents:

- The **random** opponent does any random turn (that turn does not have to be valid).
- The **easy** opponent does any valid random turn.
- The **medium** opponent does the turn to avoid the opponent from winning the game if that is possible. Otherwise, it does any valid random turn.
- The **hard** opponent does the turn to win the game if that is possible. Otherwise, it does the turn to avoid the opponent from winning with the next move when that is possible. Else, it does any valid random turn.
- The **optimal** opponent uses the Minimax algorithm to always play an optimal game. It does either win or play a draw.
- The **network** opponent queries the used neuronal network for a prediction and performs the predicted action.

## 3.2 Architecture of the Neuronal Network

The network used in the scope of this study project takes a vector of size 18 as input and a vector of size 9 as output. See Section 3.1 for a description of the usage of these vectors in the semantic of the game *Tic Tac Toe.* It consists of several layers that are aligned to a sequential model. The neuronal network is implemented in Python using the Keras [1] library with the Tensorflow [3] back-end.

After the input layer, there are four layers with 512 nodes each. The first two layers use the *sigmoid* activation function, the third layer the *relu* activation function and the fourth layer again the *sigmoid* function. *Softmax* is used as activation function for the last layer that consists of 9 nodes and is used as output layer. The architecture described above is shown in Figure 4.

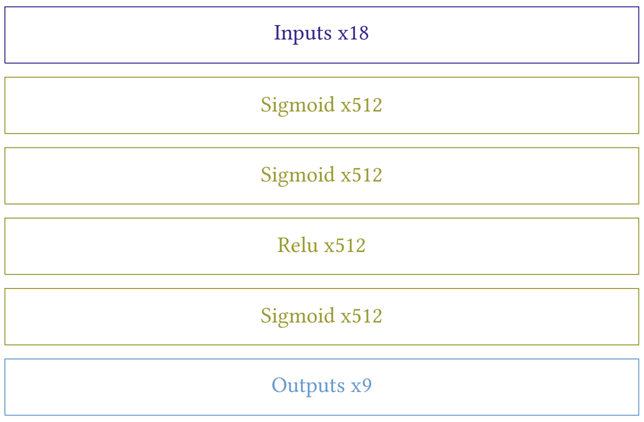


Fig. 4. Architecture of the neuronal network

In the beginning, the state was represented as a vector of size 9. A value of $-1$ represented the field was taken by player 1, a value of 1 represented the field was taken by player 2. When the value was 0, the field was empty. However, the results of that were pretty bad in a way that the network often choose actions that were invalid for the given state.

During development there was the problem that the quality of the predictions increased significantly during the first couple rounds of training. However, it strongly decreased afterwards. The root cause for the problem was that the learning rate of 0.1 was too high for the model. After some experiments, a learning rate of 0.001 seems to yield good results.

Another interesting aspect is the fact that the network is strongly depending on the opponent used for training: It learns quickly to win against the easy opponent, play a draw against the medium opponent and loose against the hard opponent (see Section 3.1 for a description of the opponents implementation). However, when the opponent changed, the network did many invalid turns before adjusting to the strategy of the new opponent. So, it did not learn how to play *Tic Tac Toe* in a valid way, but it learned directly what strategy to use against the selected opponent. This problem can be solved by frequently changing opponents: In the training phase, there are not 10 000 games against the same opponent in a row, but the opponent is changed every game.

## 3.3 Architecture of the software project

The Proof of Concept (PoC) is implemented in Python. For the GUI, the PyQT [2] framework in version 5 is used. The neuronal network is created with the Keras [1] framework using the TensorFlow [3] back-end. The PoC is attached to the submitted paper. For this reason, its structure is briefly described in the rest of this section.

I worked on a similar project during the module *Angewandtes maschinelles Lernen* of the bachelor programme. In the scope of that project, I implemented an advantage actor-critic network to learn playing *Tic Tac Toe*. I used that PoC as basis for the one submitted in the scope of this module and did several modifications and improvements to it. The architecture of the current version of the PoC as well as the most important modifications are explained in the rest of this section.

*3.3.1 Main.* The main part of the project initializes the environment and the GUI and displays the main window afterwards. It is implemented in the file `main.py`.

*3.3.2 GUI.* The GUI is mainly implemented in the file `gui.py` which contains the layout definition of the user interface. Additionally, the file `handlers.py` contains the implementation of the event handlers. Most parts of the GUI were already part of the old PoC submitted in the module *Angewandtes maschinelles Lernen*. In the scope of this module, some minor changes were done (e.g. changing the language, adding some control elements, etc.).

*3.3.3 Environment.* The entire game representation is implemented in the file `environment.py`. In that component, the state of the game is stored and modification methods are implemented. During this module, the internal handling of the state was adjusted. In addition, the state representation was modified because the old representation did not yield good results with the current network architecture. See Section 3.2 for a detailed description of the adjustments in the state representation.

*3.3.4 Agent.* The procedure of playing the game is implemented in the file `agent.py`. It handles the buffer to store the turns, selects the opponents and performs manually selected turns. During this module, among other changes, the handling of the buffer was adjusted in a way that the entire action array is stored so only the value of the action that was actually selected is adjusted afterwards and the other action values stay the same.

*3.3.5 Network.* The architecture of the neuronal network is described in Section 3.2. The network is implemented in the file `network.py`. In addition to the prediction and training features, it provides the functionality to store and load trained networks. The entire network architecture was adjusted during this project.

*3.3.6 Opponent.* The different opponents are implemented within the file `opponent.py`. See Section 3.1 for a brief description of the available opponents. In the scope of this module, the optimal and network opponents were implemented. The optimal opponents uses the Minimax algorithm (which is based on backtracking) to play only ideal games, e.g. win the game or play a draw. The network opponent uses the predictions of the neuronal network itself. However, there is no training for the network opponent implemented yet which results in a bad performance during training. See Section 6 for a more detailed description of that problem.

*3.3.7 LIME.* The adjusted version of the LIME algorithm described in Section 4 is implemented in the file `lime.py`. This implementation was created in the scope of this module. See Section 5 for a detailed description of the evaluation procedure and results.

## 4 EXPLAIN TIC TAC TOE WITH LIME

As described in Section 2.3, LIME can be used to identify parts that are relevant for the classification in images. This is done by splitting the image that should be explained into several parts and do the prediction for the parts by removing other parts. If the predicted value for the according class of a generated image that consists only of the selected parts is higher than the predicted value for the class of the entire image, the parts are classified as *relevant*. The explanation of the result is the combination of all relevant parts.

I hypothized that the algorithm described above should work for the game *Tic Tac Toe* described in Section 3.1 as well. This should be the case because the state of the game is basically equivalent to an image (e.g. an input vector with values) and the action that is predicted is equivalent to the classification for the image. If it is possible to generate *other versions* of the state, the classifier (in that case the neuronal network described in Section 3.2) should classify the other versions that were generated. Afterwards the predicted action values from the network can be used similar to the predicted classifications for images.

## 4.1 Generation of states

For that approach to work, it is required to generate *other versions* from a given state. These generated version have to be similar to the original state in order to get meaningful results. Setting fields that are empty in the original state for any player would modify the state significantly and, therefore not be useful.

The equivalent for the application case of images would be to *draw* additional parts and do a prediction based on that which is not done either. For this reason, it should be fine to just remove fields that are already set by any player. So, the *other versions* of a state can be generated by toggling the fields that are used by any player to empty fields.

Because the amount of fields is limited (it is not possible to set more than 9 fields), the state is not split into multiple regions, but each field is handled independently (in the case of an image, this would be equivalent to toggle each pixel separately). Basically, this results in $2^n$ possible states for an original state with $n$ fields set. Toggling the fields separately has the benefit that each possible subset of the state (including the empty set) is taken into account. When the opponent does its last turn (the human player does always begin), there are 7 fields set (excluding the field the opponent has set during the turn). So, in that case, there are $2^7 = 128$ generated states that have to be evaluated afterwards. In the other cases, there are less states.

## 4.2 Calculation of the influence for each field

After all states were generated, there are $2^n$ states derived from an original state with $n$ fields set. For each of these states, a prediction of the actions is done using the model (neuronal network described in Section 3.2). For each prediction, only the value on the originally predicted action is inspected (exactly one value per vector). All other values are ignored. See Figure 5 for a graphical presentation.

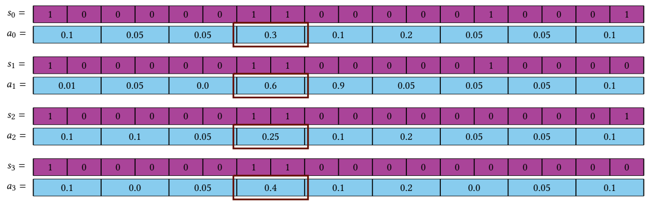

$s_0$ = 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1
$a_0$ = 0.1 0.05 0.05 0.3 0.1 0.2 0.05 0.05 0.1
$s_1$ = 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0
$a_1$ = 0.01 0.05 0.0 0.6 0.9 0.05 0.05 0.05 0.1
$s_2$ = 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1
$a_2$ = 0.1 0.1 0.05 0.25 0.1 0.2 0.05 0.05 0.1
$s_3$ = 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
$a_3$ = 0.1 0.0 0.05 0.4 0.1 0.2 0.0 0.05 0.1

Fig. 5. Predictions for different generated states

In the next step, a new vector with the dimension of the input vector (for the *Tic Tac Toe* game with a size of 18) is created and initialized with a value of 0 for all fields. This *relevance vector* stores the relevance of each input field for the final output.

For each prediction, the value of the action that was selected for the original state is added to the corresponding fields of the relevance vector. *Corresponding*, in this case, means that the value is added to each field where the state vector used for the prediction has a value of 1. In the example shown in Figure 6, it is added to the first, the 7th, the 8th, the 14th and the 18th field of the relevance vector.

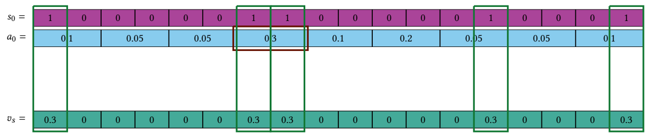

$s_0$ = 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1
$a_0$ = 0.1 0.05 0.05 0.3 0.1 0.2 0.05 0.05 0.1
$v_s$ = 0.3 0 0 0 0 0 0.3 0.3 0 0 0 0 0 0.3 0 0 0 0.3

Fig. 6. Value calculation based on given state and action vectors

That procedure is repeated for each generated state. However, there should be some additional measure that takes the *distance* between the original and the generated states into account in a way that the predictions for a state that is more similar to the original state has a higher weight and, therefore more influence to the final result. This is implemented by introducing a *weight factor* with an initial value of 1.0. For each field that differs between the original state and the generated state, the weight factor is multiplied with a value $\gamma = 0.9$. The calculated action value is multiplied with that weight factor before it is added to the values in the relevance vector.

The last step of the calculation is to reduce the relevance vector (which has a size of 18) to a vector with a size of 9 so that each value corresponds to a field from the game. Because one field can only be used by one player at maximum and there is no extension (unused fields are not used in generated states either), one field can be used by a maximum of one player. Therefore, it can have a value set either in the first nine or in the last nine values of the relevance vector. For that reason, the values can just be added (e.g. the value for the first field of player one and the value of the first field of player two) and written to the *reduced relevance vector*.

## 4.3 Normalization

The last step is to normalize the *reduced relevance vector* in order to get meaningful results. Basically, the value of all fields that are not used by any of the players in the original state have a value of 0 based on the calculation described above. In order to normalize the values, the minimum value that is not 0 is searched in the first step. Afterwards, that value is subtracted from all non-zero values. Thereby, the non-empty field with the smallest relevance is effectively set to a value of 0.

Next, the values are normalized so the sum of them is equal to a value of 1. Now, the values represent the relevance percentage of the according fields. In the version introduced during the presentation the first step of reducing the values was not included which lead to the problem that the results were in the same magnitude and often differed only slightly.

## 5 EVALUATION

In this section, the approach shown in Section 4 is evaluated in order to verify if the approach works and, thereby, the hypothesis stated earlier is correct.

As described before, the hard opponent behaves partly deterministic: When there is a turn that makes the opponent win the game, it selects that turn. When there is a turn that makes the player win the game in the next turn, it selects that turn in order to avoid the player winning. Otherwise (neither the opponent nor the player can win within one turn), it selects an empty field randomly.

If the adjusted version of the LIME algorithm works, it should be possible to generate explanations that match the implementation of the hard opponent described above: If the player creates a state where the hard opponent can win in the next turn, the hard opponent does so. If that turn is explained using the algorithm described in Section 4, the according fields of the opponent should have a higher relevance than the other fields.

It should be also possible to create a state where the player can win in the next move. In that case, the opponent should select the field that avoids the player from winning the game. Again, the algorithm should return an explanation that shows the fields of the player that were in the row, column, or diagonal that would have lead to the winning move (if the opponent would not have chosen the missing field) as more relevant than the other fields.

Because the medium opponent is basically subsets of the hard opponent (it implements only a part of its checks), there is no use in evaluating the medium opponent additionally. The easy opponent selects any random valid move independent of the state, so there is no use in evaluating the algorithm for that opponent either because it would just randomly return relevance values depending on the exact field selection of the easy opponent.

For the experiment, a game is played manually against the hard opponent. During the game, a constellation is created where one player (either the opponent or the human player) can win the game in the next round. Because the hard opponent will do its turn in order to win itself or stop the human player from winning, it will select the missing field for winning. When that turn is explained afterwards, the relevance of the other two fields is taken. See Figure 7 for an image of a game after the hard opponent (symbol "X") has selected the field in the middle column and middle row.

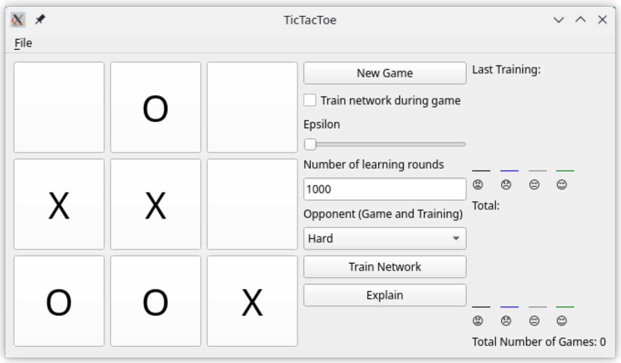
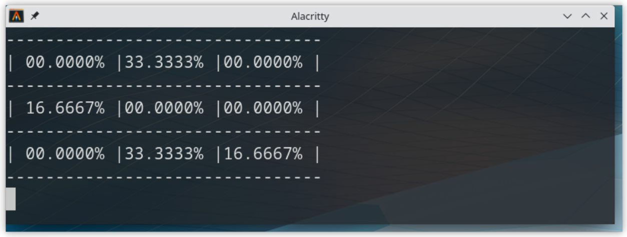


Fig. 7. State of the *Tic Tac Toe* game after the hard opponent did a turn

Figure 8 shows the explanation calculated by the modified LIME algorithm described in Section 4. The calculated relevance for the both fields in the upper and lower row and center column have a relevance value of $\frac{1}{3}$ each. The other two fields with an "X" (excluding the field that was set in the last turn because that decision is explained) have a relevance value of $\frac{1}{6}$. The field on the bottom left has a relevance value of 0. This is the case due to the calculation approach described in Section 4.3: By subtracting the minimum value from every non-zero value, the minumum value becomes zero.

In order to get a better evaluation, several games were played against the hard opponent. Table 1 shows the relevance values of the two fields that should have the highest relevance for the decision based on the knowledge about the implementation of the hard opponent. Due to the normalization, the sum of those percentage values adds up to 100 %.



```
-------------------------------
| 00.0000% |33.3333% |00.0000% |
-------------------------------
| 16.6667% |00.0000% |00.0000% |
-------------------------------
| 00.0000% |33.3333% |16.6667% |
-------------------------------
```

Fig. 8. Explanation of the *Tic Tac Toe* game after the hard opponent did a turn

| # | Winner | Field 1 | Field 2 | Sum |
|---|---|---|---|---|
| 3 | None | 66.6667 % | 33.3333 % | 100 % |
| 3 | None | 50.0000 % | 50.0000 % | 100 % |
| 7 | Opponent | 44.7368 % | 44.7368 % | 89.4736 % |
| 5 | None | 42.8571 % | 42.8571 % | 85.7142 % |
| 7 | Opponent | 39.1304 % | 36.9565 % | 76.0869 % |
| 5 | Opponent | 41.6667 % | 33.3333 % | 75 % |
| 5 | None | 33.3333 % | 33.3333 % | 66.6667 % |
| 7 | None | 25.8621 % | 24.1379 % | 50 % |
| 7 | None | 26.1905 % | 21.4286 % | 47.6191 % |
| 7 | None | 19.2308 % | 19.2308 % | 38.4616 % |

Table 1. Relevance values of the fields that are relevant for the decision of the hard opponent. The left columns displays the number of fields that were used by any player before the hard opponent did its turn. The rows are sorted by the sum in decreasing order.

In general, the sum of the relevance values of the fields decreases the more fields are used. That can be explained by the fact that there are more possible combinations in that case. Due to that, the other fields get some relevance as well during calculation. However, even when 7 fields are used, the minimum percentage of both fields was at 19.2308 % each, which was still the highest percentage in the measurement.

Another interesting fact is that the percentage of the fields is significantly higher in the cases where the opponent is winning and there are many fields: Both relevant fiels sum up to a relevance value of 89.4736 % in one and 76.0869 % in the other case where the opponent wins and 7 fields are used. In contast to that, the relevant fiels sum up to 50 %, 47.6191 %, or 38.4616 % when the opponent is not winning but avoiding the human player to win.

## 6 FUTURE WORK

During the evaluation of the adjusted LIME algorithm against the hard opponent described in Section 5, there occurred the effect that the relevance predictions were much clearer when the hard opponent won the game than when the hard opponent avoided the human player to win when many fields were used (that effect was particularly strong with 7 fields used).

A possible explanation of that behaviour could be that possible winning moves of the opponent itself are evaluated and, if possible, selected before the evaluation of winning moves of the human player. However, that situations includes the possibility of the opponent to win, so it will win the game in that case, even if it can avoid the human player to win at the same time. For this reason, that effect could lead to a decrease in the relevance value when the field is the same for both cases (e.g. the bot and the human would win when using that field).

However, that would explain a decrease in the relevance value, not an increase as measured during the experiment. Due to a lack of time, the performance of additional experiments in order to explain that behaviour has to be postponed and might be analyzed in the future.

In the current version of the PoC, there is an architectural problem with the *network* opponent: When the network plays against itself, only the current *player* is trained, not the opponent. Because the network opponent can perform invalid turns, that resulted in a situation where the opponent did an invalid turn every time. To solve that problem, the opponent was modified in a way that invalid turns are detected and automatically replaced by random valid turns. Basically, this results in the *easy* opponent being a fallback when the network opponent selects illegal turns.

That modification applies only during training, so the opponent can still perform illegal turns during a manual game. During training, the number of illegal moves that had to be adjusted are counted and shown in the console output.

In a future version of the PoC, this behaviour should be adjusted in a way that the *network* opponent is used for training as well. This could lead to an increase in the network performance because the network would learn to play "both sides" at the same time.

## 7 CONCLUSION

In this semester paper, a modified version of the LIME algorithm for explaining image classification was introduced. That version was modified in a way that it can explain turns of a bot during a *Tic Tac Toe* game based on the hypothesis that the game state is equivalent to an image and the predicted action is equivalent to the image classification.

In order to evaluate that assumption, an implementation of the game that was created in the scope of a module in the bachelor program was adjusted to match the architectural requirements of the modified version of LIME. Afterwards, that modified LIME algorithm was implemented and added to the PoC in order to evaluate it.

That explanation algorithm was evaluated afterwards using an opponent with a simple, human-understandable, behaviour in order to verify if the explanation approach is able to explain the behaviour of the algorithm. The results of the evaluation showed that the explanation matched the known algorithm of the opponent, so it is assumed that the modified version of LIME can be used for explaining *Tic Tac Toe* moves.

## REFERENCES

[1] Keras: the python deep learning api. URL https://keras.io/.
[2] Qt for python. URL https://doc.qt.io/qtforpython/.
[3] Tensorflow. URL https://www.tensorflow.org/.
[4] Jeremy Kahn. Artificial intelligence has some explaining to do. URL https://www.bloomberg.com/news/articles/2018-12-12/artificial-intelligence-has-some-explaining-to-do.
[5] Erika Puiutta and Eric M. S. P. Veith. Explainable reinforcement learning: A survey. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 77–95, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57321-8.
[6] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778. URL https://doi.org/10.1145/2939672.2939778.