Explainable Inference on Sequential Data via Memory-Tracking

Biagio La Rosa[1] , **Roberto Capobianco**[1,2*] and **Daniele Nardi**[1]
[1]Sapienza University of Rome, Italy
[2]Sony AI
biagiomattialarosa@gmail.com, {capobianco, nardi}@diag.uniroma1.it

Abstract

In this paper we present a novel mechanism to get explanations that allow to better understand network predictions when dealing with sequential data. Specifically, we adopt memory-based networks — Differential Neural Computers — to exploit their capability of storing data in memory and reusing it for inference. By tracking both the memory access at prediction time, and the information stored by the network at each step of the input sequence, we can retrieve the most relevant input steps associated to each prediction. We validate our approach (1) on a modified T-maze, which is a non-Markovian discrete control task evaluating an algorithm's ability to correlate events far apart in history, and (2) on the *Story Cloze Test*, which is a commonsense reasoning framework for evaluating story understanding that requires a system to choose the correct ending to a four-sentence story. Our results show that we are able to explain agent's decisions in (1) and to reconstruct the most relevant sentences used by the network to select the story ending in (2). Additionally, we show not only that by removing those sentences the network prediction changes, but also that the same are sufficient to reproduce the inference.

1 Introduction

Over the last decade, artificial intelligence has reached a great success due to advancements in the areas of deep learning and neural networks. While such networks achieve high performances, they are typically adopted as black box computation units, i.e. no explanation is provided on the motivation behind each decision. Due to the lack of transparent inference and decision making, deep learning cannot be easily applied in mission-critical or safety-critical domains like healthcare, where the availability of explanations is a pre-requisite for the human experts to validate predictions. Not only improving model explainability is useful to extend the applicability of current models, but also to debug and understand weaknesses of machine learning systems [Samek *et al.*, 2018].

The lack of model transparency, however, is typically exacerbated in recurrent (RNNs) or memory-augmented architectures (MANNs), since they store and elaborate multiple inputs/steps before emitting an output. Nevertheless, these architectures consistently achieve better results than standard feed-forward networks when dealing with sequential data.

In our work, we exploit the structure of MANNs to get explanations about model predictions on sequential data. They use an external memory to store and collect information for long periods, mimicking the RAM role in standard computing systems. While they have been successfully applied in multiple domains, such as visual question answering [Ma *et al.*, 2018], image classification [Cai *et al.*, 2018] and meta-learning [Santoro *et al.*, 2016], there is no work — to the best of our knowledge — focusing on methodologies to generate explanations of MANN's predictions.

Specifically, by tracking both the memory access at prediction time, and the information stored by the network at each step of the input sequence, we retrieve the most relevant parts of the input associated to each prediction. In this way, we achieve several benefits, such as: *incrementality* — the full sequence is not required, and the explanations can be obtained for each prediction; *low computational costs* — the only added costs of our method is memory tracking and processing; *flexibility* — different types of explanations can be obtained, based on how the memory is tracked and processed.

We evaluate our approach both on a modified T-maze [Bakker, 2002; Wierstra *et al.*, 2007] and on the *Story Cloze Test* [Mostafazadeh *et al.*, 2016]. In the former, our results show that we are able to explain agent's decisions. In the latter, instead, we are able to reconstruct the most relevant premises that are used by the network to select the story ending. Additionally, we show not only that by removing those premises the network prediction changes, but also that these are sufficient for the network to reproduce the inference — i.e., they are prime implicants [Shih *et al.*, 2018].

The remainder of this paper is organized as follows. First, we review existing literature (Section 2) and discuss the state-of-the-art in network explainability; then, we introduce our approach in Section 3 and present its experimental evaluation (Section 4). Finally, we discuss conclusions and future work.

*Contact Author

2 Related Work

Approaches to explainability can be categorized [Guidotti *et al.*, 2018] in two groups: *black box explanation* methods, that focus on providing a transparent model that approximates the behavior of a black box; and *black box outcome explanation* methods, that attempt to provide an outcome that locally explains the current prediction, by exploiting features of the input — i.e., without looking at how the inner model works. Our method falls in the last category, integrating the MANN architecture with a module responsible to track input portions that are used to compute the output. Hence, this section summarizes the most relevant work focusing on black box outcome explanations.

Saliency maps are especially used in computer vision, and highlight image regions that are used by the model to discriminate among classes. These can be built by using gradients [Baehrens *et al.*, 2010], by omitting subset of features [Zeiler and Fergus, 2014] or by analyzing the activation of the network units [Zhou *et al.*, 2016].

LIME [Ribeiro *et al.*, 2016] and SHAP [Lundberg and Lee, 2017], conversely, attempt to approximate models with surrogate functions that are locally faithful to the original black-box predictor, by perturbing the current input. Their approach is model-agnostic and can be used with any type of network, but suffers from high computational and time costs [Samek and Müller, 2019], as well as from strong assumptions on feature independence. Hence, these methods are not able to highlight correlation among features, which is essential when dealing with sequential data. [Ribeiro *et al.*, 2018] overcome this weakness using anchors, that are computed by perturbing the input and producing if-then rules that combine different features. While this is a step forward, realistic perturbations have to be found, worsening the time complexity problem. In all the aforementioned approaches, the whole input is required and, differently from our method, the online setting is not considered.

Another set of approaches propose to integrate the internal model structure with elements that are useful for the explanation process, therefore avoiding the time complexity and perturbation issues [Springenberg *et al.*, 2014]. For example, [Bach *et al.*, 2015; Montavon *et al.*, 2019] use a propagation-based explanation framework called "layer-wise relevance propagation" that pushes-back predictions using specific rules. Likewise, our approach integrates the MANN architecture with a module responsible to track the input portions used for computing the output.

Compared to the cited approximators that use random perturbations or rely on the tuning of several parameters that can change the provided explanations, our method is deterministic, does not involve input perturbations and does not require additional computational and time resources needed to train a new classifier or fit an external function. More importantly, our work explicitly uses the temporal dependence between features. As a result, our approach is not suited to score the relevance of a single feature of a data point, but reports on the importance of feature sequences and their correlation.

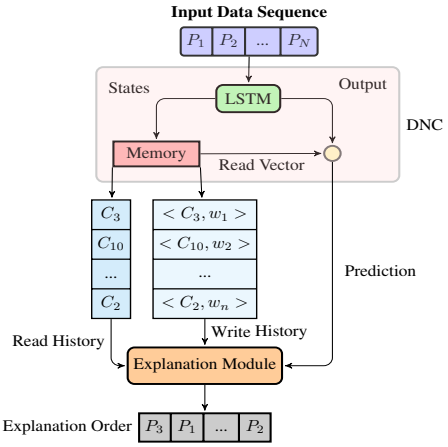

Figure 1: Sketch of the system architecture. From the memory, the read and write history are fed to the explanation module. Specifically, $< C_i, w_j >$ indicates that word w_j is stored in cell C_i. The output of the explanation module is a vector (at the bottom of the figure), containing a list of explanations sorted by their relevance.

3 Methodology

This section describes the method to achieve explainability in the context of memory-augmented networks. While we focus on Differential Neural Computers (DNCs) as a specific type of MANN, the approach could be easily extended to alternative MANNs.[1] More in detail, we first present a simplified DNC architecture, and then we introduce the explanation module composed by the memory tracking process and the explanation collection mechanism.

3.1 Simplified DNC

Differential Neural Computers (DNCs) [Graves *et al.*, 2016] have been recently introduced as a memory-augmented machine learning model based on the usage of a *controller* network. A controller is a neural network — typically recurrent with LSTM units — that (1) has read and write access to an external memory through multiple read and write heads, and (2) learns both how to perform these operations and on which data to run them.

Specifically, each read and write head is associated to some non-negative weightings that sum to one and represent a distribution over the involvement of each memory-cell in the current network operation. Both read and write weightings are the result of a combination of three attention mechanisms: content-based addressing, allocation addressing and temporal linkage addressing. Content-based addressing assigns high weightings to memory-cells with high cosine similarities computed against a key; allocation addressing is used during write operations to determine which cells can be written or erased; finally, temporal linkage addressing keeps track of the order of read/write operations.

In the original implementation, the controller can use content-based addressing to decrease memory usage, (partially) overwriting previously stored information. While this

[1] The only requirement is the ability to store all the states of previous steps and use all of them to produce the final outcome.

mechanism allows to reduce the required memory-size and to store new information about old facts, in our case it generates instability on the explanations. For example, in data sequences containing closely related information (e.g., multiple facts about the same subject in a story) the same memory-cell could be repeatedly used, due to the content-based addressing. This would make hard to both establish which part of the sequence added more information and to provide an explanation of the prediction. As a result, we disable content-based addressing during write operations and remove the possibility for the network to overwrite cells, obtaining a more stable behavior.

More in detail, considering an $N \times W$ memory matrix M_t, we compute the write weighting vector $\mathbf{w}_t^w$ at time t as:

$$\mathbf{w}_t^w = g_t^w \mathbf{a}_t \tag{1}$$

where $g_t^w \in [0, 1]$ is the write gate that controls whether to write or not in memory, and $\mathbf{a}_t$ is the allocation weighting that controls the writing location. $\mathbf{a}_t$ is defined, like in the original paper, as

$$\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]] \tag{2}$$

with ϕ_t being a sorted list of memory-cell indices in ascending order of usage. $\mathbf{u}_t$ represents the usage vector, modified only at each write, using the update rule:

$$\mathbf{u}_t = \mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w \tag{3}$$

The final output of the DNC is a linear combination of the controller output $\mathbf{o}_t^c$ and the concatenation of the R read vectors $\mathbf{r}_t$ generated by the memory as a weighted mean of its content. This is computed as

$$\mathbf{r}_t^i = M_t^\top \mathbf{w}_t^{r,i} \tag{4}$$

$$\mathbf{y}_t = \mathbf{o}_t^c + W_r[\mathbf{r}_t^1; ...; \mathbf{r}_t^R] \tag{5}$$

where $\mathbf{w}_t^{r,i}$ are the read weighting for each head.

For efficiency reasons, we further simplify the DNC architecture by removing temporal linkage. This, in fact, has a huge impact on the training time and reduces the overall network performance in our tests — probably due to long sequences with sparsely related information. The remaining parts are left unchanged from the original implementation. Note that the recurrent state of the network is both used to calculate the parameters needed to control the memory and to store the state itself.

Finally, based on the analysis from [Franke *et al.*, 2018], we apply the bypass dropout to the output of the LSTM to force an earlier memory usage during training, and a layer normalization to the input of memory to stabilize the training process.

3.2 Explanation Module

The explanation module is based on the idea of exploiting MANN architectures and tracking memory usage at prediction time. In the previous section, we mentioned that their memory contains information extracted from the states of the controller at each step, and the write and read weightings describe memory usage. More in detail, a high write weighting for a cell indicates that the same cell will contain most of the information of the current LSTM state after the write operation, while a high read weight indicates that the cell contains information similar or related to the current state — due to content-based addressing.

In this context, we can easily collect inference explanations by noticing that the encoding of a prediction should produce similar states to the ones (stored in memory) that are obtained during relevant parts of the input sequence. For example, in the Story Cloze Test — whose goal is to choose between two alternative endings given some premises — the prediction encoding should generate states that are more similar to premise states than the other alternative, which is poorly related to the premises.

In summary, our approach (Figure 1) records the content of a subset of memory-cells at each step, selecting them on the basis of the strength of the weightings. Each cell written during the input sequence and each cell read during inference is collected and linked to its recording time-step and to the input that generated the stored state. For write operations, we only track written cells whose weighting is above the mean weighting value. Tighter thresholds would not be more informative, since most of the times only one or few cells are written and most of the cells have write weightings close to zero. Conversely, for read operations we use content-based addressing and store the *top-*N_c read cells, i.e. the N_c cells with highest read weights.

Based on this, we can compute a history of memory usage and build a ranking of prediction explanations depending on the kind of information that we want to extract. Clearly, when using small N_c values, few cells are available in the history. Conversely, by setting N_c to the maximum possible value, the whole memory is included in the history. The history can be kept in memory or stored offline for later usage.

Since the state of the controller at any time t depends on (almost) all the sequence until that time-step, the explanation ranking cannot depend on a single sequence element – that does not provide sufficient information about the current prediction. Conversely, we consider longer chunks of the input sequence, and provide explanations on those. In the Story Cloze Test we build the ranking by means of the frequency of prediction readings, from which we can extract the *best* and *worst* explanation, as the sub-sequence that contains — at prediction time — respectively the most read steps and zero (or minimum) read steps.

The time complexity increase on the inference process is negligible, as it is only due to the elaboration of the memory history and depends on the information that we want to extract. In its simplest form, as shown in this paper, this elaboration only requires a summation and an access to a data structure.

While we focus mainly on prime implicants, different types of explanation can be obtained by changing how and where the information is collected from the memory. For example, in a classification problem with fixed features, information can be collected about what are the main features learned and used at training.

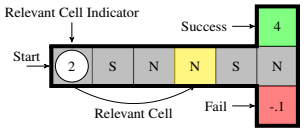

Figure 2: Example of a 5-step modified T-maze: the starting position (white) indicates the relevant cell (yellow). This cell contains the symbol *N*, indicating that the agent gets a positive reward going North in the T-junction, and a negative one going South.

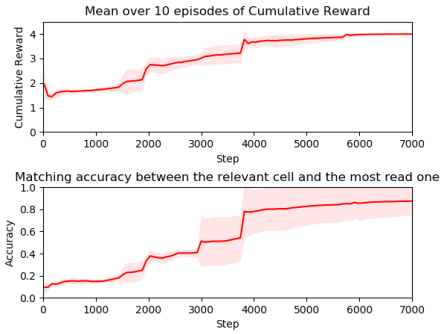

Figure 3: (Top) mean and std of the cumulative reward obtained over 10 different training runs. (Bottom) matching accuracy between the relevant corridor step and the 2 most-read memory cells.

At the current stage, our explainability module is particularly well suited for tasks where background knowledge is first fed to the network, and the model is subsequently required to answer specific queries — e.g., question-answering tasks. However, in different domains — such as vision and robotics — our approach could be easily adapted by modifying the addressing mechanism from content-based addressing to temporal linkage.

4 Experiments and Results

This section first describes the experimental setups, and then it presents and analyzes the obtained results.

4.1 Experimental Setup

As previously mentioned, we validate our approach both on a modified version of the T-maze task [Bakker, 2002; Wierstra *et al.*, 2007] and on the Story Cloze Test [Mostafazadeh *et al.*, 2016]. While we use the former as a toy-problem to validate our idea, we run a broader range of experiments on the latter to collect further insights on the proposed approach. Here, we individually describe both tasks together with their experimental setup, the training details and the obtained results.

Modified T-maze

Task. T-maze is a non-Markovian discrete control task that evaluates an algorithm's ability to learn to remember observations and correlate events far apart in history. An agent has four possible actions (move North, East, South, or West) and must learn to move from the starting position (at the beginning of the corridor) to the T-junction, where it must move either North or South to a changing goal position, which depends on the symbol that the agent has seen at the starting position. If the agent takes the correct action, it receives a reward of 4, otherwise -0.1. In both cases, the episode terminates and a new episode is started. In our tests, we modify this task to also evaluate the agent's capability of explaining his decision, adding a symbol observable by the agent at every step of the corridor, only one of which is relevant, and by specifying the relevant corridor step in the starting position. Figure 2 shows a simplified example of the modified version: the relevant step is at position 2, which contains the symbol N and tells the agent that the right decision at the T-junction is to move North. In this way, not only the agent can learn to remember previously seen observations for a long number of steps, but it also can justify its decisions by indicating the read symbol and the corridor cell. The main questions that we aim at answering in this domain are whether we can successfully solve the partially observable Markov Decision Process with our simplified DNC architecture, and whether our explanation module works as expected.

Training details. In our experiments, we choose a fixed corridor length of $N = 25$ steps, and we compute the relevant cell indicator $I \in [0, 1]$ as n/N, i.e., by normalizing the observed number n by the corridor length. The symbol observed by the agent in each cell is finally represented as a discrete value in the set $\{0, 1\}$, while actions are represented as a one-hot vector. The controller of the DNC is a single LSTM layer composed by 128 units, a memory of size 50, 1 read head, 1 write head, and a rate of 0.2 for the bypass dropout. We train our network running the Deep Recurrent Q-Learning algorithm [Hausknecht and Stone, 2015] with Tensorflow 2.0 on 1500 episodes using RMSProp with a learning rate of $1e^{-3}$ and a gradient clipping of 20. Finally, we test the learned model on 500 episodes.

Results. Figure 3 (top) presents the average cumulative reward of the agent over 10 different training runs. Our results show that the model gradually converges to a solution in which the agent can consistently achieve the highest reward. Hence, we can conclude that our architecture can successfully solve the problem. Additionally, Figure 3 (bottom) shows the accuracy of the explanation system (matching accuracy), obtained by checking whether the information from the relevant corridor step is contained in the two memory cells that are the most-read when the agent reaches the T-junction. We consider two cells instead of one to also account for the first corridor step, that could be used equally often. The plot indicates that, most of the times, the agent justifies its decisions with the observation contained in the relevant corridor step, confirming the expected behavior for the explanation module. Finally, we observe that the cumulative reward and the matching accuracy curves follow the same behavior. This suggests that the agent improves its performance when it learns to exploit the relevant step information stored in the memory. Note that the matching accuracy is influenced both by the high variability of the learning process in MANNs [Franke *et al.*, 2018] and by the fact that the memory output can be sometimes ignored, in favor of the LSTM output. This is highlighted by the variance — although it remains reliable (accuracy ≥ 0.6) also in the worst-case scenarios.

Story Cloze Test

Task. The Story Cloze Test, is a commonsense reasoning framework for evaluating story understanding. In this task, an agent has to choose between two possible endings for a set of stories composed by four premises each. Due to its particular input structure, it has been proven that it is possible to reach good performances on this task by just using one of the story premises [Srinivasan *et al.*, 2018].

Training details. In our tests, we tokenize each sentence with Stanford CoreNLP [Manning *et al.*, 2014] and we encode each word in a 300-dimensional space using word2vec [Mikolov *et al.*, 2013]. A zero-vector of the same dimension is appended to the end of each input to mark the end of the sequence and to request an answer from the network. Moreover, we add three boolean flags to each word, indicating whether the current word is taken from the story, the ending, or the query respectively. The model is trained and tested on the official development and test dataset of the challenge, while we randomly split the former into a training and development set containing respectively 90% and 10% of the original dataset. We encode the premises using the DNC and the final story state is used as the initial state for the encoding of both endings [Mihaylov and Frank, 2017], which are concatenated and passed to the last network layer with a softmax activation. The controller of the DNC consists of a single LSTM layer with 128 units, a memory of size 512, 4 read heads, 1 write head, and a bypass dropout rate of 0.2. We finally train the network with Tensorflow 2.0 using the Adam optimizer with a learning rate of $1e^{-4}$ and a gradient clipping of 20.

Task baseline comparison. As a first test, we evaluate the accuracy of our architecture on the ending prediction task (Table 1). The goal of this test is to show that the proposed architecture does not under-generalize with respect to existing baselines. Specifically, while [Mihaylov and Frank, 2017] is not the highest-scoring algorithm in the challenge leaderboard[2], it represents a good baseline because (1) it is among the top-performing published works, (2) it does not use task-dependent features, and (3) it can be easily adapted to other sequencing tasks. Our results show that our method and [Mihaylov and Frank, 2017] obtain the same score on the test set and, consequently, they achieve the same generalization capabilities. To confirm that these results mostly depend on the DNC usage of the memory content, we set the LSTM output to zero and check that the overall accuracy does not drop too much (68.8%). This confirms the importance of the memory content, and its relevance for our explainability purposes.

LIME. We compare our method against explanations obtained using LIME [Ribeiro *et al.*, 2016][3], when applied on the same network architecture. LIME is an additive feature attribution method that assigns to each input feature an importance score. While this is effective in several problems, it does not explicitly support sequences, which have to be provided all at once. Moreover, a basic assumption of LIME is the independence of features, which does not hold for sequential problems. In Figure 4, we report example results produced by LIME, where we highlight in blue and orange the words that shift the prediction towards the first or the second ending respectively, if removed. Our results show that, across all the examples in the dataset (processed at an avg. speed of 15 minutes each), LIME focuses its attention on the last part of the input (i.e., the concatenation of the premises and the answers), reporting on which part of the answers is more important. While it is true that changing the words of the answers leads to a different prediction, we are interested in knowing which premise or sequence of words in the story leads the model to output its prediction. In theory, the weights of the words in each premise could be summed. Our experiments, however, show that most of the times their weight is close to zero. Our method, conversely, exploits the sequential nature of the sentences, and checks which sub-sequence is more important. Figure 5 shows few instances of explanations obtained via our algorithm — including the ones in Figure 4 for direct comparison. Note that our algorithm only takes few seconds to produce an explanation.

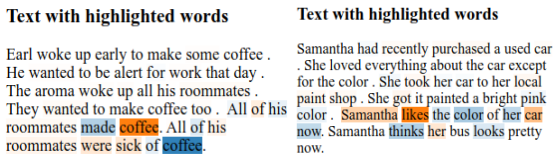

Figure 4: Example output of LIME, where each word in the sentence is weighted based on its relevance. Blue words, if removed, decrease the probability of choosing the first ending; orange words, if removed, decrease the probability of choosing the second ending.

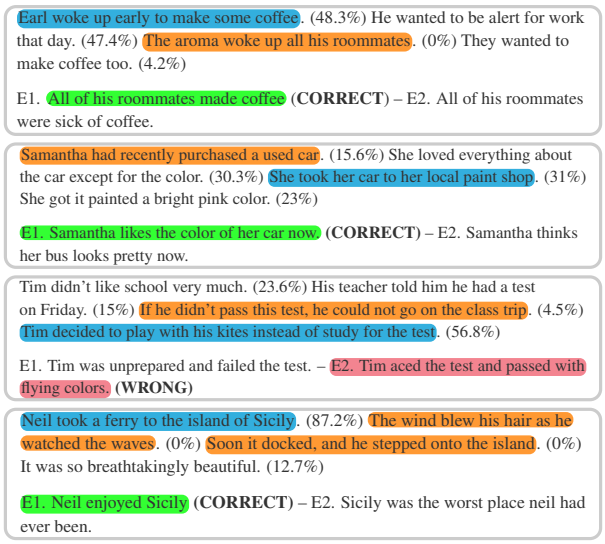

Figure 5: Example outputs on the Story Cloze Test. A relevance score is associated to each premise — ranked from best (blue) to worst (orange). Predictions are highlighted in green if correct, or red if wrong.

[2]https://competitions.codalab.org/competitions/15333\#results

[3]Note that SHAP and LIME feature similar issues.

Avg. Accuracy		
Architecture	Dev	Test
[Mihaylov and Frank, 2017]	77.12%	72.10%
DNC	71.30%	72.10%

Table 1: Avg. accuracy comparison against task baseline. For [Mihaylov and Frank, 2017], we report the scores from the original paper, while we compute our average accuracy over 10 different runs.

Avg. Explanation Accuracy			
Premise	Train	Dev	Test
Random	57.5%	60.0%	55.6%
Best	66.0%	73.3%	63.8%
Worst	51.3%	53.3%	53.6%

Table 2: Avg. explanation accuracy over 10 runs on the Story Cloze Test when feeding the model with only a random, the best and the worst premise.

Explanation accuracy. In this test, we evaluate the explanation accuracy for the Story Cloze Test[4]. As mentioned, our method generates an output similar to Figure 5, allowing us to identify the best and worst scoring explanations. Based on this, and supposing that a good explanation is sufficient to justify a certain prediction, we compute the accuracy of the model in reproducing its predictions by only using the best and worst premises — in place of the full story. Intuitively, if the explanation for a certain prediction is good, feeding that information alone to the network should result in the same prediction (and a good accuracy). Conversely, by only providing the worst explanation to the model, the prediction could change due to the missing relevant information, leading to a low explanation accuracy. The results presented in Table 2, which contain the explanation accuracy obtained when using the best, the worst and a random premise (acting as a control variable), confirm our hypothesis. Hence, the explanations produced by our algorithm can be considered prime implicants [Shih *et al.*, 2018] — extended to the sequential setting. The reported results are obtained by building the read history with the 10 most read cells at every step. Here, the worst premise can achieve a lower explanation accuracy than a random premise due to the structure of the dataset, that often contains a misleading or poorly relevant premise in the story. Figure 6 shows the evolution of the explanation accuracy for the best and the worst premises over 10 runs, confirming the results of Table 2. These plots additionally indicate that the explanations provided by our system are meaningful from the very first training steps and, consequently, the network quickly learns to use its memory (independently of the final prediction).

[4]Evaluating explanations is an open problem [Samek and Müller, 2019], and there is no metric that can be used in all cases.

Dev Set Avg. Explanation Accuracy vs Thresholds					
	Threshold				
Premise	Top1	Top5	Top10	Top25	>Median
Best	**75.6%**	**75.6%**	73.3%	71.1%	40.0%
Worst	60.0%	60.0%	53.3%	**40.0%**	48.9%

Table 3: Avg. explanation accuracy on the dev set over 10 runs for the best and worst premises using different N_c values. We highlight in bold the best and worst accuracy respectively, for each premise type among the different threshold values.

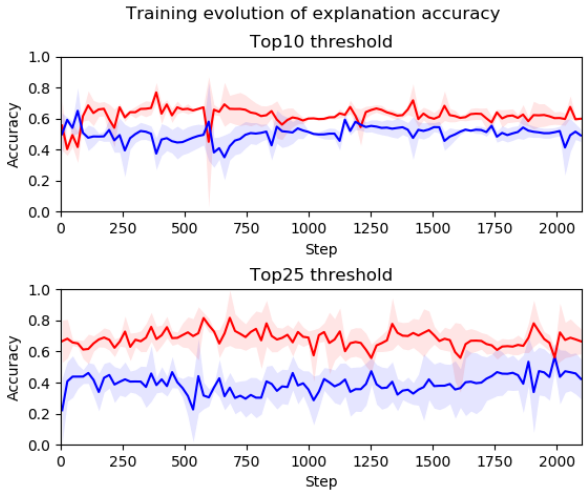

Figure 6: Explanation accuracy (mean and std) over training. Metrics are computed on the training set over 10 runs when using only the best (red line) and the worst (blue line) premise as input. Explanations are computed using a read history generated from the 10 (top) and 25 (bottom) most read cells.

Explanation accuracy vs history size. We compare the explanation accuracy over multiple values of N_c, to evaluate the relation between the explanation score, the obtained explanation accuracy and the number of most-read cells considered in the history. Results are reported in Table 3, where the notation the *top*N_c indicates the used N_c value. Specifically, we consider $N_c \in \{1, 5, 10, 25\}$ in addition to the above-median case. Our results show that the explanation accuracy of both the best and the worst premises follows the same behavior: the smaller the value of N_c, the higher the explanation accuracy; the larger the threshold, the lower the accuracy. This behavior can be explained by analyzing the information in the read history. For example, when $N_c = 1$ (*top1* threshold) a single cell for each step is available in the history, which contains — at the end of the sequence — only T read cells, where T is the number of steps (i.e., words). T is generally small, and commonly stored cells only refer to one or two premises. Moreover, since these cells are those with the highest weight, they are highly represented in the read vector returned by the memory, and they have a direct influence on the final predic-

tion. This allows to conclude that these cells contain the best explanation for the current prediction. The worst explanation scenario is more tricky: as described above, when using small N_c values, many premises do not have read cells in the history, and the explanation module has to (almost) randomly choose the worst premise. This lack of history information results in a large probability of picking a sufficiently good explanation instead of the worst one. Conversely, when we set N_c to a large value (e.g., $N_c = 25$), the generated history contains several cells. If no step of a certain sequence is contained in the history, the sequence itself is not represented in the read vector and it cannot be used in the final prediction. This allows us to identify the worst premise, and to significantly reduce the matching accuracy when feeding it to the network, as expected. The choice of the best explanation, instead, is based on a larger number of cells, resulting in an increased selection noise. In fact, while a small N_c value rewards premises that are read more constantly during inference, a larger threshold only rewards premises with the most number of readings — independently of when the read occurred. Hence, premises contained in cells that are read multiple times only at the beginning are ranked higher than premises read fewer times but more constantly, lowering the accuracy. Starting from these observations, we conclude that N_c must be set depending on whether we are interested to the most useful explanation or to a more informative analysis.

Explanation adequacy. Despite the lack of universal metrics, we study the adequacy of explanations by adapting the tests proposed in [Adebayo *et al.*, 2018]. First, in the model parameter randomization test we check the explanation accuracy of the best and worst premises, when the model is untrained and the weights are randomly initialized. As expected the test accuracy of the model (Table 4) is equivalent to random guessing. Moreover, the explanation accuracy of best and worst premise is almost the same, so the method cannot differentiate between a good and a bad premise. This confirms that the proposed method depends on the learned parameters, and in particular on how the model learns to use the memory. In the data randomization test, we shuffle the labels in the training and dev set and train a model on the modified dataset. As proposed in [Adebayo *et al.*, 2018], we stop the training when the training accuracy is above 95%, and we check the explanation provided for the original test set. Table 4 reports the obtained results, where accuracy is higher then random guessing because the model can learn useful information from several examples in dataset with the right label. In this case, the role of best and worst premises is flipped, and their accuracy is worse than before. The flipping behavior can be explained as an attempt of the model to exploit the changed label: in this case to predict the correct answer it should choose the answer that has the minimum similarity with the memory output. In fact, since the task is overabundant, this behavior can be rewarded also by some of the correctly labeled examples.

Avg. Explanation Accuracy			
Test	Accuracy	Best	Worst
Parameter Randomization	50.8%	56.0%	55.4%
Data Randomization	58.6%	51.8%	62.8%

Table 4: Adequacy tests adapted from [Adebayo *et al.*, 2018].

5 Conclusions

Nowadays, several researchers are focusing on explainable AI and machine learning, with the purpose of increasing trust from the general public, and allowing a broader usage where expert supervision is mandatory. This paper investigates how memory-augmented neural networks can be used to produce explanations at prediction time.

Our approach represents an unexplored and novel way to collect information on the inference process. Yet, our results confirm that by keeping track of the memory usage we can produce reliable explanations on two different tasks. We analyze, for these tasks, the relation between the amount of tracked memory and the explanation goodness, as well as the gap between the best and the worst explanations. These promising results make the proposed approach definitely worth additional investigation. In particular, we aim at (1) further improving the quality of provided explanations, (2) better understanding memory-based explanation mechanisms, and (3) extending the applicability of our approach to different domains in future work. In particular, the latter will allow us to compare our approach with the current state of art approaches on non-sequential data.

References

[Adebayo *et al.*, 2018] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proc. of the 32nd Int. Conf. on Neural Information Processing Systems*, page 9525–9536, Red Hook, NY, USA, 2018.

[Bach *et al.*, 2015] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):e0130140, 2015.

[Baehrens *et al.*, 2010] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, 2010.

[Bakker, 2002] Bram Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems*, pages 1475–1482, 2002.

[Cai *et al.*, 2018] Qi Cai, Yingwei Pan, Ting Yao, Chenggang Yan, and Tao Mei. Memory matching networks for one-shot image recognition. In *2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2018.

[Franke *et al.*, 2018] Jörg Franke, Jan Niehues, and Alex Waibel. Robust and scalable differentiable neural computer for question answering. In *Proc. of the Workshop on*

Machine Reading for Question Answering, pages 47–59, 2018.

[Graves *et al.*, 2016] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

[Guidotti *et al.*, 2018] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42, 2018.

[Hausknecht and Stone, 2015] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[Lundberg and Lee, 2017] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[Ma *et al.*, 2018] Chao Ma, Chunhua Shen, Anthony Dick, Qi Wu, Peng Wang, Anton van den Hengel, and Ian Reid. Visual question answering with memory-augmented networks. In *2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2018.

[Manning *et al.*, 2014] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proc. of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[Mihaylov and Frank, 2017] Todor Mihaylov and Anette Frank. Story cloze ending selection baselines and data examination. *LSDSem 2017*, page 87, 2017.

[Mikolov *et al.*, 2013] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proc. of the 2013 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[Montavon *et al.*, 2019] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: An overview. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 193–209. Springer International Publishing, 2019.

[Mostafazadeh *et al.*, 2016] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proc. of the 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, 2016.

[Ribeiro *et al.*, 2016] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proc. of the 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2016.

[Ribeiro *et al.*, 2018] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conf. on Artificial Intelligence*, 2018.

[Samek and Müller, 2019] Wojciech Samek and Klaus-Robert Müller. Towards explainable artificial intelligence. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 5–22. Springer International Publishing, 2019.

[Samek *et al.*, 2018] W. Samek, T. Wiegand, and K. Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *ITU Journal: ICT Discoveries - Special Issue 1 - The Impact of Artificial Intelligence (AI) on Communication Networks and Services*, 1(1):39–48, 2018.

[Santoro *et al.*, 2016] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, 2016.

[Shih *et al.*, 2018] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *Proc. of the Twenty-Seventh International Joint Conf. on Artificial Intelligence*, 2018.

[Springenberg *et al.*, 2014] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[Srinivasan *et al.*, 2018] S. Srinivasan, R. Arora, and M. Riedl. A simple and effective approach to the story cloze test. In *Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 2, pages 92–96, 2018.

[Wierstra *et al.*, 2007] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *Int. Conf. on Artificial Neural Networks*, pages 697–706. Springer, 2007.

[Zeiler and Fergus, 2014] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conf. on Computer Vision - ECCV 2014*, pages 818–833. Springer International Publishing, 2014.

[Zhou *et al.*, 2016] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conf. on Computer Vision and Pattern Recognition*, 2016.