Received 17 September 2023, accepted 28 September 2023, date of publication 5 October 2023,
date of current version 18 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3322367

RESEARCH ARTICLE

Developing an Algorithm for Fast Performance Estimation of Recurrent Memory Cells

SERGIU COSMIN NISTOR[1], **MOHAMMAD JARADAT**[2], **AND RĂZVAN LIVIU NISTOR**[3]
[1]Department of Computer Science, Babeş-Bolyai University of Cluj-Napoca, 400084 Cluj-Napoca, Romania
[2]Department of Management, Bogdan Vodă University of Cluj-Napoca, 400285 Cluj-Napoca, Romania
[3]Department of Management, Babeş-Bolyai University of Cluj-Napoca, 400084 Cluj-Napoca, Romania

Corresponding author: Răzvan Liviu Nistor (razvan.nistor@ubbcluj.ro)

ABSTRACT We propose a novel graph-oriented machine learning algorithm which we use for estimating the performance of a recurrent memory cell on a given task. Recurrent neural networks have been successfully used for solving numerous tasks and usually, for each new problem, generic architectures are used. Adapting the architecture could provide superior results, but would be time-consuming if it would not be automated. Neural architecture search algorithms aim at optimizing the architectures for each specific task, but without a fast performance estimation strategy it is difficult to discover high-quality architectures, as evaluating each candidate takes a long period of time. As a case study, we selected the task of sentiment analysis on tweets. Analyzing the sentiments expressed in posts on social networks offers important insights into what are the opinions on different topics and this has applications in numerous domains. We present the architecture of the estimation algorithm, discussing each component. Using this algorithm, we were able to evaluate one million recurrent memory cell architectures and we discovered novel designs that obtain good performances on sentiment analysis. We describe the discovered design that obtains the best performances. We also describe the methodology that we designed, such that it can be applied to other tasks.

INDEX TERMS Neural architecture search, graph neural network, performance estimation strategy, recurrent neural network, sentiment analysis, tweet.

I. INTRODUCTION

Recently, deep learning approaches were used to obtain state-of-the-art performance on many natural language processing tasks [1], [2], [3]. Recurrent neural networks (RNNs) are able to take advantage of the sequential structure of the text and extract information from it [4], [5].

Modern recurrent neural networks use recurrent memory cells in their architecture [5]. Even though good results were obtained using these standard cells, there is an increasing interest in finding new designs in an automatic manner [6]. Algorithms for finding new architectures already found designs that obtain high performances on a few tasks, but these search algorithms are slow as they need to evaluate the newly proposed architectures.

In this paper, we propose a novel algorithm for estimating the performance of a recurrent memory cell on a given task. As the cells have graph structures, our algorithm is adapted to this requirement. This novel performance estimation strategy receives as input the description of the recurrent neural network and predicts its performance on a given task.

To experimentally prove the efficiency of our algorithm, we selected a task as a case study: sentiment analysis on tweets. There is much information to be gained from extracting sentiment from text, especially when there is a large quantity of available data [7]. While it is interesting to process individual sentiments, by processing large quantities and aggregating them, one can find the general opinion towards an idea. A challenging task is to design an algorithm able to accurately recognize the sentiment in a piece of text. Due to the great applicability of this problem, many solutions were proposed [8], [9].

Besides the high applicability of sentiment analysis on tweets, this is also an appropriate and representative task to be solved using recurrent neural networks. This task is part of the larger domain of natural language processing, where

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Asif.

© 2023 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

recurrent networks were used with great success. Tweets, being mainly pieces of text, can be viewed as a sequence of characters that can be analyzed by RNNs.

We apply our performance estimation algorithm on this task end evaluate its ability to learn the quality of each cell. After we have trained models, we use them to discover new cells, specifically designed for this task. We present a new cell design that we discovered using our proposed algorithm.

As our work is at the intersection of neural architecture search (NAS), graph-oriented machine learning algorithms and sentiment analysis, the main contributions of this paper are the following:

1) Propose an algorithm for processing data structured as graphs, which we call Graph-Encoding Recurrent Neural Network (GERNN);
2) Propose a performance estimation method for neural architecture search algorithms used for recurrent neural network cells;
3) Discover a novel recurrent memory cell designed for sentiment analysis on tweets.

We present work related to ours in Section II. In Section III, we describe our algorithm and how to apply it on the task that we wish to solve. The experiments and the results are presented in Section IV and our conclusions are drawn in Section V.

II. RELATED WORK

In this section we present works from the scientific literature that are related to ours and that influenced the development of our proposed solution. We start by describing sentiment analysis on tweets, as this is the task that we used as a case study. Our new performance estimation algorithm is aimed at improving neural architecture search algorithms, so we present developments in this domain with a special emphasis on the performance estimation strategies.

As many people express their opinions in posts on Twitter, this platform has become a great source of information and many researchers conducted studies on how this useful information can be extracted [8], [10], [11], [12]. For recognizing the sentiment from the tweets, classical approaches used hand-crafted features like the presence of n-grams [13], [14], [15], the presence of Twitter-specific elements like emoticons, hashtags or abbreviations [16], lexicons with tagged emotional polarity [17]. These features were then classified using Naïve Bayes classifiers [14], [15], support vector machines [13], [14], decision trees [18].

As deep learning emerged, convolutional neural networks (CNNs) and recurrent neural networks were used for recognizing sentiment from tweets [19], [20], [21], [22].

As recurrent neural networks are machine learning algorithms which can model sequences [4], they are a natural choice for extracting information from text, which can be considered a sequence of words or characters. Modern RNNs use memory cells as their building blocks. Even though many problems were approached using RNNs, most of them integrate general-purpose cells such as the Long Short-Term Memory (LSTM) cell [23] or the more recent Gated Recurrent Unit (GRU) [24].

While these general-purpose RNN cells have obtained good results on a variety of tasks, one can argue that adapting the architecture to the problem should produce superior results, and there is great interest in proposing new designs [6], [25], [26]. As manually finding the optimal architecture for a given problem can prove challenging and a small variety of architectures are explored, NAS appeared from the necessity of automatically finding new designs [27]. Such algorithms were designed for finding new feed-forward neural network (FFNN) architectures [28], [29], [30], [31], CNN architectures [32], [33], [34], [35], [36] and RNN cells [37], [38], [39], [40], [41].

There is an increasing interest in NAS. Zoph and Le [37] used reinforcement learning for searching for new CNN and RNN architectures. For the RNNs, they searched for new cells which were represented as binary trees with a fixed number of nodes. Each node took input from two other nodes, applied operations on them and combined them. The operations were decided by an RNN which described these new cells. Additionally, the RNN also decided which node produces the internal state and where is the internal state used in the cell. This generator RNN was trained using reinforcement learning, encouraging the generator to propose cells that have increasing quality. For performance estimation, the cells are included in a complete RNN which is trained on a language modelling task for a fixed number of 35 epochs, which, of course, is time consuming.

Bayer et al. [38] proposed an evolutionary algorithm for creating new RNN cells. Cells were represented as directed graphs in which the nodes represented computations and the edges represented the data flow. Each individual of the population used by the evolutionary algorithm represented a candidate cell and consisted of a description of such a cell. At each generation, the individuals were mutated by changing the characteristics of the nodes and edges, adding or removing nodes and edges. No crossover operator was used. For performance estimation, full RNNs were built and trained to learn simple context-free or context-sensitive languages.

As the computational resources necessary for evaluating a large number of architectures is demanding, fast performance estimation becomes an active research topic. Kyriakides and Margaritis [42] also observed that the time required to evaluate an architecture is important in the design of a NAS algorithm. They evaluated the usage of early stopping, which is one of the most popular performance estimation strategies, where the number of training epochs is fixed to a small number, which usually does not allow convergence, but obtains the estimation faster than full training. Kyriakides and Margaritis [42] observed that for CNNs there is a strong correlation between the rankings of CNN architectures after a small number of epochs and the rankings obtained after a large number of epochs, showing that the information loss is small. Even so, training takes a considerable time, even for a small number of epochs.

In [43], a performance estimation algorithm for CNNs is proposed. The input features are characteristics of the network, like the number of layers, or hyperparameters, like the learning rate. The features are processed by random forests or support vector machine regressions.

A genetic programming algorithm is used by Rawal and Miikkulainen [40] to propose new RNN cells. These cells are represented as trees. The mutation operator changes the nodes and adds or removes branches, while the crossover operator combines subsections of these trees. The individuals are also grouped using a distance metric such that when selecting the parents of a new individual, the crossover would be more efficient. The problem that is studied is language modelling. As training RNNs on this task takes time, the authors decided to use machine learning to predict the performance of the cells. After training the candidate RNN for 10 epochs, the obtained metrics are used as input to an algorithm that predicts the metrics after the 40th epoch of training. This algorithm is an RNN which uses LSTMs as hidden units. Even though this approach obtained good results, it only takes into account the behaviour of the RNN when making the prediction. We consider that taking into account the structure of the RNN would provide superior results.

The NAS algorithm proposed in [36] integrates a ''surrogate model'' which is an algorithm that estimates the performance of a CNN. The evaluated architectures are based on building blocks called cells, and each cell is a sequence of nodes. An RNN predicts the performance of the cell by processing the nodes in the fixed order in which they describe the cell. This approach obtained good results, but it is insufficient for our task. As we evaluate recurrent memory cells, they are not simply sequences of nodes, but graphs with edges and the nodes do not have a fixed order. An algorithm which takes into consideration the graph structure and the various properties of the nodes and edges is necessary.

Our novel algorithm incorporates multiple RNNs and attention mechanisms. The utility of attention mechanisms attached to RNNs was previously proven [44]. Zheng et al. [44] presented a theoretical analysis of LSTM with attention mechanism and experimentally proved that the mechanism helps the LSTM have longer-term memory by having a slower memory decay.

Our proposed algorithm for performance estimation is designed for processing graphs. As many problems can be modeled as graphs [45], [46], machine learning algorithms designed for processing them started to emerge. While classical convolutional and recurrent neural networks process data structured as matrices and sequences, respectively, the need for processing data structured as graphs led to the proposal of new algorithms [46], [47], [48].

III. PROPOSED SOLUTION

In this section we present our proposed solution for fast evaluation of the quality of recurrent memory cells. We describe how our performance estimation algorithm can be applied to RNNs used for the task of sentiment analysis on tweets. While our solution was applied on this task, our algorithm is designed to process graphs, with very few specific consideration for the particularities of the information represented by the graph. Due to this reason, our algorithm can process generic graphs, as long as the graph preprocessing is appropriate and the hyperparameteres are tuned. In the following, we describe the mechanism of our algorithm on generic graphs and then the particular preprocessing that we used for our task.

A. GRAPH-ENCODING RECURRENT NEURAL NETWORK

Our Graph-Encoding Recurrent Neural Network takes as input graphs, which are composed of a set of nodes and a set of edges between the nodes. Each node may have any number of properties, but all nodes must have the same set of properties. Similarly, edges may have their own set of properties. Graphs may be directed or undirected. The output of our GERNN depends on the problem to be solved, as our algorithm can be used for both classification and regression problems. We present in Fig. 1 the graphical representation of our algorithm and we detail each component in this subsection. The arrows in this figure represent the flow of data. $N_1, N_2, \ldots, N_n$ are the n nodes of the input graph and $E_1, E_2, \ldots, E_m$ are the m edges.

Our GERNN is an algorithm composed of RNNs and attention mechanisms and due to this, the inputs must have numerical representations. The *Node Preprocessor* and *Edge Preprocessor* have the role of converting the graph into the numerical representation. Each node and each edge is converted into a feature vector which describes the properties associated to the node / edge. The numerical representation of the graph is a list of node-vectors and edge-vectors. This sequential representation is also imposed by the RNNs composing the algorithm, as RNNs are designed for sequence processing [4].

After the preprocessing, the nodes are the first to be processed by a recurrent neural network, the *Node RNN*. In Fig. 1, the green arrows represent the passing of the state of an RNN. The computation done for each node is described in (1), where NRO_i is the output of the Node RNN for node i, NV_i is the representation of this node (which was obtained using the Node Preprocessor), NRS_{i-1} is the state of the Node RNN after processing the previous node and NodeRNN is the function modeled by the RNN dedicated to processing nodes. In the equations presented in this section, we assume to have a graph with n nodes and m edges.

$$\mathrm{NRO}_i = \mathrm{NodeRNN}(\mathrm{NV}_i, \mathrm{NRS}_{i-1}) \tag{1}$$

The edge processing follows, using the *Edge RNN*. This is done after the node processing, as the Node RNN outputs are taken into consideration. The edge-vectors are taken as input one-by-one, but an attention mechanism helps the Edge RNN take into consideration not only the edges, but also the nodes. The *Node-Edge Attention* module takes as input for each edge

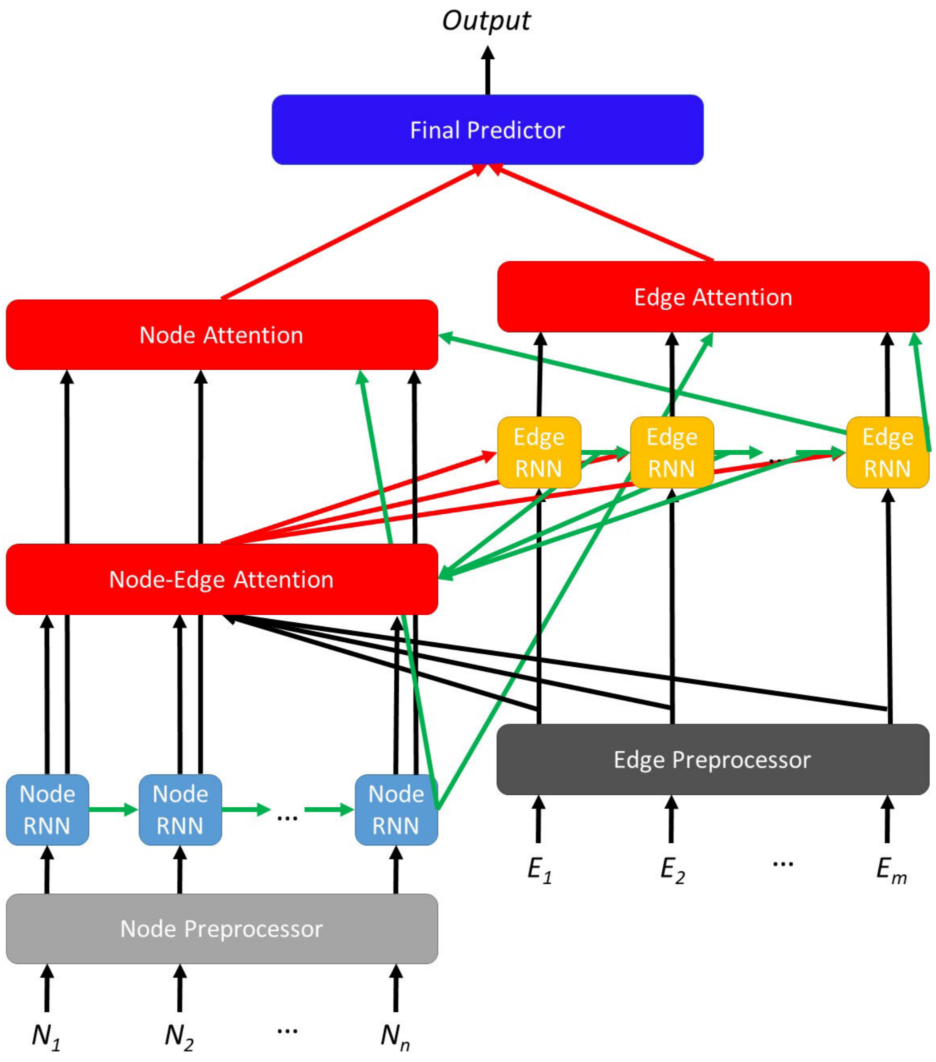


FIGURE 1. **Graphical representation of the GERNN.**

the edge-vector, the state of the Edge RNN and the Node RNN outputs for all nodes. This attention module produces scores for each node and aggregates the outputs into context vectors according to these scores. The aggregated node-output is given as input to the Edge RNN. This mechanism helps the GERNN focus for each edge on the most relevant nodes, as different nodes have different relevance with respect to a given edge. The edge-vector is useful for describing the processed edge to the attention mechanism, the node-outputs describe the already processed nodes and the state of the Edge RNN helps the mechanism have a global understanding of the edges that were already processed.

We present in (2) - (4) the computation of the Edge RNN outputs for each edge. $\mathrm{NEAtt}_{j,i}$ is the score that is associated to each node i for the processing of edge j. This score is based on the state of the Edge RNN after processing the previous edge (ERS_{j-1}), the edge representation obtained using the Edge Preprocessor (EV_j) and the output of the Node RNN for node i (NRO_i). NodeEdgeAttention is the function modeled by the Node-Edge attention component of the GERNN. After this, an aggregated representation of all nodes (AggNE_j) is computed based on the attention scores, as described in (3). This aggregate node representation together with the edge representation (EV_j) and the Edge RNN state after processing the previous edge (ERS_{j-1}) are used to produce the Edge RNN output for the current edge (ERO_j). EdgeRNN denotes the function modeled by the RNN component of GERNN that is dedicated to processing the edges of the graph.

$$\mathrm{NEAtt}_{j,i} = \mathrm{NodeEdgeAttention}(\mathrm{EV}_j, \mathrm{ERS}_{j-1}, \mathrm{NRO}_i) \quad (2)$$

$$\mathrm{AggNE}_j = \sum_{i=1}^{n}(\mathrm{NEAtt}_{j,i} \cdot \mathrm{NRO}_i) \quad (3)$$

$$\mathrm{ERO}_j = \mathrm{EdgeRNN}(\mathrm{EV}_j, \mathrm{ERS}_{j-1}, \mathrm{AggNE}_j) \quad (4)$$

Before making the final prediction, the *Node Attention* (denoted by NodeAttention in the equations) and *Edge Attention* (denoted by EdgeAttention) modules will aggregate the node-outputs and edge-outputs into context vectors

(AggNG and AggEG respectively) by associating scores to each node (NAtt_i) and edge (EAtt_j). Using these context vectors, the *Final Predictor* (denoted by FinalPredictor) has a global view of the graph, as described in (5) - (8). This aggregation also assures that the Final Predictor receives a fixed-size input. The two attention modules take as input, besides the outputs of their corresponding RNNs (NRO_i and ERO_j), the final states of the Node RNN (NRO_n) and Edge RNN (ERS_m). These states are helpful in obtaining a global view of the processed graph, as they are obtained after processing each node and each edge using the RNN modules. Attention scores are produced for each node-output and edge-output and these scores are used to compute one final feature vector. The final decision of the GERNN (denoted by P) is made by the Final Predictor, as seen in (9).

$$\mathrm{NAtt}_i = \mathrm{NodeAttention}(\mathrm{NRO}_i, \mathrm{NRS}_n, \mathrm{ERS}_m) \quad (5)$$

$$\mathrm{AggNG} = \sum_{i=1}^{n} (\mathrm{NAtt}_i \cdot \mathrm{NRO}_i) \quad (6)$$

$$\mathrm{EAtt}_j = \mathrm{EdgeAttention}(\mathrm{ERO}_j, \mathrm{NRS}_n, \mathrm{ERS}_m) \quad (7)$$

$$\mathrm{AggEG} = \sum_{j=1}^{m} (\mathrm{EAtt}_j \cdot \mathrm{ERO}_j) \quad (8)$$

$$P = \mathrm{FinalPredictor}(\mathrm{AggNG}, \mathrm{AggEG}) \quad (9)$$

As our GERNN can be used to solve various problems, the significance of the final decision, or the output, depends on the problem to be solved. We use GERNN to search for recurrent memory cells which are efficient for a given task, more exactly sentiment analysis on tweets in our case study. When deciding if such a cell is efficient or not, GERNN outputs two scores, one for each option. When estimating the performance of the cell, GERNN outputs a single value, representing the predicted performance. We provide detailed explanations in subsection IV-B in regards to the tasks to be solved and the specific outputs.

In our implementation, all attention modules and the final predictor are FFNNs. For the RNNs we used GRUs [24] as hidden units.

B. PREPROCESSING FOR RECURRENT MEMORY CELLS

In this subsection we discuss the representation of the recurrent memory cells and the preprocessing that must be done in order to be able to process these cells using the GERNN algorithm.

We represent memory cells as directed graphs, each node and edge having different properties, representation that we adopt from [41]. The edges represent the flow of data, while the nodes represent computations. Each node has an activation function. The activations that we consider are: rectified linear unit (ReLU), linear (lin), sigmoid (σ), hyperbolic tangent (tanh) and gate. The gate activation takes two values as input, passes the first one through a sigmoid activation, the second one through a linear activation and multiplies the two obtained values. For a node, the associated computation is the activation function applied on the sum over the values brought by the edges. Of course, in the case of the "gate" activation, the edges are split in two groups (branches). Each node has an identifier associated to it. The cell has an input node, from which the input of the cell is taken and an output node which describes the output of the cell.

The properties of an edge include the weight type, which may be "identity", meaning that the weight is one, or "linear", meaning that the weight is a learnable parameter. An edge may be "recurrent", when the connection is done to the previous time step, or "regular" when the connection is done to the same time step. If we ignore the recurrent edges, our cell representations become directed acyclic graphs. An incoming edge of a node A may connect it to only a node B in the same cell, in which case the edge is "single", or it may connect to the B nodes from all the cells in the layer, in which case the edge is called a "depth" connection.

We must preproces the graphs representing the cells into a feature-vector representation, so that our algorithm may be applied on them. Each node and edge are converted into a numerical feature vector. Each node has a unique numerical identifier which is included as it is in the vector. For the rest of the properties, corresponding one-hot vector representations are included in the feature vector. The activation is converted into a one-hot vector with 5 positions, one for each possible function. We mark if a node is the input node or not and we mark if a node is the output node or not. Each of these two properties is represented as a one-hot vector with two positions (true / false).

The edge-vector contains the identifiers of the source node and target node. The rest of the properties are represented as one-hot vectors. Two values represent the time delay (recurrent / regular), two values represent the weight type (identity / linear), two values represent whether the edge is a depth connection and two values represent the branch number on which the edge is placed. For the nodes with activations other than "gate", all edges are considered to be on the first (and only) branch.

All node vectors are placed in a list and all edge-vectors are placed in another list, and these sequences are given as input to the GERNN.

IV. EXPERIMENTS AND RESULTS

In this section we present the experiments that we conducted for validating our GERNN as a performance estimation algorithm. We provide descriptions of the tasks that we considered, the datasets that we used, the experiments that we made and the results that we obtained.

In Fig. 2 we present the overview of our experimental methodology, which we explain in detail in the following subsections. To be able to evaluate the capacity of our GERNN to estimate the performance of RNN cells on sentiment analysis on tweets, we need RNN cells with the corresponding qualities for training and testing. In order to obtain the corresponding qualities, we use datasets of tweets with annotated sentiments or emotions. We build our datasets of RNN cells using the tweets datasets and an evolutionary algorithm which was previously proposed [41]. We train GERNNs on these datasets of cells. As we have both tasks for sentiment analysis and for RNN cell performance estimation, we use the terms "tweet-tasks" and "cell-tasks" for the two categories respectively. We do this to avoid confusion. After training the GERNNs, we compare them and select the most promising options based on the results. GERNN experiments are presented in detail and the best-performing architectures are reported. We generate one million new RNN cells and we use our GERNNs to filter these cells based on their estimated quality. A small subset of the cells are trained on tweet-tasks to establish the overall best cells.

Training and evaluation of the models was done on an Nvidia Tesla K40 GPU. All implementation was done in the Python programming language, using the TensorFlow framework [49].

A. TWEETS DATASETS

We use three sentiment analysis on tweets tasks (tweet-tasks), which we denote by TSAD, EI-2 and EI-4, tasks that were also considered in [41]. These tweet-tasks are based on two publicly available datasets of tweets annotated with sentiment [13], [50].

TSAD is a binary classification task, in which we used a corpus of 1578627 tweets [50]. This dataset has annotations for the sentiment expressed in each tweet: positive or negative. We used 80% of the samples for the training set and the rest for the testing set. We selected this dataset as it is, by far, the largest dataset of tweets annotated with sentiment that we found.

EI-4 and EI-2 are based on a corpus of 7102 tweets [13] with annotations for four emotion classes: "anger", "fear", "joy", "sadness". For both of these tweet-tasks, the training set consists of the samples marked by "train" and "dev" by the corpus authors [13] and the testing set consists of the samples marked as "test". For the EI-4 task, all samples are considered. For the EI-2 task, only two emotions are considered, "joy" and "anger", this being a binary classification tweet-task. This tweet-task was considered for creating a binary classification task with a smaller number of samples. The emotion classes were selected such that the emotions correspond to positive or negative sentiment. The smaller number of samples leads to faster trainings which are useful in creating the datasets of RNN cells. This dataset was selected due to having multiple emotions annotated, providing a more granular task on which to evaluate our RNNs.

B. RNN CELLS DATASETS

Our algorithm must be able to help discover new recurrent memory cells that can accurately extract the sentiment from tweets. For this purpose, GERNN must be able to make fast and accurate predictions of the performance of a given cell design. We define two cell-tasks that need solving: (1) finding which cell designs are appropriate for sentiment analysis on tweets and (2) what is their accuracy. The first cell-task can be modeled as a binary classification problem which we denote by *EABC* (empathic / apathetic binary classification), while the second can be modeled as a regression problem which we denote by *ERAP* (emotion recognition accuracy prediction).

For creating datasets of recurrent memory cells for each cell-task, we need to be able to evaluate their capacity of sentiment analysis on tweets.

We created and evaluated various cells on the three considered tweet-tasks (separately) by employing the evolutionary algorithm described in [41]. This algorithm searches for new recurrent memory cells that are efficient in solving a given task. The main limitation of this algorithm is the time taken to evaluate the cells. We use our GERNN to overcome this limitation, but first we need samples to train our performance estimator. We create our training and testing datasets for the two mentioned cell-tasks: binary classification datasets for finding which cell designs are appropriate for sentiment analysis on tweets and regression datasets for predicting the accuracy of the cells on sentiment analysis on tweets.

We briefly describe the procedure that we adopted from [41] to generate new RNN cells. The evolutionary algorithm contains a population of individuals that represent RNN memory cells. The representation of individuals is the one that we described in subsection III-B. In the beginning, the population is created using the mutation operator. The starting point of the mutation can be either a basic RNN cell or a LSTM, the first option being useful for creating new designs that are not constrained by other designs, while the second is useful for finding improvements to LSTM, which is one of the most widely used RNN cells.

The mutation operator consist of multiple mutations rounds in which various mutation operations are applied with a preset probability. These operations can add new nodes, add new edges or change the characteristics of the nodes or edges.

At each new generation of the evolutionary algorithm, pairs of parents are selected to form new individuals using the crossover operator. The crossover operator will randomly select nodes from each parent and only keep these nodes and the associated edges. The two subsets of nodes and edges are combined into a single individual by adding new edges. The individual is then mutated. After the new individuals are generated, a selection operator decides which individuals (new and old) are kept in the new generation based on the quality of these individuals.

The quality of an individual is measured in [41] by creating a full RNN consisting of multiple instances of the cell that the individual represents and training and testing this RNN on a task. As no machine learning algorithm was used in [41] for performance estimation, other (less sophisticated) strategies were used. Examples would be early stopping or approximating when an RNN stopped learning based on the improvement in the most recent training steps.

We used the algorithm proposed in [41] to generate new RNN cells and associate to each the accuracy obtained on the

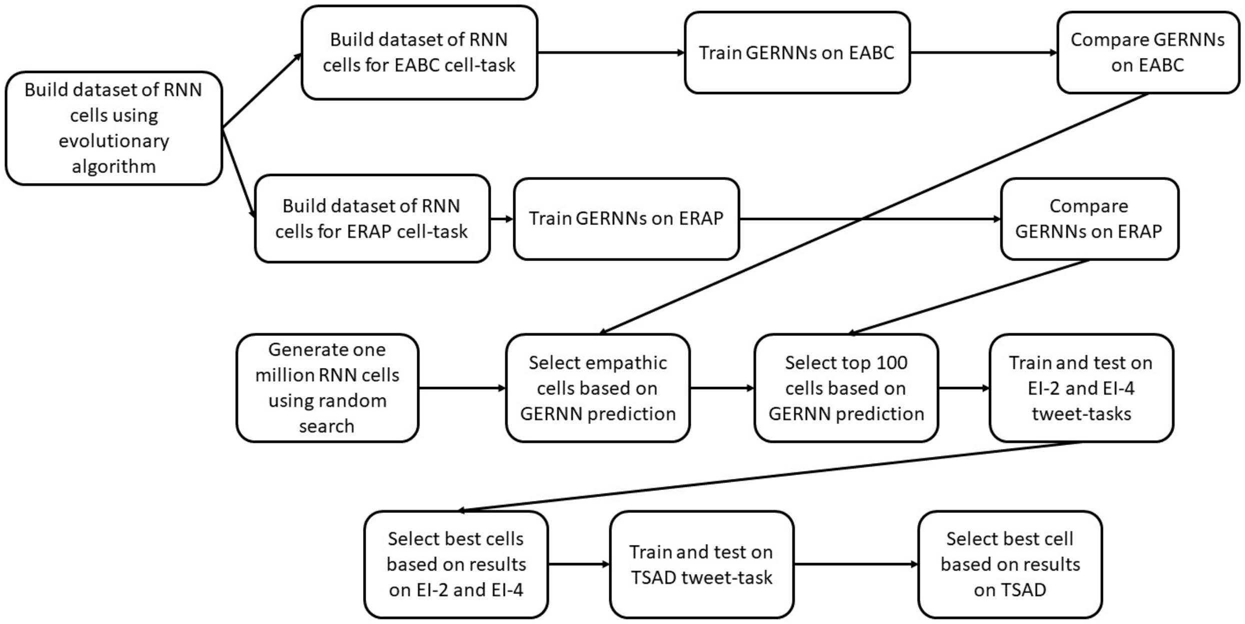

Build dataset of RNN cells using evolutionary algorithm
Build dataset of RNN cells for EABC cell-task
Train GERNNs on EABC
Compare GERNNs on EABC
Build dataset of RNN cells for ERAP cell-task
Train GERNNs on ERAP
Compare GERNNs on ERAP
Generate one million RNN cells using random search
Select empathic cells based on GERNN prediction
Select top 100 cells based on GERNN prediction
Train and test on EI-2 and EI-4 tweet-tasks
Select best cells based on results on EI-2 and EI-4
Train and test on TSAD tweet-task
Select best cell based on results on TSAD

FIGURE 2. Overview of the experimental methodology.

TABLE 1. Binary task dataset.

Task	Thresholds(%)		Cells number		
	Empathic	Apathetic	Empathic	Apathetic	Total
TSAD	55.0	52.0	53	53	106
EI-2	55.0	52.0	1322	1322	2644
EI-4	31.7	30.0	307	307	614
Total	-	-	1682	1682	3364

testing sets of the corresponding tweet-tasks (TSAD, EI-2, EI-4) as the quality. In this way, we created a dataset of pairs of RNN cells and the corresponding quality on a given tweet-task. For building a full RNN based on a given RNN cell, we use the same templates as in [41]: one layer of 64 units for TSAD and one layer of 32 units for EI-2 and EI-4.

Using this dataset, we built the two datasets for the cell-tasks that we mentioned at the beginning of this section. For the binary classification cell-task, EABC, we selected recurrent memory cell designs from each tweet-task. We applied a threshold on the accuracies obtained by the cells and the cells that passed the threshold were included in the "empathic" class, as these cells are capable of sentiment / emotion recognition. Cells which did not pass a second threshold, lower than the first, were included in the "apathetic" class. We decided to have two thresholds such that the dataset will more accurately express the difference between the two classes. Having a single threshold would mean that a difference of less than 1% may separate two similar cells into different classes, which may confuse the model more than it would help it. The thresholds applied for each tweet-task and the number of cells included can be seen in Table 1.

EABC requires the model to decide if a given cell is capable of sentiment analysis on tweets. This implies that each cell is part of either the "empathic" class or the "apathetic" class. For the GERNN to be able to make this classification, the final decision is represented by two scores - one for each possible class. The greater score decides the class of the cell.

For the regression cell-task, ERAP, we selected as the performance metric the accuracy of the network based on the cell design. In this case, the scores for each cell must represent a value on the same scale, so we could not include cells obtained from all tweet-tasks. We decided to use the cells from the EI-2 task, as they were the most numerous. We placed all the cells in buckets based on their scores. Each bucket represented an interval of size 0.001, where 1 is the accuracy of a perfect result. From each bucket we randomly sampled 10 cells for diversity of quality. For the buckets with less than 10 samples, we included the contents of the entire bucket. Our regression dataset contains 1349 cell designs and the distribution of their scores can be seen in Fig. 3, where the values for the scores were represented as percentages. The x-axis of the figure represents the score of the beginning of the interval included in the bucket and the y-axis represents the number of cells included from the bucket.

ERAP requires predicting the accuracy, on a given dataset, of an RNN based on the cell. To this purpose, the output of the GERNN that handles this task must be a single numeric value in the interval [0, 1].

C. EXPERIMENTAL SETUP

In this subsection we describe the experiments that we conducted using the model described in Section III applied on the datasets presented in subsection IV-B.

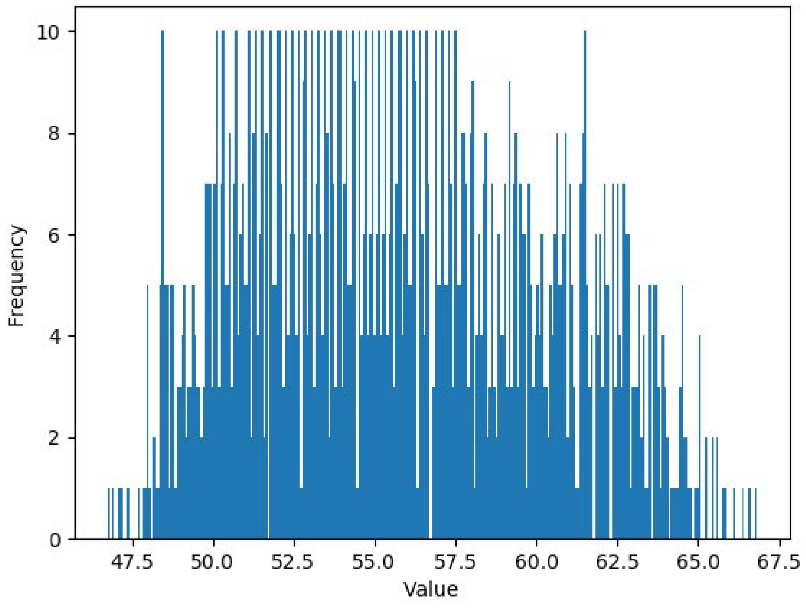


FIGURE 3. Distribution of scores in the regression dataset.

We searched for the best architecture for our GERNN, such that the metrics would be maximized. For the RNNs used within GERNN, we adjusted the number of layers and the number of units in each hidden layer. For the attention mechanisms, which are implemented as FFNNs, we adjusted the number of layers, the number of units in each layer and the activations of the artificial neurons. As possible activation functions, we considered rectified linear unit, sigmoid, linear and hyperbolic tangent.

For each cell-task (EABC / ERAP) we trained the various GERNN architectures for 50 epochs. Batches of 4 samples were given to the models. After each completed epoch, we evaluated the model to be able to monitor its progress.

As metrics for the binary cell-task, we considered accuracy, precision, recall and F-measure. The accuracy shows the percentage of samples that were correctly classified out of all the performed classifications, measuring how often the model is correct in its decision. Precision is the percentage of samples that are correctly classified as positive (in our case, empathic) out of all samples classified as positive, measuring how likely a sample considered positive by the model is truly positive. Recall is the percentage of samples correctly classified as positive (again, in our case, empathic cells) out of all the positive samples in the dataset, measuring how likely is the model to find all the positive samples. F-measure is a single metric that combines both precision and recall.

The metrics that we considered for the regression cell-task are mean squared error (MSE), mean absolute error (MAE), root mean square error (RMSE) and mean absolute percentage error (MAPE). These metrics measure the difference between the model output and the ground truth value, given that these values come from a continuous interval. All metrics compute the mean over a distance between a ground truth sample and the corresponding value predicted by the model. MSE uses as distance the squared difference of the two values. RMSE is the square root of MSE. In the case of MAE, the distance is the absolute value of the difference. MAPE also uses the absolute value of the difference, but it also divides it by the ground truth value, making the difference relative to the true value.

As loss functions, we used crossentropy for EABC and we experimented with both MSE and MAE for ERAP.

It is useful for a graph-processing algorithm to be invariant to the order in which the nodes and edges are processed [46]. The sequential nature of the RNNs that compose our algorithm assume from the beginning a given order. We experimented with establishing a fixed order in which the nodes and edges are processed, but for helping our GERNN obtain order invariance, we also experimented with randomly shuffling the sequences and feeding the same graphs to the algorithm, but with different orderings.

For each cell-task, we used 80% of the samples for training the model and 20% for testing. The metrics that we report in this paper are computed with respect to the test set.

D. RESULTS

In this subsection we present the results of the experiments that we made using the algorithm described in Section III and the methodology described in subsection IV-C.

For the EABC cell-task described in subsection IV-B, we searched for the GERNN architectures which obtains the best results. We altered the architectures of the RNNs by changing the number of layers and the number of units in each layer. For the attention mechanism, we changed the number

TABLE 2. Metrics of models on EABC (classification).

Nr.	Model description					Metrics(%)			
	NL	NU	AA	AS	IO	Accuracy	Precision	Recall	F-measure
C1	2	64	ReLU	B	R	**80.08**	**80.21**	**80.08**	**80.14**
C2	1	128	ReLU	B	R	79.34	79.47	79.33	79.40
C3	2	64	ReLU	B	F	77.39	77.52	77.38	77.45
C4	2	32	ReLU	B	R	79.64	**80.21**	79.61	79.91
C5	2	128	ReLU	B	R	77.99	78.23	77.97	78.10
C6	1	64	ReLU	B	R	78.89	78.89	78.88	78.89
C7	3	64	ReLU	B	R	76.94	77.42	76.92	77.17
C8	2	64	tanh	B	R	76.64	76.78	76.65	76.72
C9	2	64	σ	B	R	79.19	79.21	79.18	79.20
C10	2	64	ReLU	S	R	78.29	78.31	78.28	78.30
C11	1	128	ReLU	B	F	78.74	78.80	78.73	78.77
C12	1	256	ReLU	B	R	79.04	79.05	79.03	79.04
C13	1	128	tanh	B	R	71.55	72.55	71.58	72.06
C14	1	128	σ	B	R	76.79	76.82	76.79	76.80

of layers, the number of units in the layer and the activation functions used by the neurons.

The final architecture that we found to obtain the best performances for EABC has the same architecture for both RNN modules: 2 layers of 64 GRUs. All attention modules also have the same architecture: 3 layers of units with ReLU activation, of size 128, 64 and 32. The final predictor has the same architecture as the attention modules, with an additional layer of 2 units with ReLU activation.

After finding the best-performing architecture, we conducted ablation experiments to confirm that each architecture decision is the best one. The performances of our best GERNN model, marked by C1, and the comparison models can be seen in Table 2. In this table, we have the descriptions for each architecture and the results with respect to the four metrics we considered. For the network descriptions, we present the number of layers (NL) and the number of units per layer (NU) of the RNN modules, the activation function (AA) and the size (AS) of the attention modules and the type of input order (IO) that is used. The attention size of the base model is marked by B, while S marks smaller attention modules. The random input order is marked by R, while the fixed order is marked by F. Because the best results for the regression cell-task are obtained using a GERNN architecture similar to architecture C2, we also mirror the ablation experiments for this model.

As it can be seen, our base model has the overall best performance, having the highest accuracy, precision, recall and F-measure scores. The base model uses a random order of processing the preprocessed nodes and edges, so we first compared it with an identical model that is given the inputs in a fixed order, C3. It can be seen that the classification is improved by the random input order.

Next, we increased and decreased the number of units of the RNN modules and compared our base model with models that have 32 units (C4) or 128 units (C5) of GRUs. We did this to show that the size of our RNNs is the best one, as increasing or decreasing the number of units only decreases the performance. This shows that the number of units of our base model is high enough to be able to capture a diversity of features. Lowering the size will make the model unable to learn all the features it needs, while increasing the size only introduces new unnecessary learnable parameters.

Decreasing (C6) or increasing (C7) the number of layers produced lower performances. This proves that two layers are enough to learn the most complex concepts necessary for the classification cell-task, while less layers are not able to capture these complexities. We also distributed the units of the RNN modules in a single layer (C2), and found inferior results, to prove that the organization of units in layers is also important.

Other ablation experiments that we did involved changing the activation functions of the FFNNs. Using tanh (C8) or sigmoid (C9) produced GERNNs with performances inferior to our base model. We also modified the size of the attention modules. Architecture C10 has smaller attention modules (S) consisting of three layers of 64, 32 and 16 units. These smaller versions of the attention modules produced inferior results.

We mirrored most of the ablation experiments for the C2 architecture. We used a fixed order for the inputs (C11), a lower (C6) or higher (C12) number of units per layer. The number of layers was increased (C5). We changed the activation functions of the attention modules (C13 and C14). None of these alterations improved the performance of the model.

For the ERAP cell-task, we started from the GERNN architectures that we found to obtain the highest performances for the EABC cell-task. As the output of this model is a single rational number, we changed the last layer of the final predictor to a single unit with linear activation.

We conducted ablation experiments similar to the ones we conducted for the classification cell-task. Additionally, we experimented with two loss functions: MSE and MAE. The results of some of our ablation experiments can be seen in Table 3. The parameters of the model descriptions are the same as in Table 2. All models use the base size of the attention modules (B).

Most of the ablation experiments for the regression cell-task gave us the same insights as the experiments for EABC, so we present in Table 3 only the ones that we consider to provide additional information. What we found interesting is that having a fixed order of inputs produces superior performances on ERAP, as opposed to EABC. We also observed that considering MAE as the loss function helps the model obtain better performances with respect to all metrics, including MSE. As it can be seen, on the regression cell-task, the GERNN that uses as RNN modules networks with 1 layer consisting of 128 GRUs (R5) clearly outperforms the 2-layered version with 64 GRUs approach (R1). This distribution of the same number of units on a single layer is better able to solve ERAP, with respect to the considered metrics.

For visualizing what the GERNN considers important when making a prediction, we created heatmaps for the considered characteristics. A mean node vector and a mean edge vector were created. For each graph, we considered the

TABLE 3. Metrics of models on ERAP (regression).

Nr.	Model description				Loss	Metrics			
	NL	NU	AA	IO		MAE	MSE	RMSE	MAPE
R1	2	64	ReLU	F	MAE	0.0312	0.001479	0.0384	0.0555
R2	2	64	ReLU	R	MAE	0.0318	0.001546	0.0393	0.0575
R3	2	64	ReLU	F	MSE	0.0328	0.001654	0.0406	0.0601
R4	2	64	ReLU	R	MSE	0.0333	0.001637	0.0404	0.0592
R5	1	128	ReLU	F	MAE	**0.0302**	**0.001400**	**0.0374**	**0.0542**
R6	1	128	ReLU	R	MAE	0.0324	0.001589	0.0398	0.0579
R7	1	128	ReLU	F	MSE	0.0329	0.001642	0.0405	0.0595
R8	1	128	ReLU	R	MSE	0.0324	0.001585	0.0398	0.0582

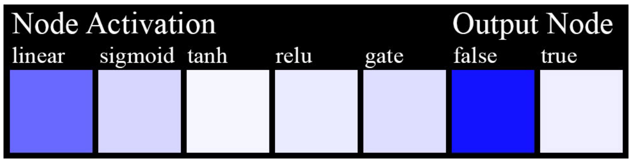


FIGURE 4. Mean node vector heatmap.

weights computed by the Node Attention and computed the mean node vector of the graph as the sum of the vectorial representations (obtained using the Node Preprocessor) of each node, weighted by the attention scores. We computed the mean node vector of the dataset as the average of all graph mean node vectors. The resulting heatmap can be seen in Fig. 4. Color close to white represents low weights, while intense blue represents high weights.

As it can be observed in the figure, the activation functions are approximately equally important, the linear activation being the only one that stands out. This shows us that linear-nodes are important for combining the results of multiple nodes and creating more complex equations before applying a function over the output. Output nodes have a relatively small weight in the mean node vector. As there are considerably more internal nodes than output nodes, this is an expected result. Also, it shows that our model considers all nodes when making the prediction, not only the final nodes that are close to the output of the cell.

Similarly, we computed the mean edge vector by considering the scores produced by the Edge Attention as the weights. Fig. 5 is the visual representation of the mean edge vector. This representation shows us that linear-edges are more important in the final decision than identity edges. This is understandable, as learnable weights are essential for a deep learning model. Of course, there are cases when weights with fixed values are needed in the recurrent cell, and that is why the identity-edges also have a fair weight in the mean edge vector. The very low score of the second branch can be explained by the fact that the gate activation is the only one that contains a second branch and, as seen in Fig. 4, it has an average weight among the node activations. The sum of the nodes with other activations is great enough to produce this low second-branch score.

Depth connections have a very low score compared to regular ones, showing that this should be a relatively rare

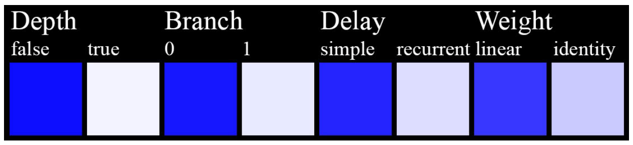


FIGURE 5. Mean edge vector heatmap.

encounter in the cell design. This insight shows us that each cell should generally have its own responsibility of identifying certain features. The depth connections are not completely ignored, as it is still important for the cells to exchange information among them in some cases. The simple edges have a greater weight in the mean edge vector than the recurrent ones. This shows us that while connections to previous time steps are important, quite essential, for RNNs, most computations of the cell are still done at the current time step.

E. FINDING NEW RECURRENT MEMORY CELLS

We applied our GERNNs to find RNN cells that obtain good performances on sentiment analysis on tweets. To this end, we used the following methodology:

1) Randomly generate one million recurrent memory cells, using the initialization and mutation operator described in [41] with various numbers of mutation rounds;
2) Apply a classification GERNN to decide which are the cells with high expected performances and keep only these cells;
3) Apply a regression GERNN on the cells, order them by the expected accuracy for EI-2 tweet-task, keep only the first 100;
4) Train full RNN architectures based on each cell on EI-2 and EI-4 and keep only the cells that obtain the highest accuracy on at least one of the tweet-tasks;
5) Train full RNN architectures based on each cell on TSAD.

The main focus of our work is performance estimation. Therefore, we paired our estimator with random search, as this is the simplest search strategy. In this way, the quality of the results that we obtain is given mostly by the estimator and other components have a lesser merit in the results.

The full RNN architectures that we used consist of a single layer of multiple instances of the same cell. For the EI-2 and EI-4 tasks, the layers contained 32 instances of the cells, while for TSAD, 64 instances were used.

We repeated this methodology with three different configurations. The configurations used different pairs of classification and regression GERNN architectures: C1 and R1, C2 and R5, C1 and R5. The first two pairs were selected due to them employing the architectures which obtained the best performances on the classification and regression tasks, respectively. The third pair employs the architecture that obtained the best performance on the classification task and

TABLE 4. **Metrics of newly discovered cells.**

Cell	GERNN		Accuracy(%)			
	C	R	EI-2 Estimation	EI-2	EI-4	TSAD
1	C2	R5	57.51	64.06	38.39	68.81
2	C2	R5	57.54	54.21	39.22	68.89
3	C1	R5	57.60	59.91	35.80	77.56
4	C1	R5	57.74	54.07	37.40	76.77
5	C1	R1	56.87	58.01	31.72	76.48
6	C1	R1	56.87	56.52	36.67	76.49

the architecture that obtained the best performance on the regression task.

In Table 4 we present the results of the best-performing recurrent memory cells that we discovered.

Out of the newly discovered cells, we consider cell 3 the best design, as it obtains the highest accuracy on the TSAD task and also the best average accuracy on all three tasks. As TSAD contains the most training samples, and so provides the most variation in the data, we consider good performance on this task as an indicative of a high ability to recognize sentiment in tweets. We name this new architecture Empathic Recurrent Memory Cell (ERMC), as it is specifically designed to recognize sentiment. The computations done by ERMC are described in (10) - (17). In these equations we denote by v_i^t the output of the node i of a cell instance, by V_i^t the vector containing the outputs of the nodes i of all the cell instances in the layer and by W_i^j the weight used by node i for the output of node j. By the superscript t we denote the current time step, while $t-1$ denotes the previous time step. X^t denotes the input of the cell, while $\sum$ represents summation over all the elements.

$$v_2^t = \text{relu}(\sum W_2^0 V_0^{t-1} + \sum W_2^I X^t) \quad (10)$$
$$v_3^t = \tanh(v_2^t) \quad (11)$$
$$v_5^t = \sigma(\sum X^t + \sum V_4^{t-1}) \cdot (W_5^3 v_3^t + \sum X^t) \quad (12)$$
$$v_0^t = \sigma(W_0^0 v_0^{t-1} + \sum W_0^I X^t) \cdot (W_0^5 v_5^t) \quad (13)$$
$$v_1^t = \sigma(\sum W_1^0 V_0^{t-1}) \quad (14)$$
$$v_6^t = \sigma(W_6^1 v_1^t) \cdot (W_6^0 v_0^t) \quad (15)$$
$$v_4^t = \sigma(W_4^6 v_6^t) \cdot (\sum W_4^I X^t) \quad (16)$$
$$v_{out} = v_0^t + v_2^t + v_4^t + v_5^t + v_6^t \quad (17)$$

A graphical representation of ERMC can be seen in Fig. 6. In this representation, each node has an associated numerical identifier written on it, which corresponds to the subscript identifier used in the equations. The color of the node denotes its corresponding activation function. For the gate activation, if the incoming arrows are connected to the yellow half, they are passed through the sigmoid part of the activation, otherwise they are passed through the linear part. Solid lines represent connections from the current time step, while dashed lines represent connections from the previous time step. Blue arrows represent connections with learnable weights, while black arrows represent identity connections. Depth connections are represented with thicker arrows.

This cell is slightly more complex than the general-purpose LSTM and GRU in terms of computations. The difference in complexity is small, so the particular abilities of this design are given by the structure that it has. It employs 3 gates for the control of the flow of data. Additionally, one of the gate-nodes, the one denoted by 0, is connected to most of the nodes of this cell. This node seems to act as a central computation node, receiving information from many nodes and distributing its outputs to many others. It additionally contains a recurrent connection to itself, leading us to believe that its role is also important for the internal state of the cell. ERMC also contains various types of connections, proving that the properties that we considered are useful for proposing new RNN cells.

TABLE 5. **Comparison of ERMC with other cells on the TSAD task.**

Cell	Accuracy(%)	Precision(%)	Recall(%)	TNR(%)
ERMC	77.56	78.56	75.79	79.33
RMC-3 [41]	78.35	77.53	79.82	76.87
LSTM [23]	80.04	80.41	79.41	80.66
GRU [24]	80.34	79.53	81.70	78.99

TABLE 6. **Performance estimation strategies comparison.**

	EABC	ERAP (EI-2)	EI-2	EI-4	TSAD
Mean time per cell (minutes)	0.0012	0.0011	9.33	18.98	4877.66
Total number of cells	2000000	582536	300	300	3

We present in Table 5 a comparison between the metrics obtained by ERMC on TSAD and the metrics obtained by other cells. The metrics for RMC-3, LSTM and GRU are as reported in [41]. As it can be seen, ERMC obtains results competitive with both the general-purpose cells, LSTM and GRU, and the cell designed for sentiment analysis on tweets, RMC-3. ERMC has better precision and true negative rate (TNR) than RMC-3 and better TNR than GRU.

We present in Table 6 a comparison of the running times and the number of cells that we evaluated in different settings. Here, EABC refers to predicting using a GERNN if a cell is capable of sentiment analysis on tweets, while ERAP (EI-2) refers to using a GERNN for predicting the accuracy of an RNN based on a cell on the EI-2 tweet-task. The other three, EI-2, EI-4 and TSAD, refer to training an RNN based on a cell until convergence and then testing it on the respective tweet-tasks. As an observation, for EABC we have evaluated two million cells as each of the one million cells that we generated were evaluated twice (using two GERNN architectures).

As observed in Table 6, using GERNN is considerably faster than actually training on either of the tasks. The comparison between ERAP and EI-2 is the most obvious example of improvement, as they handle the same tweet-task, but in substantially different amounts of time. The most dramatic difference is between the GERNN estimations and TSAD, as this tweet-task consists of a large amount of

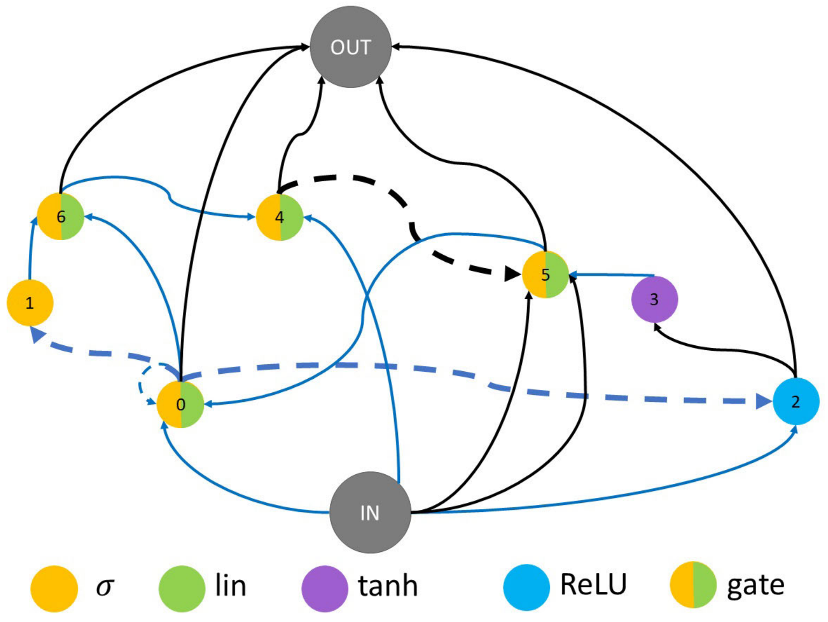


FIGURE 6. Graphical representation of the new ERMC architecture.

samples and requires many training steps until convergence. Even though the number of architectures trained on TSAD is not large enough to be statistically relevant, it still gives us an idea of the order of magnitude of the required time. As an observation, the mean number of minutes needed for TSAD translates to more than 3 days. From the results presented in Table 6 we conclude that the performance estimation strategy based on GERNN provides a significant improvement in the required time.

Even though ERMC did not clearly surpass its competition, its results are very similar to them, proving that our methodology can indeed lead to new high-performance RNN cells. We were able to evaluate one million RNN cells using GERNN, while only about 17000 were evaluated in [41], this being due to the great speed provided by the GERNN algorithm. For our current solution, GERNN was paired with random search and obtained these good results, while an evolutionary algorithm was used in [41]. Integrating GERNN with a search algorithm which better guides what architectures to evaluate would provide even better results.

V. CONCLUSION

The method of neural architecture search is useful in proposing new neural network architectures specifically designed for the task of interest. This method can be applied for finding new recurrent memory cells for a given task, but a performance estimation algorithm is necessary for being able to evaluate many designs and find the ones which obtain the best performances.

In this paper, we proposed GERNN, a novel algorithm for processing data structured as graphs. We described this algorithm and how it can be applied on estimating the performance of a recurrent neural network that is based on a certain cell. For a concrete task on which to use our algorithm, we selected sentiment analysis on tweets. The task of sentiment analysis on tweets, though challenging, has high applicability and so it is very useful to study. We applied our algorithm on this task and we presented the architectures that obtained the best results.

Based on the dataset of pairs of RNN cells and corresponding accuracies that we built, we trained multiple GERNN architectures. The GERNNs were compared based on the results and the various design decisions were discussed. We used the best-performing ones to search for new RNN cells.

Using GERNN, we evaluated one million new architectures that we generated, we discovered novel recurrent memory cells and we described and presented the one that obtains the best performances.

Even though the newly discovered cell did not clearly surpass the existing alternatives that we considered, it obtained similar results, proving that our methodology can lead to successful new RNN cell designs. Our algorithm proved to help in quickly finding new designs, but at this point we only paired it with random search. Combining our performance estimation algorithm with a more complex search strategy should produce even better results. As future work, we plan on integrating our algorithm into a more complex neural architecture search algorithm.

Using our experiments, we demonstrated that GERNN is able to accurately estimate the quality of an RNN on a given task based on the structure of the cell that is used. Moreover, GERNN is able to make this estimation much faster that other performance estimation strategies helping to improve the running times of neural architecture search algorithms.

The algorithm is able to generalize from the datasets that we used to the wider context of the task of sentiment analysis on tweets. Firstly, we used multiple smaller tasks based on different datasets to assure the capability of the method to generalize. Secondly, if limitations are found on certain sentiment analysis datasets, new data can be added to the training sets that we used in order to increase the variation in the data and further improve the generalization ability of the method.

GERNN is generic enough to make predictions for RNNs on any task and generic enough to be included in other NAS algorithms. This is due to the algorithm being tailored for RNNs, but not necessarily for sentiment analysis on tweets. To adapt our method to another task, the only modification that is needed is in the datasets that are used for the architecture search.

As practical applications of our paper, the cell that we discovered can be used to solve the problem of sentiment analysis on tweets. We demonstrated the speed with which our algorithm can discover new high-quality recurrent memory cells. Our methodology can be used to discover other architectures specific to tasks that can be modeled by RNNs: sequence processing tasks, natural language tasks etc.

Our algorithm, even though successful, has certain limitations. Firstly, it is highly dependent on the datasets that are used for the search. These datasets must be carefully selected to assure a good representation of the task that needs to be solved. Even though the GERNN will greatly reduce the search time, it still needs to first be trained. For this, it requires training data which consists of RNN cells and their corresponding performance. To obtain this, it is required to train such RNN cells to know their performance, so the datasets need to be of an appropriate size such that the trainings take a reasonable amount of time.

Another limitation is given by the random search, as previously mentioned. This is the simplest search strategy that is available. We focused our work on the performance estimation, but using a more sophisticated search strategy would provide superior solutions. We plan to overcome this limitation in our future work.

Our algorithm is also limited by certain constrains that we impose on it, but these constraints could be removed in future work. One constraint is that the number of cell instances in an RNN is fixed in the templates that we used. We could extend our work by making the templates adaptable, for example by keeping the number of learnable parameters constant, not the number of cell instances. Another constraint is that the full architectures that we build have instances of a single cell, but another option would be to build heterogeneous networks, having instances of different cells in the full architecture.

REFERENCES

[1] H. Li, "Deep learning for natural language processing: Advantages and challenges," *Nat. Sci. Rev.*, vol. 5, no. 1, pp. 24–26, 2017.

[2] L. Deng and Y. Liu, *Deep Learning in Natural Language Processing*. Springer, 2018.

[3] P. Goyal, S. Pandey, and K. Jain, *Deep Learning in Natural Language Processing*. New York, NY, USA: Apress, 2018.

[4] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015, *arXiv:1506.00019*.

[5] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Comput.*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019.

[6] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[7] M. Rambocas and B. G. Pacheco, "Online sentiment analysis in marketing research: A review," *J. Res. Interact. Marketing*, vol. 12, no. 2, pp. 146–163, May 2018.

[8] Z. Drus and H. Khalid, "Sentiment analysis in social media and its application: Systematic literature review," *Proc. Comput. Sci.*, vol. 161, pp. 707–714, Jan. 2019.

[9] K. Chakraborty, S. Bhattacharyya, and R. Bag, "A survey of sentiment analysis from social media data," *IEEE Trans. Computat. Social Syst.*, vol. 7, no. 2, pp. 450–464, Apr. 2020.

[10] C. I. Muntean, G. A. Morar, and D. Moldovan, "Exploring the meaning behind Twitter hashtags through clustering," in *Proc. Int. Conf. Bus. Inf. Syst.* , Vilnius, Lithuania. Berlin, Germany: Springer, 2012.

[11] S. M. J. Zafra, M. T. M. Valdivia, E. M. Camara, and L. A. U. Lopez, "Studying the scope of negation for Spanish sentiment analysis on Twitter," *IEEE Trans. Affect. Comput.*, vol. 10, no. 1, pp. 129–141, Jan. 2019.

[12] Y. Zhang, M. Shirakawa, Y. Wang, Z. Li, and T. Hara, "Twitter-aided decision making: A review of recent developments," *Appl. Intell.*, vol. 52, pp. 13839–13854, Feb. 2022.

[13] S. Mohammad and F. Bravo-Marquez, "Emotion intensities in tweets," in *Proc. 6th Joint Conf. Lexical Comput. Semantics (SEM)*, N. Ide, A. Herbelot, and L. Màrquez, Eds. Vancouver, BC, Canada: Association for Computational Linguistics, Aug. 2017, pp. 65–77.

[14] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining," in *LREc*, vol. 10, 2010, pp. 1320–1326.

[15] J. Awwalu, A. A. Bakar, and M. R. Yaakub, "Hybrid N-gram model using Naive Bayes for classification of political sentiments on Twitter," *Neural Comput. Appl.*, vol. 31, no. 12, pp. 9207–9220, 2019.

[16] E. Kouloumpis, T. Wilson, and J. Moore, "Twitter sentiment analysis: The good the bad and the OMG!" in *Proc. 5th Int. AAAI Conf. Weblogs Social Media*, 2011, pp. 538–541.

[17] L. Barbosa and J. Feng, "Robust sentiment detection on Twitter from biased and noisy data," in *Proc. 23rd Int. Conf. Comput. Linguistics*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 36–44.

[18] S. M. Nagarajan and U. D. Gandhi, "Classifying streaming of Twitter data based on sentiment analysis using hybridization," *Neural Comput. Appl.*, vol. 31, no. 5, pp. 1425–1433, May 2019.

[19] J. Wehrmann, W. Becker, H. E. L. Cagnini, and R. C. Barros, "A character-based convolutional neural network for language-agnostic Twitter sentiment analysis," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2384–2391.

[20] M. S. Akhtar, D. Ghosal, A. Ekbal, P. Bhattacharyya, and S. Kurohashi, "All-in-one: Emotion, sentiment and intensity prediction using a multi-task ensemble framework," *IEEE Trans. Affect. Comput.*, vol. 13, no. 1, pp. 285–297, Jan. 2022.

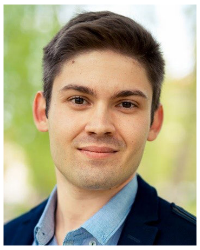
[21] A. Yadav and D. K. Vishwakarma, "Sentiment analysis using deep learning architectures: A review," *Artif. Intell. Rev.*, vol. 53, no. 6, pp. 4335–4385, Aug. 2020.

[22] T. Swathi, N. Kasiviswanath, and A. A. Rao, "An optimal deep learning-based LSTM for stock price prediction using Twitter sentiment analysis," *Appl. Intell.*, vol. 52, pp. 13675–13688, Mar. 2022.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[24] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds., Doha, Qatar, 2014, pp. 1724–1734.

[25] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc.*, Sep. 2014, pp. 338–342.
[26] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi, "Simple recurrent units for highly parallelizable recurrence," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 4470–4481.
[27] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.
[28] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[29] D. Barrios, A. Carrascal, D. Manrique, and J. Rios, "Cooperative binary-real coded genetic algorithms for generating and adapting artificial neural networks," *Neural Comput. Appl.*, vol. 12, no. 2, pp. 49–60, Nov. 2003.
[30] S. Ding, H. Li, C. Su, J. Yu, and F. Jin, "Evolutionary artificial neural networks: A review," *Artif. Intell. Rev.*, vol. 39, no. 3, pp. 251–260, Mar. 2013.
[31] K. G. Kapanova, I. Dimov, and J. M. Sellier, "A genetic approach to automatic neural network architecture optimization," *Neural Comput. Appl.*, vol. 29, no. 5, pp. 1481–1492, Mar. 2018.
[32] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–18.
[33] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, Jul. 2017, pp. 497–504.
[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[35] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.
[36] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–34.
[37] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–16.
[38] J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber, "Evolving memory cell structures for sequence learning," in *Proc. 19th Int. Conf. Artif. Neural Netw. (ICANN)*, Limassol, Cyprus, Sep. 2009.
[39] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.
[40] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," 2018, *arXiv:1803.04439*.
[41] S. C. Nistor, M. Moca, and R. L. Nistor, "Discovering novel memory cell designs for sentiment analysis on tweets," *Genet. Program. Evolvable Mach.*, vol. 22, no. 2, pp. 147–187, Nov. 2020.
[42] G. Kyriakides and K. Margaritis, "The effect of reduced training in neural architecture search," *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17321–17332, 2020.
[43] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018, pp. 1–14.
[44] W. Zheng, P. Zhao, K. Huang, and G. Chen, "Understanding the property of long term memory for the LSTM with attention mechanism," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2021, pp. 2708–2717.
[45] G. Serban and G. S. Moldovan, "A graph algorithm for identification of crosscutting concerns," Studia Universitatis Babes-Bolyai, Inf., Cluj-Napoca, Romania, Tech. Rep., 2006, pp. 53–60, vol. LI, no. 2.
[46] P. W. Battaglia et al., "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*.
[47] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, Jul. 2018.
[48] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Apr. 2021.
[49] M. Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: http://tensorflow.org
[50] I. Naji. (2012). *Twitter Sentiment Analysis Training Corpus (Dataset)*. Accessed: May 27, 2023. [Online]. Available: http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/



SERGIU COSMIN NISTOR received the Ph.D. degree in deep learning from the Faculty of Mathematics and Computer Science, Babeş-Bolyai University of Cluj-Napoca, Romania. He has published nine papers in conference proceedings and journals. His main research interests include machine learning, neural architecture search, and affective computing.

MOHAMMAD JARADAT received the Ph.D. degree in cybernetics and economical statistics from the Babeş-Bolyai University of Cluj-Napoca, in 2001. He is currently the Rector of the Bogdan Vodă University of Cluj-Napoca. He is the coauthor of more than 90 full papers in international journals and conference proceedings. His main research interest includes management, domain in which he is a scientific supervisor for Ph.D. candidates. He received the Doctor Honoris Causa awards from University of Pecs, Hungary and Medical University of Taipei, Taiwan.

RĂZVAN LIVIU NISTOR received the Graduate degree in manufacturing engineering and in business information systems, in 1992 and 1997, respectively, and the Ph.D. degree in reliability from the Technical University of Cluj-Napoca, in 1999. He is currently the Head of the Department of Management and a Professor in project management with the Babeş-Bolyai University of Cluj-Napoca. He is the coauthor of more than 60 full papers in international journals and conference proceedings. His research interests include project management, knowledge management, business analysis, and analytics.