The Journal of Supercomputing (2020) 76:3542–3554
https://doi.org/10.1007/s11227-018-2634-9

GRU: optimization of NPI performance

Wei Liu[1] · Quan Wang[2] · Yunlong Zhu[3] · Hanning Chen[2]

Published online: 19 October 2018

Abstract

Currently, artificial intelligence is being used in automatic programming by producing snippets of code. NPI (neural programmer-interpreter) is the most used technology that uses machine learning to implement automatic programming. This paper is aimed to improve the performance of traditional NPI and improve the speed of NPI training without loss of precision. To achieve this goal, we changed the core structure of NPI by adopting the GRU (gated recurrent unit) to replace LSTM (long short-term memory) in NPI. GRU has a control unit that regulates the flow of information within the hidden unit while without single memory unit. Numerical results have been presented to demonstrate the performance of the proposed methodology. That is, GRU-based NPI improved the performance of the original LSTM-based NPI by nearly 33% under the premise of ensuring equal accuracy.

Keywords Automatic programming · Neural programmer-interpreter · GRU

Abbreviations

NPI	Neural programmer-interpreter
LSTM	Long short-term memory
GRU	Gated recurrent unit
NTM	Neural Turing Machine

✉ Hanning Chen
183494284@qq.com

1 School of Information and Technology, Jilin Normal University, Siping 136000, China

2 School of Computer Science and Software, Tianjin Polytechnic University, Tianjin 300387, China

3 School of Electrical Engineering and Intellgentization, Dongguan University of Technology, Dongguan 523000, China

1 Introduction

The core problem of computer automatic programming is program synthesis. NPI (neural programmer-interpreter) would not generate code fragments, but it learns the rules of conversion from input and output data, and then a task can be achieved through these transformation rules.

1.1 Program synthesis

The task of program synthesis is to find the required programs to satisfy some form of constraint. Different from traditional compilers through semantic translation, high-level code is converted to low-level code through semantic translation. Program synthesis usually searches for programs to fit constraint in program space; the most common constraints are input and output pairs.

1.2 Neural programmer-interpreters

The main challenge of automatic programming is to let the machine learn the program itself and then quickly find the program to generate new programs to solve various tasks. NPI has a core module based on sequence model of LSTM (long short-term memory). It takes properties such as processing parameters and environment variables as input. The output is a keyword, indicating the procedure to call the next function and showing whether the program should be terminated.

The NPI has three learning components: The first is recurrent kernel, the second is a persistent key pair of the program storage module, and the third is a specific program encoder. NPI can express higher-level programs by learning lower-level programs, while reducing the complexity of samples and having a better generalization ability than the sequence-to-sequence LSTMs. The program storage modules allow for effective learning of additional tasks from existing programs. NPI can also use the environment (such as a panel with a read and write pointer) to cache the intermediate values in calculations, reducing the storage burden of the hidden unit. The NPI trains the model in a fully supervised way, which does not learn through a large number of relatively weak labels, but through a few rich samples.

Currently, the NPI model can learn more than 21 kinds of programs, including adding pixels to images, sorting, subtracting, trajectory planning. Crucially, these can be implemented *by* using a single NPI model with the same parameters.

By using neural networks to represent subprograms and to learn these subprograms from the data, it can generalize tasks with contain rich sensory input and uncertainty. The monitoring approach adopted in this article is to provide fewer tags, but tags contain more information, allowing the model to learn more complex combinations.

2 Related works

2.1 NPI related works

Rumelhart et al. [1] mentioned the use of dynamic programmable networks and the activation of the first layer network as the weight of the second layer network. Sutskever and Hinton [2] studied the relationship between high-order signs. Donnarumma et al. [3] developed a key component of the cognitive control system. Schmidhuber [4] studied the parameters of a slowly changing network and generated context-sensitive weights for the second rapidly changing network, which can only be demonstrated in very limited environments.

Schneider and Chein [5], Anderson [6], [7] proposed several theories about brain regions controlling other parts of the brain to accomplish multiple tasks. Graves et al. [8] developed a NTM (Neural Turing Machine) capable of learning and executing simple replication, simple prioritization, and associative memory.

Vinyals [9] proposed the pointer network, which summarized the concept of encoder attention, thus providing a variable output space according to the length of input sequence. This work is also closely related to program induction.

Banzhaf et al. [10] found useful programs from candidate programs. Mou et al. [11] used handler symbols to learn the embedding of the maximum margin program with the help of the parse tree. Zaremba and Sutskever [12] trained the LSTM model to read characters in the simple program text and correctly predicted the program output. Joulin and Mikolov [13] developed the push stack by adding a repetitive network, which allows for generalization of longer input sequences, rather than a few algorithm patterns during training.

Several papers also studied the application of the recursive neural network (Zaremba and Sutskever [14]; Zaremba et al. [15]; Kaiser and Sutskever [16]; Kurach et al. [17, 18]). Although we have similar motivation, our approach was different, using the combination structure of the program memory explicitly merged into the network, allowing the model to learn a new program through the composite subroutine.

2.2 The history of LSTM

Original LSTM version includes some cells, input gate, and output gate. However, original LSTM have not forgotten gate and peeking connection, even ignored the output gate in some experiments, the deviation of the unit or enter the activation function, the training process through real-time recursive learning and back propagation training. Therefore, the study did not use precise gradient training. Another feature of the original version is the use of the entire portal recursion, which means that all gates are reentered in the previous time step, except for the output of the block loop input. This feature does not appear in any subsequent release.

The first recommendation to modify the LSTM architecture suggests adding the forgotten gate that allows the LSTM to reset its own state, thus allowing the improved LSTM to learn the continuous tasks.

Gers and Schmidhuber [19] proposed that in order to learn accurate timing, cells need to control the structure of the gate. So far, this can only be done through an open output gate. To make the exact time easier to learn, a peep hole connection from the cell to the gate is added to the architecture. In addition, the output activation function is ignored because there is no evidence that it is critical to solve the current problems with LSTM testing.

The last LSTM version is developed by Graves and Schmidhuber [20], namely the vanilla LSTM. This version trained LSTM through reverse propagation and gives the results of TIMIT experimental. Using complete BPTT has an additional advantage, which can check the LSTM gradient and make the practical implementation of finite difference more reliable.

Vanilla LSTM is the most commonly used structure, but other variations have been proposed by researchers. Before the complete reverse propagation training, Gers et al. [21] proposed a training method based on extended Kalman filter, which made LSTM high computational complexity costly in some cases. Schmidhuber et al. [22] proposed a reverse propagation training method with mixed evolution method, but retained the LSTM structure.

Bayer et al. [23] improved different LSTM block structures and improved the adaptability of context-sensitive grammar to the maximum extent. Sak et al. [24] proposed a linear projection layer, which projected the output of LSTM layer to the connection of circular forwarding to reduce the number of parameters of multiple blocks in the LSTM network. Doetsch et al. [25] improved the performance of LSTM in the offline handwriting recognition data set by introducing the training scale parameters to the slope of the gate activation function. Otte et al. [26] improved the convergence rate of LSTM by adding a circular connection between the gates of individual blocks (rather than between blocks).

Cho et al. [27] proposed a variant structure to simplify the LSTM structure, called GRU. GRU does not use the peer connections, output gates and the forgotten gates are coupled to the update gates, and the final GRU reset gate (that is, the output gate corresponding to the LSTM) only connect the loops to the block input. This paper adopts the LSTM variant structure, namely GRU, to improve the performance of NPI, and the training speed of NPI can be improved significantly.

3 The improvement of NPI

3.1 NPI model

The core of NPI is the long short-term memory network. LSTM was proposed by Hochreiter and Schmidhuber. The LSTM plays a routing role between the current state and the previously hidden unit state.

As shown in Fig. 1, in NPI model, current time node is t, e_t is the status of environment, a_t is function parameters, e_t and a_t are as input to the encoder f_{enc}, generated state s_t, and then p_t, s_t are as input to MPL and f_{lstm}, the output is the output state after the update h_t, h_t as input will be as three decoder, respectively, f_{prog} decoder will generate embedding function keys; f_{end} decoder will generate the probability that

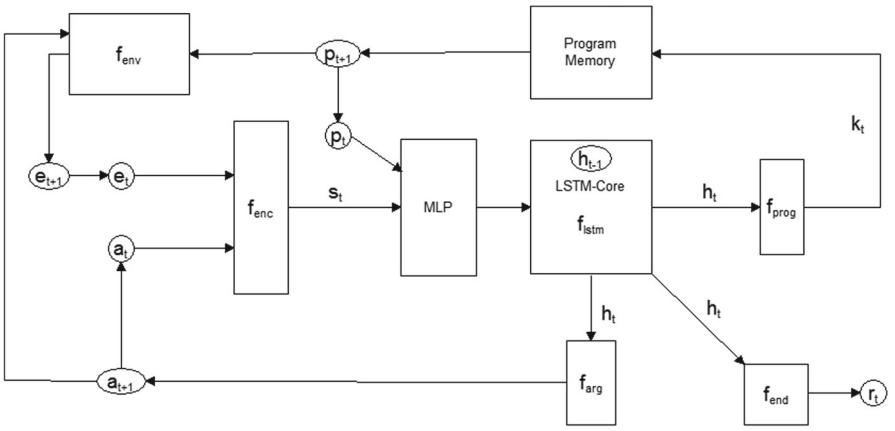


Fig. 1 NPI model. The data flow showed in this graph

the program will be terminated. The threshold value r_t in this article will be set to 0.5; f_{arg} decoder will update the function parameters of the next time node and determine the environment state of the output of the next time node through the environment change function, which is the principle of NPI operation.

3.2 LSTM model

As shown in Fig. 2, the main structure of LSTM consists of three gate structures, namely the input gate, the output gate, and the forgotten gate. First determine what information should be discarded in the cell state by forgotten door. Then the input gate determines what information needs to be stored in the cell state. Finally, the output gate determines what information needs to be exported to the next LSTM.

Unlike the weighted nonlinear recursive function that simply computes the input signal, the LSTM unit has a memory c_t at any time node t.

Hidden unit h_t at time node t:

$$h_t = o_t \tanh(c_t).$$

Output gate o_t:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t).$$

Activation function σ is the sigmoid function. Cell memory c_t:

$$c_t = f_t c_{t-1} + i_t \widetilde{c}_t.$$

New memories $\widetilde{c}_t$:

$$\widetilde{c}_t = \tanh(W_c x_t + U_c h_{t-1}).$$

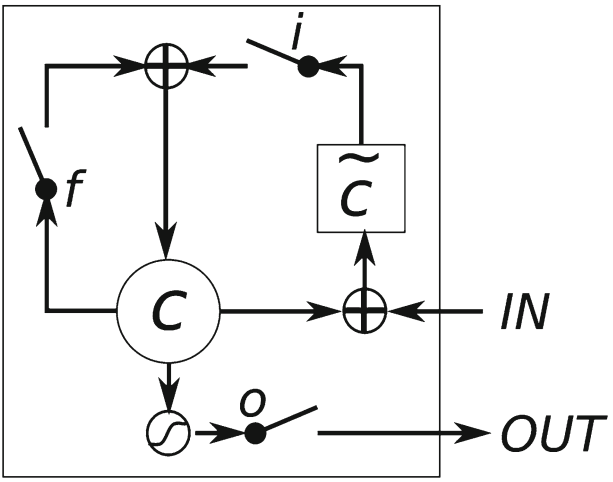


Fig. 2 Structure of LSTM. The main structure of LSTM consists of three gate structures, namely the input gate, the output gate, and the forget gate

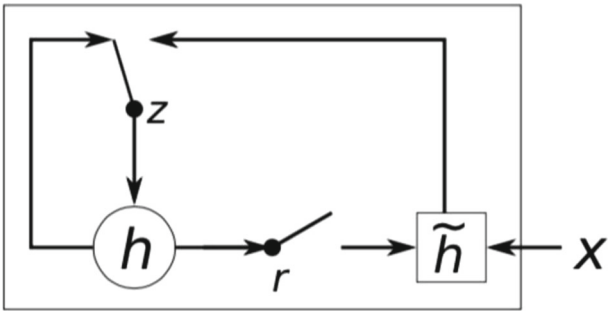


Fig. 3 Structure of GRU. We will improve the structure of LSTM in NPI, using a variant structure of LSTM, GRU

Forgotten gate f_t:

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + v_f c_{t-1}\right).$$

Input gate i_t:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1}).$$

3.3 GRU model

In this article, we will use the LSTM variant structure GRU to improve the LSTM structure in the NPI. GRU has a control unit that regulates the flow of information within the hidden unit, but no single memory unit (Fig. 3).

Hidden state h_t:

$$h_t = (1 - z_t)h_{t-1} + z_t\widetilde{h}_t$$

The update gate determines what information needs to be updated z_t:

$$z_t = \sigma(W_z \cdot x_t + U_z h_{t-1})$$

The candidate of hidden state $\widetilde{h}_t$:

$$\widetilde{h}_t = \tanh(W \cdot x_t + U(r_t \otimes h_{t-1}))$$

The reset gate that allows GRU to forget the previous calculation r_t:

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1}).$$

3.4 The difference between LSTM and GRU

Whether LSTM unit or GRU unit, the most significant feature compared to traditional RNN is the time t added to $t + 1$, which is lacking in traditional cycle units. Traditional loop units always use new values of the current input and hidden state of the cell to replace the content of the current cell, and retain the existing content of the LSTM unit, GRU unit, and on the basis of the existing content added content after screening. The advantages of this add-on are twofold. First, each unit is easily to remember a specific in the input stream and keep it. Any important property, whether it is the forgotten gate of the LSTM unit or the update door of the GRU, will not be covered by new data, but will remain as it is. Second, and more importantly, this method effectively creates a shortcut to multiple time nodes. The fast path allow error to be propagated back without quickly disappearing (if the control unit is close to saturation at time 1), as they pass through the constraints of multiple bounded nonlinear functions, thus reducing the difficulty due to gradient.

However, there are differences between LSTM and GRU units. In the GRU, the control of the memory in the LSTM unit is removed from the GRU, which simplifies the calculation of the LSTM to some extent. The GRU unit completely shows its contents and is not controlled by any gate. LSTM is another difference between unit in gate location, or is corresponding to the LSTM unit input GRU reset gate. LSTM unit to calculate the content of the new memory, without need to separate control node flow of information from the previous time, but the LSTM control unit will be independent from forgotten gate is added to the memory unit of the number of new memory unit of the memory contents. In GRU unit, when GRU calculates a new candidate activation, it will control the flow of information from a time node on the activation, but cannot add the number of candidate activation independent control, including control by update gate.

3.5 The training of NPI

This section uses input and output pairs to train the improved NPI. The input is $\varsigma_t^{\text{input}}$: $\{e_t, i_t, a_t\}$, the output is $\varsigma_t^{\text{output}}$ $\{i_{t+1}, a_{t+1}, r_t\}$, where t is the length of the sequence,

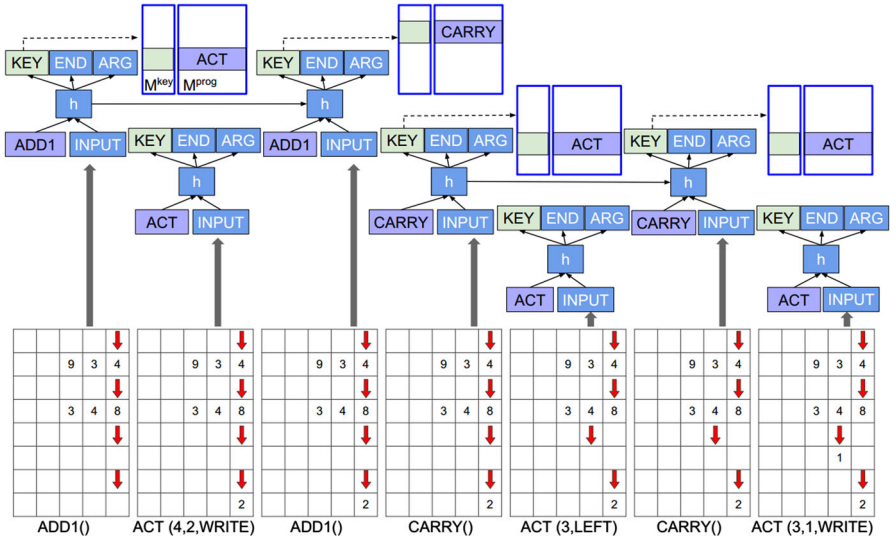


Fig. 4 Addition model based on NPI

i_t and i_{t+1} is the id of the program, corresponding to the key in the program space, according to the key can find the next step in the program space to call the subroutine.

$$\theta^* = \arg\max \sum_{(\varsigma^{\text{input}}, \varsigma^{\text{output}})} \log P\left(\varsigma^{\text{output}} \middle| \varsigma^{\text{input}}; \theta\right)$$

θ is the parameter of model

$$\log P\left(\varsigma^{\text{output}} \middle| \varsigma^{\text{input}}; \theta\right) = \sum_{t=1}^{T} \log P\left(\varsigma_t^{\text{output}} \middle| \varsigma_1^{\text{input}} \ldots \varsigma_t^{\text{input}}; \theta\right).$$

Input each h_t as a parameter into the three decoders, f_{prog} decoder will generate embedding function keys k_t, according to the corresponding values in the program memory, which is the subroutine to be executed by the next time node, in this case ACT; f_{end} decoder will generate the probability whether the program will be terminated. The threshold value r_t in this article will be set to 0.5; f_{arg} decoder will update the function parameters of the next time node, and determine the environment state of the output of the next time node through the environment change function.

$$\log P\left(\varsigma_t^{\text{output}} \middle| \varsigma_1^{\text{input}} \ldots \varsigma_t^{\text{output}}\right) = \log P(i_{t+1}|h_t) + \log P(a_{t+1}|h_t) + \log P(r_t|h_t).$$

This paper adopts adaptive curriculum learning, and the frequency of each batch of training samples is proportional to the current prediction error (Fig. 4).

4 Experiments

4.1 Addition model

As shown in the figure, add computation to the number 934 and 348. The arrow in the grid represents the pointer, which can be moved LEFT and RIGHT in the same line, namely LEFT and RIGHT, LEFT (LEFT), RIGHT (RIGHT), ADD (ADD), ACT (simplified), CARRY (CARRY), WRITE (WRITE). The left in the picture, for example, in a grid, will perform the first subroutine ADD, after a MPL and GRU in the composition of the core network function state of current time node h_t, h_t, respectively, as the parameter input into three decoders, f_{prog} decoder to generate embedding function keys to find the corresponding value in the space program, namely time nodes need to be performed under the subroutine, here is the ACT; f_{end} decoder generates the probability of terminating the program, the probability is less than 0.5. f_{arg} decoder will update the function parameters of the next time node a_{t+1}. The subsequent operations are similar to those in the first grid.

4.2 Results

In the NPI addition model, this paper adopts the experimental environment consistent with Scott Reed and Nando DE Freitas, which uses two layers of GRU; each layer contains 256 hidden units. For NPI training, adaptive moment estimation (Adam) was adopted. In practice, Adam's method works well. Compared with other adaptive learning rate algorithm, it converges faster and learns more effectively and can correct the problems existing in other optimization techniques, such as the problem that the learning rate disappears, the convergence is too slow, or the parameter update of high variance causes the loss function to fluctuate greatly. The learning rate set for NPI training is 0.0001, and the size of batch processing is 1.

The task in the NPI addition model is to read two numbers within two 10-digit and generate the number of answers. The goal is to learn to apply addition and carry operations from right to left in this algorithm.

In this environment, the network is given a grid to store the intermediate calculations. As shown in the figure, there are four Pointers: two for input numbers, one for carry and one for output. At each step, the pointer can move left or right, or a value can be recorded in the grid.

As shown in Fig. 5, under the premise that the accuracy rate is equal, the training time based on LSTM NPI uses is 105 min, and GRU-based NPI improved the performance of the original LSTM-based NPI by nearly 33% under the premise of ensuring equal accuracy.

5 Conclusion

Compared with traditional RNN, both LSTM and GRU units retain existing content and add filtered content on the basis of existing content, which enables the model

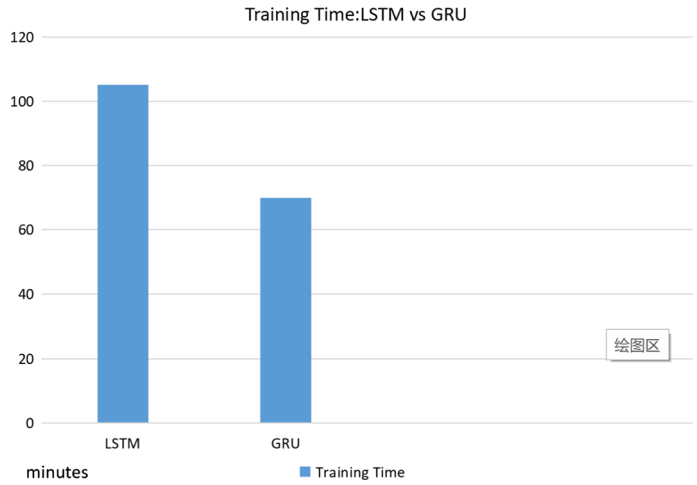


Fig. 5 Training time: LSTM versus GRU. The training time of GRU is much better than LSTM

to have memory function, and subsequent tasks can be carried out on the original basis. In this work, the control of the memory in the LSTM unit is removed from the GRU, which simplifies the calculation of the LSTM to some extent. According to the experimental results, the performance of LSTM and GRU is roughly equivalent, and the performance of GRU in some areas even exceeds that of LSTM.

Acknowledgements The research presented in this paper was supported by Ministry of Science and Technology of the People's Republic of China and National Natural Science Foundation of China.

Availability of data and materials The simulation code can be obtained by contacting the Email of corresponding authors.

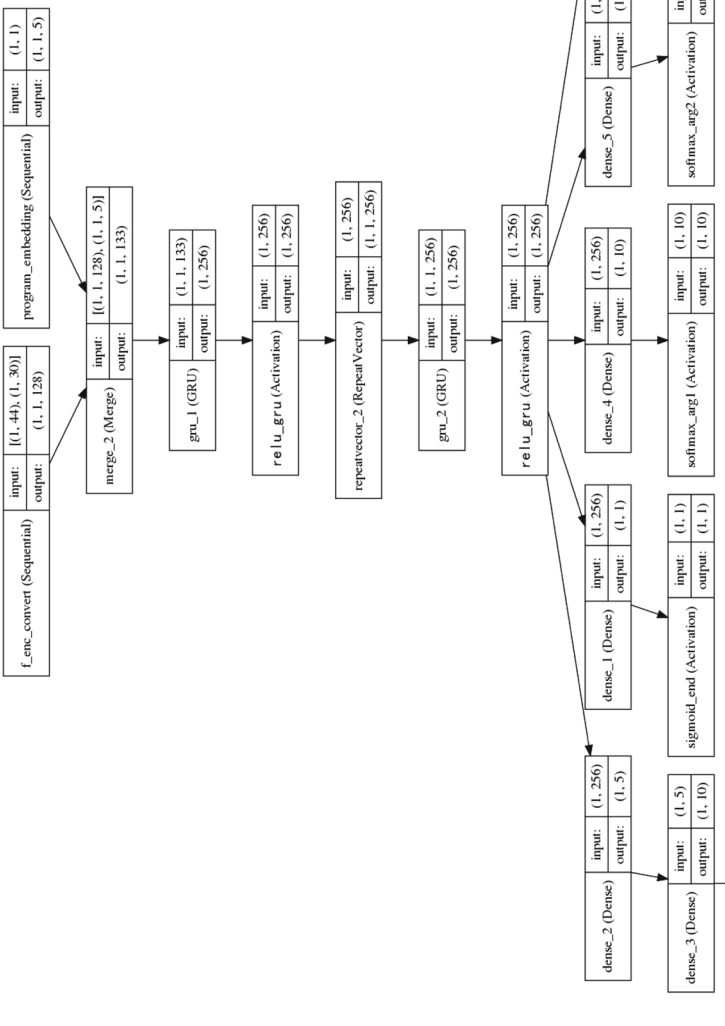
Authors contributions Wei Liu is the main writer of this paper. She proposed the main idea, completed the experiment, and analyzed the result. The other authors gave some important suggestions for the experiment. All the authors read and approved the final manuscript.

Funding This research is partially supported by the National key Research and Development Plan of China under grant No. (2016YFB1100501, 2017YFB1103603, 2017YFB-1103000), National Natural Science Foundation of China under grant No. (61772365, 41772123, 61602343, 51607122, 51575158 and 51378350), Tianjin Province Science and Technology Projects under grant No. (16JCYBJC18400, 16ZLZDZF-00150, 17ZLZXZF00310, 17JCQNJC04500, 17JCYBJC15100).

Compliance with ethical standards

Conflict of interest The authors declare that they have no competing interests.

Appendix 1: NPI flowchart based on GRU



References

1. Rumelhart DE, Hinton GE, McClelland JL (1986) Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. Chapter. In: A general framework for parallel distributed processing, MIT Press, pp 45–76
2. Sutskever I, Hinton GE (2009) Using matrices to model symbolic relationship. In: Advances in neural information processing systems, pp 1593–1600
3. Donnarumma F, Prevete R, Chersi F, Pezzulo G (2015) A programmer interpreter neural network architecture for prefrontal cognitive control. Int J Neural Syst 25(6):1550017
4. Schmidhuber J (1992) Learning to control fast-weight memories: an alternative to dynamic recurrent networks. Neural Comput 4(1):131–139
5. Schneider W, Chein JM (2003) Controlled and automatic processing: behavior, theory, and biological mechanisms. Cogn Sci 27(3):525–559
6. Anderson ML (2010) Neural reuse: a fundamental organizational principle of the brain. Behav Brain Sci 33:245–266
7. Brito R, Fong S, Cho K (2016) Towards implementation of residual-feedback GMDH neural network on parallel GPU memory guided by a regression curve. J Supercomput 72(10):1–28
8. Graves A, Wayne G, Danihelka, I (2014) Neural turing machines. arXiv preprint arXiv:1410.5401
9. Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. In: International Conference on Neural Information Processing Systems. MIT Press
10. Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming: an introduction, vol 1. Morgan Kaufmann, San Francisco
11. Mou L, Li G, Liu Y, Peng H, Jin Z, Xu Y, Zhang L (2014) Building program vector representations for deep learning. arXiv preprint arXiv:1409.3358
12. Zaremba W, Sutskever I (2014) Learning to execute. arXiv preprint arXiv:1410.4615
13. Joulin A, Mikolov T (2015) Inferring algorithmic patterns with stack-augmented recurrent nets. In NIPS
14. Zaremba W, Sutskever I (2015) Reinforcement learning neural turing machines. arXiv preprint arXiv:1505.00521
15. Zaremba W, Mikolov T, Joulin A, Fergus R (2015) Learning simple algorithms from examples. arXiv preprint arXiv:1511.07275
16. Kaiser Ł, Sutskever I (2015) Neural gpus learn algorithms. arXiv preprint arXiv:1511.08228
17. Kurach K, Andrychowicz M, Sutskever I (2015) Neural random-access machines. arXiv preprint arXiv:1511.06392
18. Cong G, Bhardwaj O, Feng M (2017) An efficient, distributed stochastic gradient descent algorithm for deep-learning applications. In: International Conference on Parallel Processing. IEEE Computer Society, pp 11–20
19. Gers FA, Schmidhuber J (2000) Recurrent nets ¨ that time and count. In: Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on, IEEE, volume 3, pp 189–194. ISBN 0769506194
20. Graves A, Schmidhuber J (2005) Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw 18(5–6):602–610
21. Gers FA, Perez-Ortiz JA, Eck D, Schmidhuber J (2002) DEFK-LSTM. In: ESANN 2002, Proceedings of the 10th Eurorean Symposium on Artificial Neural Networks
22. Schmidhuber J, Wierstra D, Gagliolo M, Gomez FJ (2007) Training recurrent networks by EVOLINO. Neural Comput 19(3):757–779
23. Bayer J, Wierstra D, Togelius J, Schmidhuber J (2009) Evolving memory cell structures for sequence learning. In: Artificial Neural Networks–ICANN 2009, Springer, pp 755–764
24. Sak H, Senior A, Beaufays F (2014) Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In: Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH)
25. Doetsch P, Kozielski M, Ney H (2014) Fast and robust training of recurrent neural networks for offline handwriting recognition. In: 14th International Conference on Frontiers in Handwriting Recognition
26. Otte S, Liwicki M, Zell A (2014) Dynamic cortex memory: enhancing recurrent neural networks for gradient-based sequence learning. In: Artificial Neural Networks and Machine Learning—ICANN 2014, number 8681 in Lecture Notes in Computer Science. Springer International Publishing, pp 1–8

27. Cho K, van Merrienboer B, Gulcehre C, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder decoder for statistical machine translation. arXiv preprint arXiv: 1406.1078