Learning Arithmetic Operations With A Multistep Deep Learning

Bastien Nollet, Mathieu Lefort, Frédéric Armetta

Learning Arithmetic Operations With A Multistep Deep Learning

Bastien Nollet, Mathieu Lefort, Frédéric Armetta
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France

***Abstract*—Deep neural networks are difficult to train when applied to tasks that can be expressed as algorithmic procedures. In this article, we propose to study how the explicit guidance of a network through all steps of the algorithm, using external memory and active choice of inputs, can improve its learning capability. The idea is to take inspiration from a child's learning and running through a procedure via interaction with an external support such as a paper. We show that this mechanism applied to a simple multilayer perceptron can significantly improve its performance when learning either a multi-digit addition or multiplication, which are simple but yet challenging operations to learn.**

I. Introduction

Deep learning methods achieve state of the art performances in multiple and various domains such as image classification, automatic translation, games playing, ... However, they usually failed to handle increasingly complex tasks [1], which is one of the major challenges remaining in artificial intelligence. In this paper, we focus on problems that humans solve by executing an algorithm. While it is known that RNNs (Recurrent Neural Network) are Turing-complete and have the potential to imitate algorithms when correctly trained, their efficiency tend to stagnate when tasks are complex [2]. In order to provide new capacities to the system, one can develop dedicated architectures which tend to alleviate the trainability problem. This is the case for the Neural Turing Machine [3] for which a neural network is complemented by a flow control mechanism and memory capabilities. Although the enhanced network is differentiable end-to-end, it remains somehow difficult to parameterize and very sensitive to initialization values [4]. Complex neural networks have often proved difficult to train.

In this paper, we study another way to tackle the problem of learning algorithmic problems. Instead of complementing the architecture with dedicated modules, we propose to modify the way the network learns by decomposing the task in successive steps that will be fed into the model. Thus the network will use the output/input loop as an interactive external medium, which can be considered as a memory. This multistep learning allows the global resolution of the algorithm and mimics the way a human would unroll an algorithm on a piece of paper. By doing so, we want to study if such a learning procedure makes the network easier to train. We propose to compare ourselves to a similar network that fails to solve the problem end-to-end for sophisticated arithmetic operations [5].

Section II reviews the most common network architectures with an emphasis on their suitability toward algorithmic learning. We introduce an alternative way to tackle algorithmic resolution in section III. The network we use is presented in section IV. Comparative results and the robustness of the model to irrelevant data are presented in section V. Section VI discusses the organization of the acquired knowledge induced by the learning protocol we introduce. Conclusion is then presented in section VII.

II. State of the art

A. Various deep learning architectures

Extending the classical Multilayer Perceptron (MLP), various deep network architectures have been proposed to solve complex tasks [6]. In the domain of computer vision, convolutional neural networks achieve good recognition performance [7] that can be close or even better than human's ones in some specific tasks. For problems with temporal dependencies between the inputs such as natural language processing, recurrent neural networks or dedicated cells such as Long Short Term Memory (LSTM) [8] or Gated Recurrent Units (GRUs) [9] are particularly suited. These approaches can be used in a supervised learning context but can also be combined with the reinforcement learning framework, e.g. to learn to play video games [10].

Even if all these models are universal approximators (see [11] for the MLP), training them in practice can be hard [2], especially when the teaching signal is sparse or when the temporal dependence is at long term e.g. (LSTM have initially been proposed to tackle these kinds of issues). To overcome this trainability issue, more recent models, mainly using an internal memory, were studied and will be described in the next section.

B. Toward an algorithmic learning

Based on a "simple" chain of continuous geometric transformations, deep learning models tend to increase their complexity in order to manage more complex tasks. This can be done thanks to attention mechanisms and recurrence or inclusion of an additional memory.

1) Natural language processing models: Deep learning models have been effective in dealing with problems such as machine translation, text summarizing, speech recognition or question-answering. Word2vec [12] and GloVe [13] are well-established architectures used to generate relevant vector representations of words. Based on these representations, End-To-End Memory Networks [14] introduce an attention mechanism

allowing to select relevant parts of texts for question answering. The model proposes to stack layers in order to allow complex multistep inferences and tackle spatial or temporal dependencies between related topics. Seq2Seq models [15], [16] manage complex relationships between words thanks to recurrence in order to encode and decode textual data. They can be complemented by an attention mechanism to focus on different parts of the input sequence allowing to preserve the context. The transformer model on its side introduces a self-attention mechanism which makes possible to eliminate recurrence [17] . Encoders and decoders blocks are stacked on top of each other which has a big influence on the performance.

2) Neural Turing Machine: The Neural Turing Machine extends the capabilities of neural networks to address algorithmic problems by coupling them to an internal memory and an attentional process [3]. The proposal is differentiable end-to-end and can be trained in a supervised way. It outperforms recurrent neural networks such as the Long Short-Term Memory (LSTM) on tasks such as storing and retrieving the i-th element of a list.

In a different way than what we propose, the gradient must be propagated over the entire network expressing the different stages of resolution, driven solely by the final output of the algorithm. During the training, the supervised signal represents the output of the total computation so that the network has to learn by itself the ways to solve the problem. Although the results are impressive, the selected algorithm for the training is relatively simple and memory oriented (storing and retrieving a list of items). It remains delicate to setup and very sensitive to initialization values ([4]) so one can wonder about the difficulties involved by increasing algorithm complexity.

3) Arithmetic processing: Learning arithmetic operations is an emergent research field in deep learning. It can be studied to extend the generalization capabilities of deep neural networks [18], or as a first step toward learning more complex algorithmic procedures, which is our concern in this article.

Our approach is intended to be general-purpose and does not include any specificity for numerical calculation but some approaches propose architectures dedicated to real numbers (see [19]).

Among the works dealing with arithmetic processing, [20] proposes a neural solver for arithmetic expression calculation. The neural solver both parses expressions and execute manageable arithmetic operations. The network is trained thanks to a hierarchical reinforcement algorithm. Tasks are hierarchically decomposed in sub-tasks, and acquired thanks to a curriculum learning. The approach does not appear to be designed to manage well large arithmetic multiplications. Results show an impressive ability to revolve medium size expressions, but that accuracy drops to 0% for long expressions when addressing multiplication expressions. The drop in performance is less drastic for expression composed by other operations (additions, divisions and subtractions). As a matter of fact, in these last cases the size of operands does not increase too much during the resolution of sub-expressions which keep the algorithmic complexity manageable. Indeed, increasing the number of digits of operands significantly increases the number of steps required for multiplications and the associated research area for the reinforcement algorithm.

It has been shown that a simple 3 hidden layers supervised network can infer with great accuracy the result of an addition taking as input the two operands (see [5]). However the same architecture and training procedure perform poorly on the multiplication, whose computation requires more complex and dependent relationship between digits. This illustrates the limitation of classical deep learning architectures to learn long algorithmic procedures.

Since our goal is to study the behavior of a simple network to learn complex operations, we propose to compare our model using comparable network settings used in [5] (the hidden layers are identical although the input and output parts are adapted to fit the needs of our approach to interact with the external medium, see section IV).

C. Active learning

Beyond the structural choice made for the training, it has been shown that an active learning strategy consisting of choosing the right next example to learn can achieve a greater performance [21], [22]. For instance, this can be done with curriculum learning where tasks of increasing complexity are successively presented to the network [23]. This dynamic can also autonomously emerge from an artificial curiosity mechanism, inspired by developmental psychology, where the system tries to maximize its learning progress [24].

III. Our multistep learning for arithmetic operations

A. General design

In the context of this article, we choose as a first step to imitate algorithms whose contents are known, i.e. algorithms that we can unfold step-by-step. Thus, intermediate resolution states are made available thanks to the flow of the algorithm. Future research should consider partially known algorithms for which partial results could be exploited to guide the model learning further.

The guiding idea is to provide an intermediate supervision signal which allows to learn many elementary operations depending on the input. The general principle relies on reading and writing on a board or memory such as shown in figure 1 for an addition (the unfolding of additions is detailed in the next section). We do not constrain access to this external memory and keep the network as simple as possible for reading and writing (see details in section III-C).

The succession of elementary operations has to lead to the resolution of the global problem. The network then needs to learn for each step:

- where the useful information takes place on the board
- what is the result of the inference
- where to report the outcome

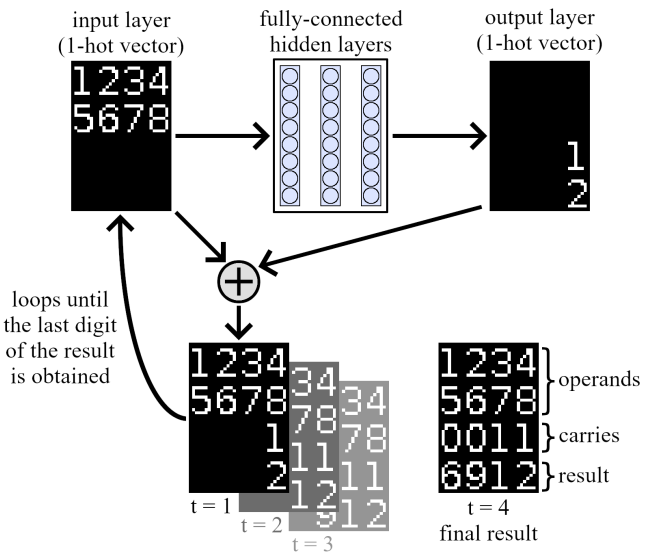


Fig. 1: Overview of the processing flow. The output of the network is combined with the current input to construct the next input. Thus, the output can be viewed as an external memory and its combination with the input as an external recurrence for the system. This allows the algorithm to be rolled out step by step on this external support.

B. Application to addition and multiplication operations

The network learns to mimic the algorithm unfolding leading to the problem solving. For the purpose of clarity, the resolution steps are first presented sequentially. The sequential nature of the problem does not prevent from learning the steps in parallel and independently as implemented. Because our training algorithm uses an external memory, at each step the network has access to all necessary data to perform the next operation. Thus, even if the dataset is constructed in a sequential manner, two successive operations are no more temporally dependent and can be learned in any order.

1) Addition: As presented in figure 2, the first two lines are used for the operands. Each resolution step consists in adding two digits, taking into consideration the previous possible carry, and reporting the new value and carry. The set of carries and the final result respectively appear on the penultimate and last lines.

2) Multiplication: The multiplication stands for a more complicated algorithm and has to be decomposed further (figure 3). As before, we choose to unroll the algorithm digit-by-digit, first having the partial multiplications and ending with the final additions also unrolled . The specificity of this algorithm is to mix both multiplications and additions to compute partial results as presented in figure 4. When considering 7-digit-numbers as in [5], this algorithm requires 35 steps which is a difficult task.

C. Problem formalization

In order to focus on the learning of arithmetic operations, we chose to represent digits using 1-hot vectors in the same

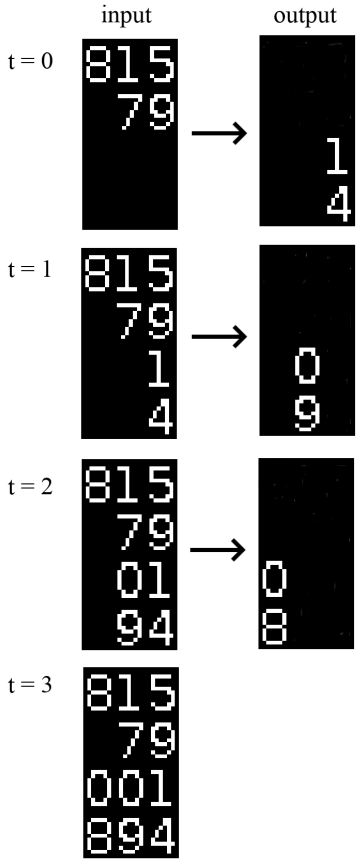


Fig. 2: Step-by-step unfolding of an addition. Two digits are written each step: a partial result digit, and a carry to consider for the next step. The last line represents the final result but is not used in the training dataset.

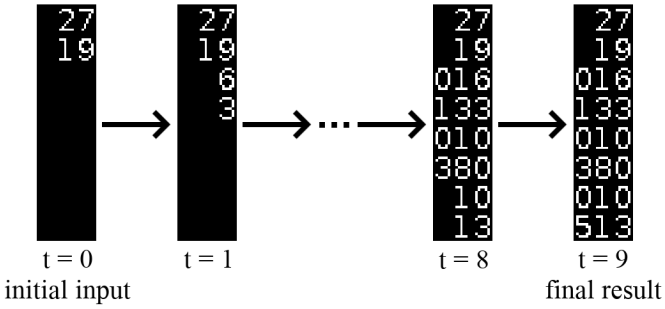


Fig. 3: Example of step-by-step unfolding of a multiplication (the resulting full-resolution map is presented in figure 4).

way it has been done in [5] (see figure 5). For the purpose of clarity, digits are represented numerically in most of the figures.

Let $D = \{(x^j_{step}, y^j_{step})\}_{step,j}$ be the dataset with x^j_{step} and y^j_{step} representing respectively the input and the desired output of the network for the $step$-th (unfolding) step of the j-th operation example. x^j_{step} and $y^j_{step} \in \{0,1\}^{s \times l \times 10}$, l being the number of lines used to unfold the operation, s being

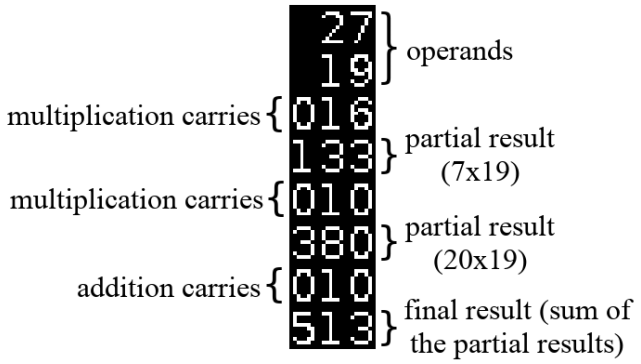


Fig. 4: Full-resolution map for the multiplication.

the number of digits used for the operation. As presented in figure 5, each digit is encoded with a 1-hot vector of size 10, an additional null value (represented in black in the numerical figures) is expressed by 0s in all the vector cells (see figure 5).

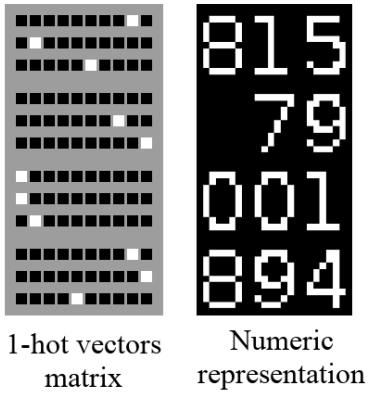


Fig. 5: 1-hot encodings for digits (for this illustration, 1-hot encodings are concatenated by three (left part) and are associated to three values (right part))

As presented in figure 1, while each resolution has to follow the naturally ordered algorithmic flow up to x^j_{end} and y^j_{end}, the training for such a system can be done independently for each of the considered steps and pairs of operands. During training, the system receives the input x^j_i and tries to predict y^j_i. Each input is computed by mixing x^j_{i-1} and y^j_{i-1} : $x^j_i = x^j_{i-1} + y^j_{i-1}$. During testing, the system has to sequentially perform the arithmetic operation, beginning with the input x^j_0. Then, the next input will be constructed with the following formula: $x^j_{i+1} = update(x^j_i, \hat{y}^j_i)$, with i the current step and $\hat{y}^j_i$ the output of the network receiving x^j_i.

The function Bin is used to transform an output vector (one-hot-vector) the following way :

$$\begin{array}{lllll} Bin: & [0,1]^{10} & \rightarrow & \{0,1\}^{10} & \\ & \mathbf{o} & \rightarrow & \mathbf{p}, \forall i, p_i = \left\{ \begin{array}{ll} 1 & \text{if } o_i = max_j o_j \\ & \text{and } o_i > thr \\ 0 & \text{else} \end{array} \right. \end{array}$$

The update function is then defined as follows :

$$\begin{array}{lllll} update: & \{0,1\}^{s\times l\times 10} & \times[0,1]^{s\times l\times 10} & \rightarrow & \{0,1\}^{s\times l\times 10} \\ & x & , \ y & \rightarrow & update(x,y) \end{array}$$

$$update(x,y)_{a,b} = \left\{ \begin{array}{ll} \mathbf{x_{a,b}} & \text{if } Bin(\mathbf{y_{a,b}}) = \mathbf{0} \\ Bin(\mathbf{y_{a,b}}) & \text{else} \end{array} \right.$$

One can note that when ground truth, $update(x^j_i, y^j_i) = x^j_i + y^j_i$ which is consistent with the way the dataset is constructed.

IV. Model

A. Network topology

The originality of our approach lies in the use of an external memory (the output been fed back and combined with the input) and the sequential learning of the algorithmic procedure (see section III). To test our hypothesis, we used a simple neural network architecture, more precisely we use the architecture proposed in [5] for comparison purpose.

The network is a multilayer perceptron with an an input layer of size $s \times l \times 10$, with s the maximum number of digits allowed per line, l the number of lines on the board, and 10 the number of cells required to encode a digit with 1-hot encoding (null value included, see section III-C). The network contains 3 hidden layers of 256 neurons with ReLU activation functions ($ReLU(x) = max(0, x)$). These layers are fully connected. The output has the same size as the input layer, needs to be the same size as the input because of our learning procedure and runs into a $softmax$ activation function. For experiments, we used the Adam optimizer [25], with a learning rate $\alpha = 0.001$, a variant of the stochastic gradient descent which uses the first and the second moments of the gradients instead of just the first moment, in order to obtain better performances. The network is trained using a binary cross-entropy loss function. This function evaluates the difference between the expected result and the output of the network interpreted as probability distributions, for a single step of an operation. Its expression is $H(p, q) = H(p) + D_{KL}(p||q)$, where $H(p)$ is the entropy of the expected output, and $D_{KL}(p||q)$ is the Kullback-Leibler divergence between the expected output and the given output. By minimizing this value, the cross-entropy of these distributions decreases, hence leading to an improvement of the results.

B. Active learning

Each of the arithmetic operation relies on the same network weights for its learning (yet we will see in section VI that when correctly trained some parts of the network tend to specialize). Our experiments showed that some learning steps are more demanding and generate more failures than others if not accurately managed. In order to compensate this phenomenon and stress the network adequately, we introduced an active learning strategy. We choose to dynamically balance the learning effort and feed the network with steps whose accuracy is the lowest.

After each training epoch, the efficiency of the network is evaluated for each step. Then a new epoch is generated, with

the different steps not equally represented. The higher the error rate on a step is (when compared to the others), the more the step is represented on the next dataset.

Let B be the function that transforms each subvector of a map into a 1-hot vector:

$B : [0,1]^{s\times l\times 10} \rightarrow \{0,1\}^{s\times l\times 10}$

$$B(\begin{bmatrix} \sigma_{1,1} & \cdots & \sigma_{1,n} \\ \vdots & \ddots & \vdots \\ \sigma_{n,1} & \cdots & \sigma_{n,n} \end{bmatrix}) = \begin{bmatrix} Bin(\sigma_{1,1}) & \cdots & Bin(\sigma_{1,n}) \\ \vdots & \ddots & \vdots \\ Bin(\sigma_{n,1}) & \cdots & Bin(\sigma_{n,n}) \end{bmatrix}$$

$$error_{step} = \frac{E_j[y^j_{step} != B(\hat{y}^j_{step})]}{\sum_s error_s}$$

The probability to educate the network for a step $step$ is then defined by $P_\theta(step) = \lambda \times error_{step} + (1-\lambda)u$, with λ the rate of curiosity and u the uniform probability. The operands are selected with a uniform distribution for each step.

V. Experiments

A. Comparative results

We take as a basis for comparison the end-to-end network presented in [5]. The neural network is equivalent in size in both cases but the training is applied step-by-step for our approach.

For each operation (addition and multiplication) we compare 3 learning methods on the same network topology and with the same loss function. The first training is a classical supervised framework which corresponds to what was done in [5]. The second is our proposition of supervised sequential learning on the unfolded algorithm (see section III-B). The last one is the same while also using the active learning mechanism (see section IV-B). All the results are shown in I. The error rate reported is the one corresponding to the whole operation, meaning that for our models the test will be considered as failed if any of the successive steps is wrong.

Our sequential learning procedure alone improves the result for the addition, even if already good without, but completely fails to train the network on the multiplication task. This drop of performance may be due to the fact that our sequential training forces the network to explicitly learn different single digit operations (addition and multiplication) and that they are not balanced in the dataset. Single digit additions take many inputs (many lines to sum for large multiplications) and are more demanding for their learning than single digit multiplications to perform the global multi-digit multiplication.

Using the active learning mechanism helps to automatically balance the number of single-digit addition and multiplication given to the network. This helps the network to learn the single digit addition which otherwise systematically fails (see figure 6). This also improves the network performance of the single digit multiplication. This leads globally to a high enhancement of the performance for both tasks (see table I). Thus, the network performs significantly better than when trained with the classical supervision protocol, which validate our hypothesis.

	addition (1 hidden layer)	**multiplication (3 hidden layers)**
end-to-end operation	1.7%	37.6%
step-by-step (no active learning)	0.25%	95.5%
step-by-step (with active learning)	**0.01%**	**2%**

TABLE I: The error rates for multiplications, with $s = 7$ maximum digits numbers, using 1-hot vectors encoding.

B. Active learning strategy

The active learning strategy greatly improves the training, and is required for multiplications that is supported by addition and multiplication elementary operations. Figure 6 stresses the importance of rebalancing the learning. The addition elementary operations are more demanding, indeed they apply as a multi-line addition for 7-digit multiplications which corresponds to a more complex task to learn.

C. Neural network robustness

In this section, we are interested in how the network makes use of the input data for the multiplication task. While some resolution steps could be inferred end to end directly from the initial operands, we look at how the network is sensitive to relevant digits which ones are protected from the random noise applied to the input. We want to study if the trained network inference is only based on the relevant digits to process the current step or also takes information from other digits, as the operands e.g. .

For that purpose, we measured the mean performance of the network when randomly corrupting some percentage of the irrelevant digits. In figure 7, 100% of the digits unnecessary for the resolution of the current step are noisy.

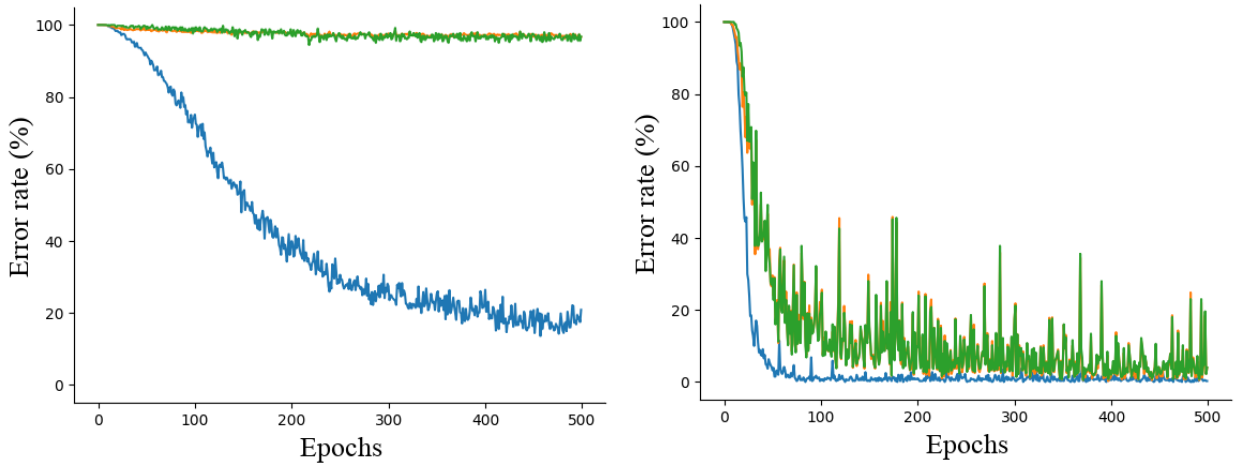
Nevertheless, the network manages to produce the right output associated with the current resolution step. In table II we vary the amount of noise applied to irrelevant digits. In table II we reported the mean performance over 500 000 trials of the network with respect to the percentage of corrupted digits for the 35 elementary operations. We can observe that the network may rely on the relevant digits as most of the obtained performance is conserved even when all the unnecessary digits are modified. However, the network yet takes some information from the other digits as the performance drops monotonously with the number of fake digits.

Fading rate	0%	25%	50%	75%	100%
Accuracy	99.97%	89.9%	82.4%	77.0%	72.7%

TABLE II: Evolution of the accuracy of the network trained with our method including active learning depending on the percentage of fake digits in the input (relevant digits for the current step are maintained)

VI. Discussion: self-organized structure

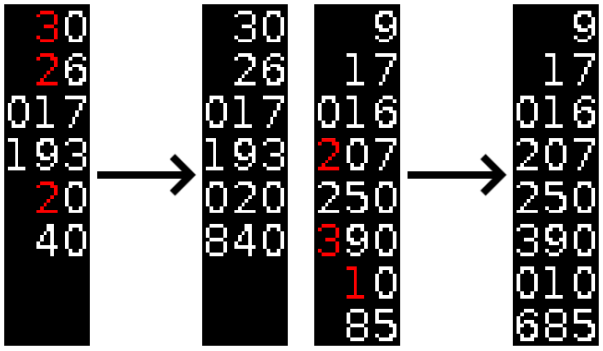
In this paper, we break the algorithmic learning down, step-by-step, promoting the simplicity of the network. One can



(a) no active learning

(b) with active learning

Fig. 6: Comparison of the evolution of the error rates on the multiplication operation over training epochs when using or not the active learning mechanism. Respectively, the blue and green curves stand for the mean failure rate for single digit multiplications and additions. The red curve represents the global efficiency of the step-by-step learning which globally follows the worse performance curve (an error in a single step of the algorithm usually leads to a global false result).



(a) Multiplication step: the network successfully outputs that 2x3 +2 is equal to 8.

(b) Final addition step of a multiplication: the network successfully outputs that 2+3+1 is equal to 6.

Fig. 7: Inferences with fake inputs. The meaningful digits are maintained (highlighted in red). Other digits are deteriorated (100% in this case). The accuracy is preserved on this instance

wonder about the ability of a simple MLP to organize its knowledge when no topology is pre-defined. In this section, we are particularly interested in the way the learning can gather some knowledge on similar subparts of the network, when they are useful at different stages of resolution. For the purpose of comparison, a child would learn times tables once for all, and would apply them whenever necessary. Similarly, we choose to focus on times tables learning, with additional carry considered (indeed we defined in the dataset the multiplication of two digits and the addition with the relevant carry as an elementary step of the global multiplication).

Figure 8 shows the mean activation of the neurons situated on the last layer for a dedicated elementary operation. One can see that some neurons are more involved in the processing of these numbers. We chose $5 \times 3 + 1$ as an illustrative example, we simulate configurations dedicated to the combination whatever the place through a multiplication, which correspond to different inputs for the network. We then measure the activity of neurons in the last layer.

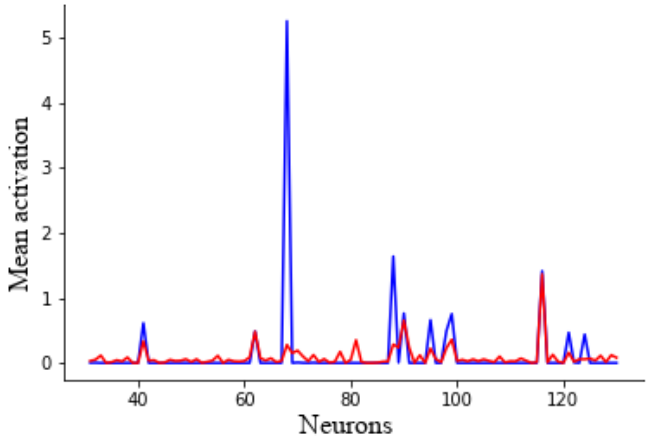


Fig. 8: Partial heat map to illustrate the activity on the third layer of the network. Blue and red curves represent respectively the average activation of neurons for dedicated digits at various places (here $5 \times 3 + 1$) and the regular case (no constraints on considered digits).

We measure the drop of the network performance when we deactivate some neurons among the most activated ones, results are shown in table III. The table shows that when neurons highly activated by this specific elementary operation are removed, the specific operation is more damaged than those not targeted. 5 neurons among 250 lead to a 48.8 points

drop in performance when it only decreases efficiency by 9 points for not targeted operations in average. These results give a first glimpse of the network structuring, including a form of information processing pooling for similar tasks, tending to show that some neurons encode a specific single digit operations of some operands, invariant to its location.

	random digits	**specific digits**			
	mean	**mean**	**standard deviation**	**min**	**max**
no neuron dropped out	99.9%	99.9%	0.17%	99.5%	99.9%
1 neuron dropped out	98.5%	85.6%	22.4%	43.9%	99.9%
3 neurons dropped out	95.8%	67.5%	36.9%	11.8%	99.5%
5 neurons dropped out	91.0%	51.2%	40.9%	2.8%	97.3%
10 neurons dropped out	72.3%	24.8%	34.1%	0%	83.0%
15 neurons dropped out	50.2%	10.3%	19.1%	0%	49.7%
20 neurons dropped out	33.3%	3.3%	6.8%	0%	18.1%

TABLE III: Accuracy of the model trained on the global multiplication operation with our sequential learning when some neurons of the last layer are deactivated, depending on if they are related to internal representation of the targeted single-digit multiplication (specific digits column) or not (random digits column). The reported results are averaged over different targeted multiplications.

VII. Conclusion

Deep learning models draw their efficiency from continuous transformations which make possible to learn a non-linear mapping of the input space to the output space. One can question their capability to extend their ability and learn an algorithmic-like data manipulation end-to-end when the number of elementary operations becomes significant, as is the case for multiplications involving multi-digit numbers. In the literature, people often propose to extend the network with internal memories and reading/writing capabilities.

We propose an alternative approach to improve trainability of models: to take advantage from intermediate algorithmic results when available and to mimic the algorithm unfolding. In order to make the resolution steps explicit, our model operates reading and writing on an external support. We also propose an active learning mechanism to balance the learning effort over the algorithm steps which can be of different nature.

Our experiments on multi-digit addition and multiplication operations show that with such a step-by-step learning and active learning, the multiplication processing is much more efficient when compared to a similar network trained with classical end-to-end supervised learning. Our experiments show that the network is able to some extent to select relevant data coming from the reading space in order to write the result of the current operation to the appropriate location. While no structure imposed, as the network used is a fully connected MLP, some sub-parts of the network seem to be naturally more active to encode some specific single-digit multiplication independently of their location in the global operation.

There is still a long way to go to understand the dynamic of the learning for the new introduced framework. If a form of emergent pooling is confirmed, it would be very challenging to guide it in order to transfer the knowledge for more and more complex algorithm tasks, as a curriculum learning would do. Another more hybrid way would rely on taking benefit from the intermediate supervision signal when using a more sophisticated model. It would also be very interesting to deepen the suitability of models toward more and more algorithmic oriented problems. More work is necessary to fully understand the influence of our training procedure on the structure learned by the network. It will be interesting to see how this learned structure can be extrapolated to operations with any number of digits as input. We also want to study if and how our unfolding of algorithms can be applied to more complex, e.g. hierarchical, tasks.

Acknowledgment

We gratefully acknowledge the support of @NVIDIA Corporation with the donation of a GPU on which the experiments of this paper where performed.

References

[1] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, p. e253, 2017.

[2] H. Siegelmann and E. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132 – 150, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000085710136

[3] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014. [Online]. Available: http://arxiv.org/abs/1410.5401

[4] M. Collier and J. Beel, "Implementing neural turing machines," in *Artificial Neural Networks and Machine Learning – ICANN 2018*, V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, Eds. Cham: Springer International Publishing, 2018, pp. 94–104.

[5] Y. Hoshen and S. Peleg, "Visual learning of arithmetic operations," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 3733–3739. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016387.3016429

[6] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[9] 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," 2014.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[13] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[14] S. Sukhbaatar, a. szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2440–2448. [Online]. Available: http://papers.nips.cc/paper/5846-end-to-end-memory-networks.pdf

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[16] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://www.aclweb.org/anthology/D14-1179

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[18] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom, "Neural arithmetic logic units," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 8035–8044. [Online]. Available: http://papers.nips.cc/paper/8027-neural-arithmetic-logic-units.pdf

[19] A. Madsen and A. R. Johansen, "Neural arithmetic units," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1gNOeHKPS

[20] K. Chen, Y. Dong, X. Qiu, and Z. Chen, "Neural arithmetic expression calculator," *CoRR*, vol. abs/1809.08590, 2018. [Online]. Available: http://arxiv.org/abs/1809.08590

[21] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009. [Online]. Available: http://axon.cs.byu.edu/ martinez/classes/778/Papers/settles.activelearning.pdf

[22] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv preprint arXiv:1808.04355*, 2018.

[23] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48.

[24] J. Gottlieb, P.-Y. Oudeyer, M. Lopes, and A. Baranes, "Information-seeking, curiosity, and attention: computational and neural mechanisms," *Trends in Cognitive Sciences*, vol. 17, pp. 585–593, 2013.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980