Algorithmic learning, a next step for AI. An application to arithmetic operations

Frédéric Armetta, Anthony Baccuet, Mathieu Lefort

Algorithmic learning, a next step for AI. An application to arithmetic operations

Frédéric Armetta[1], Anthony Baccuet[1] and Mathieu Lefort[1]

[1]*Univ Lyon, UCBL, CNRS, INSA Lyon*
LIRIS, UMR5205, F-69622
Villeurbanne, France

Abstract

Deep learning achieved state of the art performances in multiple domains (image recognition, natural language processing, etc.) One of the next steps is to be able to learn algorithms, as a way to provide some new forms of generalization for AI systems. This is currently a hard and challenging problem as it involves algorithmic recurrence, memory management and combination of subtasks, which leads to trainability problems. We show that even a simple algorithm of multiplication manifests trainability problems for neural networks. In this article, we present an original training method applied to multi-digit multiplication learning, called Unrolling Algorithmic Training (UAT). To learn the global algorithm, we use additional supporting tasks consisting in the successive subtasks composing the global algorithm (in our case the 1-digit multiplications and the final addition). The global end-to-end and the subtasks learning are then balanced with an active learning mechanism. This multi-task learning allows to overcome the problem of trainability encountered when learning directly the global algorithmic task. Our interpretation is that the network is able to somehow combine the subtasks in order to learn the global task. Moreover, we show that the global algorithm can be bootstrapped, fine-tuned and is even resilient without retraining it from scratch when we vary the size of recurrence provided to the network.

Keywords

Algorithmic Machine Learning, Deep Learning, Active learning

1. Introduction

Since the first success of convolutional neural networks on image classification [1], deep learning methods have pushed forward the state of the art in multiple domains [2]. This flows in tasks of increasing complexity such as language translation [3] or game playing [4]. One of the next steps is algorithmic learning, including mathematical expression calculation, as it will open the way to learn any task that can be expressed in a Turing machine and so to provide greater autonomy to AI systems.

Algorithmic resolution through neural networks is still an emerging area of research. It is a challenging problem as it requires to memorize values for a long period of time, to learn inferences, to combine procedures, to extrapolate to unknown domains, etc. More fundamentally this needs to fill the gap between symbolic understanding and statistical learning. Neural Turing Machine [5] was proposed to learn end-to-end algorithmic procedures, such as list sorting. It

frederic.armetta@univ-lyon1.fr (F. Armetta); anthony.baccuet@gmail.com (A. Baccuet); mathieu.lefort@liris.cnrs.fr (M. Lefort)

aims to reduce the trainability problem faced by RNN, which are Turing complete [6], by adding specific mechanisms such as a memory and differentiable ways to access it. However, it also suffers from the trainability problem itself, i.e. the learning procedure is very sensitive to the hyperparameters and/or to the initial values and may lack generality, so that the performances are hard to reproduce and inconstant [7]. These problems are due to the depth of the proposed architecture [8] but also to the intrinsic complexity of the operations to learn, especially the long-term dependencies between data or variables to manipulate.

To overcome this trainability problem for an algorithm learning, an alternative way is rather to decompose this algorithm into its successive steps so that the neural network will compute and learn iteratively by feeding back its previous output thanks to a handmade recurrence external to the model. This procedure was applied to a transformer-inspired architecture for learning some algorithmic procedures [9] and to a MLP for arithmetic operations [10, 9]. In this paper, we aim to address the trainability problem end-to-end.

We choose to focus and illustrate our proposal on arithmetic operations, especially the multi-digit multiplication algorithm, which requires multiple one-digit multiplications to add properly in order to get the final result. This computation involves many operations and variables in the calculation stream. The complexity for such an operation is also related to the propagation of the carry e.g. [11]. Thus, learning directly the end-to-end multiplication leads to poor performances [12]. To tackle this trainability problem, we propose to mix during learning both intermediate sub-tasks and the global end-to-end multiplication, weighted by an active learning strategy. This is somehow similar to a multi-task learning, using intermediate computation as relevant complementary tasks to help the network to learn the global multiplication. However, with our approach, the network will perform all these tasks without any layer dedicated to each one. Thus, the network will have to accommodate intermediate operations in order to learn the global multiplication. Another difference is that this accommodation has to follow the course of the algorithm, by sequentially connecting tasks and propagating intermediate values through recurrences. We will show that this training procedure can also be used for pre-training the network, as a kind of bootstrap, for an efficient fine-tuning relying on the algorithmic target only (the multiplication).

We present the related work in section 2. We show in this section that language models such as chatGPT are also highly concerned by algorithmic learning, and suffer from trainability problems in the same way. This can be observed when the query involves lengthy algorithmic reasoning, as for arithmetic operations. We detail the use case of multi-digit multiplications in section 3. The chosen task is the decimal multi-digit multiplication, that is one of the hardest of the four arithmetic operations as it implies a lot of steps. Then, the model on which our learning procedure is applied is detailed in section 4 and next the protocol and results are presented in section 5. We conclude and discuss the perspectives of our work in section 6.

2. State of the art

Over the years, multiple deep learning models have been proposed to solve different kinds of problems [2]. Convolutional neural networks [1] are able to classify images with performances sometimes overcoming the ones of humans in specific scenarios. Recurrent neural networks

with dedicated cells such as Long Short Term Memory (LSTM) [13] or Gated Recurrent Units (GRUs) [14] can deal with temporal data.

Deep learning models are universal function approximators [15] but they are often hard to train. For instance, in theory even a shallow architecture is sufficient, but in practice, the ability of deep networks to learn relevant representations from data is much better [16]. Moreover, RNNs are Turing complete [6], but also face a trainability problem [5]. This limitation tends to be more pronounced for complex tasks, especially when long-term inferences are needed. As an example, the error rate of an arithmetic operation is related to the number of carries for a MLP [11].

Some global strategies were proposed to make network training easier and more effective. Among them we can mention active learning strategies that consist in finding the right next example to improve the training progress [17, 18]. A common way to do it is curriculum learning where tasks are ordered by increasing complexity [19]. Another strategy is multi-task learning that consists in combining different tasks with similar objectives so that the network can better generalize and learn the underlying structure of data [20]. Even if our proposal appears similar to multi-task learning, in our model the tasks are taken on the same network and are of algorithmic nature.

Neural Turing Machine (NTM) [5] was specifically designed to learn algorithmic tasks. To limit the problem of long time dependency encountered by LSTM, it mimics some of the principles of a Turing Machine by introducing a memory and reading/writing mechanisms. Differentiable Neural Computer [21] improves these accesses to learn more complex tasks as some NLP problems of inference and reasoning. However, these models are difficult to train as they seem to be very sensitive to initial weights [7, 22]. The neural GPU architecture [8] shares similar ideas with NTM but uses convolutional GRU in order to obtain parallel computing. It is able to learn some algorithmic tasks such as binary addition and multiplication, sequence reversing, etc. Its main achievement is its ability to generalize the learning to inputs with longer size in testing, e.g. from 20-digit binary multiplication in learning up to 2000-digit ones in testing, without any error. However, this model completely fails to learn decimal multiplications [23]. This limitation may be related to carry propagation that is hard to train and appears more frequently with decimal coding of numbers and can be partially overcome by using curriculum learning (from binary to quaternary then to decimal) [24].

In [22], multiple algorithms (LSTM with or without attention, transformers) were tested on various mathematical inference tasks. The main objective was to propose a dataset and to compare the models, especially on the attentional aspect, so once again detailed performances are missing. One of the conclusions is that long-term dependencies are the harder to learn which can be compromising for the addition of multiple operands in specific cases. A transformer with operands given as variables in the text, achieves good performance except for subtraction and multiplication [25]. [26] solves mathematical equations, including differential ones, with Seq2Seq models. The originality lies in the equation being written as the prefix notation of its tree representation. Another approach proposes to solve arithmetic expressions by composing single-digit sub-tasks. This hierarchical combination is learned by reinforcement learning with curriculum [27]. The system manages to generalize to some extent to longer sequences, but the performance sometimes drops, especially for multiplication. Neural Arithmetic Logic Units [28] are cells that are able to perform arithmetic operations by introducing specific computation

modules such as log, exp, etc. The aim here is not to learn arithmetic *per se* but to provide dedicated cells able to extrapolate learning with computations that extends to unknown domains. A direct extension dealing also with negative inputs was proposed in [29]. Other derived models can compute arithmetic operations on real numbers [30]. As we saw, there is a huge variety of tasks that was explored in the literature, so that comparing different approaches is difficult.

Algorithmic reasoning is also part of the required material for Natural Language Processing. Thus, large Language Models which have been extensively developed in recent years, need to learn reasoning to answer appropriately. They can for instance learn to perform numerical reasoning when learning few examples in a few-shot setting (using a query such as "Q: What is 24 times 18?", taking GPT-J-6B as a base). It was shown that the performance is highly correlated with the frequency of the terms in the pretraining corpus [31]. When co-occurrence of terms is low, accuracy is also low. These observations underline the trainability problem and lack of generalisation to algorithms learning. As a consequence, performance decreases as the size of operands increases [32]. Moreover, while addition seems to be less impacted by the size of operands, performance collapses for the multiplication which requires more reasoning or algorithmic processing. GPT-3 is among the most popular and large language model, using about 500 billion tokens for learning [33]. ChatGPT which is fine-tuned from GPT-3 also suffers from the limitations of GPT-based models. These observations suggest that model reasoning skills for such models are still limited. They highly rely on the size of the corpus available and statistics. The corpus will never contain all the combinations of terms or parameters of algorithms we have to learn. A better approach is then to enhance the ability to learn algorithms, which would allow to make better predictions for unknown configurations.

In this article, we choose to focus on the multi-digit multiplication of two decimal numbers. This is a simple enough task to not mix different problems but in the meantime it is challenging as multiple models are unable to learn it properly. This difficulty arises from long-term dependency due to the carry propagation, but also and more generally from the inherent complexity of algorithms which involves many operations and variables in the calculation flow.

Moreover, some articles precisely measured performance on this task. [12] proposes a MLP to learn addition and multiplication either from visual inputs or numerical encoding. In both cases, the accuracy achieved for multiplication is poor (see section 5).

3. Problem statement

In this article we will consider the decimal multi-digit multiplication. Formally, let n be the (maximum) number of digits of any of the two operands. The multiplication of these two operands leads to $n + 1$ sub-tasks: n single-digit multiplications and 1 final addition of the partial multiplications (see figure 1 for an example). In our data representation, each operation will correspond to two lines of computation: one for the carries and one for the result. The maximum size of any intermediate operation is $N = 2n$ (this maximal length will be obtained for the final addition with a carry generated at the most significant digit position). All the operands will be padded with zero digits to match a N size, yet there will be exactly $n + 1$ intermediate operations even if the first operand has less than n digits.

$$
\begin{array}{rl}
0023 & (1)\\
\times 0048 & (2)\\
\hline
0012 & (3)\\
0184 & (4)\\
0010 & (5)\\
+0920 & (6)\\
\hline
0110 & (7)\\
1104 & (8)
\end{array}
$$

Figure 1: Example of the representation of a multiplication of two 2-digit operands. Note that the signs (+ and $\times$) and lines are only shown for clarity. Lines (1) and (2) are the two operands. Lines (3) and (4) (respectively (5) and (6)) represent the carries and result of the 1-digit multiplication of 8 (respectively 4) by 23. Lines (7) and (8) correspond to the carries and result of the addition of lines (4) and (6), which is also the final result of the global multiplication of 48 by 23, i.e. 1104.

4. Model

Our model is very similar to networks used for natural language processing. We choose a recurrent network in order to provide a potential for algorithm unrolling. We use an agnostic encoding for digits (no binary encoding) in order to prevent any facilities associated with the selected problem and focus on the algorithmic concerns.

4.1. Data representation

Each digit d is represented by a 10-dimension vector with a one-hot encoding, i.e. $(\delta_{di})_{i\in\{0,\dots,9\}}$. This vector can also have two other values. First, a null vector represents an empty value in the input to form empty lines (see table 1). Note that it will also be used as the starting character for the decoder presented below. Second, a one vector corresponds to the end of the line either when reading or writing the data.

4.2. Model architecture

We use a Seq2Seq model as proposed in [34] (see figure 2). It is composed of an encoder that will sequentially read the data and recurrently embed it in some hidden states. From it, a decoder will iteratively produce a sequence from previously outputted characters, beginning with some predetermined code, until an ending character is written. Both encoder and decoder are implemented with an LSTM neural network model. During training, we used teacher forcing, i.e. the next character is not produced from the previous output but from the ground truth. The current output is then compared to the expected one and the error is backpropagated through the decoder and the encoder.

A headband allows to read and write successive lines two at a time, digit by digit. The digits, encoded in one-hot vectors, are then read from right to left, with the <eol> vector at the end of each line. Similarly, encoded one-hot vectors are written at the outputs. Both the inputs of the encoder and output of the decoder are 2×10 in size. Each task can then be encoded individually in order to infer the ground truth value (see below).

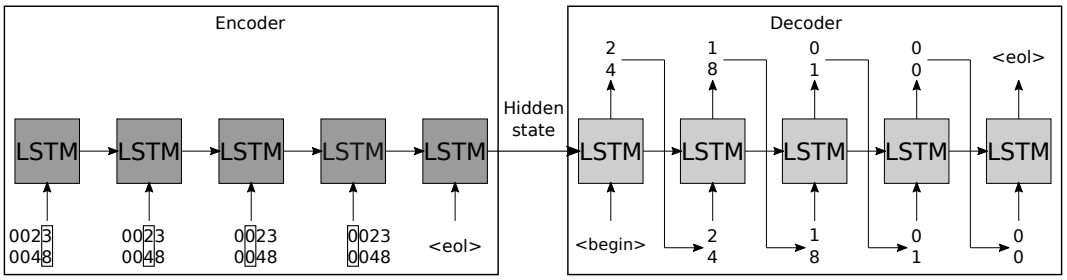

Figure 2: Seq2Seq architecture. The encoder receives successively the two digits (here boxed) of two lines (here the operands 23 and 48) from right to left. From the encoder embedding, the decoder recurrently produces the digits, from right to left, of the two next lines (here the carry and result of the first intermediate multiplication 8×23). In practice each digit is encoded as a one-hot vector (see section 4.1).

4.3. Tasks definition

The multiplication algorithm involves sub-tasks that can be submitted to the model (see table 1). The encoder records the formulated task, the decoder outputs the target value. The multiplication algorithm can also be inferred end-to-end in the same way. As formalized in section 3, the multiplication operation of two n-digit numbers can be decomposed into n single-digit multiplications and 1 final addition. These n+1 operations and the global end-to-end multiplication (ie computing the final result and carries from the two operands) compose the n+2 tasks that the network has to learn. For each task, the encoder reads the corresponding inputs and the decoder outputs the target value of the corresponding operation as illustrated in table 1. So that the network can differentiate the task of the first single digit multiplication and the end-to-end multiplication, which both require only the two operands, in the latter case we add 2n empty lines (corresponding to unfilled intermediate results lines).

Task	Encoder inputs	Decoder output
st1	(1//2)	(3//4)
st2	(1//2) (3//4)	(5//6)
st3	(1//2) (3//4) (5//6)	(7//8)
end-to-end	(1//2) (empty line $*$ 4)	(7//8)

Table 1
Sub-tasks and end-to-end inferences associated with a 4-digit multiplication (see numbered lines in figure 1, lines are read and written two at a time ("//"))

4.4. Active learning mechanism

We have so far described several tasks. One can choose to select a bundle of tasks to train concurrently on the model. In this article, different settings are compared (sub-tasks only, all the tasks, end-to-end only) for training.

In order to balance the learning effort between the chosen bundle tasks, in all cases we use the same active learning mechanism as proposed in [10]. It consists in measuring the error rate, denoted err_{task}, of each involved task in the training dataset at the end of each training epoch. For the next epoch, the training dataset is constructed by randomly picking up examples from a global fixed set of examples so that each task is present with a fraction $F_{task} = \lambda \frac{err_t}{\sum_{ta \in taskList} err_{ta}} + (1 - \lambda) \frac{1}{card(taskList)}$ where λ is an hyperparameter and *taskList* is the list of all tasks involved. The general idea is that the more difficult is a task to learn (first term), the more it is present in the next epoch with a lower bound depending on the λ value.

5. Experiments

5.1. Protocol

For training, 100000 unique couples of operands are generated. At each epoch, in order to update the dataset, we generate and attribute an operation to each of the couple so that the global distribution between operations matches the one decided by the active learning mechanism presented in section 4.4.

For the validation and test datasets, we use respectively 1000 and 10000 additional unique couples of operands. The size of the latent space of the encoder is set to 500 and the active learning parameter λ to 0.5. The model is trained using the ADAM optimizer with a learning rate of 10^{-4} and a batch size of 10. All the results presented in the next sections are averaged over 4 runs that learned during 500 epochs.

5.2. Results

	test = end-to-end		
	3×3 (6 output digits)	restricted 4×4 (7 output digits)	4×4 (8 output digits)
Hoshen et al.[12]	n.c	37.6 %	n.c.
UAT (train = end-to-end only)	4.05% (± 1.72%)	35.87% (± 28.10%)	92.42% (± 3.64%)
UAT (train = sub-tasks and end-to-end)	3.33% (± 1.32%)	**4.51% (± 1.21%)**	23.34% (± 14.69%)

Table 2
Error rate on the end-to-end multiplication task

In this section, we want to quantify the effect of our proposition to mix the learning of the sub-tasks with the global end-to-end multiplication. For the purpose of comparison, we

reproduce 4×4 multiplications, as presented in [12] and [10], with 4-digit inputs chosen so that the final outputs are restricted to 7-digit numbers.

5.2.1. Sub-tasks and end-to-end

The main purpose of this paper is to tackle the trainability problem encountered for the end-to-end multiplication task we choose as a first step towards general algorithmic learning. For the so considered bundle of tasks, sub-tasks and the end-to-end task are trained simultaneously on the same network following the active learning mechanism. We report in table 2 the performances for the end-to-end task, and compare with [12] which uses an MLP while we use a Seq2Seq. We also vary the scale and complexity of problems in order to show the improvements of our proposal.

Results clearly demonstrate the contribution of sub-tasks for the end-to-end task, lowering the error rate (from 35.87% to 4.51 % for the 4×4 restricted multiplications). This suggests that the model is able to self-organize and take advantage of sub-tasks, which facilitate the training, validating that our proposed training procedure is the key element of our model. While results are similar without the support of sub-tasks, i.e. both architectures face trainability problems, our training approach clearly outperforms [12]. The standard deviation reduction shows that stability of learning is also improved (with the exception of UAT end-to-end only 8 output digits that can be excluded from the comparison because of homogeneous but poor efficiency).

We report below other significant improvements for the most difficult problem presented (the 4×4 (8 output digits) problem) thanks to fine tuning and additional recurrences.

4x4 (8 output digits)	error rates
initial UAT train = sub-tasks + end-to-end	23.34% (± 14.69%)
UAT fine tuning (pre-training = sub-tasks + end-to-end)	**6.68% (± 4.31%)**
UAT fine tuning (pre-training = sub-tasks)	73.31% (± 11.10%)

Table 3
Influence of fine tuning

free double lines	error rates
0	88.67% (± 5.03%)
1	50.19% (± 18.16%)
2	14.39% (± 2.65%)
3	4.44% (± 1.65%)
4	6.68% (± 4.31%)
5	5.53% (± 2.86%)
6	3.84% (± 1.61%)
7	**3.83% (± 1.29%)**

Table 4
Influence of the recurrence provided to the model when fine tuning (pre-training = sub-tasks + end-to-end, 4 free double lines)

5.2.2. End-to-end only (fine tuning)

To improve further the performance of our model, we propose to fine tune the end-to-end task only, during 500 additional epochs, for the hardest 4×4 problem. As we can see on table 3, when keeping the same recurrence to the network, the fine tuning leads to a significant improvement of the performance as the error drops from 23.34% to 6.68% for the 8 output digits problem. This result also shows that once the end-to-end task is bootstrapped (by simultaneous learning with subroutines), its performance can be further improved. On the contrary, table 3 shows that not including the end-to-end task in the pre-training bundle leads to trainability problems. This confirms that the model has to learn simultaneously the intermediate operations and the end-to-end one to overcome the trainability problem.

5.2.3. Adaptability

In our model, the input of the end-to-end multiplication is composed by some empty lines to match with the input size of the final addition subroutine (see table 1). These empty lines provide recurrence steps for the network (encoder). To test the influence of this factor, we ran multiple fine tuning of our model, setting each with a different number of empty lines (initial training done with 4 double lines).

We can observe on table 4 that the error rates tend to decrease monotonically with the number of free recurrences. This may sound logical as increasing the number of recurrences also increases the computational power of the model.

5.2.4. Active learning dynamic

In section 4.4, we introduced the active learning mechanism that we used to balance the learning effort between the different type of operations involved. In figure 3 is represented the evolution of their distribution in the training dataset. We can observe that it is relatively constant after some time and that the hardest task is the end-to-end one. However, one can notice that the second hardest task is the addition procedure and not any of the intermediate multiplication, whose mean error rate is even close to 0. This is surprising as in the literature the addition appears to be a simpler task to learn. This may be due to the fact that in this case the addition involves many operands.

5.2.5. Sub-tasks only

We will train the network only on the sub-tasks (ie excluding the end-to-end operation). To obtain the global result, we will provide the network with the successive intermediate tasks in the right order (with intermediate lines filled iteratively with previous outputs of the network) and consider only the output of the final addition as the global result. This method was also used in [9]. This task is easier to process (as the global operation is decomposed in its successive steps for its execution). Especially, we aim to estimate the effect of the recurrence available in our network knowing that [10] uses a network without recurrence.

Table 5 shows that our model reduces the error rate, in this context of execution. This shows that the recurrent model and data representation we use succeeds in capturing all the

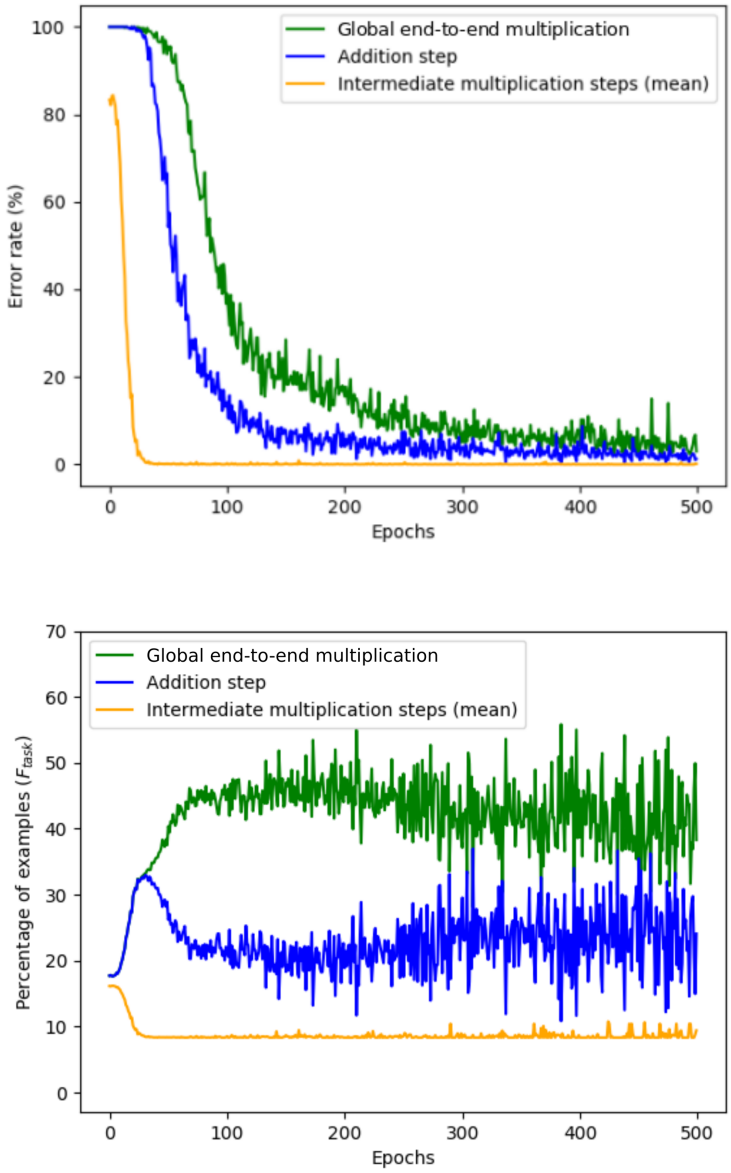

Figure 3: (Top) Evolution of the error rate for each kind of operation (estimated from the training dataset) used by the active learning. (Bottom) Resulting evolution of the distribution of the various tasks in the training dataset due to active learning

subroutines.

	test = *handmade* recurrence
	restricted 4×4 (7 output digits)
[10]	2 %
UAT	**0.31% (± 0.11%)**

Table 5
Error rate comparison between [Nollet et al., 2020] and UAT, when combining sub-tasks only thanks to an handmade recurrence

6. Conclusion and perspectives

Despite its multiple successes, deep learning architectures still struggle to learn algorithms, such as arithmetic operations. Algorithmic learning is highly expected as it would allow new classes of problems to be addressed, such as algorithmic generalisation. This is nevertheless a hard task, due to the complexity of the learning, especially the long-term dependencies, but also a trainability problem faced by multiple models. This limitation can as well be observed for language models whose performances come from the corpus available, and do not apply well to algorithmic inferences when variability in parameter values is large and usual generalisation do not apply as observed for arithmetic calculations, which underlines the need for new inference mechanisms.

In this article, we propose an original way to learn the end-to-end multi-digit multiplication of two (decimal) operands by guiding the model via the introduction of sub-tasks (or subroutines) concurrently with the targeted end-to-end task while applying an active learning strategy. While in the literature the addition appears to be a simple task to learn, multiplication is especially challenging. We show through analysis of our experiments that the multiplication can nevertheless be learned end-to-end, we measure improvements directly caused by UAT, the learning procedure we introduce. Learning the target end-to-end task after the sub-tasks can improve the results but the best way, by a large margin, is to mix all the tasks together during training in order to bootstrap the target end-to-end training task. Once bootstrapped, the targeted task can efficiently be fine-tuned alone. This process can be described as a new algorithmic transfer, i.e. the tasks complement and support each other to achieve the overall arithmetic task.

By fine tuning the model on the final end-to-end task we show an interesting additional property of our learning method. Once the end-to-end multiplication learned, the network can adapt to a number of recurrence different from the one it was first been trained for. This seems to indicate that the model is able not only to combine the sub-tasks to resolve the global end-to-end task but also to autonomously extract and to adapt some kind of high level algorithmic knowledge from it. Restricting the provided recurrences for the computing and maintaining accuracy looks like a constrained parallelization of the algorithmic task.

The main motivation for this work is not only the learning of the multiplication, but to provide a new method for alleviating the hard trainability problem that is observed for algorithms. This work raises multiple research questions. We want to investigate more precisely how the transfer from intermediate steps to the global task is achieved by the network. To overcome

the trainability problem encountered by the classical training procedure, we provide all the intermediate steps to the network during learning. A specific case that we want to study is to provide only some of the supporting tasks and see if the network can complement the unknown tasks by itself. For that we will investigate how the transfer from intermediate steps to the global task is achieved in the network. This understanding may give us a way to control the flow of interaction between supporting steps and the general dynamic of the algorithmic transfer, and thus get a more detailed understanding of how an AI could exploit and adapt its algorithmic knowledge in order to expand its capabilities.

Acknowledgment

This work was performed using HPC resources from GENCI–IDRIS and a GPU donated by @NVIDIA Corporation. We gratefully acknowledge this support.

References

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

[2] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, volume 1, MIT press Cambridge, 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[4] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, L. Zitnick, Elf opengo: An analysis and open reimplementation of alphazero, in: International conference on machine learning, PMLR, 2019, pp. 6244–6253.

[5] A. Graves, G. Wayne, I. Danihelka, Neural turing machines, CoRR abs/1410.5401 (2014). URL: http://arxiv.org/abs/1410.5401. arXiv:1410.5401.

[6] H. Siegelmann, E. Sontag, On the computational power of neural nets, Journal of Computer and System Sciences 50 (1995) 132 – 150. URL: http://www.sciencedirect.com/science/article/pii/S0022000085710136. doi:https://doi.org/10.1006/jcss.1995.1013.

[7] M. Collier, J. Beel, Implementing neural turing machines, in: Artificial Neural Networks and Machine Learning – ICANN 2018, Springer International Publishing, Cham, 2018, pp. 94–104.

[8] Ł. Kaiser, I. Sutskever, Neural gpus learn algorithms, arXiv preprint arXiv:1511.08228 (2015).

[9] Y. Yan, K. Swersky, D. Koutra, P. Ranganathan, M. Hashemi, Neural execution engines: Learning to execute subroutines, CoRR abs/2006.08084 (2020). URL: https://arxiv.org/abs/2006.08084. arXiv:2006.08084.

[10] B. Nollet, M. Lefort, F. Armetta, Learning Arithmetic Operations With A Multistep Deep

Learning, in: The International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020. URL: https://hal.archives-ouvertes.fr/hal-02929738.
[11] S. Cho, J. Lim, C. Hickey, B.-T. Zhang, Problem difficulty in arithmetic cognition: Humans and connectionist models (2019).
[12] Y. Hoshen, S. Peleg, Visual learning of arithmetic operations, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, pp. 3733–3739. URL: http://dl.acm.org/citation.cfm?id=3016387.3016429.
[13] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.
[14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. arXiv:1406.1078.
[15] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (1989) 303–314.
[16] J. Ba, R. Caruana, Do deep nets really need to be deep?, in: Advances in neural information processing systems, 2014, pp. 2654–2662.
[17] B. Settles, Active Learning Literature Survey, Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009. URL: http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.
[18] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, A. A. Efros, Large-scale study of curiosity-driven learning, arXiv preprint arXiv:1808.04355 (2018).
[19] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, ACM, 2009, pp. 41–48.
[20] Y. Zhang, Q. Yang, A survey on multi-task learning, arXiv preprint arXiv:1707.08114 (2017).
[21] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016) 471–476.
[22] D. Saxton, E. Grefenstette, F. Hill, P. Kohli, Analysing mathematical reasoning abilities of neural models, arXiv preprint arXiv:1904.01557 (2019).
[23] K. Freivalds, R. Liepins, Improving the neural gpu architecture for algorithm learning, arXiv preprint arXiv:1702.08727 (2017).
[24] E. Price, W. Zaremba, I. Sutskever, Extensions and limitations of the neural gpu, arXiv preprint arXiv:1611.00736 (2016).
[25] A. Wangperawong, Attending to mathematical language with transformers, arXiv preprint arXiv:1812.02825 (2018).
[26] G. Lample, F. Charton, Deep learning for symbolic mathematics, arXiv preprint arXiv:1912.01412 (2019).
[27] K. Chen, Y. Dong, X. Qiu, Z. Chen, Neural arithmetic expression calculator, CoRR abs/1809.08590 (2018). URL: http://arxiv.org/abs/1809.08590. arXiv:1809.08590.
[28] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, P. Blunsom, Neural arithmetic logic units, in: Advances in Neural Information Processing Systems 31, Curran Associates, Inc., 2018, pp. 8035–8044. URL: http://papers.nips.cc/paper/8027-neural-arithmetic-logic-units.pdf.
[29] D. Schlör, M. Ring, A. Hotho, inalu: Improved neural arithmetic logic unit, arXiv preprint

arXiv:2003.07629 (2020).
[30] A. Madsen, A. R. Johansen, Neural arithmetic units, in: International Conference on Learning Representations, 2020. URL: https://openreview.net/forum?id=H1gNOeHKPS.
[31] Y. Razeghi, R. L. Logan IV, M. Gardner, S. Singh, Impact of pretraining term frequencies on few-shot numerical reasoning, in: Findings of the Association for Computational Linguistics: EMNLP 2022, Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2022, pp. 840–854. URL: https://aclanthology.org/2022.findings-emnlp.59.
[32] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.
[33] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, arXiv preprint arXiv:2203.02155 (2022).
[34] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 3104–3112. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.