Towards a curriculum for neural networks to simulate symbolic arithmetic

Samuel Lozano Iglesias[1,2], Markus Spitzer[3], Younes Strittmatter[4], Korbinian Moeller[4,5], and Miguel Ruiz-Garcia[1,2]

[1]Departamento de Estructura de la Materia, Física Térmica y Electrónica, Universidad Complutense Madrid, 28040 Madrid, Spain
[2]Grupo Interdisciplinar de Sistemas Complejos, Universidad Complutense Madrid, 28040, Madrid, Spain
[3]Department of Psychology, Martin-Luther University Halle-Wittenberg, Halle, Germany
[4]Department of Psychology, Princeton University, Princeton, NJ 08544, USA
[5]Department of Mathematics Education, Loughborough University, Loughborough, UK,
[6]LEAD Geaduate School & Research Network, University of Tuebingen, Germany

Corresponding Authors: *samuel.lozano@ucm.es, markus.spitzer@psych.uni-halle.de*

Abstract

Understanding and operating on multi-digit numbers is a critical step in the development of mathematical skills and concepts. Empirical and computational modeling evidence suggests that multi-digit numbers are processed decomposed into units, tens, hundreds, etc. for magnitude comparison. Yet, there is currently no computational model to simulate multi-digit number arithmetic. Accordingly, we developed a neural network model with three interconnected modules reflecting single-digit additions, recognition of a carry-over, and decision-making. We then compared model performance after following two training curricula: i) a *step-by-step* curriculum, where single-digit addition are trained before progressing to multi-digit problems and ii) an *all-at-once* curriculum, where all modules of the model were trained simultaneously. Our results indicated that only the *step-by-step* curriculum made the model learn multi-digit addition successfully as reflected by replicating empirical effects of carry-over and problem size. These findings highlight the importance of structured, incremental learning in both cognitive modeling and education.

Keywords: numerical cognition; multi-digit addition; carry-over; problem size; neural networks; curriculum learning

Introduction

Understanding how multi-digit numbers are represented and processed is a central question in numerical cognition research, with significant implications for teaching and learning multi-digit arithmetic (Fuson et al., 1997; Nuerk et al., 2015; Yuan et al., 2019). Typically, learning arithmetic operations starts with the addition of single-digit numbers to acquire the concept of addition as an essential building block for generalizing to the addition of multi-digit numbers using standard algorithms. These often reflect column-wise addition of units, tens, hundreds, etc. digits. This progression reflects the increasing complexity of mathematical representations—in particular considering the combination of individual digits according to the place-value structure of the Arabic number system to successfully operate on multi-digit numbers. This emphasizes the importance of understanding the underlying cognitive processes for which computational modeling (e.g., simulating cognitive processes using artificial neural networks) has been applied. However, there is currently no neural network model simulating multi-digit arithmetic.

Accumulating empirical evidence suggests that multi-digit numbers are processed decomposed into their constituting digits (e.g., units, tens, hundreds, etc.). For instance, Poltrock and Schwartz (1984) found that reaction times (RTs) for comparing the magnitude of two numbers increased the further to the right the first unequal digit pair occurred (e.g., comparing 1361 vs. 2361 was faster than comparing 1734 vs. 1739). Later, Nuerk et al. (2001) observed longer RTs and higher error rates for comparisons of two-digit numbers when separate comparisons of tens and unit digits led to incompatible decision biases (e.g., $47 < 92$ with tens $4 < 9$ but units $7 > 2$) compared to compatible decision biases (e.g., $42 < 87$ with tens $4 < 8$ and units $2 < 7$). This so-called unit-decade compatibility effect provides further empirical evidence for the idea that multi-digit numbers are processed decomposed into their constituting digits (Moeller et al., 2012; for evidence on 4 and 6-digit numbers see Meyerhoff et al., 2012).

Importantly, further empirical evidence suggesting decomposed processing of multi-digit numbers extends beyond magnitude comparison to mental arithmetic. For instance, the carry-over effect reflected by longer RTs and higher error rates for additions requiring a carry-over compared to additions not requiring one (e.g., 7 + 8 = 15 vs. 12 + 3 = 15) is inherently based on decomposed processing of units and tens as it is the sum of the unit digits that indicates the need for a carry over to the decades when being 10 or larger (Imbo et al., 2007; Klein et al., 2010; Lambert & Moeller, 2019; Moeller et al., 2011). Interestingly, this also perfectly reflects the typical algorithms for performing additions of multi-digit numbers taught at school.

Building on these and other findings, Verguts and De Moor (2005) proposed that multi-digit numbers are processed by systematically combining representations of single-digit numbers. This perspective was further supported by neural network modeling results on magnitude comparisons.

For instance, Moeller et al. (2011) developed a neural network model for *magnitude comparisons* of multi-digit numbers, which implemented fully decomposed representations of units and tens (also see Huber et al., 2016). The model combining representations of single-digit number magnitudes was the most parsimonious explanation for several hallmark effects in number magnitude comparison, including the unit-decade compatibility effect (cf. Nuerk et al., 2001), the problem-size effect (where comparisons become more challenging as the magnitudes of the numbers increase, e.g., 2 vs. 3 is easier than 92 vs. 93; e.g., Ashcraft, 1992; Zbrodoff and Logan, 2005), and the numerical distance effect (where comparison performance improves with larger numerical distance, e.g., 4 vs. 5 compared to 1 vs. 8; Moyer and Landauer,

In D. Barner, N.R. Bramley, A. Ruggeri and C.M. Walker (Eds.), *Proceedings of the 47th Annual Conference of the Cognitive Science Society*

1967). These findings collectively support the view that multi-digit number processing is fundamentally grounded in decomposed single-digit representations. However, there is currently no neural network model generalising the principle of decomposed processing to multi-digit arithmetic.

Accordingly, this study set out to develop and evaluate a neural network model to simulate multi-digit arithmetic, drawing on Huber et al. (2016)'s concept of fully decomposed representations of single-digit numbers. Inspired by the hierarchical nature of mathematics learning, we trained the model progressively, starting with single-digit number addition before moving to multi-digit addition. We evaluated the proposed model by comparing its results with empirical findings from multi-digit arithmetic obtained in humans, such as the carry-over effect and the problem-size effect (see Klein et al., 2010). Thereby, our modeling approach represents a critical step towards generalizing the theoretical idea of decomposed processing of multi-digit numbers beyond magnitude comparison as well as a first step towards simulating the hierarchical development of mathematical skills and concepts.

In particular, we developed a neural network model simulating addition of single-digit and two-digit numbers. We considered a neural network architecture that reflects the typical process of how humans approach multi-digit addition first acquiring single-digit addition before proceeding to multi-digit addition. In other words, we conceptualized our model in a way replicating the standard algorithm of adding multi-digit numbers taught at school and not producing sums from memory based on extensive learning. For instance, when adding $35+47$, the model (i) identifies the position of units and tens, (ii) performs single-digit additions on identified units and tens (e.g., $5+7$ and $3+4$), and (iii) considers the carry-over from units to tens ($5+7=12$; considering 2 as the unit and 1 as the carry-over to tens).

To evaluate potential benefits of such componential processing, we contrasted an algorithmic *step by step* curriculum, (i.e., a part of the model is first trained to perform single-digit addition and, once successful, multi-digit addition is trained) and an *all at once* curriculum (i.e., all components of the entire model are trained simultaneously on all types of additions). The idea of structuring learning this way aligns with the concept of *curriculum learning* proposed by Bengio et al. (2009), who emphasized the advantages of starting with simpler tasks and progressively increasing complexity (see also Elman (1993). Employing these two types of training procedures, we investigated whether our neural network model learns multi-digit addition similarly to humans by incrementally increasing the complexity of the task difficulty. As we trained our model the same multi-addition algorithm as typically performed by humans, we expected to observe known hallmark effects previously observed in humans, such as the carry-over effect and the problem size effect. In particular, we evaluated i) how different training curricula (*all at once* vs. *step by step*) impacted training performance before ii) we looked at the emergence of the carry-over and problem size effect during the training depending on the training curriculum. Finally, we compared modeling results to the empirical findings of (Klein et al., 2010) to validate simulated effects.

While our model can be straightforwardly scaled to simulate componential processing of larger multi-digit addition problems, we limited our simulations to two-digit additions to enable direct comparison with available empirical data.

Methods

The Neural Network Model

The model consisted of three different modules: a single-digit addition Unit Extractor Module, a single-digit addition Carry-over Extractor Module, and a Decision Module (see Figure 1). To add two numbers, the neural network considered the following simulation steps. The input layer of the neural network received the two numbers ($a = a_2 \cdot 10 + a_1$ and $b = b_2 \cdot 10 + b_1$) to be added decomposed into the unit digit of the first number (a_1), tens digit of the first number (a_2), unit digit of the second number (b_1) and tens digit of the second number (b_2). These were paired in four different combinations reflecting single-digit additions of both tens and unit digits ($a_1 + b_1$ and $a_2 + b_2$), as well as additions of the tens digit of one and the unit digit of the other addend ($a_2 + b_1$ and $a_1 + b_2$). Although counterintuitive, this is in line with empirical findings of suggesting that individual digits are compared (Wood et al., 2005), added (Lefevre et al., 1988), or even multiplied automatically (Galfano et al., 2003) when they are encountered. Implementation allowed decisions about which single-digit additions are correct/incorrect for the simulation of multi-digit addition. Processing through Unit and Carry-over Extractor Modules for single-digit additions resulted in eight possible values (units and tens of the sum of each possible pairing). Finally, these were integrated in the Decision Module to establish the digits of the final result.

For the Unit and Carry-over Extractor Modules, we considered long short-term memory (LSTM) neural networks (Hochreiter, 1997), each of them with two inputs (the two single digits). We obtained a categorical output associated with the resulting digit from a final layer with a *softmax* activation function that returned the probability of each of the possible values. For the sum $5+7=12$, the inputs of both modules would be the single digits 5 and 7, while the expected output of the Unit Extractor Module would be a 2 and the expected output of the Carry-over Extractor Module would be a 1.

For example, considering a perfectly trained model that adds $35+47$, the input would be $a_2 = 3$ and $a_1 = 5$, $b_2 = 4$ and $b_1 = 7$. Then, the Unit Extractor Module would be applied to $a_1 + b_1 = 5+7 = 12$, returning a 2 (unit digit of the sum of unit digits), and the same for $a_2 + b_2 = 3+4 = 7$, returning a 7. Applying the Carry-over Extractor Module to $a_1 + b_1 = 5+7 = 12$ would result in a 1 (tens digit and thus carry-over of the unit-digit sum), while applying it to $a_2 + b_2 = 3+4 = 7$ would return a 0. These two single-digit addition modules would also be applied to the sums $a_1 + b_2$ and $a_2 + b_1$, but a perfectly trained model would not consider

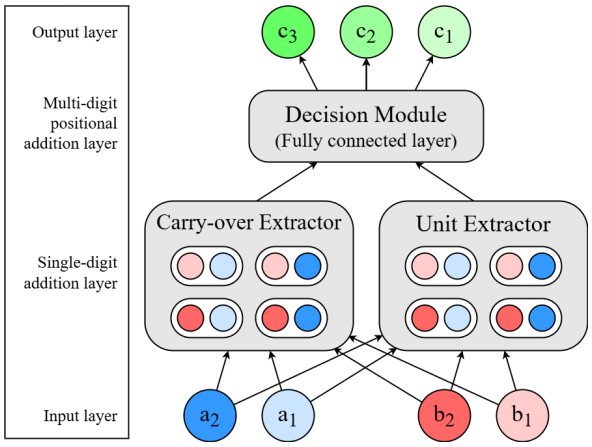

Figure 1: Neural network architecture for simulating multi-digit addition. Input consists of four digits representing the two numbers to be added (units and tens of each number, i.e., $a+b$ with $a = a_2 \cdot 10 + a_1$ and $b = b_2 \cdot 10 + b_1$). The output reflects the three digits of the final sum ($a+b=c$ with $c = c_3 \cdot 100 + c_2 \cdot 10 + c_1$). Processing steps involve pairing digits (i.e., a_2+b_2, a_2+b_1, a_1+b_2 and a_1+b_1), getting their sums through the single-digit addition modules, and combining results in the Decision Module for the final result.

them, as they add units to tens. Finally, the Decision Module would consider that it has to add units to units, tens to tens..., so the unit digit of the final sum would be a $c_1 = 2$, the tens digit would be a $c_2 = 7+1 = 8$ and the hundreds digit would be a $c_3 = 0$. Thus, the output of the model would be the number 082.

Unit Extractor Module The Unit Extractor Module is designed to predict the unit digit of the sum of single digits by categorizing the input data into one of ten possible outcomes (0 to 9). This first module comprises three hidden long short-term memory (LSTM) layers with 16, 32, and 16 neurons, respectively, followed by an output layer of 10 neurons. In Figure 1, the entire Unit Extractor Module is represented as a gray box, which encapsulates the three long short-term memory (LSTM) layers and the output layer. This abstraction is used to simplify the diagram while preserving the hierarchical structure of the overall model.

The output layer computes the probability distribution over the ten possible outcomes, enabling the module to estimate the likelihood of each digit being the unit of the single-digit sum. The final prediction is obtained by selecting the category with the highest probability and converting it into the corresponding integer.

During the *step by step* training, this module was trained independently using the categorical cross-entropy loss function, which is commonly applied in classification tasks involving one-hot encoded target variables. Let $(\hat{y}_0, \ldots, \hat{y}_9)$ denote the outputs of the module. To obtain the probability distribution over the possible digit categories (0 to 9), we apply the *softmax* function. Additionally, let $(y_0, \ldots, y_9)$ represent the true one-hot encoded label, where $y_j = 1$ for the correct digit j, and $y_i = 0$ for all $i \neq j$. The loss function is then defined as:

$$\mathcal{L}_{\text{Unit Extractor}} = -\sum_{n=1}^{N}\sum_{i=0}^{9} y_i^{(n)} \cdot \log\left(\frac{\exp\left(\hat{y}_i^{(n)}\right)}{\sum_{k=0}^{9}\exp\left(\hat{y}_k^{(n)}\right)}\right), \quad (1)$$

where the superscript (n) indicates the n-th training example from a total of N examples. This loss function ensures that the model minimizes the discrepancy between the predicted and actual probabilities for the correct category, enabling accurate identification of the unit digit in the single-digit sum.

Carry-over Extractor Module The Carry-over Extractor Module is designed to determine whether the sum of the digits exceeds 10, acting as a binary classifier. Structurally, this module consists of a single hidden long short-term memory (LSTM) layer with 16 neurons, followed by a two-dimensional output layer. As with the Unit Extractor Module, the entire Carry-over Extractor Module is represented as a gray box in Figure 1, encapsulating both the long short-term memory (LSTM) layer and the output layer.

The output layer computes the probabilities corresponding to the two possible categories: the sum being either less than 10 (category 0) or equal to or larger than 10 (category 1). The final output is determined by selecting the category with the highest probability, either 0 or 1.

During the *step by step* training curriculum, this module was trained independently using the binary cross-entropy loss function, suitable for classification problems with two categories. Therefore, the loss function for this module is analogous to the one used in the Unit Extractor Module, i.e., equation (1), but considering a vector with two predicted probabilities (less than or greater than 10) instead of ten.

Decision Module The Decision Module serves as the final component of the architecture, responsible for determining the relevance of the outputs from the previous modules and combining them to produce up to three digits of the final sum. This module is implemented as a fully connected layer with 3 neurons, corresponding to the hundreds, tens, and unit digits of the potential addition result.

Let $a+b$ represent the addition to be performed, with a_1, b_1 as the unit digits and a_2, b_2 as the tens digits of a and b, respectively. For simplicity, we express the correct result as $c = c_3c_2c_1$, where $c = c_3 \cdot 100 + c_2 \cdot 10 + c_1$. The single-digit addition modules process all possible digit pairs (a_1, b_1), (a_1, b_2), (a_2, b_1), and (a_2, b_2), producing outputs for both the Unit and Carry-over Extractors:

$$\begin{cases} u[a_i, b_j] = \text{UnitExtractor}(a_i, b_j) \\ t[a_i, b_j] = \text{Carry-overExtractor}(a_i, b_j) \end{cases} \quad \text{for } i, j \in \{1, 2\}. \quad (2)$$

The Decision Module computes the final result $\hat{c}_3\hat{c}_2\hat{c}_1$, corresponding to the hundreds, tens, and unit digits of the sum. This is achieved through a linear combination of 24 learned parameters, which are applied to the 8 values ($u[a_i, b_j]$ and

$t[a_i, b_j]$) produced by the previous modules:

$$\hat{c}_k = \sum_{i,j=1}^{2} w^u_{k,i,j} \cdot u[a_i, b_j] + w^t_{k,i,j} \cdot t[a_i, b_j] \text{ with } k \in \{1,2,3\}, \tag{3}$$

where $w^u_{k,i,j}$ and $w^t_{k,i,j}$ represent the weights associated with the unit and tens outputs, respectively.

The training of this module uses a *digit-wise mean squared error* (MSE) as the loss function, ensuring that the predicted result $\hat{c}_3\hat{c}_2\hat{c}_1$ approximates the correct sum $c_3c_2c_1$. Given a dataset of N examples, where the predicted and true results are $\{(\hat{c}_3\hat{c}_2\hat{c}_1)^{(1)}, \ldots, (\hat{c}_3\hat{c}_2\hat{c}_1)^{(N)}\}$ and $\{(c_3c_2c_1)^{(1)}, \ldots, (c_3c_2c_1)^{(N)}\}$, the loss function is defined as:

$$\mathcal{L}_{\text{Decision Module}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{3} \left(\hat{c}_k^{(n)} - c_k^{(n)} \right)^2 \tag{4}$$

This implementation ensures that the Decision Module effectively integrates the outputs from the previous modules to compute accurate addition results, thereby achieving the overall goal of the architecture.

Although the current implementation focuses on two-digit numbers, the architecture is inherently scalable to the addition of any N-and M-digit number. This is because both, Unit Extractor and Carry-over Extractor, operate over all possible pairs of single digits from the two operands. Then, it results in $N \times M$ such pairings, each processed independently by the extractors. The Decision Module then integrates their outputs to compute the final result, just as it does in the two-digit case. For example, when adding $3451 + 678$, the model would consider all $4 \times 3 = 12$ digit pairings (e.g., $3+6$, $3+7$, $3+8$, ..., $1+8$), extract units and carries of each pairing, and combine these into the four-digit sum in the Decision module. Thus, applying the model to larger numbers requires just an expansion in the number of digit pairings and the dimensionality of the inputs and outputs.

Learning Process and Validation

To train the model, we divided the set of all possible single- and two-digit addition problems (100×100 examples) into two subsets: a test set comprising 192 problems and a training set comprising the remaining 9808 problems. The test set reflected the stimuli presented to human participants in Klein et al. (2010), ensuring direct comparability of empirical and simulation results. The training set included all addition problems where one number from 0 to 99 was added to another number from 0 to 99, excluding problems in the test set.

Addition problems of the test set were manipulated realizing a 2 x 2 design of the factors carry-over (i.e., requiring vs. not requiring a carry operation) and problem size (considered "small" when the sum was smaller than 40 and "large" in case the sum was larger than 60 but smaller than 100). The test set included 48 problems for each of the four resulting categories (examples shown in Table 1).

Training problems were presented in random order with problems decreasing in frequency as the sum increased to reflect the empirical distribution of number frequencies (Dehaene & Mehler, 1992). In particular, we used the following function implementing an exponential decay in frequency, so that smaller problems were trained with a higher frequency than larger ones:

Table 1: Example problems with and without carry-over and small or large problem size (sum < 40 vs. > 60), respectively.

	Without carry-over	With carry-over
Small problem	$4+3$	$17+19$
Large problem	$25+62$	$38+47$

$$p(a,b) = \frac{\exp\left(-\frac{a+b}{100}\right)}{\sum_{a,b} \exp\left(-\frac{a+b}{100}\right)} \tag{5}$$

The network weights were initialized using a uniform probability distribution within the range $[-1,1]$, scaled by a variable ε. This variable took values from the set $\varepsilon \in \{1,2,3,5,10,15,20,25,30,50\}$. To ensure robust and unbiased results, we trained 10 models for each value of ε, resulting in 100 trained models in total. The results reported below reflect the average across all models to capture general behavioral patterns and reduce the influence of specific initialization conditions.

Impact of Curriculum on Model Training To evaluate the impact of the curriculum on the learning process, we employed two distinct training curricula for the models: i) in the *all at once* curriculum all modules of the model were trained simultaneously as part of a single, integrated model whereas ii) in the *step by step* curriculum modules were trained sequentially, with modules for processing single-digit additions being trained separately before integrating them into the Decision Module for final training. This reflected the idea of decomposed processing of multi-digit numbers in addition as discussed in the introduction. To ensure a fair comparison between both training curricula, we maintained consistent conditions and parameters wherever possible.

In both cases, the total training process was limited to a maximum of 1,000,000 repetitions. For the *all at once* curriculum, the integrated model was trained over 1,000 batches, each consisting of 1,000 problems. Conversely, in the *step by step* curriculum, the training process was divided among the individual modules: the Unit Extractor Module was trained on 300,000 repetitions (3,000 batches of 100 problems each), the Carry-over Extractor Module was trained on 100,000 repetitions (1,000 batches of 100 problems each), and the Decision Module was trained afterwards on 600,000 repetitions (600 batches of 1,000 problems each). Variations in the number of training repetitions for each module reflect the differing levels of complexity inherent to their respective tasks.

Importantly, the total number of training repetitions does not correspond to an equally large number of distinct training problems (there are only 9808 distinct problems). Instead,

many problems need to be repeated across epochs, similar to how learners repeatedly encounter the same arithmetic problem in educational settings. This design ensures that each module receives sufficient training tailored to its specific demands before being integrated.

Learning Algorithm To optimize network parameters, we employed the *stochastic gradient descent*, applied over batches containing varying numbers of training samples, depending on the training curriculum. Learning rate was fixed at $\eta = 0.01$. Let $w^{(t)}$ be the parameters of the neural network at batch t, the update of these parameters is obtained through the following equation:

$$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla \mathcal{L}\left(w^{(t)}\right) \tag{6}$$

For the *step by step* training curriculum, we only updated network parameters $w^{(t)}$ based on equation (6) of the module we were training at that moment, and the loss function used was the specific one for that module. In contrast, for the *all at once* training curriculum, parameters of all three modules were updated following equation (6), with the loss function being the digit-wise mean squared error given by (4).

Evaluation Metrics To evaluate the performance of the different models in terms of their capability of simulating the carry-over effect as well as the problem size effect, we considered the average percentage of errors committed by the model as the primary evaluation metric. This metric provides a clear measure of how well the model generalizes across various types of addition problems. The lower the average error, the better the models' ability to calculate the correct sum across the respective problem types.

Results

Impact of Curriculum on Model Training

Figure 2 illustrates the models' training progress (i.e., average error rates) for both training curricula. Even after 1,000,000 training repetitions, the performance remained comparably poor when the network was trained on the *all at once* curriculum. In contrast, the network was able to learn multi-digit arithmetic when trained on the *step by step* curriculum.

For the following analyses, we focus on simulations following the *step by step* training curriculum. The results presented specifically refer to the learning process of the Decision Module. Throughout the training of the model, we systematically collected performance data on the test set to evaluate i) aspects of the overall learning progress and ii) the occurrence of the carry-over and problem size effect.

Occurrence of Carry-over and Problem Size Effect

We observed the carry-over effect (as the difference between light and dark colored lines) and problem size effect (as the difference between bluish and reddish lines) from early stages of the training process (see Figure 3).

Interestingly, we observed that the problem size effect was especially pronounced in cases of small problem sizes without carry-over (i.e., fewer average errors in this category), with a significant difference appearing as early as batch 30. This separation remained consistent throughout the remainder of the training process. In contrast, the small problem size with carry-over did not exhibit a noticeable difference until much later—around batch 400. This delayed emergence suggests that the presence of carry-over introduces additional complexity to the learning process, requiring more exposure before it begins to manifest clearly.

Another key observation is the interaction between the carry-over effect and the problem size effect in large problem sizes. Notably, errors for both large problem sizes with and without carry-over merged around batch 950. The same was observed for small problem sizes from around batch 1,200. This indicates that, while the carry-over effect is initially present, it diminishes as the model learns. This aligns with the general pattern in neural networks, where distinctions between different types of errors fade as training converges.

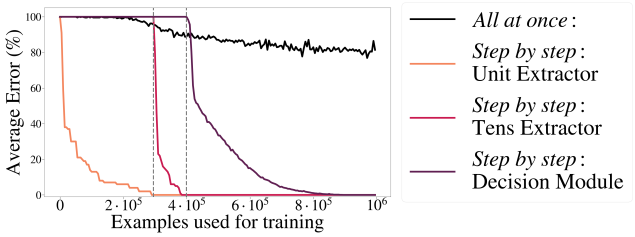

Figure 2: Comparison of *all at once* and *step by step* training curricula depicting how average error (%) for the test dataset decreases as the number of training problems increases. In the *step by step* curriculum, training is divided into three stages (indicated by vertical dashed lines), corresponding to the sequential training of the individual modules.

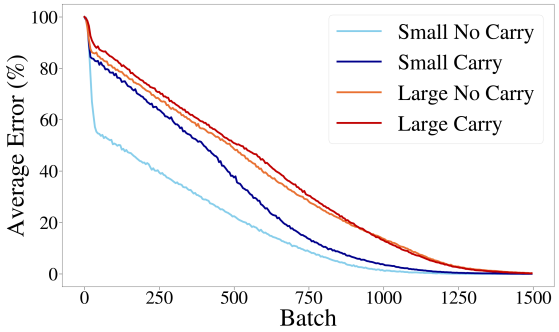

Figure 3: Average error progression over training batches for the Decision Module in the *step by step* curriculum. Error computed on the test dataset and averaged over 10 models trained for each of the 10 initial values of ε.

Comparison with Empirical Data

After training the 10 models for each of the 10 initial conditions over a maximum of 600 batches with 1,000 problems each, we evaluated their performance on the test set from Klein et al. (2010) to evaluate the carry-over and problem size

effects. This allowed direct comparisons between our models' behavior and empirical data from human participants.

As shown in Figure 4, ANOVA results clearly indicated significant effects of both the requirement of carry-over [$F(1,198) = 7.70$; $p = 0.006$] and problem size [$F(1,198) = 19.07$; $p < 0.001$]. Closely matching the pattern observed by Klein et al. (2010) for human participants, model performance was better for problems with smaller compared to larger problem sizes as well as those requiring no carry operation compared to those requiring one.

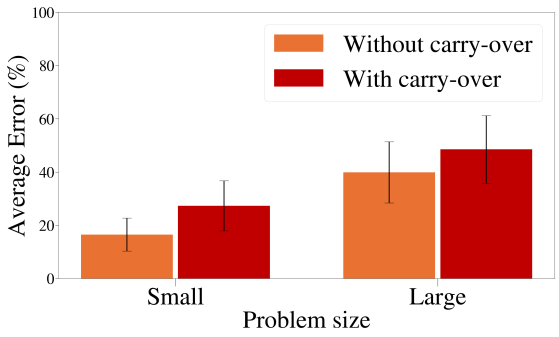

Figure 4: Average performance of the models after 600 training batches, showing the test error rates for different carry-over and problem size categories. The curriculum used follows the *step by step* approach. Error bars represent 1 SEM).

Discussion

The aim of the present study was to develop a neural network model that successfully replicates hallmark behavioral effects human participants show when doing multi-digit arithmetic—the carry-over effect and the problem size effect. Therefore, we trained a neural network model applying a training procedure mimicking how children typically learn multi-digit addition—by first learning single-digit additions and then proceeding to multi-digit additions. We then compared the effectiveness of this *step by step* curriculum to an alternative *all at once* curriculum, where the neural network was not pre-trained on single-digit arithmetic.

Our results clearly indicate that the model successfully learned to perform multi-digit addition when trained following the *step by step* curriculum but failed under the *all at once* curriculum. This learning was achieved from a relatively small and structured training set (9,808 distinct problems) and took only about 4 hours of training on a standard GPU, highlighting that the approach is not computationally intensive. This aligns with previous research showing that humans benefit from structured, progressively incremental learning (e.g., Church et al., 2013; Roads et al., 2018) as commonly applied for learning mathematics (e.g., Anderson et al., 1995). In addition, this finding substantiates the idea that multi-digit arithmetic is decomposed into operations on single-digit numbers. As such, our findings provide first evidence of a neural network model to support the notion of decomposed processing of multi-digit numbers to generalize to multi-digit arithmetic.

Moreover, the model trained following the *step by step* curriculum replicated key behavioral patterns observed in humans on the same item set, indicating a significant carry-over effect and a problem-size effect (cf. Klein et al., 2010). This further substantiates our neural network approach as a valuable tool for describing and better understanding the cognitive processes underlying arithmetic learning.

Importantly, the model not only replicated the behavioral patterns observed in human participants, but also provided insight into how these effects emerge over the course of learning (see Figure 3). For instance, the carry-over effect emerged naturally and early on as the model learned to handle multi-digit sums. This seems to suggest that the cognitive difficulty associated with a carry operation is intrinsic to the arithmetic process rather than a superficial byproduct of training. In contrast, the problem size effect was initially more pronounced in small problems without carry-over, with the model committing fewer errors in these cases from early stages of the training on. A possible explanation for this is that smaller sums appeared more frequently in the training set to reflect their higher occurrence in daily life. This may have led the model to prioritize them initially. This suggests that the problem size effect may not necessarily reflect an inherent cognitive difficulty with larger sums but rather a consequence of frequency of exposure during training (also see Huber et al., 2016).

The current model does not build on a representation of single-digit number magnitudes, as it focuses solely on multi-digit addition. A desirable perspective is to implement the number magnitude representation proposed by Huber et al. (2016) as a foundational element of the present model. This would substantiate the models' foundation in previous findings on decomposed processing of multi-digit number magnitude which would further strengthen the idea of an hierarchical development of mathematical skills. In turn, this may also improve the models' ability to generalize to other arithmetic operations. An ultimate future avenue would be to apply our model to predict the transfer of multi-digit addition skills to other arithmetic domains, such as multi-digit subtraction as the invert operations, multiplication as conceptualized by repeated addition, and even more advanced mathematical concepts such as the addition of rational numbers.

In conclusion, our model represents a crucial first step in simulating the hierarchical development of advanced numerical cognition, particularly in multi-digit arithmetic. By successfully replicating key behavioral effects observed in human performance, it underscores the significance of structured, stepwise learning in acquiring arithmetic skills.

Acknowledgments

This work was supported by a Research Grant from HFSP (Ref.-No: RGEC33/2024). This work was partially supported by the UKRI Economic and Social Research Council (ES/W002914/1). Data will be made available upon request.

References

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The journal of the learning sciences*, *4*(2), 167–207.

Ashcraft, M. H. (1992). Cognitive arithmetic: A review of data and theory. *Cognition*, *44*(1-2), 75–106.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. *Proceedings of the 26th annual international conference on machine learning*, 41–48.

Church, B. A., Mercado III, E., Wisniewski, M. G., & Liu, E. H. (2013). Temporal dynamics in auditory perceptual learning: Impact of sequencing and incidental learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *39*(1), 270.

Dehaene, S., & Mehler, J. (1992). Cross-linguistic regularities in the frequency of number words. *Cognition*, *43*(1), 1–29.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, *48*(1), 71–99.

Fuson, K. C., Wearne, D., Hiebert, J. C., Murray, H. G., Human, P. G., Olivier, A. I., Carpenter, T. P., & Fennema, E. (1997). Children's conceptual structures for multidigit numbers and methods of multidigit addition and subtraction. *Journal for research in mathematics education*, *28*(2), 130–162.

Hochreiter, S. (1997). Long short-term memory. *Neural Computation MIT-Press*.

Huber, S., Nuerk, H.-C., Willmes, K., & Moeller, K. (2016). A general model framework for multisymbol number comparison. *Psychological review*, *123*(6), 667.

Imbo, I., Vandierendonck, A., & De Rammelaere, S. (2007). The role of working memory in the carry operation of mental arithmetic: Number and value of the carry. *Quarterly Journal of Experimental Psychology*, *60*(5), 708–731.

Klein, E., Moeller, K., Dressel, K., Domahs, F., Wood, G., Willmes, K., & Nuerk, H.-C. (2010). To carry or not to carry—is this the question? disentangling the carry effect in multi-digit addition. *Acta psychologica*, *135*(1), 67–76.

Lambert, K., & Moeller, K. (2019). Place-value computation in children with mathematics difficulties. *Journal of Experimental Child Psychology*, *178*, 214–225.

Lefevre, J.-A., Bisanz, J., & Mrkonjic, L. (1988). Cognitive arithmetic: Evidence for obligatory activation of arithmetic facts. *Memory & Cognition*, *16*(1), 45–53.

Meyerhoff, H. S., Moeller, K., Debus, K., & Nuerk, H.-C. (2012). Multi-digit number processing beyond the two-digit number range: A combination of sequential and parallel processes. *Acta Psychologica*, *140*(1), 81–90.

Moeller, K., Huber, S., Nuerk, H.-C., & Willmes, K. (2011). Two-digit number processing: Holistic, decomposed or hybrid? a computational modelling approach. *Psychological research*, *75*, 290–306.

Moeller, K., Klein, E., Nuerk, H.-C., & Cohen Kadosh, R. (2012). A unitary or multiple representations of numerical magnitude?–the case of structure in symbolic and non-symbolic quantities. *Frontiers in psychology*, *3*, 191.

Moyer, R. S., & Landauer, T. K. (1967). Time required for judgements of numerical inequality. *Nature*, *215*(5109), 1519–1520.

Nuerk, H.-C., Moeller, K., Klein, E., Willmes, K., & Fischer, M. H. (2015). Extending the mental number line. *Zeitschrift Für Psychologie*.

Nuerk, H.-C., Weger, U., & Willmes, K. (2001). Decade breaks in the mental number line? putting the tens and units back in different bins. *Cognition*, *82*(1), B25–B33.

Poltrock, S. E., & Schwartz, D. R. (1984). Comparative judgments of multidigit numbers. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *10*(1), 32.

Roads, B. D., Xu, B., Robinson, J. K., & Tanaka, J. W. (2018). The easy-to-hard training advantage with real-world medical images. *Cognitive Research: Principles and Implications*, *3*, 1–13.

Verguts, T., & De Moor, W. (2005). Two-digit comparison: Decomposed, holistic, or hybrid? *Experimental Psychology*, *52*(3), 195–200.

Wood, G., Mahr, M., & Nuerk, H.-C. (2005). Deconstructing and reconstructing the base-10 structure of arabic numbers. *Psychology Science*, *47*(1), 84–95.

Yuan, L., Prather, R. W., Mix, K. S., & Smith, L. B. (2019). Preschoolers and multi-digit numbers: A path to mathematics through the symbols themselves. *Cognition*, *189*, 89–104.

Zbrodoff, N. J., & Logan, G. D. (2005). What everyone finds: The problem-size effect. In *The handbook of mathematical cognition* (pp. 331–345). Psychology Press.