L'apprentissage algorithmique, une nouvelle étape pour l'IA. Une application aux opérations arithmétiques

Frédéric Armetta, Anthony Baccuet, Mathieu Lefort

L'apprentissage algorithmique, une nouvelle étape pour l'IA. Une application aux opérations arithmétiques

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

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

{frederic.armetta, mathieu.lefort}@univ-lyon1.fr

Résumé

L'apprentissage profond fournit les meilleures performances de l'état de l'art dans de nombreux domaines (reconnaissance d'images, traitement naturel du langage, ...). Une des prochaines étapes consiste en l'apprentissage d'algorithmes, afin de proposer de nouvelles formes de généralisation pour l'IA. Ce problème est actuellement très difficile car il nécessite des récurrences algorithmiques, la gestion d'une mémoire et la combinaison de sous-tâches, ce qui engendre des problèmes d'apprentissage et de faibles performances. Nous présentons une méthode d'apprentissage originale (UAT : Unrolling Algorithmic Training) pour aborder ces spécificités, appliquée à la multiplication multi-digit.

Mots-clés

Apprentissage automatique d'algorithmes, Apprentissage profond, Apprentissage Actif

Abstract

Deep learning achieved state of the art performances in multiple domains (image recognition, natural language processing, ...). 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 sub-tasks to combine, which leads to trainability problems and poor performance. We present an original training method to address these specificities for the multi-digit multiplication learning, called Unrolling Algorithmic Training (UAT).

Keywords

Algorithmic Machine Learning, Deep Learning, Active learning

1 Introduction

Depuis les premiers succès des réseaux de neurones convolutifs pour la classification d'images [18], les méthodes d'apprentissage profond ont fait progresser l'état de l'art dans de nombreux domaines [12]. Cela se traduit par des tâches de plus en plus complexes comme la traduction automatique [32] ou la résolution de jeux combinatoires [30]. L'une des prochaines étapes correspond à l'apprentissage d'algorithmes, qui peut être utile au calcul d'expressions mathématiques, afin de permettre l'apprentissage de toute tâche pouvant être exprimée dans une machine de Turing et de développer ainsi l'autonomie des intelligences artificielles.

L'exécution d'algorithmes par réseaux de neurones est encore un domaine de recherche émergent. Il s'agit d'un problème difficile car il nécessite de mémoriser des variables pendant de longues périodes, d'apprendre des inférences, de combiner des procédures, d'extrapoler à des domaines inconnus, etc. Plus fondamentalement, il s'agit de concilier la compréhension et la manipulation symboliques avec l'apprentissage statistique. La machine de Turing neuronale [13] a été proposée pour apprendre des procédures algorithmiques de bout en bout (résolution *end-to-end*), telles que le tri de listes. Ce modèle propose de réduire le problème d'apprentissage (*trainability problem*) rencontré par les RNN (*Récurrent Neural Networks*), qui sont Turing complets [28], en ajoutant des mécanismes spécifiques tels qu'une mémoire externe et des possibilités d'accès différentiables. Cependant, ce modèle rencontre malgré tout des problèmes d'apprentissage, de sorte que les performances sont difficiles à reproduire et inconstantes [8]. Ces problèmes sont causés par la profondeur de l'architecture proposée [17] mais aussi par la complexité intrinsèque des opérations à apprendre, en particulier les dépendances à long terme dans les données ou les variables à manipuler.

Pour surmonter ce problème d'apprentissage d'algorithmes, une autre solution consiste à décomposer cet algorithme en étapes successives que le réseau neuronal calculera et apprendra de manière itérative en réintroduisant sa sortie précédente par le biais d'une récurrence externe au modèle. Cette procédure a été appliquée à une architecture inspirée d'une architecture transformeur pour l'apprentissage de certaines procédures algorithmiques [34] et à un MLP (*multilayer perceptron*) pour les opérations arithmétiques [21, 34].

Dans cet article, nous avons choisi comme application les opérations arithmétiques pour illustrer notre proposition. En particulier, la multiplication à plusieurs chiffres qui peut être résolue par un algorithme. Ce calcul implique de nombreuses opérations et variables dans le flux de calcul (calculer plusieurs multiplications à un chiffre, additionner correctement les résultats intermédiaires afin d'obtenir le résultat final). La complexité d'une telle opération est également liée à la propagation des retenues [7]. L'apprentissage direct de bout en bout des multiplications conduit à des performances

médiocres [16]. Pour résoudre ce problème d'apprentissage, nous proposons avec la méthode UAT (*Unrolling Algorithmic Training*) d'introduire conjointement pendant l'apprentissage différentes sous-tâches intermédiaires utiles à la résolution et la multiplication complète de bout en bout, en pondérant les différentes tâches par une stratégie d'apprentissage actif. Cette approche est en quelque sorte similaire à un apprentissage multitâche, en utilisant des tâches complémentaires pertinentes pour aider le réseau à apprendre la multiplication complète. Cependant, pour l'approche que nous proposons, le réseau apprend toutes ces tâches sans aucune couche dédiée à chacune d'entre elles. Ainsi, le réseau devra s'accommoder des opérations intermédiaires afin d'apprendre la multiplication complète. Une autre différence est que cette adaptation doit suivre le cours de l'algorithme, en connectant séquentiellement les tâches et en propageant les valeurs intermédiaires utiles à la résolution. Nous montrons que cette procédure d'apprentissage peut également être utilisée pour pré-entraîner le réseau.

Nous présentons les travaux connexes à ce travail dans la section 2. Nous montrons dans cette section que les modèles de langage larges ne fournissent pas de bons résultats concernant les opérations arithmétiques telles que la multiplication et connaissent des difficultés à généraliser. Nous détaillons le cas d'utilisation des multiplications à plusieurs chiffres dans la section 3. Cette tâche est l'une des plus difficiles des quatre opérations arithmétiques car lorsque résolue algorithmiquement elle s'appuie sur un grand nombre d'étapes intermédiaires. Le modèle sur lequel notre procédure d'apprentissage est appliquée est ensuite détaillé dans la section 4. Le protocole et les résultats sont présentés dans la section 5. Nous conclurons et discuterons des perspectives de ce travail dans la section 6.

2 État de l'art

Au cours des années, de nombreux modèles d'apprentissage profond ont été proposés pour résoudre différents types de problèmes [12]. Les réseaux neuronaux convolutifs [18] permettent de classer des images avec des performances dépassant parfois celles des humains dans des scénarios spécifiques. Les réseaux neuronaux récurrents (*RNN*) permettent quant à eux de manipuler des données temporelles, certains modèles permettent d'atténuer les phénomènes d'oubli des données manipulées [15] [6].

Les modèles d'apprentissage profond sont des approximateurs de fonctions universels [9], mais ils sont souvent difficiles à entraîner. En théorie, une architecture neuronale même peu profonde est suffisante pour apprendre toute fonction. En pratique, la capacité des réseaux profonds à apprendre des représentations pertinentes à partir des données est bien meilleure [1]. Les réseaux de neurones récurrents sont Turing-complets [28], mais ils sont également confrontés à des difficultés d'apprentissage [13]. Cette limitation tend à être plus prononcée pour les tâches complexes, en particulier lorsque des inférences à long terme sont nécessaires. Par exemple, le taux d'erreur pour une opération arithmétique est lié au nombre de retenues pour un perceptron multicouche [7].

Certaines stratégies globales ont été proposées pour rendre l'apprentissage des réseaux plus efficace. Parmi celles-ci, les stratégies d'apprentissage actif s'intéressent à sélectionner les données pour l'apprentissage qui permettent de maximiser la progression de l'apprentissage [27, 4]. Une façon courante de le faire est l'apprentissage par curriculum pour lequel les tâches soumises sont ordonnées par complexité croissante [2]. Une autre stratégie repose sur de l'apprentissage multitâche qui consiste à combiner différentes tâches avec des objectifs similaires afin que le réseau puisse mieux généraliser et apprendre la structure sous-jacente des données [35]. Dans ce cas, certaines couches en sortie de réseau sont spécifiques aux différentes tâches apprises. Même si notre proposition peut sembler s'apparenter à de l'apprentissage multitâche, dans notre modèle les tâches sont apprises sur le même réseau et sont de nature algorithmique.

La machine de Turing neuronale (*NTM*) [13] a été spécifiquement conçue pour apprendre des tâches algorithmiques. Afin de limiter le problème de dépendance temporelle rencontré par les réseaux récurrents, elle imite certains des mécanismes d'une machine de Turing en introduisant une mémoire et des accès différenciables en lecture/écriture. L'ordinateur neuronal différentiable [14] améliore ces accès pour apprendre des tâches plus complexes comme des problèmes d'inférence ou de raisonnement en langage naturel. Malgré tout, ces modèles sont difficiles à entraîner avec une très forte sensibilité aux conditions d'initialisation. [8, 25].

L'architecture neuronale GPU [17] partage des idées similaires avec la machine de Turing neuronale mais utilise des réseaux réccurents GRU (*Gated Recurrent Unit*) convolutifs afin d'obtenir un calcul parallèle. Elle est capable d'apprendre certaines tâches algorithmiques telles que l'addition et la multiplication binaires, l'inversion de séquence, ... Sa principale réussite est sa capacité à généraliser l'apprentissage à des entrées de taille plus importante dans les tests. C'est le cas pour la multiplication binaire à 20 chiffres dans l'apprentissage, et testée jusqu'à 2000 chiffres dans les tests sans aucune erreur. Cependant, ce modèle ne parvient pas à apprendre les multiplications décimales [10]. Cette limitation peut être liée à la propagation de retenue qui est difficile à entraîner et apparaît plus fréquemment avec le codage décimal des nombres et peut être partiellement surmontée en utilisant l'apprentissage par curriculum (de binaire à quaternaire puis vers le décimal) [23].

De nombreux articles ont étudié l'apprentissage du traitement des opérations mathématiques, souvent en utilisant des modèles issus du traitement du langage. Dans [11], différents types d'architectures d'encodage-décodage sont comparés sur des tâches simples d'évaluation d'expressions numériques. Cependant, l'article s'intéresse principalement au mélange de formats d'écriture textuelle des nombres et des opérations pour la formulation des opérations arithmétiques. Il est donc difficile d'analyser finement les performances des opérations arithmétiques. Dans [25], plusieurs algorithmes (réseaux réccurents LSTM avec ou sans mécanisme d'attention, architecture neuronale de type *transformeur*) ont été testés sur différentes tâches d'inférence mathématique.

L'objectif principal était de proposer un jeu de données et de comparer les modèles, en particulier sur l'aspect attentionnel, de sorte qu'une fois de plus, les performances détaillées manquent. L'une des conclusions est que les dépendances à long terme sont les plus difficiles à apprendre, ce qui peut être compromettant lorsque les expressions se complexifient. Un transformeur, qui lit l'opération arithmétique caractère par caractère, obtient de bonnes performances sauf pour la soustraction et la multiplication [33]. [19] résout des équations mathématiques, y compris des équations différentielles, avec des modèles Seq2Seq. L'originalité réside dans le fait que l'équation est codée par une représentation arborescente. Une autre approche propose de résoudre des expressions arithmétiques en exploitant des modules feuilles pour la résolution de sous-tâches à un chiffre (multiplication ou addition), ou des modules noeuds exploitant d'autres modules noeuds ou feuilles. Les modules feuilles sont préentrainés. Les modules noeuds apprennent par renforcement avec curriculum les modules à exploiter séquentiellement, l'emplacement mémoire des données à leur communiquer et l'emplacement ou reporter leurs résultats [5]. L'expression à calculer est communiquée au système sous la forme des deux opérandes séparés par le signe de l'opération arithmétique. Les résultats indiquent la résolution de multiplications pour des expressions allant jusqu'à la taille 10, cependant la difficulté de résolution, sous forme de décomposition hiérarchique dans le cadre de cette approche, est liée à la taille minimale des deux opérandes, celle-ci ne peut être déduite de la simple longueur de l'expression. Contrairement à ce qui est indiqué les données utilisées ne sont pas précisées en annexe. Les auteurs soulignent que l'apprentissage par curriculum est nécessaire pour atteindre les valeurs de performance reportées. Les unités logiques arithmétiques neurales [31] sont des cellules capables d'effectuer des opérations arithmétiques en introduisant des modules de calcul spécifiques tels que log, exp, ... L'objectif ici n'est pas d'apprendre l'arithmétique mais de fournir des cellules dédiées capables d'extrapoler l'apprentissage avec des calculs qui s'étendent à des domaines inconnus. Une extension directe de ce travail traitant également des entrées négatives a été proposée dans [26]. D'autres modèles modèles dérivés peuvent calculer des opérations arithmétiques sur des nombres réels [20]. Comme nous l'avons vu, une grande variété de tâches a été explorée dans la littérature, avec des objectifs variés qui ne peuvent être comparées facilement.

Le raisonnement algorithmique fait également partie du matériel qui peut être utile pour le traitement du langage naturel. Les modèles de langage préentrainés, qui se sont beaucoup développés ces dernières années, sont sollicités pour leurs capacité à raisonner. Ils peuvent par exemple apprendre à effectuer un raisonnement numérique à partir de peu d'exemples (en utilisant une requête telle que "Q : Combien font 24 fois 18 ?", en prenant par exemple GPT-J-6B comme base). Il a été démontré que la performance est alors fortement corrélée à la fréquence des termes dans le corpus d'entraînement[24]. Lorsque la cooccurrence des termes est faible, la précision est également faible. Ces observations soulignent le problème de la capacité d'entraînement des algorithmes et les problèmes de généralisation sous-jacents. En conséquence, les performances diminuent à mesure que la taille des opérandes augmente [3]. En outre, alors que l'addition semble moins affectée par la taille des opérandes, les performances s'effondrent pour la multiplication, qui nécessite davantage de raisonnement ou de traitement algorithmique. Les modèles comme GPT-3 sont très larges, GPT-3 exploite environ 500 milliards de tokens pour son apprentissage [22]. Cependant, les résultats montrent que la taille du corpus ne semble pas suffisante pour permettre une généralisation efficace sur des données inconnues. ChatGPT qui repose sur des modèles basés sur GPT hérite des mêmes limitations.

Ces observations suggèrent que les compétences en matière de raisonnement pour de tels modèles ne doivent pas être surestimées. Elles dépendent fortement de la taille du corpus disponible et des inférences statistiques. Le corpus ne contiendra jamais toutes les combinaisons de termes ou les paramètres des algorithmes envisageables. Une meilleure approche consiste donc à améliorer la capacité d'apprentissage des algorithmes, ce qui permettrait de faire de meilleures prédictions pour des configurations inconnues.

Dans cet article, nous avons choisi de nous concentrer sur la multiplication à plusieurs chiffres de deux nombres décimaux. Cette opération est suffisamment simple pour ne pas mélanger différents problèmes, mais elle constitue néanmoins un défi, car de nombreux modèles sont incapables de l'apprendre correctement. Cette difficulté provient de la dépendance à long terme due à la propagation de la retenue, mais aussi et plus généralement de la complexité inhérente aux algorithmes qui mettent en jeu de nombreuses opérations et variables dans le flux de calcul.

De plus, certains articles ont mesuré précisément les performances de cette tâche. [16] propose un MLP pour apprendre l'addition et la multiplication soit à partir d'entrées visuelles, soit à partir d'un encodage numérique. Dans les deux cas, la précision obtenue pour la multiplication est faible.

3 Énoncé du problème

Dans cet article, nous considérons la multiplication à plusieurs chiffres comme tâche d'apprentissage par un réseau de neurones. Sachant n le nombre de chiffres des opérandes considérés. La multiplication des deux opérandes conduit à $n+1$ sous-tâches : n multiplications à un chiffre et 1 addition finale du résultat des multiplications partielles (voir figure 1). Selon la représentation des données choisie, chaque opération correspondra à deux lignes de calcul : une pour les retenues et une pour le résultat. La taille maximale de toute opération intermédiaire est $N = 2n$ (cette longueur maximale sera obtenue pour l'addition finale avec une retenue générée à la position du chiffre le plus significatif). Tous les nombres sont complétés par des chiffres de valeur nulle pour correspondre à une taille de N, mais il y aura exactement $n+1$ opérations intermédiaires même si le premier opérande a moins de n chiffres.

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

FIGURE 1 – Exemple de représentation d'une multiplication de deux opérandes à 2 chiffres. Notez que les signes (+ et $\times$) et les lignes horizontales ne sont représentées que pour plus de clarté. Les lignes (1) et (2) correspondent aux deux opérandes. Les lignes (3) et (4) (respectivement (5) et (6)) représentent les retenues et le résultat de la multiplication à 1 chiffre de 8 (respectivement 4) par 23. Les lignes (7) et (8) correspondent aux retenues et au résultat de l'addition des lignes (4) et (6), qui est aussi le résultat final de la multiplication globale de 48 par 23, soit 1104.

4 Modèle

Notre modèle adopte un fonctionnement très similaire à celui des réseaux récursifs utilisés par certains modèles pour le traitement du langage naturel. La récurrence ainsi représentée permettant de potentiellement dérouler étape par étape l'algorithme. Nous n'utilisons pas d'encodage spécifique tel que le binaire, afin de maintenir une manipulation symbolique plus agnostique et générale.

4.1 Représentation des données

Chaque chiffre c est représenté par un vecteur à 10 dimensions selon un codage *one-hot*, soit $(\delta_{ci})_{i\in\{0,\dots,9\}}$. Ce vecteur peut également prendre deux autres valeurs. Tout d'abord, un vecteur nul (composé uniquement de bits à 0) est utilisé pour coder d'éventuelles lignes vides (voir la liste des tâches sur le tableau 1). Notez qu'il sera également utilisé comme caractère de départ pour le décodeur détaillé ci-dessous. D'autre part, un vecteur 1 (composé uniquement de bits à 1) correspond à la fin de la ligne lors de la lecture ou de l'écriture des données.

4.2 Architecture du modèle

Nous utilisons un modèle *Seq2Seq* tel que proposé dans [29] (voir figure 2) . Celui-ci est composé d'un encodeur qui va lire séquentiellement les données et les intègre par récurrence dans l'espace latent du réseau. À partir de l'espace latent résultant, un décodeur produira itérativement une séquence de valeurs, en commençant par un code prédéterminé.

L'encodeur et le décodeur sont tous deux implémentés par des réseaux récurrents LSTM (*Long-Short-Term-Memory*). Pendant l'apprentissage, nous avons utilisé un apprentissage forcé (*teacher forcing*), c'est-à-dire que les caractères qui alimentent récursivement le décodeur sont issus des valeurs cibles (et ne correspondent pas aux valeurs inférées par le réseau, tel qu'utilisé pour la phase de test). La sortie du décodeur est alors comparée pour chacune des valeurs à celles attendues, l'erreur est rétropropagée à travers le décodeur et l'encodeur.

Un bandeau permet de lire et d'écrire deux lignes successives à la fois, chiffre par chiffre. Les chiffres, encodés dans des vecteurs *one-hot*, sont ensuite lus de droite à gauche, avec le vecteur de fin de ligne <eol>(*end of line*) positionné à la fin de chaque ligne. De la même manière, les vecteurs *one-hot* sont écrits sur les sorties. L'entrée du codeur et la sortie du décodeur ont ainsi une taille de 2×10. Chacune des différentes tâches (détaillées ci-dessous) utilise ainsi le même codage.

4.3 Définition des tâches

L'algorithme de multiplication étudié comporte des sous-tâches que l'on soumet également au modèle. L'encodeur enregistre chaque tâche selon sa formulation et le décodeur produit la valeur cible associée (voir le tableau 1). Tel que présenté dans la section 3, l'opération de multiplication de deux nombres à n chiffres peut être décomposée en n multiplications à un chiffre et 1 addition finale. Ces n+1 opérations et la multiplication globale (c'est-à-dire le calcul du résultat final et des retenues) constituent les n+2 tâches d'apprentissage pour le réseau. Pour chaque tâche, l'encodeur lit les entrées correspondantes et le décodeur produit la valeur cible de l'opération correspondante, comme illustré dans le tableau 1. On remarque que la tâche de multiplication globale de bout en bout (*end-to-end*) est soumise sans distinction, avec une phase d'encodage de lignes vides complémentaires dont on pourra par la suite faire varier le nombre afin d'en étudier l'influence. Ces lignes complémentaires permettent au réseau de différencier dans l'encodage la tâche globale de la première multiplication à un chiffre (st1).

Tâche	Entrée Encodeur	Sortie Décodeur
st1	(1//2)	(3//4)
st2	(1//2) (3//4)	(5//6)
st3	(1//2) (3//4) (5//6)	(7//8)
tglobale	(1//2) (ligne double vide $*$ 2)	(7//8)

TABLE 1 – Sous-tâches et tâche globale associées à une multiplication à 2 chiffres (voir les lignes numérotées sur la figure 1, les lignes sont lues et écrites deux par deux ("//"))

4.4 Mécanisme d'apprentissage actif

Nous avons décrit jusqu'à présent plusieurs tâches. On peut choisir de sélectionner un ensemble de tâches à entraîner simultanément sur le modèle. Dans ce qui suit, différentes configurations sont utilisées (sous-tâches uniquement, toutes les tâches, tâche globale uniquement) pour la phase d'apprentissage.

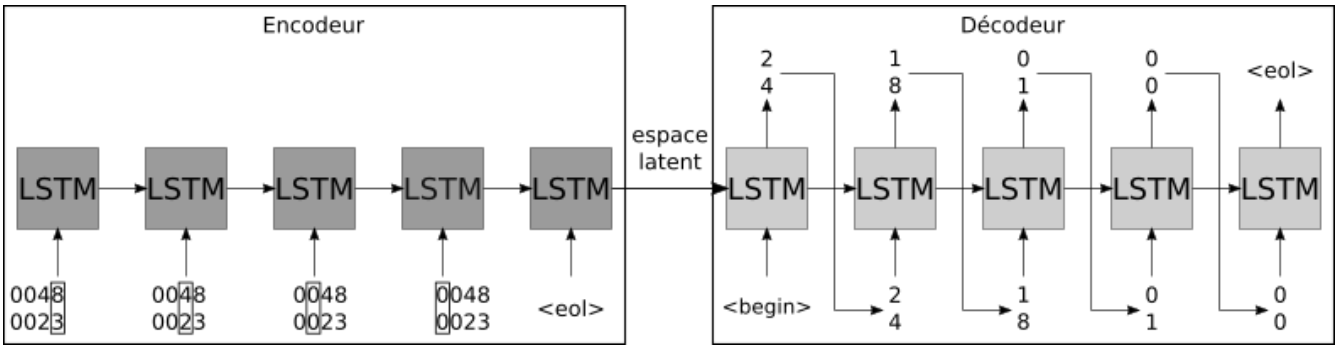

FIGURE 2 – Architecture *Seq2Seq*. L'encodeur reçoit successivement les deux chiffres (encadrés) de deux lignes consécutives (les opérandes 23 et 48) de droite à gauche. À partir de l'encodage, le décodeur produit de façon récurrente les chiffres, de droite à gauche, des deux lignes suivantes (ici la retenue et le résultat de la première multiplication intermédiaire 8×23). Dans la pratique, chaque chiffre est codé sous la forme d'un vecteur *one-hot*.

Afin d'équilibrer l'effort d'apprentissage entre les différentes tâches, nous utilisons le même mécanisme d'apprentissage actif que celui proposé dans [21]. Il consiste d'une part à mesurer le taux d'erreur, noté $err_{tâche}$, de toute tâche pour chaque période d'apprentissage. Pour chaque période (*epoch*), les instances de tâches à apprendre sont sélectionnées aléatoirement parmi l'ensemble des données d'apprentissage préalablement générées. Les instances de tâches sont sélectionnées en respectant la proportion :

$$F_{tâche} = \lambda \frac{err_{tâche}}{\sum_{ta \in listeTâche} err_{ta}} + (1-\lambda)\frac{1}{card(listeTâche)}$$

où λ est un hyperparamètre et *listeTâche* est la liste de toutes les tâches concernées. L'idée générale étant que plus une tâche est difficile à apprendre (premier terme), plus elle est présente pour la période d'apprentissage, avec une limite inférieure dépendant de la valeur lambda.

5 Expérimentations

5.1 Protocole

Pour l'apprentissage, 100 000 couples uniques d'opérandes sont générés. Pour chaque période d'apprentissage, les entrées et sorties du réseau sont générées pour chacun de ces couples, tout en respectant la proportion de chacune des tâches calculée par le mécanisme d'apprentissage actif présenté dans la section 4.4.

Pour les ensembles de données de validation et de test, nous utilisons respectivement 1000 et 10000 couples uniques supplémentaires d'opérandes. La taille de l'espace latent du codeur est fixée à 500 et le paramètre d'apprentissage actif λ à 0.5. Le modèle est entraîné à l'aide de l'optimiseur ADAM avec un taux d'apprentissage de 10^{-4} et un *batch* de taille 10. Tous les résultats présentés dans les sections suivantes sont moyennés sur 4 exécutions de 500 périodes d'apprentissage (*epochs*).

5.2 Résultats

Dans cette section, nous quantifions l'effet de notre proposition d'apprendre simultanément les différentes sous-tâches avec la multiplication globale de bout en bout.

Afin de nous comparer, nous reproduisons le format des multiplications présentées dans [16] et [21], avec des opérandes à 4 chiffres sélectionnés de manière à ce que les résultats soient restreints à des nombres de 7 chiffres.

5.2.1 Sous-tâches et tâche globale

L'objectif principal de cet article est d'aborder le problème d'apprentissage rencontré pour la tâche de multiplication de bout en bout que nous avons choisie comme une première étape vers l'apprentissage algorithmique. Pour le groupe de tâches considérées, les sous-tâches et la tâche globale sont entraînés simultanément sur le même réseau en respectant les proportions de l'apprentissage actif décrit. Nous présentons dans le tableau 2 les performances pour la tâche de multiplication globale, et les comparons avec [16] qui utilise un perceptron multicouches (*MLP*) alors que nous utilisons un modèle *Seq2Seq*. Pour les expérimentations reportées, nous faisons également varier l'échelle et la complexité des problèmes afin de souligner les améliorations apportées par notre proposition.

Les résultats démontrent clairement la contribution des sous-tâches pour l'apprentissage de la tâche globale, en réduisant le taux d'erreur (de 35,87% à 4,51% pour les multiplications restreintes 4×4).

Cela suggère que le modèle est capable de s'auto-organiser et de mettre à profit les sous-tâches complémentaires apprises, validant ainsi le fait que la procédure d'apprentissage proposée est l'élément clé de notre modèle. Bien que les résultats soient similaires sans l'introduction de ses sous-tâches, notre approche est nettement plus performante que [16]. La réduction de l'écart-type lorsque l'apprentissage comprend les sous-tâches montre que la stabilité de l'apprentissage est également améliorée (à l'exception de UAT (train = tglobale) à 8 chiffres de sortie qui peut être exclu de la comparaison par ses faibles résultats).

Nous présentons ci-dessous d'autres améliorations significatives pour le problème le plus difficile présenté (le problème 4×4 (8 chiffres de sortie)) suite a un apprentissage de finition (*fine tuning*) et à l'introduction de récurrences supplémentaires.

	test = tglobale		
	3×3 (6 chiffres en sortie)	4×4 restreinte (7 chiffres en sortie)	4×4 (8 chiffres en sortie)
Hoshen et al.[16]	n.c	37.6 %	n.c.
UAT (train = tglobale)	4.05% (± 1.72%)	35.87% (± 28.10%)	92.42% (± 3.64%)
UAT (train = sous-tâches et tglobale)	3.33% (± 1.32%)	**4.51% (± 1.21%)**	23.34% (± 14.69%)

TABLE 2 – Taux d'erreurs pour la multiplication (tglobale)

4x4 (8 chiffres en sortie)	taux d'erreurs
initial UAT train = sous-tâches + tglobale	23.34% (± 14.69%)
UAT fine tuning (pre-training = sous-tâches + tglobale)	**6.68% (± 4.31%)**
UAT fine tuning (pre-training = sous-tâches)	73.31% (± 11.10%)

TABLE 3 – Influence de l'apprentissage de finition (*fine tuning*)

ligne double vide	taux d'erreurs
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 de la quantité de récurrences disponible lors de la phase d'apprentissage de finition (pre-training = sous-tâches + tglobale, 4 lignes doubles vides)

5.2.2 Apprentissage de finition de la tâche globale

Afin d'améliorer les performances de notre modèle, nous proposons de poursuivre l'apprentissage pour la tâche globale seule, durant 500 périodes supplémentaires, pour le problème le plus difficile de taille 4×4.

Comme nous pouvons le voir sur le tableau 3, en conservant la même récurrence pour le réseau, cet apprentissage de finition conduit à une amélioration significative des performances puisque l'erreur passe de 23,34% à 6,68% pour le problème à 8 chiffres de sortie.

Ce résultat montre également qu'une fois que la tâche globale a pu être amorcée (grâce aux sous-tâches complémentaires), elle peut être maintenue seule en apprentissage afin de finaliser celui-ci. Au contraire, le tableau 3 montre que le fait de ne pas inclure la tâche globale pour le pré-entraînement entraîne des problèmes également lors de la phase de finalisation de l'apprentissage. Cela confirme que le modèle doit apprendre conjointement les opérations intermédiaires et la tâche globale afin de permettre l'apprentissage plus efficace de la tâche globale cible.

5.2.3 Flexibilité

Dans notre modèle, l'entrée de la tâche globale (multiplication) est composée de quelques lignes vides (voir le tableau 1). Ces lignes vides constituent des étapes de récurrence pour le réseau lors de l'encodage. Afin de mesurer l'influence de ces lignes encodées, nous en avons fait varier la quantité lors de l'apprentissage de finition uniquement (l'entraînement initial est effectué avec 4 lignes doubles tel que précédemment).

Nous pouvons observer sur le tableau 4 que les taux d'erreur tendent à diminuer de façon monotone avec le nombre de récurrences additionnelles. Cela peut sembler logique puisque l'augmentation du nombre de récurrences augmente également la puissance de calcul du modèle.

5.2.4 Dynamique de l'apprentissage actif

Dans la section 4.4, nous avons présenté le mécanisme d'apprentissage actif que nous avons utilisé pour équilibrer l'effort d'apprentissage entre les différentes tâches soumises. La figure 3 représente l'évolution des erreurs au cours de l'apprentissage pour les différentes tâches. Nous pouvons observer qu'elle est relativement constante après un certain temps et que la tâche la plus difficile est bien la tâche globale. Cependant, on peut remarquer que la deuxième tâche la plus difficile est la procédure d'addition et non l'une des multiplications intermédiaires, dont le taux d'erreur moyen est même proche de 0. Ceci est surprenant car dans la littérature, l'addition semble être une tâche simple à apprendre. Cela peut cependant s'expliquer par les nombreux opérandes impliqués pour cette addition.

5.2.5 Sous-tâches uniquement

Nous n'entraînons le réseau que sur les sous-tâches (c'est-à-dire en excluant l'opération de bout en bout). Pour obtenir le résultat global, nous déroulons l'algorithme par une récurrence manuelle (les différentes sous-tâches "s'alimentent" consécutivement). Seule la sortie de l'addition finale comme résultat global est évaluée. Cette méthode a également été utilisée dans [34], avec un simple perception multi-couches. Cette tâche est plus facile à traiter (car l'opération globale est décomposée en ses étapes successives pour son exécution). En particulier, nous cherchons à estimer l'effet de la récurrence disponible dans notre réseau sachant que [21] utilise

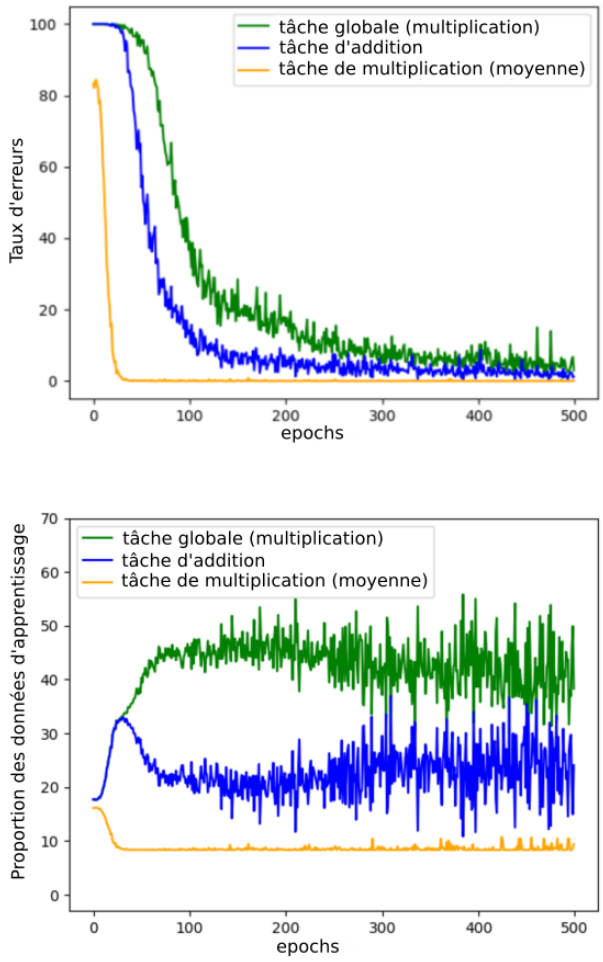

FIGURE 3 – (En haut) Évolution du taux d'erreur pour chaque tâche (estimé à partir de l'ensemble des données d'apprentissage) utilisé pour l'apprentissage actif. (En bas) Évolution associée de la proportion des différentes tâches dans l'ensemble des données d'apprentissage

un réseau sans récurrence.

Le tableau 5 montre que notre modèle réduit le taux d'erreur, dans ce contexte d'exécution. Cela montre que le modèle récurrent et la représentation des données que nous utilisons parviennent à apprendre toutes les sous-tâches.

	test = récurrence manuelle
	4×4 restreinte (7 chiffres en sortie)
[21]	2 %
UAT	**0.31% (± 0.11%)**

TABLE 5 – Comparaison du taux d'erreur entre [Nollet et al., 2020] et UAST, en combinant des tâches secondaires uniquement grâce à une récurrence manuelle

6 Conclusion and perspectives

Malgré ses multiples succès, les réseaux profonds rencontrent des difficultés à apprendre les algorithmes, comme certaines opérations arithmétiques. Cela s'explique par la complexité de la tâche, en particulier les dépendances à long terme, mais correspond aussi à un problème d'entraînement rencontré par de multiples modèles.

Cette limitation peut également être observé pour les modèles de langue, dont les performances proviennent du corpus disponible, et ne s'applique pas bien aux inférences algorithmiques lorsque la variabilité des valeurs des paramètres est importante et que le processus de calcul de l'algorithme implique de nombreuses étapes, comme pour la multiplication arithmétique.

Dans cet article, nous proposons une méthode originale pour apprendre la multiplication à plusieurs chiffres de bout en bout de deux opérandes (décimaux), en guidant le modèle par l'introduction de sous-tâches (ou sous-routines) en même temps que la tâche cible, tout en appliquant une stratégie d'apprentissage actif entre tâches. Nous montrons à travers l'analyse de nos expériences que l'amélioration des performances mesurée est directement causée par la méthode d'apprentissage que nous introduisons. L'apprentissage de la tâche globale suite à celui des sous-tâches peut améliorer les résultats. Cependant, la meilleure façon de procéder est sans équivoque de soumettre l'ensemble des tâches simultanément à l'apprentissage pour permettre d'amorcer la tâche d'apprentissage globale de bout en bout (la multiplication complète). Une fois amorcée, la tâche globale peut poursuivre son apprentissage seule de manière efficace.

En finalisant l'apprentissage sur la tâche globale, nous montrons une propriété supplémentaire intéressante de notre méthode d'apprentissage. Une fois que la multiplication globale est apprise, le réseau peut s'adapter à un nombre de récurrences différent de celui pour lequel il a initialement été paramétré. Cela semble indiquer que le modèle est capable non seulement de combiner les sous-tâches pour résoudre la tâche globale de bout en bout, mais aussi d'extraire et d'adapter de manière autonome une sorte de connaissance de haut niveau. La restriction des récurrences fournies pour le calcul et le maintien de la précision ressemble à une parallélisation contrainte de la tâche algorithmique. Bien que l'approche décrite permette d'apprendre plus efficacement la multiplication, elle met en oeuvre de nombreuses récurrences qui limitent les capacités de résolution pour les opérations de plus grande taille. Le problème du passage a l'échelle a également été observé sur les autres approches pour la multiplication décimale. Tout en conservant le mixage des différentes sous-tâches lors de l'entraînement algorithmique, qui est central à l'approche proposée, certaines optimisations pourraient être étudiées pour de futurs travaux.

La principale motivation de ce travail n'est pas seulement la résolution de la multiplication, mais également de développer un procédé permettant d'atténuer le problème d'entraînement difficile qui a été observé pour les algorithmes. Il serait intéressant d'étudier comment permettre d'apprendre des algorithmes afin de réaliser de meilleures inférences lorsque

une approche statistique ne permet pas de généraliser, tel qu'observé pour les multiplications sur les grands modèles de langage.

Ce travail soulève également de nombreuses questions de recherche concernant la dynamique d'apprentissage. Il serait intéressant d'étudier plus précisément comment le transfert des étapes intermédiaires vers la tâche globale est réalisé par le réseau. Pour surmonter le problème d'apprentissage rencontré, nous fournissons toutes les étapes intermédiaires au réseau pendant l'apprentissage. Un cas spécifique que nous souhaiterions étudier serait de ne fournir que certaines des tâches de soutien et d'observer la capacité du réseau à compléter les tâches inconnues de lui-même. Pour cela, il serait intéressant d'étudier le mécanisme de transfert des étapes intermédiaires vers la tâche globale. Cette compréhension pourrait nous donner la possibilité de contrôler le flux d'interaction entre les étapes de soutien et la dynamique générale du transfert algorithmique, et donc de peut-être mieux appréhender la façon dont une IA pourrait exploiter et adapter ses connaissances algorithmiques afin d'étendre ses capacités.

Remerciements

Ce travail a été réalisé en utilisant les ressources HPC de GENCI-IDRIS et un GPU offert par @NVIDIA Corporation. par @NVIDIA Corporation. Nous remercions chaleureusement ce soutien.

Références

[1] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep ? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33 :1877–1901, 2020.

[4] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv :1808.04355*, 2018.

[5] Kaiyu Chen, Yihan Dong, Xipeng Qiu, and Zitian Chen. Neural arithmetic expression calculator. *CoRR*, abs/1809.08590, 2018.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[7] Sungjae Cho, Jaeseo Lim, Chris Hickey, and Byoung-Tak Zhang. Problem difficulty in arithmetic cognition : Humans and connectionist models. 2019.

[8] Mark Collier and Joeran Beel. Implementing neural turing machines. In *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 94–104, Cham, 2018. Springer International Publishing.

[9] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4) :303–314, 1989.

[10] Karlis Freivalds and Renars Liepins. Improving the neural gpu architecture for algorithm learning. *arXiv preprint arXiv :1702.08727*, 2017.

[11] Isha Ganguli, Rajat Subhra Bhowmick, and Jaya Sil. Study of recurrent neural network models used for evaluating numerical expressions. In *2019 IEEE Region 10 Symposium (TENSYMP)*, pages 403–408. IEEE, 2019.

[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[14] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626) :471–476, 2016.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[16] Yedid Hoshen and Shmuel Peleg. Visual learning of arithmetic operations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3733–3739. AAAI Press, 2016.

[17] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv :1511.08228*, 2015.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv :1912.01412*, 2019.

[20] Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *International Conference on Learning Representations*, 2020.

[21] Bastien Nollet, Mathieu Lefort, and Frédéric Armetta. Learning Arithmetic Operations With A Multistep Deep Learning. In *The International Joint Conference on Neural Networks (IJCNN)*, Glasgow, United Kingdom, July 2020.

[22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang,

Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv :2203.02155*, 2022.

[23] Eric Price, Wojciech Zaremba, and Ilya Sutskever. Extensions and limitations of the neural gpu. *arXiv preprint arXiv :1611.00736*, 2016.

[24] Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot numerical reasoning. In *Findings of the Association for Computational Linguistics : EMNLP 2022*, pages 840–854, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[25] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv :1904.01557*, 2019.

[26] Daniel Schlör, Markus Ring, and Andreas Hotho. inalu : Improved neural arithmetic logic unit. *arXiv preprint arXiv :2003.07629*, 2020.

[27] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[28] H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1) :132 – 150, 1995.

[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[30] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and Larry Zitnick. Elf opengo : An analysis and open reimplementation of alphazero. In *International conference on machine learning*, pages 6244–6253. PMLR, 2019.

[31] Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems 31*, pages 8035–8044. Curran Associates, Inc., 2018.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[33] Artit Wangperawong. Attending to mathematical language with transformers. *arXiv preprint arXiv :1812.02825*, 2018.

[34] Yujun Yan, Kevin Swersky, Danai Koutra, Parthasarathy Ranganathan, and Milad Hashemi. Neural execution engines : Learning to execute subroutines. *CoRR*, abs/2006.08084, 2020.

[35] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv :1707.08114*, 2017.