Tentative de prédiction de la réussite à un exercice de programmation

Maxime Bouton[1], Yvan Peter[1], Marielle Léonard[1,2], and Nour El Mawas[2]

[1] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
prénom.nom@univ-lille.fr

[2] Univ. Lille, ULR 4354 - CIREL - Centre Interuniversitaire de Recherche en Éducation de Lille, F-59000 Lille, France
prénom.nom@univ-lille.fr

Résumé Nous présentons dans cet article une première approche d'utilisation d'un algorithme d'apprentissage basé sur le mécanisme d'attention pour prédire le succès d'un élève dans le cadre d'un exercice de programmation par blocs. Nous comparons nos résultats à des résultats de référence obtenus sur le jeu de données ASSIT2015. Les résultats obtenus sont satisfaisants, avec un AUC maximal de 75%, mais présentent une évolution contre-intuitive lorsque les paramètres du modèle sont modifiés. Nous proposons finalement des explications et des pistes pour y remédier.

Keywords: Programmation par blocs · Apprentissage de la programmation · Mécanisme d'attention · Traçage de connaissances

1 Introduction

L'apprentissage de l'informatique et notamment de la programmation a fait son retour dans les programmes scolaires du primaire jusqu'au lycée. Cela pose de nombreuses questions sur la didactique de l'informatique, les ressources pédagogiques, la formation des enseignants... Nous avons commencé à étudier les questions de didactique dans le cadre de l'apprentissage de la programmation à l'école primaire et au collège [6, 7] en utilisant un langage de programmation par blocs dans un environnement en ligne mis à disposition par l'association France-IOI [3]. L'analyse des données prélevées sur la plateforme nous a permis d'identifier des paliers de difficultés, en particulier lors de l'apprentissage de la notion de répétition [6].

Dans cet article, nous présentons une première approche de l'utilisation d'un algorithme d'apprentissage profond destiné à prédire le résultat d'un élève (réussite ou échec) face à un problème donné. Pour cela nous nous sommes basés sur la proposition de Pandey and Karypis [8]. Afin d'avoir un point de référence, nous avons répliqué l'étude réalisée dans leur article sur le jeu de

3. http://www.france-ioi.org/

données ASSIST2015 [4] et nous avons appliqué l'algorithme sur notre propre base de données.

Dans une première partie nous présentons un état de l'art des travaux portant sur les algorithmes d'apprentissage modélisant l'acquisition des *Composantes de Connaissances* (*Knowledge Tracing* (KT)) pour la prédiction de la réussite à un exercice. Dans un deuxième temps, nous décrivons le modèle, son fonctionnement général, son implémentation et le formatage des données nécessaire à son utilisation. Nous présentons ensuite les résultats obtenus sur les données ASSIST2015 et les nôtres avant de conclure.

2 État de l'art

Le traçage de connaissances (*knowledge tracing*) [2] cherche à suivre l'évolution des connaissances d'un apprenant à travers les activités pédagogiques réalisées afin de permettre une personnalisation de l'apprentissage et de prédire la performance de cet étudiant. Ce traçage de connaissances s'appuie sur les interactions de l'apprenant avec les *composantes de connaissances* (CC) du domaine étudié. Ces CC sont associées aux ressources pédagogiques et exercices proposés. Le résultat de l'interaction (par ex. exercice réussi ou non) est utilisé pour modéliser le niveau de maîtrise de la ou des CC associée(s). Ce traçage de connaissance a été introduit par Corbett and Anderson [2], qui ont utilisé un réseau Bayésien (*Bayesian Knowledge Tracing* (BKT)) pour modéliser la probabilité de maîtrise des différentes CC indépendamment. Deux revues récentes de la littérature font le point sur les algorithmes de traçage des connaissances d'une manière générale [3] et dans le cadre particulier de l'apprentissage de la programmation [5]. Dans l'ensemble, ces algorithmes utilisent les résultats des exercices pour réaliser leur prédiction. Certains modèles sont enrichis de données relatives à l'apprenant (compétences acquises, temps de réponse...) ou aux exercices (relations entre CC, nombre d'essais...). Ces deux revues de littérature identifient les familles d'algorithmes suivantes :

- les modèles probabilistes bayésiens (*Bayesian Knowledge Tracing*) cherchent à associer à chaque CC un état *connu/non connu* en s'appuyant sur les résultats des exercices. La principale critique de ce type d'algorithme porte sur le fait qu'il ne considère qu'une composante de connaissance par exercice. Un ensemble de propositions ont été faites pour pallier ce problème mais elles mènent à des problèmes de calculabilité.
- les modèles utilisant une régression logistique, type *Additive Factor Model* [4]. Ceux-ci sont adaptés à de multiples CC par exercices.
- les modèles basés sur de l'apprentissage profond. Les premiers apparus dans le contexte du KT comme le Deep Knowledge Tracing (DKT) [9], montrent de bonnes performances, mais peinent à se généraliser sur des

4. https://sites.google.com/site/assistmentsdata/datasets/2015-assistments-skill-builder-data

bases de données relativement creuses. Des alternatives utilisant des réseaux à mémoire dynamique clé-valeur (DKVMN) [10] permettent une meilleure interprétabilité des paramètres. Plus récemment, des modèles comme Self-Attentive Knowledge Tracing (SAKT) [8], basés sur les mécanismes d'attention [1], permettent *a priori* une meilleure généralisation des résultats.

3 Modèle

Nous avons implémenté le modèle défini par Pandey and Karypis [8] pour prédire le résultat d'un exercice en fonction de l'historique d'un apprenant. Celui-ci utilise le mécanisme d'attention développé initialement dans le cadre du Traitement Automatique du Langage (TAL) et vise à prédire le résultat au prochain exercice en fonction de l'historique des résultats précédents.

3.1 Mécanisme d'attention

L'objectif du mécanisme d'attention est de capturer de l'information de contexte dans une séquence de *tokens* (mots) où l'information contenue par l'ensemble est plus importante que la somme des informations individuelles de chaque *token*.

Par exemple, selon le contexte de la phrase, le mot *cartes* peut avoir différentes significations (cartes à jouer, géographiques, postales...). La signification réelle du mot ne peut être déduite que grâce au contexte. Nous décrivons dans la suite le fonctionnement général du mécanisme d'attention [1] présenté figure 1.

Considérons la séquence *Il joue aux cartes*. Chaque token de cette séquence peut être encodé sous forme de vecteur : ce changement de représentation est appelé *embedding*. L'intérêt, en plus d'avoir une représentation mathématique plus pratique de l'objet, c'est que les vecteurs sont dotés du *produit scalaire* : 2 mots peuvent alors être «comparés». Plus la valeur du produit scalaire entre les *embeddings* de mots est grande, plus ces mots sont susceptibles d'être liés.

Cependant, le produit scalaire entre les embeddings de mot est trop limité. Parfois, des mots qui ne sont pas du tout similaires (comme un verbe et un nom par exemple) peuvent avoir une relation forte dans la séquence : leur produit scalaire devrait également être élevé. Dans notre exemple, on s'attend à ce que le mot *cartes* soit fortement lié à *joue*, puisque c'est ce dernier qui donne son sens à l'autre.

L'idée de l'*attention* pour obtenir des niveaux de relations plus fins entre les *tokens* est de les projeter linéairement sur d'autres espaces pour construire 3 vecteurs : la **Key** (K), la **Query** (Q) et la **Value** (V). Ces projections sélectionnent les composantes les plus pertinentes des embeddings et les orientent afin que le produit scalaire reflète les relations importantes.

Pour des questions de stabilité numérique, la fonction d'activation *softmax* est ensuite appliquée à la matrice des produits scalaires pour que les relations

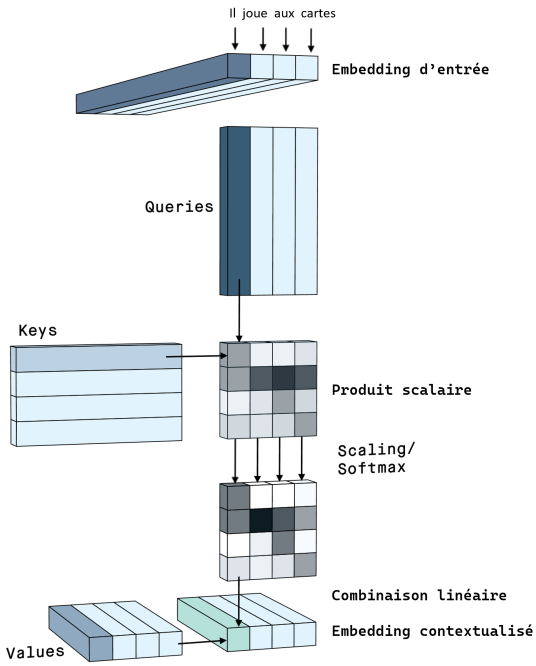

Figure 1. Schéma du mécanisme d'attention
(adapté de https://peltarion.com/blog/data-science/self-attention-video)

les plus fortes deviennent exponentiellement plus importantes. Cette opération introduit également de la non-linéarité dans le modèle, permettant ainsi au processus d'être répété plusieurs fois, pour obtenir des transformations de plus en plus précises.

Grâce à une combinaison linéaire de la matrice résultante avec les *Values*, les embeddings contextualisés des mots sont enfin obtenus. Ces embeddings de mots sont alors plus riches car adaptés au contexte. Ce mécanisme, appliqué à notre séquence d'exercices, revient à mettre en avant dans l'historique de l'apprenant les exercices qui ont le plus d'impact sur la résolution de l'exercice suivant (i.e., ceux qui sont en lien avec la ou les CC impliquée(s) dans cet exercice).

3.2 Structure et implémentation

Maintenant que le mécanisme central a été présenté, nous allons décrire la structure générale de l'ensemble du modèle schématisé en figure 2.

Données d'entrée Le modèle prend en entrée 3 vecteurs de même longueur fixée n (hyper-paramètre du modèle) :

— vecteur **Interactions** : représentant les interactions antérieures d'un apprenant (i.e., l'historique des paires exercice - résultat) ;

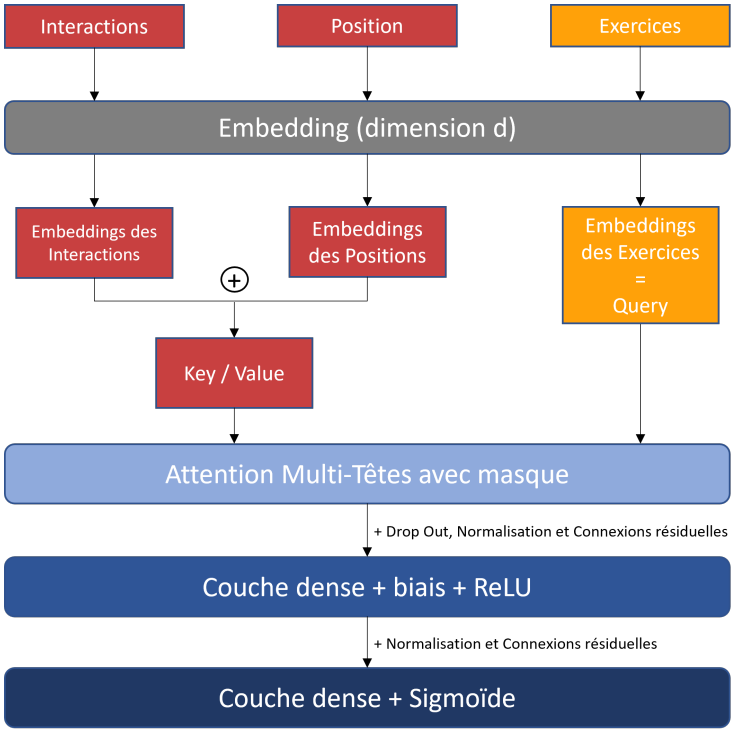

Figure 2. Schéma du modèle utilisé

— vecteur **Position** : indiquant l'ordre de réalisation des interactions (i.e. l'ordre dans lequel les exercices ont été réalisés) ;
— vecteur **Exercices** : contenant les derniers identifiants d'exercices réalisés. C'est sur ce dernier vecteur que porte la prédiction lors de l'entraînement du modèle.

Embeddings La dimension d'Embedding, c'est-à-dire la taille du vecteur censé représenter un token (ici une interaction ou un identifiant d'exercice), est noté d. Il s'agit d'un hyper-paramètre du modèle. On a les embeddings de tailles $n \times d$ suivants :

— **Embedding du vecteur Exercices** : la taille du vocabulaire pour cet embedding est E (car la bibliothèque est composée de E exercices différents) ;
— **Embedding du vecteur Interactions** : la taille du vocabulaire pour cet embedding est de $2 \times E$ (en considérant les E exercices différents, et un booléen pour le résultat obtenu) ;
— **Embedding du vecteur Position** : la taille du vocabulaire pour cet embedding est n (car il y a n positions dans une séquence).

Les embeddings des vecteurs Interactions et Position sont ensuite sommés avant de passer dans la couche d'*attention*.

Couche d'attention Le mécanisme d'attention décrit précédemment est mis en œuvre dans cette couche. La seule particularité notable est l'utilisation d'un masque destiné à ne prendre en compte que les interactions passées pour chaque exercice considéré (i.e., le résultat d'un exercice ne dépendant que des interactions précédentes), alors qu'en TAL, on considère l'ensemble des *tokens* d'une phrase.

Connections résiduelles et Normalisation Les connexions résiduelles consistent à sommer une couche inférieure à une couche supérieure, afin de propager les caractéristiques inférieures à travers le réseau et les rapprocher de la prédiction. Si ces caractéristiques se trouvent être importantes dans la prédiction, elles seront plus facilement prises en compte. La normalisation est utilisée pour la stabilité numérique et l'accélération du processus d'entraînement.

Couches denses Pour introduire de la non-linéarité à la sortie de la couche d'attention, qui fournit toujours une combinaison linéaire de valeurs, et pour considérer également les interactions des dimensions latentes, 2 couches denses avec biais et activation *Rectified Linear Unit* (ReLU) sont utilisées sur la sortie de la couche d'attention.

Prédiction et fonction de perte Une couche dense avec activation sigmoïde est finalement utilisée pour calculer la probabilité de réussite p_i. La fonction de perte qui doit être minimisée lors de l'apprentissage est l'entropie croisée :

$$\mathcal{L} = \sum_i \Big(r_i \log(p_i) + (1 - r_i) \log(1 - p_i) \Big)$$

3.3 Présentation et formatage des données d'entrée

Dans cette partie nous abordons la mise en forme des traces d'activité capturées sur la plateforme de France-IOI en vu de leur traitement par le modèle.

Définitions et formalisme Un apprenant est représenté par son historique, à savoir, la suite des *interactions* retraçant, dans l'ordre de leur réalisation, les exercices qu'il a tentés.
Plus formellement, l'historique d'un apprenant s est associé à $\big(e_t, r_t\big)_{1 \leq t \leq T_s}$, où

— $e_t \in [\![1, E]\!]$ est l'identifiant d'un exercice de la base de donnée (qui contient E exercices distincts)

— $r_t \in \{0, 1\}$ représente le résultat obtenu (1 si correct, 0 sinon)
— (e_t, r_t) est appelé une *interaction*
— $T_s \in \mathbb{N}$ représente le nombre total de tentatives (également appelées *soumissions*) de l'apprenant

Des exemples d'historiques sont donnés dans le tableau 1. On note ici $S = \max_s(T_s)$, le plus grand nombre de soumissions atteint par l'un des apprenants : chaque apprenant a donc réalisé au plus S tentatives. Dans cet exemple, la bibliothèque à disposition des apprenants dispose de 100 exercices (i.e. $E = 100$).

Tableau 1. Exemple d'historiques d'apprenants (avec ici, $E = 100$)

ID Apprenant	Soumission 1	Soumission 2	Soumission 3	...	Soumission S
0	15 0	15 1	20 0	... ...	17 1
1	35 0	62 1	35 1	... ...	- -
2	20 1	- -	- -	... ...	- -

Comme on peut le voir, l'apprenant **0** a réalisé S tentatives : une première ratée sur l'exercice 15, puis une seconde, cette fois réussie. Il s'est ensuite essayé à l'exercice 20, etc. Les lignes suivantes montrent d'autres cas typiques d'historique : les exercices ne sont pas forcément résolus dans l'ordre et un apprenant peut revenir sur un exercice (apprenant 1) et la séquence d'exercices réalisée peut être arbitrairement courte (apprenant 2).

Formatage des données d'entrée Les exemples d'historiques du tableau 1 permettent de constater que les apprenants n'ont pas nécessairement tous réalisé le même nombre de soumissions. Or le modèle utilisé ne fonctionne qu'avec des séquences de même longueur, fixée arbitrairement, et dénotée n. Ces historiques doivent donc être uniformisés, et transformés par la suite pour nourrir le modèle selon deux étapes décrites ci-après.

Découpage en séquences de même longueur : le découpage d'un historique s présente 3 cas, en fonction de sa longueur initiale T_s :

1. $T_s = n + 1$: l'historique est conservé tel quel
2. $T_s \leq n$: la paire $(0, 0)$ est ajoutée en amont de l'historique, jusqu'à atteindre un historique de taille $n + 1$. Ce *padding*, met en jeu des interactions qui ne correspondent à aucun exercice de la base afin de ne pas introduire de biais.
3. $T_s > n+1$: on divise l'historique en séquences de longueur $n+1$ en partant de la droite, et on complète si besoin la séquence de début d'historique comme dans le cas 2 (voir l'exemple en figure 3).

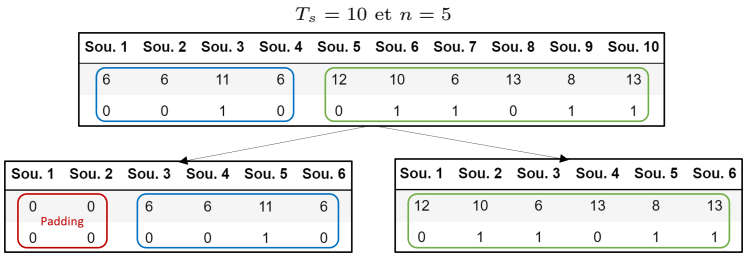

Figure 3. Illustration du découpage dans le cas $T_s > n + 1$

Transformation en données d'entrée : à partir de chacune de ces séquences de longueur $n + 1$, deux entrées de longueur n vont être construites pour l'algorithme d'apprentissage (exemple figure 4) :

— la première est le vecteur **Exercices** (ou *Exercise Input*). C'est un vecteur composé des identifiants d'exercices des n dernières tentatives de la séquence. Dans le mécanisme d'attention (cf. section 3.1), ce vecteur joue le rôle de la **Query**, et le modèle est évalué sur la prédiction qu'il donne au dernier élément de ce vecteur.

— la seconde est le vecteur **Interactions** (ou *Interaction Input*). Il est composé des n premières interactions de la séquence. Le modèle ne pouvant travailler qu'avec des vecteurs d'entiers, chaque paire (e_t, r_t) est représentée de manière unique par un entier $y_t = e_t + E \times r_t$. Dans le mécanisme d'attention (cf. section 3.1), c'est ce vecteur qui va jouer le rôle de **Key** et de **Value**.

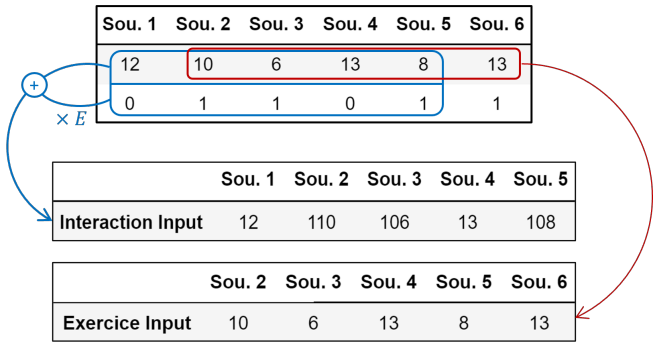

Figure 4. Exemple de transformation en vecteurs d'entrée ($E = 100$). Ici, le modèle doit prédire la probabilité que l'élève réussisse l'exercice 13 (dernier élément du vecteur **Exercices**) à sa prochaine tentative (i.e. *Soumission 6*). Pour évaluer cette probabilité le modèle utilise comme contexte l'ensemble des interactions antérieures de l'élève représentées par le vecteur **Interactions**. Si le modèle est performant et bien entraîné, alors la probabilité prédite par le modèle devrait être proche de 1 car l'élève a effectivement réussi.

4 Résultats

En guise de référence, nous avons dans un premier temps appliqué ce modèle à la base ASSIST2015 et retrouvons, dans les mêmes configurations, les résultats annoncés par Pandey and Karypis [8]. Dans un second temps, nous l'avons appliqué à notre propre jeu de données. Ces dernières sont issues de la plateforme Chticode, mise à disposition par l'association France-IOI, et dédiée aux activités de recherche sur l'apprentissage de la programmation du laboratoire CRIStAL de l'université de Lille. Les problèmes analysés sont les mêmes que ceux du concours Algorea [5].

Pour comparer les performances obtenues par le modèle sur les différents jeux de données, nous avons choisi d'utiliser 3 métriques différentes, chacune représentée sur la figure 5 en fonction de la longueur de séquence n choisie :

— l'*Accuracy* : correspondant au taux de bonnes prédictions du modèle (graphique 5.A)
— l'*Aire sous la courbe ROC* (AUC) : métrique plus fine que la précédente, prenant en compte les taux de vrais et faux positifs en fonction du seuil de classification choisi (graphique 5.B)
— l'*Aire sous la courbe Précision/Rappel* (AUCPR) : métrique similaire à l'AUC, mais plus pertinente vis-à-vis de données très déséquilibrées (graphique 5.C)

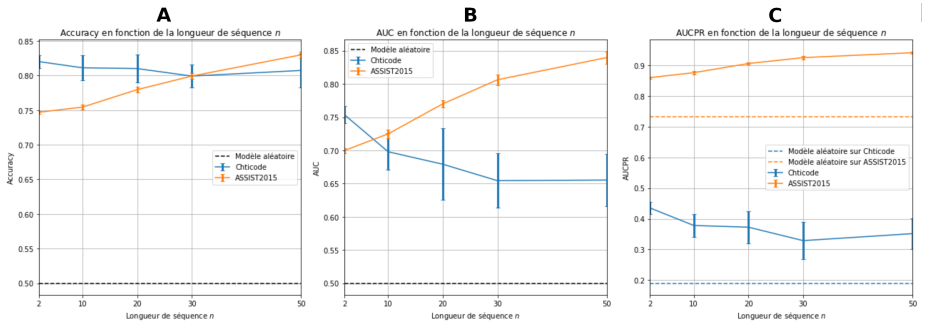

Figure 5. Résultats comparatifs Chticode / ASSIST2015 : Les droites pointillées représentent la valeur moyenne obtenue par un modèle aléatoire (i.e., le modèle sans entraînement). À noter que dans le cas de l'AUCPR (graphique **C**), cette valeur dépend du jeu de données choisi.

Le graphique 5.A donne l'impression que le modèle obtient de bons résultats sur les données Chticode (courbe bleue), indépendamment de n. Ces valeurs s'expliquent par l'équilibre des classes des données Chticode : 20% de tentatives réussies contre 80% d'échecs. Un modèle qui prédirait donc systématiquement un échec aurait alors lui aussi une accuracy de 80%.

5. https://algorea.org/main.html

Le graphique 5.B montre une claire détérioration des performances du modèle sur les données de Chticode lorsque n augmente, ce qui est contre-intuitif. En effet, on s'attend à obtenir de meilleurs résultats lorsque la longueur de séquence n augmente, puisque le modèle dispose d'un vecteur **Interactions** plus grand et donc d'une meilleure visibilité sur l'état de connaissance de l'apprenant. C'est bien ce qu'on observe sur les données ASSIST2015 (courbes orange), contrairement à Chticode. Sur nos données, l'amélioration du contexte lié à l'augmentation de n ne compense pas la diminution du nombre d'échantillons résultant (car en effet, un n plus grand implique moins de séquences produites lors du découpage présenté en 3.3, et donc moins d'échantillons pour l'entraînement).
Cette particularité peut s'expliquer par des caractéristiques différentes des données (cf. tableau 2). Par exemple, le ratio interactions/exercices est près de 23 fois plus important pour ASSIST2015 : le nombre de séquences contenant un exercice donné est donc plus grand, ce qui confère au modèle plus d'exemples pour capter les relations intrinsèques aux interactions. De plus, le rapport des classes n'est pas le même, il est non seulement inversé, mais davantage déséquilibré pour Chticode, avec près de 1/5 de positifs, contre près de 3/4 pour ASSIST2015. Ce déséquilibre joue donc sur le modèle : lorsque la longueur de séquence n diminue, le modèle prédit avec une forte probabilité un label négatif, ce qui s'avère correct dans près de 80% des cas.

Le graphique 5.C montre que du point de vue de la prédiction des éléments positifs (en minorité sur Chticode), le gain en performance avant-après entraînement est quasiment indépendant du jeu de données.

Finalement, quelles que soient la métrique utilisée et la valeur de n choisie, le modèle obtient des résultats corrects (gain minimum de 15% après entraînement), même si ces derniers sont sensiblement moins bons sur les données Chticode.

Tableau 2. Comparaison des deux jeux de données

	Chticode	ASSIST2015
E	154	100
Nombre d'étudiants	580	19840
Nombre total d'interactions	47630	683801
Ratio Interactions / Exercices	309	6838
Taux de soumissions correctes (%)	19	73
Nombre minimum d'interactions par élève	1	1
Nombre maximum d'interactions par élève	372	618
Nombre moyen d'interactions par élève	82	34
Médiane (interactions par élève)	71	20
Ecart type (interactions par élève)	52	41
Nombre moyen d'exercices différents par élève	19	6

5 Conclusion & Perspectives

Nous nous intéressons aux modèles de traçage de connaissances visant à prédire la réussite d'un apprenant à un exercice en s'appuyant sur l'historique de ses résultats précédents. L'étude des revues de la littérature (en particulier [3]) indique un fort développement des modèles basés sur l'apprentissage profond depuis 2017. Ceux-ci semblent plus performants (sous réserve de la quantité de données disponible pour l'entraînement).

Le travail présenté ici visait à évaluer un modèle de traçage de connaissances récent, basé sur de l'apprentissage profond, avec les données prélevées dans le cadre de nos expérimentations en apprentissage de la programmation. L'implémentation du modèle proposé par Pandey and Karypis [8] nous a permis de retrouver les résultats qu'ils avaient atteints sur le jeu de données ASSIST2015. Si on compare ces résultats avec ceux obtenus sur nos données, on constate que l'amélioration de la prédiction suite à l'entraînement est sensiblement équivalente mais à des niveaux différents. Cette différence s'explique par la quantité et la nature des données dont nous disposons (nombre d'interactions pour un exercice dans les données d'entraînement et déséquilibre entre les exercices réussis ou non). Nous observons un comportement de prédiction différent en fonction de la longueur de séquence n choisie. Les performances semblent diminuer lorsqu'elle augmente. Ce modèle ne se généralise donc pas nécessairement très bien lorsque la base de données est plus creuse et/ou moins conséquente.

Plusieurs solutions peuvent être envisagées pour tenter de résoudre ce problème, comme l'enrichissement des données d'entrées (en travaillant sur le regroupement d'exercices) ou en changeant l'espace de prédiction (comme prédire le nombre de tentatives nécessaires plutôt que la probabilité de réussite).

Remerciements

Ce travail est partiellement financé par le projet Interreg Teach Transition (https://teachtransition.eu).

Bibliographie

[1] Bahdanau, D., Cho, K., Bengio, Y. : Neural Machine Translation by Jointly Learning to Align and Translate (May 2016)

[2] Corbett, A.T., Anderson, J.R. : Knowledge Tracing : Modeling the Acquisition of Procedural Knoweldge. User modeling and user-adapted interaction **4**(4), 253–278 (1994)

[3] Dai, M., Hung, J.L., Du, X., Tang, H., Li, H. : Knowledge Tracing : A Review of Available Technologies. Journal of Educational Technology Development and Exchange **14**(2), 1–20 (2021), ISSN 19418035, https://doi.org/10.18785/jetde.1402.01, URL https://aquila.usm.edu/jetde/vol14/iss2/1/

[4] Durand, G., Goutte, C., Belacel, N., Bouslimani, Y., Leger, S. : Review, computation and application of the additive factor model (afm). Nat. Res. Council Canada, Winnipeg, MB, Canada, Tech. Rep **23002483** (2017)

[5] Lei, P.I., Mendes, A.J. : A systematic literature review on knowledge tracing in learning programming. In : 2021 IEEE Frontiers in Education Conference (FIE), pp. 1–7, IEEE, Lincoln, NE, USA (Oct 2021), ISBN 978-1-66543-851-3, https://doi.org/10.1109/FIE49875.2021.9637323, URL https://ieeexplore.ieee.org/document/9637323/

[6] Léonard, M., Peter, Y., Secq, Y. : Reconnaissance et synthèse de motifs redondants avec des élèves de 6-7 ans. Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation **27**(2), 13 (Oct 2020)

[7] Léonard, M., Peter, Y., Secq, Y., Alvarez, J., Fluckiger, C. : MOTIF..MOTIF.. : initier à la notion de répétition en maternelle sans mobiliser de repérage spatial. Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation **28**, 35 (2021)

[8] Pandey, S., Karypis, G. : A Self-Attentive model for Knowledge Tracing. 12th International Conference on Educational Data Mining (2019)

[9] Piech, C., Spencer, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L., Sohl-Dickstein, J. : Deep Knowledge Tracing. Advances in neural information processing systems (2015)

[10] Zhang, J., Shi, X., King, I., Yeung, D.Y. : Dynamic Key-Value Memory Networks for Knowledge Tracing. Proceedings of the 26th International Conference on World Wide Web (Feb 2017)