Vers des Réseaux de Neurones Récurrents Flexibles

Françoise Le Bolzer[1], Anne Lambert[1], et François Schnitzler[1]

[1]InterDigital Inc.

3 juin 2020

Résumé

Nous présentons une méthode permettant d'ajuster le point de fonctionnement des réseaux de neurones récurrents (RNN) pendant l'analyse d'une séquence en jouant sur le compromis précision/coût de calcul. Cela ouvre la possibilité d'adapter les RNN en temps réel pour s'adapter aux contraintes de calcul lors de l'exécution. Cela a un intérêt sur du matériel dont les ressources sont partagées entre plusieurs processus, notamment sur des équipements mobiles. Cette approche implique des modifications minimes du modèle et évite donc de charger de nouveaux paramètres depuis une mémoire lente. Nous évaluons cette approche sur 2 types de tâches : l'addition et la reconnaissance d'activité humaine. Nous démontrons que cette approche se comporte bien et modifie le comportement d'un RNN. La précision et le coût de calcul qui en résultent, correspondent à une moyenne pondérée des différentes configurations utilisées. Le poids de chaque configuration augmente avec le nombre de pas de temps pendant lequel cette configuration est utilisée.

Mots-clef : Réseaux de neurones récurrents, Flexibilité, Ressources de calcul.

1 Introduction

Ce papier propose les Réseaux de Neurones Récurrents Flexibles (FRNN) : des réseaux de neurones récurrents dont le coût de calcul peut être modifié dynamiquement au cours de l'analyse d'une séquence. Les réseaux de neurones récurrents sont un type d'architecture d'apprentissage profond spécialement conçus pour traiter des données séquentielles. Ils ont été largement utilisés pour analyser des données telles que l'audio [ZSLC17], la vidéo [YWH+16] ou d'autres séries temporelles collectées par des capteurs [HHP16]. Les modèles d'apprentissage profond sont de plus en plus souvent exécutés sur les équipements des utilisateurs, au dépend d'une exécution dans le cloud. La réduction du coût de calcul de ces modèles a par conséquent été beaucoup étudiée récemment. Diverses approches permettent de réaliser des économies substantielles en coût de mémoire et / ou de calcul. Cependant, elles ne permettent pas d'adapter ces coûts au contexte d'exécution et de répondre dynamiquement aux éventuels changements de charge de l'environnement de calcul.

Or les équipements personnels, comme les smartphones, intègrent désormais plusieurs processeurs, certains dédiés à l'exécution de réseaux de neurones profonds. Ces équipements exécutent de plus en plus de modèles d'apprentissage profond qui peuvent fonctionner en parallèle. En fonction des ressources disponibles, certains processus devront peut-être être arrêtés pour libérer des ressources de calcul au profit de processus plus prioritaires. Sans flexibilité, cela pourrait conduire à l'arrêt d'un modèle récurrent en cours de séquence. Notre approche permet de limiter l'impact des variations de contexte en réduisant temporairement les besoins en calcul du modèle récurrent pour lui permettre de poursuivre son traitement. Cela est possible au prix d'une dégradation de la précision. De plus, la réduction des besoins en calcul s'accompagne d'une plus faible consommation d'énergie. Cette approche offre donc plus de flexibilité pour la gestion de la batterie.

Pour relever ces défis, cet article présente une méthode pour construire des réseaux de neurones récurrents flexibles (FRNN) permettant d'adapter leur coût de calcul au cours du traitement d'une séquence en jouant sur leur niveau de précision. A notre connaissance, c'est la première approche qui permette un tel ajustement à l'exécution. Cette méthode utilise un modèle unique et contrôle le coût de calcul via un seul paramètre. La précision et le coût de calcul qui en résultent, correspondent à une moyenne pondérée des différentes configurations utilisées. La

méthode proposée exploite des architectures existantes qui accélèrent les RNN par calcul conditionnel, tel que skip-RNN [CJiN+18]. Nous démontrons l'intérêt de la méthode avec skip-RNN, mais elle peut très bien être utilisée avec d'autres architectures. Cette méthode est illustrée sur une tâche d'addition et une tâche de reconnaissance d'activité humaine à base de vidéo.

2 Skip-RNN

Les FRNN que nous développons sont basés sur l'architecture skip-RNN [CJiN+18]. Dans cette section nous regardons de plus près ce modèle et introduisons des notations. Un RNN peut être vu comme une fonction qui accepte une séquence d'entrées $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ et qui calcule récursivement un ensemble d'états $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_T)$. Chaque état $\mathbf{s}_t$ est calculé à partir de $\mathbf{s}_{t-1}$ et $\mathbf{x}_t$ par une cellule S du RNN. Les architectures de RNN les plus utilisées en pratique sont les réseaux LSTM (Long Short-Term Memory : reseaux à mémoire court-terme et long terme) [HS97] et GRU (Gated Recurrent Unit : réseau récurrent à portes) [CVMG+14].

Skip-RNN augmente la cellule S pour permettre au modèle d'ignorer certaines entrées. Quand une entrée est traitée, une porte de mise à jour calcule une quantité $\Delta\tilde{u}_t$ qui décide du nombre d'entrées à ignorer. En pratique, les entrées $\mathbf{x}_t$ sont ignorées et $\tilde{u}_t$ incrémenté de $\Delta\tilde{u}_t$ à chaque pas de temps jusqu'à ce que $\tilde{u}_t \geq 0.5$. L'architecture résultante peut être décrite comme suit :

$$u_t = f_{binarize}(\tilde{u}_t) \tag{1}$$
$$\mathbf{s}_t = u_t S_t(\mathbf{s}_{t-1}, \mathbf{x}_t) + (1 - u_t)\mathbf{s}_{t-1} \tag{2}$$
$$\Delta\tilde{u}_t = \sigma(W\mathbf{s}_t + b) \tag{3}$$
$$\tilde{u}_{t+1} = u_t\Delta\tilde{u}_t + (1 - u_t)(\tilde{u}_t + \min(\Delta\tilde{u}_t, 1 - \tilde{u}_t)) \tag{4}$$

Dans ces équations, $f_{binarize}(z)$ désigne une fonction de binarisation ($f_{binarize}(z) = 0$ si $z < 0.5$ et 1 sinon), σ est une fonction non linéaire et W et b des paramètres. La fonction d'erreur utilisée pour entraîner ce modèle contient deux termes :

$$L_{skip_RNN} = L_{acc}(x, y) + \lambda L_{budget} \ . \tag{5}$$

Le premier mesure la précision de la tâche (par exemple en utilisant une entropie croisée pour un problème de classification ou une distance euclidienne pour une régression). Le second terme pénalise les opérations de calcul effectuées par le RNN : $L_{budget} = \sum_t u_t$. L'hyperparamètre λ pondère cette pénalité.

3 FRNN

Pour faciliter la présentation de la méthode, nous décrivons notre RNN flexible en partant d'une architecture basée sur la cellule skip-RNN. Comme skip-RNN, notre méthode est générique et peut être appliquée à plusieurs Architectures RNN dont LSTM (FLSTM) et GRU (FGRU).

La méthode est simple et ne nécessite qu'un changement minimal de l'architecture d'origine. Aucun réapprentissage du modèle skip-RNN n'est nécessaire. Notre approche modifie le seuil de la porte de mise à jour, celle qui décide du traitement (ou non) de l'entrée suivante. Par conséquent, le modèle est presque le même que le skip-RNN. Seule $f_{binarize}$ change. La fonction accepte ainsi un paramètre supplémentaire : thr. L'architecture résultante est identique à l'architecture décrite dans la section 2, à l'exception de l'équation 1 :

$$u_t = f_{binarize}(\tilde{u}_t, thr) = \begin{cases} 0 \text{ if } \tilde{u}_t < thr, \\ 1 \text{ otherwise .} \end{cases} \tag{6}$$

En faisant varier le paramètre thr, le comportement du modèle change. Lorsque thr augmente, le modèle saute plus d'entrées mais conserve sa capacité à traiter la tâche au prix d'une précision moindre. Bien qu'il s'agisse d'un simple changement de seuil, nos expériences suggèrent que l'apprentissage d'un modèle avec une valeur fixe pour thr permet d'ajuster le point de fonctionnement du modèle pendant l'inférence.

4 Résultats expérimentaux

Dans cette section, nous montrons que cette approche permet d'adapter le coût de calcul d'un RNN pendant l'inférence. Nous montrons également que la précision et le coût de calcul obtenus en changeant thr sont proches de la moyenne pondérée de la valeur correspondante pour les différentes configurations utilisées lors de l'inférence. Cela signifie que l'utilisation d'un RNN plus rapide pour libérer des ressources de calcul pendant une courte période par rapport à la longueur d'une séquence n'aura pas un grand impact sur la précision du résultat. Nous évaluons nos approches sur deux ensembles de données :

Addition : Cette tâche repose sur des données synthétiques correspondant à une tâche d'addition, couramment utilisée pour évaluer les performances des modèles RNNs [HS97, NPL16, CJiN+18]. Chaque échantillon d'entrée se présente sous la forme d'une séquence de paires (valeur, marqueur) où les éléments

valeur sont échantillonnés uniformément dans la plage (-0.5, 0.5) et où les éléments marqueurs sont 1 ou 0. La sortie de la tâche est la somme de toutes les valeurs avec un élément marqueur égal à 1. Pour générer l'ensemble de données, nous avons reproduit la configuration expérimentale définie dans [NPL16]. Seules deux valeurs ont été marquées pour l'addition. Le premier marqueur a été placé au hasard parmi les premiers 10% de la séquence. De même, le deuxième marqueur a été placé dans la dernière moitié de la séquence. L'apprentissage a été effectué avec des batchs de 256 séquences qui ont été générés au hasard pendant la boucle d'entraînement. La validation a été effectuée sur 15 autres batchs. Nous avons utilisé la même architecture que [CJiN+18] et avons entrainé des modèles RNN constitués d'une couche de 110 unités LSTM sur des séquences de longueur 50.

Reconnaissance d'Activité Humaine (HAR) : Cet ensemble de données contient des séries temporelles de poses 2D extraites de vidéos. Les vidéos sont un sous-ensemble de la base *Berkeley MHAD* (Multimodal Human Activity Database) [OCK+13] qui a été enregistrée sur 12 sujets avec des caméras fixes. Dans HAR, seul six actions sont considérées. Il y a un total de 1438 vidéos. Les poses 2D sont extraites de chaque image et chaque pose est constituée de 18 articulations du squelette, localisées dans un système de coordonnées 2D. Chaque action doit être identifiée à partir d'une séquence de 32 images (1.5 seconde à 22 Hz), soit 32 éléments de 36 coordonnées. Nous avons suivi la méthodologie expérimentale de [Eif] : l'ensemble de données a ainsi été divisé en 2 partitions indépendantes : 22625 séquences pour l'apprentissage, 5751 pour la validation. L'apprentissage a été effectué avec des batchs de 4096 séquences. Le réseau contient une couche entièrement connectée et 2 couches LSTM. Chacune de ces 3 couches est constituée de 40 cellules. La couche de sortie est réalisée par un classificateur softmax à 6 cellules.

Pour chaque ensemble de données, nous avons évalué la méthode pour une adaptation du modèle entre 2 séquences et une adaptation en cours de séquence. Le coût de calcul a été quantifié par le nombre moyen d'entrées traitées par un modèle.

4.1 Addition

4.1.1 Flexibilité entre séquences

FRNN permet d'obtenir des compromis intéressants sur cette tâche. La figure 1 présente un exemple typique de ces compromis. En partant d'un modèle entrainé avec $thr = 0.5$, augmenter thr pendant l'inférence réduit le nombre moyen d'entrées analysées dans la séquence mais augmente également l'erreur quadratique moyenne. Diminuer thr produit l'effet inverse.

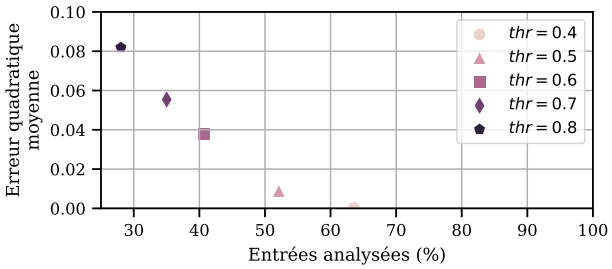

FIGURE 1 – Exemples de compromis précision / nombre moyen d'entrées utilisées pour FLSTM sur la tâche *Addition* pour un modèle entrainé avec $thr = 0.5$ et $\lambda = 10^{-4}$.

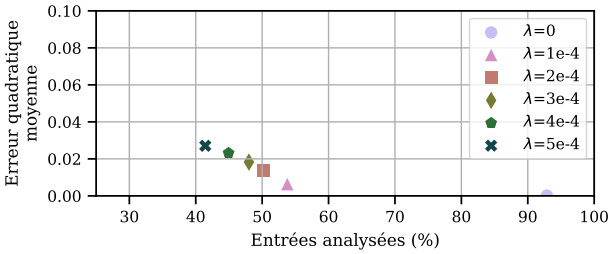

FIGURE 2 – Exemples de compromis précision / nombre moyen d'entrées utilisées pour skip-LSTM sur la tâche *Addition*.

Dans la figure 2, nous illustrons les compromis qui peuvent être obtenus par la méthode originale skip-RNN. Les compromis sont meilleurs que ceux obtenus avec FRNN. Par exemple, en traitant une moyenne de 40 % des entrées, skip-RNN atteint une erreur quadratique moyenne d'environ 0.27 tandis que FRNN atteint dans ces conditions 0.4. Cette différence est logique car skip-RNN n'est pas flexible et peut optimiser les paramètres pour cette complexité de calcul spécifique alors que FRNN utilise les mêmes paramètres pour tous les points de compromis. Cela étant dit, les deux valeurs d'erreur diffèrent seulement d'un facteur 2.

4.1.2 Flexibilité intra-séquence

Les résultats sont également bons lors de la modification du coût de calcul du modèle pendant l'inférence. La figure 3 présente le nombre moyen d'entrées analysées et la précision pour un modèle typique lors de la

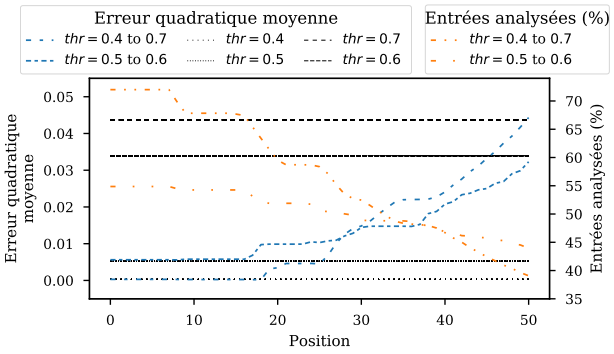

FIGURE 3 – Compromis obtenus par FLSTM en modifiant *thr* pendant l'analyse de la séquence sur la tâche *Addition*.

modification de *thr* respectivement de 0.5 à 0.6 ou de 0.4 à 0.7. Ces changements entrainent une réduction du nombre d'opérations effectuées pour analyser la séquence. Cela correspond à un cas d'utilisation où les ressources de calcul doivent être libérées pour un autre processus. Ces métriques sont rapportées en fonction de l'indice de l'entrée où le compromis est modifié. Ainsi, les valeurs à $x = 0$ et $x = 50$ correspondent à une valeur *thr* constante et respectivement égale à 0.4 ou 0.5 ($x = 0$) et 0.7 ou 0.6 ($x = 50$).

La précision et le coût de calcul en modifiant *thr* restent entre les limites des modèles sans changement de *thr*. Cela suggère que nos méthodes peuvent être utilisées pour adapter dynamiquement le coût de calcul d'un modèle. En effet, la pénalité sur la précision résultant d'un mélange de deux valeurs *thr* n'est pas déraisonnable, dans le sens où elle est inférieure à la précision obtenue en utilisant sur toute la séquence la valeur élevée de *thr*, et donc moins de calculs pour toute la séquence. En d'autres termes, le modèle n'est pas pénalisé pour avoir utilisé plus de ressources lorsque celles-ci ne sont pas nécessaires.

4.2 Reconnaissance d'activité

4.2.1 Flexibilité entre séquences

Cet ensemble d'expériences montre que les observations faites jusqu'à présent sont également valables sur un problème plus réaliste. En premier lieu, on peut voir que FRNN fonctionne bien sur ce problème. La Figure 4 affiche le compromis précision / nombre d'entrées analysées d'un modèle FLSTM sur HAR. En utilisant un modèle appris avec $thr = 0.5$, incrémenter *thr* entraîne une lente diminution en précision et des gains

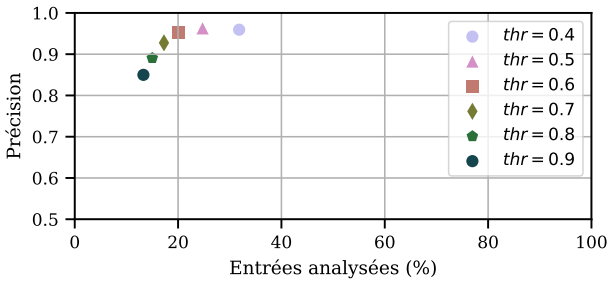

FIGURE 4 – Exemple d'un compromis précision / nombre d'entrées analysées pour FLSTM sur HAR. Le modèle a été entrainé avec $\lambda = 0.01$. Diminuer *thr* diminue la précision et le coût de calcul, comme sur l'addition.

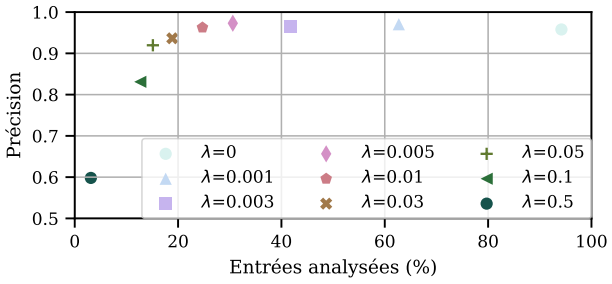

FIGURE 5 – Exemples de compromis précision / nombre d'entrées utilisées pour skip-RNN sur HAR.

importants en coût de calcul.

La Figure 5 montre des résultats obtenus par skip-RNN sur HAR. En la comparant à la figure précédente, on peut voir que FRNN obtient ici des compromis dont les valeurs sont similaires à ceux de skip-RNN.

4.2.2 Flexibilité intra-séquence

Lors d'une modification de *thr* pendant l'analyse d'une séquence, les résultats sont également similaires à ceux de la tâche d'addition. La Figure 6 fournit deux exemples de ce comportement en terme de compromis précision / coût de calcul. Ces mesures sont à nouveau affichées en fonction de l'index de l'entrée où *thr* a été modifié. Dans cet exemple, *thr* est élevé (respectivement 0.8 et 0.9) pour les premiers modèles et plus faible (0.5) dans le second modèle. Par conséquent, la précision et le coût de calcul sont élevés pour les faibles valeurs de x, car le modèle avec $thr = 0.5$ analyse presque l'entièreté de la séquence pour ces valeurs. La précision obtenue en changeant *thr* est une fois de plus comprise entre la précision des modèles indi-

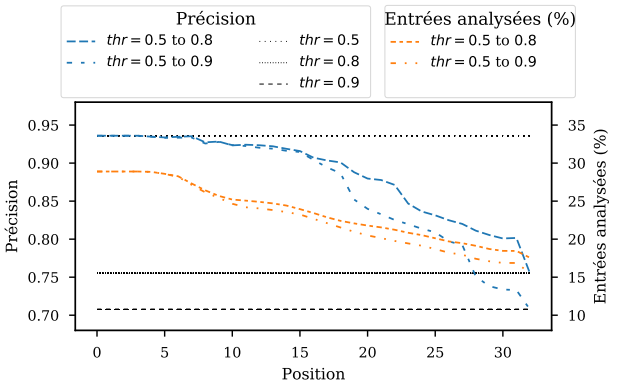

FIGURE 6 – Précision moyenne et nombre d'entrées utilisées sur HAR avec FLSTM en utilisant $thr = 0.8$ ou 0.9 jusqu'à x puis $thr = 0.5$. Le modèle a été entrainé avec $\lambda = 0.01$. Les lignes horizontales indiquent la précision des modèles sans changer thr.

viduels. De plus, elle diminue progressivement à mesure que les modèles avec thr plus élevés traitent un plus grand nombre d'entrées (x plus élevés). Enfin, le nombre moyen d'entrées traitées diminue plus vite que la précision.

5 Conclusion

Nous proposons de modifier dynamiquement le compromis entre précision et coût de calcul d'un réseaux de neurones récurrents en fonction des ressources de calcul disponibles. Notre méthode ne modifie qu'un paramètre et ne nécessite donc pas de recharger un nouveau modèle en mémoire. Elle est bien adaptée à un environnement avec une mémoire limitée.

Notre méthode permet d'échanger de la précision contre une réduction du coût de calcul au milieu de l'analyse d'une séquence. Nos expériences montrent que la précision résultante est généralement meilleure que la précision obtenue en utilisant sur l'ensemble de la séquence le modèle le plus efficace en terme de ressources. Par conséquent, notre approche peut également être utilisée pour tirer parti de ressources temporairement disponibles.

Références

[CJiN+18] Víctor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN : Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[CVMG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[Eif] Stuart Eiffert. Rnn for human activity recognition - 2d pose input.

[HHP16] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1533–1540, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[NPL16] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm : Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.

[OCK+13] Ferda Ofli, Rizwan Chaudhry, Gregorij Kurillo, René Vidal, and Ruzena Bajcsy. Berkeley mhad : A comprehensive multimodal human action database. In *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, pages 53–60. IEEE, 2013.

[YWH+16] Haonan Yu, Jiang Wang, Zhiheng Huang, Yi Yang, and Wei Xu. Video paragraph captioning using hierarchical recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4584–4593, 2016.

[ZSLC17] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge : Keyword spotting on microcontrollers. *CoRR*, abs/1711.07128, 2017.