Exploration de l'Oubli Catastrophique en Apprentissage Continu

Etudiants:
El Halawani Yara - 20235309
CHATTAH Talout - 20235542

Encadré par
M. Hamidi

16 Décembre 2024
le code source du projet est disponible sur ce répertoire GitHub.

Table des matières

1 Introduction

L'apprentissage continu s'agit d'apprendre à partir d'une séquence de tâches, l'une après l'autre, sans avoir accès à toutes les tâches simultanément. Cependant, l'un de ses principaux défis, est, l'oubli catastrophique.

L'oubli catastrophique est la possibilité d'un modèle à oublier des tâches précédemment apprises s'il n'est pas correctement formé sur une nouvelle tâche.

L'objectif de l'apprentissage continu est de développer des systèmes capables de maintenir de bonnes performances sur l'ensemble des tâches, passées comme nouvelles, sans nécessiter un réentraînement complet ni subir l'effet de l'oubli catastrophique.

Dans le cadre de ce projet, notre objectif principal est d'étudier et d'analyser les différents facteurs qui contribuent au phénomène de l'oubli catastrophique dans l'apprentissage continu. Nous cherchons à comprendre comment des caractéristiques des ensembles de données influencent ce phénomène.

2 Apprentissage continu

L'apprentissage continu, également appelé apprentissage tout au long de la vie, est un paradigme d'apprentissage automatique conçu pour permettre aux modèles d'évoluer et de s'adapter progressivement à un flux constant de données ou de tâches.

L'apprentissage continu est essentiel pour répondre aux défis des environnements réels, où les données et les tâches évoluent constamment. Il permet aux systèmes d'intégrer de nouvelles informations ou compétences au fil du temps, tout en conservant les connaissances précédemment acquises. En imitant le mode d'apprentissage humain, cette approche favorise l'accumulation et le raffinement progressif des savoirs. De plus, elle réduit les besoins en ressources en évitant le réentraînement complet des modèles chaque fois que de nouvelles données ou tâches apparaissent. Enfin, l'apprentissage continu garantit des performances optimales dans des environnements non stationnaires, où les distributions des données ou les exigences des tâches changent fréquemment.

3 Algorithmes de l'apprentissage continu

Les approches les plus simples et courantes de l'apprentissage continu reposent sur la régularisation des poids pour prévenir l'oubli catastrophique. Cette méthode consiste à ajouter de nouveaux neurones au réseau pour représenter les nouvelles tâches, tout en maintenant les nouveaux poids proches des poids précédents afin de préserver les connaissances acquises.

3.1 Variational continual learning (VCL)

L'apprentissage continu variationnel (VCL) est une approche avancée de l'apprentissage continu qui utilise l'inférence bayésienne pour surmonter l'oubli catastrophique. Contrairement aux méthodes traditionnelles qui mettent à jour les paramètres du modèle comme des estimations ponctuelles, le VCL considère les paramètres du modèle comme des distributions, capturant l'incertitude et permettant un apprentissage plus flexible. En maintenant une distribution a posteriori sur les paramètres et en utilisant l'inférence variationnelle pour l'approcher, le VCL permet au modèle de conserver les connaissances des tâches précédentes tout en s'adaptant aux nouvelles.

3.2 Coreset Variational continual learning

L'algorithme Coreset VCL améliore le VCL traditionnel, en introduisant un coreset, un petit sous-ensemble de points de données conservés à partir de tâches précédentes. Ces points de coreset sont utilisés lors de l'entraînement sur de nouvelles tâches pour revisiter explicitement les anciennes tâches et renforcer les connaissances acquises.

4 Concept et Fonctionnement de Task2Vec

Task2Vec est un framework développé pour l'intégration des tâches de classification visuelle sous forme de vecteurs dans un espace vectoriel réel. Il est un outil puissant pour définir et analyser les relations entre différentes tâches à partir des données. Il améliore la capacité à comprendre comment les tâches sont interconnectées et comment les connaissances peuvent être transférées entre elles.

L'exécution de Task2Vec sur un jeu de données produit un objet d'intégration contenant les représentations numériques des tâches générées par la méthode Task2Vec.

Ces plongements sont des vecteurs à dimension fixe, codant mathématiquement les propriétés essentielles de l'ensemble de données.

Cet embedding reflète la structure statistique de l'ensemble de données ainsi que sa similarité avec d'autres tâches, ouvrant la voie à diverses applications telles que:

- Analyse de similarité des tâches: comparaison d'ensembles de données pour identifier les tâches liées.
- Méta-apprentissage: information sur la sélection de modèles ou les stratégies d'apprentissage par transfert.
- Sélection de modèles: sélection d'un extracteur de fonctionnalités ou d'un modèle pré-entraîné le mieux adapté à l'ensemble de données.

Lorsque Task2Vec est exécuté sur un ensemble de données, il utilise un réseau neuronal pré-entraîné pour calculer les intégrations de tâches. Cela implique:

- Extraction de caractéristiques: le réseau traite l'ensemble de données en extrayant des représentations de caractéristiques intermédiaires.
- Matrice d'informations de Fisher (FIM): Task2Vec estime l'importance de chaque paramètre du réseau pour l'ensemble de données à l'aide de la FIM, représentant la sensibilité des sorties du modèle aux changements dans l'ensemble de données.
- Génération d'intégrations: les intégrations résultantes sont des vecteurs de grande dimension résumant les caractéristiques de l'ensemble de données.

5 Approche de l'article

Pour mieux comprendre l'oubli catastrophique, les auteurs de l'article [1] ont proposé d'utiliser le framework Task2Vec pour analyser la relation entre un ensemble de tâches. Ils ont ensuite attribué les mêmes tâches à des algorithmes d'apprentissage continu. En se basant sur les performances des algorithmes sur ces tâches et sur les résultats fournis par Task2Vec, qui permet de représenter la similarité entre les tâches, ils ont réussi à calculer deux mesures : la complexité totale (total complexity) et l'hétérogénéité séquentielle (sequential heterogeneity). Enfin, ils ont étudié la relation entre ces deux mesures.

Le modèle effectue bien la classification des labels dans chaque tâche de classification binaire. Cependant, l'objectif de l'étude n'est pas seulement ça, mais plutôt d'analyser comment la performance du modèle est affecté par les propriétés des séquences de tâches déjà mentionnés.

5.1 Génération des tâches

À partir d'un jeu de données initial, nous générons un nouveau jeu de données mieux adapté de la manière suivante :

1. Nous traitons toutes les paires d'étiquettes distinctes comme des tâches de classification binaire. Par exemple, pour MNIST, qui comprend 10 classes, cela donne 45 combinaisons de paires possibles et uniques : (0,1), (0,2), ..., (8,9).

2. À partir de ces tâches unitaires, nous générons ensuite 120 séquences de cinq paires de tâches, choisies aléatoirement. L'exemple 1 : [(0,1), (3,4), (8,9), (2,7), (5,6)], est une exemple possible d'une séquence des tâches.

5.2 Similarité des tâches à l'aide de Task2Vec

Dans le but de quantifier de manière utile les propriétés des tâches (complexité et hétérogénéité), révélant des relations importantes entre ces propriétés et les performances des modèles en apprentissage continu, Task2Vec a été appliqué sur jeux de données.

La proprieté "Complexité des tâches" est mesurée par la distance entre le vecteur d'une tâche et celui d'une tâche triviale (située à l'origine de l'espace vectoriel). De même pour la proprieté, "Hétérogénéité séquentielle", elle est calculée comme la dissimilarité entre les vecteurs des tâches consécutives dans une séquence.

Une exemple de résultats de similarité entre des différentes tâches de MNIST, est representé par la figure 1.

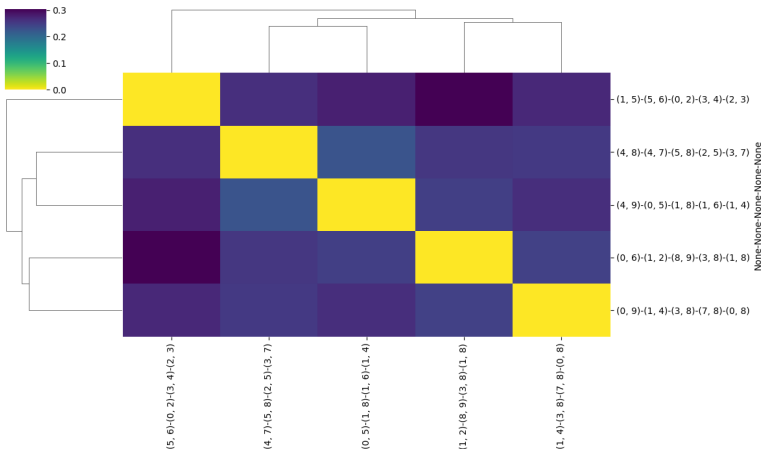

Figure 1: Embeddings fourni par Task2Vec, sur 5 paire d'une tâche de MNIST.

La matrice de similarité mesure la distance ou dissemblance entre les tâches en fonction de leurs embeddings dans l'espace Task2Vec. Les valeurs faibles sont reprensentés par la couleur violet foncé, et indiquent que les tâches sont dissemblables. Tandis que, les valeurs élevées qui sont reprensentés par la couleur jaune, indiquent que les tâches sont similaires, avec peu de variations dans leurs distributions.

5.3 Traitement des tâches générés

1. Apprentissage séquentiel :
   Les tâches seront appris par le modèle une par une, selon l'ordre de la séquence des tâches, sans retourner vers les données des tâches précédentes. Reconsidérons la séquence des tâches de l'exemple 1, dont on a déjà parlé, le modèle apprend la tâche (0,1), et passe à la tâche suivante (3,4) et ainsi de suite.

2. Entrées du modèle :
   Pour chaque tâche, les entrées du modèle seront les images correspondant aux deux classes du tâche. Par example, pour la tâche (0,1), seules les images étiquetées "0" et "1" sont fournies en entrée.

3. Sorties du modèle :
   Le modèle générera une prédiction pour chacune des deux classes de chaque tâche. Par exemple, pour la tâche (0,1), la sortie sera [p(0), p(1)]. À la fin,

les sorties du modèle incluront les prédictions pour toutes les classes de la tâche en cours ainsi que celles des tâches précédemment apprises.

4. Évaluation :
   Une fois toutes les tâches de la séquence sont appris, le "Catastrophic Forgetting" sera évalué, en mesurant la performance du modèle sur les tâches précédentes, après avoir appris les nouvelles tâches, en utilisant une matrice d'accuracy, de taille n x n, où n est le nombre des tâches concernées. Les valeurs de chaque ligne de la matrice représente l'accuracy des tâches déjà entraînées avant la tâche actuelle.

6 Notre Approche

Dans le cadre de notre projet basé sur l'article étudié, nous avons proposé une nouvelle approche visant à explorer la similarité entre différentes rotations de MNIST. L'objectif principal de notre approche est d'évaluer l'effet des rotations sur les relations entre les tâches, tout en analysant comment elles influencent le phénomène d'oubli catastrophique.

6.1 Générations des tâches

Pour préparer les ensembles de tâches, on a proposé d'étudier deux propriétés et leur effet sur l'oubli catastrophique dans l'apprentissage continu, la première propriété est la différence entre l'angle de rotation des tâches, et la deuxième propriété est l'ordonnancement des angles de rotation des tâches.

Pour réaliser les différentes expérimentations, dont nous parlerons plus tard, nous avons généré 7 ensembles de données autre que MNIST basés sur elle, en appliquant des rotations aux images. Chaque ensemble de données correspond à une tâche distincte, définie comme suit :

- MNIST-45 avec une rotation de 45°,
- MNIST-90 avec une rotation de 90°,
- MNIST-135 avec une rotation de 135°,
- MNIST-180 avec une rotation de 180°,
- MNIST-225 avec une rotation de 225°,

- MNIST-270 avec une rotation de 270°.
- MNIST-315 avec une rotation de 315°,

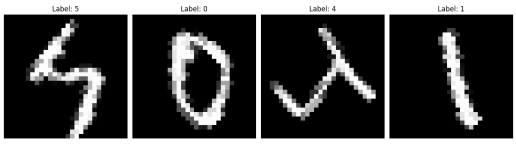

Figure 2: MNIST avec rotation 45°.

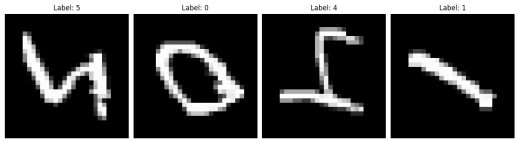

Figure 3: MNIST avec rotation 270°.

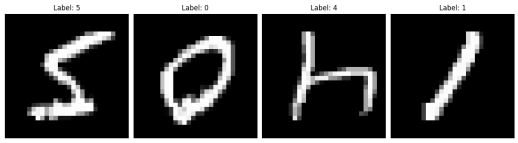

Figure 4: MNIST avec rotation 180°.

1. Pour évaluer la première propriété, nous avons mené deux expérimentations distinctes basées sur une rotation progressive des images:
    - La première impliquait 4 tâches, avec une différence de 90° entre chaque tâche.
    - La seconde utilisait 8 tâches, avec une différence de 45° entre chaque tâche.
2. Pour évaluer la deuxième propriété, nous avons mené une seule expérimentation utilisant 8 tâches, avec une différence de 45° entre elles. Dans ce cas,

les angles de rotation des tâches ne suivent pas un ordre croissant. Par exemple, les tâches peuvent être MNIST-90, MNIST-135, MNIST-45, et ainsi de suite.

Ces transformations permettent de définir des tâches ayant des propriétés similaires mais différenciées, facilitant l'analyse de leur impact sur les modèles utilisés, notamment dans des scénarios d'apprentissage continu.

6.2 Similarité des tâches à l'aide de Task2Vec

Après avoir généré les tâches, nous appliquons Task2Vec sur les ensembles générés afin d'évaluer leur similarité. Les résultats obtenus sont présentés dans le graphique de similarité ci-dessous.

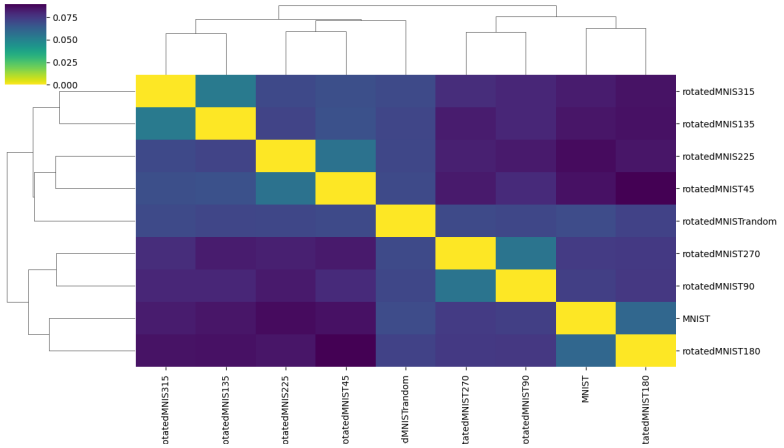

Figure 5: Embeddings fourni par Task2Vec, sur les 4 rotations du MNIST.

6.3 Traitement des tâches générés

Le modèle, a été entraîné séquentiellement, avec l'ensemble des tâches générés, pour modéliser un scénario d'apprentissage continu. Puis on a analysé les performances du modèle sur chaque tâche après l'entraînement, en oberservant l'effet de l'oubli catastrophique.

1. Entrées du modèle :
   Pour chaque tâche, les entrées du modèle seront les images correspondant aux ensemble de donnée concerné.

2. Sorties du modèle :
   Le modèle générera une prédiction pour chacune des deux classes de chaque tâche. Par exemple, pour la tâche (0,1), la sortie sera [p(0), p(1)]. À la fin, les sorties du modèle incluront les prédictions pour toutes les classes de la tâche en cours ainsi que celles des tâches précédemment apprises.

3. Évaluation :
   Une fois toutes les tâches sont appris, le "Catastrophic Forgetting" sera évalué, en mesurant la performance du modèle sur les tâches précédentes, après avoir appris les nouvelles tâches, en utilisant une matrice d'accuracy, de taille n x n, où n est le nombre des tâches concernées. Les valeurs de chaque ligne de la matrice représente l'accuracy des tâches déjà entraînées avant la tâche actuelle.

6.4 Résultats

Comme le montre la figure 6, le VCL a oublié la première tâche lorsqu'il est arrivé à la dernière. En revanche, le coreset VCL maintient de bonnes performances grâce au stockage d'un petit ensemble d'échantillons représentatifs des tâches précédentes.

Figure 6: Matrices de précision pour l'expérimentation 1.

En revanche, nous avons constaté que le VCL a bien fonctionné dans l'expérimentation 2, où le degré de rotation entre les tâches a été réduit et le nombre de tâches augmenté. Il a réussi à produire de bons résultats sur la première tâche après avoir traité la dernière, en raison de la forte similarité entre ces deux tâches, la différence d'angle entre elles étant de seulement 45°. De même, le coreset VCL a obtenu de meilleurs résultats lorsque l'angle de rotation a été réduit.

Figure 7: Matrices de précision pour l'expérimentation 2.

Enfin, comme le montre la figure 8, la 3ème expérimentation a obtenu les meilleurs résultats, avec peu de phénomènes d'oubli catastrophique pour les deux algorithmes, VCL et coreset VCL.

Figure 8: Matrices de précision pour l'expérimentation 3.

L'entraînement du modèle avec ces expérimentations, nous a permis de conclure que l'oubli catastrophique est influencé par la différence d'angle de rotation entre les tâches. En d'autres termes, plus la différence entre les angles de rotation est réduite, moins l'oubli catastrophique est remarqué. Par exemple, lorsque les tâches diffèrent de 45°, l'oubli catastrophique est peu significatif, tandis qu'avec une différence de 90°, son effet devient nettement perceptible dans les performances du modèle.

Par ailleurs, l'oubli catastrophique est également influencé par l'ordre des angles de rotation. Lorsque les angles de rotation des tâches ne suivent pas un ordre précis, l'oubli catastrophique tend à diminuer dans ce cas.

En conclusion, les résultats obtenues, montrent que le coreset VCL surpasse le VCL classique en maintenant une meilleure précision moyenne grâce à la con-

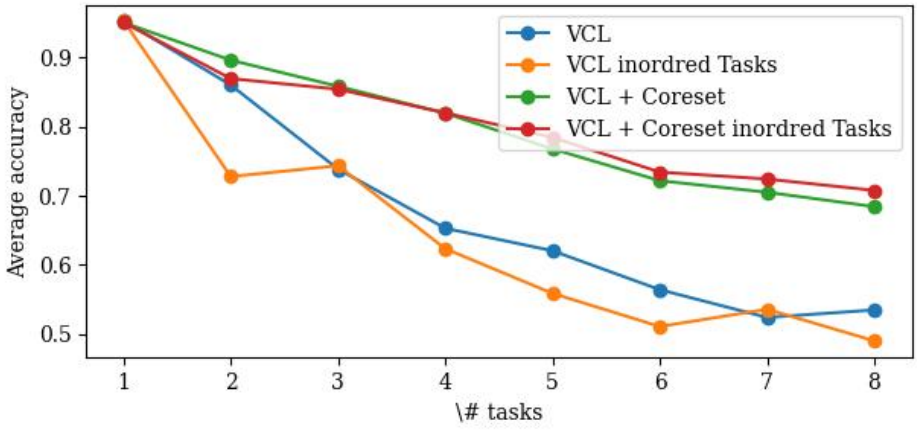

Figure 9: Précision moyenne de VCL et Coreset VCL sur expériences 2 & 3.

servation d'un sous-ensemble représentatif des données, ce qui réduit significativement l'oubli catastrophique, comme le montre la figure 9.

De plus, la performance des deux modèles est influencée par la différence angulaire entre les rotations des tâches et leur ordre de présentation. Une faible différence angulaire, comme 45°, et un ordre non linéaire des tâches limitent l'effet de l'oubli. Ainsi, coreset VCL se révèle plus robuste pour gérer des séquences de tâches complexes, tandis que la similarité entre les tâches reste un facteur clé pour atténuer l'oubli, même pour VCL.

7 Problèmes rencontrés et solutions apportées

7.1 Problèmes matériels

Pour les problèmes matériels, nous avons rencontré des difficultés liées à la RAM, car les modèles présentés dans l'article consomment beaucoup de mémoire. Nous n'avons pas réussi à exécuter le modèle avec un échantillon de 1000 images sur une RAM de 32 Go. Nous avons donc réduit le nombre d'images à 500 et on a gardé le nombre de tâches à 120, puis avons laissé le code s'exécuter pendant 9 heures, mais l'exécution s'est arrêtée sans raison apparente. Finalement, en réduisant le nombre de tâches à 10 et le nombre d'images à 500, nous avons réussi à obtenir la matrice des précisions à la fin. Cependant, nous avons rencontré un nouveau problème lié à la taille trop réduite du jeu de données, ce qui a entraîné

un problème de sous-apprentissage

7.2 Erreurs et bugs software

- Pour l'algorithme Synaptic Intelligence, nous avons trouvé d'anciennes implémentations datant de 2017-2018, utilisant Python 3.5.2 avec Keras 2.2.2. Nous souhaitions exécuter l'algorithme dans cet environnement. Cependant, après avoir installé Python 3.5.2 et essayé d'installer Keras 2.2.2 avec la commande pip install keras==2.2.2, pip n'a pas trouvé cette version de Keras.Nous avons alors téléchargé Keras 2.2.2 sous forme de fichier .tar.gz et tenté de l'installer avec la commande pip install keras-2.2.2.tar.gz, mais l'installation a également échoué. Ensuite, nous avons mis à jour pip vers une version plus récente, mais lors de l'installation de Keras 2.2.2, un message indiquait que les versions de pip et de Keras étaient incompatibles.

- Dans le framework Task2Vec, dans le fichier nommé models.py, nous avons rencontré des problèmes de compatibilité de version avec torchvision. Pour résoudre ce problème, nous avons modifié les deux lignes de code suivant:

```
state_dict =
↪  model_zoo.load_url(resnet.model_urls['resnet18'])
state_dict = {k: v for k, v in state_dict.items() if 'fc'
↪  not in k}
```

avec:

```
state_dict = models.resnet34(pretrained=True)
state_dict = {k: v for k, v in
↪  state_dict.state_dict().items() if 'fc' not in k}
```

- Pour l'algorithme de l'apprentissage continu variationnel, nous l'avons adapté à une version plus récente de TensorFlow et avons également corrigé de nombreuses erreurs dans les fichiers cla_models_multihead.py. Nous avons utilisé ces deux lignes pour désactiver le comportement de TensorFlow 2 et rendre le comportement de TensorFlow 1 par défaut.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

et on a changer la ligne 71 avec

```
perm_inds = list(range(x_train.shape[0]))
```

8 Conclusion

En conclusion, l'approche présentée dans l'article ainsi que nos expérimentations mettent en évidence les défis posés par l'oubli catastrophique dans les modèles d'apprentissage continu, et offrent des solutions pour mieux comprendre et atténuer ce phénomène. L'article démontre que des outils comme Task2Vec permettent de mesurer la similarité entre les tâches, ce qui peut guider l'entraînement de modèles afin de limiter l'oubli. Nos propres expérimentations, confirment que la différence entre les tâches, notamment en termes de rotation, ainsi que leur ordre de présentation, influencent fortement l'oubli catastrophique.

De plus, nos résultats montrent que le modèle coreset VCL, grâce à son mécanisme de stockage d'échantillons représentatifs, est plus performant pour gérer des séquences de tâches complexes en minimisant l'oubli. Par ailleurs, nous avons constaté que réduire la différence angulaire entre les tâches ou désordonner leur séquence réduit également l'oubli, même pour un modèle comme VCL. Ces observations soulignent l'importance de la similarité et de l'organisation des tâches dans l'apprentissage continu et offrent des pistes prometteuses pour améliorer les performances des modèles face à des environnements d'apprentissage dynamique.

Cependant, les résultats que nous avons obtenus sont basés sur nos expérimentations spécifiques et pourraient ne pas être généralisés à tous les ensembles de données. En effet, un ensemble de données particulier, comme MNIST et ses variantes, peut présenter des propriétés uniques qui influencent les performances des modèles. L'oubli catastrophique pourrait également être affecté par d'autres facteurs qu'il serait pertinent d'explorer. Par exemple, le degré de bruit dans les données, pourrait jouer un rôle important. Tels facteurs mériteraient une analyse approfondie pour mieux comprendre leur impact sur l'apprentissage continu et pour concevoir des modèles plus robustes et généralisables.

References

[1] Cuong V Nguyen, Alessandro Achille, Michael Lam, Tal Hassner, Vijay Mahadevan, and Stefano Soatto. Toward understanding catastrophic forgetting

in continual learning. *arXiv preprint arXiv:1908.01091*, 2019.