INF8225 - Rapport de Projet

Autoencoders Convolutionnels Empilés (Stacked Convolutional Autoencoders)

Maxime Schmitt
1719088

20 avril 2016

1 Introduction

L'idée initiale derrière ce projet était de trouver un moyen de permettre un apprentissage sur des données dont une partie seulement serait étiquettée. Cependant, les solutions apportées pour ce type de problèmes se révèlent être complexes et reposent sur des méthodes tout d'abord relativement récentes dans la littérature mais qui proposent surtout des modes de fonctionnement pour l'apprentissage nouveaux et inhabituels étant donnée mon expérience récente dans ce domaine. C'est pourquoi, pour ce projet, je me suis principalement concentré sur la compréhension des concepts impliqués et sur une première utilisation simple d'un des modèles proposés, afin de me familiariser avec celui-ci dans le but de poursuivre ce travail dans le cadre de ma recherche.

C'est pourquoi je me suis concentré sur la compréhension et l'utilisation des autoencoders convolutionnels et en particuliers à plusieurs couches. L'article auquel je me suis le plus référé[5] les présente de façon progressive au travers des autoencoders conventionnels, des réseaux de neurones convolutionnels et enfin des autoencoders convolutionnels. J'ai également consulté[2] qui présente un modèle à base de réseau de neurones et autoencoder dont l'entraînement est régi par une fonction de perte hybride entre les deux entités. L'objectif initial était de revisiter cette idée avec cette fois-ci des réseaux de neurones convolutionnels ainsi que des autoencoders convolutionnels. Cependant, pour des raisons que j'évoque dans la partie 4 de ce rapport, je n'ai pas pu concrétiser cet objectif.

Je vais donc dans ce rapport aborder les concepts théoriques autour desquels j'ai travaillé au cours de ce projet, y compris les points que je n'ai pas pu implémenter. Puis je présenterai quelques résultats que j'ai obtenus au cours de mes diverses expériences. Et enfin je reviendrai sur la façon dont j'ai abordé ce projet et ai effectué mes recherches ainsi que sur les erreurs que j'ai commises dans la poursuite de ce travail.

2 Concepts théoriques

Ce projet avait pour objectif initial d'apporter une innovation en se basant sur deux idées déjà développées dans la littérature : les autoencoder convolutionnels simples et empilés ainsi que l'apprentissage semi-supervisé d'autoencoders profonds. Cette partie présente les principaux apprentissages que j'ai fait sur ces sujets au cours de ce projet et qui m'ont été utiles pour définir mon modèle que je présenterai dans la partie suivante.

Parmi les nombreux articles que j'ai eu l'occasion de parcourir, j'ai concentré mon attention sur deux articles en particulier. Lee premier[6] introduit de façon progressive les autoencoders convolutionnels empilés avant de présenter un exemple de leur utilisation, tandis que le second[2] propose une méthode pour entraîner un réseau de neurones multicouches de façon hybride en utilisant à la fois la prédiction du réseau mais également le résultat de la reconstruction de son entrée en formant un autoencoder profond. Je vais développer dans cette partie les enseignements que j'ai tiré de ces lectures pour avancer dans ce projet.

Le premier article présente donc tout d'abord les autoencoders tels qu'on les a vu dans le cours puis les denoising autoencoders introduits par [7] qui ont été évoqués dans le cours. Ces derniers permettent de forcer les autoencoders à apprendre les caractéristiques importantes de l'image en introduisant une variation dans celle-ci en entrée uniquement sous la forme

de bruit. Puis, après avoir présenté brièvement les réseaux de neurones convolutionnels que l'on a vus en cours, il présente les autoencoders convolutionnels que l'on peut imaginer comme l'équivalent de ce que sont les autoencoders aux réseaux de neurones pour les réseaux de neurones convolutionnels. Parmi les éléments mathématiques, on en retient deux qui se révèlent particulièrement intéressants :

— La matrice des paramètres des différentes couches situées après la couche d'encodage (la partie décodage du réseau) est simplement la transposée de la matrice des paramètres de la couche de codage correspondante.

— L'apprentissage se fait par la méthode de la descente du gradient avec comme fonction de perte à minimiser l'erreur quadratique moyenne :

$$E(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (x_i - y_i)^2 \tag{1}$$

Il introduit enfin les stacked convolutional autoencoders qui consistent à l'empilement de convolutional autoencoders de la même façon que l'on empile les autoencoders conventionnels. L'article termine par présenter un exemple d'application pour ceux-ci : de la même façon qu'on peut utiliser des autoencoders profonds que l'on aurait pré-entraîné de façon non supervisée sur les données avec un apprentissage couche par couche glouton pour initialiser des réseaux de neurones profonds, il est proposé ici d'utiliser les stacked convolutional autoencoders pré-entraînés de la même façon pour initialiser des réseaux de neurones convolutionnels profonds. L'idée est de produire un autoencoder convolutionnel profond donc l'architecture jusqu'à la couche d'encodage centrale est la même que pour le réseau de neurones convolutionnels profond voulu. On entraîne ensuite chaque couche depuis l'extérieur du réseau vers l'intérieur de façon que chaque couche apprenne à repconstruire la sortie de la couche précédente. Une fois cet apprentissage terminé, on récupère la partie du réseau jusqu'à la couche d'encodage et on l'utilise comme réseau de neurones convolutionnels profond. Leurs expériences montrent l'efficacité de cette méthode qui permet donc bien d'obtenir un état initial intéressant pour l'apprentissage, c'est-à-dire qui ne convergera pas vers un minimum local de la fonction de perte qui est un des principaux problèmes des réseaux profonds lorsqu'on les initialisent aléatoirement en particulier.

Le second article[2] quant à lui propose un modèle d'apprentissage qui profite lui aussi d'une composante non supervisée mais non plus dans le cas d'un pré-apprentissage mais bien pendant l'apprentissage lui-même. L'article s'intéresse sur une application spécifique mais la partie qui m'a intéressée dans cet article est bien celle portant sur l'introduction de ce qu'il appelle le perceptron multicouches hybride. L'idée est de compléter un réseau de neurones utilisé pour une tâche de classification afin d'en faire un autoencoder profond : on rajoute donc les couches après la couche de classification de façon à construire un autoencoder profond dont la couche centrale d'encodage serait la couche de sortie du réseau initial. Une fois cela fait, on va entraîner le réseau en utilisant une fonction de perte hybride combinant la fonction de perte du réseau initial et la fonction de perte de l'autoencoder profond créé précédemment avec l'introduction d'un nouvel hyper-paramètre λ qui varie de 0 à 1. Son expression est de la forme suivante :

$$E(\theta) = (1 - \lambda) \cdot E_{sup}(\theta) + \lambda \cdot E_{unsup}(\theta) \tag{2}$$

où $E_{sup}(\theta)$ et $E_{unsup}(\theta)$ sont respectivement les fonctions de perte du réseau de neurones initial et de l'autoencoder. On voit donc qu'en particulier choisir $\lambda = 0$ nous donnerait le réseau de neurones initial tandis que choisir $\lambda = 1$ nous donnerait l'autoencoder. L'expression de la fonction $E_{sup(\theta)}$ dépend de l'application mais pour ce qui est de la fonction $E_{unsup}(\theta)$ on prend en général l'erreur quadratique moyenne comme précisé précédemment et donc l'équation a été donnée dans l'équation 1. Un autre article[1] aborde cette question et donne une précision supplémentaire sur la façon dont peut être construit l'autoencoder profond. Dans leur proposition, une couche supplémentaire est ajoutée au niveau de la couche de sortie du réseau initial de telle sorte qu'on a deux couches parallèles au centre de l'autoencoder, la nouvelle remplissant permettant de garder assez de dimensionnalité pour représenter suffisamment d'informations pour permettre une reconstruction en sortie de l'autoencoder. En effet, en utilisant uniquement la couche de sortie du réseau initial on risque de perdre trop d'information sur le contenu en entrée pour pouvoir effectuer la régénération de l'information puisque cette couche de sortie est de taille fixe dépendante du nombre de classes ce qui peut être insuffisant pour représenter suffisamment d'informations.

3 Expériences et résultats

L'implémentation des convolutional autoencoders ayant permis ce travail peut être trouvée sur Github[6] même s'il n'est actuellement plus fonctionnel en raison d'évolutions de ses dépendances. J'ai donc dû en premier lieu, après avoir installé l'ensemble des dépendances requises, corriger de nombreuses erreurs dans le code liées à ces évolutions. J'avais ensuite un autoencoder convolutionnel avec lequel j'ai essayé de travailler comme je le décris dans les prochaines parties.

3.1 Description de l'architecture proposée

L'objectif était donc de définir un nouveau modèle mêlant à la fois apprentissage supervisé et non supervisé comme présenté à la partie précédente mais cette fois avec des réseaux de neuroens convolutionnels. L'idée est donc d'utiliser les autoencoders convolutionnels en place des autoencoders classique et de remplacer le réseau de neurones profond par un réseau de neurones convolutionnels profond dans le modèle présenté dans la première partie. En pratique, on construit un stacked convolutional autoencoder avec une couche centrale de la taille du classificateur (dans le cas de MNIST[4] on choisit une couche centrale de taille égale à 10) et on le modifie ensuite de façon à obtenir le réseau hybride voulu. La fonction de perte que l'on voudrait implémenter est donc de la forme donnée dans l'équation 2 où E_{sup} est le log-vraisemblance mesurant la perte sur la classification et E_{unsup} est l'erreur quadratique moyenne mesurant la perte sur la reconstruction des images.

Pour l'implémentation faite dans ce projet, j'ai réussi à obtenir un autoencoder convolutionnel à plusieurs couches de la forme voulue et à obtenir les deux sorties qui nous intéressent, mais je n'ai pas réussi, en raison de difficultés avec les outils Theano, Lasagne et nolearn, à implémenter la fonction de perte hybride. Les résultats présentés dans les parties suivantes sont donc des résultats obtenus avec ce réseau là.

L'architecture du réseau est présentée sur la figure 1. La couche *encode* centrale est celle qui servirait à la classification et qui sert le rôle d'encodage de l'information.

#	name	size
0	input0	1x28x28
1	conv2dcc1	16x26x26
2	conv2dcc2	16x24x24
3	maxpool2dcc3	16x12x12
4	conv2dcc4	32x10x10
5	maxpool2dcc5	32x5x5
6	reshape6	800
7	dense7	128
8	encode	10
9	dense9	128
10	dense10	800
11	reshape11	32x5x5
12	upscale2d12	32x10x10
13	conv2dcc13	16x12x12
14	upscale2d14	16x24x24
15	conv2d15	16x26x26
16	conv2d16	1x28x28
17	reshape17	784

FIGURE 1 – Architecture de l'autoencoder convolutionnel utilisé

3.2 Optimisation par parallélisation sur carte graphique

En premier lieu, j'ai pu constater l'effet de la parallélisation sur GPU des calculs dans le cas des réseaux de neurones convolutionnels. En effet, à la première utilisation l'apprentissage se faisait en utilisant uniquement le CPU et le temps moyen pour une epoch sur ma machine était d'environ 330 secondes. Après installation de CUDA cette valeur descendait déjà aux environ de 70 secondes par epoch, mais elle est encore améliorée par l'utilisation d'implémentation différentes pour la parallélisation. En effet, utiliser *cuDNN*, une implémentation proposée par Nvidia, ou *cuda_convnet*, l'implémentation proposée par Krizhevsky[3], permet de passer à un temps d'apprentissage par epoch de 50 secondes environ. Enfin, l'utilisation de *CNMeM*, une méthode d'optimisation de l'allocation de la mémoire sur le GPU proposée par Nvidia, permet d'abaisser encore cette valeur à 38 secondes.

3.3 Résultats observés sur l'autoencoder convolutionnel actuel

Dans cette partie, je propose quelques résultats que j'ai pu obtenir avec le réseau présenté dans le début de cette partie.

On retrouve en figure 2 les courbes de la fonction de perte au cours du temps pour l'ensemble d'apprentissage et l'ensemble de validation. L'apprentissage a été effectué après

100 epochs puis sauvegardé afin de pouvoir être réutilisé pour des expériences sans avoir besoin de ré-entraîner un réseau à chaque fois.

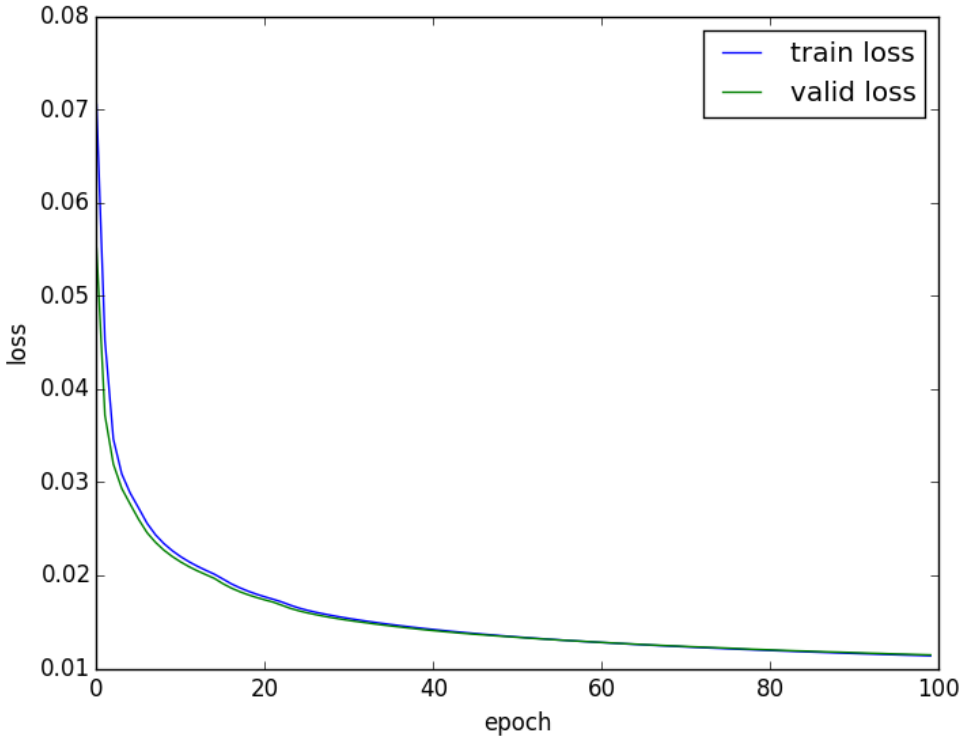


FIGURE 2 – Fonction de perte sur l'ensemble d'apprentissage et l'ensemble de validation au cours des epochs

Après cet apprentissage, notre réseau est capable de reconstruire les chiffres de la base MNIST avec certes un perte en résolution mais avec une précision suffisante pour reconnaître les chiffres introduits initialement comme on peut le voir sur la figure 3 qui présente la sortie du réseau pour un ensemble de chiffres tests en entrée.

Comme précisé précédemment, j'ai pu avoir accès aux valeurs de sorties de la couche *encode* en plus de celles de la dernière couche du réseau. Grâce à cela j'ai pu implémenter deux fonction qui permettent de réaliser respectivement l'encodage et le décodage des images. Une application ici serait la compression avec pertes d'images puisque l'encodage tient sur une dimension plus faible que l'image en entrée (10 valeurs contre 46 dans le cas présent). On notera cependant qu'avec une telle réduction de dimensionnalité l'information initiale de l'image ne peut pas être extraite par un humain directement, il faut la décoder à l'aide de la deuxième fonction pour pouvoir l'analyser. Le résultat visuel après décodage est le même que celui montré à la figure 3.

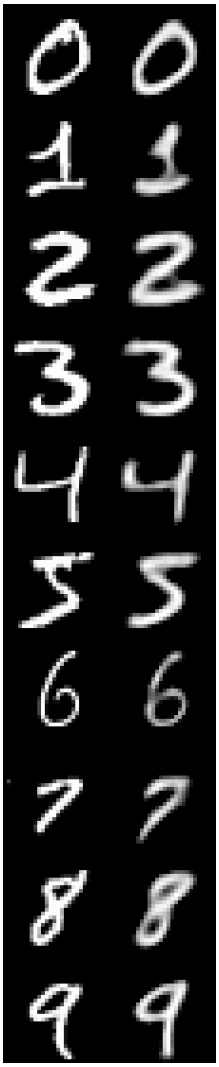
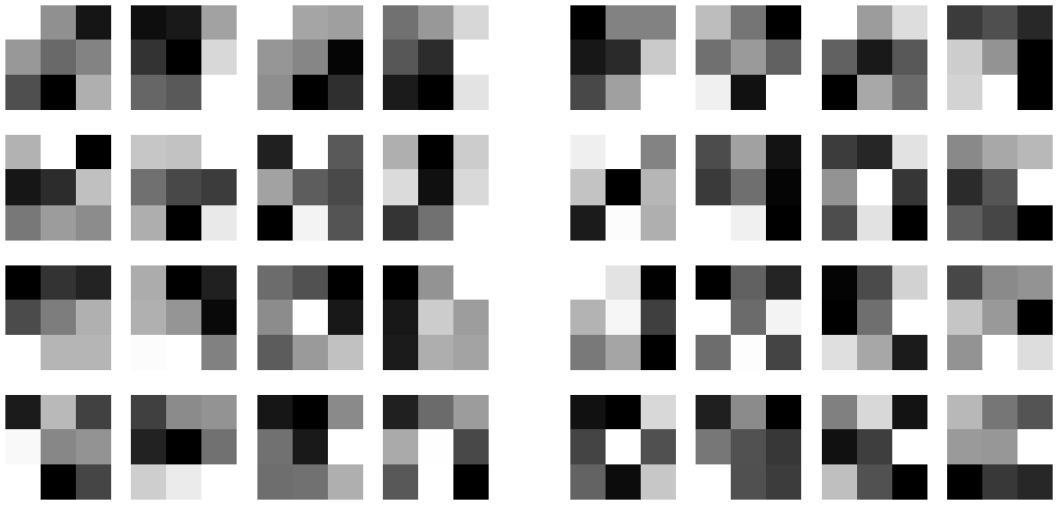


FIGURE 3 – Comparasion de l'entrée (à gauche) avec la sortie (à droite) du réseau

Enfin, la figure 4 présente les filtres de la première couche de convolution 4(a) et de l'avant-dernière couche de déconvolution 4(b). J'ai choisi la première couche de convolution car c'est celle présentant des filtres qui peuvent encore être interprétés alors que les couches suivantes proposent des filtres qui offrent un niveau d'abstraction plus élevé, ce qui est un avantage pour l'extraction de l'information dans l'image mais qui se révèle difficile à appréhender. Et l'avant-dernière couche de déconvolution est celle correspondant au décodage de cette information.

Même s'il reste difficile d'analyser ces filtres, et en particulier ceux des dernières couches, on peut constater que la première couche présente des filtres qui cherchent à détecter les contours ainsi que l'orientation locale de ceux-ci comme on pouvait s'y attendre.



(a) Filtres de la première couche de convolution

(b) Filtres de l'avant-dernière couche de déconvolution

FIGURE 4 – Filtres des couches de convolution et de déconvolution de même taille

4 Analyse de ma méthode d'approche du problème

Ma façon d'approcher ce projet a été une approche exploratoire : je désirais trouver une méthode ou un ensemble de méthode que je pourrais utiliser pour une application dans mon sujet de recherche. Cette façon de fonctionner m'a à la fois permis de découvrir et approfondir plusieurs sujets que l'on avait simplement évoqués dans le cours, mais en contre-partie je manquais d'une ligne directrice pour m'amener à produire quelque chose avant la fin du temps imparti. En particulier, il a été difficile de trouver des articles réellement en lien avec l'idée initiale d'utiliser les autoencoders convolutionnels pour obtenir un meilleur apprentissage, par exemple sur une classification et cela a donc été un problème pour ensuite avoir le temps de développer complètement un système fonctionnel, malgré ma compréhension des sujets abordés grâce à la lecture de divers articles en lien avec cette problématique. En effet, le bloquage rencontré en fin de projet, et que je n'ai pas réussi à résoudre dans la durée de ce projet, est la création de la fonction de perte hybride à l'aide des outils Theano et Lasagne alors que je suis contraint, par l'implémentation des autoencoders convolutionnels que j'ai trouvé, d'utiliser nolearn. En effet, il est déjà difficile de trouver des exemples ou explications pour modifier la fonction de perte de son réseau dans les cas non triviaux (c'est-à-dire les cas où l'on ne veut pas simplement utiliser une autre fonction présente dans Lasagne mais bien en créer une nouvelle) mais je n'ai pas encore trouvé de tel exemple dans le cas de l'utilisation de nolearn. La solution que j'envisage pour la suite sera donc de réimplémenter un autoencoder convolutionel directement avec Lasagne pour pouvoir définir la fonction de perte hybride. Mais le temps disponible dans la durée du projet ne me permettait plus d'adopter cette solution. Mon avis est donc évidemment que j'aurais du me rendre compte de la difficulté de la tâche plus tôt et réorienter mon projet vers une direction plus réalisable, du moins dans le temps imparti.

Cependant je pense qu'il y avait du bon dans ma méthodologie également puisque j'ai

davantage cherché à comprendre les éléments auxquels j'étais confrontés en profondeur plutôt que simplement les survoler et me plonger directement dans une programmation qui n'aurait pas eu de sens. Cette part de mon travail est, je pense, importante et doit être conservée, mais je devrai faire davantage attention à garder un meilleur équilibre entre les périodes de recherche et approfondissement et celles de réalisation, afin de ne pas être pris de court par le temps comme ce fut le cas ici.

Enfin, comme je l'évoquais dans la partie 2, l'objectif de mon projet était de développer un modèle nouveau en utilisant des idées de modèles déjà et existant et de les adapter. En particulier, je pense que cet objectif n'était pas réalisable dans le cadre de ce projet et me demandera encore un travail important avant d'arriver à obtenir un résultat probant. Il est donc un projet très intéressant dans le cadre de ma recherche mais il n'était pas du tout adapté au cadre de ce projet de cours pour des raisons de temps principalement et on peut en observer les conséquences dans les résultats de ce rapport.

5 Conclusion

Durant ce projet je n'aurai donc pas eu la productivité espérée en termes de réalisation technique, mais j'ai pu explorer le thème des autoencoder convolutionnels et de l'apprentissage semi-supervisé, ce qui m'a donné une bonne vision globale de leur utilité et de leur utilisation. A défaut d'avoir pu produire un système, j'ai au moins eu l'occasion de me plonger dans la littérature sur ces sujets ce qui me permettra une meilleure capacité de compréhension pour la suite de ce travail dans le cadre de ma recherche. De plus, cela m'a permis de mettre en ordre mes connaissances et facilitera la progression pour les prochaines étapes.

Références

[1] Brian Cheung, Jesse A. Livezey, Arjun K. Bansal, and Bruno A. Olshausen. Discovering hidden factors of variation in deep networks. *CoRR*, abs/1412.6583, 2014.

[2] K. Cho and X. Chen. Classifying and visualizing motion capture sequences using deep neural networks. In *Computer Vision Theory and Applications (VISAPP), 2014 International Conference on*, volume 2, pages 122–130, Jan 2014.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[4] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database. `http://yann.lecun.com/exdb/mnist/`, 2015. [Online ; accessed 19-Apr-2016].

[5] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. *Artificial Neural Networks and Machine Learning – ICANN 2011 : 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I*, chapter Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction, pages 52–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6] mikesj public. convolutional_autoencoder. `https://github.com/mikesj-public/convolutional_autoencoder`, 2015. [Online ; accessed 19-Apr-2016].

[7] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.