Application des techniques d'appentissage profonds à l'étude de la classification de protéines

Rémy Sun

3 juin 2016

Résumé

Mots-clés. Deep Learning ; Protéines ; Séquence

Introduction

Les techniques dites d'apprentissage profond ont permis de grand progrès dans de nombreux domaines qui vont de la reconnaissance d'image à l'étude de langages naturels (CHO et al. 2014, SOCHER et al. 2011, SUTSKEVER, VINYALS et LE 2014). Néanmoins, les protéines demeurent un domaine relativement inexploré par l'apprentissage profond à l'exception de quelques travaux (SPENCER, EICKHOLT et CHENG 2015) malgré un certain nombre de travaux sur l'expression des gènes (GUPTA, WANG et GANAPATHIRAJU 2015).

L'applications de techniques dévelloppées en langage naturel ou dans d'autres branches de la bioinformatiques permettent de faire des recoupement sémantiques, de la classification ou même de la prédiction sur des chaînes de « mots » ou d'acides aminés. De fait, il semble raisonnable de penser qu'une étude des protéines avec des techniques similaires devraient permettre de similaires progrès dans la compréhension des mécanismes gouvernants le fonctionnement des protéines dont la séquence peptidique est fortement liée au fonctionnement.

Nous nous sommes donc interessés dans ce stage à l'applications de telles techniques aux familles de protéines. Notre intêret ne portait pas seulement sur la classification de telles familles ou la reconstitution d'une proteine à partir d'un séquençage bruité, mais aussi sur l'étude de la possibilité de comprendre des mécanismes des protéines à partir des représentations intermédiaires acquises par les algorithmes d'apprentissages profonds entraînés.

1 Apprentissage profond ?

Une technique d'apprentissage machine Comme toutes les techniques d'apprentissage machine, l'apprentissage profond vise à entrainer un système pour qu'il résolve des situations sans que tous les paramètres nécessaires à la résolution du problème n'aient été calculés par l'implémentateur. Traditionellement, la façon de procéder serait d'utiliser par exemple un réseau de neurones pour faire un perceptron : il y a un couche entrée, une couche cachée et une couche sortie. Entre chaque couche une transformation paramétrée par des variables est effectuée et en sortie on évalue par une fonction de score le résultat renvoyé. Une optimisation sur les paramètres est ensuite effectuée, par descente de gradient par exemple.

Ce qui différencie le Deep Learning de ces techniques d'aprentissage dites « creuses » est d'utiliser plusieurs couches cachées (d'où la notion de profondeur). Cela augmente remarquablement l'abstraction du problème et permet de mieux traiter les problèmes dits compositionnels, c'est-à dire qui peuvent se décomposer en plusieurs composantes.

1.1 Entrainement non supervisé

Autoencodeur Comment faire pour demander à une machine de s'entrainer par elle même sans qu'on lui dise quelle conclusion tirer à partir de son résultat ? Une façon de faire consiste à lui demander de retrouver son entrée. Ainsi, on "encode" l'information dans un premier temps, puis on la décode : c'est l'auto-encodeur.

Traditionellement, c'est à cela (ou aux réseaux basés sur les machines restreintes de Boltzmann) qu'on fait référence quand on parle d'entraînement non-supervisé. A strictement parlé, il s'agit plus d'un entraînement auto-supervisé que non-supervisé, mais aucune réelle méthode d'entrainement non-supervisée n'existe (il faudrait probablement revoir la structure même de l'entrainement).

Donc on entraîne un réseau complexe à... retrouver l'identité ? Ce n'est pas tant le but qui est intéressant ici, mais la façon de l'atteindre. Evidemment, il y a un réel risque que cette manière de l'atteindre ne soit pas particuliérement intéressante non plus (on encode l'identité, puis on décode l'identité). Pour éviter cela on dipose de plusieurs moyens :

— Forcer l'encodeur à encoder l'entrée en une représentation intermédiaire de dimension inférieure. Ainsi, on s'assure de ne pas pouvoir juste propager l'identité. Mieux encore, la représentation intermédiaire, "condensée" peut permettre de découvrir des "points robustes" de ce qu'on est en train d'apprendre. Ce qui caractérise une famille de protéines par exemple... Néanmoins, le grand défaut de cette approche est que la perte d'information est ici inévitable.

— Corrompre l'information donnée en entrée (comme proposé par VINCENT et al. 2008)pour forcer l'encodeur à "deviner" ce qu'il manque, ce qui permettrait par exemple de mettre en lumière des corrélations non évidentes à l'oeil nu.

— Poser des contraintes sur la représentation intermédiaire acquise

— Aléatoirement désactiver certains neurones, ce qui force le système à introduire de la redondance (et donc à choisir ce qui est réellement important). C'est une technique aussi employée dans d'autres réseaux neuronaux pour éviter de spécialiser le réseau sur l'ensemble d'entraînement et non sur le domaine où on veut l'appliquer. Il est raisonnable qu'un auto-encodeur entrainé sur des familles de protéines gére mal les chaînes purement aléatoires, mais il serait problèmatique que cet auto-encodeur n'arrive pas à traiter des protéines proches des exemples d'entraînement mais n'en faisant pas partie.

Une partie de notre travail s'est focalisée sur l'étude de ces représentations intermédiaires et sur ce qu'elles nous apprennent sur les familles de protéine ayant servi à l'entraînement de l'auto-encodeur.

1.2 Réseaux convolutifs

Il y a un motif caractéristique dans les images de chameaux C'est probablement intéressant pour classifier des images d'animaux, mais de là à le repérer sur un pixel... Ce qu'on peut faire par contre, est « scanner » des carrés 3×3. En pratique, c'est créer une couche cachée dont chaque neurone applique une même transformation sur les 9 neurones d'un carré 3×3. Il s'agit donc de faire une convolution sur l'image d'une fonction de transformation !

Nous venons de décrire une couche générant une image réduite dont chaque neurone/pixel donne la valeur de l'application d'une fonction à un carré 3×3. Nous appellerons une telle représentation `feature map` : elle donne la représentation d'une « feature » sur l'image.

Pourquoi s'arrêter à une `feature map` ? Typiquement on crée plusieurs `feature maps` en parrallèle dans un réseau convolutionnel. Rappellons déjà que nous n'avons pas beaucoup de contrôle sur la caractèristique que le

réseau va retrouver puisqu'il va apprendre la caractéristique qui donne le meilleur résultat par lui-même. Multiplier les `feature maps` permet de distinguer plus de points caractéristiques qui seront ensuite traités par une couche supérieure.

Si on veut rajouter une couche convolutionnelle au dessus, elle opérera non seulement une opération sur un carré de chaque `feature map`, mais elle effectura une opérations sur les résultats de cette opération sur chaque feature map.

Réduire la charge de calcul Pour augmenter un peu la robustesse du système à une translation par exemple, on effectue un `pooling` qui consiste à regrouper ensemble des blocs de neurones. Non seulement cela permet de réduire la taille de la représentation considérée, cela fait qu'une simple translation a moins de chance de changer quoi que ce soit à l'image obtenue après transformation.

1.3 Réseaux récurrents

Un unique neurone peut constituer un réseau profond ! L'idée est que l'entrée est traitée comme une suite d'entrées : typiquement c'est le cas d'une phrase par exemple. Chaque mot est traité par la couche récurrente qui va calculer deux choses à partir de cette entrée : une sortie « publique » et une sortie caché qui détermine un paramètre de la couche (une sorte d'état interne caché). Le second élément de la suite est ensuite traité par ce même neurone dont l'état caché a évolué suite au traitement du premier élément de la suite.

Si on « déplie » l'exécution du problème, on réalise que la séquence passe en fait par un nombre de couches cachées égal au nombre d'éléments dans la suite ! Ainsi, nous avons établit un réseau qui est capable de traiter un élément d'une suite en se rappelant en quelque sortes de ce qui s'est passé avant. La seule question qui reste à traiter est le fonctionnement exact de la couche cachée.

Il est possible de faire fonctionner cette couche cachée comme un réseau neuronal complétement relié classique avec une sortie et entrée supplémentaire représentant l'état caché, mais cela pose d'est problème d'évanouissement du gradient lors de la rétropropagation.

Long Short-Term Memory (LSTM) Une architecture de couche dont le comportement naturel est de se rappeler de ce qui s'est passé avant est exposée dans . L'idée est qu'on a un état de cellule, un état caché et une entrée. A chaque passage dans la couche, on calcul quel degré d'information il faut retenir de l'état de la cellule en fonction de l'état caché et de l'entrée. Ensuite on calcule s'il faut ajouter de l'information à l'état de la cellule. Enfin, on fait le calcul de la sortie et de l'état caché à partir des trois données précedentes.

Pourquoi utiliser un réseau récurrent ? Les réseaux récurrents se sont avérés très utiles à l'études du langage naturel puisqu'il permettent de détecter des corrélations dans une phrase qu'un réseau convolutionnel ne détecterait pas forcément (la fenêtre de détection d'une couche convolutionnelle est après tout de taille finie). La sortie récupérée peut être de deux types différents : on peut récupérer la suite des sorties du réseau récurrent, ou on peut choisir de récupérer la dernière sortie de la couche récurrente qui servirait alors de « résumé » de ce qui s'est passé à la lecture de la suite.

Un auto-encodeur récurrent ? Une architecture d'auto-encodeur profonde que nous avons particulièrement étudié (et qui fournit de bons résultats pour la détection de structures comme la copie dans une chaîne) est celle proposée dans CHO et al. 2014. L'idée est d'utiliser un premier auto-encodeur pour servir d'encodeur : on ne récupére que sa sortie sur le dernier terme de la suite d'entrée. Cette sortie est de dimension finie et nous servira de résumé (qui compresse éventuellement l'information). Il suffit ensuite de passer ce résumé dans un autre auto-encodeur récurrent en récupérant ctte fois chaque sortie (qu'on

encode ensuite en une lettre avec un réseau dense et un `softmax`).

1.4 Initialisation et optimisation des réseaux d'apprentissage profond

Entraînement non-supervisé ? Il a été montré dans de nombreux travaux qu'il est possible d'entrainer des auto-encodeurs les uns à la suite et d'aboutir à des situations permettant d'éviter de trop mauvais minima locaux. C'est d'ailleurs ce qui a motivé le second souffle du deep learning vers la fin des années 2000

Néanmoins, des travaux plus récents ont montrés qu'il suffit d'utiliser des initialisation aléatoires bien choisies pour largement passer outre les problème de minima locaux : nous n'avons jamais besoin que d'une instance d'entraînement évitant les minima locaux. Nous avons utilisé dans ce stage l'initialisation proposée dans .

Rétropropagation Nous disposons d'une fonction de score en fin de réseau. Notre but est de minimiser cette fonction en modifiant les paramètres interne du réseau (qui déterminent les transformations effectuées sur l'entrée.). Pour ce faire, on utilise typiquement le gradient de la fonction de coût. Cela peut se faire au travers de quelquechose d'aussi simple que la descente de gradient stochastique, mais cette dernière a de nombreux défauts. Dans cette article nous utiliserons les optimisations dites `adagrad` et `RMSProp`.

De la non linéarité En un sens, les réseaux neuronaux cherchent à observer les données « sous un certain angle » qui permet de trouver des corrélations : cela permet de faire de la classification, de la reconstruction, voire de la prédiction. Néanmoins si on se borne à effectuer des opérations linéaires (sommes pondérées d'entrés, trnasformations affines) il est tout a fait possible de passer à coté d'un regroupement intéressant des points (les fonction linéaire préservent notemment l'alignement ce qui peut se révéler être un grand problème). De fait, il y a toujours « à la fin » d'une couche une fonction dite « d'activation » qui opére une transformation non-linéaire sur la sortie. Traditionnellement, on applique une sigmoïde ou une tangente hyperbolique, mais Hinton a montré qu'il est plus judicieux d'utiliser une fonction de seuillage `ReLU`.

2 Travail réalisé

3 Vérification

4 Contribution

Conclusion

Références

CHO, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In : *CoRR* abs/1406.1078. URL : http://arxiv.org/abs/1406.1078.

GUPTA, Aman, Haohan WANG et Madhavi GANAPATHIRAJU (2015). "Learning structure in gene expression data using deep architectures, with an application to gene clustering". In : *bioRxiv*. DOI : 10.1101/031906. eprint : http://biorxiv.org/content/early/2015/11/16/031906.full.pdf. URL : http://biorxiv.org/content/early/2015/11/16/031906.

SOCHER, Richard et al. (2011). "Semi-supervised recursive autoencoders for predicting sentiment distributions". In : *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, p. 151–161.

SPENCER, Matt, Jesse EICKHOLT et Jianlin CHENG (2015). "A Deep Learning Network Approach to Ab Initio Protein Secondary Structure Prediction". In : *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 12.1, p. 103–112. ISSN : 1545-5963. DOI : 10.1109/TCBB.2014.2343960. URL : http://dx.doi.org/10.1109/TCBB.2014.2343960.

SUTSKEVER, Ilya, Oriol VINYALS et Quoc V LE (2014). "Sequence to Sequence Learning with Neural Networks". In : *Advances in Neural Information Processing Systems 27*. Sous la dir. de Z. GHAHRAMANI et al. Curran Associates, Inc., p. 3104–3112. URL : http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

VINCENT, Pascal et al. (2008). "Extracting and Composing Robust Features with Denoising Autoencoders". In : *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland : ACM, p. 1096–1103. ISBN : 9781605582054. DOI : 10.1145/1390156.1390294. URL : http://doi.acm.org/10.1145/1390156.1390294.