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

Rémy Sun

27 juin 2016

Résumé

L'application des techniques d'apprentissage profond à des chaînes peptidiques demeure un domaine peu exploré. Nous avons cherché à trouvé des caractéristiques importante à l'aide d'auto-encodeurs et avons étudié la possibilité d'utiliser des architecture profonde pour la classifictation

Mots-clés. Deep Learning ; Protéines ; Séquence

Introduction

La bonne compréhension des mécanismes régissant le fonctionnement des protéines existantes est un enjeu important dans de nombreux domaines. Cependant il est peu réaliste au vu des moyens actuels de procéder à l'étude détaillée de tous les mécanismes en jeu pour chaque protéine en raison du coût élevé des manipulations nécessaires. Néanmoins, nous disposons d'un grand nombre de séquences peptidiques de protéines, qui influent directement sur les fonctions d'une protéine.

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 bioinformatique permettent de faire des recoupements 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.

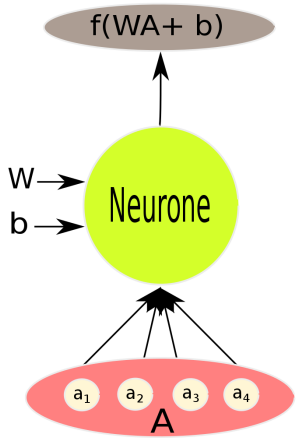

FIGURE 1 – Exemple de neurone

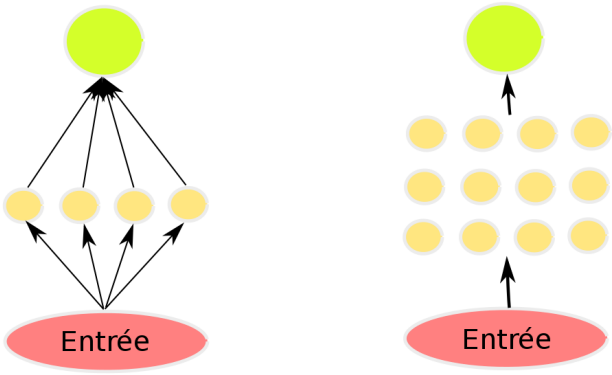

FIGURE 2 – Réseau de neurones classique à gauche, Réseau profond à droite

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. Il est facile en première approximation de comprendre pourquoi cette façon de procéder est intéressantes : nous comprenons des choses simples avant de comprendre les choses plus complexes à partir de ce qui a déjà été appris

1.1 Entrainement non supervisé

Autoencodeur Une grande problèmatique de l'apprentissage profond est le besoin d'accéder à de grandes bases de données pour pouvoir entrainer les réseaux. Si cette difficulté n'est plus d'actualité dans nombreux domaines, elle est indissociable de notre étude des protéines. 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 dispose 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

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 protéines ayant servi à l'entraînement de l'auto-encodeur.

1.2 Réseaux convolutifs

Il y a un motif caractéristique (feature) dans les images d'oiseaux 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.

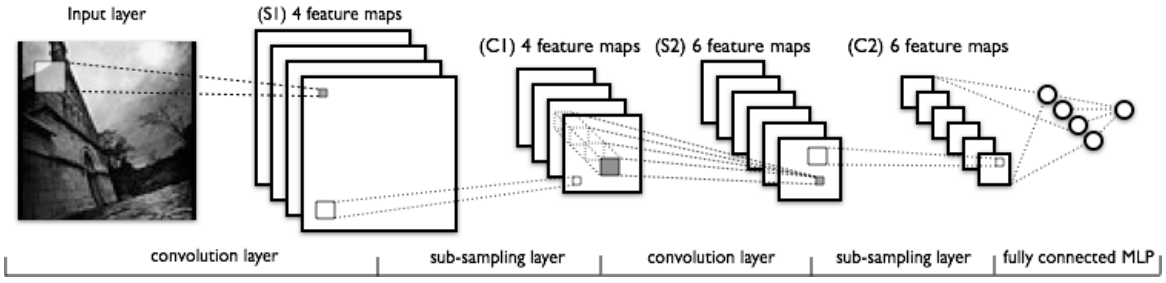

FIGURE 3 – Réseau convolutionnel LeNet5

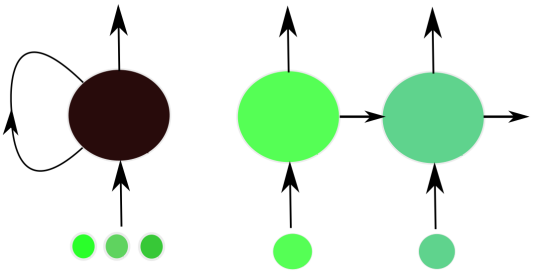

FIGURE 4 – Réseau récurrent à gauche, forme dépliée temporellement à droite

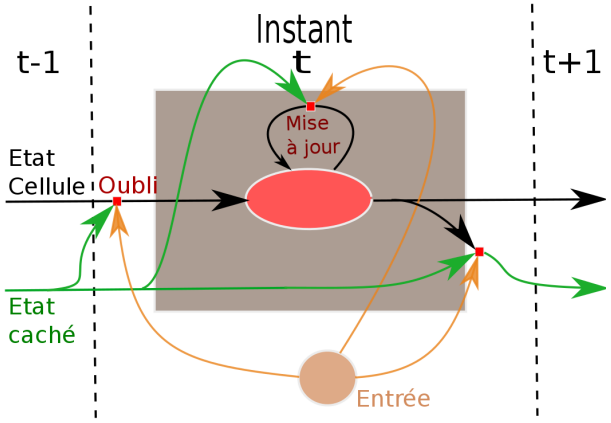

FIGURE 5 – Exemple d'unité LSTM

Une architecture typique se constitue donc d'un enchaînement de couches de convolution et de pooling, avec une ou plusieurs couches « classiques » permettant de tirer une conclusion sur les information récupérées. Une telle architecture est proposée dans LECUN et al. 1998 sous la forme du réseau LeNet5

1.3 Réseaux récurrents

Il est utile de se rappeler des mots précédents à la lecture d'une phrase 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 HOCHREITER et SCHMIDHUBER 1997. L'idée est qu'on a un état de cellule, un état caché et une entrée. A chaque passage dans la couche, on calcule 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 cette fois chaque sortie (qu'on encode ensuite en une lettre avec un réseau dense et un `softmax`). Dans l'article original, la sortie du décodeur au temps t lui est aussi donné en entrée au temps $t+1$, mais ce n'est pas le cas dans notre étude.

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 montré qu'il suffit d'utiliser des initialisations aléatoires bien choisies pour largement passer outre les problème de minima locaux quand l'ensemble d'apprentissage est de taille élevée : 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 . Cependant, nous avons aussi dû travailler avec des jeux de données de taille faible, ce qui a nécéssité le passage par de l'apprentissage non-supervisé pour les taches de classification où peu d'exemples labélisés existent.

Rétropropagation Nous disposons d'une fonction de score en fin de réseau. Notre but est de minimiser cette fonction en modifiant les paramètres internes 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` (très efficace dans les réseaux convolutifs) et `RMSProp` (très efficace dans les réseaux récurrents).

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, transformations 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` (Rectified Linear Unit).

Eviter le sur-apprentissage On peut léatoirement désactiver certains neurones, ce qui force le système à éviter de spécialiser le réseau sur l'ensemble d'entraînement et non sur le domaine où on veut l'appliquer (SRIVASTAVA et al. 2014). Il est raisonnable qu'un réseau entrainé sur des familles de protéines gére mal les chaînes purement aléatoires, mais il serait problèmatique qu'il n'arrive pas à traiter des protéines proches des exemples d'entraînement mais n'en faisant pas partie. En effet, si on veut catégoriser les pompiers, et que par hasard tous ceux de l'exemple d'entraînement sont bruns, il y a un risque d'associer pompier et brun.

L'idée est la suivante : s'il est autorisé de copier sur ses camarades à un examen, et qu'il est connu qu'un éléve aura 20 à l'épreuve, un comportement possible est de ne rien faire et simplement compter sur l'éléve qui travaille. Cela est problématique car on dépend alors uniquement du résultat de cet éléve/neurone, et la perception des choses qu'on a est fortement biaisée par ce que ce neurone sait. La solution adoptée revient à annoncer qu'un certain pourcentage de la classe sera malade le jour de l'examen, ce qui force tout le monde à apprendre.

Il a néanmoins été montré dans GAL 2015 qu'une telle architecture est complètement inutile dans le cas des réseau récurrents. Néanmoins, il y est proposé d'effectuer un oubli dans la dépendance temporelle du réseau (c'est à dire au niveau de l'état caché) ce qui permet d'obtenir des résultats similaires à la technique précédente pour des réseaux classiques.

2 Travail réalisé

2.1 Matériel

Nous avons effectué l'ensemble de nos travaux à l'aide librairies python.

Pour ce qui est de l'aspect apprentissage profond, nous avons principalement utilisé `keras` (CHOLLET 2015) qui est une libraire haut niveau permettant de manipuler facilement des couches d'apprentissage. Ce choix vient d'une part de la nature des problèmes étudiés qui nous mènent à tenter l'utilisation de plusieurs architectures (ce qui invalide des librairies comme Caffee pour C++ spécialisé en réseaux convolutifs) et de l'autre de la grande souplesse offerte par Keras. En effet, `keras` peut utiliser deux librairies bas niveau python (il est possible de choisir entre les deux) : le grand classique `Theano` (AL-RFOU et al. 2016) et le plus récent `Tensor Flow` (MARTIN ABADI et al. 2015) de Google.

Pour ce qui est de la manipulation de séquence peptidiques à proprement parlé, nous avons utilisé des données de la base de donnée `SCOPe` (FOX, BRENNER et CHANDONIA 2014) et la librairie python Biopython (COCK et al. 2009)

2.2 Représentation des chaînes peptidiques

Ce qui a été fait à de maintes reprises par le passé est la représentation des protéines non pas comme chaines d'acides, mais comme un vecteur dépendant de différents paramètres. Ces paramètres vont de notions simplistes comme le nombre d'occurence de chaque acide aminé dans la chaîne à des considérations beaucoup plus complexes comme l'étude des fonctions de la protéine.

Notre travail s'est principalement basé sur l'étude de proteines en tant que chaîne d'acides aminés représentés par des lettres. En quelque sorte, nous traitons donc des suites de « mots ». De fait, il est nécessaire de déterminer une façon de représenter ces différents mots. Il est facile de voir pourquoi la première idée de simplement indexer les acides par des entiers est problématique : pourquoi la lettre « A » serait elle plus proche du « B » que du « Z » ? Pour résoudre ce problème, nous nous sommes intéréssés à ce qui a été en traitement de langages naturels.

Une représentation répandue, bien qu'assez simpliste consiste à créer un espace contenant autant de dimensions qu'il y a d'acides aminés et représenter chaque acide aminé par un vecteur d'une base orthogonale de cet espace. Plus simplement, cela veut dire qu'on représente chaque acide par un vecteur ne contenant que des 0 et un 1. Cette représentation permet d'avoir des résultats, mais elle présente en langage naturel l'inconvénient d'engendrer une représentation dans l'espace très lacunaire, ce qui crée des espaces inutilement grands. Ce problème est secondaire dans notre cas puisqu'il n'y a qu'une vingtaine d'acides à considérer.

Néanmoins, il est intéressant de remarquer que des outils ont été inventés en langages naturels pour passer outre ce problème. L'idée est d'apprendre un représentation de dimension inférieure distribuée prenant en compte les mots apparaissant généralement avec celui considéré. Cela se fait avec des techniques d'apprentissages qui cherchent à maximiser le score de phrases valides et minimiser celui de phrases non valides. Ces techniques ne sont pas applicables à notre problème à cause de la façon dont les séquences peptidiques fonctionnent, mais nous avons voulu retenir l'idée qu'on peut placer les acides dans un espace de dimension faible en tenant compte de leur spécificités. En regardant 4 propriétés physico-chimiques, nous avons donc créé une telle représentation. L'aspect important d'une telle représentation est qu'elle permet de justifier du fait que deux acides sont proches du point de vue de leur représentation, ou du fait que trois acides sont equidistants. En effet, la représentation précédente posait le problème inverse de l'indexage : pourquoi tous les acides sont-ils équidistants ?

2.3 Choix des architectures d'apprentissage profond

Dans le cadre du traitement de séquences peptidiques, il semble tout indiqué d'utiliser des réseaux récurrents puisqu'une structure temporelle semble apparaître dans l'enchaînement des acides aminés. Néanmoins, le grand défaut de réseaux récurrent est que, contrairement aux réseaux profonds dits « classiques », ils repérent des dépendances temporelles et non hiérarchiques. Pour pallier à ce problème, il semble nécessaire d'empiler les réseaux récurents, ce qui a vite des conséquences significatives en termes de temps de calculs dès qu'on traite des séquences un peu longues.

Une autre architecture intéressante est celle des réseaux convolutionnels exposés plus haut. Dans la façon de procéder décrite plus haut, un acide aminé est traité comme un « mot ». Cela ne correspond pas nécéssairement à une quelconque réalité, à moins qu'on pense qu'une lettre est un mot. Il n'existe cependant pas de moyen d'extraire à priori un « mot » (un groupe d'acides) d'une chaîne peptidique : il n'y a pas d'acide « séparateur ». Cependant, les réseaux convolutionnels offrent un moyen alternatif de repérer ces mots : si une `feature map` est ap-

pliquée sur 7 acides, alors cette `feature map` sera capable de repérer (après entraînement) un mot (caractéristique) de longueur inférieure ou égale à 7.

2.4 Etude préliminaire sur séquences artificielles

Nous avons généré aléatoirement plusieurs jeux de données pour comparer les capacités de diverses architectures d'auto-encodeurs et classificateurs. Nous avons testé deux types de jeux de données : un jeu de données où la première moitié de la chaine est fixée et où la seconde moitié est telle que chaque lettre est tirée avec probabilité uniforme ; et un jeu de données où la premiére moitié est tirée avec probabilité uniforme sur chaque lettre et où la seconde moitié est égale à chaque moitié.

2.4.1 Auto-encodeurs

Nous envisageons principalement deux architectures d'encodeurs : un réseau convolutionnel (à couche multiples) et un réseau récurrent LSTM (à une ou plusieurs couches) dont nous récupérons la dernière sortie. Le décodeur est dans tous les cas un réseau récurrent LSTM.

Les deux architectures n'ont aucun problème à repérer la caractéritique des données dont le début est fixé, ce qu'elle reconstitue très bien. Par contre, il semble que l'architecture dont l'encodeur est un réseau convolutionnel ne parvient pas à décoder correctement les caractères aléatoires. De plus, il ne semble même pas repérer la relation de « copie » présente dans le second jeu de données. Au contraire, il semble que l'encodeur récurrent, même à une seule couche apprend très vite à créer des mots dont les deux moitiées sont très similaires, même si elles n'ont pas grand chose à voir avec l'entrée réelle. Sous condition de disposer d'un temps d'entraînement assez long, l'encodeur récurrent permet aussi de reconstituer correctement les caractéres tirés aléatoirement.

2.4.2 Classificateurs

2.5 Clustering de protéines

Les auto-encodeurs génèrent un représentation intermédiaire dans un premier temps. Nous avons voulu étudier cette représentation intermédiaire pour en apprendre plus sur les choses auquelles l'auto-encodeur fait attention quand il encode la protéine et qui lui permettent de reconstruire par la suite la protéine. JIAN-WEI et al. 2013 a déjà été effectuée sur la capacité de réseaux d'auto-encodeurs empilés à classifier la structure secondaire de protéines, mais cette étude ne s'est malheureusement pas attardé sur le problème de la représentation intermédiaire.

En première approximation, on pourrait penser retrouver un hyperplan ou une nappe aparentable à une feature connue. Néanmoins une telle étude est difficile à exploiter. Nous nous sommes penchés sur la piste du clustering de protéines en fonction de leur représentation intermédiaire. En effet, si on regroupe ensemble des protéines « proches » du point de vue de leur représentation intermédiaire, il est peut-être possible de remarquer des points remarquables dans la structure de protéines qui pourront être ré-utilisés de manière indépendante de l'apprentissage profond plus tard.

Un simple algorithme de K-means permet de faire le découpage en parties de l'ensemble d'apprentissage.

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.

CHOLLET, François (2015). *Keras*. https://github.com/fchollet/keras.

COCK, Peter J. A. et al. (2009). "Biopython : freely available Python tools for computational molecular biology and bioinformatics". In : *Bioinformatics* 25.11, p. 1422–1423. DOI : 10.1093/bioinformatics/btp163. eprint : http://bioinformatics.oxfordjournals.org/content/25/11/1422.full.pdf+html. URL : http://bioinformatics.oxfordjournals.org/content/25/11/1422.abstract.

FOX, Naomi K, Steven E BRENNER et John-Marc CHANDONIA (2014). "SCOPe : Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and classification of new structures". In : *Nucleic acids research* 42.D1, p. D304–D309.

GAL, Yarin (2015). "A theoretically grounded application of dropout in recurrent neural networks". In : *arXiv preprint arXiv :1512.05287*.

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.

HOCHREITER, Sepp et Jürgen SCHMIDHUBER (1997). "Long short-term memory". In : *Neural computation* 9.8, p. 1735–1780.

JIAN-WEI, Liu et al. (2013). "Predicting protein structural classes with autoencoder neural networks". In : *Control and Decision Conference (CCDC), 2013 25th Chinese*. IEEE, p. 1894–1899.

LECUN, Yann et al. (1998). "Gradient-based learning applied to document recognition". In : *Proceedings of the IEEE* 86.11, p. 2278–2324.

MARTIN ABADI et al. (2015). "TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems". In : Software available from tensorflow.org. URL : http://tensorflow.org/.

AL-RFOU, Rami et al. (2016). "Theano : A Python framework for fast computation of mathematical expressions". In : *arXiv e-prints* abs/1605.02688. URL : http://arxiv.org/abs/1605.02688.

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.

SRIVASTAVA, Nitish et al. (2014). "Dropout : a simple way to prevent neural networks from overfitting." In : *Journal of Machine Learning Research* 15.1, p. 1929–1958.

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.