Batista Reda
Debbah Nagi
Fortuné Audrey
Vander Meersche Yann

Master 2 Bioinformatique
Année 2020-2021

PROJET KAGGLE

OpenVaccine : Prévision de la dégradation de l'ARNm du vaccin contre la COVID-19

Université de Paris

I. Introduction

Près de 200 laboratoires du monde entier sont actuellement dans une course effrénée à l'élaboration d'un vaccin contre la COVID-19. Cependant, seulement dix de ces vaccins-candidats ont atteint la phase 3 des études cliniques chez l'Homme. C'est le cas du *mRNA-1273* de *Moderna Therapeutics* qui utilise une stratégie innovante. En effet, au lieu d'injecter un virus inactivé, il est possible d'administrer de l'ARN messager du virus pour produire l'antigène directement dans les cellules cibles.

Cependant, le problème majeur de ce type de vaccin est sa fragilité. Pour rester fonctionnel, un vaccin à ARN doit être conservé à très basse température et ne subir aucun choc, ce qui complique sa distribution à travers le monde.

L'objectif de ce projet proposé par l'université de Stanford est de développer un modèle capable de prédire des indicateurs de fragilité des ARNm. Pour cela, nous utiliserons un réseau de neurones profond pour prédire les probabilités de dégradation des ARN dans différentes conditions.

II. Matériel et méthodes

1. Analyse des jeux de données

Pour ce projet, nous avons à notre disposition un jeu de données d'apprentissage et un jeu de test.

Le jeu d'apprentissage est composé de données tirées de 2400 séquences d'ARNm de 107 pb. En plus de sa séquence en acide nucléique, nous disposons de différentes variables prédites qui seront utilisées pour encoder les données du réseau de neurone, ainsi que des valeurs déterminées expérimentalement que nous chercherons à prédire.

Parmi ces ARNm, seulement 1589 ont réussi à passer les différents filtres de qualité vérifiant que les ratios signal sur bruit des mesures expérimentales sont satisfaisants. En représentant les relations entre les valeurs moyennes par ARNm des variables expérimentales *(Fig. 1)*, nous pouvons voir une nette amélioration de leurs distributions une fois filtrées (S/N filter = 1). Nous utiliserons donc uniquement les données filtrées pour le reste des analyses.

Figure 1 : Pair plot représentant les relations entre les valeurs moyennes par ARNm des variables expérimentales. Les points bleus représentent les ARNm ayant passés les filtres, et les points rouges les ARNm ne les ayant pas passés.

Nous avons ensuite étudié le pourcentage d'homologie entre les séquences restantes. Pour cela, nous avons utilisé cd-hit-est [1], un serveur web permettant de regrouper des séquences d'acides nucléiques selon leur identité de séquence. Nous avons déterminé que parmi ces 1589 séquences filtrées, 848 possèdent au moins un homologue à 90% d'identité, et que ces séquences sont regroupées en seulement 266 clusters. Cette forte homologie diminue donc encore la diversité d'information pour l'apprentissage du réseau.

Enfin, nous avons représenté la fréquence de chaque acide nucléique le long de la séquence pour toutes les séquences filtrées (*Fig. 2*) avec le serveur web WebLogo 3 [2]. Nous apprenons que sur les 107 pb, 30 sont présentes dans toutes les séquences et ajoutent de la redondance à notre jeu de données. Ces bases sont problématiques puisqu'elles risquent de bruiter l'apprentissage, mais doivent être prédites pour le projet.

Ces séquences peuvent avoir plusieurs rôles possibles, tels que stabiliser et/ou protéger les ARNm comme le feraient la coiffe et la queue poly-A, ou avoir un rôle fonctionnel.

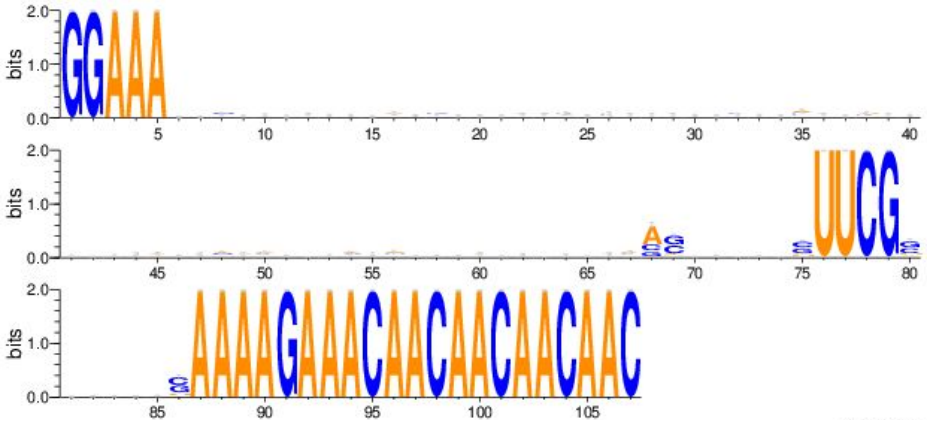


Figure 2 : Logo des séquences d'apprentissage filtrées.

Le jeu de test quant à lui, met à notre disposition la séquence en acide nucléique et des variables prédites à partir de cette dernière, mais pas les valeurs expérimentales, puisque ces valeurs permettront d'évaluer les performances du modèle pour la compétition. Ces données sont issues de 629 séquences d'ARNm de 107 pb ayant passé les filtres de qualité et ayant un faible pourcentage d'homologie, et de 3005 séquences de 130 pb non filtrées ayant beaucoup d'homologie. Enfin nous retrouvons également les 30 pb conservées dans toutes les séquences en représentant les logo de séquences (*Fig. S1 et S2*).

2. Transformation des données

Un réseau de neurones n'est pas capable d'interpréter une séquence biologique directement. Pour cela, nous devons soigneusement choisir l'information à lui présenter. Pour caractériser nos ARNm, nous utiliserons les descripteurs suivants :

- La **séquence** en acides nucléiques (A, U, G, C),
- La **structure** prédite décrivant si une base donnée de la séquence est appariée ("()") ou non (".") à une autre base,
- Une prédiction du **type de boucle** (S, M, I, B, H, E, X) réalisée par *bpRNA* du logiciel *Vienna RNAfold 2 structure*.

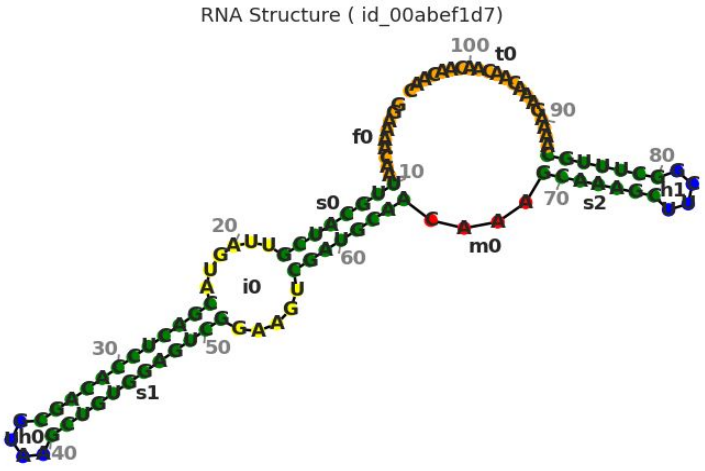


Figure 3 : Structure d'ARNm *abefd7* visualisé et prédit à l'aide de *forgi* [3]. ***Vert (s)** : Base Appariée, **Orange (f,t) :** Bases non appariées, **Bleu (h)** : Boucle en épingle à cheveux, **Jaune (i)** : Boucle interne, **Rouge (m)** : Boucle multibranche.*

Ensuite, pour rendre lisible ces informations par le réseau, nous devons les encoder. Pour cela nous avons choisi le one-hot encoding. Nous nous retrouvons donc avec 14 features, 4 pour la séquence, 3 pour la structure et 7 pour le type de boucle, et cela pour toutes les positions de la séquence (*Fig 4, droite*).

Enfin, nous utiliserons ces données pour prédire 5 descripteurs de l'ARNm suivant :

- La **reactivity** : probabilité que la structure secondaire existe expérimentalement,
- Le **deg_ph10** : probabilité de dégradation de liaisons des nucléotides à pH 10,
- Le **deg_Mg_ph10** : probabilité de dégradation de liaisons des nucléotides à pH 10 en présence de magnésium,
- Le **deg_50C** : probabilité de dégradation de liaisons des nucléotides à une température de 50 °C,
- Le **deg_Mg_50C** : probabilité de dégradation de liaisons des nucléotides à une température de 50 °C en présence de magnésium.

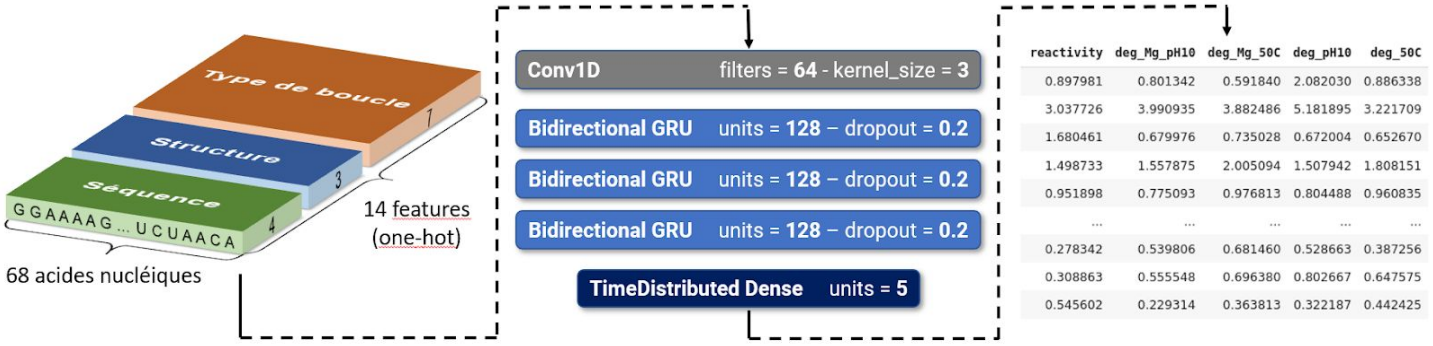
A cause d'un problème expérimental, la stabilité des ARNm n'a pas pu être mesurée sur les 39 derniers résidus des différentes séquences. Nous nous retrouvons donc avec seulement 68 valeurs pour les séquences d'apprentissage de 107 pb, alors que le projet Kaggle requiert une prédiction sur la totalité de la séquence du jeu de test (107 et 130 pb).

Pour remédier à ce problème, nous avons choisi de réaliser notre apprentissage uniquement sur les 68 premières bases de la séquence. Nous avons donc tronqué nos données d'input, ce qui permet d'éliminer 25 des 30 bases communes à toutes les séquences (*Fig. 1*).

3. Architecture du réseau de neurone

Pour notre réseau, nous avons choisi d'utiliser une architecture relativement simple puisque notre jeu d'apprentissage est petit, et que ces séquences partagent un pourcentage d'homologie élevé. Il faut donc un réseau capable d'identifier les features importants, sans pour autant surapprendre et ne pas réussir à généraliser sur le jeu de test.

Comme vu précédemment, nous utiliserons seulement les 68 premières bases pour l'apprentissage du réseau. Mais, puisque la totalité des séquences de tests doit être prédite (107 ou 130 pb), une fois entraînée, nous transférons les poids de notre réseau dans un nouveau modèle en changeant la taille des données d'entrée du réseau.



reactivity	deg_Mg_pH10	deg_Mg_50C	deg_pH10	deg_50C
0.897981	0.801342	0.591840	2.082030	0.886338
3.037726	3.990935	3.882486	5.181895	3.221709
1.680461	0.679976	0.735028	0.672004	0.652670
1.498733	1.557875	2.005094	1.507942	1.808151
0.951898	0.775093	0.976813	0.804488	0.960835
...	...	...	...	...
0.278342	0.539806	0.681460	0.528663	0.387256
0.308863	0.555548	0.696380	0.802667	0.647575
0.545602	0.229314	0.363813	0.322187	0.442425

Figure 4 : Workflow de notre réseau de neurones. Encodage des données (droite), architecture du réseau (milieu), format des prédictions du réseau (droite)

Ce réseau est composé de 4 couches cachées :

- Une couche Conv1D avec un kernel size de 3 ayant pour but de détecter des features importants dans les données et simplifier l'apprentissage. Cela va aussi aider notre réseau à s'adapter aux différentes tailles d'input / output du jeu de test.
- Trois couches GRU (*Gated Recurrent Unit*) bidirectionnelles, permettant de prendre en compte l'information de façon récurrente, puisque l'information de séquence à une position donnée a peu de sens si nous ne prenons pas en compte ses voisins.

  Même si les LSTM sont plus complexes que les GRU, ils ne donnent pas de meilleurs résultats dans notre cas, puisque notre jeu de données est trop petit pour optimiser correctement ses paramètres.
- Enfin, pour la couche de prédiction, nous utilisons une couche dense TimeDistributed. Cela n'améliore pas les performances par rapport à une couche dense classique, mais réduit légèrement le nombre de paramètres, et par conséquent diminue le temps d'apprentissage.

Pour l'apprentissage, nous avons utilisé la *mean square error* comme fonction de *loss* et nous suivrons l'évolution du MCRMSE (*mean columnwise root mean squared error*), la fonction évaluant les résultats soumis.

Nous utiliserons également Adam comme optimiseur. Plusieurs autres optimiseurs plus performants ont été testés (radam, ranger…), mais ceux-ci ne donnent pas de meilleurs résultats. En effet, même s'ils réduisent le surapprentissage en stabilisant rapidement le réseau, Adam permet de poursuivre l'apprentissage plus longtemps ce qui permet d'améliorer légèrement les résultats.

Enfin, nous utilisons les callbacks "*early stopping*", permettant d'arrêter le réseau avant qu'il surapprenne lorsqu'il n'y a pas d'amélioration de validation loss pour 10 epochs, et "*ReduceLROnPlateau*", permettant de ralentir la vitesse d'apprentissage du réseau lorsque le réseau commence à ne plus apprendre.

Puisque beaucoup de nos séquences sont homologues, nous avons tenté de pondérer leur importance avec "*sample_weight*", où le poids serait égal à 1 sur le nombre de séquences dans le cluster. Mais, cela dégrade nos performances, nous ne les utiliserons donc pas.

III. Résultats et discussions

Pour ce projet, nous avons entraîné différentes architectures de réseaux et comparé leurs performances.

Dans un premier temps, nous avons testé un réseau dense simple (*Fig. S6, haut gauche*), ainsi qu'un réseau de convolution 1D simple combiné au réseau dense simple (*Fig. S6, droite)*. Nous avons également testé une architecture plus complexe, le GRU (*Fig. S6, bas* gauche), ainsi que cette architecture GRU combinée avec une couche de convolution 1D (*Fig. 4, milieu*). Ces réseaux ont été testés sur des données de plus en plus complexes : la "reactivity" seule, les 3 descripteurs évalués par Kaggle, et la totalité des descripteurs mesurés expérimentalement.

		Reactivity	Reactivity Deg_Mg_ph10 Deg_Mg_50C	All five metrics at once
Dense	MSE	0.1842 ± 0.0333	0.2302 ± 0.0249	0.2439 ± 0.0084
	MCRMSE	0.4132 ± 0.0357	0.4269 ± 0.0283	0.4764 ± 0.0084
Conv1D + Dense	MSE	0.1103 ± 0.0584	0.1992 ± 0.0641	0.2292 ± 0.0432
	MCRMSE	0.3070 ± 0.0808	0.3952 ± 0.0666	0.4588 ± 0.0514
GRU	MSE	0.0504 ± 0.0024	0.0619 ± 0.0032	0.0601 ± 0.0035
	MCRMSE	0.2121 ± 0.0048	0.2154 ± 0.0049	0.2293 ± 0.0056
GRU + Conv1D	MSE	0.0495 ± 0.0030	0.0595 ± 0.0037	0.0573 ± 0.0023
	MCRMSE	0.2098 ± 0.0062	0.2107 ± 0.0056	0.2234 ± 0.0040

Tableau 1 : Récapitulatif des performances des différents modèles (en lignes) évaluées sur le jeu de test en cross validation pour des données à prédire de plus en plus complexes (colonnes).

Tout d'abord, nous pouvons remarquer que quelque soit le réseau, plus les données à prédire sont complexes, moins les performances sont bonnes.

Le réseau dense qui est le réseau le plus simple, produit sans surprise les performances les plus mauvaises. Cependant, lorsque nous l'associons avec des couches de convolution, nous observons une amélioration des résultats. Cette amélioration montre que les couches de convolution permettent au réseau d'extraire plus d'information en détectant les features importants dans les données, et ainsi simplifier le reste de l'apprentissage. Mais, même après cette amélioration, les performances restent mauvaises. Nous allons donc tester des architectures plus complexes comme les GRU.

Nous pouvons remarquer que le réseau GRU améliore significativement les performances, en diminuant le MCRMSE de moitié pour la prédiction des 5 metrics simultanément. Nous pouvons là aussi nous demander si l'ajout de couches de convolution permettrait d'améliorer les résultats. Pour cela, nous avons ajouté une couche de convolution 1D avant notre réseau GRU, ce qui nous a en effet permis d'améliorer légèrement nos résultats. Nous avons également entraîné des réseaux plus complexes, avec par exemple plus de couches de convolution, mais sans observer d'amélioration significative des performances, et entraînant parfois du surapprentissage.

Puisque nous cherchons à prédire les 5 variables pour la compétition Kaggle, nous réaliserons le reste de nos tests sur le réseau Conv1D+GRU.

Après avoir réalisé la cross-validation, nous avons représenté l'évolution du MCRMSE (*Fig. 5*). Notre réseau ne disposant que de peu de données informatives pour l'apprentissage, celui-ci est légèrement instable. En effet, en fonction des données utilisées pour l'apprentissage, notre réseau pourra soit stagner prématurément, soit réussir à dépasser ce plateau et entraîner une amélioration du MCRMSE, ainsi qu'une légère amélioration du val_MCRMSE.

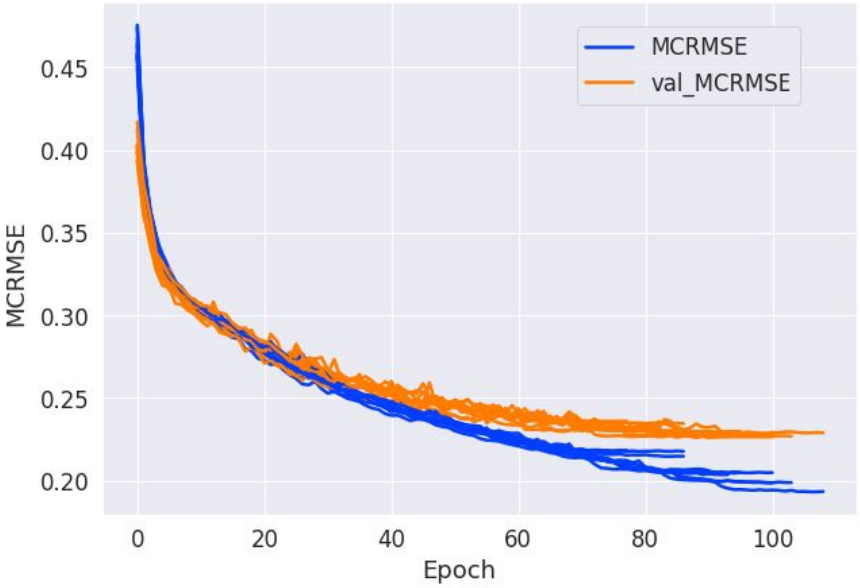


Figure 5 : Graphique de l'évolution des valeurs d'apprentissage de MCRMSE et de val_MCRMSE en cross-validation.

Enfin, nous avons superposé les distributions réelles des différents indicateurs de fragilité des ARN et leurs prédictions en cross-validation (*Fig. 6*). Nous pouvons voir que les deux distributions sont très proches.

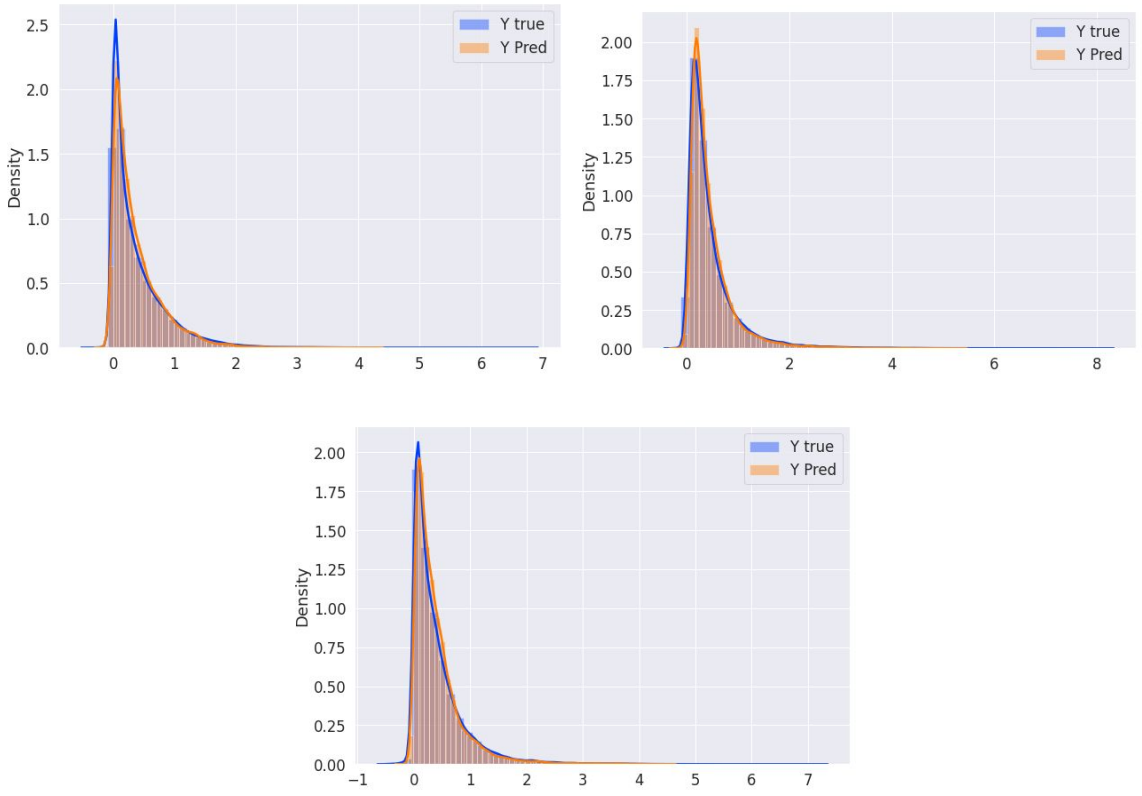


Figure 6 : Graphique des valeurs prédites par le modèle contre les valeurs expérimentales. Reactivity (en haut à gauche), deg_Mg_ph10 (en haut à droite) et deg_ph10 (en bas).

IV. Conclusion

Afin de mener à bien ce projet, nous avons tout d'abord exploré les données qui nous ont été fournies. Pour cela, diverses analyses ont été réalisées et nous ont permis de déterminer que les données expérimentales à prédire sont bruitées, et que les séquences que nous utiliserons pour la prédiction possèdent un fort pourcentage d'homologie.

Nous avons donc décidé de n'utiliser que les données filtrées pour réduire le bruit, et tenté d'utiliser du *sample weight* pour compenser l'homologie de séquence, mais sans résultat probant.

Après avoir expérimenté plusieurs architectures, nous avons retenu un réseau GRU bidirectionnel couplé à une couche de convolution, puisqu'il nous donne nos meilleurs résultats.

Mais, lorsque nous testons notre réseau sur les données de test de Kaggle, nos résultats ne sont pas aussi satisfaisants, et nous montre que notre modèle peut être amélioré. En effet, nous avons atteint un score public de 0.32240, ce qui correspondrait à la place 1491^{e}/1636 et un score privé de 0.41490 soit 1431^{e}/1636.

Dans un premier temps, avoir plus de données d'apprentissage à notre disposition nous aiderait sûrement à améliorer nos performances. Cela nous permettrait d'être plus strict dans la sélection des données. Ainsi nous aurions moins d'homologie et pourrions utiliser des architectures plus complexes sans surapprendre.

Pour cela, nous pourrions tenter d'augmenter de manière artificielle le nombre de séquences de notre jeu de données d'apprentissage, ce qui risque de ne pas être très pertinent biologiquement parlant.

Nous pourrions également utiliser un autre format pour nos données, en prédisant par exemple chaque acide nucléique un à un en les encodant comme une fenêtre de séquence entourant la position à prédire.

Enfin, nous pourrions réaliser du transfert learning, en entraînant un réseau avec d'autres données d'ARN sur un problème proche et en adaptant le réseau pour notre problème ensuite.

V. Références

[1] Y. Huang, B. Niu, Y. Gao, L. Fu, W. Li, CD-HIT Suite: a web server for clustering and comparing biological sequences, Bioinformatics. 26 (2010) 680–682.

[2] G.E. Crooks, WebLogo: A Sequence Logo Generator, Genome Res. 14 (2004) 1188–1190.

[3] B. Thiel, I. Beckmann, P. Kerpedjiev, I. Hofacker, 3D based on 2D: Calculating helix angles and stacking patterns using forgi 2.0, an RNA Python library centered on secondary structure elements. [version 2; peer review: 2 approved], F1000Research. 8 (2019).

Annexes

Code des réseaux de neurones :
https://colab.research.google.com/drive/1yiatNBCJ4r1eXKiHLox0bZQE0NJS3n_o?usp=sharing

Code des graphiques :
https://colab.research.google.com/drive/1LZ2ZK_HhI83N23__rsabnXbppktCq7di?usp=sharing

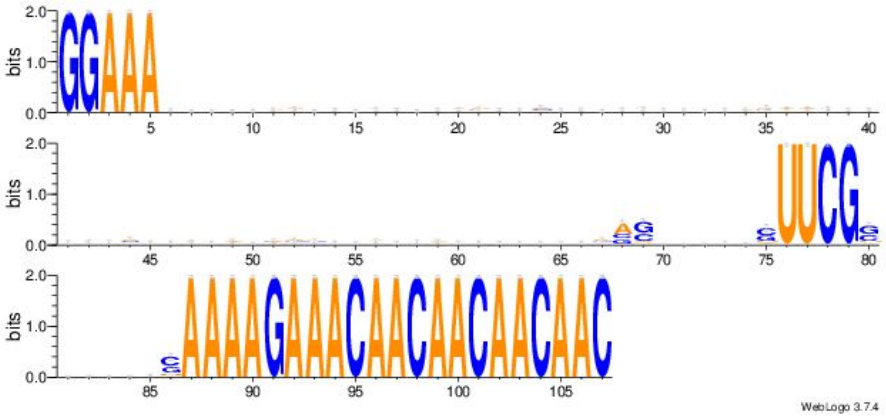


Figure S1 : Logo des séquences de test de 107 pb.

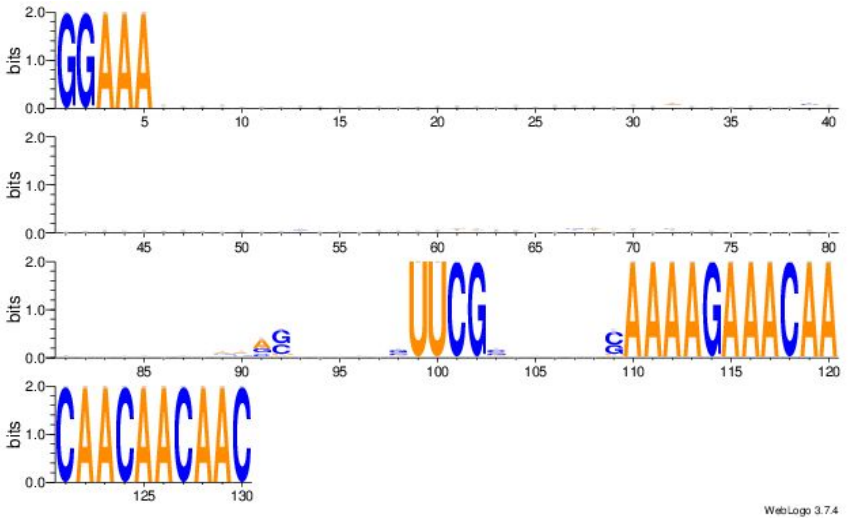


Figure S2 : Logo des séquences de test de 130 pb.

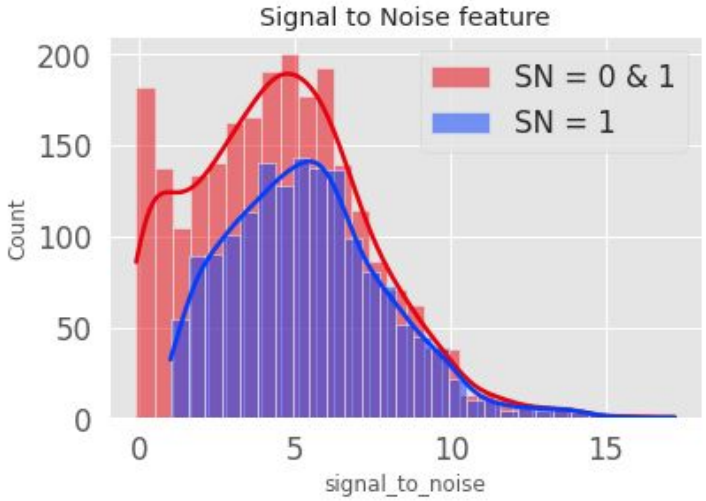


Figure S3 : Distribution du rapport Signal/Bruit avec ou sans filtre

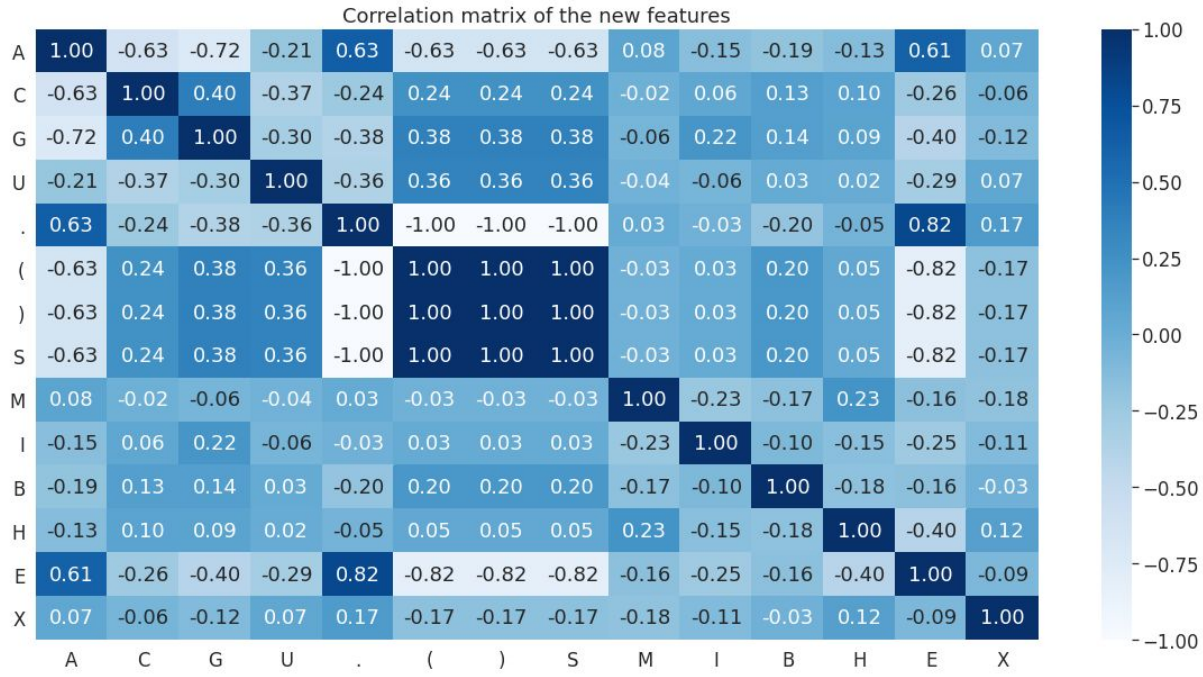


Figure S4 : Matrice de corrélation des différents descripteurs possibles pour l'apprentissage

On remarque des corrélations positives élevées entre L'Adénine, les bases non appariées et les structures de type *dangling end*. A contrario, on retrouve de fortes corrélations négatives entre l'Adénine, les structures en *dangling end* et les bases appariées (Fig. S4). Retirer la mesure des bases appariées (**S**) pourrait être pertinent.

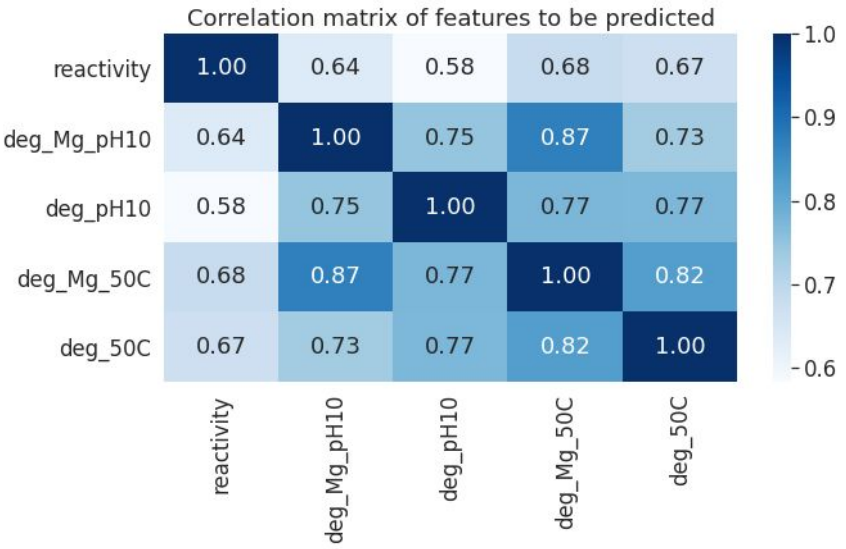


Figure S5 : Matrice de corrélation des descripteurs à prédire

On regarde également les 4 facteurs à prédire et de la même manière, on crée une matrice de corrélation (*Fig. S5*). Elle nous apprend que chaque facteur est fortement corrélé aux autres positivement.

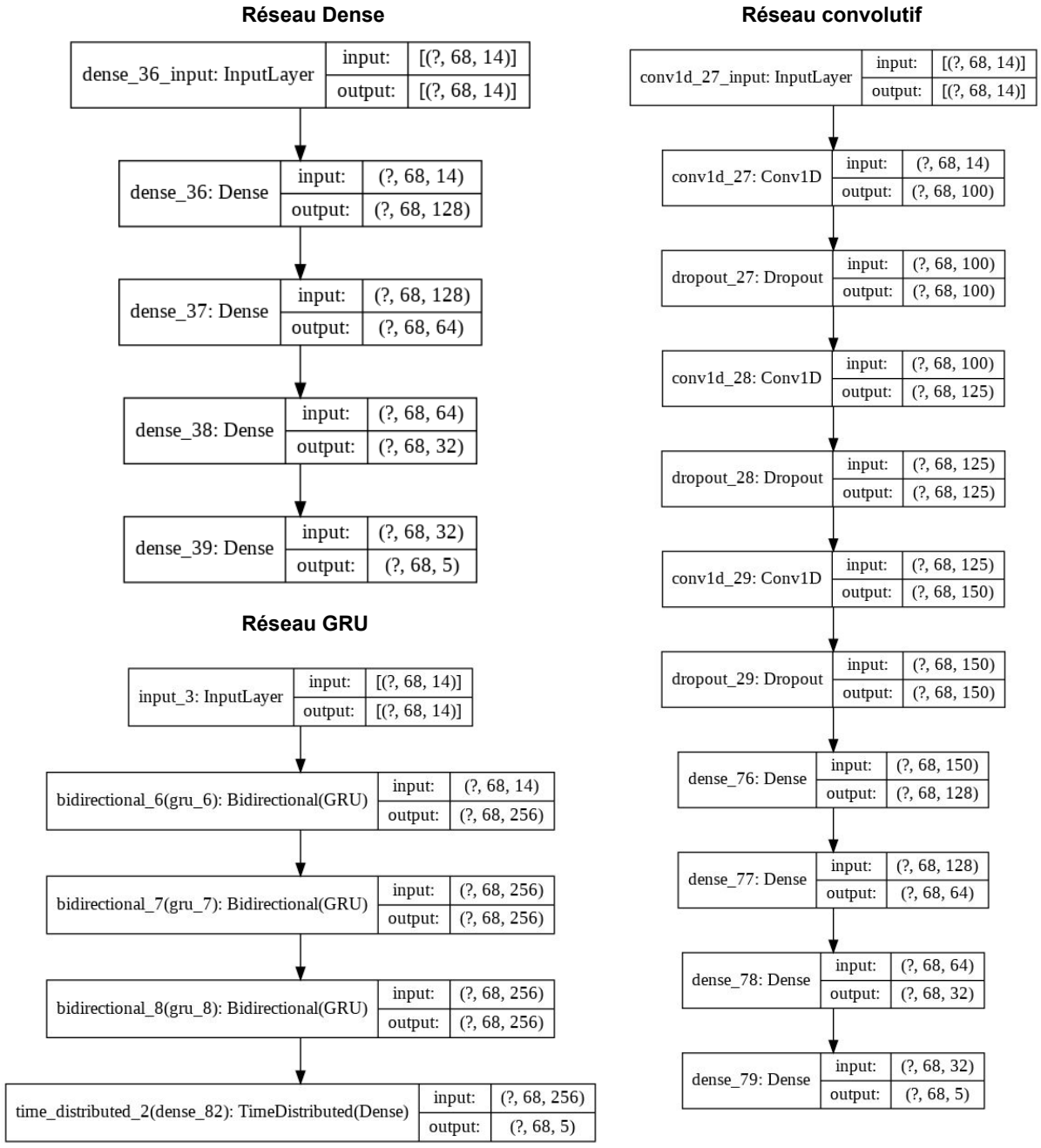


Figure S6 : Architecture dense (en haut à gauche), convolutionnelle (à droite) et GRU (en bas à gauche).