Master M2 - Biologie Informatique
Responsables: Catherine Etchebest & Jean-Christophe Gelly

Rapport

OpenVaccine: COVID-19 mRNA Vaccine Degradation Prediction

Frédéric Guyon & Jean-Christophe Gelly

Hollier Laetitia - Messad Dit Mahtal Lynda - Rambaud Opale

https://github.com/lyndamessad/Covid_vaccin.git

Année : 2020-2021

SOMMAIRE

I. Introduction

Suite à la crise sanitaire actuelle et les difficultés engendrées, les académies mondiales scientifiques sont à la recherche d'un vaccin efficace afin de lutter contre la pandémie mondiale de COVID-19 provoquée par le SARS-CoV-2 qui est une famille de virus à ARN simple brin enveloppés, de sens positif.. Ce virus fait partie d'une famille de virus "Coronaviridae" capable de provoquer des infections respiratoires commençant par un simple rhume allant à des détresses respiratoires aiguës puis dans certains cas au décès. Dans le but de remédier à cette pandémie, les chercheurs se sont intéressés aux vaccins ARNm connus pour être utilisés comme intermédiaire à la synthèse des protéines. Cependant, ces ARNm ne sont pas très stables et il faut les garder à des températures très faibles, d'où la difficulté d'établir un vaccin mondial actuellement. En effet, ces derniers ont tendance à se dégrader et un simple bout coupé aux extrémités de la séquence ARNm peut rendre le vaccin inefficace.

Ce projet a été initié et déposé sur la plateforme Kaggle par la communauté Eterna dirigé par le Professeur Rhiju Das, biochimiste informatique à la Stanford's School de Medicine. **Le but est de prédire les régions des molécules d'ARNm susceptibles de posséder un taux de dégradations élevé selon la position des bases et leur type.**
Pour apporter une réponse à cette problématique nous mettons en place une méthode de Deep Learning de type régression.

Dans le cadre de la prédiction de la fragilité des ARN nous allons effectuer une comparaison de deux méthodes de traitement des données :

- Troncage des données en amont du réseau
- Troncage des données au sein du tenseur

Nous allons ensuite utiliser un réseau récurrent GRU (Gated Recurrent Unit) afin d'effectuer la régression. Nous établirons ensuite laquelle des deux méthodes testées est la plus efficace.

II. Matériel et Méthodes

1. Présentation des jeux de données

Trois fichiers de données sont mis à disposition sur la plateforme Kaggle :

- train.json possédant les données d'entraînement (training data)
- test.json comportant les données de test (training test)
- sample_submission.csv montrant le fichier de sortie au bon format que nous devons obtenir à la fin de notre recherche.

Les données présentes dans les fichiers .json ont été obtenues par les scientifiques de Stanford. Ils avaient en leur possession deux jeux de données : 3029 séquences ARN de type public mesurant 107 bases et 3005 séquences ARN de type privé mesurant 130 bases. La répartition des séquences est détaillée dans la figure 1 ci-dessous.

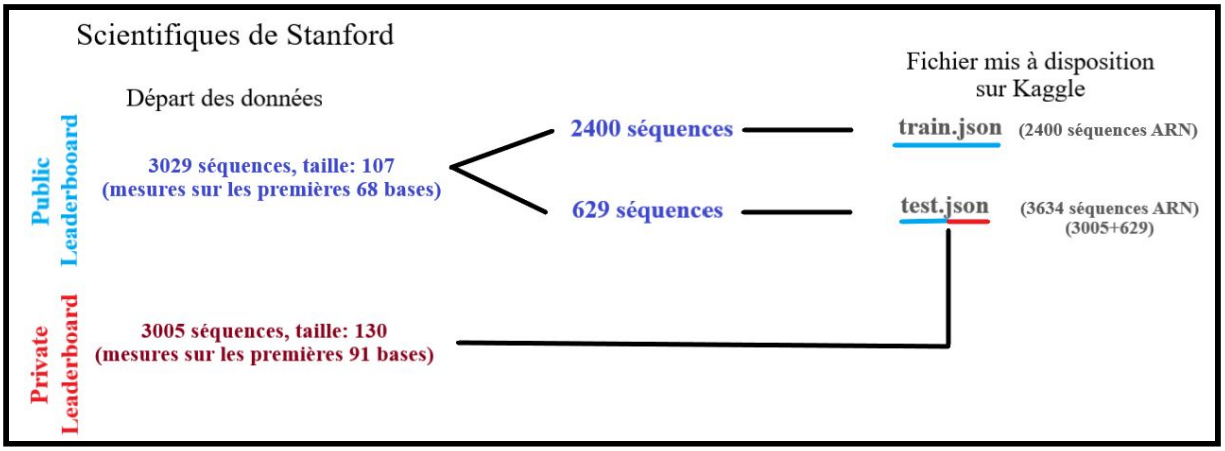

Figure1 – Schéma récapitulatif des données et comment elles ont été obtenues par les scientifiques de l'université de Stanford.

Les jeu de données sont constitués des éléments suivants :

- le fichier ***train.json*** possède ***2400 lignes et 19 colonnes*** **(index, id, sequence, structure, predicted_loop_type, signal_to_noise, SN_filter, seq_length, seq_scored, reactivity_error, deg_error_Mg_pH10, deg_error_pH10, deg_error_Mg_50C, deg_error_50C, reactivity, deg_Mg_pH10, deg_pH10, deg_Mg_50C, deg_50C)**
- le fichier ***test.json*** contient ***3634 lignes pour 7 colonnes*** **(index, id, sequence, structure, predicted_loop_type, seq_length, seq_scored)**

Le descriptif des variables ainsi que leur type ont été détaillés en Annexe 2. Celui-ci nous informe qu'il y a une variable de type float, 4 de type integer et 14 de type object.

2. Vérification des données

Une observation statistique de leur contenu a pu être effectuée (Annexe 3). Ces résultats affirment les constatations faites dans la Figure 1 sur la longueur des séquences: **107 bases dans le Train et Test Public, 130 bases dans le Test Privé**.

De plus, dans la data Train, la longueur des vecteurs pour les colonnes reactivity, deg_error_Mg_pH10, deg_error_pH10, deg_error_Mg_50C, deg_error_50C, deg_Mg_pH10, deg_pH10, deg_Mg_50C, deg_50C est de 68 bases.

Une étape de troncage des données va donc être nécessaire pour les données de train afin de faire correspondre la longueur des séquences avec les vecteurs de données expérimentales.

Ensuite, une vérification de la corrélation des variables numériques a été faite. On conclut que ces variables ne sont pas corrélées. Cependant, on ne va pas s'intéresser à toutes les 19 variables . On procède à une sélection des variables d'intérêt pour l'apprentissages, qui seront donc "***la séquences de l'ARN", "la structure" et la variable 'predicted_loop_type***".

Afin de prédire le taux de dégradation à divers endroits le long de la séquence d'ARN, cinq variables seront à prédire mais seulement trois d'entres elles sont importantes : **reactivity, deg_Mg_pH10, deg_Mg_50C,** deg_pH10, deg_50C.

Comme indiqué dans le descriptif du projet sur Kaggle, la manière dont les 629 séquences de 107 bases de test.json ont été filtrées. Pour assurer la diversité des séquences, les séquences résultantes ont été regroupées en grappes ayant moins de 50 % de similitude, et les 629 séquences de l'ensemble de test ont été choisies parmi des grappes de 3 membres ou moins. Autrement dit, toute séquence de l'ensemble de test doit être similaire à deux autres séquences au maximum. Ceci a été vérifié par un alignement multiple(CF. Annexe 4). En effet, les pourcentage de similitude des séquences est trés faibles. Ensuite, on a donc appliqué ce principe sur les données Train en faisant un CLUSTAL Omega. Ceci va nous permettre d'avoir une idée sur l'alignement des séquences d'apprentissage. Le résultat d'alignement est présenté en annexe 4. Ceci nous a permis de voir le degré de similitude entre les différentes séquences. On trouve donc un pourcentage d'identité qui va de 6.5% à 74% de similarité entre différents alignements possible.

3. Préparation des données (pré-traitement)

a. Débruitage

Dans le jeu de données la variable signal_to_noise indique le bruit moyen. Afin de ne sélectionner que les séquence de bonne qualité elles seront filtrées par cette variables afin de ne garder que celles dont le rapport signal sur bruit est supérieur à 1. Nous obtenons donc un jeu de données de train de dimension 2097 lignes pour 19 descripteurs.

b. Formatage des données

Il est important par ailleurs de transmettre des données conformes afin que les valeurs des colonnes servant à l'apprentissage puissent être apprises par le réseau. Pour ce faire, un dictionnaire de mapping (token2int) ainsi que 2 fonctions (pandas_list_to_array et preprocess_inputs) ont été créés. Le dictionnaire affecte un integer à chaque caractère présents dans les colonnes input (token2int = {x:i for i, x in enumerate('().ACGUBEHIMSX')}) ce qui donne comme dictionnaire:
{'(': 0, ')': 1, '.': 2, 'A': 3, 'C': 4, 'G': 5, 'U': 6, 'B': 7, 'E': 8, 'H': 9, 'I': 10, 'M': 11, 'S': 12, 'X': 13}.

La fonction pandas_list_to_array convertit la dataframe (df) de dimensions (x, y) possédant des listes de taille l en array de dimension (x, l, y) lisible par le réseau.

La fonction preprocess_inputs prend en paramètres la dataframe df, le dictionnaire token2int et le nom des colonnes input (sequence, structure, predicted_loop_type) et retourne un array mappé en utilisant la 1ère fonction.

Les données de train contiennent uniquement les colonnes suivantes : sequence, structure et predicted_loop car ce sont les features permettant de prédire les variables d'intérêt (reactivity, deg_Mg_pH10, deg_Mg_50C).

c. Troncage des données

Selon la méthode utilisée, le troncage va s'effectuer en amont du réseau ou au sein de celui-ci selon le schéma suivant :

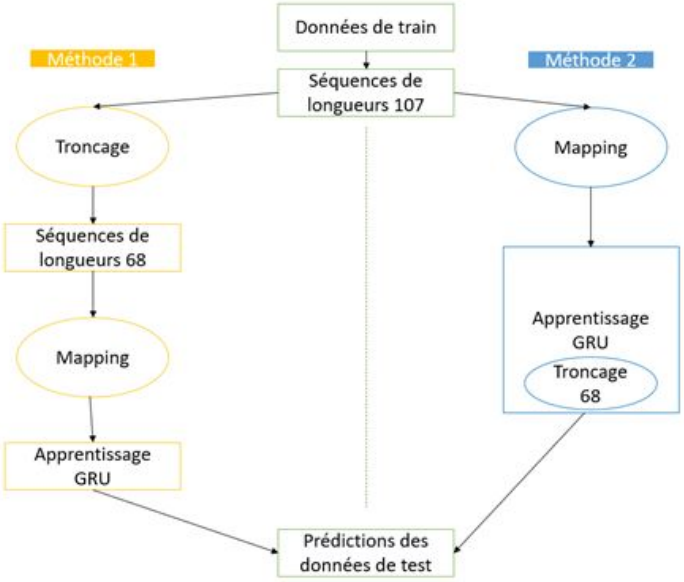

Figure 2– Schéma récapitulatif des deux méthodes de troncage utilisées.

4. Design des réseaux

En premier lieu, nous avons testé un réseau simple constitué d'une succession de couches dense. Celui-ci a présenter des performances de loss et d'accuracy plutôt médiocres. C'est pourquoi nous avons choisi de mettre en place un réseau récurrent GRU (Gated Recurrent Unit).

Un réseau de neurones récurrents est constitué de neurones interconnectées interagissant non-linéairement et pour lequel il existe au moins un cycle dans la structure :

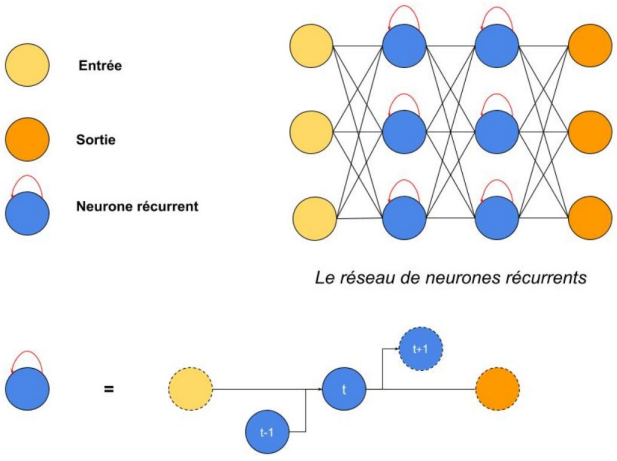

Figure 3 – Schéma récapitulatif du principe général des réseaux récurrents.

Les GRU sont des réseaux récurrents se basant sur un système de portes. Les portes sont des zones de calculs qui régulent le flot d'informations. Les GRU ne possèdent que deux portes : Reset Gate et Update Gate.

Reset Gate sert à contrôler combien d'information passée le réseau doit oublier. L'état caché précédent, concaténé avec les données d'entrée, passe par une sigmoïde (pour ne conserver que les coordonnées pertinentes) puis est multiplié par l'ancien état caché : on n'en

conserve donc que les coordonnées importantes (telles qu'elles) de l'état précédent (on a donc perdu une partie de l'état précédent dans cette porte).

Update Gate décide des informations à conserver et de celles à oublier. Les données d'entrées et l'ancien état caché sont concaténés et passent par une fonction sigmoïde dont le rôle est de déterminer quelles sont les composantes importantes.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 68, 3)]           0
_________________________________________________________________
embedding (Embedding)        (None, 68, 3, 100)        1400
_________________________________________________________________
tf_op_layer_Reshape (TensorF [(None, 68, 300)]         0
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 68, 300)           0
_________________________________________________________________
bidirectional (Bidirectional (None, 68, 300)           406800
_________________________________________________________________
bidirectional_1 (Bidirection (None, 68, 300)           406800
_________________________________________________________________
bidirectional_2 (Bidirection (None, 68, 300)           406800
_________________________________________________________________
tf_op_layer_strided_slice (T [(None, 68, 300)]         0
_________________________________________________________________
dense_3 (Dense)              (None, 68, 5)             1505
=================================================================
Total params: 1,223,305
Trainable params: 1,223,305
Non-trainable params: 0
```

A

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 107, 3)]          0
_________________________________________________________________
embedding_1 (Embedding)      (None, 107, 3, 100)       1400
_________________________________________________________________
tf_op_layer_Reshape_1 (Tenso [(None, 107, 300)]        0
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 107, 300)          0
_________________________________________________________________
bidirectional (Bidirectional (None, 107, 300)          406800
_________________________________________________________________
bidirectional_1 (Bidirection (None, 107, 300)          406800
_________________________________________________________________
bidirectional_2 (Bidirection (None, 107, 300)          406800
_________________________________________________________________
tf_op_layer_strided_slice (T [(None, 68, 300)]         0
_________________________________________________________________
dense (Dense)                (None, 68, 5)             1505
=================================================================
Total params: 1,223,305
Trainable params: 1,223,305
Non-trainable params: 0
```

B

Figure 4 – design des deux réseaux utilisés :non tronqué(A) et tronqué(B).

On voit bien que le premier réseau prend en entrée des séquences déjà tronquées de longueur 68 alors que le deuxième réseau prend en entrée une séquence de 107 puis effectue une couche de troncage qui réduit les données à une taille de 68.

La loss fonction utilisée est la **MCRMSE** (Mean Columnwise Root Mean SquareError). Il s'agit d'une moyenne des RMSE (Root Mean Square Error) de chaque colonnes prédites.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

L'**erreur** calculée correspond à une distance (normalisée) entre le vecteur des valeurs prédites et le vecteur des valeurs observées. C'est un metric très utilisé dans l'étude des performances des réseaux de neurones. L'**accuracy** est aussi calculée pour évaluer le réseau.

III. Analyse des deux méthodes

Afin d'effectuer une comparaison des deux méthodes, leur performances vont être étudiées.

➢ ***Méthode 1 :***

Voici les performances générales obtenues lors de l'évaluation du modèle:

[loss = 0.1820251897213951, acc = 0.55021006]

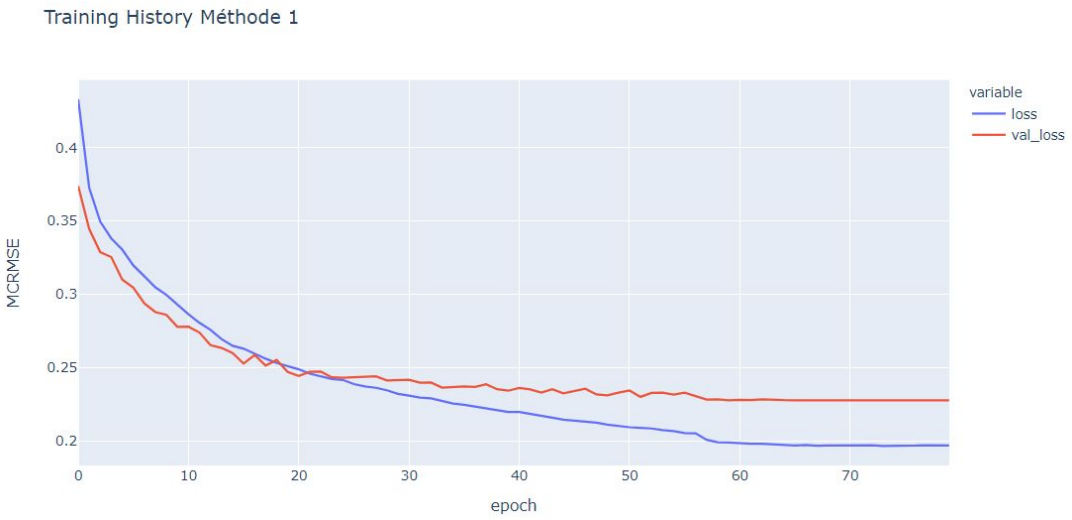

Figure 5 – Graphique représentant l'évolution de la Loss en fonction des epochs pour la méthode 1.

Dans la figure 5, on observe qu'à partir de 60 epoch, la loss se stabilise autour de 0.2. Cela veut dire que c'est la loss minimale que notre réseau peut atteindre. On voit que la loss du train et de la validation semblent diminuer de la même manière. Cela montre que nous n'avons pas de sur-apprentissage.

Deuxièmement, le graphique de la figure 6 représentel'évolution de l'accuracy au cours des epoch.

On observe que l'accuracy stagne à partir de 60 epoch autour de 0,52.

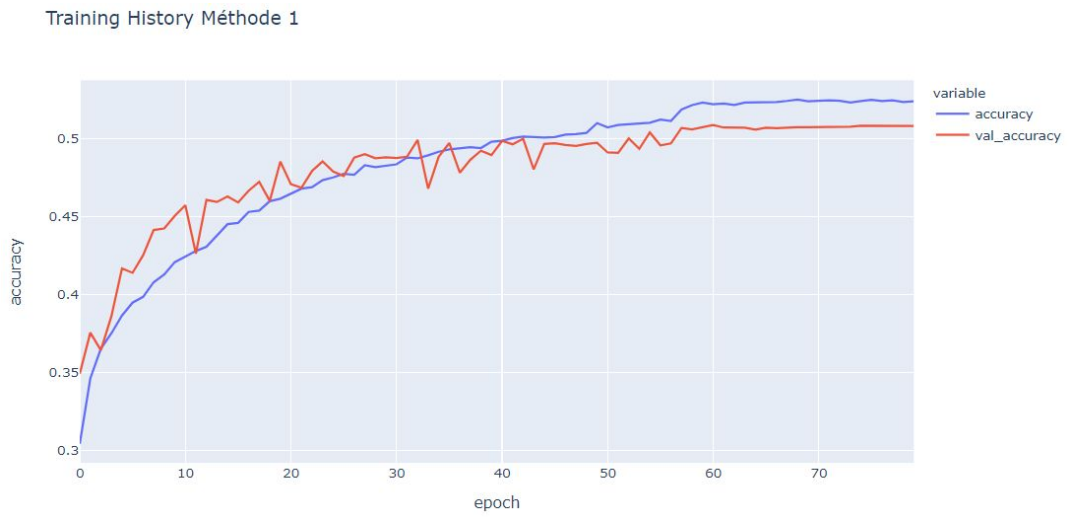

Figure 6 – Graphique représentant l'évolution de l'Accuracy en fonction des epochs de la méthode 1.

Enfin, voici les scores obtenus sur Kaggle pour la prédictions des données de test :

Submission and Description	Private Score	Public Score	Use for Final Score
submission2.csv a few seconds ago by Opale Rambaud obtain with gru and truncate data before the network	0.38704	0.27812	☐

En conclusion pour cette méthode, on peut dire que les performances obtenues sont plutôt moyennes. Le jeu de test "Private" obtient un meilleur résultat que le jeu "Public"

- ***Méthode 2 :***

Nous avons générés les même résultats pour la méthode consistant à tronquer les données dans le réseau.

Performance sur l'évaluation : ***[loss = 0.1838255857913956, acc = 0.54921024]***

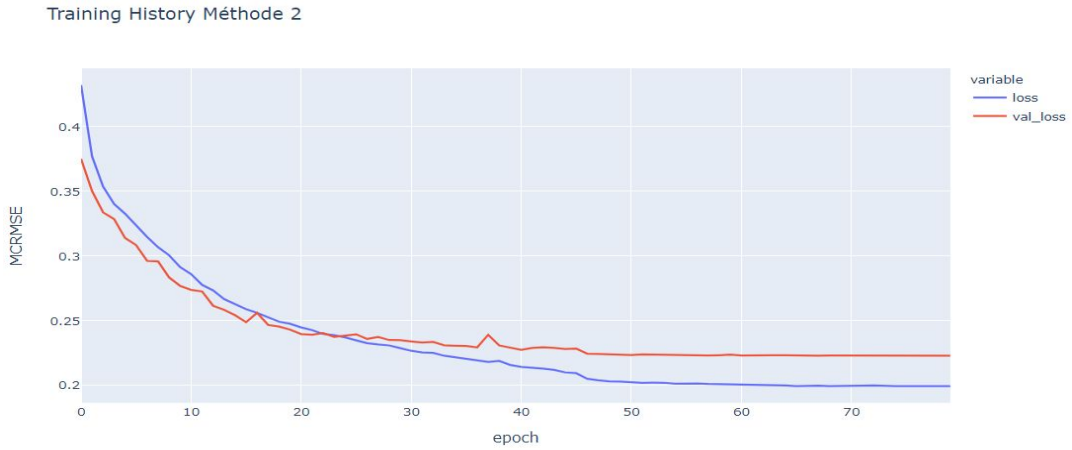

Figure 7 – Graphique représentant l'évolution de la Loss en fonction des epochs de la méthode 2.

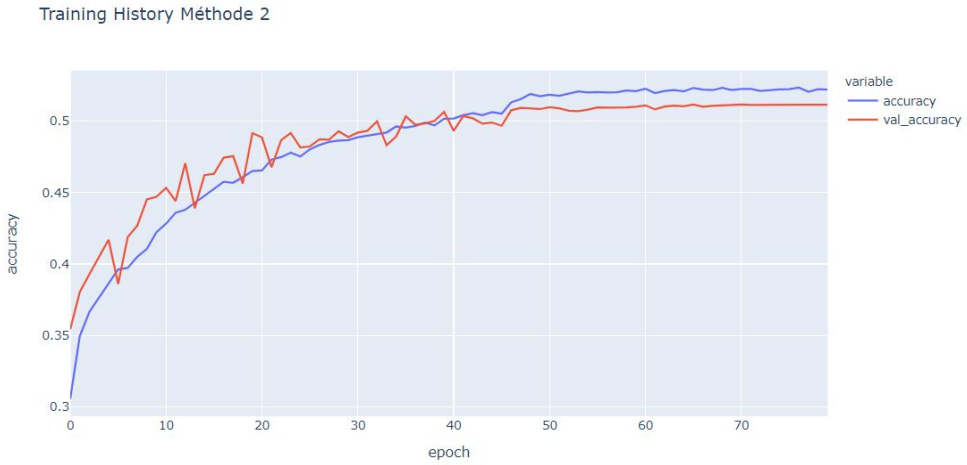

Figure 8 – Graphique représentant l'évolution de l'Accuracy en fonction des epochs de la méthode 8.

Submission and Description	Private Score	Public Score	Use for Final Score
submission.csv just now by Opale Rambaud Obtain with truncate data inside the network	0.38772	0.27014	☐

On observe des résultats très similaires. Les valeurs de loss et d'accuracy sont très proches. Elles se stabilisent autour de 50 epochs. Les scores obtenus sur Kaggle sont extrêmement similaires.

IV. Conclusion

On observe donc que les deux méthodes ne permettent pas d'obtenir des résultats significativement différents. Le fait de tronquer les données en amont ou au sein du tenseur ne change pas les performances du réseaux.
Il existe une troisième méthode consistant à créer une fenêtre glissante de taille 68 sur la séquence afin de ne prédire que le nucléotide central. Il serait alors intéressant de comparer cette méthode avec les deux première pour éventuellement améliorer les résultats de prédictions.

ANNEXES

Annexe 1 – Visualisation des data Test et Train

⇒ Aperçu des données Test

```
test.head()
```

	index	id	sequence	structure	predicted_loop_type	seq_length	seq_scored
0	0	id_00073f8be	GGAAAAGUACGACUUGAGUACGGAAAACGUACCAACUCGAUUAAAA...	.....((((((((((.(((((.....)))))))))(((((((.....	EEEEEESSSSSSSSSSBSSSSSHHHHHSSSSSSSSSSSSSSSHHH...	107	68
1	1	id_000ae4237	GGAAACGGGUUCCGCGGAUUGCUGCUAAUAAGAGUAAUCUCUAAAU...	.....((((..(((((((...(((((.....(((((....))))....	EEEEESSSSIISSSSSSIIISSSSSIIIIISSSSHHHHSSSSIIII...	130	91
2	2	id_00131c573	GGAAAACAAAACGGCCUGGAAGACGAAGGAAUUCGGCGCGAAGGCC...	...........((.(((.(.(..((..((((...)))).)).))...	EEEEEEEEEEESSISSSISIISSSIISSSSHHHHSSSSIISS...	107	68
3	3	id_00181fd34	GGAAAGGAUCUCUAUCGAAGGAUAGAGAUCGCUCGCGACGGCACGA...	.....((((((((((....))))))))))(((((((..((.(((.....	EEEEEESSSSSSSSSSHHHHSSSSSSSSSSSSSSSIISSISSSHH...	107	68
4	4	id_0020473f7	GGAAACCCGCCCGCGCCCGCCCGCGCUGCUGCCGUGCCUCCUCUCC...	.....((((((((((((((((((((((((((((((((((((((...	EEEEESSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS...	130	91

⇒ Aperçu des données Train

```
train.head()
```

	index	id	sequence	structure	predicted_loop_type	signal_to_noise	SN_filter	seq_length	seq_scored
0	0	id_001f94081	GGAAAAGCUCUAAUAACAGGAGACUAGGACUACGUAUUUCUAGGUA...	.....((((((.....)))))).)).((.....((..((((((.....	EEEEESSSSSSHHHHHHHSSSSBSSXSSIIIIISSIISSSSSSHHH...	6.894	1	107	68
1	1	id_0049f53ba	GGAAAAAGCGCGCGCGGUUAGCGCGCGCUUUUGCGCGCGCUGUACC...	..... (((((((((((((((((((....)))))))))).)))...	EEEEESSSSSSSSSSSSSSSSSSSSSSSSSHHHHSSSSSSSSSBSSS...	0.193	0	107	68
2	2	id_006f36f57	GGAAAGUGCUCAGAUAAGCUAAGCUCGAAUAGCAAUCGAAUAGAAU...	.....(((((.((.....(((((.(((.....))))..(((((.....)...	EEEEESSSSISSIIIIISSSSMSSSHHHHHSSSMMSSSSHHHHHHS...	8.800	1	107	68
3	3	id_0082d463b	GGAAAAGCGCGCGCGCGCGCGCGAAAAAGCGCGCGCGCGCGCGCGC...	.....(((((((((((((((.....))))))))))))))) ((...	EEEEEESSSSSSSSSSSSSSSSSSHHHHHHSSSSSSSSSSSSSSSSS...	0.104	0	107	68
4	4	id_0087940f4	GGAAAAUAUAUAAUAUAUUAUAUAAAUAUAUUAUAGAAGUAUAAUA...	.....(((((((.(((((((((((. (((((((((....)))))))))....	EEEEESSSSSSSBSSSSSSSSSSSSBSSSSSSSSSHHHHSSSSSSS...	0.423	0	107	68

reactivity_error	deg_error_Mg_pH10	deg_error_pH10	deg_error_Mg_50C	deg_error_50C	reactivity	deg_Mg_pH10	deg_pH10	deg_Mg_50C	deg_50C
[0.1359, 0.20700000000000002, 0.1633, 0.1452, ...	[0.26130000000000003, 0.38420000000000004, 0.1...	[0.2631, 0.28600000000000003, 0.0964, 0.1574, ...	[0.1501, 0.275, 0.0947, 0.18660000000000002, 0...	[0.2167, 0.34750000000000003, 0.188, 0.2124, 0...	[0.3297, 1.5693000000000001, 1.1227, 0.8686, 0...	[0.7556, 2.983, 0.2526, 1.3789, 0.637600000000...	[2.3375, 3.5060000000000002, 0.3008, 1.0108, 0...	[0.35810000000000003, 2.9683, 0.2589, 1.4552, ...	[0.6382, 3.4773, 0.9988, 1.3228, 0.78770000000...
[2.8272, 2.8272, 2.8272, 4.7343, 2.5676, 2.567...	[73705.3985, 73705.3985, 73705.3985, 73705.398...	[10.1986, 9.2418, 5.0933, 5.0933, 5.0933, 5.09...	[16.6174, 13.868, 8.1968, 8.1968, 8.1968, 8.19...	[15.4857, 7.9596, 13.3957, 5.8777, 5.8777, 5.8...	[0.0, 0.0, 0.0, 2.2965, 0.0, 0.0, 0.0, 0.0, 0....	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[4.947, 4.4523, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[4.8511, 4.0426, 0.0, 0.0, 0.0, 0.0, 0.0,...	[7.6692, 0.0, 10.9561, 0.0, 0.0, 0.0, 0.0, 0.0...
[0.0931, 0.13290000000000002, 0.11280000000000...	[0.1365, 0.2237, 0.1812, 0.1333, 0.1148, 0.160...	[0.17020000000000002, 0.178, 0.111, 0.091, 0.0...	[0.1033, 0.1464, 0.1126, 0.09620000000000001, ...	[0.14980000000000002, 0.1761, 0.1517, 0.116700...	[0.44820000000000004, 1.4822, 1.1819, 0.743400...	[0.2504, 1.4021, 0.9804, 0.49670000000000003, ...	[2.243, 2.9361, 1.0553, 0.721, 0.6396000000000...	[0.5163, 1.6823000000000001, 1.0426, 0.7902, 0...	[0.9501000000000001, 1.7974999999999999, 1.499...
[3.5229, 6.0748, 3.0374, 3.0374, 3.0374, 3.037...	[73705.3985, 73705.3985, 73705.3985, 73705.398...	[11.8007, 12.7566, 5.7733, 5.7733, 5.7733, 5.7...	[121286.7181, 121286.7182, 121286.7181, 121286...	[15.3995, 8.1124, 7.7824, 7.7824, 7.7824, 7.78...	[0.0, 2.2399, 0.0, 0.0, 0.0, 0.0, 0.0, 0....	[0.0, -0.5083, 0.0, 0.0, 0.0, 0.0, 0.0, 0...	[3.4248, 6.8128, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...	[0.0, -0.8365, 0.0, 0.0, 0.0, 0.0, 0.0, 0...	[7.6692, -1.3223, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...
[1.665, 2.1728, 2.0041, 1.2405, 0.620200000000...	[4.2139, 3.9637000000000002, 3.2467, 2.4716, 1...	[3.0942, 3.015, 2.1212, 2.0552, 0.881500000000...	[2.6717, 2.4818, 1.9919, 2.5484999999999998, 1...	[1.3285, 3.6173, 1.3057, 1.3021, 1.1507, 1.150...	[0.8267, 2.6577, 2.8481, 0.40090000000000003, ...	[2.1058, 3.138, 2.5437000000000003, 1.0932, 0....	[4.7366, 4.6243, 1.2068, 1.1538, 0.0, 0.0, 0.7...	[2.2052, 1.7947000000000002, 0.7457, 3.1233, 0...	[0.0, 5.1198, -0.3551, -0.3518, 0.0, 0.0, 0.0,...

Annexe 2 - Noms des variables (descripteurs) et leur type

- **id** - An arbitrary **identifier** for each sample.
- **seq_scored** - (**68 in Train and Public Test, 91 in Private Test**) **Integer value** denoting the number of positions used in scoring with predicted values. This should match the length of **reactivity**, **deg_*** and ***_error_*** columns. Note that ***molecules used for the Private Test will be longer than those in the Train and Public Test data***, so the size of this vector will be different.
- **seq_length** - (**107 in Train and Public Test, 130 in Private Test**) **Integer values**, denotes the length of **sequence**. Note that molecules used for the Private Test will be longer than those in the Train and Public Test data, so the size of this vector will be different.
- **sequence** - (**1x107 string** in **Train and Public Test, 130 in Private Test**) Describes the RNA sequence, a combination of A, G, U, and C for each sample. ***Should be 107 characters long, and the first 68 bases should correspond to the 68 positions specified in*** **seq_scored** (note: indexed starting at 0).
- **structure** - (**1x107 string** in **Train and Public Test, 130 in Private Test**) An array of (,), and . characters that describe whether a base is estimated to be paired or unpaired. Paired bases are denoted by opening and closing parentheses e.g. (....) means that base 0 is paired to base 5, and bases 1-4 are unpaired.
- **reactivity** - (**1x68 vector** in Train **and Public Test, 1x91 in Private Test**) An array of floating point numbers, should have the same length as **seq_scored**. These numbers are reactivity values for the first 68 bases as denoted in **sequence**, and used to determine the likely secondary structure of the RNA sample.
- **deg_pH10** - (**1x68 vector** in **Train and Public Test, 1x91 in Private Test**) An array of floating point numbers, should have the same length as **seq_scored**. These numbers are reactivity values for the first 68 bases as denoted in **sequence**, and used to determine the likelihood of degradation at the base/linkage after incubating without magnesium at high pH (pH 10).
- **deg_Mg_pH10** - (**1x68 vector** in **Train and Public Test, 1x91 in Private Test**) An array of floating point numbers, should have the same length as **seq_scored**. These numbers are reactivity values for the first 68 bases as denoted in **sequence**, and used to determine the likelihood of degradation at the base/linkage after incubating with magnesium in high pH (pH 10).
- **deg_50C** - (**1x68 vector** in **Train and Public Test, 1x91 in Private Test**) An array of floating point numbers, should have the same length as **seq_scored**. These numbers are reactivity values for the first 68 bases as denoted in **sequence**, and used to determine the likelihood of degradation at the base/linkage after incubating without magnesium at high temperature (50 degrees Celsius).
- **deg_Mg_50C** - (**1x68 vector** in **Train and Public Test, 1x91 in Private Test**) An array of floating point numbers, should have the same length as **seq_scored**. These numbers are reactivity values for the first 68 bases as denoted in **sequence**, and used to determine the likelihood of degradation at the base/linkage after incubating with magnesium at high temperature (50 degrees Celsius).
- ***_error_*** - **An array of floating point numbers**, should have the same length as the corresponding **reactivity** or **deg_*** columns, calculated errors in experimental values obtained in **reactivity** and **deg_*** columns.
- **predicted_loop_type** - (**1x107 string**) Describes the structural context (also referred to as 'loop type')of each character in **sequence**. Loop types assigned by bpRNA from Vienna RNAfold 2 structure. From the bpRNA_documentation: **S: paired "Stem" M: Multiloop I: Internal loop B: Bulge H: Hairpin loop E: dangling End X: eXternal loop**
  - **S/N filter** Indicates if the sample passed filters

Annexe 3 – Observation statistiques des données issues des fichiers test.json (data Test) et train.json (data Train)

Descriptif de la data Test

	index	seq_length	seq_scored
count	3634.000000	3634.000000	3634.000000
mean	1816.500000	126.018987	87.018987
std	1049.189767	8.702624	8.702624
min	0.000000	107.000000	68.000000
25%	908.250000	130.000000	91.000000
50%	1816.500000	130.000000	91.000000
75%	2724.750000	130.000000	91.000000
max	3633.000000	130.000000	91.000000

Descriptif de la data Train

	index	signal_to_noise	SN_filter	seq_length	seq_scored
count	2400.000000	2400.000000	2400.000000	2400.0	2400.0
mean	1199.500000	4.530456	0.662083	107.0	68.0
std	692.964646	2.835142	0.473099	0.0	0.0
min	0.000000	-0.103000	0.000000	107.0	68.0
25%	599.750000	2.391000	0.000000	107.0	68.0
50%	1199.500000	4.442500	1.000000	107.0	68.0
75%	1799.250000	6.294250	1.000000	107.0	68.0
max	2399.000000	17.194000	1.000000	107.0	68.0

Annexe 4 – Observation des alignements multiples des séquences test (figure 1) et séquences train (figure 2) obtenue par CLUSTAL Omega.

```
sequence1260      --GGA---A-AA-------------------------------------------------  6
sequence2070      --GGA---A-AA-------------------------------------------------  6
sequence1145      --GGA---A-AGCGG-A----------------------A--------------------  11
sequence819       --GGA---A-A--------------------------------------------------  5
sequence335       --GGA---A-ACUCG-A----------------------A----AU-A-A----------  15
sequence3415      --GGA---A----------------------------------------------------  4
sequence745       --GGA---A-AAC------------------------------------------------  7
sequence2965      --GG---------------------------------------------------------  2
sequence639       --GGA---A-ACAAA----------------------------------------------  9
sequence1309      --GGA---A-ACAAC-A----------------------A----AC-A-A----------  15
sequence2894      --GGA---A-AAACA-A----------------------A----AG-A-A----------  15
sequence1572      --GG---------------------------------------------------------  2
sequence1258      --GGA---A-ACCGU-U----------------------A----AC-C-AA---------  16
sequence645       --GG---------------------------------------------------------  2
```

Figure1: Alignement sequences test.json par CLUSTAL Omega(lien vers l'alignement: https://www.ebi.ac.uk/Tools/services/web/toolresult.ebi?jobId=clustalo-I20201027-013613-0487-41933118-p1m)

```
sequence1908      -----C-AA-G------------------G------AU-CG-GG----UGGGA-------  45
sequence165       -----C-UG-G------------------G------AU-GA-GG----UGCGG-------  45
sequence1430      -----G-UG-G------------------C------AU-GC-GA----AGUGG-------  45
sequence325       -----G-UC-G------------------G------AG-GA-GG----AGGAA-------  45
sequence740       -----G-UC-G------------------G------AG-GA-GG----AGGAA-------  45
sequence612       -----G-UC-G------------------C------AU-GA-GG----UGCGA-------  45
sequence1500      -----G-UC-G------------------G------AG-GA-GG----AGGAA-------  45
sequence964       ----AG-CG-GA-C-G-AAU-AUA-CG-UU---CGCAU-AG-CG----AUCGA-------  58
sequence1378      ----AG-CG-AG-C-U--CG-ACG-CG-UG---CACAU-AA-UG----CACGC-------  48
sequence807       --------A-GA-G-C-ACC-CGC-CG-AU---CGUAC-AG-CC----GACGA-------  58
sequence554       ----GG-AG-AA-G-U-CAU-AGU-CA-GA---CGCGG-AG-UG----GAAGU-------  58
sequence1713      ----GC-AG-AA-G-U-GAU-AGU-UG-GA---CGCGG-AG-UG----GAAGU-------  58

sequence1119      UA-A---AGA------CAAG----A-CA------AGA--U-AG-AAACC-AUA-AUUUCG  79
sequence976       UA-U---ACA------CAAC----A-CA------AGA--U-AG-AAACG-UGA-UAUUCG  79
sequence1293      UA-U---ACA------CAAC----A-CA------AGA--U-AG-UACCA-UAA-UGUUCG  79
sequence491       UA-U---ACA------CAAC----A-CA------AGA--U-AG-UACCU-UGA-GGUUCG  79
```

Figure2: Alignement sequences train.json (filtré des denoise <1) par CLUSTAL Omega(lien vers l'alignement: https://www.ebi.ac.uk/Tools/services/web/toolresult.ebi?jobId=clustalo-I20201026-194035-0805-86063053-p1m&analysis=alignments)