Option Professionnelle Recherche et Développement
Rapport de projet

Identification des paramètres d'une classe de réseaux de régulation de gènes hybrides

Élèves :
Lili Crétaigne
Gatien Chopard

Équipe encadrante :
Honglu Sun
Morgan Magnin

4 avril 2023

Table des matières

1 Introduction

L'expression des gènes et la production des protéines associées ne sont pas des processus individuels. Autrement dit, l'expression d'un gène peut activer ou inhiber les expressions des autres gènes. Toutes ces interactions entre les gènes définissent ce qu'on appelle des réseaux de régulation de gènes.

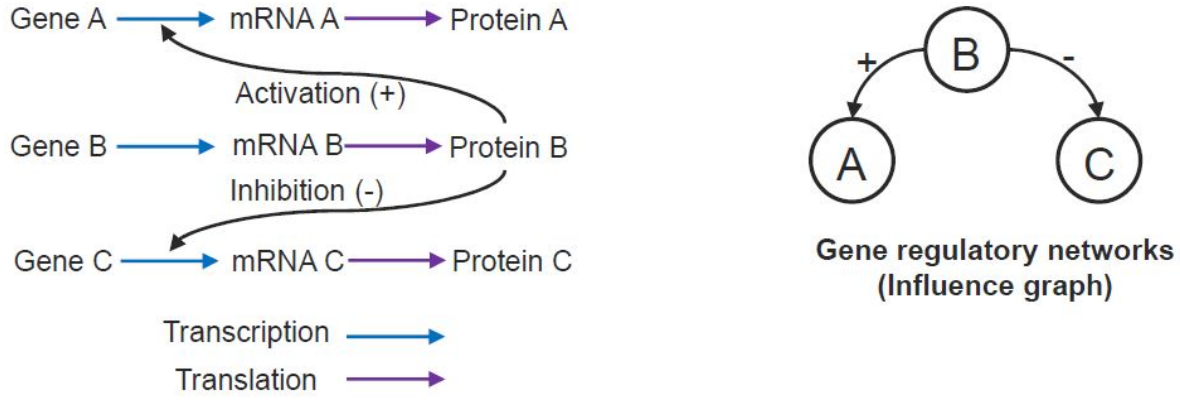


FIGURE 1 – Un réseau de régulation de gènes et son graphe d'influence associé

Les réseaux de régulation de gènes peuvent être représentés sous la forme de graphes (figure 1), où les états sont les couples (gène, protéine) et les transitions sont l'action du gène de l'état de départ sur l'expression du gène de l'état d'arrivée (+ pour activation, - pour inhibition). Pour comprendre la dynamique d'un réseau de régulation de gènes, plusieurs questions peuvent être posées, telles que :

— Combien existe t-il d'états stables dans le système ?
— Est-ce qu'il existe des oscillations entre plusieurs états dans le système ?

Pour répondre à ces interrogations, une approche consiste à modéliser les réseaux de régulation de gènes par des modèles mathématiques et ensuite analyser les modèles mathématiques pour mieux comprendre les systèmes biologiques.
Dans la littérature, plusieurs types de modèles ont été utilisés pour modéliser les réseaux de régulation de gènes, par exemple les équations différentielles, réseaux booléens, etc.

Dans ce projet, nous allons utiliser un modèle hybride, qui comporte en même temps des composantes discretes et des composantes continues (figure 2) :

— D'une part, on définit pour chaque gène un seuil θ correspondant à la concentration en protéine nécéssaire à l'action de ce gène sur un autre gène du réseau. Par exemple, dans le réseau de la figure 2, le gène B inhibe le gène A dès que la concentration en protéine B, notée x_b, dépasse le seuil θ_{ba}. On obtient alors 4 états discrets correspondant à toutes les combinaisons possible d'activité des gènes en fonction de la valeur des seuils. Par exemple, dans l'état 01, le gène A n'agit pas sur le gène B, mais le gène B inhibe le gène A.
— D'autre part, l'évolution de la concentration est modélisée de manière continue avec une vitesse qui dépend de l'état discret dans lequel le système se trouve.

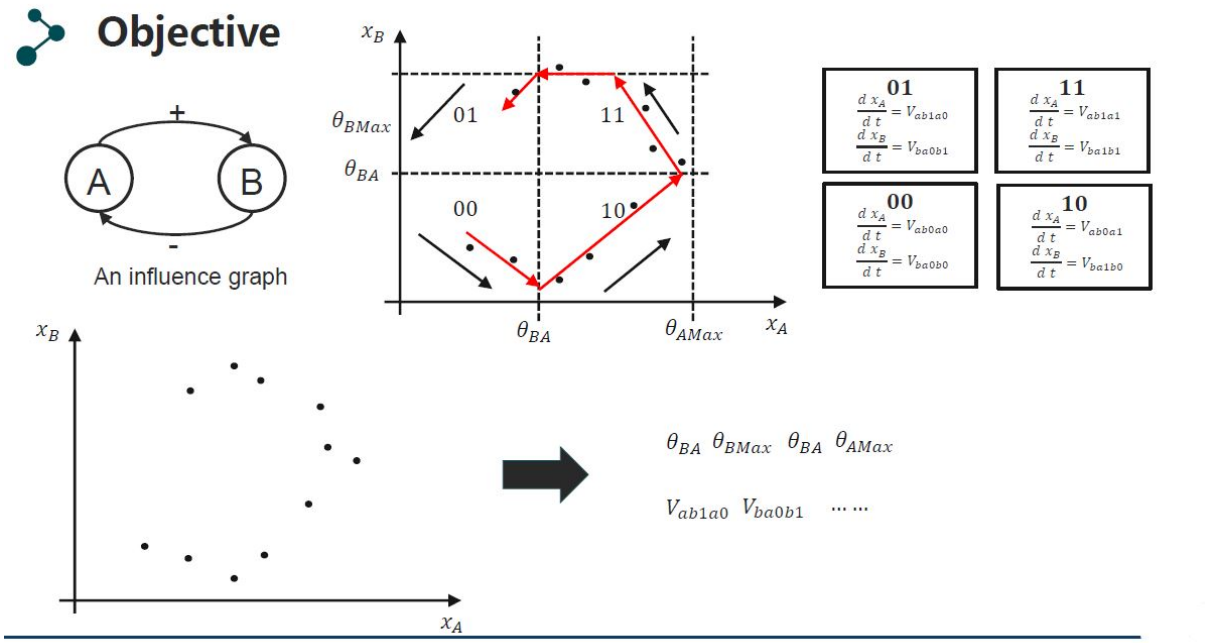
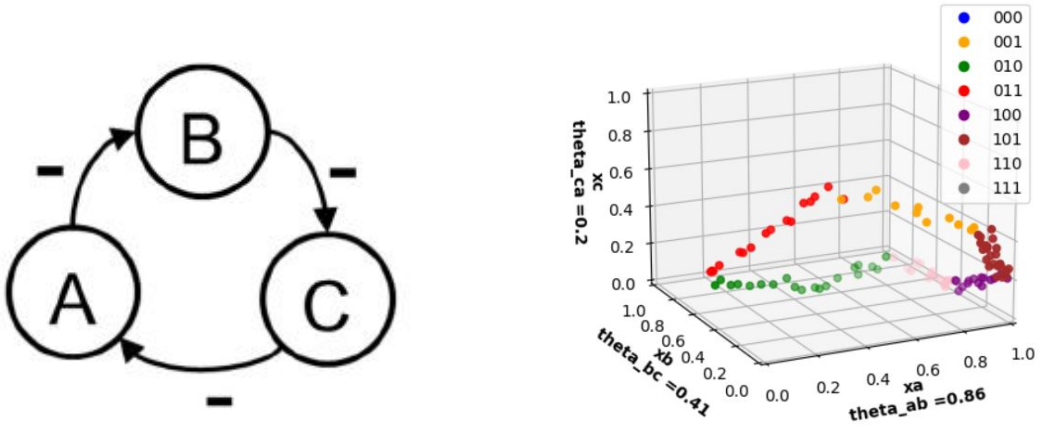


FIGURE 2 – Modèle hybride d'un réseau de régulation de gène

Ce modèle hybride fait l'objet d'études depuis plusieurs années dans plusieurs équipes de recherche en France, dont l'équipe MeForBio du LS2N (Laboratoire des Sciences du Numérique de Nantes).
Il existe d'autres possibilités pour modéliser les réseaux de régulation de gène, par exemple les équations différentielles. Dans ce cas, les paramètres du modèle sont plus difficiles à estimer, surtout lorsque les équations ne sont pas linéaires.
Une modélisation entièrement discrète est également possible, chaque état discret correspondant à un intervalle pour la concentration. Cette modélisation réduit considérablement le nombre de paramètres ainsi que les valeurs qu'ils peuvent prendre.
Le modèle hybride, de par ses composantes discrètes, présente moins de paramètres qu'avec les équations différentielles et est plus simple à utiliser. De plus, son caractère continu au sein de chaque état discret permet d'être plus précis, notamment en permettant de suivre la trajectoire des différentes concentrations.

Une problématique importante de la modélisation de réseaux de régulation de gènes est donc l'identification des paramètres du modèle (seuils et vitesses). Dans ce contexte, l'objectif de notre projet sera d'écrire un algorithme et d'utiliser des techniques de machine learning afin de prédire les paramètres d'un réseau de régulation de gène, avec pour données d'entrée l'évolution de la concentration de chaque protéine. (partie inférieure de la Figure 2).
Ces données sont obtenues grâce à des simulations numériques dont le code nous a été fourni par notre encadrant Honglu SUN. Elles se présentent sous la forme de séries temporelles avec un pas de temps fixé, et sont perturbées par un bruit Gaussien modélisant les incertitudes de mesures.



(a) Repressilator (b) Un exemple de simulation

FIGURE 3 – Réseau de régulation de gènes étudié dans ce projet

Dans ce projet, nous nous intéréssons en particulier au repressilator de la figure 3, qui est un réseau de régulation de gènes constitué de 3 gènes qui s'inhibent mutuellement. Le réseau présente donc $2^3 = 8$ états discrets.

Il y a au total 15 paramètres à identifier :

- 3 seuils : θ_{ab}, θ_{bc}, θ_{ca}, compris entre 0 et 1.
- 12 vitesses : V_{a0b0}, V_{a0b1}, V_{a1b0}, V_{a1b1}, V_{b0c0}, V_{b0c1}, V_{b1c0}, V_{b1c1}, V_{c0a0}, V_{c0a1}, V_{c1a0}, V_{c1a1}, comprises entre 0 et 10.

Nous avons mis en place deux approches différentes :

D'une part, une méthode déterministe utilisant une estimation de la dérivée temporelle en chaque point de la simulation et recherchant les valeurs de seuil qui expliqueraient au mieux ces dérivées.

D'autre part, une approche Deep Learning dans laquelle nous entraînons un réseau de neurones à partir d'un grand nombre de simulations. Après avoir exploré chaque approche séparement, nous avons finalement essayé de les associer afin de combiner leur avantages et d'améliorer les prédictions.

2 Méthode des dérivées

2.1 Présentation et structure

Dans cette première partie, nous nous concentrons sur l'identification de la valeur des seuils θ_{ab}, θ_{bc}, θ_{ca}. Pour rappel, le seuil détermine la concentration au dessus de laquelle le gène va inhiber un autre gène. Le gène est alors dans l'état 1 s'il inhibe un autre gène et 0 s'il est inactif (concentration associée en dessous du seuil).

Ainsi, si le gène A est dans l'état 1 et inhibe B, alors la concentration x_b va décroître au cours du temps. A l'inverse, si le gène A est dans l'état 0 et est inactif, alors le gène B pourra coder la protéine B sans être inhibé et x_b sera croissante au cours du temps.

Il s'en suit que chaque état discret peut être caractérisé de manière unique par le sens de variation des concentrations x_a, x_b et x_c, autrement dit le signe de leur dérivée :

Etat discret	Signe dérivée
000	111
001	011
010	110
011	010
100	101
101	001
110	100
111	000

TABLE 1 – Tableau de correspondance entre états discrets et signe des dérivées

Cette caractérisation est évidemment liée aux seuils puisque par définition, l'état discret est obtenu en les comparant avec les concentrations. C'est donc à partir de ce tableau que nous allons construire une première méthode de détermination des seuils.

La méthode dite des dérivées consiste à déterminer les valeurs de seuil qui présentent la meilleure corrélation avec le signe des dérivées des concentrations au cours du temps, suivant le tableau de la figure 5. L'algorithme parcours pour cela un grand nombre de valeurs de seuils possibles et, pour chacune d'entre elles, calcule un score traduisant la corrélation. On retourne finalement un tableau des scores ; la fonction score sera construite de manière à ce que les valeurs de seuils testés avec les meilleurs scores correspondent aux valeurs réelles des seuils.

Algorithm 1 Méthode des dérivées

Require: simulation X, step
  for each (a, b, c) in the grid **do**
    Suppose that the threshold (θ_{ab}, θ_{bc}, θ_{ca}) are (a, b, c)
    Evaluate the score of (a, b, c) using derivatives and add it to the score table
  end for
  return Score table

— X correspond à la simulation de l'évolution des concentrations A,B et C au cours du temps. C'est un array avec 3 colonnes et un nombre de lignes correspondant aux instants auxquels les concentrations ont été mesurées.
— L'argument 'step' correspond au pas avec lequel les valeurs de seuils testées sont choisies. Par exemple, en 2 dimensions avec un step de 0.5, le score est calculé en testant les valeurs de seuils suivantes : (0,0), (0,0.5), (0,1), (0.5,0), (0.5,0.5), (0.5,1), (1,0), (1,0.5), (1,1).

Nous pourrons ajuster la valeur du step en fonction de la précision souhaitée, et suivant temps de calcul disponible. En effet, le nombre de score calculés étant de l'ordre de $\frac{1}{step}^3$, nous utiliserons souvent un step de 0.1 pour appliquer la méthode des dérivée, notamment au moment de l'appliquer sur un grand dataset dans la dernière partie.

La fonction score est le coeur de l'algorithme, car c'est elle qui détermine si les seuils testés ont de bonnes chances de correspondre aux seuils réels ou non. Nous en avons donc codé et comparé plusieurs.

2.2 Fonctions scores testées et résultats

La première fonction score implémentée considère simplement le sens de variation de la concentration. Chaque point de la simulation est associé à son état discret selon la valeur des seuils testés (a, b, c). On en déduit les sens de variations attendus avec le tableau de correspondance de la figure 5. Enfin, on compare ces sens de variations attendus avec les sens de variation réels que nous calculons. S'il y a correspondance, le score est incrémenté de 1 ; sinon le score reste le même.

Algorithm 2 Score signe

Require: simulation X, tested thresholds (a, b, c)
  $score \leftarrow 0$
  for each instant t of the simulation **do**
    Associate to the point (x_a^t, x_b^t, x_c^t) its discrete state according to (a, b, c)
    Evaluate $sign(x_a^{t+1} - x_a^t)$, $sign(x_b^{t+1} - x_b^t)$ and $sign(x_c^{t+1} - x_c^t)$
    if the signs match the discrete states as presented in table 1 **then**
      $score \leftarrow score + 1$
    end if
  end for
  return score

Dans un premier temps, pour tester l'efficacité de l'algorithme, nous l'appliquons sur quelques centaines de simulations et calculons l'erreur absolue moyenne (ou mean absolute error MAE). Ce premier score ne donne pas de bons résultats, puisque qu'on obtient une MAE de 0.39 pour des seuils entre 0 et 1. Au delà de la MAE, le principal problème associé à ce score est la présence d'instants pour lesquels la concentration est constante, le plus souvent dans lorsqu'on atteint la concentration minimum 0 ou maximum 1. Dans ce cas, on aura $x^{t+1} - x^t$ très proche de 0, avec un signe pouvant être positif ou négatif à cause du bruit. Le signe des dérivées dans ce cas-ci ne traduit donc pas parfaitement l'évolution des concentrations.

De plus, ne considérer que 2 points pour le calcul du sens de variation rend le résultat très sensible au bruit.

On écrit donc une nouvelle fonction score en prenant en compte ces observations :

— On calcule à présent la valeur de la pente/dérivée en plus de son signe. L'incrémentation du score en cas de correspondance avec l'état discret sera alors proportionnelle à la valeur de la dérivée. Ainsi, les instants où la dérivée est proche de 0 (maximum ou minimum) ne fausseront pas la valeur du score.
A l'inverse, les correspondances aux instants présentant de grandes valeurs de dérivée seront mieux récompensées, traduisant notre certitude sur le sens de variation dans ce cas.

— La dérivée sera calculée à partir d'un plus grand nombre de point (le nombre de point étant un paramètre à ajuster) permettant de réduire les effets du bruit. Il y a un compromis à trouver entre l'atténuation des effets du bruit et la variation de la valeur de la dérivée sur les points considérés (éloignement temporel de plus en plus grand quand on augmente le nombre de points). On introduit alors le paramètre nb_points qui est le nombre de points de part et d'autre de l'instant t utilisés pour calculer la dérivée. Par exemple, $nb\_points = 3$ signifie que la dérivée est calculée à partir des 3 points avant et des 3 points après.

Dans la partie suivante sur les réseaux de neurones, nous utiliserons également la version '1D' de cet algorithme, ne calculant le score et les pentes associés qu'à un seul seuil au lieu des trois.

Algorithm 3 Score signe+pente

Require: simulation X, tested thresholds (a, b, c), nb_points number of points (before and after) to evaluate the derivative with
  $score \leftarrow 0$
  for each instant t of the simulation **do**
    Associate to the point (x_a^t, x_b^t, x_c^t) its discrete state according to (a, b, c)
    for x_a^t, x_b^t and x_c^t (written as x_k^t) **do**
      $score\_temp \leftarrow 0$
      for j=1 to nb_points **do**
        $score\_temp \leftarrow score\_temp + \frac{x_k^{t+j} - x_k^{t-j}}{2j}$
      end for
      if the sign of $score\_temp$ matches the discrete states as presented in table 1 **then**
        $score \leftarrow score + score\_temp$
      end if
    end for
  end for
  return score

Le tableau ci-dessous présente les MAE obtenus pour ce nouvel algorithme pour différentes valeurs de nb_points et de $step$ pour la grille des seuils testés :

step \ nb_points	1	2	5
0.1	0.223	0.228	0.221
0.05	0.23	0.229	0.224

Les MAE sont bien plus faibles que celles obtenues avec le score signe, confirmant qu'ajuster le score en fonction des pentes est pertinent. Toutefois, pour des seuils compris entre 0 et 1, une MAE de 0.20 reste trop importante.
Le score signe+pente présente aussi ses limites. Certes, atténuer la contribution des dérivées nulles permet d'éviter les erreurs liées au signe fluctuant, mais en échange l'information associée à ces points est 'perdue', ou du moins négligeable en comparaison avec des dérivées plus importantes. De même, les gènes caractérisés par de grandes vitesse seront prédominants dans le calcul du score, au détriment des autres gènes.

D'autre part, les valeurs très proches des erreurs laissent à penser que la MAE n'est pas idéale afin d'évaluer la performance de l'algorithme. Il est possible en effet que les seuils prédits (ie avec le score maximal) soient assez différents des seuils réels, mais que ceux-ci aient un très bon score également. On s'intérèsse donc à la proximité entre le score maximal et le score associé aux seuils réels. La figure 4 représente le score associé à 2 seuils (par exemple θ_{ab} et θ_{bc}) calculé pour chaque valeur de seuil testée dans la grille :

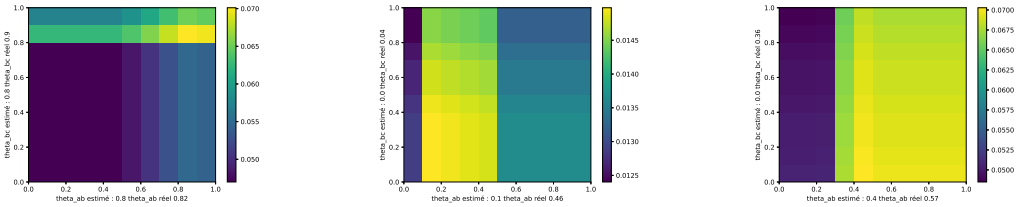


FIGURE 4 – 3 exemples de score map utilisant le score signe+pente avec nb_points = 4 et step = 0.1

Chacune des scoremap de la figure 4 a produit un résultat différent :

— Score map à gauche : θ_{ab} estimé = 0.8, θ_{ab} réel = 0.82 ; θ_{bc} estimé = 0.8, θ_{bc} réel = 0.9
La scoremap réussit bien à isoler par un meilleur score la zone correspondant aux seuils réels et produit une bonne estimation.

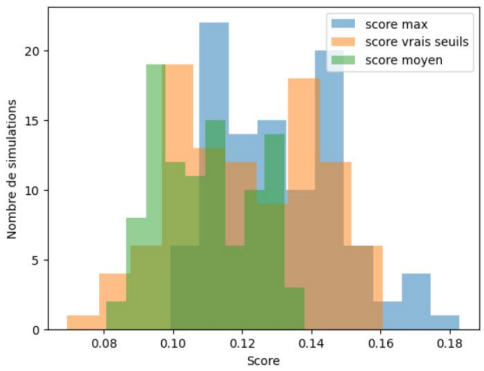
— Score map au centre : θ_{ab} estimé = 0.1, θ_{ab} réel = 0.46 ; θ_{bc} estimé = 0.0, θ_{bc} réel = 0.04
Bien que la prédiction soit assez éloignée des seuils réels, ces derniers ont obtenus un très bon score et sont situés dans la zone de la scoremap où les scores sont les plus importants. La scoremap fournit donc une région assez restreinte pour les valeurs de seuils possibles.

— Score map à droite : θ_{ab} estimé = 0.4, θ_{ab} réel = 0.57 ; θ_{bc} estimé = 0.0, θ_{bc} réel = 0.36
Comme pour la map précédente, la prédiction n'est pas très bonne, mais les seuils réels se situent dans une région où les scores sont très bons. Cependant, la région est beaucoup moins restrictive dans ce cas-ci.

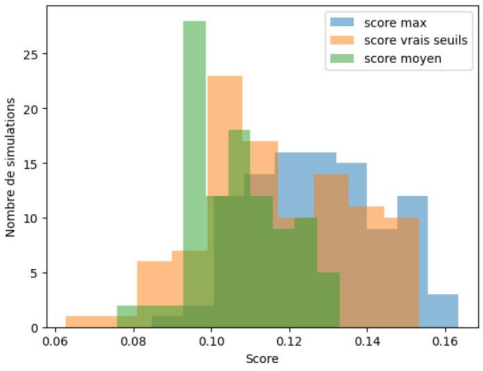
Contrairement aux techniques d'apprentissage statistique qui utilisent un grand nombre de simulations pour leur apprentissage, la méthode des dérivées n'en utilise qu'une seule, soulevant la question de sa robustesse. On évalue alors la robustesse de la méthode des dérivées en calculant le tableau des scores sur un nombre plus important de simulations et en comparant les distributions suivantes :

— La distribution du score maximum, c'est-à-dire la plus grande valeur dans le tableau des scores.
— La distribution du score associé au vrai seuil, à savoir la valeur dans le tableau des scores correspondant à celui obtenu pour les valeurs de seuils réelles, à *step* près.
— Enfin, la distribution du score moyen, c'est-à-dire la valeur moyenne du tableau des scores. Elle sert de référence pour comparer avec les deux autres distributions.

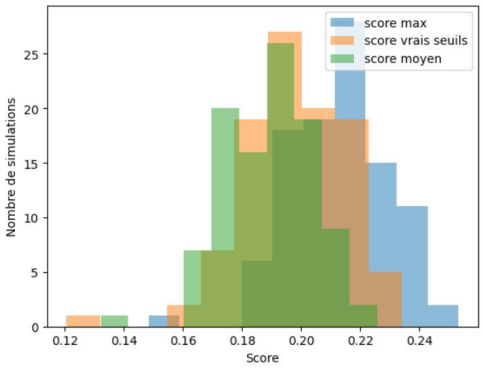
La figure 5 présente les distributions obtenues pour le score signe + pente. On compare également les distributions pour des simulations sans bruit à celles obtenues avec un bruit Gaussien de 0.2 (écart type).



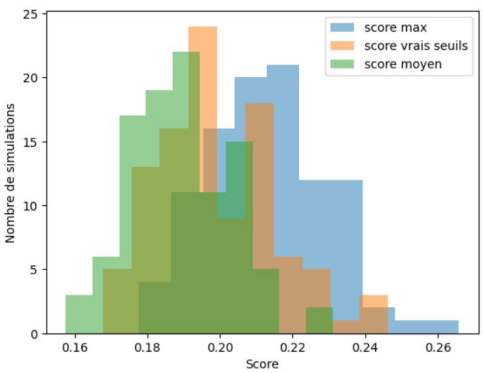
(a) Score signe + pente, pas de bruit



(b) Score signe + pente, pas de bruit



(c) Score signe + pente, bruit = 0.2



(d) Score signe + pente, bruit = 0.2

FIGURE 5 – Distribution des scores maximums, scores des vrais seuils et scores moyens calculée avec $N = 100$ simulations et $nb\_points = 5$

Les distributions obtenues montrent que le score associé aux vrais seuils, bien qu'il ne soit pas le meilleur, est plus élevé que le score moyen. La méthode des dérivées arrive donc dans une certaine mesure à traduire la corréllation qui existe entre valeur des seuils et valeur des dérivées.
On constate également que les résultats sont affectés par l'ajout d'un bruit.

Ainsi, la méthode des dérivées ne fournit pas une prédiction très précise des seuils, en raison de la grande variabilité des allures des simulations. En revanche, nous avons vu que le tableau des scores fournit des informations sur la région dans laquelle les seuils ont plus forte probabilité de se trouver. Ce sont donc des informations que nous allons pouvoir fournir à un réseau de neurone lors de son apprentissage.
Une piste d'amélioration possible du calcul du score serait de tenir compte de la saturation des concentrations. En effet, nous avons vu que le score tel qu'il est actuellement est augmenté proportionnellement à la valeur des dérivées. Autrement dit, le score ne varie que très peu lorsque la concentration reste constante à son maximum (concentration de 1) ou à son minimum (concentration de 0), et ce quelle que soit la valeur des seuils testés. Or, la saturation contient bien une information sur le seuil : si la concentration reste à 1, on peut en déduire qu'elle aurait de bonnes chances d'être croissante s'il n'y avait pas de plafond pour la concentration, et inversement pour le minimum. L'enjeu pour prendre en compte la saturation serait donc d'une part de la caractériser, et d'autre part de trouver une manière d'incrémenter le score qui soit compatible avec celui associé à la valeur des pentes pour les combiner.

3 Deep Learning

Pour identifier les paramètres du modèle hybride, une autre approche est de traiter ce problème comme un problème de Machine Learning classique. On peut considérer que les entrées sont des données séries temporelles et les sorties sont des paramètres du modèle hybride. On va entraîner différents réseaux de neurones pour prédire les sorties à partir des entrées.
Dans notre cas, les entrées vont être des séries temporelles représentant l'expression des 3 gènes, x_a, x_b et x_c.

Mr. Honglu SUN nous a fourni un code permettant de simuler ces données séries temporelles.

Les différents paramètres d'une simulation sont les suivants :
— *nb* : nombre de points dans la simulation
— *num* : nombre d'états qui touchent un nouveau bord dans la simulation (combien de fois la simulation change de dérivée temporelle)
— *initial_state* : état initial
— *(noise1, noise2, noise3)* : écart type du bruit gaussien qu'on rajoute à la simulation
— $\theta_{ca}, \theta_{ab}, \theta_{bc}$: seuils
— $V_{ac0a0}, V_{ac0a1}, V_{ac1a1}, V_{ba0b0}, V_{ba0b1}, V_{ba1b0}, V_{ba1b1}, V_{cb0c0}, V_{cb0c1}, V_{cb1c0}, V_{cb1c1}$: vitesses

On rajoute un bruit gaussien à la simulation pour mieux se rapprocher de la réalité. En effet, les données biomédicales sont souvent bruitées pour plusieurs raisons. Tout d'abord, les signaux biologiques sont souvent générés par des processus complexes qui peuvent être perturbés par de nombreux facteurs externes, tels que les mouvements du patient, les interférences électromagnétiques, les variations de l'environnement, etc. De plus, les appareils de mesure biomédicale peuvent présenter des limites inhérentes en termes de résolution, de sensibilité, de précision, etc. Enfin, les signaux mesurés peuvent être contaminés par des artefacts tels que les signaux de mouvement, les signaux électromagnétiques, les signaux de bruit de fond, etc.

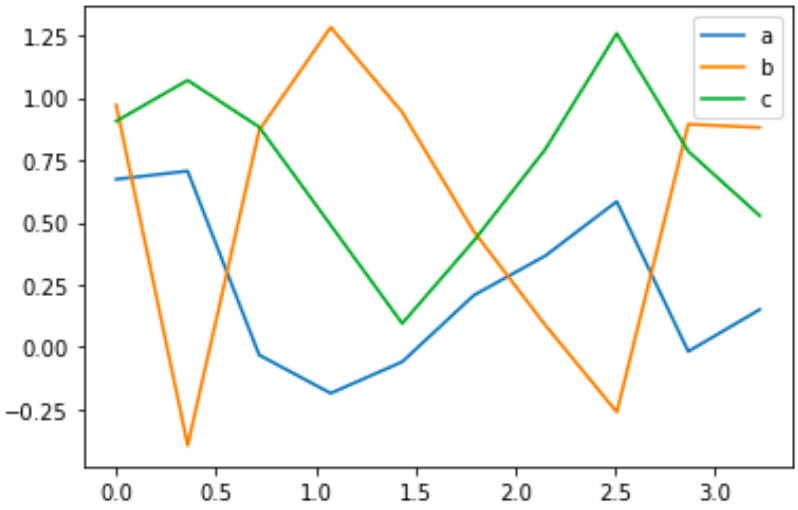


FIGURE 6 – Exemple de simulation

Pour créer le dataset de simulations, nous avons choisi les paramètres suivants :

— $nb = 100$
— $num = 20$
— $initial\_state$ choisi aléatoirement dans $[0, 1]^3$
— *(noise1, noise2, noise3)* fixés à 0 dans un 1^{er} temps pour choisis aléatoirement dans $[0, 0.2]^3$
— $\theta_{ca}, \theta_{ab}, \theta_{bc}$ choisis aléatoirement entre 0 et 1
— $V_{ac0a0}, V_{ac0a1}, V_{ac1a1}, V_{ba0b0}, V_{ba0b1}, V_{ba1b0}, V_{ba1b1}, V_{cb0c0}, V_{cb0c1}, V_{cb1c0}, V_{cb1c1}$ choisis aléatoirement entre 0 et 10

3.1 1^{er} modèle

Notre $1^{ère}$ idée a été de prédire les 15 paramètres (les 3 seuils et les 12 vitesses) avec un seul modèle.

Architecture du réseau :
- *Dense(256, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Flatten()*
- *Dense(15, activation = "linear")*

Voici quelques explications sur les différentes couches :
- Couche Dense : C'est une couche de neurones où chaque neurone est connecté à chaque neurone de la couche précédente (ou à l'entrée du réseau, si c'est la première couche). Chaque connexion est pondérée par un poids, et la somme pondérée des entrées est passée à travers une fonction d'activation pour produire la sortie de chaque neurone. Dans notre cas, la fonction d'activation est la fonction ReLU (Rectified Linear Unit) définie par $f(x) = max(0, x)$. Pour la couche de sortie, la fonction d'activation est la fonction d'activation linéaire définie par $f(x) = x$. On l'utilise en sortie car on veut ici prédire des paramètres continus.

- Couche Dropout : C'est une technique de régularisation qui peut être utilisée dans les réseaux de neurones pour prévenir le surapprentissage (overfitting) des données d'entraînement. Elle consiste à supprimer aléatoirement certains neurones de la couche pendant la phase d'entraînement. La Dropout rate (taux de suppresion) a été fixée à 0.2. Nous avons déterminé le taux optimal de suppression en utilisant la technique de "grid search".

- Couche Flatten : C'est une couche de traitement dans un réseau de neurones qui transforme une entrée multidimensionnelle en un vecteur à une seule dimension. Cette transformation est nécessaire pour permettre à l'entrée d'être connectée à une couche dense ou à une autre couche de traitement qui s'attend à recevoir des données unidimensionnelles. Cette couche est nécéssaire dans notre cas car les données d'entrées sont de taille (nb, 3) et on veut prédire 15 paramètres.

Dans ce modèle, les entrées sont les 3 simulations de taille nb et les sorties sont les 15 paramètres.

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.2 $2^{ème}$ modèle

Les seuils et les vitesses étant des paramètres ayant des significations différentes, nous avons décidé de définir 2 modèles différents : un qui prédit les 3 seuils et un qui prédit les 12 vitesses.

Architecture du réseau pour les seuils :
- *Dense(256, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Flatten()*
- *Dense(3, activation = "linear")*

Dans ce modèle, les entrées sont les 3 simulations de taille nb et les sorties sont les 3 seuils : $\theta_{ca}, \theta_{ab}, \theta_{bc}$.

Architecture du réseau pour les vitesses :
- *Dense(256, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Dense(128, activation = "relu")*
- *Dropout(0.2)*
- *Flatten()*
- *Dense(12, activation = "linear")*

Dans ce modèle, les entrées sont les 3 simulations de taille nb et les sorties sont les 12 vitesses : $V_{ac0a0}, V_{ac0a1}, V_{ac1a1}, V_{ba0b0}, V_{ba0b1}, V_{ba1b0}, V_{ba1b1}, V_{cb0c0}, V_{cb0c1}, V_{cb1c0}, V_{cb1c1}$.

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.3 $3^{ème}$ modèle

Comme nous travaillons avec des données séries temporelles, nous avons pensé aux réseaux de neurones récurrents. C'est un type de réseau de neurones artificiels qui est conçu pour traiter des données séquentielles. Contrairement aux réseaux de neurones classiques, qui prennent des entrées indépendantes les unes des autres, les RNN sont conçus pour traiter des séquences de données. Les RNN sont dotés d'une mémoire interne qui leur permet de prendre en compte les données précédentes dans la séquence lorsqu'ils traitent les données actuelles. Cette mémoire interne est souvent appelée "état caché" ou "mémoire à court terme". Grâce à cette mémoire, les RNN peuvent apprendre à identifier des motifs dans les données séquentielles et à générer des prédictions sur la base de ces motifs.
Nous avons ici utilisé un réseau de neurones récurrent pour prédire les 3 seuils.

Architecture du réseau :
— *SimpleRNN(32)*
— *Dense(3)*

La couche SimpleRNN est une couche récurrente de base qui prend en entrée une séquence de données et renvoie une séquence de sorties. Chaque sortie de la couche SimpleRNN est calculée à partir de la valeur de sortie précédente et de la nouvelle entrée.

On a en entrée les 3 simulations de taille nb et en sortie les 3 seuils θ_{ca}, θ_{ab} et θ_{bc}.

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.4 $4^{ème}$ modèle

Nous avons décidé de nous concentrer uniquement sur la prédiction des seuils, et en particulier sur un seul seuil : θ_{ab}. En entrée, au lieu d'avoir les 3 séries temporelles, on utilise seulement x_a et x_b. Il est inutile de considérer l'expression du gène c pour prédire θ_{ab}.

Architecture du réseau :
— *SimpleRNN(32)*
— *Dense(8, activation = "relu")*
— *Dropout(0.3)*
— *Dense(4, activation = "relu")*
— *Dropout(0.3)*
— *Dense(2, activation = "relu")*
— *Dropout(0.3)*
— *Dense(1, activation = "linear")*

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.5 $5^{ème}$ modèle

L'idée est qu'à partir des observations à 2 instants consécutifs pour x_a et x_b on peut obtenir des informations.

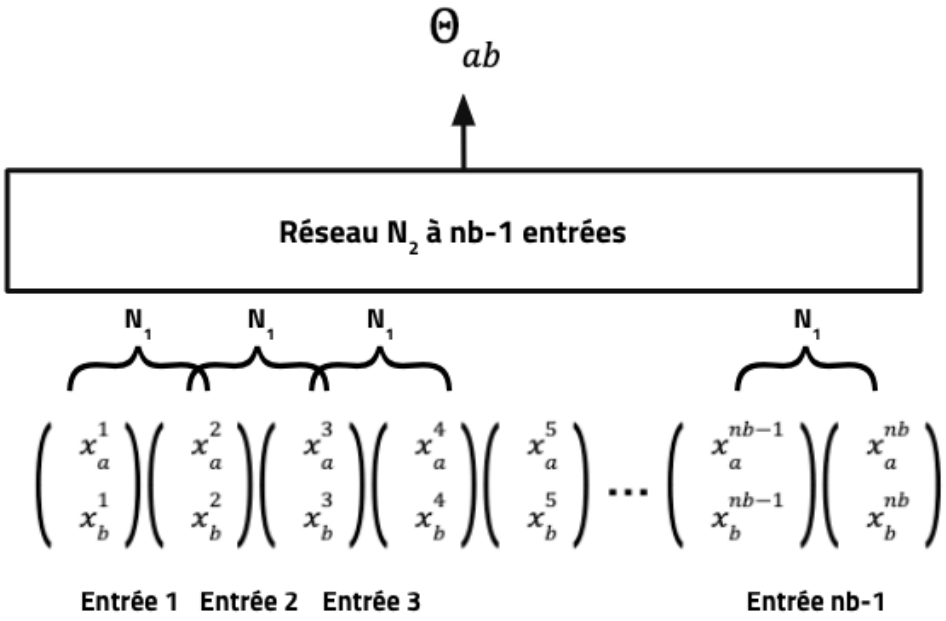


FIGURE 7 – Schéma explicatif du $5^{ème}$ modèle

Architecture du réseau N_1 :
— *Dense(32, activation = "relu")*
— *Dropout(0.3)*
— *Dense(16, activation = "relu")*
— *Dropout(0.3)*
— *Dense(8, activation = "relu")*
— *Dropout(0.3)*
— *Flatten()*

Architecture du réseau N_2 :
— *Dense(4, activation = "relu")*
— *Dense(2, activation = "relu")*
— *Dense(1, activation = "linear")*

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.6 $6^{ème}$ modèle

Pour améliorer nos résultats, nous avons décidé de combiner nos 2 méthodes.
Le réseau de neurones précédent va prendre une entrée supplémentaire : la score map en 3D.

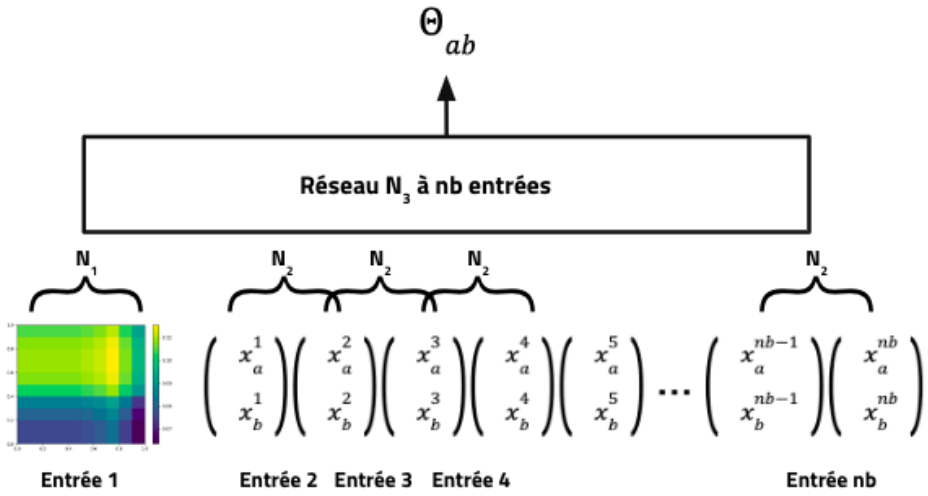


FIGURE 8 – Schéma explicatif du $6^{ème}$ modèle

Architecture du réseau N_1 :
— *Conv2D(32, kernel_size = (3, 3), activation = "relu")*
— *MaxPooling2D(pool_size = (2, 2))*
— *Conv2D(64, kernel_size = (3, 3), activation = "relu")*
— *MaxPooling2D(pool_size = (2, 2))*
— *Flatten()*
— *Dense(8, activation = "relu")*

Architecture du réseau N_2 :
— *Dense(32, activation = "relu")*
— *Dropout(0.3)*
— *Dense(16, activation = "relu")*
— *Dropout(0.3)*
— *Dense(8, activation = "relu")*
— *Dropout(0.3)*
— *Flatten()*

Architecture du réseau N_3 :
— *Dense(4, activation = "relu")*
— *Dense(2, activation = "relu")*
— *Dense(1, activation = "linear")*

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.7 $7^{ème}$ modèle

Au lieu de la score map en 3D, nous allons donner en entrée du réseau un score 1D (uniquement pour θ_{ab}).

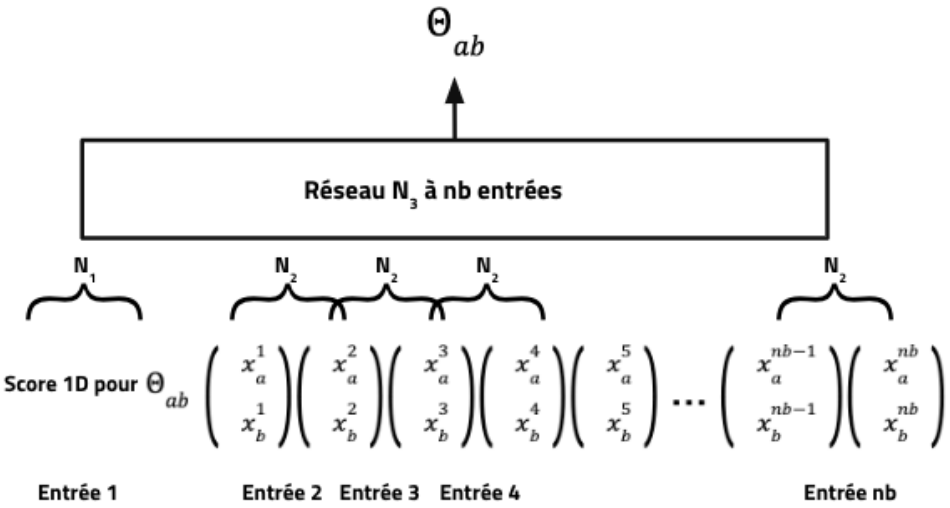


FIGURE 9 – Schéma explicatif du $7^{ème}$ modèle

Architecture du réseau N_1 :
— *Dense(16, activation = "relu")*
— *Dropout(0.5)*
— *Dense(8, activation = "relu")*
— *Dropout(0.3)*
— *Flatten()*

Architecture du réseau N_2 :
— *Dense(32, activation = "relu")*
— *Dropout(0.3)*
— *Dense(16, activation = "relu")*
— *Dropout(0.3)*
— *Dense(8, activation = "relu")*
— *Dropout(0.3)*
— *Flatten()*

Architecture du réseau N_3 :
— *Dense(4, activation = "relu")*
— *Dense(2, activation = "relu")*
— *Dense(1, activation = "linear")*

Les résultats sont présentés dans les parties 3.8.1 et 3.8.2.

3.8 Tableau des résultats

Nous avons évalué la performance de nos modèles grâce à la validation croisée k-fold. C'est une technique courante utilisée pour évaluer la performance d'un modèle pour estimer sa capacité à généraliser à de nouveaux ensembles de données.

Le processus de validation croisée k-fold consiste à diviser les données en k "folds" de taille égale. Ensuite, le modèle est entraîné k fois en utilisant k-1 partitions pour l'entraînement et la partition restante pour la validation. À chaque itération, une partition différente est utilisée pour la validation.

Une fois que chaque partition a été utilisée une fois pour la validation, les performances sont moyennées pour fournir une estimation plus fiable de la performance du modèle.

Dans notre cas, nous avons utilisé 5 folds.
Le réseau est entraîné sur 4 partitions puis tester sur celle restante. $score_k$ représente l'erreur absolue moyenne entre les prédictions et les valeurs réelles du test set. On va ensuite moyenner les 5 scores.

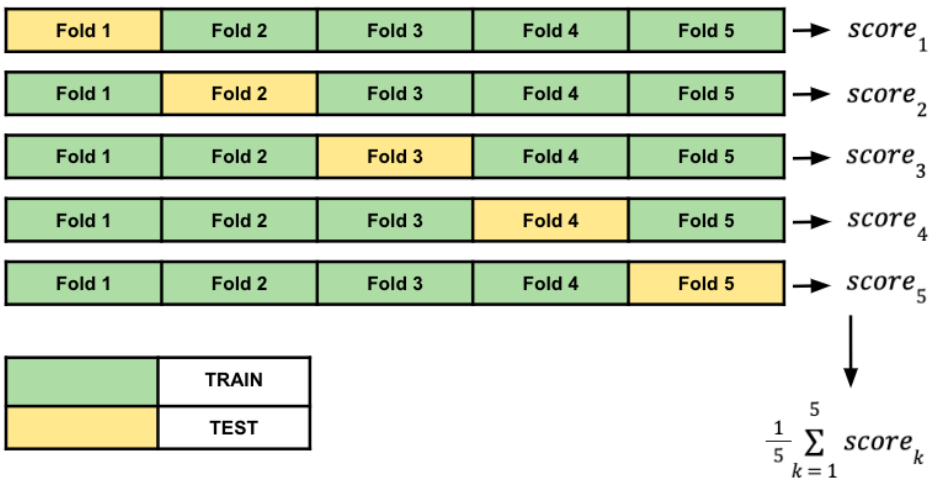


FIGURE 10 – Calcul du score par validation croisée 5-fold

3.8.1 Sans bruit

Ces scores ont été calculés avec un nombre d'epochs égal à 1000 et $batch\_size = 64$. La taille du dataset est de 1000 simulations.

Modèle \ Paramètres	Seuils	Vitesses
1	0.189	2.67
2	0.158	2.65

TABLE 2 – Erreurs absolues moyennes pour le 1^{er} et $2^{ème}$ modèles

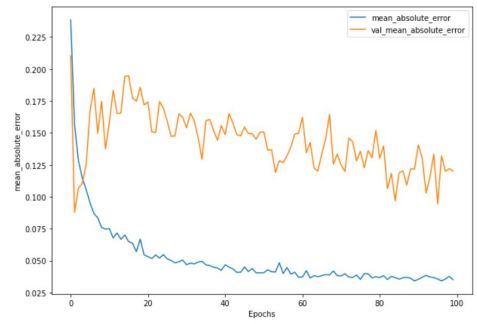
Modèle 3	0.163
Modèle 4	0.171
Modèle 5	0.211

TABLE 3 – Erreurs absolues moyennes pour les modèles 3 à 5

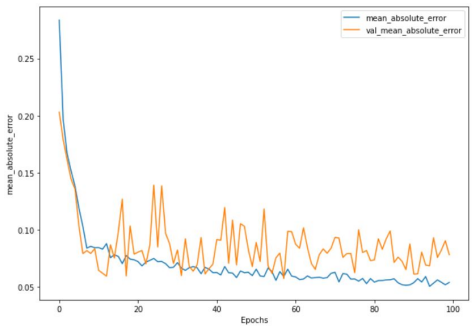
Modèle \ nb_points	1	3	5
Modèle 6	0.124	0.080	0.076
Modèle 7	0.236	**0.042**	0.064

TABLE 4 – Erreurs absolues moyennes pour les modèles 6 et 7

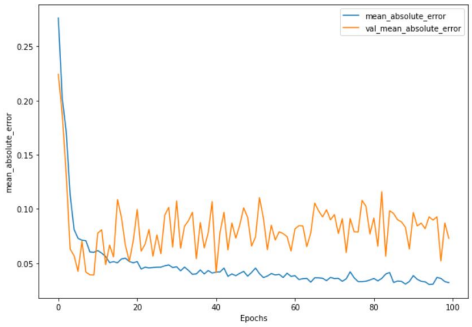
Le $7^{ème}$ modèle avec $nb\_points = 3$ est celui qui a donné les meilleurs résultats.



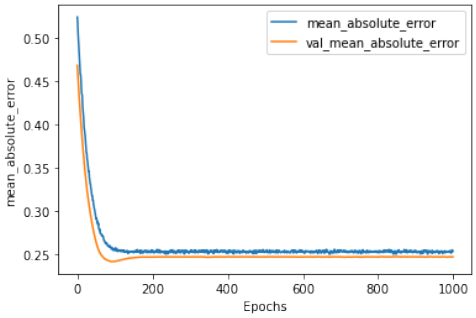
(a) Modèle 6, $nb\_points = 1$



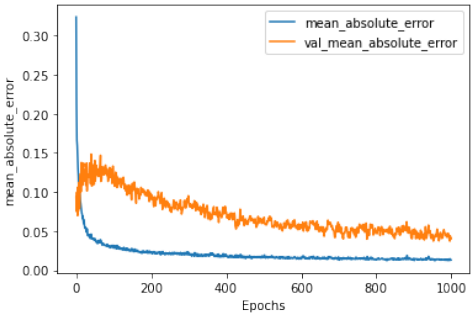
(b) Modèle 6, $nb\_points = 3$



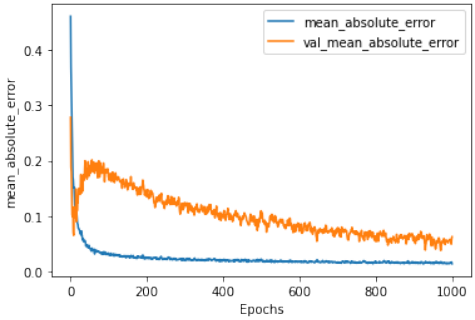
(c) Modèle 6, $nb\_points = 5$



(d) Modèle 7, $nb\_points = 1$



(e) Modèle 7, $nb\_points = 3$



(f) Modèle 7, $nb\_points = 5$

FIGURE 11 – Évolution de l'erreur absolue moyenne en fonction du nombre d'epochs pour des simulations sans bruit

3.8.2 Avec bruit

Ces scores ont été calculés avec un nombre d'epochs égal à 1000 et $batch\_size = 64$. La taille du dataset est de 1000 simulations.

Modèle \ Paramètres	Seuils	Vitesses
1	0.180	2.53
2	0.159	2.56

TABLE 5 – Erreurs absolues moyennes pour le 1^{er} et $2^{ème}$ modèles

Modèle 3	0.19
Modèle 4	0.188
Modèle 5	0.253

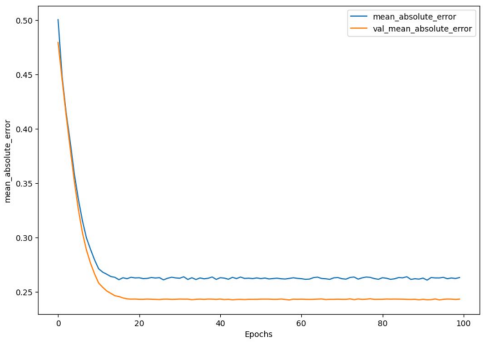
TABLE 6 – Erreurs absolues moyennes pour les modèles 3 à 5

Modèle \ nb_points	1	3	5
Modèle 6	0.247	0.094	0.085
Modèle 7	0.247	**0.041**	0.043

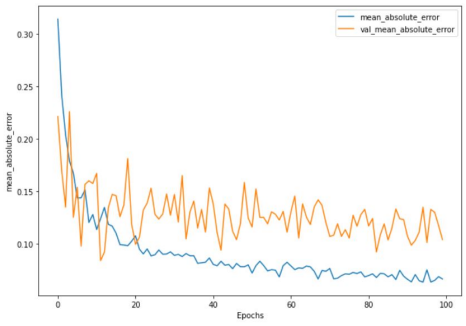
TABLE 7 – Erreurs absolues moyennes pour les modèles 6 et 7

Le $7^{ème}$ modèle avec $nb\_points = 3$ est celui qui a donné les meilleurs résultats.

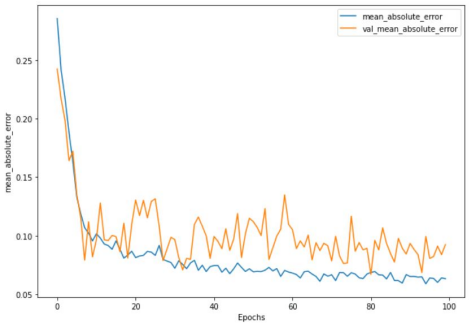
La combinaison des deux méthodes en général donne les meilleurs résultats, même avec du bruit. Lorsque le paramètre nb_points est supérieur à 1, la convergence du réseau de neurones est plus rapide. Trop augmenter nb_points finira toutefois par ne plus améliorer voire dégrader les résultats, à l'image des résultats des tableaux 2 et 7.



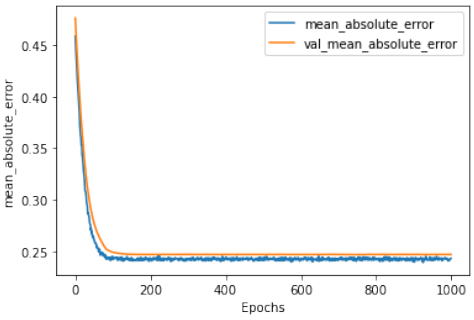
(a) Modèle 6, $nb\_points = 1$



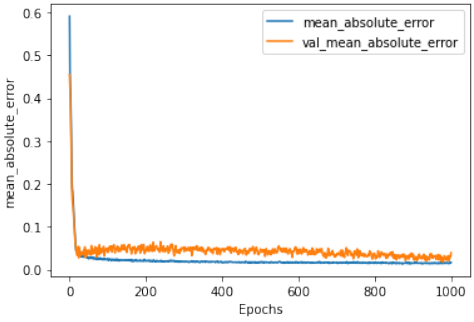
(b) Modèle 6, $nb\_points = 3$



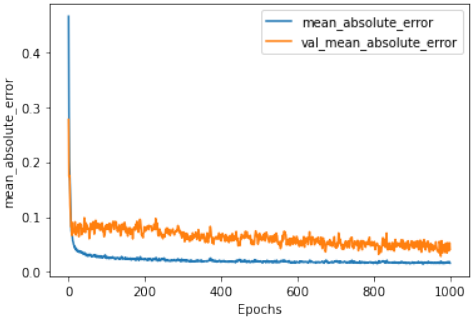
(c) Modèle 6, $nb\_points = 5$



(d) Modèle 7, $nb\_points = 1$



(e) Modèle 7, $nb\_points = 3$



(f) Modèle 7, $nb\_points = 5$

FIGURE 12 – Évolution de l'erreur absolue moyenne en fonction du nombre d'epochs pour des simulations avec bruit de 0.2

4 Conclusion

A l'issue de cette étude, nous avons réussi à mettre en oeuvre les méthodes que nous avons envisagées en début de projet, puis à les combiner. Nous nous sommes concentrés sur la prédiction des seuils, mais des techniques similaires pourront également être utilisées pour la prédiction des vitesses : les réseaux de neurones tout comme le calcul des pentes pourraient fonctionner.

Il n'est pas évident de déterminer si les résultats obtenus sont satisfaisants ou non. Les réseaux de régulation de gènes sont actuellement un sujet de recherche et il est difficile d'obtenir des données réelles. Pour autant, nous pouvons considérer que les résultats sont prometteurs dans la mesure où l'erreur de prédiction a diminué tout au long du projet pour au final atteindre sa valeur minimale avec la combinaison des deux méthodes. Nous transmettons ce rapport ainsi que nos avancées à nos encadrants qui pourront les réutiliser voire approfondir notre travail.

Le projet nous a permis de progresser sur des aspects techniques comme les réseaux de neurones, et d'avoir une idée du déroulement d'un projet de recherche. Nous remercions nos encadrants Honglu SUN et Morgan MAGNIN pour cette opportunité ainsi que pour leur suivi tout au long de l'année.

5 Références

— Cornillon, E. et al., advances in Systems and Synthetic Biology., 2016 Cornillon, E., Comet, J. P., Bernot, G., & Enee, G. (2016). *Hybrid gene networks : a new framework and a software environment. advances in Systems and Synthetic Biology.*

— Behaegel, J. et al., Journal of bioinformatics and computational biology, 2016 Behaegel, J., Comet, J. P., Bernot, G., Cornillon, E., & Delaunay, F. (2016). *A hybrid model of cell cycle in mammals. Journal of bioinformatics and computational biology*

— `https://pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/`