le cnam
Paris

I. Table des matières

II. Contexte

Jeune ingénieur diplômé de l'école nationale supérieure des arts & métiers, j'ai toujours été intéressé par les outils informatiques et l'aide que ces derniers apportent dans notre quotidien. Avec des amis nous menons un projet qui implique l'utilisation de réseau de neurones à des fins de meilleures consommations énergétiques. Afin d'approfondir mes connaissances sur ces outils, et de m'initier au monde de la programmation informatique, je me suis inscrit au C.N.A.M. dans l'ensemble des modules constituant le certificat de spécialisation d'intelligence artificielle sur l'année 2020-2021.

III. Objectifs

Les objectifs dans ce rapport sont de mettre en pratique les connaissances acquises lors de l'enseignement du module RCP 217 : Intelligence Artificielle pour les données multimédia, autour d'un cas d'étude. Parmi ceux proposés j'ai sélectionné le sujet « Deep Kalman » : mise en place du réseau de neurones permettant l'inférence des fameux filtres de Kalman basé sur le papier de Rahul et coll., une fois le réseau constitué, le second objectif est d'en analyser ses capacités de prédictions sur des séries temporelles.

IV. Périmètre de l'étude

Dans le cadre de cette étude, je me suis basé sur un ensemble de documents préalablement étudié. Il s'agit d'archives météo (cf. source en fin de section) qui recensent des relevés de différentes métriques : le débit entrant dans le réservoir du lac Saint-Jean, la pluie dans la région du lac Saint-Jean, et la fonte des neiges dans le bassin Daval. À noter que les données sont sans unités, ce qui ne nous gênera pas pour cette étude puisque nous nous intéresserons aux valeurs uniquement.

J'ai décidé de mener l'étude en deux temps :

- Mettre en place le réseau tel que décrit dans le papier associé à l'étude
- Mesurer les performances des réseaux neuronaux avec de différentes compositions « paramétrique » sur les prédictions de débit entrant du quart de mois suivants

Pour chaque composition de réseau, il y a 2 jeux de données :

- Le premier regroupe les « observations », sur une fenêtre de temps donné, en l'occurrence le débit entrant dans le lac
- Le second jeu contient l'ensemble des « actions » qui ont entraîné l'observation, c'est-à-dire le débit entrant, la pluie dans la région, et la fonte des neiges.

Chacun de ces jeux est donné sur une fenêtre de temps paramétré manuellement. Les entrées sont construites avec le jeu de donnée par le principe de fenêtre glissante : par exemple, pour une fenêtre de temps de 12 mois, il sera considéré l'ensemble des 48 observations précédent à chaque quart de mois, en prenant en compte les conditions aux limites.

Afin de mener l'étude, j'ai au préalable :

- Concaténé l'ensemble des informations dans un seul tableau (de type dataframe) nommé Big_data ayant comme colonne (dans l'ordre) : « année », « quart de mois de l'année », « débit entrant », « pluie », « fonte des neiges »
- Défini 80% du jeu de donnée comme jeu d'entraînement, en l'occurrence les 26 premières années. Je ne mélangerai pas les données ainsi constitué, le but étant que le modèle repère s'il y a une dérive au sein des données
- Écrit une fonction qui crée les jeux de donnée des observations et des actions (train, validation, test) avec comme entrée : la fenêtre de temps, le nombre d'années d'entraînement et de validation

Dans la section suivante se trouve une introduction aux données utilisées dans le cadre de cette étude, et les traitements apportés pour leurs exploitations dans le reste des sections.

(Source : http://fisher.stats.uwo.ca/faculty/aim/epubs/mhsets/thompsto/)

V. Analyse des données

Dans cette section nous allons mettre en forme les données afin de pouvoir tirer des premiers éléments de réflexions sur les liens de cause à effet. À savoir que la fréquence des relevés est par quart de mois, soit 48 données à l'année, et sur une période de 30 ans de 1953 à 1982. Ce qui fait un total de 1440 observations par métriques.

A. Prétraitements

Après avoir transformé les données téléchargées en format exploitable (stocké dans le code dans une classe de type ndarray), puis transformé dans un dataframe, sont tracés les données de chaque métrique en fonction du temps, avec en bleu le débit entrant, en vert la pluie dans la région, et en orange la fonte des neiges, segmentées par année :

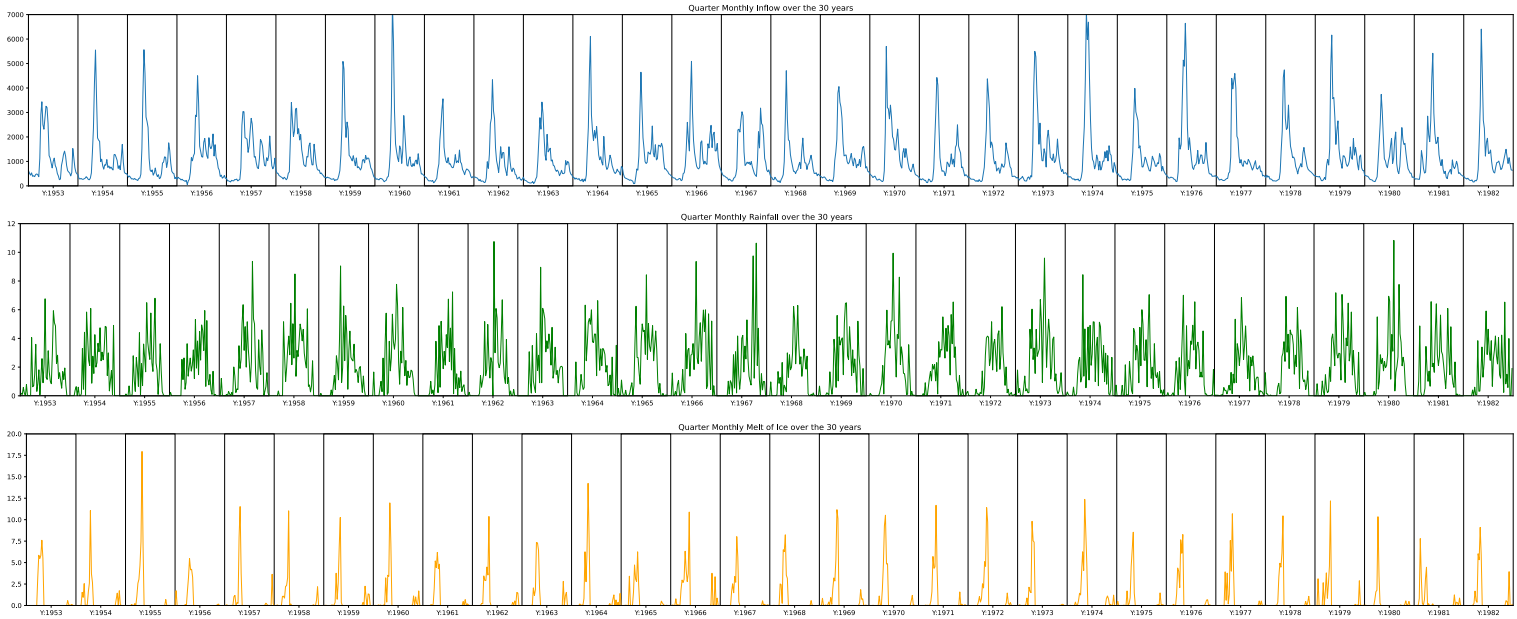


Figure 1 tracée des 3 métriques sur les 30 ans

Les images sont importées au format SVG, et sont également disponibles en annexe 1,2 & 3 dans le dossier du rapport.

Sur les graphiques ci-dessus une certaine périodicité des observations est remarquée d'une année à l'autre, ce que je vais tenter de justifier dans la section suivante avec un traitement des données.

B. Post-traitements

Afin de faire ressortir la périodicité des données, je trace les graphiques par métrique avec une superposition des observations sur une fenêtre d'un an, avec une moyenne transversale en pointillé noir (moyenne sur un quart de mois de toutes les années) :

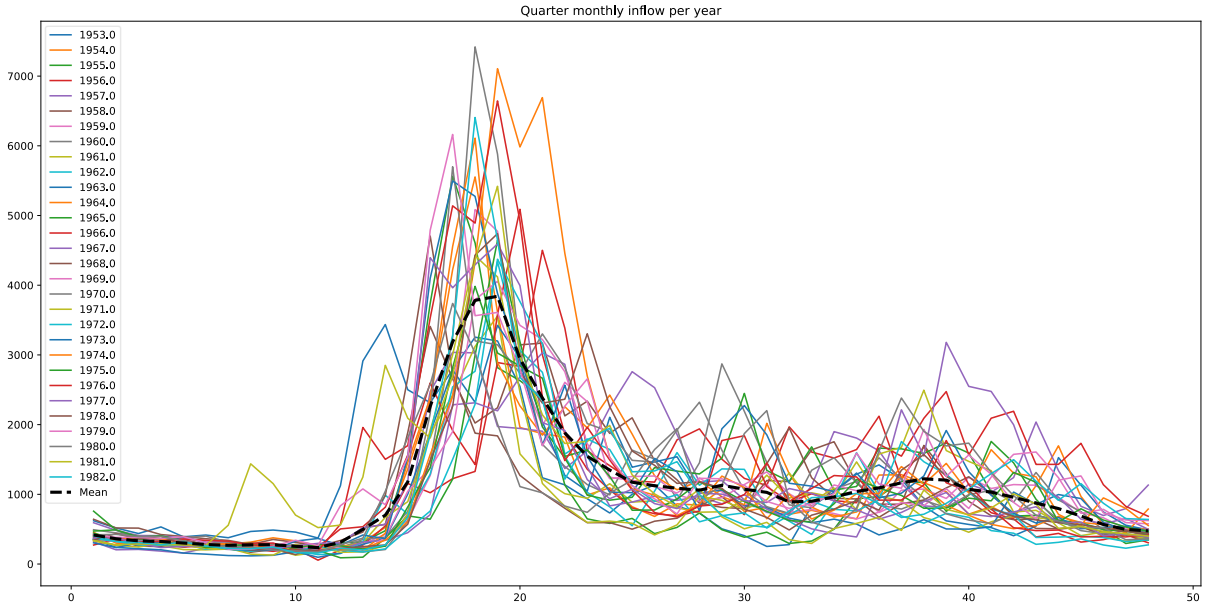


Figure 3 Superposition des observations de débit entrant sur une fenêtre d'un an, avec une moyenne "transversale"

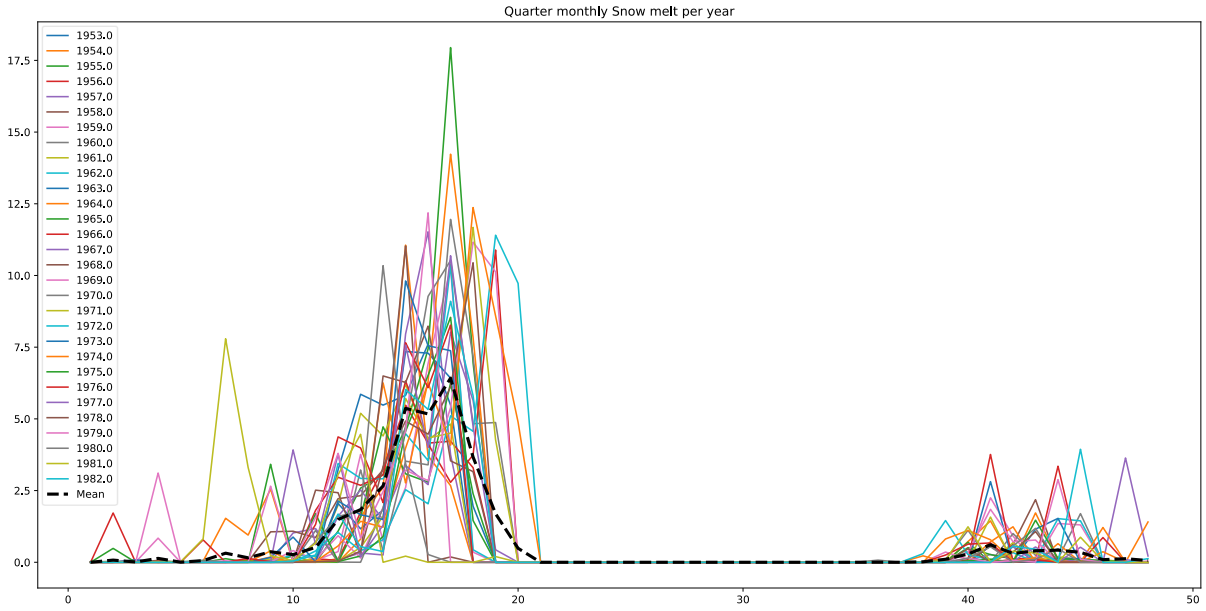


Figure 2 Superpositions des observations de la fonte des neiges sur une fenêtre d'un an, avec une moyenne "transversale"

Les images sont importées au format SVG, et sont également disponibles en annexe 4,5 & 6 dans le dossier du rapport.

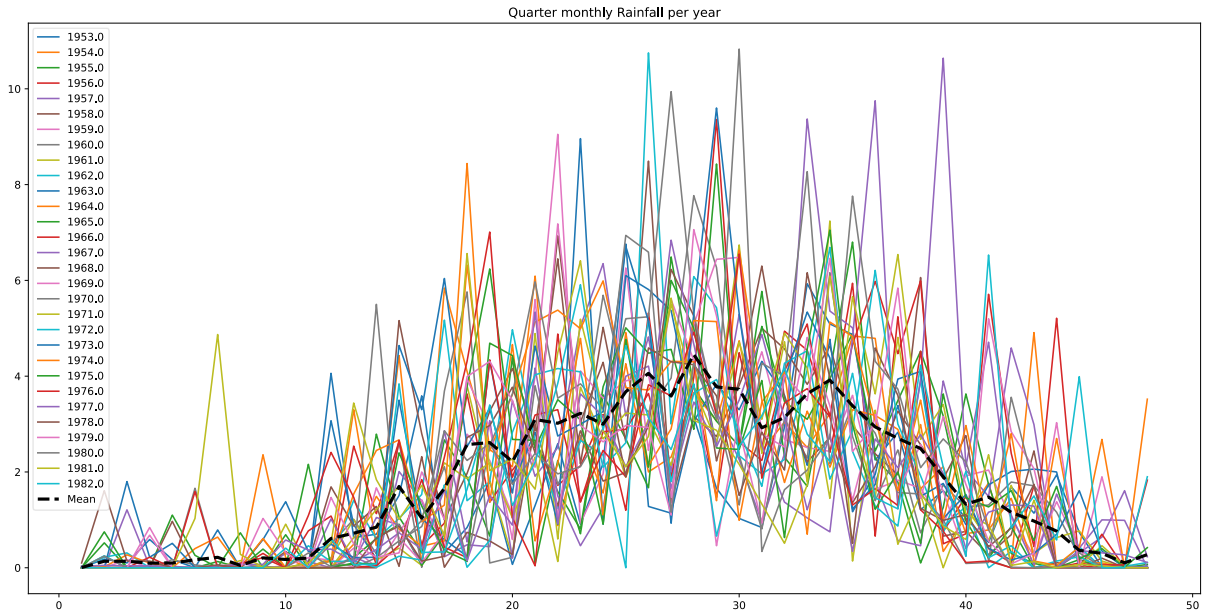


Figure 4 Superpositions des observations de la pluie sur une fenêtre d'un an, avec une moyenne "transversale"

Les images sont importées au format SVG, et sont également disponibles en annexe 4,5 & 6 dans le dossier du rapport.

Plusieurs remarques sont à faire en adéquation avec le monde et sa physique.

Vis-à-vis du débit :

- On constate en effet que le débit entrant dans le lac suit une tendance relativement similaire d'une année à l'autre avec parfois des décalages sur l'abscisse, ou des pics d'intensité différents, sûrement dû aux saisonnalités

Vis-à-vis de la fonte des neiges

- De la même manière que le débit, la fonte des neiges suit une tendance d'une année à l'autre
- On constate une première corrélation, également basé sur le savoir scientifique, entre les pics de fonte des neiges et les pics de débit entrant

Vis-à-vis de la pluie :

- La moyenne transversale nous permet d'observer une tendance ou l'indice de pluie et plus fort sur milieu de l'année, cependant la disparité des observations est très forte, sûrement dû également au caractère plus aléatoire de la pluie

Bien qu'il est difficile de tirer une conclusion sur la métrique de pluie telle qu'elle est présentée ci-dessus, je me doute qu'en période de forte pluie cette dernière influe sur le débit entrant de la même manière que la fonte des neiges lors de la période estivale.

Afin de mieux observer cette corrélation, j'ai superposé sur 4 années différentes les données normalisées de 0 à 1 du débit entrant, de la pluie, et de la fonte des neiges :

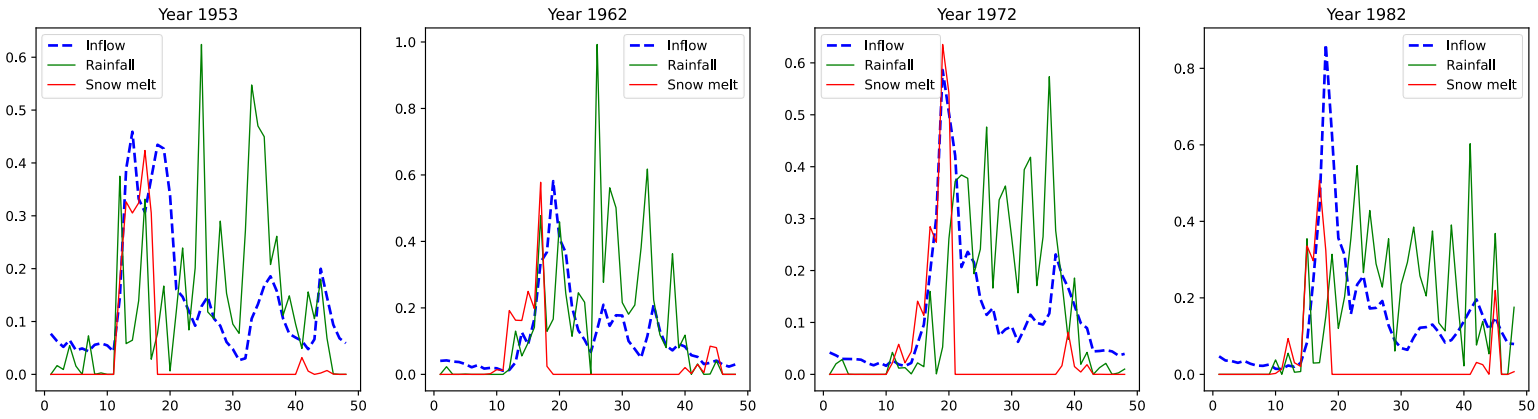


Figure 5 - Superposition des données normalisées pour 4 années

Maintenant que nos données sont formatées, que les éléments de cause à effets basés sur le savoir scientifique sont vérifiés, nous pouvons concevoir les premiers modèles de réseaux de neurones afin de capturer cette corrélation et de pouvoir prédire les observations futures.

VI. Mise en place du réseau Deep Kalman Filter

Le filtre de Kalman tient son nom de l'ingénieur hongrois Rudolf Kalman, c'est un algorithme qui permet d'estimer l'état d'un système dynamique à partir des données mesurées, même si ces dernières sont incomplètes ou bruitées. L'algorithme du filtre est un processus en deux étapes : la première étape consiste à prédire l'état du système, communément appelé étape de prédiction et la seconde étape utilise les mesures bruitées pour affiner l'estimation de l'état du système, appeler également étape de mis à jour / correction. Grâce à ce système d'estimation récursive, le filtre à trouver des applications dans de nombreux domaines tel que la robotique, la finance, traitement de signal, etc.

Cependant le filtre est construit sur 2 hypothèses fortes : que tout est gaussien, et que tous les modèles sont linéaires, le filtre n'est donc optimal que sur une petite plage linéaire des phénomènes réels. Ainsi des extensions de ce filtre ont été introduites afin que la non-linéarité soit associée au modèle du processus, au modèle d'observation ou bien aux deux. Le papier ci-étudié introduit une nouvelle technique de non-linéarité par le biais des réseaux de neurones.

A. Synthèse du papier

L'objectif du papier est de mettre en place un réseau qui est une inférence du filtre de Kalman à l'aide d'un réseau de neurones profond, tout en gardant les étapes de prédiction et de mises à jour récursives. L'objectif est de trouver des liens de cause à effet entre actions et observations au sein des données. Plus précisément, il souhaite être en mesure de prédire les corrélations entre les actions des prescriptions médicales sur les patients et les observations de leurs registres médicales, dans le but de pouvoir prédire de meilleures prescriptions aux futurs patients.

L'utilisation des réseaux de neurones permet de complexifier de manière « arbitraire » ces classes de transition (équivalent de l'étape de prédiction) et correction, sans pour autant avoir un coût de calcul qui soit trop élevé, ce qui est la limite rencontrée par les extensions non linéaires de ce filtre. L'idée est donc de mettre en place un modèle génératif probabiliste des séquences d'observations complexes, non linéaires. La distribution étant intraitable analytiquement, l'approximer correctement de façon numérique est donc un enjeu majeur pour le modèle, c'est à ce titre que sont testés dans le papier différents réseaux de reconnaissance :

- Avec des perceptrons multicouches, dont un premier modèle intégrant les données du passé T-1, et un second modèle intégrant les données du passé T-1, du présent T, et du futur T+1
- Avec des réseaux de neurones récurrents dont un premier modèle monodirectionnel et un second bidirectionnel, prenant en compte toutes les données du passé au présent, de 1 à T

Le but étant d'appliquer le principe des « variational auto-encoder » avec la mesure de divergence de Kullback-Leibler pour approximer cette vraie distribution. Pour mener son étude, le papier introduit une nouvelle base de données : Healing MNIST, basé sur le célèbre jeu de chiffres manuscrits, auquel il apporte des modifications telles que des rotations, de la superposition aléatoire de carrés dans les coins en haut à gauche, et du bruitage aléatoire sur l'ensemble de l'image. Ces modifications sont censées imiter les mesures de laboratoires qui sont bruités, avec des diagnostics qui sont parfois des tentatives, parfois des redondants, ou différés.

C'est sur cette base de données nouvellement introduite que le papier montre la capacité du réseau à démontrer des inférences à l'aide de la non-linéarité, et ce, malgré la complexité des observations. Au travers de cette étude, les chercheurs se demandent qu'elles sont les capacités des espaces latents à encoder des caractéristiques identifiables.

Après cette synthèse qui introduit la structure générale du réseau, je vais introduire quelques schémas du modèle général et tel qu'il sera développé dans notre programme.

B. Représentation graphique du modèle et de ses paramètres

Le modèle général prend la forme d'un réseau de neurones récurrent, cependant la transition d'un neurone de $(\tau-1)$ à τ passe par des « sous-réseaux » qui viennent appliquer les 2 principes décrits précédemment :

- Celui de l'auto-encodeur avec un espace latent structuré par une distribution, qui sera défini ici comme une distribution normale dans le cadre du projet
- Celui du filtre de Kalman avec une phase de transition et une phase de correction par le postérieur

Ces « sous réseaux » sont non-linéaires, et ce sont leurs paramètres qui seront optimisés lors des itérations dans la phase d'entraînement. À noter que la dimension τ dépend de la largeur de la fenêtre des observations (exemple 12 mois, soit 48 quarts de mois), c'est un paramètre défini manuellement lors de la création des jeux d'entraînement, de validation, et de test des observations et d'actions.

Pour initialiser le réseau, un premier espace latent Z_1 est défini de manière aléatoire dont le décodeur en déduit une première observation X_1. À l'étape suivante, l'espace latent précédemment défini passe dans le réseau de transition pour en tirer une distribution normale paramétrée par : $\mu_{transition}$ et $\sigma_{transition}$. Ces deux paramètres ainsi que les données du jeu d'observations X et d'actions U à $\tau=1$ sont données en entrée du réseau postérieur, appelé également « réseau de correction ».

Une première façon de faire serait de donner séparément en entrée du postérieur les observations du débit entrant, ainsi que les actions concaténant la pluie et la fonte de neiges. Cependant, les données d'actions telles que je l'ai défini sont constituées des métriques de débit, de pluie et de fontes des neiges, le jeu d'action U intègre donc les données d'observations X.

Par conséquent afin de faciliter la synthèse du modèle ainsi que le développement, j'ai pris la liberté de donner en entrée du postérieur ces données d'actions U.

Le réseau postérieur en déduit une nouvelle distribution normale paramétrée par : $\mu_{postérieur}$ et $\sigma_{postérieurs}$. À la sortie du réseau de correction, le modèle utilise une astuce de reparamétrisation pour générer des éléments qui vont définir le nouvel espace latent. Et ainsi de suite jusqu'à ce que la fin de la fenêtre de temps à τ=T.

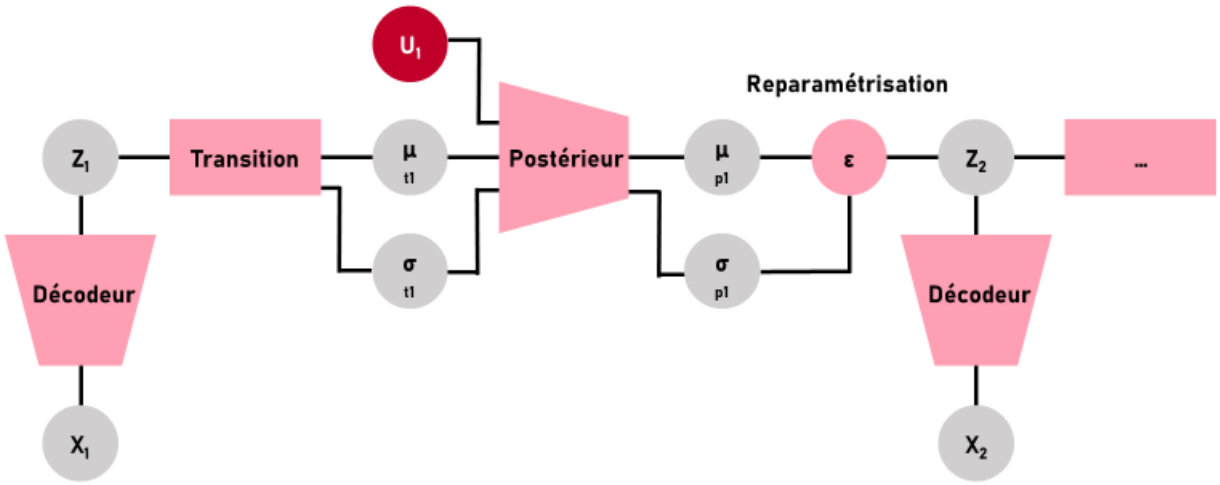
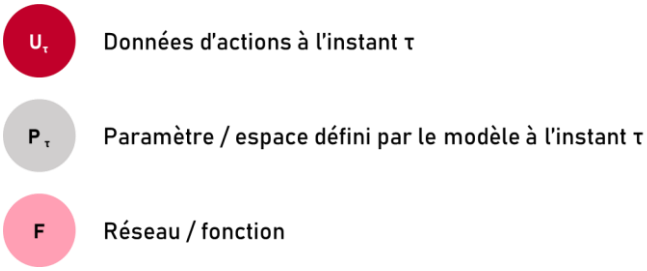


Figure 6 Schéma illustrant l'initialisation du réseau Deep Kalman Filter tel que défini dans ce projet

Légende :



Les données d'actions n'intervenant pas sur le premier espace latent, la dimension U est inférieure d'une unité de temps aux données d'observations X. Chaque « sous-réseau » est constitué d'une couche cachée, voici les paramètres pour chacun d'eux :

Réseau du décodeur :

- z_dim : dimension de l'espace latent z_t
- decodeur_dim_cachee : dimension de la couche cachée du décodeur entre l'espace latent et la sortie
- decodeur_dim_sortie : dimension de la sortie du décodeur

Réseau de transition :

- z_dim : dimension de l'espace latent z_t
- transition_dim_cachee : dimension de la couche cachée du réseau de transition entre l'espace latent z_{t-1} et la sortie

Réseau du postérieur :

- z_dim : dimension de l'espace latent z_t
- actions_dim : dimension des actions prises en compte dans le réseau de postérieur, la plupart du temps il sera de 3, pour l'ensemble des métriques
- posterieur_dim_cachee : dimension de la couche cachée du réseau de postérieur entre la (transition + actions) et la sortie

Pour simplifier l'étude, les dimensions cachées sont égales (decodeur_dim_cachee = transition_dim_cachee = posterieur_dim_cachee). Ces « sous-réseaux » ne sont pas dupliqués d'un temps τ à un autre, ils sont partagés. Une autre façon de voir le réseau ci-dessous, centrée sur l'espace latent Z_τ :

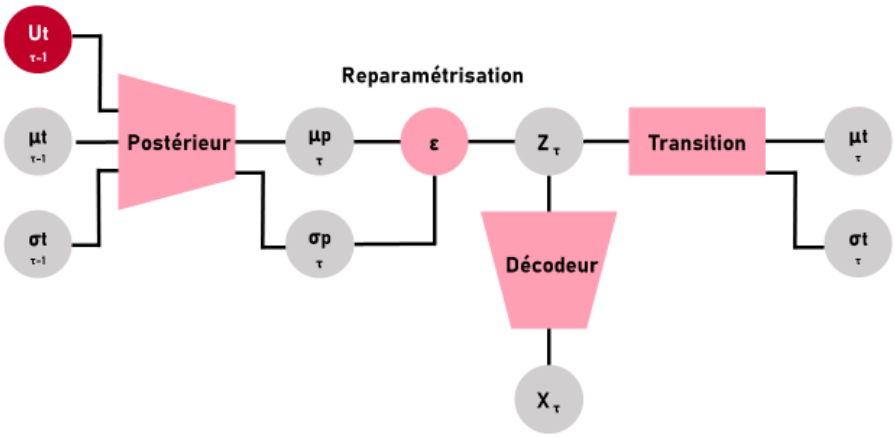


Figure 7 Schéma illustrant la récurrence du réseau et le partage des "sous-réseaux"

Maintenant que la disposition du modèle et que son fonctionnement a été éclaircit, il sera abordé dans la section suivante son entraînement.

C. Développement de la classe « Deep Kalman Filter »

Le programme est développé en suivant le principe de programmation orienté objet, avec des classes héritant du module de neural network de la librairie PyTorch. Chaque « sous-réseau » et donc développé au sein d'une classe du même nom, avec la réécriture de la fonction d'initialisation, et de la fonction forward pour créer les réseaux basés sur les paramètres nécessaires à leurs conceptions (cf. section VI. B.).

La classe Deep Kalman Filter initialise chaque « sous-réseau », ainsi que le premier espace latent. Cette classe contient également deux fonctions de calcul :

- Fonction de reparamétrisation pour générer des échantillons du nouvel espace latent selon un bruit ε
- Fonction de calcul de la divergence de Kullback-Leibler entre deux distributions normales

La fonction *Loss* est elle aussi redéfini pour correspondre à nos besoins : elle s'occupe de faire évoluer les données à travers le réseau de façon récursive pour toute la période donnée, de calculer l'erreur quadratique moyenne entre les prédictions et la réalité. Elle retourne les valeurs cumulées des divergences et des reconstructions sur une *epoch*.

À des fins de facilité d'exploitation, l'ensemble des paramètres sont rassemblés dans deux dictionnaires. Le premier nommé config_data, contient toutes les valeurs relatives à la création des jeux de données, le second nommé config_model contient toutes valeurs relatives à la préparation de l'entraînement.

D. Entraînement du modèle

Dans la phase d'entraînement pour chaque *epochs* et pour chaque pas de temps, le modèle calcule à l'aide des données du jeu d'actions du temps précédent :

- La divergence de Kullback-Leibler entre la distribution après réseau de transition et la distribution après réseau postérieur
- L'erreur quadratique moyenne entre l'observation prédite et l'observation réelle normalisée selon l'écart-type défini manuellement

Chaque calcul est cumulé sur la fenêtre de temps pour une *epoch*, et est combiné de façon linéaire pour former la fonction de perte du modèle :

$$Loss = reconstruction + \beta * KL$$

Le modèle rétro propage cette valeur au sein du réseau, en déduit ses nouveaux paramètres puis itère son optimisateur. De façon similaire, le modèle calcule en parallèle la valeur de la *loss* sur les données de validations, c'est un indicateur de plus pour évaluer l'évolution du comportement au cours des *epochs*.

Une fois le modèle entraîné, je l'utilise pour prédire des observations sur le jeu de test, j'en profiterai pour analyser les différents résultats obtenus et de différents paramètres dans le chapitre suivant.

VII. Analyses des performances sur des séries temporelles

Une fois que le modèle est entraîné, il est testé. Pour cela, le modèle fonctionne comme en phase d'entraînement à l'exception qu'ici il n'est pas question de calculer les divergences et l'erreur quadratique moyenne, mais de récolter les prédictions des observations à la suite des données d'actions que le modèle n'a jamais vues. Dans ce but, une nouvelle fonction nommée prediction_modele est introduite.

Pour être en mesure de comparer les projections des prédictions par rapport à la réalité, j'ai introduit une autre fonction nommée : projection_sur_test qui crée un graphique superposant ces deux jeux de données, et affichant les paramètres avec lesquels le modèle est entraîné.

Lors de la conception du réseau, il est arrivé que le modèle prédise toujours la vérité, ce qui paraissait suspect, et pour éclaircir le doute j'ai introduit un jeu de donnée d'actions aléatoire, et les prédictions étaient également correctes. Cette façon de tester le modèle par l'absurde m'a permis de comprendre un problème au niveau de la temporalité du réseau : les actions introduites dans le postérieur contenaient la valeur de l'observation suivante. Après que l'incident a été réglé, j'ai gardé ce test sur un jeu aléatoire comme un indicateur sur le bon fonctionnement du modèle.

Maintenant que le modèle est établi, entraîné, et dont le bon fonctionnement est assuré, il est désormais possible de comparer ses performances de prédictions. Dans la suite du projet, chaque modèle est testé pour les fenêtres de temps fixés ci-dessous. Pour chacune de ces fenêtres de temps, les modèles sont testés en faisant varier les paramètres ci-dessous :

- La fenêtre de temps des données : 1,3,6,9, 12 mois
- Le nombre *d'epochs* : 500, 1000, 10000
- La dimension de l'espace latent Z : 1,2,3,50
- La dimension des couches cachées des « sous-réseaux » : 50, 100
- Le facteur β dans le calcul de la *loss* : 0.1, 0.5, 1

Chaque modèle entraîné avec sa famille de paramètre puis testé, pour garder une trace de ses performances elles sont enregistrées dans le dossier « ./Graphs » du projet. Chaque modèle est enregistré dans un sous-dossier dont le nom est la composition des paramètres et leurs valeurs (ex : epochs_1000_mois_9_z_dim_2_dimension_sous_réseaux_50). Ce sous-dossier contient 4 graphiques :

- Le graphique combiné des valeurs des *loss* au cours des *epochs*
- Les prédictions sur le jeu de test
- Les prédictions du modèle sur les deux dernières années du jeu d'entraînement
- Les prédictions du modèle sur des jeux aléatoires

Dans les sections suivantes, je vais aborder différents aspects du modèle ainsi que ses performances, pour cela je vais exposer une sélection des graphiques du dossier « ./Graphs ».

A. Tendance générale des *loss*

Les valeurs des *loss* ont une tendance générale, quels que soient les paramètres du modèle : elles sont décroissantes au fur et à mesure des *epochs* puis atteignent un plateau avec un certain bruit. Elles sont décroissantes notamment parce que la constante choisie pour la valeur de sigma est négative. Cependant la *loss* de l'entraînement à tendance à décroître de façon bien plus « exponentielle » que la *loss* de validation.

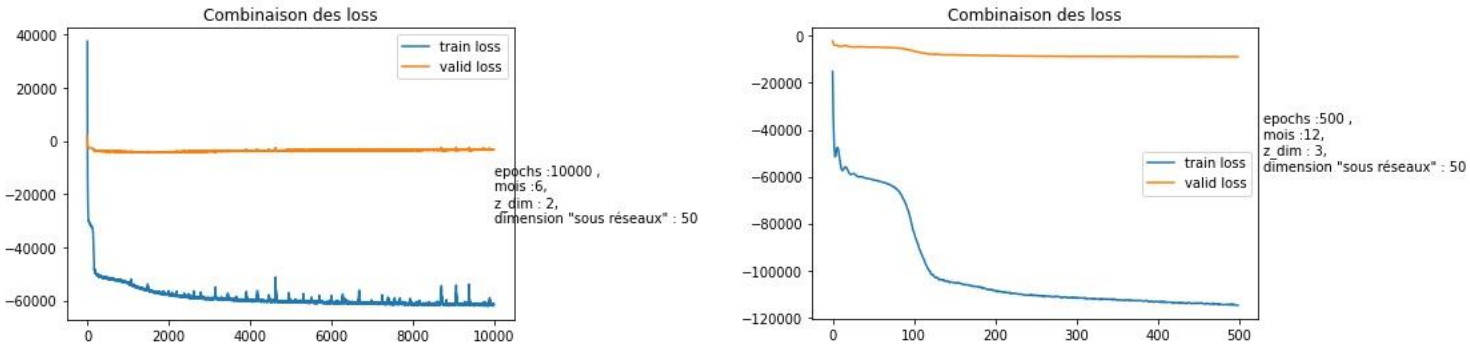


Figure 8 Comparaison des loss de deux modèles paramétrés différemment

À noter que les données sont des données d'un environnement continu, et donc nous n'avons pas de fonction qui converge vers 0, cela ne va pas pour autant dire que le modèle ne s'améliore pas au fur et à mesure des *epochs*. Au même titre que le test sur le jeu aléatoire, c'est un indicateur de bon fonctionnement de notre modèle.

B. Comparaison du modèle sur différentes fenêtres de temps

Voici ci-dessous les graphiques de comparaison pour une même famille de paramètre (epochs : 1000, z_dim : 2, dim_cc : 50, β : 1), afin d'identifier l'influence de la fenêtre de temps sur la performance (figure 9 à 12).

Un des premiers éléments à remarquer, est que la toutes les premières prédictions qui dépendent uniquement de l'espace latent, est assez décalé de la vérité, et il faut quelques itérations sur la fenêtre de temps pour que le modèle corrige sa trajectoire pour qu'elle soit plus « juste ». Ce qui soulève la question de la pertinence des paramètres d'initialisation du réseau.

Autre élément, les fenêtres de 1 et 3 mois ne contiennent pas suffisamment d'information pour que le modèle saisisse les variations « locales », autrement dit il détecte les pics de débit, mais entre les 2 les projections plafonnent à un minimum.

La fenêtre de 6 mois permet au modèle de prédire plus ou moins correctement les 2 pics, et épouse un peu plus la tendance générale entre les pics, cependant les variations restent lissées.

La fenêtre de 12 mois donne plus de fil à retordre au modèle puisque ce dernier écrase complètement le premier pic, et a du mal à déceler la tendance générale, elle est encore plus lisse que dans le cas de la fenêtre de 6 mois.

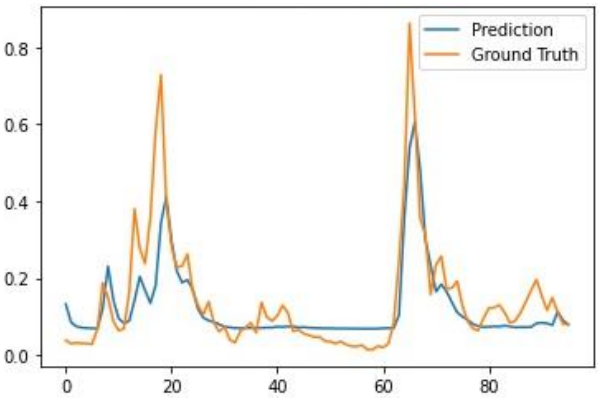


Figure 9 Prédiction pour une fenêtre de 1 mois

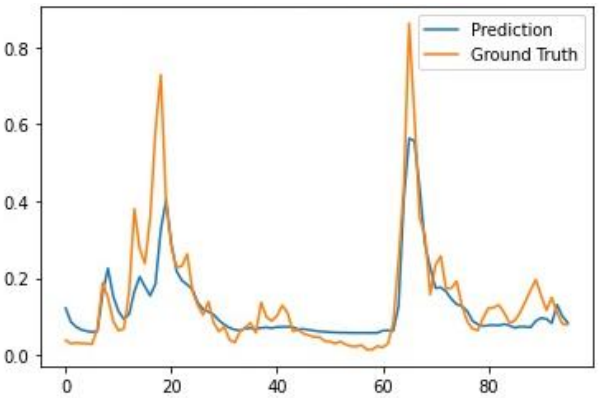
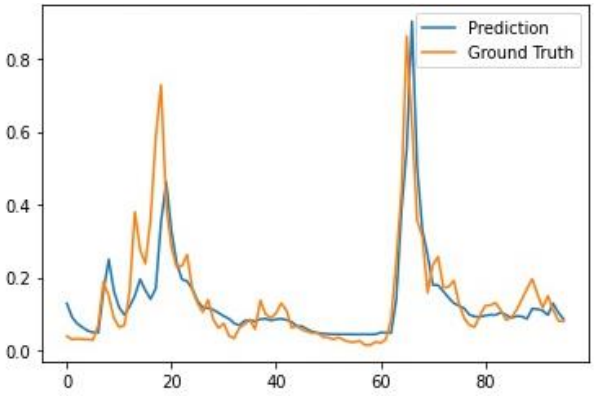


Figure 10 Prédiction pour une fenêtre de 3 mois



1. *Figure 11 Prédiction pour une fenêtre de 6 mois*

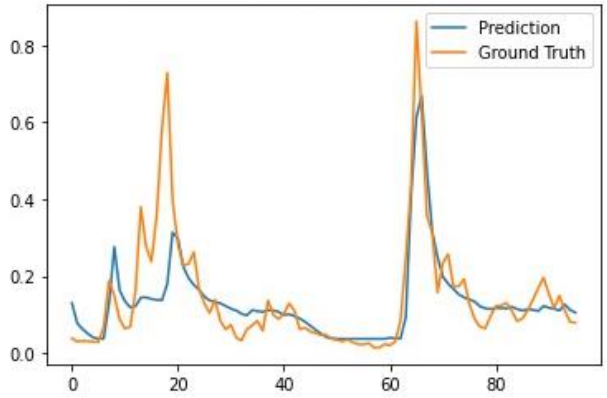


Figure 12 Prédiction pour une fenêtre de 12 mois

On constate de cette première étude que la fenêtre de temps à un impact quant aux performances de prédiction. Il est également à noter que pour un même nombre *d'epochs*, plus la fenêtre de temps est grande, plus il itère dans le réseau, ce qui a pour conséquence l'augmentation de la valeur du cumul de la *loss* , dans la configuration de cette sous partie la valeur de la *train loss* pour 1 mois est d'environ -8000 comparée a une valeur d'environ -100 000 pour une fenêtre de 12 mois, ce qui est linéairement proportionnel.

C'est donc naturellement que je vais comparer dans la section suivante l'influence de ce paramètre.

C. Comparaison du modèle pour différents *epochs*

Voici ci-dessous les graphiques de comparaison pour la famille de paramètre suivante : epochs : 500, 1000 et 10 000, z_dim : 2, dim_cc : 50, β : 1, et la fenêtre de temps est fixé à 6 mois.

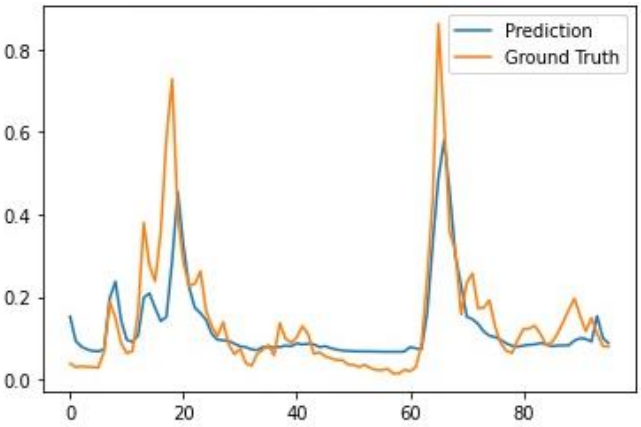


Figure 13 Prédiction pour 500 epochs

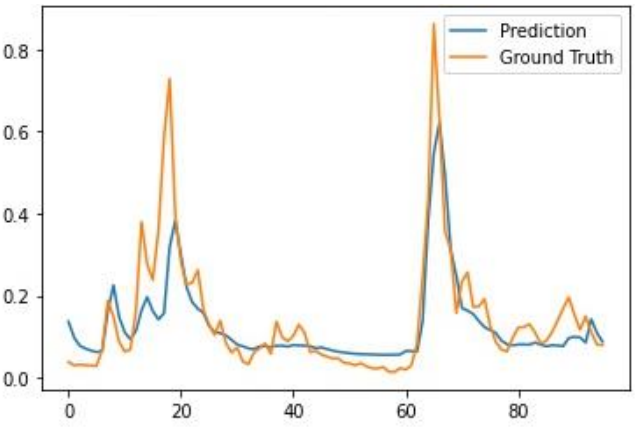


Figure 14 Prédiction pour 1000 epochs

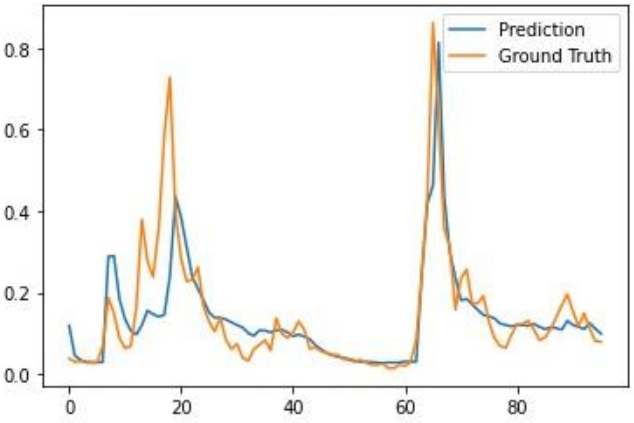


Figure 15 Prédiction pour 5000 epochs

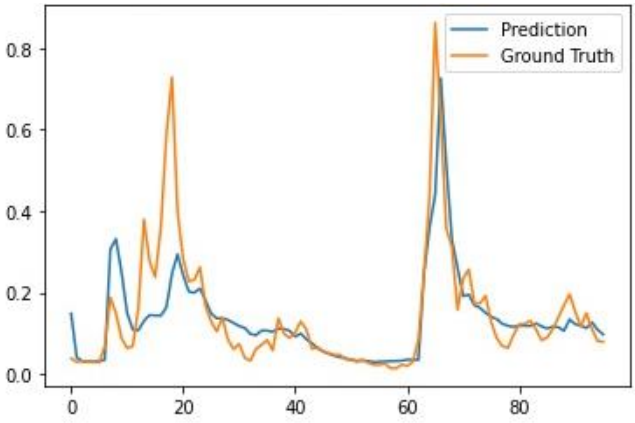


Figure 16 Prédiction pour 10 000 epochs

Entre 500 et 1000 *epochs* les différences sont vraiment mineures, les pics sont légèrement mieux définis et la tendance générale des prédictions est tout aussi légèrement plus proche de la vérité.

En revanche avec un nombre d'epochs très élevé, le problème de lissage des pics est de nouveau présent, les prédictions sont similaires à la fenêtre de 12 mois avec 1000 epochs de la section précédente. Il y a donc une forme de saturation d'apprentissage à partir d'un certain seuil, qui est intrinsèquement lié à la largeur de la fenêtre de temps des données. Définir cette corrélation permettrait d'optimiser les performances du modèle et d'être en mesure de sélectionner une fenêtre de temps optimal pour les prédictions futures.

Un autre élément qui est intéressant de faire varier est le nombre de paramètres du modèle.

D. Comparaison avec des réseaux ayant plus de paramètres

Voici ci-dessous les graphiques de comparaison pour la famille de paramètre suivante : epochs : 1000, z_dim : 2, dim_cc : 50 et 100, β : 1, et la fenêtre de temps est fixé à 6 mois.

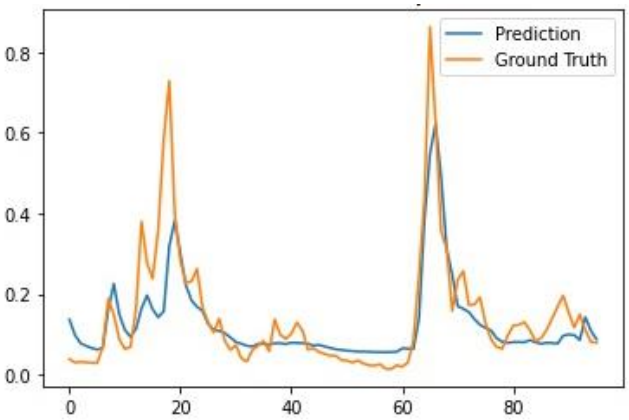


Figure 17 Prédiction pour 50 neurones en couche cachée

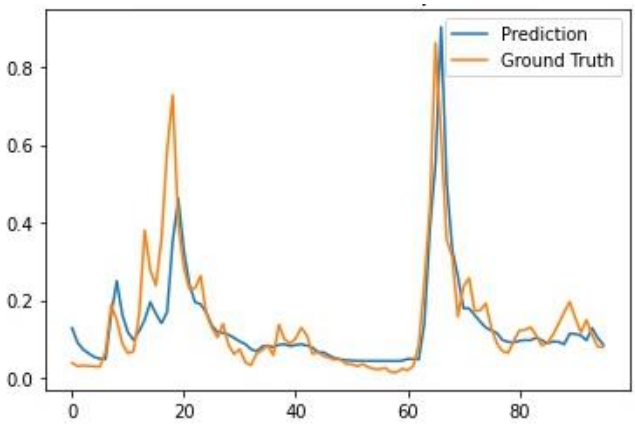


Figure 18 Prédiction pour 100 neurones en couche cachée

Bien que l'erreur quadratique moyenne soit en faveur du modèle avec 50 neurones, le modèle de 100 neurones fait de meilleures prédictions sur la période entre les deux pics, il est plus proche des variations « locales ».

Voici un autre ensemble de graphiques avec la famille de paramètre suivante : epochs : 500, z_dim : 2 et 3, dim_cc : 50, β : 1, et la fenêtre de temps est fixé à 6 mois.

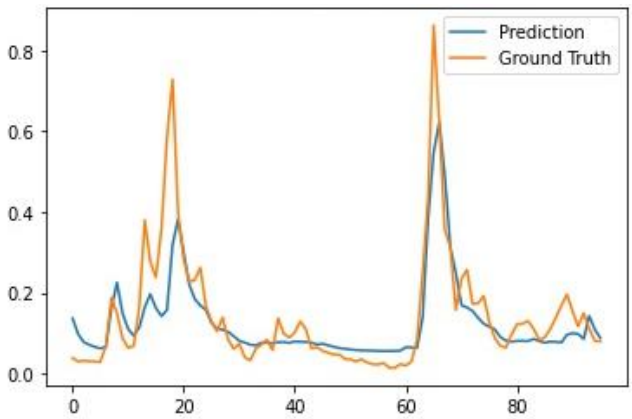


Figure 19 Prédiction pour un espace latent de dimension 2

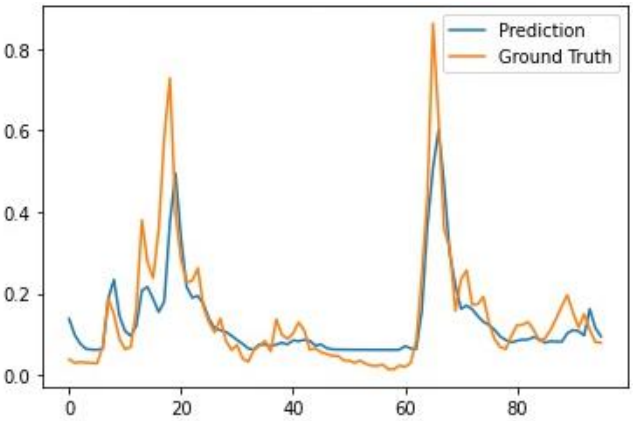


Figure 20 Prédiction pour un espace latent de dimension 3

Bien qu'augmenter l'espace latent n'est pas favorable dans le cas de VAE (Variational Auto-Encoder), car il augmente la quantité de calcul, et ne condense pas les informations de la même manière, il peut être intéressant de le faire dans le cas où cet espace est saturé et ne parvient pas à rétablir correctement l'information. Ce qui n'est pas le cas ici, et ce qui conclut qu'un espace de dimension 2 est suffisant dans cette étude.

L'augmentation des paramètres peut être un élément intéressant dans la recherche d'optimisation du modèle, cependant cette section montre qu'il serait plus intéressant d'augmenter la taille des « sous-réseaux » du modèle plutôt que son espace latent.

E. Comparaison avec différentes valeurs de β

Voici ci-dessous les graphiques de comparaison pour la famille de paramètre suivante : epochs : 500, z_dim : 2, dim_cc : 50, β : 0.1, 0.5 et 1, et la fenêtre de temps est fixé à 6 mois.

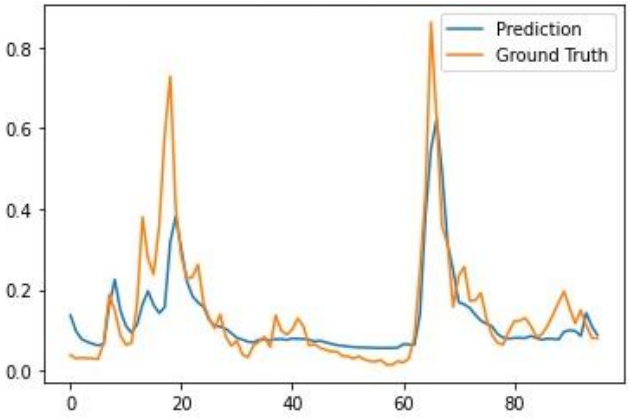


Figure 21 Prédiction pour un facteur β = 1

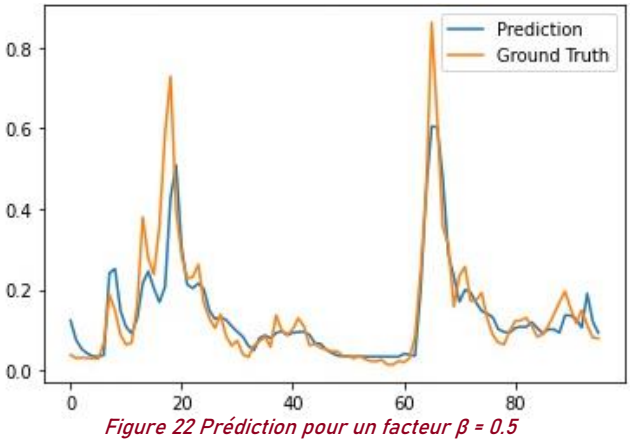


Figure 22 Prédiction pour un facteur β = 0.5

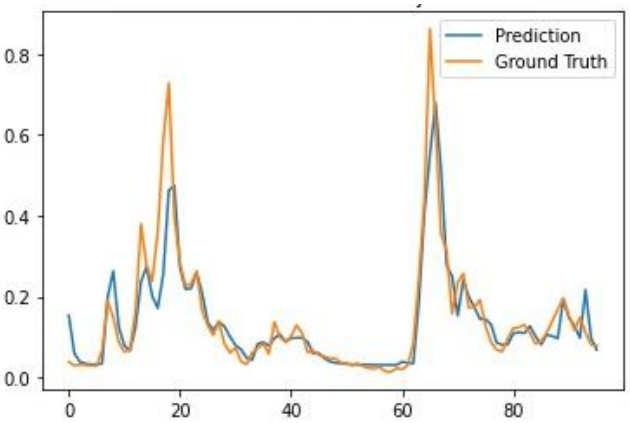


Figure 23 Prédiction pour un facteur β = 0.1

Pour rappel, β est le facteur de prise en compte de la divergence de Kullback-Leibler entre la distribution après le réseau de transition et après le réseau postérieur. Plus ce dernier est petit, plus la valeur de la loss se base sur la reconstruction du signal.

On remarque que pour un facteur 10, la prédiction du débit s'est améliorée de façon significative avec une erreur quadratique moyenne de $4{,}8e^{-3}$. Les prédictions au niveau des pics sont correctes, et les variations locales sont beaucoup mieux prédites. C'est la meilleure amélioration de toutes celles vues au cours de l'étude.

Ce qui entraîne donc une réflexion sur les choix qui sont faits lors de la conception du modèle, notamment sur le fait de considérer des distributions normales, ou bien même sur la nature du réseau : un auto-encoder, ou un variational auto-encoder ne suffisent-ils pas pour essayer de prédire le débit futur ?

F. Comparaison du modèle avec des travaux précédents

Lors de précédents travaux d'introduction à l'intelligence artificielle, j'ai développé des réseaux de neurones « simples » dans le même but que celui de prédire le débit sortant, ci-dessous la prédiction d'un réseau à une couche caché de 128 neurones :

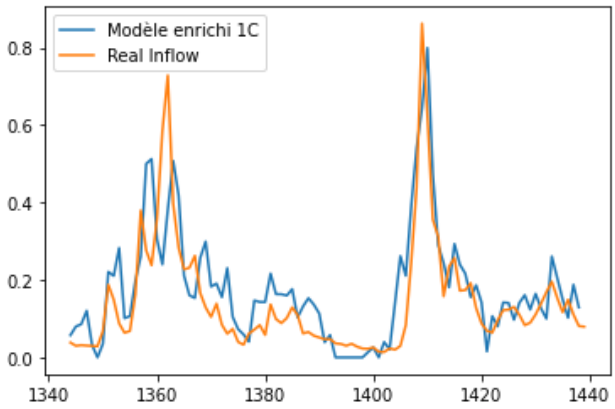


Figure 24 Prédiction d'un réseau à une couche cachée de 128 neurones

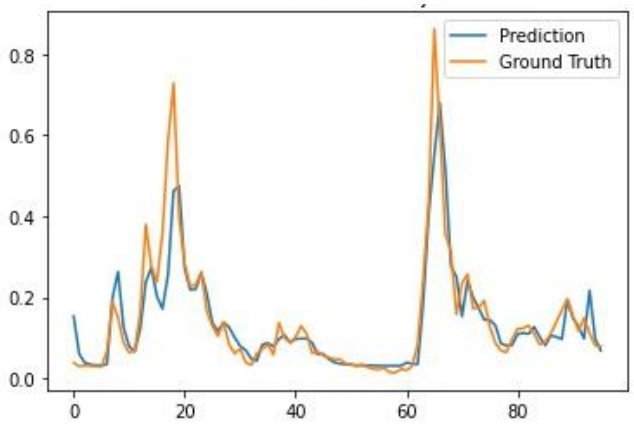


Figure 25 Meilleure prédiction du modèle Deep Kalman Filter

L'erreur moyenne quadratique est de $4.48e^{-2}$, et le réseau a beaucoup de mal à suivre la tendance générale, les prédictions sont très bruitées. Avec le modèle décrit dans ce rapport et grâce à sa complexité, il y a un facteur 10 sur la valeur de la quadratique moyenne, et une prédiction qui est plus performante sur les variations « locales ».

VIII. Conclusions

Dans les données étudiées, il existe une corrélation physique entre la pluie, la fonte des neiges et le débit entrant dans le lac, la relation entre chaque métrique est intuitive. Cependant capturer cette corrélation au sein d'un modèle est plus difficile.

Dans l'ensemble des réseaux construit dans cette étude, le modèle a réussi à mettre à profit les données sur les relevés de pluie et de fonte des neiges dans ses prédictions. Le principe des filtres de Kalman avec la mise à jour récursive combiné à la puissance de modélisation des réseaux neuronaux à démontrer une certaine efficacité à modéliser la corrélation physique entre les différentes métriques.

Le réseau conçu ici n'est qu'un des réseaux proposés par le papier, en effet ce dernier suggère pour les « sous-réseaux » l'utilisation de modèles plus complexes, avec des MLP prenant en compte les temporalités adjacentes, ou bien encore l'utilisation de réseau récurrent pour mieux exploiter la structure temporelle des données, autant de modèles dont il serait intéressant de comparer les comportements face à cette série.

Bien que de nombreuses pistes ont été explorées dans cette étude, il existe encore un certain nombre d'éléments qui enrichiraient la réflexion sur les prédictions temporelles, en voici quelques-uns :

- Changer la formule de calcul d'erreur de reconstruction, car les variations locales entre 2 pics sont souvent mal reconstruites
- Envisager d'autres distributions que les distributions normales
- Optimiser l'initialisation du réseau, et notamment de son espace latent et de l'écart-type σ
- Faire prédire au décodeur plusieurs observations à la suite, cela permettra de mieux intégrer les tendances, et éventuellement d'augmenter la performance du modèle sur les variations locales

Une amélioration du modèle concerne le choix de ne pas mélanger les données lors de la construction des jeux d'actions, ceci afin de mesurer la déviation du modèle au travers du temps. Cependant si ce phénomène est négligé, il serait intéressant de voir si le modèle se comporte différemment sur le jeu de test. Dans la continuité de la réflexion précédente, et pour amener le modèle à l'échelle supérieur, il faudrait introduire des itérations par batch.

Comme ouverture, voici d'autres réseaux / méthodes avec lesquels il serait intéressant de comparer les performances :

- Un prétraitement des données avec des réseaux convolutifs
- L'utilisation des réseaux de Long Short Term Memory
- Réseaux neuronaux récurrents