INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Department of Applied Mathematics

AI Frameworks

Défi IA 2022

Benjamin Viard

Yannick Selly

Martin Magnan

Florian Lajugie

Eva Membrado

5MA - Group A

Contents

1 Introduction

Ce travail se place dans le contexte de la 6e édition du Défi IA, une compétition Kaggle où peuvent s'affronter des étudiants de différentes écoles francophones et des agents Météo-France. Cette année, l'objectif de ce défi est de prévoir les précipitations journalières cumulées le jour J aux les stations d'observation.

1.1 Présentation des données

Deux types de données sont mises à disposition : des données d'observation et des données de prévisions.

Les données d'observation correspondent aux mesures faites au niveau des 325 stations réparties sur la zone (figure 1) au temps t, elles sont disponibles au format csv et sont de la forme : date (une par heure), number_sta (numéro de station), paramètres météorologiques (vent, température, humidité, précipitations) et id (identifiant de la station). Les observations sont disponibles le jour précédent la prévision (X_station). L'évaluation des méthodes employées s'effectuent avec les observations de précipations cumulées sur le jour de prévisions: elles sont données pour effectuer l'entraînement (Y_train) mais non disponibles pour le test (présentes dans Kaggle).

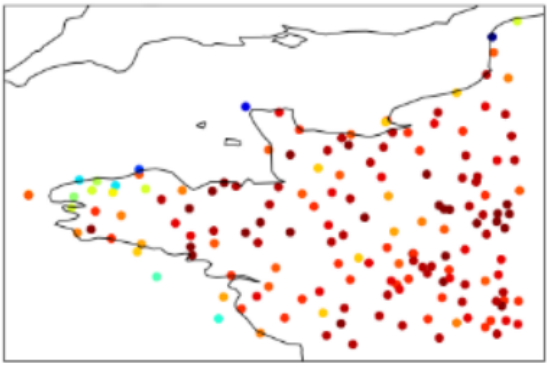

Figure 1: Répartition des stations sur la zone d'étude

Les données de prévisions correspondent aux prévisions des modèles météorologiques (AROME, ARPEGE). Dans le cadre de notre étude, seules les données issues de AROME (appelée par la dénomination X_forecast par la suite), qui possède une meilleure résolution spatiale qu'ARPEGE, sont utilisées. Ces données, disponibles au format netCDF, sont de la forme : latitude, longitude, forecast time, température à 2m, humidité à 2m, température de point de rosée à 2m, caractéristique du vent à 10m (FF, DD, U, V), pression au niveau de la mer, précipitations.

A partir de ces données brutes, chaque équipe doit, par ses propres moyens répondre à la problématique posée. Dans ce rapport, la démarche et les travaux effectués par notre équipe sont présentés avec: dans un premier temps le déroulement de notre pre-processing, puis le choix du modèle et des techniques d'apprentissage et enfin le calcul du score et les résultats.

2 Pre-processing

2.1 Traitement et mise en forme des données

Le premier point de notre travail est de modifier les données brutes mises à disposition pour qu'elles puissent être utilisables dans nos futurs modèles.

En premier lieu, les données d'observations et de prévisions sont récupérées à partir des fichiers csv et netcdf pour être mises au même format, dans des tenseurs afin de les rendre comparables (les données de prévisions sont interpolées sur les stations avec une méthode spécifiée dans la section suivante).

Les tenseurs sont ensuite comparés afin d'identifier les combinaisons de dates et stations pour lesquelles soit les observations ou soit les prévisions venaient à manquer. Une intersection entre ces deux sources de données est effectuée pour éliminer ces combinaisons de notre jeu de données : dès la génèse de notre projet l'idée était d'utiliser un réseau de neurones prenant à la fois les observations et les prévisions d'où l'intersection.

La dernière étape consiste à normaliser les données afin d'éviter un déséquilibre entre les différentes données, en effet différents paramètres météorologiques d'unité SI et d'ordres de grandeur différents sont pris en compte dans nos données.

2.2 Interpolation de $X\_forecast$

Pour les données modèle, un traitement supplémentaire a dû être réalisé. En effet, les prévisions ne sont disponibles que sur les points de grille d'AROME et non aux coordonnées des stations ; de ce fait, une interpolation des données est nécessaire.

Pour notre étude, une interpolation par méthode des proches voisins a été choisie. Les coordonnées de chaque station de mesure permettent d'identifier les points de grille les plus proches de celle-ci. Les distances entre le point de la station et les points de grille voisins sont calculées de cette manière :

$$dist = \frac{1}{a^2} + \frac{1}{b^2} + \frac{1}{c^2} + \frac{1}{d^2}$$
$$prev = (\frac{prevA}{a^2} + \frac{prevB}{b^2} + \frac{prevC}{c^2} + \frac{prevD}{d^2})/dist$$

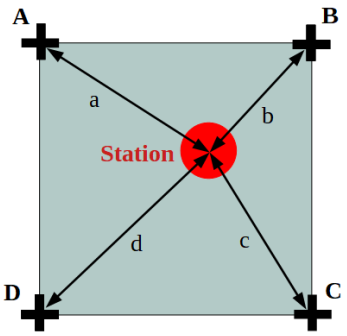

Figure 2: Schéma de la méthode d'interpolation spatiale choisie

De cette manière, les prévisions sont maintenant disponibles aux stations d'observation. Les observations sont alors associées aux prévisions aux stations pour effectuer une régression avec les modèles statistiques qui sont présentés par la suite.

3 Implémentation des modèles et analyse des résultats

Afin de répondre à la problématique, plusieurs modèles ont été testés : l'étude s'est portée sur différents réseaux de neurones.

Comme évoqué précédemment, dès la génèse de notre projet l'objectif était de développer un réseau de neurone utilisant à la fois les observations et les prévisions. Le formatage des données a d'ailleurs été fait dans ce sens. De plus, les réseaux de neurones sont relativement modulables et permettent de diversifier les expérimentations.

L'objectif de ce projet est d'approcher au mieux la réalité observée, c'est-à-dire minimiser l'écart entre les prédictions et la réalité. Afin de se comparer à l'existant, des références ont été des références à battre afin de nous assurer que les différents résultats des modèles que nous produiront auront une réelle plus-value et seront capables de faire mieux qu'une référence naïve.

Parmi les références, il y en avait une qui nous était imposé et qui nous devions battre afin de remplir le cahier des charges du défi IA. Cette dernière était la $baseline\_forecast$ et correspondait aux prévisions de cumuls de précipitations sur 24h d'un modèle météo. Celle-ci a obtenu un score de 40,6 après l'ultime mise à jour du classement sur kaggle, qui clôtura ainsi officiellement le défi IA.

A cette première référence, nous en avons ajouté une seconde qui correspondait à un tenseur entièrement vide, rempli de zéros, que nous avons soumis sur kaggle, afin de voir quel résultat obtiendrait une prévision simpliste qui consisterait à dire qu'il ne pleut jamais, soit un cumul constant de 0 mm tous les jours. Les résultats obtenus sur cette référence furent de 34,7 pour le $private\_score$, et de 29,5 pour le $public\_score$, soit une moyenne de 32,1. Ces résultats surprenant au premier abord, nous indique en effet qu'une prévision consistant à dire qu'il ne pleut jamais, obtient un meilleur score de MAPE sur kaggle que la $baseline\_forecast$. Après analyse, ces résultats sont en réalité assez cohérents, car la majorité des échantillons de cumul journalier de précipitation sont nuls ou très faibles.

Ce résultat met en revanche en évidence deux choses. La première, est qu'aucun score de MAPE obtenu en sortie de modèle ne pourra être satisfaisant si ce dernier est en moyenne supérieur à 32,1. La seconde, est le manque de pertinence de l'estimateur statistique du score de MAPE pour vérifier la qualité d'un système de prévision sur un cumul de précipitation journalier.

3.1 Réseaux de neurones entièrement connectés

3.1.1 Expériences avec $X\_station$ uniquement

Nous avons débuté nos expériences numériques avec les réseaux de neurones les plus simples qui existe, soient les réseaux de neurones entièrement connectés avec des couches Dense.

Notre première expérience a consisté à ne prendre comme données que les observations des cumuls de précipitation horaire dans $X\_station$, et de les rentrer dans un réseau de neurones constitué exclusivement d'une couche cachée Dense (figure 3), portant le nombre de coefficients de ce réseau à 833.

```
def sequential_1():
  inputs = Input(shape=(24), name='inputs')
  x = Dense(32, activation='relu', name='dense_1')(inputs)
  outputs = Dense(1, activation='relu', name='dense_2')(x)
  return Model(inputs, outputs, name='Sequential_1')
```

```
Model: "Sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inputs (InputLayer)         [(None, 24)]              0

 dense_1 (Dense)             (None, 32)                800

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 833
Trainable params: 833
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Architecture d'un premier réseau Dense

Après entraînement du réseau avec un *batch_size* de 200 (qui sera la valeur que nous conserveront par défaut durant toutes les expériences de réseaux de neurones) sur 20 epochs, nous avons obtenu un score de MAPE en interne par validation croisée (sur 20% des échantillons d'entraînement) de 32,7.

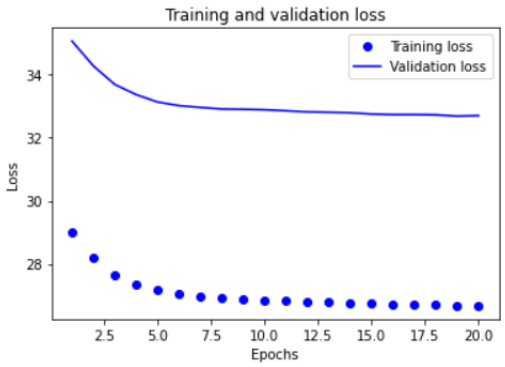

Figure 4: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un premier réseau Dense

A partir de ce premier résultat prometteur en interne supérieur à la *baseline_forecast*, nous avons décidé dans un second temps d'ajouter aux données d'entrées de ce même réseau les observations de température, augmentant ainsi le nombre de coefficients entraînables à 1601. Les résultats obtenus en validation croisée furent très légérement supérieurs avec une MAPE à 32,5.

Les courbes de validation de la fonction loss, comme illutré sur la figure 4, étant monotones décroissantes pour ces deux premières expériences, il a donc été choisi d'augmenter la capacité du réseau, pour voir si sur les données d'observations horaires de précipitations et de températures, des informations supplémentaires pouvaient être extraites afin d'améliorer le score de prévision. Pour ce faire, un deuxième réseau entièrement connecté a été implémenté, constitué cette fois de 2 couches cachées Dense, pour un total de poids entraînables de plus de 20 000. Après entraînement sur 20 epochs, le score de MAPE en validation croisée en est resté inchangé, stagnant à 32,5, tandis qu'on observait cette fois sur la courbe de validation un début de surajustement aux données (overfitting), comme illustré sur la figure 5.

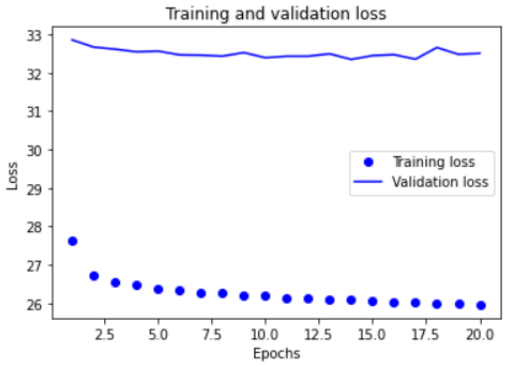

Figure 5: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un deuxième réseau Dense

Afin de tester au maximum les possibilités d'un réseau entièrement connecté sur nos séquences de données d'observations, nous avons implémenté un troisième et dernier réseau avec 5 couches cachées Dense et des couches cachées de Dropout à 20%, pour un nombre total de coefficients d'environ 38 000. Le résultat de MAPE à l'issue de 20 epochs fut cette fois-ci légérement mieux que les précédents, avec un score de 32,1, mais sans faire de réel saut qualitatif dans les performances de prévisions.

Afin de clôturer cette première phase d'expérimentation avec seulement des données d'observation ($X\_station$), nous avons soumis les résultats de test de ces 4 expériences sur kaggle. Les résultats furent moins bons sur kaggle qu'en interne avec la validation croisée, avec des scores de MAPE moyen (entre $private\_score$ et $public\_score$) allant de 38,9 à 40,3. Ces premiers résultats sur kaggle sont satisfaisants car ils sont tous inférieurs à la $baseline\_forecast$ qui est de 40,6, mais demeurent cependant insufisants, car faisant moins bien que la référence nulle de 32,1.

3.1.2 Expériences avec $X\_station$, $X\_forecast$ et prédicteurs constants

Dans cette partie nous avons gardé la structure du dernier réseau de neurones entièrement connecté qui a été implémenté dans la section précédente avec 5 couches Dense, et nous avons modifié et ajouté de nouvelles données en entrée du réseau. Dans la suite des expériences, tous les entraînements seront dorénavant menés sur 10 epochs. Quatre nouvelles expériences ont ainsi été menées, les entrées et les résultats de MAPE en validation croisée sont consignés dans le tableau ci-dessous (figure 6), faisant varier le nombre de paramètres entraînables du réseau de 49 000 à 55 000:

inputs	MAPE en validation croisée
X_forecast	28,2
X_forecast + prédicteurs_constants	29,7
X_forecast + X_station	27,3
X_forecast + X_station + prédicteurs constants	27,9

Figure 6: Résultats des expériences menées avec différentes sources de données sur un réseau de neurones entièrement connecté

On observe une nette amélioration des scores de MAPE avec des scores tous inférieurs à 30 et nettement inférieur à l'expérience où on ne rentrait que les données de $X\_station$ dans le réseau, qui était de 32,1. On note ainsi une réelle plus-value à l'ajout de données supplémentaires dans le réseau de neurones, notamment le fait de combiner les observations et les prévisions. Cependant on remarque également une dégradation systématique du score de MAPE lorsque l'on ajoute les données des prédicteurs constants, augmentant le score de 0,5 à 1,6.

Après soumission des prévisions de ces différentes expériences pour l'échantillon de test sur kaggle, on obtient comme en interne une franche amélioration de la MAPE avec des scores moyens allant de 26,3 à 29,3, soient des résultats nettement meilleurs que la $baseline\_forecast$ et également meilleurs que la référence nulle. Le meilleur résultat des quatre expériences étant celui où on envoie en entrée du réseau de neurones que le $X\_forecast$.

3.2 Réseaux de neurones récurrents (RNN) avec une source de données

3.2.1 Expériences avec $X\_station$ uniquement

La première approche avec un réseau entièrement connecté nous a donné comme meilleur résultat un score de MAPE sur kaggle de 26,3. Cependant, cette manière de procéder a entraîné un applatissement des séries temporelles des différents prédicteurs, supprimant la notion de temps dans les données d'entrées. Nous avons donc dans un second temps, choisi de travailler avec un modèle RNN (Recurrent Neural Network) afin de pouvoir récupérer les informations de structures temporelle au sein des données. En effet, Un RNN est un réseau de neurones qui permet de traiter des séquences temporelles et conserve la mémoire des données du passé pour prédire des séquences de données dans le futur proche grâce à l'utilisation d'une boucle interne implémentée au sein de la couche (figure 7). Ce type de modèle se trouve donc adapté à notre problème.

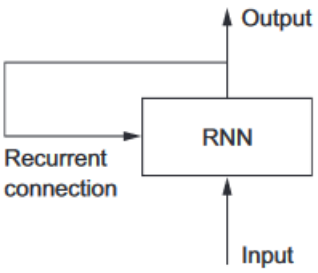

Figure 7: Schéma principe du RNN

Le but de cette section est donc de voir si un réseau récurrent (RNN) prenant en compte l'ordre temporel des données contrairement à la première approche, peut nous fournir des scores de MAPE inférieurs à 26,3.

La couche récurrente choisie tout au long de ce projet a été la couche LSTM (Long short term memory). Nous avons commencé par expérimenter l'implémentation d'un LSTM uniquement sur les données d'observation de $X\_station$.

La première expérience avec une couche récurrente a été celle d'un LSTM simple en couche cachée associé à une couche Dense de 8 unités en sortie du réseau (figure 8) avec 4 753 coefficients entraînables.

```python
def lstm_1():
  inputs = Input(shape=(24, 2), name='encoder_inputs')

  x = LSTM(32, name='lstm')(inputs)
  x = Dense(8, activation='relu')(x)
  outputs = Dense(1, activation='relu')(x)

  return Model(inputs, outputs, name='lstm_1')
```

Figure 8: Architecture d'un premier réseau récurrent

Les résultats sur kaggle nous donne un score de 31,2, ce qui est assez décevant comparativement aux meilleurs résultats obtenus avec les réseaux entièrement connecté. On note cependant une monotonie de la courbe de validation sur la figure 9, indiquant sans doute un manque d'entraînement ou la possibilité de complexifier le réseau en ajoutant des coefficients.

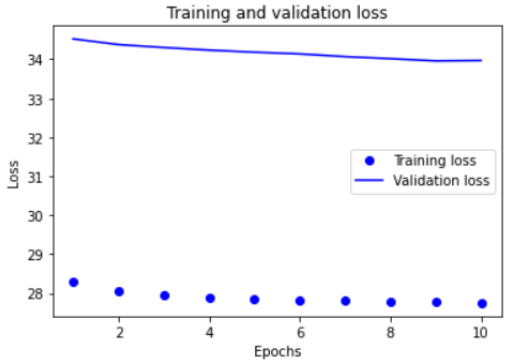

Figure 9: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un premier réseau récurrent

Dans une deuxième expérience, on a repris une couche LSTM pour laquelle on a doublé les unités (soient 64 unités), portant le réseau à 17 681 paramètres entraînables. On obtient comme score sur kaggle une MAPE de 31,4, ce qui est moins bien que la première expérience où on avait 4 fois moins de paramètres dans le réseau. On observe également toujours une monotonie de la courbe de validation indiquant la possibilité de continuer à complexifier le réseau. En effet, il est généralement pertinent d'augmenter la capacité d'un réseau jusqu'à ce que le sur-ajustement devienne le principal obstacle à l'apprentissage. L'augmentation de la capacité d'un réseau se fait soit en augmentant le nombre d'unités par couche, comme nous venons de le faire au travers de cette expérience, soit en ajoutant des couches supplémentaires (Chollet [1]).

Comme notre modèle n'est toujours pas en sur-ajustement, mais qu'il semble stagner au niveau des performances, on evisage cette fois d'augmenter la capacité du réseau en ajoutant une couche LSTM supplémentaire (voir figure 10). L'empilement de couches récurrentes est un bon moyen de construire des réseaux récurrents plus performants.

Afin de pouvoir entraîner les poids de ce réseau, il a été nécessaire de modifier le $kernel\_initializer$ qui était à $glorot\_uniform$ par défaut, et de le changer avec $he\_uniform$. La figure 11 présente les résultats. On peut voir que la couche ajoutée n'améliore pas les résultats en validation croisée, ce qui est confirmé avec le score sur kaggle qui se dégrade à 33,5. On peut en conclure que le modèle n'étant pas encore sur-ajusté d'après la courbe de validation sur la figure 11, qu'il serait possible d'envisager d'augmenter la taille des couches.

```python
def lstm_3():
  inputs = Input(shape=(24, 2), name='encoder_inputs')

  x = LSTM(32, kernel_initializer='he_uniform', return_sequences=True, name='lstm')(inputs)
  x = LSTM(64, activation='relu', kernel_initializer='he_uniform', name='lstm_2')(x)
  x = Dense(8, activation='relu')(x)

  outputs = Dense(1, activation='relu')(x)

  return Model(inputs, outputs, name='lstm_3')
```

Figure 10: Architecture d'un réseau récurrent constitué d'une pile de couches LSTM

Néanmoins le coût de calcul que cela engendrerait serait non négligeable, d'autant que l'augmentation de la capacité du réseau jusqu'à présent n'a aucunement améliorer les performances de MAPE.

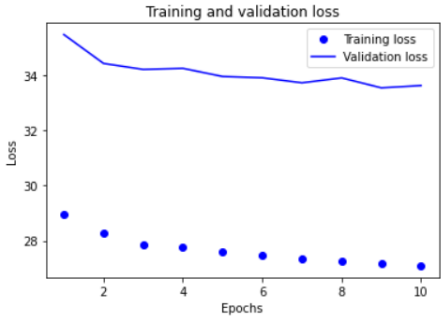

Figure 11: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un réseau constitué d'une pile de couches LSTM

On a donc décidé à l'issue de ces expériences, de modifier les données en entrée du réseau et de mettre à la place des observations, les prévisions modèle.

3.2.2 Expériences avec $X\_forecast$ uniquement

Dans cette partie, on a repris une simple couche LSTM pour le réseau de neurones dans lequel on a envoyé en entrée le $X\_forecast$. Dans une première expérience, on a fixé à 32 unités la couche LSTM comme sur la figure 8, pour obtenir un réseau à 5 665 coefficients. On choisit dans cette expérience que le réseau nous retourne toute la séquence en fixant l'argument $return\_sequences = True$, car on considère qu'en prenant en entrée du réseau le $X\_forecast$, celui-ci réalisera un calibrage des prévisions pour obtenir le cumul journalier de précipitation. Ce cumul correspond en effet à la même journée que les prévisions, contrairement aux observations qui sont décalées d'une journée par rapport au label. Afin de transformer la séquence de sortie de la couche LSTM en une seule valeur de prévision, on implémente en bout de réseau des couches TimeDistributed afin de traiter avec un réseau Dense chacune des 24 prévisions de la séquence en parallèle (figure 12).

```python
def lstm_5():
  decoder_inputs = Input(shape=(24, 7), name='decoder_inputs')

  decoder_lstm = LSTM(32, return_sequences=True, name='lstm')(decoder_inputs)
  decoder_lstm = TimeDistributed(Dense(16, activation='relu'))(decoder_lstm)
  decoder_lstm = TimeDistributed(Dense(1, activation='relu'))(decoder_lstm)
  decoder_lstm = Flatten()(decoder_lstm)

  decoder_outputs = sum(decoder_lstm, axis=1, keepdims=True)

  return Model(decoder_inputs, decoder_outputs, name='LSTM_5')
```

Figure 12: Architecture d'un réseau récurrent prenant en entrée les $X\_forecast$

Après entraînement, on a obtenu sur kaggle un score moyen de 26,9. C'est nettement mieux avec les prévisions en entrée de ce réseau que lorsqu'on entrait les observations dans la section précédente. Cependant on ne fait pas mieux que notre meilleur réseau entièrement connecté qui avait obtenu un score de 26,3.

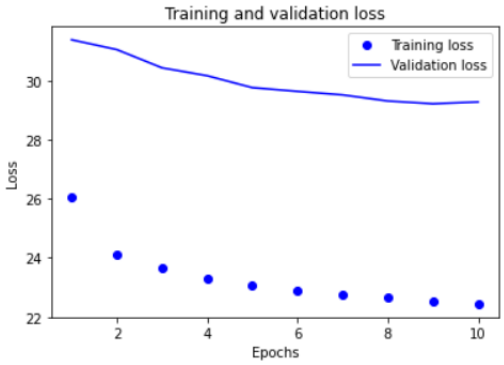

Figure 13: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un réseau récurrent prenant en entrée les $X\_forecast$

Comme avec les $X\_station$, on obtient une courbe de validation monotone, nous incitant à augmenter la capacité du réseau (figure 13).

Dans une autre expérience, on a donc augmenté à 64 le nombre d'unités de la couche LSTM, portant le nombre de coefficients du réseau à près de 20 000. Après entraînement et soumission sur kaggle, on tombe sur un score moyen de MAPE de 26,6. On fait légèrement mieux que précédemment, mais on ne parvient toujours pas à battre notre meilleur réseau Dense, avec en revanche un coût en temps de calcul nettement supérieur.

Les couches récurrentes ne semblent pas apporter une réelle plus-value pour notre problème, et tendent à faire juste aussi bien qu'un réseau entièrement connecté. On observe également tout comme avec des couches Dense, une meilleure prévision avec le $X\_forecast$ en entrée plutôt que le $X\_station$. Cela est sûrement dû au fait que le $X\_forecast$ dispose de 7 prédicteurs contrairement au $X\_station$ qui n'en possède que 2, apportant de fait beaucoup plus d'informations utiles pour le réseau.

Dans la partie suivante, nous avons cependant poursuivi avec l'approche des réseaux récurrents mais en apportant deux sources de données en entrée, à savoir les prévisions et les observations afin de voir si la combinaison des deux permettra d'obtenir un gain substantiel dans nos scores de prévision.

3.3 Réseaux de neurones récurrents (RNN) avec deux sources de données

3.3.1 Approche avec 2 LSTM en série

Dans cette section, on a choisi d'implémenter un réseau de neurones récurrent avec une entrée multi-sources. Cette approche nous a été inspiré par un article (T. Lin et al. [2]) sur la prévision des impacts de foudre avec des couches Conv2DLSTM avec des données d'entrées multi-sources, à savoir dans un premier temps les observations du prédictand qui ont été récupérées dans le passé afin de produire une première prévision, qui sera dans un second temps enrichie, voire calibrée, par des prévisions modèles de différents prédicteurs pour produire finalement une prévision plus proche de la réalité, à de multiples échéances. Cette vision est illustrée sur la figure 14 ci-dessous, extraite de l'article précédemment mentionné.

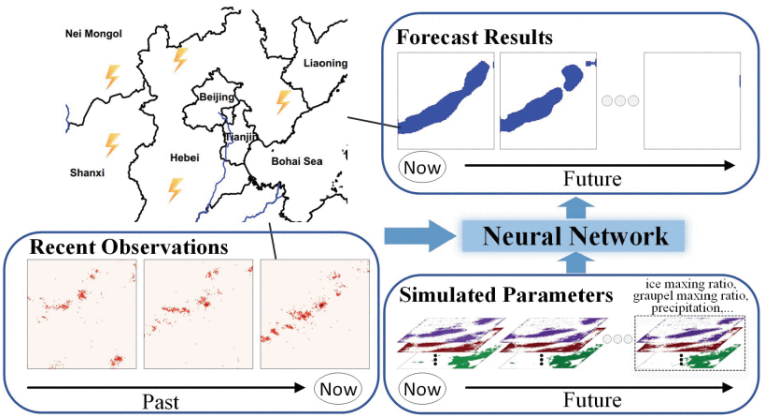

Figure 14: Cadre de la prévision d'un prédictand avec une double sources de données basées sur des observations récentes du prédictand et des simulations numériques [2]

Dans notre projet, nous avons repris exactement la même approche pour la confection d'un réseau de neurones récurrent avec des couches LSTM, prenant en entrées à la fois $X\_station$ dans un premier temps, et $X\_forecast$ dans la suite du réseau. Ci-dessous (figure 15) un schéma résumant le réseau qui a été construit pour répondre au problème de prévision de cumul de précipitation sur 24h. Pour nos expériences, le réseau construit disposait de près de 80 000 coefficients entraînables.

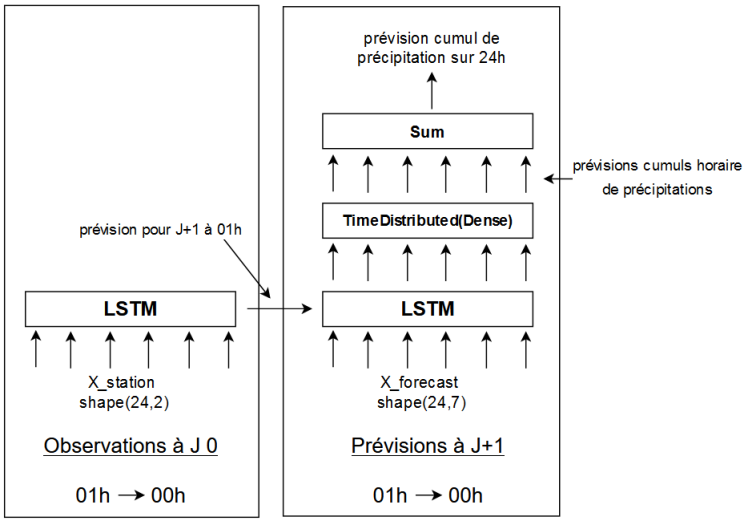

Figure 15: Représentation du RNN multi-sources implémenté avec 2 couches LSTM en série

Une fois ce réseau entraîné sur 10 epochs, nous avons obtenu un score moyen de MAPE sur kaggle de 26,7. Ce score est très satisfaisant dans le cadre de notre défi IA, mais reste décevant comparativement au score du meilleur réseau entièrement connecté à 26,3. Cependant, il est à noter que ce réseau, relativement aux deux jeux de données qui lui sont fournis en entrée $X\_station$ et $X\_forecast$, fait nettement mieux que son homologue entièrement connecté qui n'arrivait pas à aussi bien tirer parti de ces deux sources d'informations et qui obtenait un score moyen sur kaggle de 29,3. On en déduit donc que la structure même du réseau que nous avons implémenté avec l'approche temporelle (figure 15) apporte une réelle plus-value dans l'utilisation de données multi-sources.

A l'issue de l'entraînement, on observe sur la courbe de validation (figure 16), un début de sur-ajustement aux données.

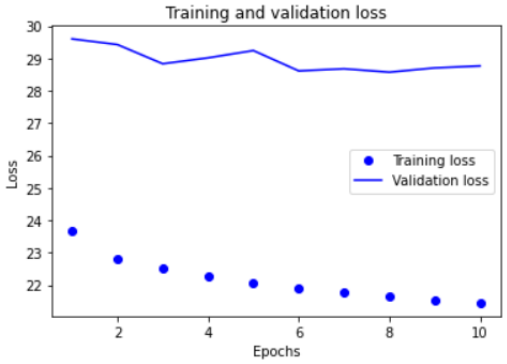

Figure 16: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un réseau récurrent composé de deux LSTM en série

Les courbes d'entraînement et de validation commencent à diverger au bout d'une dizaine d'epochs. Afin de lutter contre le sur-ajustement, nous avons introduit au sein des couches LSTM des dropout qui réinitialise de manière aléatoire les unités d'entrée des couches afin de rompre des corrélations accidentelles qui pourraient être présentes dans les données d'apprentissage. Pour implémenter le dropout dans une couche récurrente avec Keras, on dispose de deux arguments: l'argument dropout spécifiant le taux de dropout pour les unités d'entrée de la couche, et *recurrent_dropout* spécifiant le taux de dropout pour les unités récurrentes (Chollet [1]). Nous avons donc ajouté du dropout et du *recurrent_dropout* à nos deux couches LSTM, et nous avons fixé le taux de ces dropout à 20%.

Nous avons donc refait un entraînement sur 10 epochs avec l'ajout de dropout, et nous avons obtenu le résultat suivant ci-dessous (figure 17):

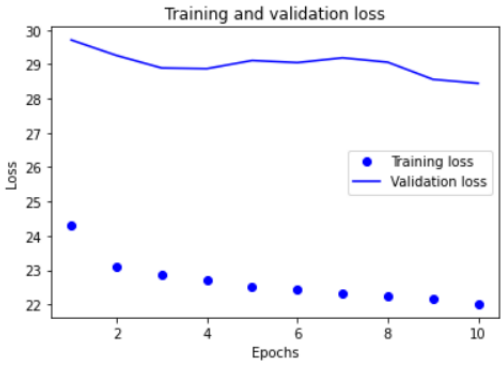

Figure 17: Perte pendant l'entraînement et la validation sur la tâche de prévision du cumul de précipitation sur 24h, avec un réseau récurrent composé de deux LSTM en série et utilisation de dropout

Grâce au dropout, le modèle a cessé d'être en sur-ajustement, et le score moyen obtenu sur kaggle a été de 26,4. Ce résultat est légèrement meilleur qu'avant l'ajout du dropout, mais il n'est pas tellement plus bas, et reste toujours très légèrement au-dessus du meilleur résultat obtenu avec le réseau entièrement connecté de 26,3.

Pour répondre au défi IA, nous avons décidé tout au long des expériences de nous focaliser sur l'implémentation de différents réseau de neurones afin d'évaluer les avantages et les inconvénients de chacun d'entre eux et de leurs performances respectives ; plutôt que de nous appesantir sur un réseau de neurones en particulier qu'on aurait essayé d'ajuster au maximum pour essayer d'améliorer toujours un peu plus le score de MAPE. A l'issue de ces deux expériences, nous n'avons donc pas essayé d'optimiser outre mesure ce réseau, mais nous avons décidé d'en implémenter un autre assez similaire, avec des couches LSTM, et prenant également en entrée les deux sources de données à notre disposition.

3.3.2 Approche avec un U-LSTM

Dans cette section, nous avons repris le réseau récurrent multi-sources précédent auquel nous avons ajouté une couche Bidirectional(LSTM) au dessus des deux couches LSTM, donnant à ce réseau un aspect en U, qui nous a inspiré pour le nom U-LSTM que nous lui avons donné. Ci-dessous un schéma représentant le réseau et ses différentes connections entre les couches récurrentes (figure 18) :

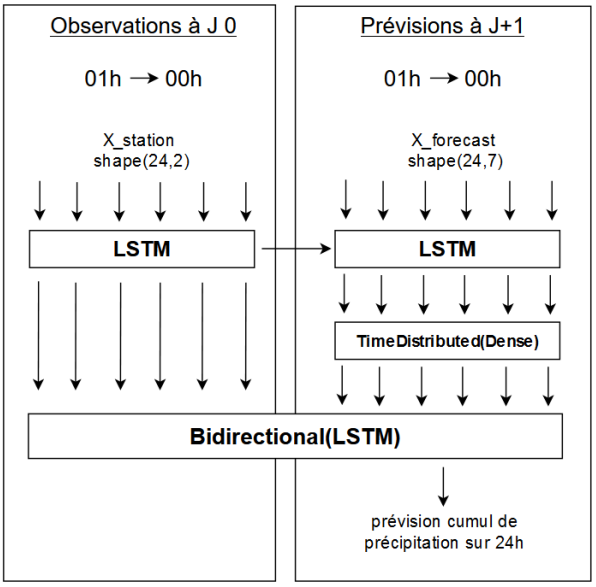

Figure 18: Représentation de l'architecture du U-LSTM

Après entraînement, ce réseau nous a donné un score moyen de MAPE de 27,8 sur kaggle, ce qui est très décevant compte tenu de la complexité du réseau implémenté, de sa capacité et de son coût en temps de calcul élevé.

Ce réseau était effectivement un des plus longs à entraîner car disposant de 4 couches LSTM.

Un RNN bidirectionnel est en effet composé de deux couches RNN parallèles, qui prend chacune en entrée la séquence de données, l'une dans un sens chronologique et l'autre dans un sens inversé, puis fusionnent les deux représentations produites dans un second temps. Ainsi en traitant une séquence dans les deux sens, un RNN bidirectionnel peut capter des motifs et des représentations qui peuvent passer inaperçu pour un RNN unidirectionnel.

Pour pallier le coût en temps de calcul que nécessitait l'entraînement de ce réseau, il a été choisi d'implémenter un nombre d'unités par couche plus faible que pour le réseau précédent, portant son nombre total de paramètres entraînables à 20 000.

Ce modèle fonctionne donc légèrement moins bien que le modèle précédent avec seulement 2 LSTM en série, ou que le meilleur réseau entièrement connecté. Il est à noter cependant que la courbe de validation de ce réseau indiquait un absence de sur-ajustement aux données, et donc la possibilité de l'entraîner sur plus d'epochs, voire d'augmenter sa capacité comme détaillé dans la section 3.1.2.

Plusieurs techniques auraient pu être testées afin d'améliorer les performances de ces modèles de prévisions récurrents multi-sources, sur ce problème de prévision du cumul journalier de précipitation:

- augmenter le nombre d'unités de chaque couche récurrente ; les choix effectués dans ces expériences étaient arbitraires et devaient surtout limiter le temps de calcul ; ils étaient donc très certainement sous-optimaux;
- modifier le nombre de couches Dense et leurs nombre d'unités, utilisées entre les couches récurrentes ;
- ajuster les taux de dropout au niveau des couches récurrentes ;
- modifier le taux d'apprentissage utilisé par l'optimiseur Adam.

Dans la section suivante, une dernière approche de réseau de neurones a été testée avec des couches convolutionnels.

3.4 Réseaux de neurones convolutionnels

Comme dernière tentative pour essayer de faire mieux sur kaggle que notre meilleur réseau entièrement connecté, nous avons choisi d'implémenter un ultime réseau de neurones avec des couches convolutionnelles. On sait que les réseaux de neurones convolutifs sont très performants sur les problèmes de vision par ordinateur, en opérant des convolutions sur des images afin d'en extraire des caractéristiques au travers de patches d'entrées locaux. Ces couches sont également très pertinentes pour le traitement des séquences temporelles. Le temps peut-être traité comme une dimension spatial, tout comme la hauteur et la largeur d'une image en 2D (Chollet [1]). On sait que les couches Conv1D peuvent rivaliser avec des RNN sur certains problèmes de prédictions sur des séries temporelles, à un coût de calcul beaucoup moins élevé.

Pour le défi IA, nous avons construit un réseau simple avec deux couches de Conv1D de 32 filtres chacune, comme illustré sur la figure 19, portant le nombre de paramètres entraînables du réseau à environ 10 000:

```
def Convolution():
  inputs = Input(shape=(24,2), name='inputs')
  inputs_2 = Input(shape=(24,7), name='inputs_2')
  x = Concatenate(axis=2)([inputs, inputs_2])
  x = Conv1D(32,7, activation='relu')(x)
  x = MaxPooling1D(2)(x)
  x = Conv1D(32,7, activation='relu')(x)
  x = tf.keras.layers.GlobalMaxPooling1D()(x)
  outputs = Dense(1, activation='relu', name='dense_6')(x)
  return Model([inputs, inputs_2], outputs, name='Convolution')
```

Figure 19: Architecture d'un réseau convolutif 1D

Nous avons réalisé plusieurs expériences à partir de ce réseau en faisant varier uniquement le jeu de données en entrée du réseau. Nous avons donc tour à tour entraîné le réseau convolutionnel avec $X\_station$, $X\_forecast$ et/ou les prédicteurs constants comme données d'entraînement. Après soumission des prévisions de l'échantillon de test sur kaggle pour les différentes expériences, nous avons obtenu les résultats suivants (figure 20):

inputs	Score moyen MAPE sur kaggle
X_station	31,8
X_forecast	26,3
X_forecast + prédicteurs_constants	26,6
X_forecast + X_station	26,6
X_forecast + X_station + prédicteurs constants	26,8

Figure 20: Résultats des expériences menées avec différents inputs sur un réseau de neurones convolutif 1D

Nous observons des scores semblables qu'avec le même plan d'expérience effectué dans la première section avec les réseaux entièrement connectés. On a une nette amélioration des performances de prévisions avec l'ajout du $X\_forecast$ dans les données d'entrées. On note également une baisse systématique du score de prévision avec l'ajout des prédicteurs constants pour lesquels le réseau ne parvient pas à en extraire des informations utiles. Le réseau convolutif ne parvient pas, tout comme les réseaux entièrement connectés, à tirer une plus-value de l'utilisation de deux sources de données avec les prévisions et les observations, contrairement aux réseaux récurrents multi-sources définis dans la section précédente. Le meilleur score est obtenu avec la même source de données ($X\_forecast$) que pour le meilleur réseau Dense, avec un score identique de 26,3.
On remarque donc que le plus performant réseau convolutionnel ne parvient également pas à battre le meilleur score obtenu au tout début des expériences avec le meilleur réseau Dense, et que les meilleurs résultats obtenus avec les réseaux convolutionnels plafonnent aussi au niveau des 26, comme ce fût également le cas avec les meilleurs réseaux récurrents.

On peut donc conclure à ce stade des expériences, qu'il est assez aisé avec des réseaux de neurones, d'obtenir un score de MAPE moyen entre 26 et 27, avec un jeu de données et une architecture de modèle un minimum adéquats pour la tâche de prévision demandée. Il semble en revanche plutôt difficile de tomber en-dessous des 26 de MAPE, et aucun de nos réseaux aussi différents soient-ils n'ont réussi cette performance. Vis-à-vis de nos concurrents dans le cadre de ce concours, on peut observer sur kaggle que le meilleur groupe a obtenu un score de 23,8, et que 7 groupes sont parvenus à un score de MAPE inférieur à 26 sur 84 participants.

Afin d'améliorer une dernière fois notre score de MAPE, nous avons décidé de faire une agrégation de nos meilleurs réseaux de neurones afin d'obtenir une prévision d'ensemble de modèles. Il est en effet courant dans ce type de compétition d'avoir recours à une moyenne de plusieurs modèles très différents les uns des autres, afin d'obtenir un score moyen supérieur à n'importe quel modèle pris individuellement. Un ensemble de modèles permet effectivement d'associer les performances de prévisions de chacun des membres, et donc d'obtenir des résultats plus robustes.
Pour réaliser cela, nous avons décidé de faire la moyenne des meilleurs réseaux obtenus pour chacune des quatre approches exposées dans cette partie, soit une moyenne de quatre réseaux de neurones, un réseau entièrement connecté, un réseau récurrent une source, un réseau récurrent multi-sources, et un réseau convolutif.
En effectuant ainsi la moyenne des quatre prévisions réalisées par ces quatre meilleurs réseaux de neurones, et en soumettant cette prévision d'ensemble sur kaggle, nous avons obtenu un score moyen de MAPE de 26,1. Cela constitue ainsi notre meilleur score de MAPE devançant très légèrement notre précédent record de 0,2 point, mais restant malgré tout très en-deça, des meilleurs résultats affichés sur kaggle.

4 Conclusion

A l'issue de l'ensemble des expériences, nous avons réussi à obtenir comme meilleur résultat un score de MAPE moyen de 26,1 avec une moyenne d'ensemble de quatre types de réseaux de neurones différents. Ce résultat nous permet ainsi de valider le défi IA en battant la $baseline\_forecast$, et s'avère globalement satisfaisant car nous permettant de battre notre référence naïve de prévisions nulles de 6 points de MAPE.

Vis-à-vis des autres équipes ayant participé au concours sur kaggle, certaines ont réussi à obtenir des scores de MAPE inférieurs à 26, la meilleure ayant obtenu un score de 23,8, tandis que tous nos modèles ont stagné entre 26 et 27. Les pistes d'amélioration qui nous auraient permis de dépasser le support des 26 de MAPE et de faire aussi bien que les meilleurs groupes du défi IA sont nombreuses :

- nous aurions pu traiter nos données différemment lors du pré-traitement ;
- nous aurions pu utiliser plus de données qui étaient à disposition sur kaggle, comme les prévisions du modèle Arpège, la $baseline\_forecast$, ou encore des données modèle 3D ;
- Nous aurions pu adopter une approche globale de prévision sur l'ensemble du Nord-Ouest de la France qui nous aurait permi de rentrer dans le modèle de prévision des informations sur la spatialisation des différentes variables météorologiques, utiles pour établir une prévision du cumul journalier de précipitations ; pour ce faire un réseau convolutionnel 2D aurait pu être implémenté ;
- D'autres modèles de machine learning aurait pu être mis en place en-dehors des réseaux de neurones, comme le gradient boosting ou encore les forêts aléatoires, qui aurait pu nous permettre à la fin d'obtenir une prévision d'ensemble multi-modèles, plus globale que la prévision d'ensemble que nous avons établi uniquement avec un panel de réseaux de neurones ;
- Les codes développés en python auraient pu être exécutés sur un GPU afin de bénéficier d'un potentiel gain de performance dans l'entraînement des réseaux de neurones.

References

[1] François Chollet, *Deep Learning with Python*, Manning, 2017

[2] T. Lin et al., "Attention-Based Dual-Source Spatiotemporal Neural Network for Lightning Forecast," in IEEE Access, vol. 7, pp. 158296-158307, 2019, doi: 10.1109/ACCESS.2019.2950328.