Département Génie Mathématiques et Modélisation

AI Framework

Report of mini-project: Daily rainfall prediction

NGUYEN Hai Vy- HOANG Van Hao-BERTIN Alexandre-BENZITOUNI Féthi

Engineering students - 5GMM
Institut National des Sciences Appliquées de Toulouse
135 Avenue de Rangueil, 31400 Toulouse

Supervisor : David BERTOIN

December 2021

Contents

1 Introduction to subject

The goal of this project is to predict the accumulated daily rainfall on the D day on the observation ground stations. This is the 6th edition of the so-called Défi IA. This challenge takes place between students from various French speaking universities and students from Meteo France. The dataset used in this project is MeteoNet, an open reference weather dataset for AI. MeteoNet contains measurement of various weather parameters in the zone of North-West quarter of France in the period 2016-2019. In particular, setup for training and testing is as follows:

- Data for training : 2016-2017
- Data for testing : 04/01/2018-2019

The train (2016-2017) and test (2018-2019) parts contain 2 years of data. We can consider it is enough to capture the weather variability during one year.

In this project, we build different deep neural network models and verify the performance of each model on a validation set and a test set on Kaggle. Many models in this project are inspired by TP courses of AI Framework and HDDL of GMM (INSA Toulouse). Our loss function for all the models in this project is Mean Square Erreur (MSE). We choose the best model based on score on Kaggle. All the experiments performed in this project are launched on Google Collab.

2 Preparing data for training and testing

2.1 Exploring data

In this project, organizer provides many weather parameters that we can use to predict the daily rainfall. However, not all the variables are necessary. Besides, there are lots of missing data. Hence, we perform some basic data exploration to have a better idea about the dataset as well as to have appropriate approach in building dataset for training and testing. We see that there are two main files that we can use:

- $X\_station\_train$: this file contains the measurement of weather parameters of the day D-1. We can use these data to predict the rainfall of day D. (i.e. the following day).
- $2D\_arpege\_\{date\}.nc$: {date} is the date that we choose. This file contains weather parameters of day D, but these parameters are also calculated and forecasted based on a physical model. So, we can use these forecasted parameter to predict the rainfall of the same day.

Hence, we can see that we have 2 principal approaches to predict rainfall of day D: by using the measurement of weather parameters of the previous day or using the forecasted weather parameters of the same day. Certainly we do not rule out the possibility of combining these two approaches.

Now, let us take a look at the file $X\_station\_train.csv$. Figure 1 illustrates the data in this file. We check the missing rate of each variable. The result is shown in table 1

	number_sta	date	ff	t	td	hu	dd	precip	Id	date_with_hour	Id2
0	14066001	2016-01-01	3.05	279.28	277.97	91.4	200.0	0.0	14066001_0	2016-01-01 00:00:00	14066001_0_0
1	14066001	2016-01-01	2.57	278.76	277.45	91.4	190.0	0.0	14066001_0	2016-01-01 01:00:00	14066001_0_1
2	14066001	2016-01-01	2.26	278.27	277.02	91.7	181.0	0.0	14066001_0	2016-01-01 02:00:00	14066001_0_2
3	14066001	2016-01-01	2.62	277.98	276.95	93.0	159.0	0.0	14066001_0	2016-01-01 03:00:00	14066001_0_3
4	14066001	2016-01-01	2.99	277.32	276.72	95.9	171.0	0.0	14066001_0	2016-01-01 04:00:00	14066001_0_4
...	...	...	...	...	...	...	...	...	...	...	...
4409469	95690001	2017-12-30	9.10	286.68	283.44	80.8	239.0	0.0	95690001_729	2017-12-30 19:00:00	95690001_729_19
4409470	95690001	2017-12-30	8.58	286.39	283.21	81.1	231.0	0.0	95690001_729	2017-12-30 20:00:00	95690001_729_20
4409471	95690001	2017-12-30	8.74	286.28	283.40	82.6	226.0	0.0	95690001_729	2017-12-30 21:00:00	95690001_729_21
4409472	95690001	2017-12-30	9.04	286.21	283.29	82.4	224.0	0.0	95690001_729	2017-12-30 22:00:00	95690001_729_22
4409473	95690001	2017-12-30	9.11	285.92	282.42	79.4	221.0	0.0	95690001_729	2017-12-30 23:00:00	95690001_729_23

4409474 rows × 11 columns

Figure 1: Illustration of X_station_train file

Variable	Missing rate (%)
ff	39.7
t	5.2
td	32.4
hu	32.3
dd	39.7
precip	7.0

Table 1: Missing rate of variables of *X_station_train* file

We see that for variables ff, td, hu and dd, the missing rate is more than 30%. This missing rate is too large, and training on this will make the model lost in generality and representativity of the data. So, we do not choose these variables for training our model and we keep t and $precip$ as input variables to train models.

Another file that we considered to train model is $2D\_arpege\_\{date\}.nc$, where {date} is the date that we choose. Different weather parameters at different locations are available in these files (see Figure 2). Locations are in a grid of 58 latitudes and 80 longitudes.

▾ Data variables:

ws	(valid_time, latitude, longitude)	float32	...
p3031	(valid_time, latitude, longitude)	float32	...
u10	(valid_time, latitude, longitude)	float32	...
v10	(valid_time, latitude, longitude)	float32	...
t2m	(valid_time, latitude, longitude)	float32	...
d2m	(valid_time, latitude, longitude)	float32	...
r	(valid_time, latitude, longitude)	float32	...
tp	(valid_time, latitude, longitude)	float32	...
msl	(valid_time, latitude, longitude)	float32	...

Figure 2: Illustration of $2D\_arpege\_\{date\}.nc$ file

Next, we will observe the file Y_train.

	date	number_sta	Ground_truth	Id
0	2016-01-02	14066001	3.4	14066001_0
249	2016-01-03	14066001	11.7	14066001_1
2491	2016-01-12	14066001	1.0	14066001_10
24978	2016-04-11	14066001	5.6	14066001_100
25228	2016-04-12	14066001	3.2	14066001_101
...	...	...	...	...
182747	2017-12-27	95690001	3.2	95690001_725
182997	2017-12-28	95690001	0.0	95690001_726
183247	2017-12-29	95690001	4.4	95690001_727
183497	2017-12-30	95690001	5.4	95690001_728
183746	2017-12-31	95690001	1.2	95690001_729

183747 rows × 4 columns

Figure 3: Illustration of Y_train file

This file contains daily rainfall of different stations in different time (date). To match the value of Y to the X_train, we base on the Id. The Id is unique for each station for a given date. Using pandas, it is practical to do this task.

2.2 Forming dataset for training

First of all, as mentioned before, we merge $X\_station\_train$ with Y_train based on Id. Then for each Id (i.e a station at a given date), we extract only t and $precip$ as input variables corresponding to a value of Y. Notice that for a value of Y (a scalar), we have a sequence of length 24 for t and similarly for $precip$ (24 corresponding to 24 hours of the day D-1 since we have hourly record for each day). Besides, we choose only the Id that we have full information about $precip$ and t for 24 hours.(i.e if on the day D-1, there exists some hours that we do not have measurement, we will not add it to the training set). Besides, we create another array to contain the month of each training example because information about month could be interesting.

Finally, we have:

- An array X_precip of shape (156605, 24), saved in file $X\_precip\_train.csv$.
- An array X_t of shape (156605, 24), saved in file $X\_t\_train.csv$.
- An array $month$ of shape (156605), saved in file $month\_train.csv$.
- An array y of shape (156605,), saved in file $y\_train.csv$.

Next, we get data from the files $2D\_arpege\_\{date\}.nc$'s. To get a variable from this file, we have to precise the value of latitude and longitude (these values must be the values available in the grid). Since the stations are not exactly located on the nodes of the grid, we need to find the nodes nearest to the station that we consider. For this purpose, one may use Haversine distance, however it is quite complex. So, we find the node corresponding to a station by finding nearest value of latitude and longitude to the position of the station, knowing that we have information about location of stations (latitude, longitude and altitude). By this, we can get the data from these file for each stations. Three variables that we extract from these files

are $p3031$ (mean sea level pressure), $d2m$ (dew point temperature) and r (relative humidity).

2.3 Forming dataset for testing

In the test set, the days are shuffled in a random way. Each day is indexed by a number. However, the month can be an interesting information to predict rainfall and we can find in test part a file called $Id\_month\_test.csv$, which gives us the corresponding month to each day index. Based on day index, we can form the dataset for testing in a way completely similar to forming the training dataset.

3 Experiment with different models

At the beginning, we worked only with the file $X\_station\_train$ as it is quite simple to get data. Hence, our initial models are only trained and then predict on $X\_station$ data. After some research, we finally succeeded to open and to form data from the files $2D\_arpege\_\{date\}.nc$'s for training and predicting. Hence, the models that we present below will be in chronological order that we built in this project. This also allows us to explain the reasons why we built each model (the intuition behind it) and how the ideas gradually came in our mind naturally in chronological order. For all the training we performed in this project, we fix batch size to 32 and train/validation split ratio is 0.8.

3.1 Multi-layer Perception (MLP) model

First of all, we want to approach the problem in a simple way. So MLP is a good point to start. The input of our MLP is a vector of dimension 48,including 24 components for the sequence of precipitation of length 24 the day D-1 and a similar sequence of t. After tuning and changing hyper-parameters, we came up with following model shown in Figure 4.

```
model =km.Sequential()
model.add(kl.Dense(64, input_shape=(48,),  activation = "relu"))
model.add(kl.Dense(128,  activation = "relu"))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(128,  activation = "relu"))
model.add(kl.Dropout(0.2))
model.add(kl.Dense(64,  activation = "relu"))
model.add(kl.Dense(32,  activation = "relu"))
model.add(kl.Dense(1, activation='relu'))
model.compile(loss='mse', optimizer='adam', metrics=['mse'])
model.summary()
```

Figure 4: Structure of the model

After training, the Mean Square Error (MSE) on training set is 21.3564. In general, for different MLP models that we tested, MSE on training set always stops around 21.3.

3.2 Multi-layer Perception (MLP) mode with Conv1D layer

We think that using Conv1D layer to capture some characteristics in time dimension could help improve the prediction. So, we try following model shown in Figure 5. Notice that *timesteps* is 24 and *input_dim* is 2 (*precip* and t).

```
model3 =km.Sequential()
model3.add(kl.Conv1D(32, 3, activation='relu', input_shape=(timesteps, input_dim)))
model3.add(kl.MaxPooling1D(pool_size=3))
model3.add(kl.Flatten())
model3.add(kl.Dense(64, input_shape=(48,),  activation = "relu"))
model3.add(kl.Dense(128,  activation = "relu"))
model3.add(kl.Dropout(0.2))
model3.add(kl.Dense(128,  activation = "relu"))
model3.add(kl.Dropout(0.2))
model3.add(kl.Dense(64,  activation = "relu"))
model3.add(kl.Dense(32,  activation = "relu"))
model3.add(kl.Dense(1, activation='relu'))
```

Figure 5: Structure of the model

The prediction is not improved: MSE equals 21.6005 on training set and 20.3801 on validation set. This is quite disappointing as we add Conv1D layer but the performance is not enhanced.

3.3 MLP model with fast Fourier transform (FFT)

A fast Fourier transform (FFT) is algorithm that computes the discrete Fourier transform (DFT) of a sequence. It converts a signal from the original data, which is time for this case, to representation in the frequency domain. Because the frequency represents the variability of sequence, it could be interesting to work with sequence after FFT. So for each sequence (length 24) of *precip* and t, we will centralize it and then perform FFT to obtain two other sequences of *precip* and t, but in the frequency domain. Finally, we concatenate the 4 sequences of length 24 (2 initial sequences and 2 sequences after FFT of *precip* and t) to obtain a vector of dimension 96. Hence, the vector we input in our MLP model is of dimension 96. The reason why we do not work only with sequences FFT but preserve also the initial sequences is that the initial sequences of *precip* et t contain useful direct information (intuitively, for instance, if temperature is high and no rain the day D-1 then one could predict that it would not rain the day D) whereas the sequences after FFT indicate some interesting information concerning the variability (in time dimension) in day D-1 of *precip* and t. So the initial sequences and the sequences after FFT of t *precip* could be complementary to each other.

```
model4 =km.Sequential()
model4.add(kl.Dense(64, input_shape=(96,),  activation = "relu"))
model4.add(kl.Dense(128,  activation = "relu"))
model4.add(kl.Dropout(0.2))
model4.add(kl.Dense(128,  activation = "relu"))
model4.add(kl.Dropout(0.2))
model4.add(kl.Dense(64,  activation = "relu"))
model4.add(kl.Dense(32,  activation = "relu"))
model4.add(kl.Dense(1, activation='relu'))
opt = tensorflow.keras.optimizers.Adam(learning_rate=1e-3)
model4.compile(loss='mse', optimizer=opt, metrics=['mse'])
model4.summary()
```

Figure 6: Structure of the model

However, with this modification for input, MSE on validation set is always 21.3565.

3.4 Recurrent neural network model with SimpleRNN and Bidirectional GRU layers

As seen above, the MLP models do not work really well to predict rainfall even though we have applied some techniques as Conv1D and FFT. So, we think of using other types of model which are more effective. Since the input in a sequence, we try to build some recurrent neural network (RNN). Our first RNN model is shown in Figure 7. The input shape here is (24,2) since we have two sequences of *precip* and t of the day D-1, each sequence is of length 24 (for 24 hours).

```
model1 = km.Sequential()
model1.add(kl.SimpleRNN(units=20 ,activation="relu", input_shape=(24, 2),return_sequences=True))
model1.add(kl.Bidirectional(kl.GRU(units=20 ,activation="relu")))
model1.add(kl.Dense(1))
model1.summary()
```

Figure 7: Structure of the model

After training, MSE on validation set is 15.0170, which is much better than MLP models and this result is super encouraging for us. Hence, we try to put the prediction on Kaggle and the score on Kaggle is 49.09164.

3.5 Recurrent neural network model with LSTM and Bidirectional GRU layers

With only SimpleRNN layer, the result is much improved compared to MLP models, so we want to replace SimpleRNN layer by LSTM layer. The reason for this choice is that LSTM could have longer memory in time and also in almost every research paper on forecasting that we found, one always use LSTM layers.

```
model12 = km.Sequential()
model12.add(kl.LSTM(units=20 ,activation="relu", input_shape=(24, 2),return_sequences=True))
model12.add(kl.Bidirectional(kl.GRU(units=20 ,activation="relu")))
model12.add(kl.Dense(1))
model12.summary()
```

Figure 8: Structure of the model

After training, MSE on validation set is 14.8701, which is an improvement compared to SimpleRNN layer. So once again, we use this model and predict the test set on Kaggle and the score on Kaggle is 64.56795, which is worse than the previous model. This result is quite disappointing, because on validation set the result is better. To explain this, we think that LSTM layer is more complex than simple RNN, and so it learns more non-generalized details. As the training and validation set is in period 2016-2017 and test set is 2018-2019, so maybe the details learned by LSTM do not appear much or could change in perdiod 2018-2019. This is our best explanation for the degradation of the model's performance on test set.

3.6 Recurrent model with Conv1D

Then, we think of adding Conv1D layer because this type of layer allows us to capture some characteristics of the sequences in time dimension. We add a Conv1D layer at beginning of the network and then we tried with different recurrent layers. One of the models that we built is shown in Figure 9.

```
model3 = km.Sequential()
model3.add(kl.Conv1D(32, 3, activation='relu', input_shape=(24, 2),padding='same'))
model3.add(kl.LSTM(10, return_sequences=True))
model3.add(kl.SimpleRNN(units=10 ,activation="relu"))
model3.add(kl.Dense(1,activation='relu'))
model3.summary()
```

Figure 9: Structure of the model

For different models of this type (i.e we add a Conv1D layer), MSE on validation set is always 21.3564 after training, which is similar to MLP models and much worse than the previous RNN models without Conv1D. To explain this, we think that the Conv1D is not "intelligent" enough in this case to capture appropriate details in the sequences, and it make the initial sequences lose some important information. Consequently, the RNN layers after this layer cannot learn effectively.

3.7 Uni-dimensional Recurrent Neural Network model

After building all the models above where we use a sequence of *precip* and a sequence of t of day D-1 as input to predict the rainfall of day D, we think of model using only a sequence of *precip* as input. And the intuition for this is based on the fact that many simple model predicting the rainfall of day D equal to day D-1. So, in the worst case, our Recurrent Neural Network model just learns to sum up the 24 elements of the input sequence to have the output which is the accumulated daily rainfall on the D day. The structure of our model is shown in Figure 10.

```
model3 = km.Sequential()
model3.add(kl.Dense(units=48 ,activation="relu", input_shape=(24,)))
model3.add(kl.Dense(units=24 ,activation="relu"))
model3.add(kl.Reshape((24,1),input_shape=(24,)))
model3.add(kl.GRU(units=20 ,activation="relu", input_shape=(24, 1),return_sequences=True))
model3.add(kl.Bidirectional(kl.GRU(units=20 ,activation="relu")))
model3.add(kl.Dense(units=20 ,activation="relu"))
model3.add(kl.Dense(1))
model3.summary()
```

Figure 10: Structure of the model

After training, MSE on validation set is 14.8847 and score on Kaggle is 46.78 which is our best result so far. This result is really encouraging and we thought that Uni-dimensional Recurrent Neural Network model could be a game changer for us because it is simpler but more efficient. So we think that building different variant versions of this model could be a good idea to continue. In particular, we did following variants:

- Replacing simpleRNN and GRU by LSTM. The result on Kaggle gets worse.
- Do not modify the model, but eliminating the outlier of Y: using only the training examples that have $Y < 0.95 - quantile$. As result, the MSE on validation set is reduced to 3.9010 but score on Kaggle is 49.11628, which is worse than training the model on the whole training data-set.
- Do not modify the model, but eliminating the outlier of Y as above. Moreover, instead of training on Y as label, we trained on $log(Y)$. The reason why we do it is because here we use MSE as loss function, and it will not distinguish the order of magnitude of Y,i.e. for 2 cases (1): $Y = 3$ and $Y_{predict} = 3.5$ and (2): $Y = 20$ and $Y_{predict} = 20.5$, the model will consider the errors are the same in the two cases. But we can see that the error in the case (1) is more serious. So, by training on $log(Y)$ as label, we hope that the model would pay more attention to small value of Y. Since there are many values $Y = 0$, we add $\epsilon = 10^{-6}$ in the log, so finally we work with $log(Y + \epsilon)$ as label to avoid invalid value. After training, we obtained the score on Kaggle of 97.65889, which is much worse than the initial model. As conclusion, this does not work!

3.8 Recurrent Model Network trained based on months

After some research on forecasting techniques, we found out that the weather condition could change dramatically from month to month. So, the characteristics of temperature and precipitation could be very different from month to month. Hence, training one unique model could not be really reasonable to generalize the predictability of the model for all months. So, we think of training model based on months. For that, we built 12 RNN model with the same structure but trained on data of 12 months separately. As in the test set on Kaggle, the day is hidden but we know which month it is, we can apply 12 models for 12 months on test set. The structure of model is shown in Figure 11. Our input here is once again sequence of *precip* and t of the day D-1.

```
def make():
  model = km.Sequential()
  model.add(kl.SimpleRNN(units=20 ,activation="relu", input_shape=(24, 2),return_sequences=True))
  model.add(kl.Bidirectional(kl.GRU(units=20 ,activation="relu")))
  model.add(kl.Dense(1))
  model.compile(loss="mse", optimizer="adam")
  return model
```

Figure 11: Structure of the model

The reason why we choose this structure is because it is one of our best model by far (Section 3.4) but its structure is quite simple so that we can avoid over-fitting as we already train 12 models for 12 months. After training 12 models for 12 months and predicting on test set, we obtain the score on Kaggle of 86.212, which is much worse than training a unique model for all the months.

3.9 Recurrent Model Network trained based on groups of months

Maybe training 12 models on 12 months separately does not give us good result because it is over-fitted since training on each month is not generalized enough. So, we think of training model on different groups of months that have similar characteristics. For that, let us observe following boxplot of Y (daily rainfall) of 12 months in Figure 12. Notice that the months from January to December are numbered from 1 to 12.

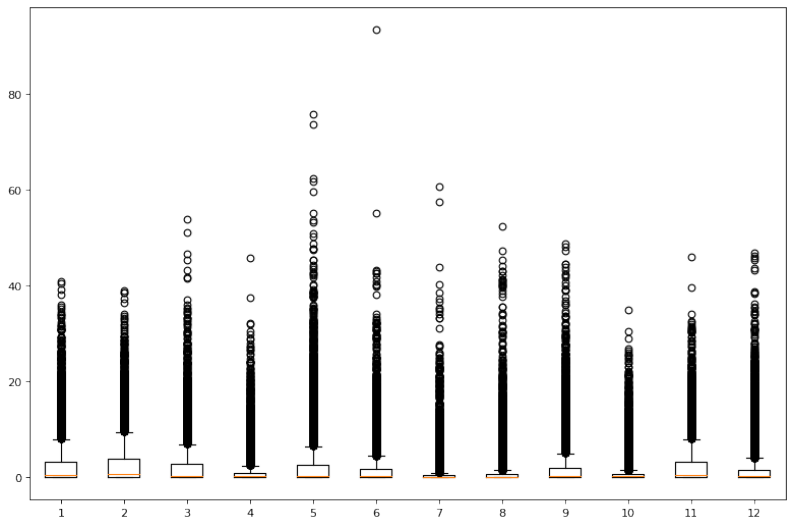

Figure 12: Structure of the model

Based on the form of boxplots, we can divide the 12 months into groups that have similar distribution of daily rainfall:

- Group 1: April, July, August, October
- Group 2: June, September, December
- Group 3: January, February, March, May, November

```
def make_group():
  model_group = km.Sequential()
  model_group.add(kl.Dense(units=48 ,activation="relu", input_shape=(24,)))
  model_group.add(kl.Dense(units=24 ,activation="relu"))
  model_group.add(kl.Reshape((24,1),input_shape=(24,)))
  model_group.add(kl.GRU(units=20 ,activation="relu", input_shape=(24, 1),return_sequences=True))
  model_group.add(kl.Bidirectional(kl.GRU(units=20 ,activation="relu")))
  model_group.add(kl.Dense(units=20 ,activation="relu"))
  model_group.add(kl.Dense(1))
  return model_group
```

Figure 13: Structure of the model

Moreover, we also eliminate outlier by choosing only training examples that have $Y < 0.95 - quantile$. After training, the score on Kaggle is 54.243, which is not too bad compared to all the models so far, but it is worse than training on the whole training example. So, for conclusion, training on groups of months is not effective!

3.10 RNN model training on data from the files $2D\_arpege\_\{date\}.nc$'s

As we mentioned, at the beginning, we did not know how to get data from these files. That is why all the models we built above is trained only on data from $X\_station\_train$. But after some research, we have been able to get data from these files. As we said in the Section of Preparing Data, there are many variables available (see Figure 2) but we choose three variables: $p3031$ (mean sea level pressure), $d2m$ (dew point temperature) and r (relative humidity) because we think that these weather parameters are very correlated to the rainfall of that day. However, to verify if these parameters are really correlated to the rainfall, we do as follows:

- Building a uni-dimensional model, and we train only using sequence of one variable as input.
- After training, we calculate the loss function (MSE) on validation set. If this error is above a threshold that we set a priori, then we consider the variable is not correlated with the rainfall.

For the uni-dimensional model, we choose the model in section 3.7 (see Figure 10) and we set threshold=17. After this step, we see that only two variables r (relative humidity) and $p3031$ (mean sea level pressure) are correlated to rainfall based on our criterion. So, we will use only these 2 variables as input to train models.

```
model2 = km.Sequential()
model2.add(kl.GRU(units=20 ,activation="relu", input_shape=(24, 2),return_sequences=True))
model2.add(kl.Bidirectional(kl.GRU(units=20 ,activation="relu")))
model2.add(kl.Dense(units=20 ,activation="relu"))
model2.add(kl.Dense(1))
model2.summary()
```

Figure 14: Structure of the model

After training, MSE on validation is 13.7765. This result is really encouraging because this is the first model with MSE on validation under 14. To explain the reason why the performance is clearly improved when using data from files

$2D\_arpege\_\{date\}.nc$'s instead of $X\_station\_train$, we think that it is because the data from $2D\_arpege\_\{date\}.nc$ are from the day D, i.e the same day that we need to predict the rainfall, whereas $X\_station$ contains the data for the day D-1, i.e the previous day. Consequently, the weather parameters in $2D\_arpege\_\{date\}.nc$ are more correlated with the rainfall than the weather parameters in $X\_station$. However, we should notice that the data in $2D\_arpege\_\{date\}.nc$ are also calculated and predicted based on a physical model while the data in $X\_station$ is the measurement of these parameters without the impact of any model. So, the risk when we use data from $2D\_arpege\_\{date\}.nc$ is that the performance of our model depends also on the exactitude of the prediction of other physical models using to have $2D\_arpege\_\{date\}.nc$.

In the phase of predicting, we realized that there are some days in the test set that are not available in the files $2D\_arpege\_\{date\}.nc$'s. To handle this, we apply two strategies:

- Calculating the average over month of input parameters: for the days that we do not have data, we determine which month it is (thanks to the file $Id\_month\_test.csv$), we get all the data of other days of that month, then we calculate the average for r (relative humidity) and $p3031$ of that month. The missing data will be replaced by these average values. Then we predict as normal. The score we obtained on Kaggle is 83.28281

- Calculate the average over month of output of model: we predict for all the days that do not miss input variables. Then for the days missing data, we predict the rainfall as the average of the month they belongs to. The score we obtained on Kaggle is 77.28280.

4 Discussion and Conclusion

In this this report, we presented the models most representative among all the models that we built. Reader can notice that our models are built around MLP model and RNN model. We started by MLP models because they are simple. Then, we developed different RNN models because RNN models have shown many success with the sequence data.

Besides the technique FFT and adding Conv1D layers, we also tested with wavelet: we approximate by $db1$ wavelet to different levels (1,2,3) to smooth the sequence of *precip* before inputting to the model. However, this did not work, so we do not present it in this report due to the limit of this report. Moreover, we also tested with Random Forest Model but the result was not good compared to our best model based on the Score on Kaggle. So, Random Forest model is not appropriate in this case and so we did not present it here.

Overall, the Uni-dimensional RNN model in Section 3.7 work the best. To explain this, we think that Uni-dimensional model is less complex, so it helps the model to avoid learning unnecessary details and moreover, more multi-dimensional model will also make the optimization process complicated to converge to the optimal point. Of course, it must also be mentioned again if based on MSE on validation set, the model in Section 3.10 is the best. So, without missing data, we think that this model should be used to predict the daily rainfall.

Finally, as a critique of our work, we should surely have worked directly with the loss function that serves as an assessment for the Defi-IA and not go through the MSE. However, when we implemented Mean Absolute Percentage Error (MAPE) as loss function to train model: we have MAPE=29% on validation but only 94% for the result on Kaggle. To explain this, we need more study because it is not a simple question. Furthermore, while we focused on the different possible models to build, we neglected the importance of the input parameters of the networks by providing poor features. We should certainly have put more thought into the inputs to the models, such as bringing the month's information directly into the input instead of training 12 months separately.