Elodie NOËLÉ

End of study report

Learning with incomplete data and model: Using Data Assimilation to train Machine Learning models

August 24, 2019

Encadrants de stage:
Pr. D. BÉRÉZIAT
Pr. A.A. CHARANTONIS
Pr. J. BRAJARD

Jury :
Pr. L. Hardouin
Pr. L. Jaulin
Pr. B. Zerr

IETA2019 - Promotion Jean-Baptiste Vaquette de Gribeauval - SPID/ROB

Acknowledgments

I would like to thank my internship tutors, Dominique Béréziat, Alexandre Anastase Charantonis and Julien Brajard, who gave me the opportunity to carry out this research internship, who were very available during this internship and for reviewing this report. I also thank my fellow intern Arthur Filoche with me for all the help he provided during the internship.

Finally, I would like to thank all the people at LIP6 who helped me to go through this internship and all the people who contributed to make this internship an excellent experience.

Contents

List of Figures

List of acronyms

BLUE	*Best linear unbiased estimator or least squares analysis*
BFGS	*Broyden-Fletcher-Goldfarb-Shanno method*
CNRS	*Centre national de la recherche scientifique*
DA	*Data Assimilation*
ML	*Machine Learning*
LBFGS	*Limited-memory Broyden-Fletcher-Goldfarb-Shanno method*
LIP6	*Laboratoire d'informatique de Paris 6.*
LOCEAN	*Laboratoire d'océanographie et du climat*
NetCDF	*Network Common Data Form*
PEQUAN	*Laboratoire d'océanographie et du climat*

CHAPTER 1

Résumé

Au cours de ce stage de recherche, l'objectif était d'améliorer les prévisions de la vitesse et de la température de surface de l'océan à l'aide de techniques d'assimilation de données et d'apprentissage automatique. Grâce à l'assimilation des données, nous sommes donc en mesure d'obtenir une bonne ébauche pour les prévisions à court terme. Cependant, étant donné la difficulté de prendre en compte tous les termes non linéaires par les méthodes numériques conventionnelles, nous avons dégradé le modèle physique initial en un modèle physique linéaire calculé par assimilation de données et un modèle non linéaire obtenu en utilisant notre algorithme d'apprentissage machine.

La première partie du stage visait à réaliser un état de l'art de l'assimilation de données, du machine learning ainsi que de la combinaison des données.

La seconde partie du stage consistait donc en l'implémentation et la validation d'un algorithme d'assimilation de données avec le modèle physique complet, issu de la littérature.

Une fois ce modèle validé et ayant produit des résultats jugés acceptables, nous avons tenté de dégrader le modèle initial en une partie linéaire et une partie non-linéaire. Notre objectif était donc de rechercher et de valider un algorithme de machine learning permettant de prédire la partie non-linéaire du modèle afin de pouvoir utiliser notre algorithme d'assimilation de données sur la partie linéaire du modèle, calculée numériquement, et la partie non-linéaire, prédite par l'algorithme de machine learning.

CHAPTER 2

Abstract

During this research internship, the objective was to improve predictions of ocean surface speed and temperature using data assimilation and automatic learning techniques. By assimilating the data, we are therefore able to obtain a good draft for short-term forecasts. However, given the difficulty of taking into account all non-linear terms by conventional numerical methods, we have degraded the initial physical model into a linear physical model calculated by data assimilation and a non-linear model obtained using our machine learning algorithm.

The first part of the internship aimed to achieve a state of the art in data assimilation, machine learning and data combination.

The second part of the internship consisted of the implementation and validation of a data assimilation algorithm with the complete physical model from the literature.

Once this model was validated and produced acceptable results, we tried to degrade the initial model into a linear and a non-linear part. Our objective was therefore to research and validate a machine learning algorithm to predict the non-linear part of the model in order to be able to use our data assimilation algorithm on the linear part of the model, calculated numerically, and the non-linear part, predicted by the machine learning algorithm.

Introduction

Predicting the evolution of the atmosphere is a complicated problem that requires the most accurate initial conditions to obtain an accurate estimate of the atmospheric state variables at a given time and point. The data from sensors is often partial, distorted or too inaccurate. That is why data assimilation is a mathematical discipline that corrects the state of the atmosphere thanks to observations at different times and points in space. This technique consists in linking sensor data with a physical model of the atmosphere allowing to find an error on the background, correcting the initial conditions, and an error on the model allowing to find the missing part of the dynamics. We are therefore trying to optimize the whole trajectory from our initial state to our final state. However, this approach is based only model-driven. Therefore, knowledge of the data is not wisely used. This is why we seek to combine data assimilation and deep learning, which makes it possible to predict and understand complex spatio-temporal phenomena, sometimes in an optimal way compared to traditional data assimilation approaches. The combination of the two should enable us to obtain much better results by taking advantage of the purely physical approach through data assimilation, useful for finding the linear parts of our model, and the purely given approach through machine learning, useful for taking advantage of observations and for finding the non-linear parts of the model.

CHAPTER 3

Overview of the LIP6

LIP6 is a research laboratory under the supervision of Sorbonne University and the CNRS1. Its name comes from the acronym of the historical name, the Laboratoire d'Informatique de Paris 6. With 220 permanent researchers and 199 doctoral candidates, LIP6 is the largest computer science research laboratory in France.

The laboratory covers a wide spectrum of activities grouped into 5 departments: Scientific computing; Decision, Intelligent Systems and Operational Research; Data and Artificial Learning; Networks and Distributed Systems; System On Chip.

During this internship, my team, PEQUAN, was attached to the Scientific Computing Department. The Scientific Computing department groups together activities concerning symbolic or numerical calculation. The main objective is to establish and implement efficient algorithms that provide guaranteed results. Some important contributions of this project concern applied sciences such as cryptography, code error correction, robotics or signal theory.

Within the PEQUAN team, researchers are working closely with LOCEAN to predict large-scale oceanographic movements. As part of the internship, work had already been done on predictions of ocean speed fields and surface temperatures. The internship is therefore a continuation of this work.

CHAPTER 4

Data Assimilation Process

4.1 Data Assimilation

Data assimilation is a method of combining a physical model with observations. It can be used for multiple purposes. DA can be used to estimate the optimal state of a model, to estimate the initial state of a system in order to use it to predict the future state of the system. The most known use of data assimilation is predicting the state of the atmosphere using meteorological data.

Numerical prediction of atmospheric evolution is critically dependent on the initial conditions provided to it. However, it is difficult to determine due to the chaotic nature of the environment even with a good understanding of the physical laws that drive it the state of the atmosphere at any point of our spatio-temporal domain, i.e. all the atmospheric variables (pressure, temperature, humidity, etc.) over the entire volume, with good resolution and accuracy.

The information available at any given time is meteorological observations of various kinds (radio soundings, weather stations, ocean buoys, etc.). But this information is not enough to fully describe the conditions of the model not to mention the fact that the observations may contain errors. Indeed, the atmospheric model requires about 10^7 values (for all physical fields considered, at all points of the model). However, the observations are in the order of 10^6. A simple interpolation is not enough under these conditions that is when data assimilation can be applied.

Data assimilation is a "prediction/correction" method. A forecast, calculated at the previous time step and valid at the time considered, is used as a predictor. The available observations allow this first guess to be corrected to best estimate the actual state of the atmosphere.

In practice, there are two main numerical methods for data assimilation:

- The Kalman filter which is based on stochastic data assimilation methods using BLUE and K gain matrix calculations. Its application requires that both the model and observation operators be linear.
- The 4D-Var algorithm based on variational data assimilation methods by minimizing a cost function using optimization and adjoint calculation methods. The observation model or operations need not be linear.

It is on this second method that we will focus in the rest of this report.

4.2 Study of the internship

4.2.1 Mathemical overview of 4DVar

We suppose having $\mathbf{X} = \begin{bmatrix} w(x,t) \\ I(x,t) \end{bmatrix} = \begin{bmatrix} u(x,t) \\ v(x,t) \\ I(x,t) \end{bmatrix}$, the state vector of the system depending on the space coordinates x and time coordinates t. $\mathbf{X}$ is defined on space $\mathbb{A} = \Omega * [0,T]$, where Ω represents the spatial domain and $[0,T]$ represents the temporal domain. The state equation of the vector $\mathbf{X}$ is as follows:

$$\frac{\partial \mathbf{X}}{\partial t}(x,t) + \mathbb{M}(\mathbf{X})(x,t) = \mathcal{E}_m(x,t) \tag{4.1}$$

With:

- M, the evolutionary model, supposedly differentiable. A 2D Lagrangian model is used for computing the velocity field. M consists of a known linear part, $\mathbb{M}_l$, and a non-linear part, partially known, but considered unknown in our frame, $\mathbb{M}_{nl}$.
- ϵ_m, the model error used to translate observational errors or errors in model knowledge.

In more detail, the equation 4.1 is written: [4]

$$\frac{dw(x,t)}{\partial t} = \frac{\partial w(x,t)}{\partial t} + \nabla w.w(x,t) = \mathcal{E}_v(x,t) \tag{4.2}$$

In the case of a perfect model, $\mathcal{E}_v(x,t) = 0$.

$$\frac{\partial I(x,t)}{\partial t} + w(x,t)\nabla I(x,t) = 0 \tag{4.3}$$

With:

- $w(x,t)$, the velocity field at any point x and at any time t;
- $u(x,t)$, the horizontal component of the velocity field at any point x and at any time t;
- $v(x,t)$, the vertical component of the velocity field at any point x and at any time t;
- $I(x,t)$, the passive temperature tracer at any point x and at any time t;
- $\mathcal{E}_v(x,t)$, the model error.

We also assume that we have the initial condition of the state vector at $t = 0$:

$$\mathbf{X}(x.0) = \mathbf{X}_b(x) + \mathcal{E}_b(x) \tag{4.4}$$

With:

- $\mathbf{X}_b$, the initial draft or state vector
- $\mathcal{E}_b$, the error on the background with covariance matrix $\mathbf{B}$

The equation on observations $\mathbf{Y}(x,t)$ is written:

$$\mathbf{Y}(x,t) = \mathbb{H}(\mathbf{X})(x,t) + \mathcal{E}_O(x,t) \tag{4.5}$$

With:

- $\mathbb{H}$, the observation operator;
- $\epsilon_O(x,t)$, the error on the model at any point and the measurement errors with covariance matrix $\mathbf{R}$.

We are therefore trying to optimize the following cost function:

$$\mathbb{J}(\mathbf{X}) = \frac{1}{2}||\mathbf{X} - \mathbf{X}^b||_B^2 + \frac{1}{2}\int_{t_{initial}}^{t_{final}} ||\mathbb{H}(\mathbf{X} - \mathbf{Y}(t)||_{R(t)}^2 = \mathbb{J}^B + \mathbb{J}^O \tag{4.6}$$

With :

- $\mathbb{H}$, the observation operator;

In Appendix C, the trajectory optimized by the 4dvar algorithm is presented.

Thus, we will use the following 4D Var algorithm:

For Motion Estimation by Data Assimilation, we will use the 4D-Var algorithm [4] :

4.2.2 Problems of the internship

As part of the observation of ocean data, we wish to obtain information on the state of the atmosphere which represents an N-dimensional space with observations that are contained in a P-dimensional space (where N > P. Generally, N is in the range of 10^7 to 10^{11} values while P is in the range of 10^6 to 10^{10} values [7]). In our case, we will use ocean surface temperature images as shown in Figure 4.1.

We therefore have as input F images of ocean surface temperature I_1^O to I_F^O, as mentioned in section 4.2. From these observations, we can estimate the velocity field

Algorithm 1 4D-Var

4D-VAR inputs : $MaxIter$, $I^O_{l=1->N^0}$, $\mathbb{M}$, function BFGS,
covariance matrix R for $\mathcal{E}_R$, covariance matrix B for $\mathcal{E}_B$
Read the observation images I^O_l for l ranging from 1 to N^0
$X_b \leftarrow$(0 0 I^O_1)^T
k $\leftarrow 0$
while t!= T_f **do**
$X(0)^k \leftarrow X_b$
Compute the model $\mathbb{M}$ and compute J^k (**Forward**)
Compute the adjoint model for obtaining the gradient and compute $\nabla \mathrm{J}^k$ (**Backward**)
while $|\nabla J^k| > \mathcal{E}$ and $k < MaxIter$ **do**
($X(0)^{k+1}, a^{k+1}) \leftarrow BFGS(\mathrm{X}(0)^{k+1}, a^{k+1}, J^k, \nabla J^k)$
Compute the model $\mathbb{M}$ and J^{k+1}
Compute the model assistant and ∇J^{k+1}
k $\leftarrow k+1$
end while
k $\leftarrow k-1$
end while
return $X(0)^k$ for each time t

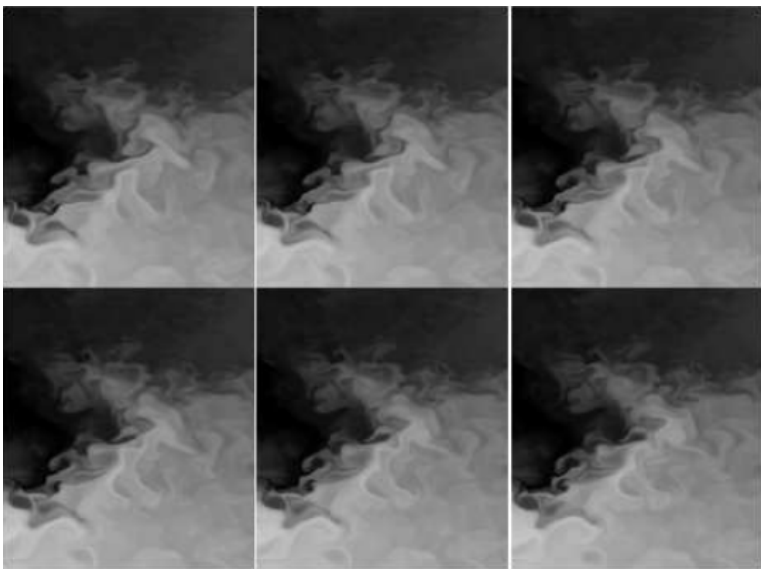

Figure 4.1: Satellite images of sea surface temperature[4]

$w(x,t)$ such that $w(x,t) = \begin{bmatrix} u(x,t) \\ v(x,t) \end{bmatrix}$ thanks to a physical model $\mathbb{M}$. We will then try to correct our model when searching for w. In order to optimize w, we will optimize a cost function, solutions of the equations 4.1, 4.4 and 4.5, using the BFGS optimizer as seen in appendix A.

We are analyzing ocean surface temperature images, 4.1, taken by different satellites

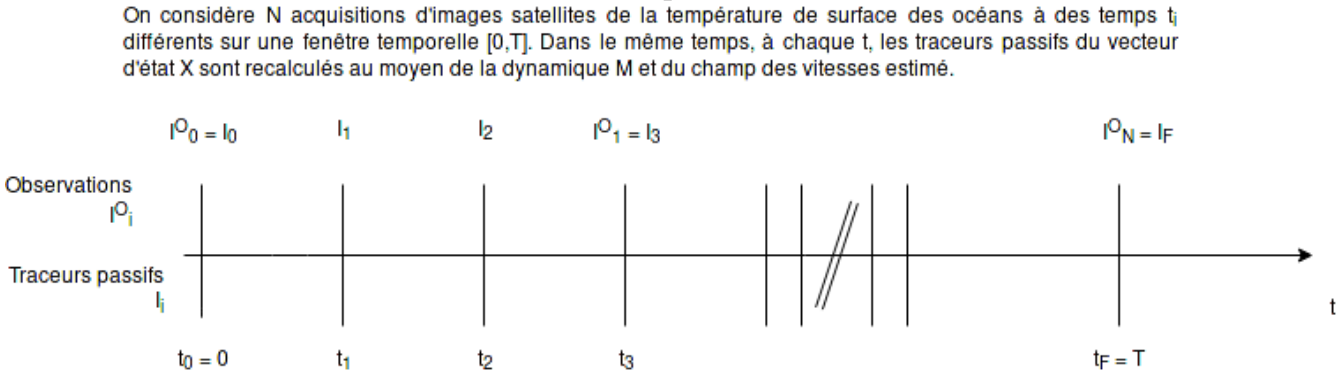

Figure 4.2: Diagram explaining the distribution of passive tracers and acquisitions in the time window

at different times in order to estimate a velocity field that reflects the dynamics of temperature evolution over time. This phase will be called the Motion Estimation phase, described by the 4D Var algorithm, 1.

In another part, we will try to inject this dynamic to the last known observation in order to predict the future evolution of the temperature. This phase will be called the Forecast phase.

4.2.3 Working hypothesis

We will therefore try to use Data Assimilation techniques to correct, using observations, the state of the atmosphere of a weather forecast. At each time, the passive tracers I of our state vector $\mathbf{X}$ are computed using the current dynamics and velocity field. The purpose of data assimilation will therefore be to minimize the cost function J, 4.7, by calculating ∇J compared to $X(t_0)$, with constraints on B, R, $\mathbb{H}, X_b, \mathbb{M}$ data. [4]

Even if we cannot have complete confidence in our observations, because they can be noisy because of the sensors, we do not modify these observations. This is the hypothesis of the field truth **(which is false, but essential to performing our calculations)**. The following assumptions should also be considered:

$$J(x,t) = ||\epsilon_B|||^2_B + \int_0^T ||^2_R dt + \alpha||\nabla w(t_0)||^2 + \beta||div(w(t_0))||^2 \tag{4.7}$$

$$\Leftrightarrow J(x,t) = ||X(t_0) - X_b||^2_B + \int_0^T |||\mathbb{Y}(x,t) - \mathbb{H}(x,t)|||^2_R dt + \alpha|||\nabla w(t_0)||^2 + \beta|||div(w(t_0))|^2 \tag{4.8}$$

We must also considers the following hypothesis:

- It is assumed that $\epsilon_R(x,t)$ and $\epsilon_B(x)$ are independent, Gaussian and unbiased

- α is the term of regularization on the velocity gradient
- β is the term of regularization on the rotational speed
- α and β are of the same order of magnitude as X_b, that is 10^7 10^{10}.
- In the initial state t_0, we will note:
  - w_0, the initial velocity field, initialized to 0, because we assume that before our observations are launched (i.e. t_0), nothing happens. Although this assertion is false, we may use it as we assume there is a ground truth through our observations, moreover, we are working on a limited space-time domain. That is why we will have a edge-effect that we will have to take into account. Nevertheless it does not hinder our predictions, because we are trying to find out what is happening within our study area and not on the edges.
  - I_b, our background or initial passive tracer to which observation is generally assigned at t_0.
- The covariance matrix B models the uncertainty on X_b and is defined as :

$$B(x) = \begin{bmatrix} B_w & 0 \\ 0 & B_I \end{bmatrix} \tag{4.9}$$

$$\Leftrightarrow B(x) = \begin{bmatrix} B_u & 0 & 0 \\ 0 & B_v & 0 \\ 0 & 0 & B_I \end{bmatrix} \tag{4.10}$$

- The uncertainty on matrix B actually depends only on $B_I(X)$. This is why, since this uncertainty is equal to the opposite of the uncertainty on the acquisition sensors of the observations, we will set B_I to 10^8.
- Since it is impossible to quantify the uncertainty of speed, because we have no information about speed, we will set B_w to 0.
- The matrix R models the uncertainty on the dynamics and is directly related to X_b. Usually $R(x,t)$ takes the same value as $B_I(X)$

4.2.4 Working basis

In the beginning of the internship, we had an efficient 4DVar code in C++. This code allows the motion estimation and forecast phases to be carried out. Our first goal was therefore to study this code and analyze it in order to then translate it to Python under the PyTorch library, see paragraph 5.1.1.

In this figure, the blank spaces correspond to a lack of sea-surface data i.e, they are landmasses.

We work with NetCDF data (satellite images of temperature, velocities and other state variables taken daily over several years) which a Those are sets that include images

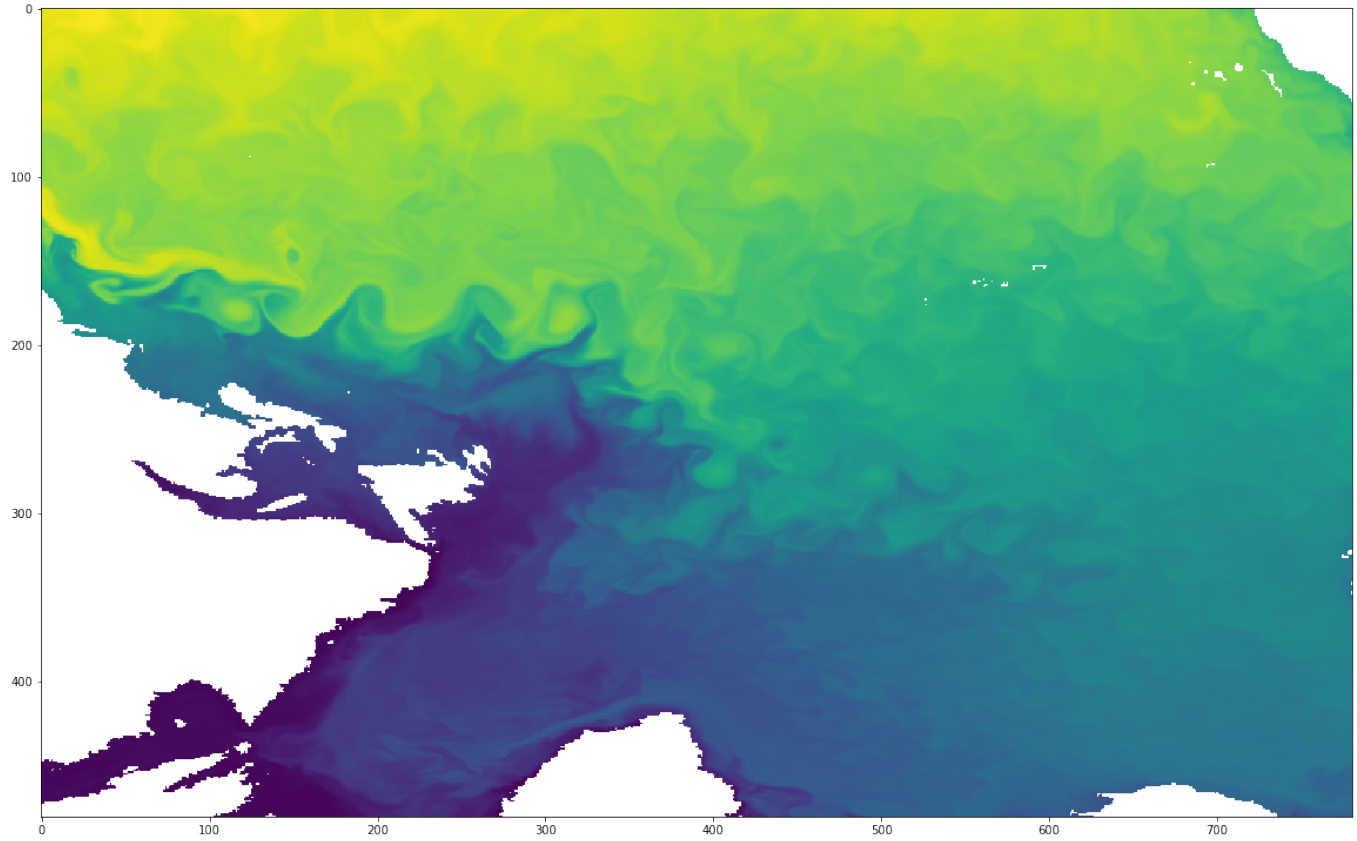

Figure 4.3: NetCDF raw data

of the surface temperature of the Atlantic Ocean, as well as many other state variables at any point in this image, see 4.3. Each data is taken at the same time, every day from satellites. In our study, our space steps dx and dy were set for images of shape 32x32, so, we only keep the data in areas that present a significant sea-surface velocities otherwise, the data assimilation algorithm won't perform well to avoid any numerical problems when computing our algorithm.

When we apply 4DVar (1), we initialize our background $\mathbf{X}_0 = \begin{matrix} 0 \\ 0 \ I_0 \end{matrix}$, where I_0 represents the temperature map at the initial time. Since we don't have prior knowledge of the motion field, it is initialized to 0. This assumption is false, but will be corrected through the optimization process. The optimizer takes as input the background error $\mathcal{E}_B(x, t_0)$ and the model errors $\mathcal{E}_M(x, t)$. These two errors are initialized as zero. It will then be up to the BFGS optimizer to update them for each run of our 4DVar algorithm.

Firstly, we do not remove the linear part, partially known thanks to the term non-linear advection in the equation of state, 4.1.

In order to compute $\mathbf{X}$(t) from $\mathbf{X}$(t-1), for t ranging from 1 to T_f, we use the state equations 4.1. These state equations are numerically computed using a first-order Godunov scheme for the non-linear part of the velocity, a first-order Upwind scheme for the linear part of the velocity and a first-order centered scheme for the passive tracer, which

is only composed of a linear part. Numerically, those numerical schemes are written as in Appendix B.

For avoiding any edge effects, we set the edge-value of the matrix of U and V as 0 and we duplicate the value of the borders for the matrix I.

During the optimization, the BFGS optimizer is used. It is an iterative algorithm for solving non-linear optimization problems without constraints or with weak constraints as in our case. It is a quasi-Newtonian optimizer, based on the computing of the gradient of the cost function and the gradient descent direction in order to update the parameters to be optimized, $\mathcal{E}_B$ and $\mathcal{E}_M$. It is therefore necessary to compute the gradient of the cost function 4.7 where the state model $\mathbb{M}$ and the observation model $\mathbb{H}$ are not linear and not necessarily inversible. In the code initially provided, it is necessary to calculate the model adjoint using automatic differentiation methods.

Until the number of iterations does not reach $MaxIter$ or until the cost function does not have a norm below a certain threshold or as long as the gradient norm is not below a certain threshold, the algorithm does not stopped. But, the numeric schemes, ref here, must ensure that the CFL conditions are met. Otherwise, the algorithm stopped with an error.

$$|u(x,t)| * \frac{dt}{dx} \leq 1 \tag{4.11}$$

$$|v(x,t)| * \frac{dt}{dy} \leq 1 \tag{4.12}$$

Numerically, we set the parameters dx, dy and dt in such a way that the CFL condition is as close as possible to 1 without exceeding it to respect the CFL condition to avoid any problem of explosion of the velocity field, therefore of the gradient as well as to avoid any numerical problem related to too low speeds. Therefore, we do not take dx, dy, dt neither too small nor too large.

However, in order to compute the adjoint of the model, to obtain the gradient of the cost function, 4.7, it was necessary to use automatic differentiation techniques to compute this adjoint. This automatic differentiation was achieved thanks to Tapenade, an INRIA software, which allows automatic differentiation. It allows to compute numerically derivatives of complex functions by decomposing them into basics functions whose derivatives are perfectly known. Nevertheless, this technique remains expensive in terms of memory space and not very flexible, as it does not permit the use of more complex models.

4.3 Literature review

4.3.1 Data-driven and model-driven approaches

Thanks to the improvement of satellite image sequences, it is possible to visualize the ocean surfaces as well as the dynamics induced by these images. Successfully using these images to predict weather phenomena is a complex project that is of interest to the scientific community. In their work on the Black Sea, D. BEREZIAT and I. HERLIN demonstrated that from the velocity advection equations and the surface temperature equation from Navier Stokes' equations, it was possible to use a data assimilation method from a cost function that minimized the difference between satellite observations and model values. [3]

In a second method, a variational approach, the 4D Var, was used. The acquisitions are placed temporally on a sliding window. From the model mentioned above and the observations, it is possible to determine the physical forces applied to the system based on a background vector. This background will initially be a first guess on the velocity and temperature fields, then through the iterative process of data assimilation, it will be improved to allow accurate prediction of model values from it. [4]

In a purely machine learning approach, R. FABLET has succeeded in demonstrating that it is possible to model dynamic systems using residual bilinear neural networks. In this context, his approach solved Lorenz's equations using several Runge-Kutta schemes of order 4. Instead of directly using this numerical scheme, the approach was to model a multi-layered neural network based on this scheme. The conclusive results of this study show that it is possible to model dynamic systems using machine learning and physical models. [9]

In an approach similar to ours, J. BRAJARD proposed a methodology to produce a hybrid model combining a known physical model and a neural model driven from observations. In this approach, the model used is a shallow-water model where the terms forcing, dissipation and diffusion are unknown. The study shows that it is possible to obtain very good results from this approach. [7]

4.3.2 Goal

Our final goal was therefore to adapt these 4DVar C++ codes to Python. One of the first objectives was to free ourselves from Tapenade by using only the automatic differentiation of Pytorch via the autograd module, then, after having validated our code by means of different twin experiments, we integrated machine learning into this code.

CHAPTER 5

Works during the internship

5.1 Translating 4DVar Code to Python

One of the first objectives of this internship was to translate the 4DVar code from C++ to Python. The Python language was chosen for its flexibility and its many machine learning libraries that are rather simple to use. Indeed, in the long term, we would like to set up a proof of concept to demonstrate that it is possible to combine data assimilation with deep learning.

5.1.1 Pytorch

To translate the C code to Python, the Pytorch module was chosen. Pytorch is an imperative langage module compared to Keras or Tensorflow which are symbolic programming module. In comparison with Keras or Tensorflow, Pytorch dynamically computes its graph. In case of neural network training, this means that the size of neural network does not require to be fixed. In case of data assimilisation, this means that it is not required to know forehand the number of iterations of running the 4DVar Algorithm. Also, with Pytorch, we can use our custom loss function.They have to be coded like in NumPy and one simply call the *.backward* function for retropropagating the gradient on this loss function. Thus, Pytorch uses Tensor which are similar to numpy arrays. That is another reason why we choose Pytorch as it makes easier to translate the numerical schemes from C to Python.

It allows us to completely free ourselves from the calculation of the adjoint model thanks to its autograd module which automatically calculates the function gradient from a graph of the calculation operations which allows us to retropropagate the gradient on

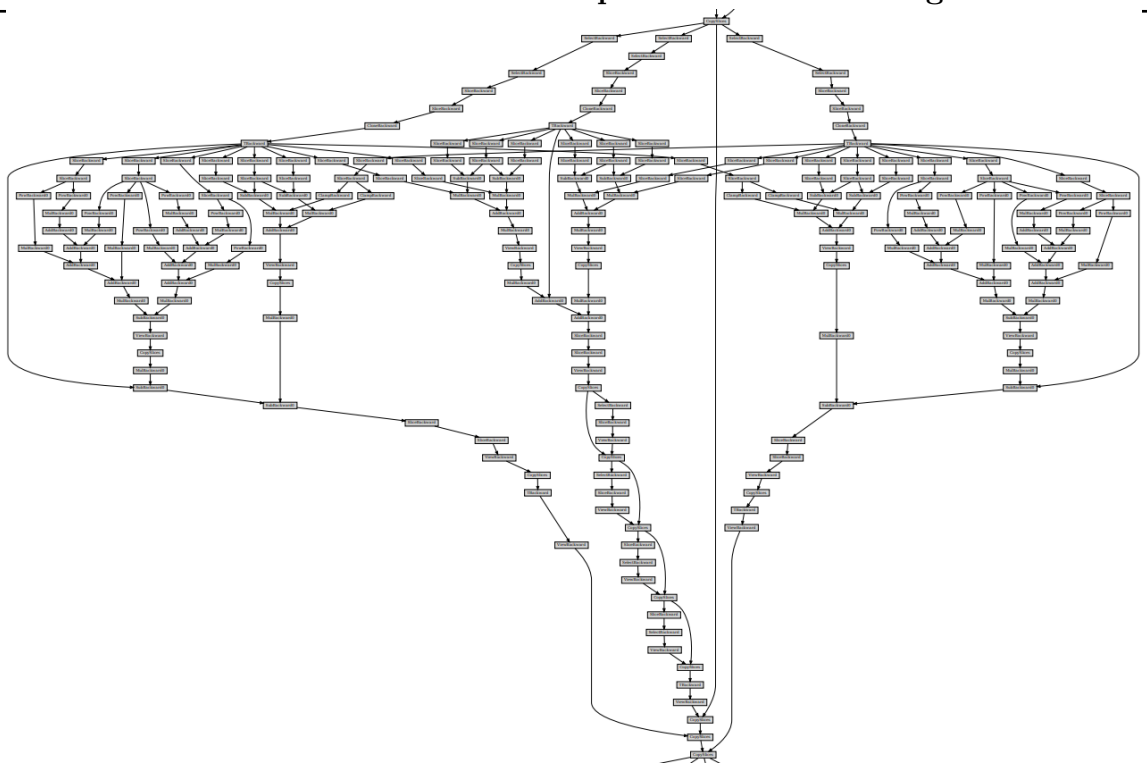

Figure 5.1: Graphe de calcul d'un 4DVar sous Pytorch

a tensor by using the *.backward* function.

The backward function allows the autograd module to record any operations made on variables for which we want to compute the gradient. As seen in the graph figure 5.1, every computational operations have been recorded in this graph. That is how Pytorch efficiently compute the gradient of $\mathbb{J}$ according the background error $\mathcal{E}_B$ of the model errors $\mathcal{E}_M$.

5.2 Evaluating the model

5.2.1 Testing the perfect model

To ensure the validity of our model, we carried out various tests to justify the veracity of the results obtained. We therefore first ensured that the data values generated using our perfect numerical scheme were of the same order of magnitude as those generated with the initial C code.

As part of this experiment, we chose a netCDF data size 32x32. We will integrate this data over a number of iterations N_t that we set at 31. We will then compare these data generated in Python with data generated in C, with the same netCDF data and with the numerical scheme in C++.

From these two data sets, a histogram of the average error on U, V, I is drawn.

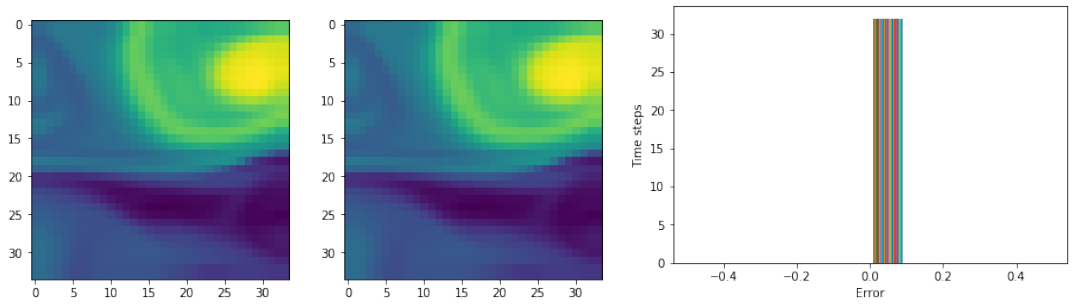

Figure 5.2: Generated image from the C++ Code (left), from the Python code (right) and the histogram of average errors on the whole set

The average RMSE error between the two sets is around 10^{-17} which is equal to the computer precision.

5.2.2 Gradient residue test

In order to validate our optimization with the Pytorch LBFGS, we performed a gradient test. This test ensures that the gradient is correctly calculated, in addition to ensuring that the same cost function value is obtained for iteration 0 between code C and code Pytorch. However, since gradient backpropagation is not done in the same way in C, where part of the gradient is not differentiated by the Tapenade tool, as in Python, where the gradient is automatically calculated by the Pytorch automatic differentiation module, we do not obtain exactly the same cost function values at the end of our optimization. As a result, we do not get strictly similar model errors and background error between C++ code and code under Python.

To validate the gradient computing, the gradient residue test can be performed. The approach consists in choosing an alpha disturbance coefficient greather than 0 as well as a delta disturbance, generated as a Gaussian matrix of the same size as the inputs of the

cost function, here, $\mathcal{E}_B$. We wish to evaluate the residual difference between the gradient of the cost function relative to $\mathcal{E}_B$ and the gradient of the cost function relative to $\mathcal{E}_B + \alpha$ *Δ. Through first order limited development of Taylor, the equation 5.1 gives :

$$\mathbb{J}(\mathcal{E}_B + \alpha * \Delta) = \mathbb{J}(\mathcal{E}_B) + \alpha * < \Delta, grad[\mathbb{J}(\mathcal{E}_B)] > + O(\alpha^2) \tag{5.1}$$

If we compute the logarithm of the residue of the quantities 5.2, we then obtain the following equation:

$$log(R(\alpha)) = log(\mathbb{J}(\mathcal{E}_B + \alpha * \Delta)) - log(\mathbb{J}(\mathcal{E}_B)) - log(\alpha) - log(< \Delta, grad[\mathbb{J}(\mathcal{E}_B)] >) \simeq 2 * log(\alpha) + C \tag{5.2}$$

Where:

- C is a constant offset.
- α ranges between 0 and 2.
- Δ was generated using a random normalized law of an unitary standard deviation and a null mean.

We are therefore trying to compute the slope of $log(R(\alpha))$ as a function of $log(\alpha)$. Theoretically, the slope of this function should be in the order of 2 to an offset, otherwise:

- If we choose an alpha that is too small, numerically, the computing will be wrong and the slope should tend towards 1.
- If you choose an alpha that is too large, numerically, the value of the slope will diverge.
- If our gradient is wrong, we will also get a slope of 1

We therefore performed this test with our Python code and our C code, in order to validate the calculation of the two gradients. In both cases, we obtain a correct gradient as shown in the figure:

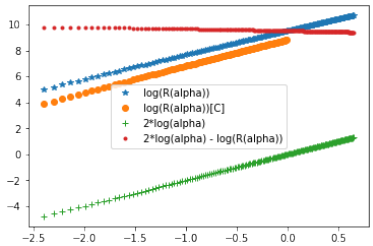

Figure 5.3: Graph of the residue of the gradient

5.2.3 Twin experiments

After validating our field truth and gradient computing, we wanted to make sure that we were able to do twin experiments. The latter consisted in generating a field truth from the perfect model. This field truth therefore contains N velocity and temperature data generated over a fixed time step. From these data, only the temperature map is kept. We then train our iterative algorithm of the 4DVar on these data, first in a perfect model, then with the imperfect model. The goal is therefore to know if we are able to find the velocity field in error on the background for the perfect model, and to find the velocity field missing for each time in the errors on the model in a second time. The purpose of these experiments is to ensure that our 4DVar is able to find the missing part of the dynamics for incomplete data. Once all these steps had been validated, it was then possible for us to move on to the important experience of this internship.

From the perfect model, introduced previously, we generate $N_t = 41$ velocity field and temperature map data to get our field truth. These data are tensors of size 3x32x32x32. The first plane corresponds to the hotizontal velocity U, the second plane to the vertical velocity V and the last plane corresponds to the temperature value. We take as no time dt = 8640, which means that each observation is spaced 10 time t steps or 1 day between each observation. We will take as no spaces dx=dy=10000, which means that each pixel corresponds to a surface of 100 km^2.

We thus obtain a state tensor $\mathbf{X}_{truth}$ of size 41x3x32x32x32 $(Nt, 3, Nx, Ny)$ where:

- t represents the time step
- Nx = 32 represents the size of the tensors according to the direction x
- Ny = 32 represents the size of the tensors according to the direction y
- Nt = 41 represents the number of time steps for each assimilation window.

We also obtain a tensor of observations I^O_{truth} of size 4x3x32x32 where the 4 observations retained are derived from data previously generated so that the observations are spaced 10 time steps apart (1 observation per day).

Then the covariance matrices R and B are fixed at 1. We set the regularization parameters α and β to 10^7 and to 10^9. Those two parameters are part of the cost function to be optimized by the LBFGS algorithm:

$$\mathbb{J} = ||X(t_0) - X_b||^2_B + \int_0^T ||\mathbb{Y}(x,t) - \mathbb{H}(x,t)||^2_R dt + \alpha||\nabla w(t_0)||^2 + \beta||div(w(t_0))||^2 \quad (5.3)$$

The more α is important, the more the velocity field will tend towards a translational movement. On the other hand, the more β will be important, the more the field of the tendencies will tend towards a rotational movement. These parameters must be set by the user through a battery of tests or a grid search in order to find the optimal parameters.

In this case, we used the alpha and beta parameters chosen in the 4DVar algorithm of the C code.

To optimize our cost function, we will use the 4DVar algorithm, 1. This algorithm uses the Pytorch LBFGS optimizer which requires the following parameters to be set:

- the maximal number of iterations, $MaxIter$. It will be set to 800. This means that the optimizer will not make more than this maximal number. If the optimization ends reaching $MaxIter$, the optimizer will return a value that might be close to a local minimum.
- the tolerance change. It will be set to $10^{-4} * Nx * Ny * 3$. If the difference of two successive values of the cost function is less than this value, the optimization stops.
- the accuracy of the optimizer, *accuracy*. This optimization parameter allows the user to define what precision he wants to obtain on the value of the minimum that the optimizer will return. Generally, in the literature, the values for this precision are as follows:
  1. 10^1 for excellent accuracy
  2. 10^7 for medium accuracy
  3. 10^{12} for low accuracy
- the tolerance on the gradient, *pgtol*. It will be set to *accuracy* * (machine computing precision). It is a termination tolerance on the norm of the gradient. When $grad\mathbb{J}_{\mathcal{E}} \leq pgtol$, the optimization stops.

For evaluating our model, we use the average relative norm errors and the average angular errors on the velocity field. The goal of the experiment is to get lower errors values than the C code.

In the case of the perfect model, at the end of the optimization, we obtain the following results:

Code	Average Angular Error	Average Relative Norm Error
Python	24.8899	0.1775
C	29.2903	0.1936

Figure 5.4: Results for the perfect model

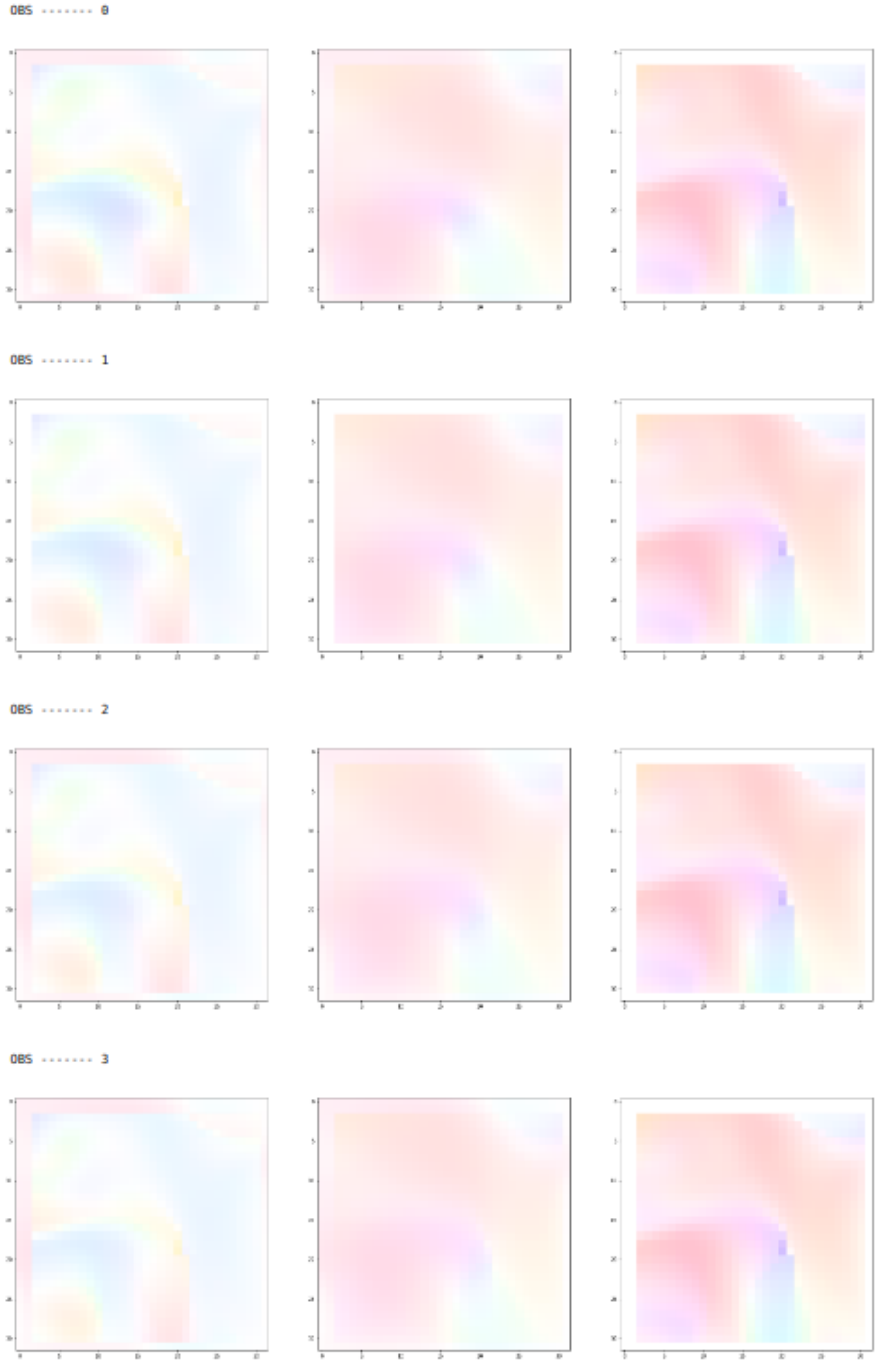

Figure 5.5: Results for the perfect model. Colorized velocity field. For each observations, on the right, the last velocity field computed by Python, on the middle, the last velocity field computed by C and on the left, the difference of those two velocities field

In the case of the imperfect model, the regularization parameter γ should be examined too. In our case, it has been set to 10^8 The cost function becomes:

$$\mathbb{J} = ||X(t_0) - X_b||_B^2 + \int_0^T ||\mathbb{Y}(x,t) - \mathbb{H}(x,t)||_R^2 dt + \alpha ||\nabla w(t_0)||^2 + \beta ||div(w(t_0))||^2 + \gamma \int_0^T \nabla \mathcal{E}_M(t) dt \tag{5.4}$$

And, at the end of the optimization, we obtain the following results:

Code	Average Angular Error	Average Relative Norm Error
Python	24.2402	0.1522
C	33.1865	0.2366

Figure 5.6: Results for the imperfect model

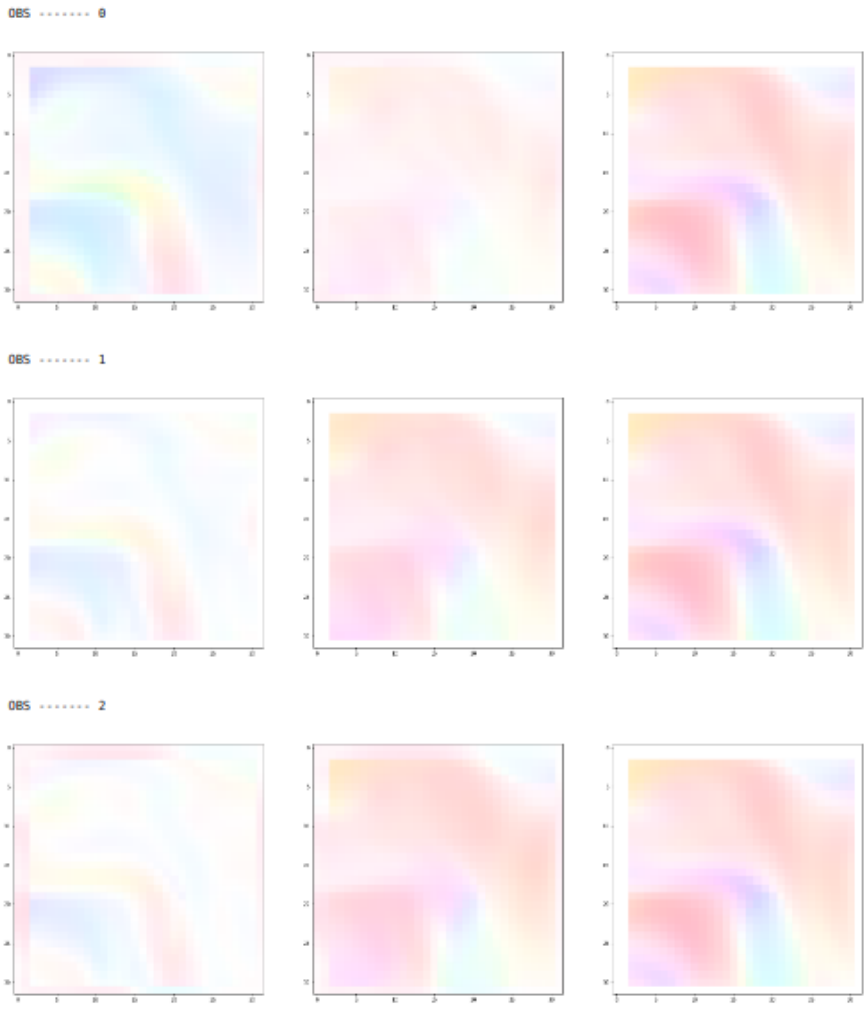

Figure 5.7: Results for the imperfect model. Colorized velocity field. For each observations, on the right, the last velocity field computed by Python, on the middle, the last velocity field computed by C and on the left, the difference of those two velocities field

5.3 Experiments

Firstly, from a NetCDF data (I, U, V), we had to generate a field truth by integrating it with the numerical scheme of the perfect model. We therefore made a forward on N_t = 31 time steps by taking a time step dt of 8640, which corresponds to 3 observations at times t = 10, 20, 30 or one observation per day as shown in Figure 5.9. The original data was not kept in the observations, see Figure 5.8.

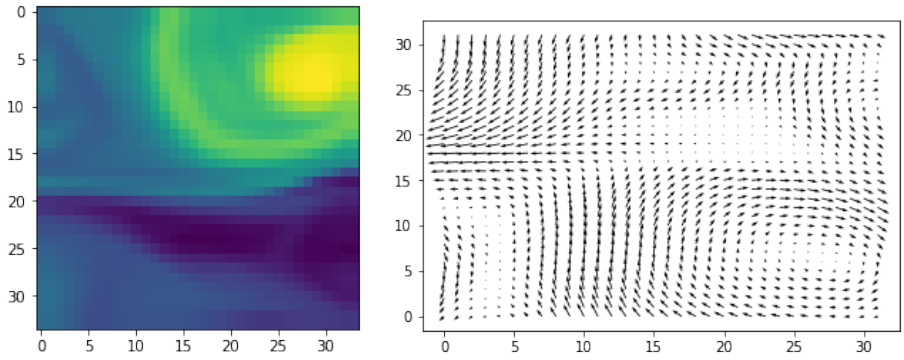

Figure 5.8: Initial sea surface temperature and initial motion field to be retrieved by data assimilation

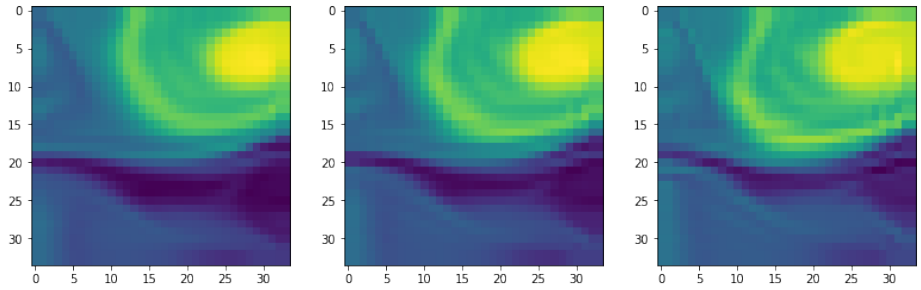

Figure 5.9: Observations of sea surface temperature at t = 10, 20 , 30

Second, for these 31 vectors generated, we masked the velocity field data (U and V), then, using the perfect 4D-Var, we tuned alpha and beta in order to reconstruct the background error as best we could. By using the same metrics as in section 5.2.3, we obtain a quite small norm and angular error. We have therefore validated this second step.

Average Angular Error	Average Relative Norm Error
79.915	0.1206

Figure 5.10: Results for the second step of the experiments

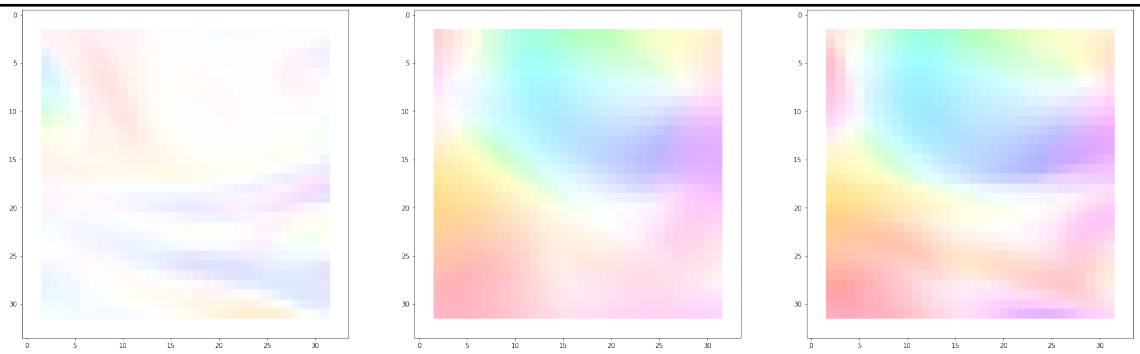

Figure 5.11: Results for the second step of the experiments. On the right, the colorized velocity field obtained by data assimilation. On the middle, the colorized background velocity field. On the left, the difference between those two velocity fields.

Thirdly, we degraded the integration scheme by removing the non-linear parts of the speed. By using the imperfect 4D Var, we tried to find the missing terms of the dynamics. We find a field of similar speeds with a small error in angle, but a significant error in standard norm. However, this may be explained by the chaotic nature of the data to be retrieved and their relatively low initial standard value. We already know that the 4DVar perform bad when trying to retrieve small velocity field due to the vanishing gradient problem. The main purpose of this experiment was to get the right background error for correcting the first guess and to get the non-linear part of the speed in model errors.

Average Angular Error	Average Relative Norm Error
80.1837	0.1150

Figure 5.12: Results for the third step of the experiments on the background error $\mathcal{E}_B$

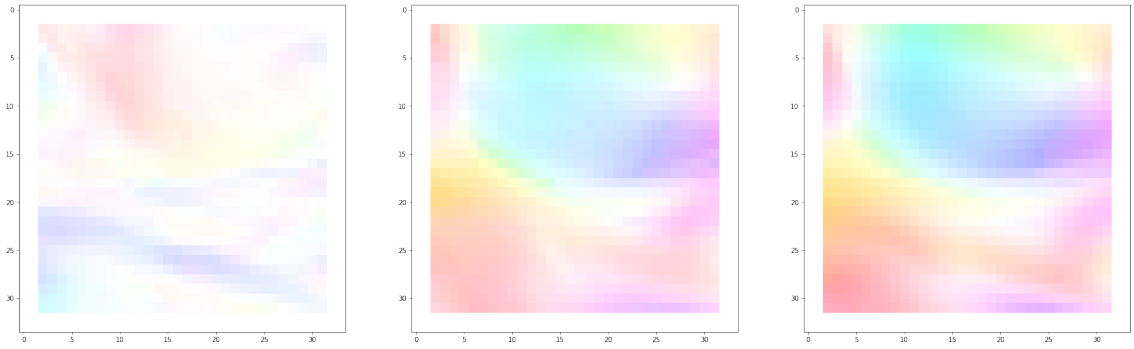

Figure 5.13: Results for the third step of the experiments on the background error $\mathcal{E}_B$. On the right, the colorized velocity field obtained by data assimilation. On the middle, the colorized background velocity field. On the left, the difference between those two velocity fields.

Average Angular Error	Average Relative Norm Error
89.9273	0.4373

Figure 5.14: Results for the third step of the experiments on the model errors $\mathcal{E}_M$

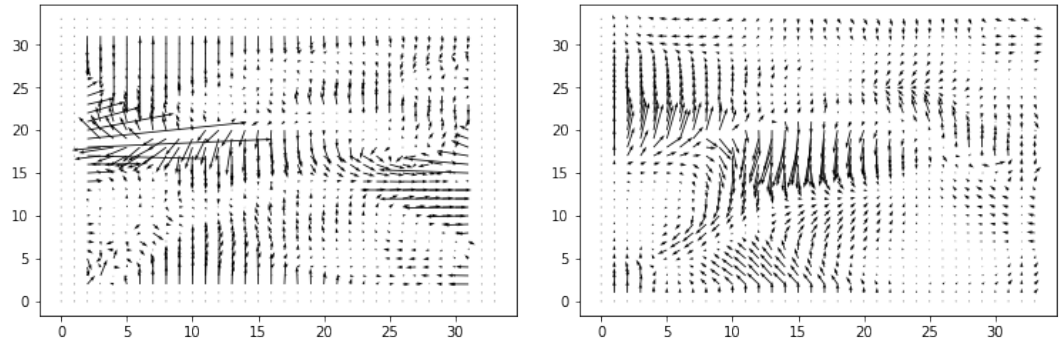

Figure 5.15: Results for the second step of the experiments on the model errors $\mathcal{E}_M$. For t = 20, on the left, the expected model errors. On the right, the model error computed by data assimilation

This third step was not successful. Indeed, we are getting strong relative errors on the velocity field standard. This is due to the low absolute value of the model errors we want to discover. This therefore confirms that it is not possible to estimate all the non-linearities of the model from data assimilation alone. It is necessary to use other processes, that is why we will try to find its non-linearities by means of deep learning.

The goal of the fourth step was to parameterize the model error in a neural network so that the neural network could learn this error in order to combine the 4DVar model, based on physical knowledge, and the neural network, based on the data.

In the later part of this report, the model will refer to the partial model containing only velocities U and V. Indeed, in the initial model, the tracer I has no non-linear dependencies, so it is not useful to integrate it into our databases.

This step is subdivided into 4 points:

- Create 6 synthetic training databases of size 10000 each where the inputs and outputs are as follows:
    1. Dataset 1 - X = Complete model at t-1 and Y = Non-linear part of the model at t ;
    2. Dataset 2 - X = Complete model at t-1 and Y = Complete model at t;
    3. Dataset 3 - X = Complete model at t-1 with the linear part of the t and Y model = Non-linear part of the t model ;
    4. Dataset 4 - X = Complete model at t-1 with the linear part of the model at t and Y = Complete model at t;
    5. Dataset 5 - X = Linear part of the t and Y model = Non-linear part of the t model ;

6. Dataset 6 - X = Linear part of the t and Y model = Complete model at t;

datasets.

- Create a neural network model capable of learning about these different databases.
- Recode the perfect (then imperfect) 4D Var algorithm to include the neural network driven on the non-linear parts of the dynamics in the physical model of the integration scheme.
- Find a way to drive the neural network while optimizing the model with 4DVar

The three last points will be discussed in the following sections.

5.4 Retrieving non linear part of the speed thanks to deep learning only

5.4.1 Generating the database

To generate the database, the initial data were extracted from the NetCDF data. To remove any bias, we integrated these images with the perfect integration scheme. These new data will constitute the model at t-1. Then, we reintegrate those data in order to have the data at t. We extract the data that are useful to us (either the complete data to have the complete model, or just the linear or non-linear part...). The Figure 5.16 show some samples of the dataset 2:

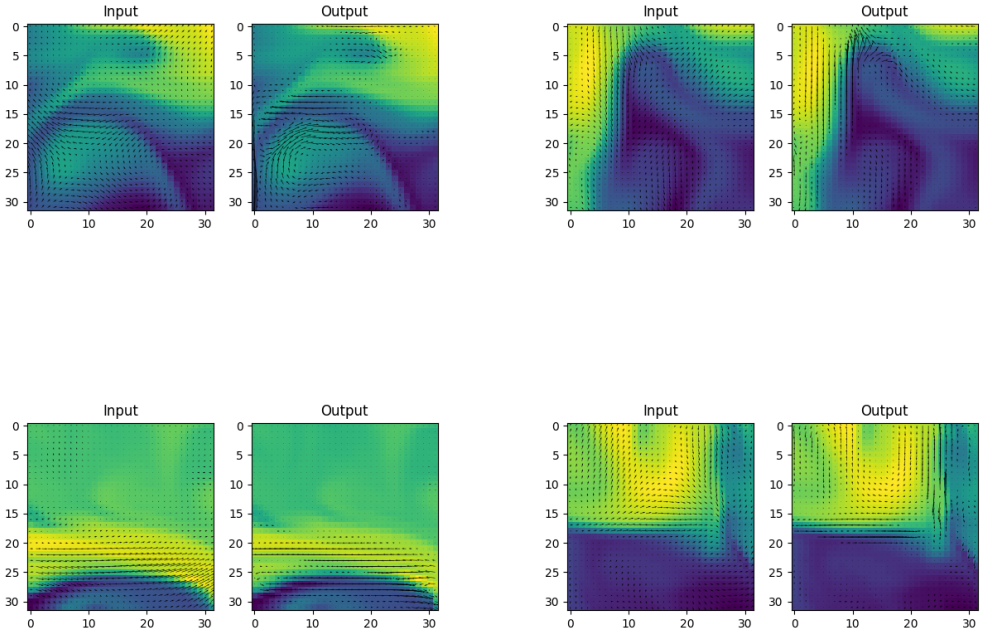

Figure 5.16: Samples of the dataset 2 - X = Complete model at t-1 and Y = Complete model at t

5.4.2 Learning the complete model

Our first goal was therefore to guess the complete model from t from different combinations of the complete or partial state vector $\mathbf{X}$ at t and t-1. To do this, we used dataset 1, 3 and 5. The first model we tested, based on the article [1], was the following:

To carry out the training, we chose to test our models on reduced sets, then to extend them to our complete database. We have therefore made the following choices:

Layer type	Output Shape
Batch Normalization	(BATCH_SIZE, 2, 32, 32) or (BATCH_SIZE, 4, 32, 32) for dataset 3
Conv2D	(BATCH_SIZE, 32, 32, 32) or (BATCH_SIZE, 32, 32, 32) for dataset 3
Conv2D	(BATCH_SIZE, 2, 32, 32) or (BATCH_SIZE, 2, 32, 32) for dataset 3

Figure 5.17: Model proposed for learning the complete model

- For the reduced train set, the size of the set is 960 data. For the whole train set, the size was set at 6400 data.
- For the reduced validation set, the size of the set is 320 data. For the whole validation set, the size was set at 2300 data.
- For the reduced test set, the size of the set is 320 data. For the whole test set, the size was set at 2300 data.

The metric chosen to evaluate the error of our model was the MSE as written in the equation 5.5. The cost function of the model is therefore written:

$$MSE = \frac{1}{N}\sum_{1}^{N}(o_i - y_i)^2 \tag{5.5}$$

With:

- N represents the number of data in our test set. This is N = 320.
- o_i represents the estimated output of our network for the i-th data.
- y_i represents the expected output of our network for the i-th data.

In addition, we plotted the scatter plot of our input/output model to determine if our network was performing properly. The closer the slope of this scatter plot is to 1, the better our network is. The red line has an unit-slope. Therefore, the neural network performs well.

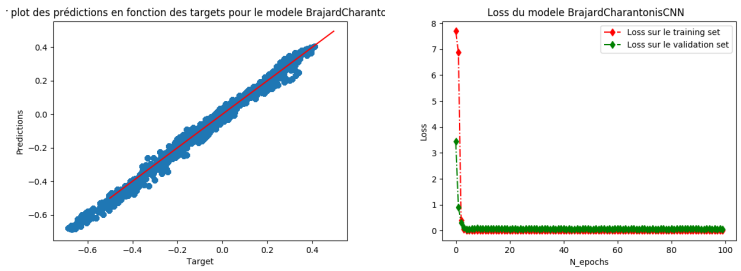

Figure 5.18: Left, scatter plot of the model on the reduced test set. Right, training and validation loss of the model

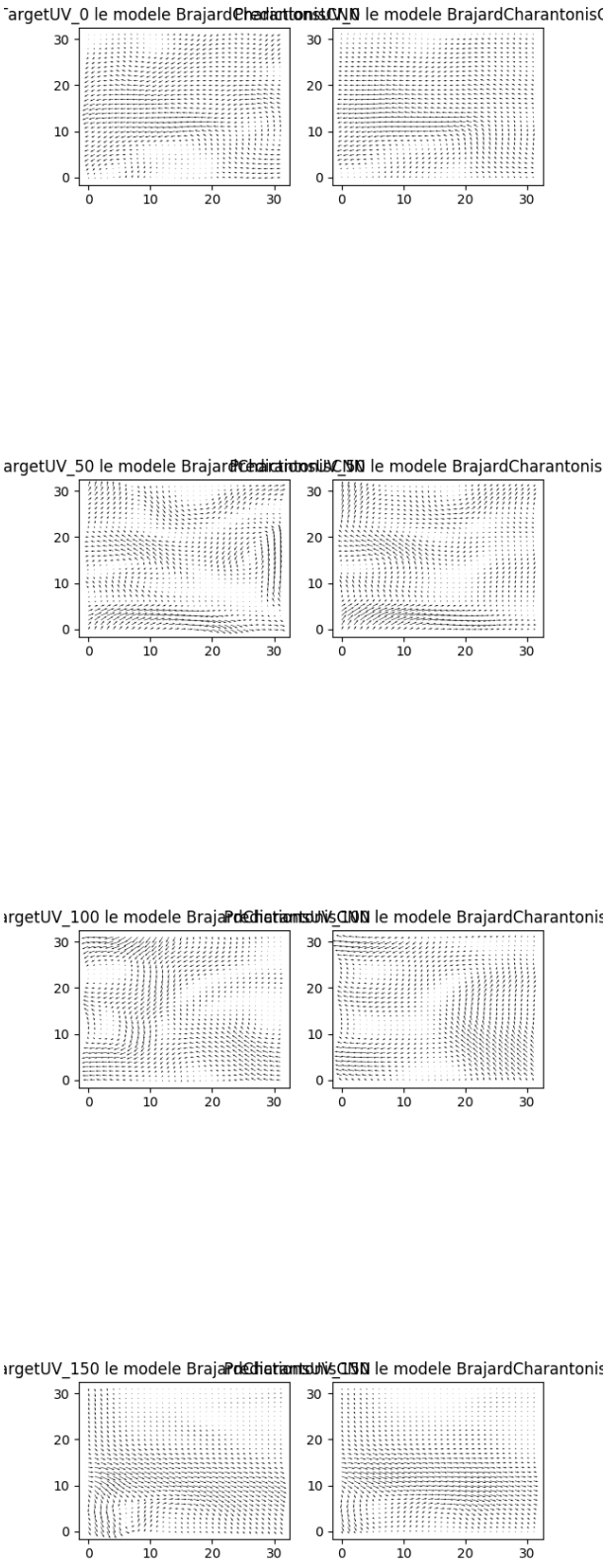

Figure 5.19: For each image, on the left is the expected output, on the right, the result of the model on the test set.

5.4.3 Learning the non-linear part only

After ensuring that it was possible to compute the complete model from t from the model at t-1 and/or its linear part at t, we tried to estimate the non-linear part of the model from the same inputs. Our second goal was therefore to guess the non-linear part of the model from t from different combinations of the complete or partial state vector $\mathbf{X}$ at t and t-1. To do this, we used the datasets 2, 4 and 6.

Firstly, we tried the 5.17 model, used to predict the complete model $\mathbf{X}(t)$. Because of the lack of sophistication of this model, it does not learn correctly the non-linear parts of the model. We have therefore set up a more complex model :

Layer type	Output Shape
Batch Normalization	(BATCH_SIZE, 2, 32, 32) or (BATCH_SIZE, 4, 32, 32) for dataset 4
Conv2D	(BATCH_SIZE, 32, 32, 32) or (BATCH_SIZE, 32, 32, 32) for dataset 4
Conv2D	(BATCH_SIZE, 32, 32, 32) or (BATCH_SIZE, 32, 32, 32) for dataset 4
Conv2D	(BATCH_SIZE, 8, 32, 32) or (BATCH_SIZE, 8, 32, 32) for dataset 4
Conv2D	(BATCH_SIZE , 2, 32, 32) or (BATCH_SIZE, 2, 32, 32) for dataset 4

Figure 5.20: Model proposed for learning the non-linear part of model

With the same cost function (5.5) as in section 5.4.2, we plotted the training and validation loss of the model 5.20. As shown in the Figures 5.21 and 5.22, our network does not perform well. That is why it is necessary to optimize the hyperparameters of our training process to achieve better results.

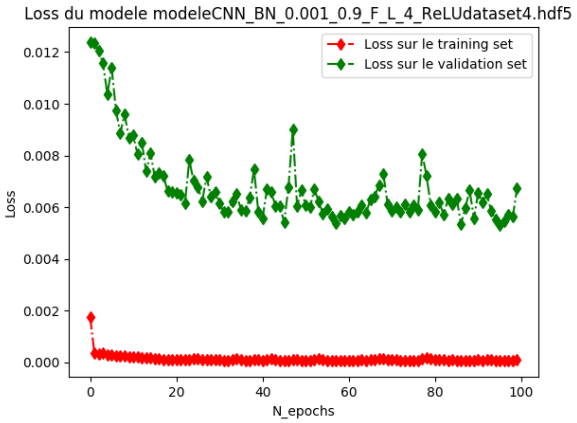

Figure 5.21: Training and validation loss of the model

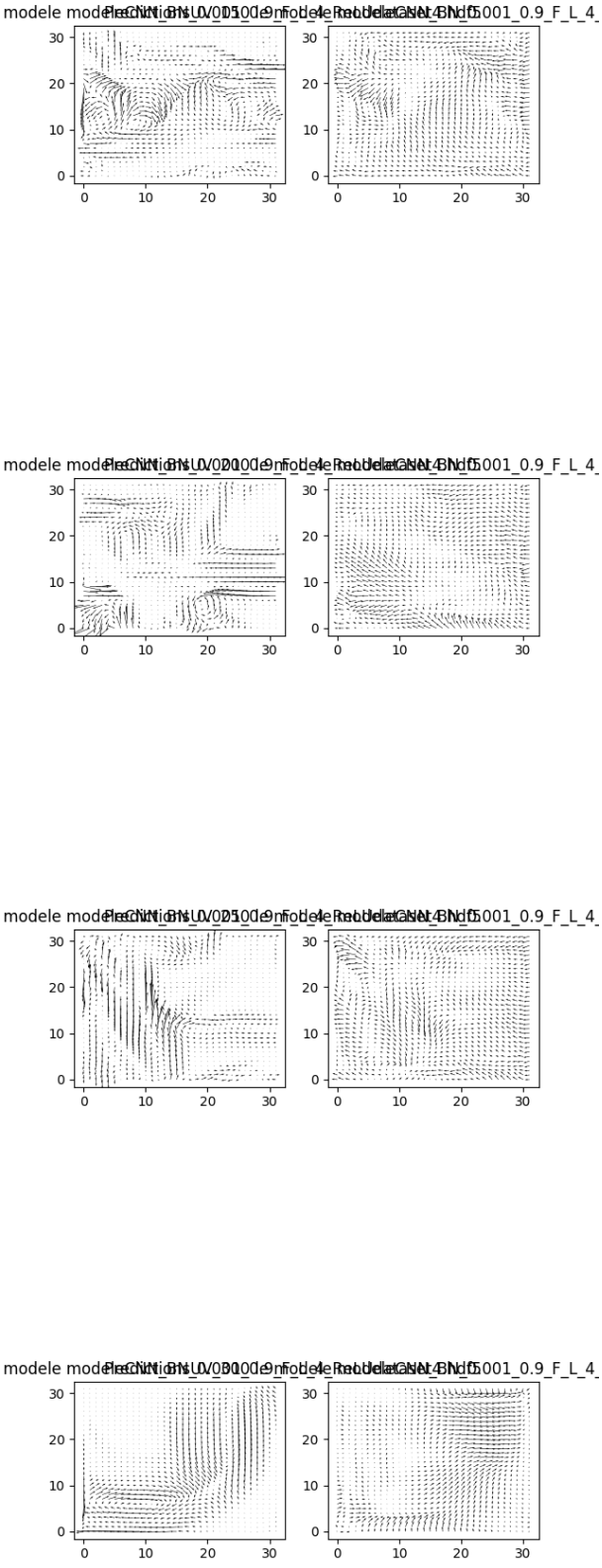

Figure 5.22: Estimating the non-linear part of model. For each image, on the left is the expected output, on the right, the result of the model on the test set.

5.4.4 Hyperparameter search

We were willing to know if it was possible to get better results by searching the hyperparameters of our neural network. The hyperparameters of the neural network are all parameters set by the user before training this network, to be distinguished from the parameters of the neural network which are the weights adjusted by this neural network. In these hyperparameters, we find those of the model as well as those of the optimizer used to compute the weights of the neural network. In our case, after testing different optimizers like the stochastic gradient descent, Adagrad, Adadelta and Adam, we kept the Adam optimizer. Although the network training with the latter is slower than with other optimizers, it guarantees stability in the search for optimal parameters as well as the path of the entire cost function with which our network is trained. The hyperparameters sought are therefore the following:

- The learning rate
- The maximal number of epochs (maximal number of iterations of optimization)
- The number of filter for each convolutional layer
- The size of the filter window
- The number of layers. In our case, a layer is composed of a 2D-convolutional layer, a ReLU activation and a Dropout layer
- The momentum of the batch normalization layer (first layer)
- The epsilon precision of the batch normalization layer. It is used for avoiding zero-division.
- The learning rate of the optimizer
- The betas of the Adam optimizer
- The L2-regularization or weight decay of the optimizer
- The L1-regularization or dropout for each layer.

To achieve this optimization, the scikit-learn module is used, which offers several methods for optimizing hyperparameters. In our case, we chose to use the Random Hyperparameter Optimization Search. Traditionally, hyperparameter optimization is perfomed by grid search which is an exhaustive search through an hyperparameter space defined by the user. This method tends to be time-costing if there is a lot of parameters to be tested. Another technique is the random search which randomly selects some values from the hyperparameter space. The metric for selected the best set of hyperparameters is specified by user. In our case, we will use MSE.

As an input to this method, the weights of the network to be optimized are given as well as several lists of values for the hyperparameters to be optimized. The search stops

when all combinations of values have been scanned or the maximum number of searches has been reached. The method then returns the optimal hyperparameters to us to obtain the lowest possible cost function value.

CHAPTER 6

After the internship

6.1 Combining deep learning with Data Assimilation

To combine data assimilation and machine learning, we choose to proceed in different steps.

First, we use a neural network, pre-trained in the previous step and we optimize our error on the background and our errors on the model as in the section 4 where the model would be written:

$$\mathbb{M}(\mathbf{X}(t)) = \mathbb{M}_{linear}(\mathbf{X}(t-1))+\mathbb{M}_{non-linear}(\mathbf{X}(t-1)+\mathbb{M}_{linear}(\mathbf{X}(t)) = \mathbb{M}linear+NN(\mathbf{X}(t-1)+\mathbb{M}_{linear}(\mathbf{X} \tag{6.1}$$

Once this step is achieved, we use our network with randomly initialized weights. We will perform some data assimilation cycles with the state model 4.1 in order not to start from errors on the draft and with a null neural network model.

In data assimilation, first guess errors are rarely zero, because it is very difficult and time consuming to compute these errors from scratch. Once a certain number of assimilation cycles have been performed, it will then be possible to proceed to the data assimilation training cycle with the machine learning.

At each iteration, we generate a training database for our neural network based on 300 observations generated by our data assimilation model. We are training our network as in the section 5.4.3. So we get a network with new weights that will constitute our new $\mathbb{M}_{non-linear}$. We then proceed with the 4DVar optimization with the model 6.1.

In the future, the goal is to write a cost function to carry out both the data assimilation process and the training of the neural network. This part will be the subject of my fellow

intern's thesis which will continue in thesis on a subject similar to that of the internship and with the same team.

Conclusion et perspectives

During this research internship, the aim was to improve ocean surface velocity and temperature predictions using data assimilation and machine learning techniques. With data assimilation, we are therefore able to obtain a great background for short-term predictions. However, given the difficulty of taking into account all non-linear terms by conventional numerical methods, we degraded the initial physical model into a linear physical model calculated by data assimilation and a non-linear model obtained using our machine learning algorithm.

To validate the code, we performed various validation tests to ensure that the results obtained by the programmed numerical methods were correct.

As a continuation of this internship, a thesis will be carried out by Arthur FILOCHE, which should lead to the creation of a model that combines the optimization of the data assimilation and machine learning process.

Finally, this internship gave me the opportunity to do a research internship at LIP6 while having the opportunity to work in a field that interested me a lot: artificial intelligence. I would have learned the methodology used in the research world, which gives me one more experience for my professional career that is very different from my previous ones.

Appendices

APPENDIX A

BFGS

The BFGS algorithm is an iterative quasi-newtonian optimization method that consists in :

- 1 - Generating an initial guess x_0 and an approximate inverse Hessian Matrix $B_0 = I$
- 2 - Compute a search direction p_k at step k by evaluting:

$$B_k p_k = -\nabla f(x_k) \tag{A.1}$$

  Where $\nabla f(x_k)$ is the gradient of the evaluated function f for the point x_k

- 3 - Perform a line search to find an acceptable step size α_k in the direction p_k and update:

$$x_{k+1} = x_k + \alpha_k p_k \tag{A.2}$$

APPENDIX B

Numerical methods

B.1 First order Upwind Scheme

In computational physics, upwind schemes are a type of numerical discretization method used for approximating first order derivate thanks to finite difference computation. This scheme is generally used for linear advection term like in equation **??**.

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \tag{B.1}$$

Numerically, it writes:

For $a > 0$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \tag{B.2}$$

For $a < 0$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0 \tag{B.3}$$

B.2 First order Godunov Scheme

The Godunov Scheme is a numerical method developped for computing a numerical solution for shallow water equation. Those equations contains non-linear term as they contains a product between a variable x and its partial first-order derivatives.

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(g(Q_{i-1}^n, Q_i^n) - f(Q_{i-1}^n, Q_i^n)) \tag{B.4}$$

Where f and g are given is our case. In both Upwind and Godunov scheme, a Courant–Friedrichs–Lewy condition must be satisfied to ensure the numerical stability of the schemes. It is written:

$$c = |\frac{a\Delta t}{\Delta x}| \leq 1 \tag{B.5}$$

B.3 First order Centered scheme

The first order centered scheme is a numerical method based on a first order Euler scheme where the derivatives is approximated by :

$$\partial_h[f](x) = f(x + 0.5h) - f(x - 0.5h) \tag{B.6}$$

APPENDIX C

Trajectory optimized

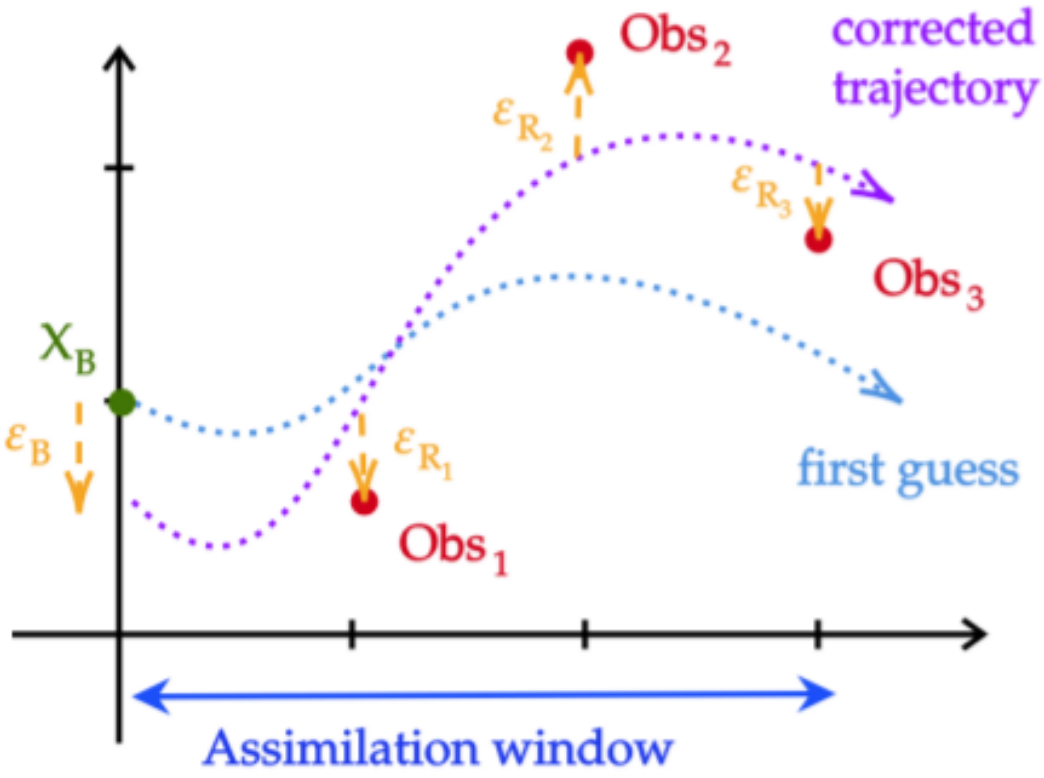

Figure C.1: Trajectory optimized through the 4DVAr Algorihtm

APPENDIX D

Gantt Chart

A Gantt chart is not necessarily appropriate, as I have completed a research internship. In accordance with the guidelines of ENSTA Bretagne, this document will therefore have a Gantt diagram that is not to be compared with an industrial Gantt diagram, but rather to be considered as a research plan for the internship.

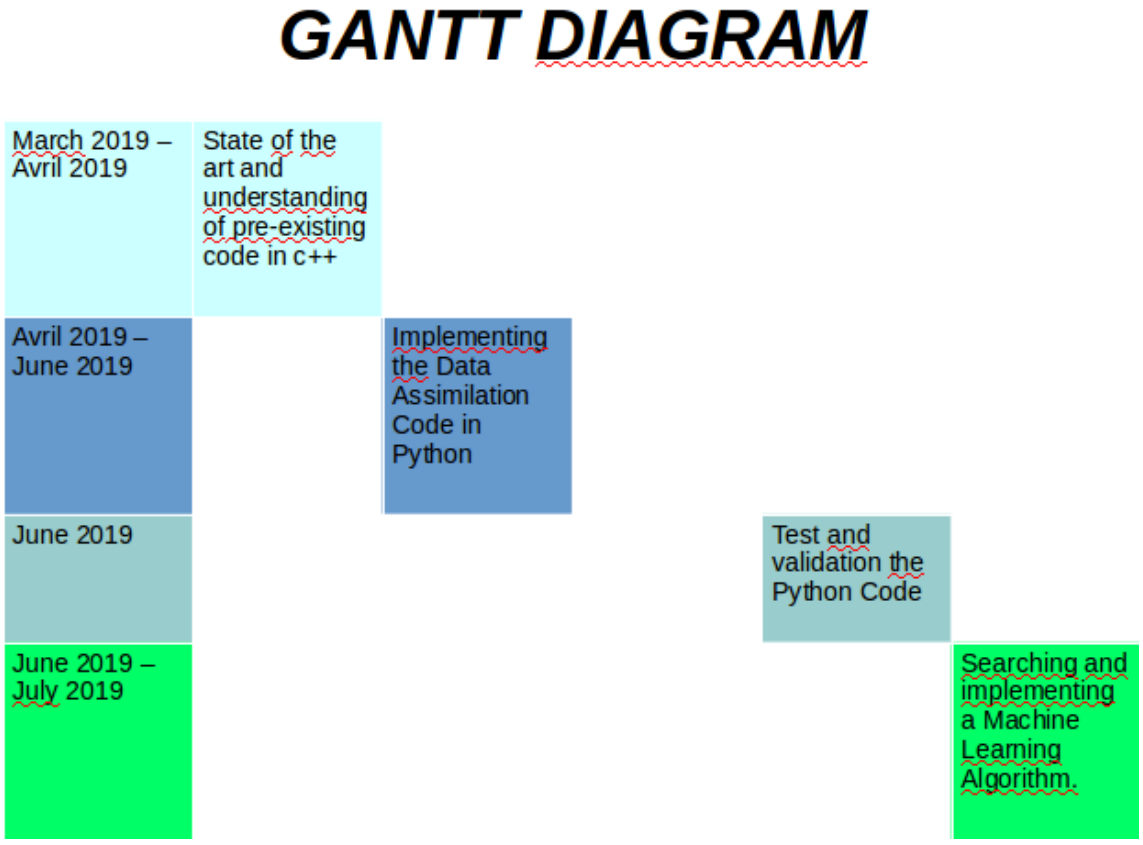

Figure D.1: Gantt Chart of the project

Bibliography

[1] S. Thiria A.A. Charantonis, F. Badran. Retrieving the evolution of vertical profiles of chlorophyll-a from satellite observations using hidden markov models and self-organizing topological maps. *Elsevier*, 0034-4257, 2015.

[2] Marc Bocquet. Introduction to the principles and methods of data assimilation in the geosciences : Lecture notes of the master m2 oacos and wape at École des ponts paristech. revision 0.25. 2018.

[3] Isabelle Herlin Dominique Béréziat. Image-based modelling of ocean surface circulation from satel-lite acquisitions. visapp - international conference on computer vision theory and applications,jan 2014, lisbon, portugal. *ScitePress*, 10.5220/0004669602880295. hal-00908791(83):288–295, 2014.

[4] Isabelle Herlin Dominique Béréziat. Motion and acceleration from image assimilation with evolution models. *Digital Signal Processing, Elsevier*, 10.1016/j.dsp.2018.08.008. hal-01857811(83):45–58., 2018.

[5] Olivier Talagrand François-Xavier Le Dimet. Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects. *Tellus*, 38A(10):97, 1986.

[6] Michelle Girvan Zhixin Lu Edward Ott Jaideep Pathak, Brian Hunt. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *PHYSICAL REVIEW LETTERS*, 120, 024102, 2018.

[7] Jérôme Sirven Julien Brajard, Anastase Charantonis. Representing ill-known parts of a numerical model using a machine learning approach. *Geophysical Research Letters and arXiv*, 1903.07358v1, 2019.

[8] Alberto Carrassi Marc Bocquet, Julien Brajard and Laurent Bertino. Data assimilation as a deep learning tool to infer ode representations of dynamical models. *Processes Geophys. Discuss., https://doi.org/10.5194/npg-2019-7*, 2019.

[9] Cédric Herzet Ronan Fablet, Said Ouala. Bilinear residual neural network for the identification and forecasting of dynamical systems. *arXiv*, 1712.07003v1 [cs.LG], 2017.