Data Assimilation in the Latent Space of a Convolutional Autoencoder

Maddalena Amendola[1], Rossella Arcucci[1], Laetitia Mottet[2], César Quilodrán Casas[1], Shiwei Fan[3], Christopher Pain[1,2], Paul Linden[4], and Yi-Ke Guo[1,5]

[1] Data Science Institute, Department of Computing, Imperial College London, UK
[2] Department of Earth Science & Engineering, Imperial College London, UK
[3] Department of Chemistry, University of Cambridge, UK
[4] Department of Applied Mathematics and Theoretical Physics, University of Cambridge, UK
[5] Department of Computer Science, Hong Kong Baptist University, HK

Abstract. Data Assimilation (DA) is a Bayesian inference that combines the state of a dynamical system with real data collected by instruments at a given time. The goal of DA is to improve the accuracy of the dynamic system making its result as real as possible. One of the most popular technique for DA is the Kalman Filter (KF). When the dynamic system refers to a real world application, the representation of the state of a physical system usually leads to a big data problem. For these problems, KF results computationally too expensive and mandates to use of reduced order modeling techniques. In this paper we proposed a new methodology we called Latent Assimilation (LA). It consists in performing the KF in the latent space obtained by an Autoencoder with non-linear encoder functions and non-linear decoder functions. In the latent space, the dynamic system is represented by a surrogate model built by a Recurrent Neural Network. In particular, an Long Short Term Memory (LSTM) network is used to train a function which emulates the dynamic system in the latent space. The data from the dynamic model and the real data coming from the instruments are both processed through the Autoencoder. We apply the methodology to a real test case and we show that the LA has a good performance both in accuracy and in efficiency.

Keywords: Data Assimilation · Machine Learning · Neural Network · Convolutional Autoencoder · Long Short Term Memory.

1 Introduction and Motivation

Data Assimilation (DA) is an approach for fusing data (observations) with prior knowledge (e.g., mathematical representations of physical laws; model output) to obtain an estimate of the distribution of the true state of a process [24]. In order to perform DA, one needs observations (i.e., a data or measurement model), a background (i.e., a priori state or process model) and information about the distribution of the errors on these two. In real world applications, DA is usually used to improve the accuracy of dynamic systems which represent the evolution

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_30

of some physical fields in time (e.g. weather, climate, ocean, air pollution). For those applications, the background is defined in big computational grids which lead to a big data problem sometimes impossible to handle without introducing approximations or space reductions. To overcome this problem, Reduced Order Modeling (ROM) techniques are used [1, 2]. ROM allows to speed up the dynamic model and the DA process. Popular approaches to reduce the domain are the Principal Component Analysis (PCA) and the Empirical Orthogonal Functions (EOF) technique both based on a truncated singular value decomposition (TSVD) analysis [17]. The simplicity and the analytic derivation of those approaches are the main reasons behind their popularity in atmospheric and ocean science. However, the physical interpretability of the obtained patterns is a matter of controversy because of the constraints satisfied by these approaches, e.g. orthogonality in both space and time. Also, despite those are powerful approaches, the accuracy of the obtained solution exhibits a severe sensibility to the variation of the value of the truncation parameters. This issue introduces a severe drawback to the reliability of these approaches, hence their usability in operative software in different scenarios [16]. An approach to reduce the dimensionality maintaining information of the data is the Neural Network (NN), precisely the Autoencoders. NNs have the ability to fit functions and they can fit almost any unknown function theoretically. That is the ability which makes it possible for neural networks to face complex problems. Autoencoders (AE) with non-linear encoder functions and non-linear decoder functions can thus learn a more powerful non-linear generalization of methods based on TSVD. The codomain of an encoder function is named latent space. The latent space is also the domain of the decoder function. In the latent space, the evolution of the transformed state variables defined in time, is still a dynamical system. Thanks to what the Universal Approximation Theorem [14, 21] claims, we can assume that any non-linear dynamical system can be approximated to any accuracy by a Recurrent Neural Network (RNN), with no restrictions on the compactness of the state space, provided that the network has enough sigmoidal hidden units. In the present work, we propose a new methodology we called Latent Assimilation (LA). It consists in reducing the dimensionality with an AE and perform both prediction through a surrogate dynamic model and DA directly in the latent space. In the latent space, the surrogate dynamical system is built by a RNN which learns the dynamic of the transformed variable. We apply LA to improve the prediction of air flows and indoor pollution transport. In fluid dynamics problems such as the propagation of air pollution, the data represents physical variables that are spatial distributed and contains information about geographical position (e.g. the sensor placement). We have to take into account those variables in our process. On the other hand, usually, such problems are defined in high dimensional spaces. To reduce the complexity of the problem we use the Autoencoder. The AE performs a non-linear transformation on the input. We process the system states and the observations coming from sensors using the same AE. In this way, we have a latent version of the model states and observations transforming the physical variables in the same way.

1.1 Related works and contribution of the present work

The use of machine learning in correcting model forecasts is promising in several geophysics applications [15, 10]. In some operational centres, data driven models have been already introduced to support CFD simulations [13, 20, 9]. The future challenges of numerical weather prediction (NWP) include more accurate initial conditions that take advantage of the increasing volume of real-time observations, and improved post-processing of model outputs, among others [9]. Neural networks for correction of error in forecasting have been previously studied in [5–7]. However, in this literature, the error correction by NN does not have a direct relation with the updated model system in each step and the training is not on the results of the assimilation process. In [3, 25] the authors describe a framework for integration of NN with physical models by DA algorithms. In this approach, the NNs are iteratively trained when observed data are updated. However, this approach presents a limit due to the time complexity of the numerical models involved, which limits the use of the forecast model for large data problems. In [11], the authors presented an approach for employing artificial neural networks (NNs) to emulate the local ensemble transform Kalman filter (LETKF) as a method of data assimilation. In [19], the authors combined Deep Learning and Data Assimilation to predict the production of gas from mature gas wells. They used a modified deep Long Short Term Memory (LSTM) model as their prediction model in the EnKF framework for parameter estimation. In [8] and [23], modified versions of KF based on NNs are applied to simulated pendulum and other four visual tasks (an agent in a plane with obstacles, a visual version of the classic inverted pendulum swing-up task, balancing a cart-pole system, and control of a three-link arm with larger images), respectively.

In the present work, we focus on the problem of assimilating data to improve the prediction of air flows and indoor pollution transport [22]. We propose a methodology we called Latent Assimilation (LA) which consists in:

- **Dimensionality reduction:** The dimensionality of both background of the dynamic model and observations (real data coming from the instruments) is reduced by Autoencoder;
- **Surrogate Model:** A data driven version of the dynamic model we call "surrogate dynamic model" is build using RNN. In particular, an LSTM is used to train a function which emulates the dynamic system in the latent space;
- **Data Assimilation:** In the latent space, DA is performed by a modification of a standard KF;
- **Physical space:** The results of the DA in the latent space are then reported in the physical space through the Decoder.

In this paper, we apply Latent Assimilation to improve the prediction of air flows and indoor pollution transport. However, the technology and the model are very general and can be applied to other kinds of computational fluid dynamic systems which simulate other dynamical systems. Both prediction and DA show a speed up in the reduced space. We apply the methodology to a real test case and we

show that the LA has a good performance both in accuracy and in time.
This paper is structured as follows. In the next section the Kalman Filter is described. Then, it follows a section where the Latent Assimilation is introduced. Next, experimental results are provided. Finally, in the last section we summarize conclusions and future works.

2 Kalman Filter

DA merges the estimated state $x_t \in \mathbb{R}^n$ of a discrete-time dynamic process at time t:

$$x_{t+1} = M_{t+1} x_t + w_t \tag{1}$$

with an observation $y_t \in \mathbb{R}^m$:

$$y_t = H_t x_t + v_t \tag{2}$$

where M_t is a dynamics linear operator and H_t is called linear observation operator. The vectors w_t and v_t represent the process and observation errors, respectively. They are usually assumed to be independent, white-noise processes with Gaussian probability distributions

$$w_t \sim \mathcal{N}(0, Q_t), \quad v_t \sim \mathcal{N}(0, R_t)$$

where Q_t and R_t are called errors covariance matrices of the model and observation respectively.
DA is a Bayesian inference that combines the state x_t with y_t at each given time. The Bayes theorem conducts to the estimation of x_t^a which maximise a probability density function given the observation y_t and a prior from x_t. This approach is implemented in one of the most popular DA methods which is the Kalman filter (KF) [18]. The goal of the KF is to compute an optimal a posteriori estimate, x_t^a , that is a linear combination of an a priori estimate, x_t, and a weighted difference between the actual measurement, y_t , and the measurement prediction, $H_t x_t$ as described in equation (5) where the quantity $d_t = y_t - H_t x_t$ is called misfit and represents the distance between the actual and the predicted measurements at time t. For big data problems, KF is usually implemented in a simplified version as an Optimal Interpolation method [4] which consists of fix the covariance matrix $Q_t = Q$, for each time step t. It mainly consists of two steps: a prediction and a correction step:

1. **Prediction**:
$$x_{t+1} = M_{t+1} x_t^a \tag{3}$$
2. **Correction**:
$$K_{t+1} = QH^T (HQH^T + R_{t+1})^{-1} \tag{4}$$
$$x_{t+1}^a = x_{t+1} + K_{t+1}(y_{t+1} - H x_{t+1}) \tag{5}$$

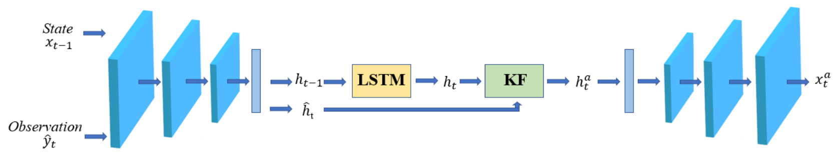

Fig. 1. The work flow of the Latent Assimilation model. Let us assume that we want to predict the state of the system at time t and we assume that the LSTM needs one observation back to predict the next time-step. The input of the system is the state x_{t-1}. We encode x_{t-1} producing its encoded version h_{t-1}. From h_{t-1} we compute h_t through LSTM. To perform the Kalman Filter, we need the observation y_t at time-step t. We encode y_t and we combine the result, $\hat{h}_t$, with the prediction h_t through the KF. The result h_t^a is the updated prediction. We report the updated prediction in its physical space through the Decoder, producing x_t^a.

The prediction-correction cycle is complex and time-consuming and it mandates the introduction of simplifications, approximations or data reductions techniques. In the next section, we present the Latent Assimilation approach which consists in performing KF in the latent space of an Autoencoder with nonlinear encoder and nonlinear decoder functions. In the latent space, the dynamic system in (3) is replaced by a surrogate model built with a RNN.

3 Latent Assimilation

Latent Assimilation is a model that implements the new idea of assimilating real data in the Latent Space. Instead of using PCA or others mathematical approaches to reduce the space, we decide to experiment the reduction with non-linear transformations using Deep NNs. Specifically, we choose to use Convolutional Autoencoder to reduce the space.
Figure 1 is a graphical representation of the Latent Assimilation model.
The model is structured in four main parts:

Dimensionality reduction: The dimensionality reduction is implemented by a Convolutional Autoencoder which produces a representation of the state vector $x_t \in \mathbb{R}^n$ in (1) in a "latent" state vector $h_t \in \mathbb{R}^p$ defined in a Latent Space where $p < n$. We denote with $f : \mathbb{R}^n \to \mathbb{R}^p$ the Encoder function

$$h_t = f(x_t) \tag{6}$$

which transforms the state x_t in a latent variable h_t.

Surrogate model: In the latent space we perform a regression through a LSTM function $l : \mathbb{R}^{p\times q} \to \mathbb{R}^p$

$$h_{t+1} = l(\boldsymbol{h_{t,q}}) \tag{7}$$

where $\boldsymbol{h_{t,q}} = \{h_i\}_{i=t,\ldots,t-q}$ is a sequence of q encoded time-steps up to time t.

Data Assimilation: The assimilation is performed in the latent space. In order to merge the observations in (2) with the "latent" state vector h_t, the observations are processed by the Encoder in the same way as the state vector. As $y_t \in \mathbb{R}^m$ where usually $m \leq n$, i.e. the observations are usually held or measured in just few point in space, the observations vector y_t is interpolated in the state space $\mathbb{R}^n$ obtaining $\hat{y}_t \in \mathbb{R}^n$. The observations $\hat{y}_t$ are then processed in the same way as the state vector trough f:

$$\hat{h}_t = f(\hat{y}_t) \tag{8}$$

The "latent" observations $\hat{h}_t$, transformed by the Encoder in the latent space, are then assimilated by the prediction-correction steps as described in equations (9)-(11):

1. **Prediction**:
$$h_{t+1} = l(\boldsymbol{h_{t,q}}) \tag{9}$$
2. **Correction**:
$$\hat{K}_{t+1} = \hat{Q}\hat{H}^T(\hat{H}\hat{Q}\hat{H}^T + \hat{R}_{t+1})^{-1} \tag{10}$$
$$h^a_{t+1} = h_{t+1} + \hat{K}_{t+1}(\hat{h}_{t+1} - \hat{H}h_{t+1}) \tag{11}$$

where l in (9) is the surrogate model defined in (7) computed by the LSTM, $\hat{Q}$ and $\hat{R}$ are the errors covariance matrices of the transformed background h_t and observations $\hat{h}_t$ respectively, they are computed directly in the latent space. $\hat{H}$ is the observation operator: if $m = n$, it is the identity function, otherwise it is an interpolation function. The background covariance matrix $\hat{Q}$ is computed with a sample of s model state forecasts $\boldsymbol{h}$ that we set aside as background such that:

$$\boldsymbol{h} = [h_1, ..., h_s] \in \mathbb{R}^{p\times s}, \quad V = (\boldsymbol{h} - \bar{h}) \in \mathbb{R}^{p\times s} \tag{12}$$

where $\bar{h}$ is the mean of the sample of background states, then $\hat{Q} = VV^T$. The observations errors covariance matrix $\hat{R}$ can be computed with the same process in (12) replacing h_t with $\hat{h}_t$, $\forall t$ or, it can be estimated by evaluations of measurements (instruments) errors. $\hat{K}$ is the Kalman Gain matrix defined in the latent space and $\hat{H}$ is the observation operator.

Physical space: The results of the DA in the latent space are then reported in the physical space through the Decoder, applying the function $g : \mathbb{R}^p \to \mathbb{R}^n$ to compute

$$x^a_{t+1} = g(h^a_{t+1}). \tag{13}$$

The Decoder is almost a mirror of the Encoder: it is composed of a Fully Connected Layer followed by some Convolutional Layers.
In the next Section we apply Latent Assimilation to the problem of assimilating data to improve the prediction of air flows and indoor pollution transport in a real scenario [22]. We show the performance of the model step by step and we compare results with a standard DA performed in the physical space.

4 Experimental Results

Latent Assimilation is here applied to merge indoor air quality measurements from sensors with an indoor air quality simulation made by a computational fluid dynamic (CFD) software named Fluidity.

CFD data: The domain is an office room in the Clarence Centre in the Borough of Southwark, London [22]. The CFD simulation constitutes a time series composed of 3500 time-steps. For each time step, the CFD simulation provides a matrix of dimension 180×250 where each value of the matrix represents the concentration of CO_2 expressed in PPM in a specific location of the room.
The time series we use is composed of 2500 time-steps, then the size of the data set is $\mathcal{O}(10^8)$ just considering the data related to the CFD simulation. The CFD data represents the real simulation of a flow and it doesn't change much between two consecutive steps. For this reason, we decided to divide the data in train, validation and test set making small jumps. We consider two consecutive time-steps for train and we make a jump. Every position not considered yet (the ones we jump) is assigned to validation and test set alternately.
Considering a jump=1, the series is divided as in Figure 2:

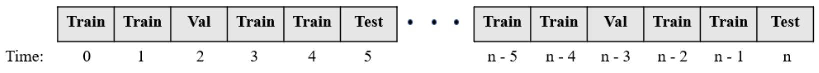

Fig. 2. Train, Validation and Test split

Observed data from sensors: For the observations, we have measurements from 7 sensors providing information for 10 time-steps. The sensors are spatially distributed in the room. In fluid dynamics problems, the data contains physical variables that are spatial distributed and contains information about geographical position. It's important to maintain those information during the data assimilation process. We preserve them processing both CFD and observed data with the same AE. We first bring the observation in the same space of the CFD data using an interpolation function. Then we process the system states and the observations coming from sensors using the same AE. In this way, we have a latent version of the model states and observations transforming the physical variables in the same way. The measurements from sensors are extended in the radius of 30cm in the room. Then the values are linearly interpolated using the Scipy library obtaining matrices of the same dimension of the data from the CFD, i.e. 180×250. The interpolation is repeated for each time-step. All the data are normalized in the range [0, 1]. We normalized the dataset with Min-Max Normalization considering the global minimum and maximum PPM values found in both dataset and observations.
The LA code and the pre-processed data can be downloaded using the link `https://github.com/DL-WG/LatentAssimilation`.

4.1 Dimensionality reduction

The Autoencoder we implement is a Convolutional Autoencoder. In particular, the Encoder is composed of some Convolutional layers followed by a Fully connected layer that determines the shape of the Latent space. The Decoder is nearly the mirror of the Encoder.
The construction of the Autoencoder is divided in two steps: the search of the structure and the Grid Search of hyper-parameters. All the experiments were performed with 4 GPUs K80.

Structure: The search of the structure is conducted progressively. We start comparing few autoencoders (that, for example, differ on the number of hidden convolutional layers). We pick the best structure and we create new configurations to compare. The model used to create new configurations, is here called *baseline*. We use the Mean Squared Error[6] (MSE) as metrics to evaluate the model. For each configuration, we compute a 5-Cross Fold Validation[7] (CV) and we choose the model with lower mean and standard deviation of the MSE values (Mean-MSE and Std-MSE respectively). A low standard deviation tells us that the model is stable and it does not depend on the data we use to train and validate it. We fix the following parameters: number of filters 32, activation function ReLu[8], Kernel size 3, latent space 7, optimizer Adam, epochs 300 and batch size 32. We use the MSE as loss function. This choice of value for the latent space is the result of an analysis of accuracy and efficiency.
We shuffle the data before to start to make the neural network independent from the order of the data. We first check the good number of hidden Convolutional layers. We try three different configurations:

1. Encoder with 3 Convolutional layers and Decoder with 4 Convolutional layers
2. Encoder with 4 Convolutional layers and Decoder with 5 Convolutional layers
3. Encoder with 5 Convolutional layers and Decoder with 6 Convolutional layers

Comparing in table 1 configurations 1, 2 and 3 for which only the number of convolutional layers is changing, configuration 2 is the one highlighting the best performance in terms of both Mean-MSE and Mean-MAE with MSE two order of magnitude lower than configurations 1 and 3. Moreover, configuration 2 is the most stable regarding the standard deviations, reflecting well that this CAE network architecture does not depend on the data used to train and validate it. In addition, the execution time of configuration 2 is relatively acceptable to answer real-time problems. Hence, in the following, the number of layers is taken as the same than configuration 2: 4 for the encoder and 5 for the decoder.

[6] `https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredError`
[7] `https://scikit-learn.org/stable/modules/cross_validation.html`
[8] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU`

Table 1. Convolutional AutoEncoder performance. N denotes the configuration number as listed in the main text. Time is given in seconds.

N	Mean-MSE	Std-MSE	Mean-Time	Std-Time
1	1.324e-02	2.072e-02	775.665	5.606e+00
2	**2.435e-04**	**3.851e-05**	**812.293**	**1.436e+00**
3	2.293e-02	2.763e-02	828.328	1.574e+00

Configuration 2 is then our *baseline*. We substitute the Convolutional layers with the Convolutional Transpose layers in the Decoder and we will call this the configuration number 4. As we can see from Table 2, the accuracy (Mean-MSE) and the stability (Std-MSE) are slightly better, while the execution time is slightly longer, when using convolutional layers (config. 2) rather than transpose convolutional layers in the decoder. As no major improvements in terms of MSE is observed when switching from convolutional (config. 2) to transpose convolutional layers in the decoder, convolutional layers are used for the decoder.

Table 2. Performance of the baseline (config. 2) and the baseline with Convolutional Transpose layers (config. 5).

N	Mean-MSE	Std-MSE	Mean-Time	Std-Time
2	**2.435e-04**	**3.851e-05**	**812.293**	**1.436e+00**
4	2.587e-04	4.114e-05	746.804	3.089e+00

Finally, we discard the Convolutional Transpose layers and we change the kernel size increasing it a little bit. We build the model with Kernel size equal to 5×5 everywhere, defining the configuration number 5. Table 3 shows that this choice works well but not better than the *baseline*.

Table 3. Performance of the baseline (config. 2) and the baseline with kernel size equal to 5 (config. 5)

N	Mean-MSE	Std-MSE	Mean-Time	Std-Time
2	**2.435e-04**	**3.851e-05**	**812.293**	**1.436e+00**
5	1.055e-02	2.099e-02	1222.251	6.627e+00

Grid Search: We make a grid search varying (i) the number of filters $\in \{16, 32, 64\}$, (ii) the activation function $\in \{ReLu, Elu\}$, (iii) the number of epochs $\in \{250, 300, 400\}$ and (iv) the batch size $\in \{16, 32, 64\}$. Table 4 shows the optimal hyper-parameters found.

Table 4. Grid Search results of the Autoencoder.

Filters	Activation	Epochs	Batch size
64	ReLu	400	32

The performance of the *baseline* with the hyper-parameters found are reported in Table 5. Because we shuffle the data making the neural network independent from the order of the data, we can say that the model is not dependent on the set of input we choose to train it. This is important in our case because we will use the encoder to reduce the data from the observations too.

Table 5. Autoencoder performance with the chosen hyper-parameters.

Mean-MSE	Std-MSE	Mean-Time	Std-Time
8.509e-05	1.577e-05	1887.612	6.845e+00

4.2 Surrogate Model

The surrogate model is built implementing an LSTM on the results of the Autoencoder. All data are encoded with the Autoencoder: each sample is a vector of 7 scalar. We followed the same strategy as for the Autoecoder: we define the structure of the model and then we compute the grid search. To this purpose, we encode the train and validation sets defined in Figure 2. We split the data in small sequences based on the number of time-steps we look back and the number of steps we want to predict. In this case, we predict one step forward. We do not perform the CV but we repeated the fitting and the validation of the model 5 times. We fix the following parameters: neurons 30, activation function ReLu, number of time-steps 3, optimizer Adam, epochs 300, batch size 32. We use the MSE as loss function. LSTMs are stacked from 1 to 5 times in order to see if the model gains in accuracy, stability and efficiency: the results are shown in Table 6. The single layer LSTM is the one highlighting the best accuracy with the lowest Mean-MSE value. Indeed, the input of the LSTM consists of a 7×1 vector and adding more LSTM layer introduces overfitting bias. In addition, the standard deviation, reflecting the stability, of the single layer LSTM are about one order of magnitude lower than the other tested LSTM. The single layer LSTM is also the most efficient in term of computation cost.
We compute the grid search changing the hyper-parameters: (i) number of neurons $\in \{30, 50, 70\}$, (ii) activation function $\in \{ReLu, Elu\}$, (iii) number of steps $\in \{3, 5, 7\}$, (iv) number of epochs $\in \{200, 300, 400\}$ and (v) batch size $\in \{16, 32, 64\}$.
Table 7 shows the result of the Grid Search considering a single LSTM, while table 8 shows the performance of the LSTM with the chosen hyper-parameters.

Table 6. LSTM performance evaluation for 5 network architectures. N denotes the number of stacked LSTMs. Time is given in seconds.

N	Mean-MSE	Std-MSE	Mean-Time	Std-Time
1	**1.634e-02**	**8.553e-03**	**230.355**	**1.050e+00**
2	2.822e-02	7.244e-03	360.877	6.618e-01
3	4.619e-02	1.942e-02	494.254	2.258e+00
4	5.020e-02	1.675e-02	658.039	2.632e+00
5	4.742e-02	1.183e-02	806.001	5.921e+00

Table 7. Grid Search results of the single LSTM.

Neurons	Activation	Steps	Epochs	Batch size
30	Elu	3	400	16

Table 8. Single LSTM performance with the chosen hyper-parameters.

Mean-MSE	Std-MSE	Mean-Time	Std-Time
1.233e-02	1.398e-03	949.328	7.508e+00

4.3 Data Assimilation

In this phase, we encoded both the states and the observations. The assimilation is performed in the latent space. From the Test set of the CFD data, we select the sequences that predict the time-steps where we have measurements from sensors. We make the prediction through the LSTM and we update the prediction using the corresponding observation with the Kalman Filter. In the KF, the error covariance matrix $\hat{Q}$ is computed as $\hat{Q} = VV^T$ where V is computed as described in (12). Since both predictions of the model and observations are values of CO_2, i.e. the observations don't have to be transformed, the operator $\hat{H}$ is an identity matrix.
We studied how KF improves the accuracy of the prediction testing different values of $\hat{R}$ computed by the procedure in (12) or, fixed as $\hat{R} = 0.01\,I, 0.001\,I, 0.0001\,I$ where $I \in \mathbb{R}^{p\times p}$ denotes the identity matrix. This last assumption is usually made to give higher fidelity and trust to the observations.
The MSE of the background data in the latent space, without performing data assimilation is MSE$= 7.220e-01$. Tables 9 shows values of MSE after the assimilation in the latent space. It also reports values of execution time. As expected, we can observe an improvement in the execution time in assuming $\hat{R}$ as a diagonal matrix instead of a full matrix.

Table 9. Value of MSE of h_t^a and execution time of the Latent Assimilation for different values of the observations errors covariance matrix $\hat{R}$.

R̂	**Cov**	**0.01 I**	**0.001 I**	**0.0001 I**
MSE	3.215e-01	1.250e-02	1.787e-03	3.722e-05
Time	2.053e-03	3.541e-04	2.761e-04	2.618e-04

4.4 Physical space

After performing DA in the latent space, the results h_t^a are reported in the physical space through the Decoder which gives x_t^a. Table 10 and Table 11 show values of MSE after the assimilation in the physical space for LA and for a standard DA respectively. They also report values of execution time. The MSE in the physical space without the assimilation is MSE= $6.491e-02$. Tables 10 and 11 show that both LA and DA improve the accuracy of the forecasting. Comparing the tables we can also observe that LA performs better both in terms of execution time and accuracy with respect to a Standard DA where the assimilation works directly with big matrices becoming very slow.

Table 10. Values of MSE of of x_t^a in the Physical space for different values of the observations errors covariance matrix $\hat{R}$.

R̂	**Cov**	**0.01 I**	**0.001 I**	**0.0001 I**
MSE	3.356e-02	6.933e-04	1.211e-04	2.691e-06
Time	3.191e+00	2.899e+00	2.896e+00	2.896e+00

Table 11. Standard Assimilation in the physical space performed by a KF (see Equations (3)-(5)). Here $R \in \mathbb{R}^{n\times n}$ is defined in the physical space.

R	**Cov**	**0.01 I**	**0.001 I**	**0.0001 I**
MSE	5.179e-02	6.928e-03	6.928e-03	6.997e-03
Time	2.231e+03	2.148e+03	2.186e+03	2.159e+03

5 Conclusion and future works

In this paper we proposed a new methodology we called Latent Assimilation (LA) to efficiently and accurately perform DA. LA consists in performing the KF in the latent space obtained by an Autoencoder with non-linear encoder functions and non-linear decoder functions. In the latent space, the dynamic system is represented by a surrogate model built by an LSTM network to train a function which emulates the dynamic system in the latent space. The data from the dynamic model and the real data coming from the instruments are both processed through the Autoencoder. We apply the methodology to a real test case and we show that the LA performs better than a standard DA in both accuracy and efficiency. An implementation of LA to emulate variational DA [4] will be developed as future work. In particular, we will focus on a 4D variational (4DVar) method. 4DVar is a computational expensive method as it is developed to assimilate several observations (distributed in time) for each time step of the forecasting model. We will develop an extended version of LA able to assimilate set of distributed observations for each time step and, then, able to perform a 4DVar.

Acknowledgements

This work is supported by the EPSRC Grand Challenge grant Managing Air for Green Inner Cities (MAGIC) EP/N010221/1, the EP/T003189/1 Health assessment across biological length scales for personal pollution exposure and its mitigation (INHALE), the EP/T000414/1 PREdictive Modelling with QuantIfication of UncERtainty for MultiphasE Systems (PREMIERE) and the Leonardo Centre for Sustainable Business at Imperial College London.

References

1. Arcucci, R., Pain, C., Guo, Y.:Effective variational data assimilation in air-pollution prediction, Big Data Mining and Analytics 1(4), 297-307 (2018)
2. Arcucci, R., Mottet, L., Pain, C., Guo, Y.: Optimal reduced space for variational data assimilation, Journal of Computational Physics 379, 51-69, Elsevier (2019)
3. Arcucci, R., Zhu, J., Hu, S., Guo, Y.: Deep Data Assimilation: Integrating Deep Learning with Data Assimilation,Applied Sciences, 11, 3, 11-14, Multidisciplinary Digital Publishing Institute (2021)
4. Asch, M., Bocquet, M.,Nodet, M.: Data Assimilation: Methods, Algorithms, and Applications, Fundamentals of Algorithms, SIAM, (2016)
5. Babovic, V., Keijzer, M., Bundzel, M.: From global to local modelling: a case study in error correction of deterministic models, Proceedings of fourth international conference on hydroinformatics, (2000)
6. Babovic, V.,Caňizares, R., Jensen, H., Klinting, A.,:Neural networks as routine for error updating of numerical models, Journal of Hydraulic Engineering, 127(3), 181-193,American Society of Civil Engineers, (2001)
7. Babovic, V., Fuhrman, D.: Data assimilation of local model error forecasts in a deterministic model, International journal for numerical methods in fluids 39(10), 887-918, Wiley Online Library, (2002)
8. Becker, P.,Pandya, H., Gebhardt, G., Zhao, C., Taylor, J., Neumann, G.: Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces, arXiv preprint arXiv:1905.07357, (2019)
9. Boukabara, S., Krasnopolsky, V., Stewart, J., Maddy, E., Shahroudi, N., Hoffman, R.: Leveraging Modern Artificial Intelligence for Remote Sensing and NWP: Benefits and Challenges, Bulletin of the American Meteorological Society, (2019)

10. Campos, R., Krasnopolsky, V., Alves, J., Penny, S.: Nonlinear wave ensemble averaging in the Gulf of Mexico using neural networks, Journal of Atmospheric and Oceanic Technology 36(1), 113-127, (2019)
11. Cintra, R., Campos Velho, H.: Data assimilation by artificial neural networks for an atmospheric general circulation model, Advanced Applications for Artificial Neural Networks, 265, BoD–Books on Demand, (2018)
12. Dozat, T.: Incorporating nesterov momentum into adam, (2016)
13. Dueben, P., Bauer, P.: Challenges and design choices for global weather and climate models based on machine learning, Geoscientific Model Development 11(10), 3999-4009, Copernicus GmbH, (2018)
14. Funahashi, K., Nakamura, Y.: Approximation of dynamical systems by continuous time recurrent neural networks, Neural network 6(6), 801-806, Elsevier, (1993)
15. Gagne, D., McGovern, A., Haupt, S., Sobash, R., Williams, J., Xue, M.: Storm-based probabilistic hail forecasting with machine learning applied to convection-allowing ensembles, Weather and forecasting 32(5), 1819-1840, (2017)
16. Hannachi, A.: A primer for EOF analysis of climate data, Department of Meteorology University of Reading, 1-33, (2004)
17. Hansen, P., Nagy, J., O'leary, D.: Deblurring images: matrices, spectra, and filtering, 3, Siam, (2006)
18. Kalman, R.: A new approach to linear filtering and prediction problems, Journal of basic Engineering 82(1), 35-45, American Society of Mechanical Engineers, (1960)
19. Loh, K., Omrani, P.s., van der Linden, R.: Deep learning and data assimilation for real-time production prediction in natural gas wells, arXiv preprint arXiv:1802.05141., (2018)
20. Rasp, S., Dueben, P., Scher, S., Weyn, J., Mouatadid, S., Thuerey, N.: WeatherBench: A benchmark dataset for data-driven weather forecasting, arXiv preprint arXiv:2002.00469, (2020)
21. Schäfer A.M., Zimmermann H.G. (2006) Recurrent Neural Networks Are Universal Approximators. In: Kollias S.D., Stafylopatis A., Duch W., Oja E. (eds) Artificial Neural Networks – ICANN 2006. ICANN 2006. Lecture Notes in Computer Science, vol 4131. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11840817_66
22. Song, J., Fan, S., Lin, W., Mottet, L., Woodward, H., Davies Wykes, M., Arcucci, R., Xiao, D., Debay, J-E., ApSimon, H., others: Natural ventilation in cities: the implications of fluid mechanics, Building Research & Information 46(8), 809-828, Taylor & Francis, (2018)
23. Watter, M., Springenberg, J., Boedecker, J., Riedmiller, M.: Embed to control: A locally linear latent dynamics model for control from raw images, 2746–2754, Advances in neural information processing systems, (2015)
24. Wikle, C., Berliner, M.: A Bayesian tutorial for data assimilation, Physica D: Nonlinear Phenomena 230(1-2), 1-16, Elsevier, (2007)
25. Zhu, J., Hu, S., Arcucci, R., Xu, C., Zhu, J., Guo, Y.: Model error correction in data assimilation by integrating neural networks, Big Data Mining and Analytics 2(2), 83-91, TUP, (2019)