Machine Learning in Data Assimilation

Eviatar Bach 13 June 2025

Inverse Problems and Data Assimilation: A Machine Learning Approach

Eviatar Bach #, Ricardo Baptista #, Daniel Sanz-Alonso b, Andrew Stuart #

https://www.arxiv.org/abs/2410.10523

1	Inv	erse Problems	11
1	Bay 1.1 1.2 1.3 1.4	Jesian Inversion Bayesian Inversion MAP Estimation and Optimization Weil-Posediness of Bayesian Inverse Problems Model Error Model Error Interface 1.4.1 Representing Error in Data Space 1.4.2 Representing Error in Data Space 1.4.3 Parametering Error in Parameter Space 1.4.4 Representing Error in Parameter Space 1.4.3 Parameterizing the Forward Model Bibliography	13 13 15 17 19 19 20 21 22
2	Var	iational Inference	22
1	2.1	Variational Formulation of Bawar Theorem	23
	2.2	Variational Information of Dayes Theorem	- 95
	2.2	2.2.1 Moon-Field Family	- 25
		2.2.2 Gaussian Distributions	20
		2.2.2 Gaussian Distributions	20
		2.2.4 Exidence Lower Bound	- 29
	2.3	Bibliography	30
3	For	ward Surrogate Modelling	33
	3.1	Accelerating Bayesian Inversion	33
	3.2	Posterior Approximation Theorem	34
	3.3	Accelerating MAP Estimation	36
	3.4	Bibliography	37
4	Lea	arning Prior and Regularizers	39
	4.1	Learning the Prior	39
	4.2	Representing the Prior via a Pushforward	40
	4.3	Perturbations to the Prior	41
		4.3.1 Smooth Approximation of the Prior	41
		4.3.2 Empirical Approximation of the Prior	42

	4.4	Learning Regularizers for MAP Estimation	44
	4.5	Learning Regularizers for Posterior Approximation	45
	4.6	Bibliography	47
5	Tra	nsporting to the Posterior	49
	5.1	Learning the Prior to Posterior Map	49
	5.2	Connection to Variational Inference	51
	5.3	Learning Other Posterior Maps	52
	5.4	Bibliography	52
6	Lea	arning Dependence on Data	55
	6.1	Likelihood-Based Inference	55
	6.2	Likelihood-Free Inference	57
		6.2.1 Consequences of Block-Triangular Pushforward	58
		6.2.2 Learning Block-Triangular Pushforward Maps	62
	6.3	Learning Likelihood Models	64
	6.4	Amortized MAP Estimation	66
	6.5	Bibliography	67
п	Da	ata Assimilation	69

Filte	ering and Smoothing Problems	71
7.1	Formulation of the Filtering Problem	73
7.2	Formulation of the Smoothing Problem	74
7.3	Kalman Filter	75
7.4	3DVar	76
7.5	Extended Kalman Filter (ExKF)	77
7.6	Ensemble Kalman Filter (EnKF)	78
	7.6.1 The EnKF Algorithm	78
	7.6.2 Inflation	79
	7.6.3 Localization	80
7.7	Bootstrap Particle Filter	81
7.8	Optimal Particle Filters	82
7.9	4DVar	83
7.10	Reanalysis	84
7.11	Model Error	85
	7.11.1 Model Error: Smoothing	85
	7.11.2 Model Error: Filtering	86
	7.11.3 Filtering with Small Model Error	86
7.12	Bibliography	88

8	Lea	rning Forecast and Observation Model	91
	8.1	The Setting	92
	8.2	Expectation Maximization	93
		8.2.1 Learning Observation and Model Error Covariances	95
		8.2.2 Monte Carlo EM	9€
	8.3	Auto-Differentiable Kalman Filters	97
	8.4	Correcting Model Error Using Analysis Increments	99
	8.5	Discussion and Bibliography	00
9	Lea	rning Parameterized Filters and Smoothers 1	03
	9.1	Variational Formulations of Smoothing and Filtering	03
		9.1.1 Variational Formulation of Smoothing	04
		9.1.2 Variational Formulation of Filtering	0-
	9.2	Smoothing: State Estimation	07
	9.3	Smoothing: Probabilistic Estimation	08
		9.3.1 Smoothing: Gaussian Approximate Probabilistic Estimation 1	08
		9.3.2 Smoothing: Amortized Gaussian Approximate Probabilistic Esti-	
		mation	05
	9.4	Filtering: State Estimation	09
		9.4.1 3DVar	09
		9.4.2 EnKF Gain	12
		9.4.3 EnKF Localization and Inflation	1-
		9.4.4 Optimal Particle Filter	14
		9.4.5 Spread–Error Relationship	13
		9.4.6 Learning State Estimators in the Presence of Model Error 1	16
9.5 Filtering: Probabilistic Estimation		Filtering: Probabilistic Estimation	17
		9.5.1 Learning Filters Using Variational Inference	18
		9.5.2 Learning Filters Using Strictly Proper Scoring Rules 1	19
		9.5.3 Bootstrap Particle Filter	20
		9.5.4 Optimal Particle Filter	21
	9.6	Bibliography	22
10	Lea	rning the Filter or Smoother Using Transport 1	23
	10.1	Learning the Forecast to Analysis Map	23
	10.2	Learning Dependence of the Map on Data	27
		10.2.1 Minimizing the Energy Distance	28
		10.2.2 Composed Maps Learned with Maximum Likelihood 1	28
	10.3	Optimal Particle Filter Transport	3(
		10.3.1 Learning with Likelihood Weights	3
		10.3.2 Learning the Data Dependence	32
	10.4	Bibliography	32

11	Dat	a Assimilation Using Learned Forecasting Models	133
	11.1	Smoothing and Filtering Using Surrogate Models	133
		11.1.1 Smoothing Problem	133
		11.1.2 3DVar	135
	11.2	Multifidelity Ensemble State Estimation	136
		11.2.1 Multi-Model (Ensemble) Kalman Filters	136
		11.2.2 Multifidelity Monte Carlo	138
		11.2.3 Model Selection and Sample Allocation	141
	11.3	Multifidelity Covariance Estimation	141
	11.4	Bibliography	143

III Learning Frameworks	
-------------------------	--

 - 64	

12	Met	rics, Divergences and Scoring Rules	147
	12.1	Metrics	147
		12.1.1 Metrics on the Space of Probability Measures	147
		12.1.2 Total Variation and Hellinger Metrics	148
		12.1.3 Transportation Metrics	151
		12.1.4 Integral Probability Metrics	153
		12.1.5 Maximum Mean Discrepancy and Energy Distance	154
		12.1.6 Metrics on the Space of Random Probability Measures	156
	12.2	Divergences	157
		12.2.1 f-Divergences	157
		12.2.2 Relationships between f-Divergences and Metrics	159
		12.2.3 Invariance of f-Divergences under Invertible Tranformations	161
	12.3	Scoring Rules	162
		12.3.1 Energy Score	163
		12.3.2 Continuous Ranked Probability Score	164
		12.3.3 Quantile Score	166
		12.3.4 Logarithmic Score	168
		12.3.5 Dawid–Sebastiani Score	169
		12.3.6 Noise in the Verification	169
		12.3.7 Distance-Like Deterministic Scoring Rule	170
	12.4	Bibliography	170
13	Uns	supervised Learning and Generative Modeling	173
	13.1	Density Estimation	174
	13.2	Transport Methods	175
	13.3	Normalizing Flows	177
		13.3.1 Structure in the Optimization Problem for θ	177
		13.3.2 Neural ODEs	178
	13.4	Score-Based Approaches	180

	13.6 13.7 13.8	Variational Autoencoders Generative Adversarial Networks Bibliography	185 186 187		
14	Sup 14.1	provised Learning Neural Networks	189 190		
	14.2 14.3	Random Features	191 193 193		
	$14.4 \\ 14.5$	14.3.2 Regression	194 196 197		
15	Tim	e Series Forecasting	199		
	15.1	Analog Forecasting	198 200 200		
	15.3	15.2.2 Kernel Analog Forecasting	201 202 202		
	15.4	15.3.2 Memory and Prediction 15.3.3 Memory and Reservoir Computing Non-Gaussian Auto-Regressive Models	202 203 204		
	15.5	Bibliography	205		
10	16.1	Gradient Descent	207 207 207 209		
	16.2	Automatic Differentiation	210 211 212		
	16.3 16.4	Expectation-Maximization . Newton and Gauss-Newton . 16.4.1 Newton's Method	213 215 215		
	$ \begin{array}{r} 16.5 \\ 16.6 \end{array} $	Ensemble Kalman Inversion	210 216 217		
Bil	oliogr	aphy	219		
Alj	lphabetical Index 25				

Preliminaries

Consider the dynamical system

$$x_{j+1}^{t} = \mathcal{M}(x_{j}^{t}) + \xi_{j}^{\dagger}, \qquad (1a)$$

$$x_0^{\mathrm{t}} \sim \mathcal{N}(m_0, C_0), \quad \xi_j^{\dagger} \sim \mathcal{N}(0, Q) \text{ i.i.d.},$$
 (1b)

Consider the dynamical system

$$x_{j+1}^{t} = \mathcal{M}(x_{j}^{t}) + \xi_{j}^{\dagger}, \qquad (1a)$$

$$x_0^{\mathrm{t}} \sim \mathcal{N}(m_0, C_0), \quad \xi_j^{\dagger} \sim \mathcal{N}(0, Q) \text{ i.i.d.},$$
 (1b)

The observations are given by

$$y_{j+1}^{\dagger} = h(x_{j+1}^{t}) + \eta_{j+1}^{\dagger},$$
 (2a)
 $\eta_{i}^{\dagger} \sim \mathcal{N}(0, R)$ i.i.d. (2b)

Consider the dynamical system

$$x_{j+1}^{t} = \mathcal{M}(x_{j}^{t}) + \xi_{j}^{\dagger}, \qquad (1a)$$

$$x_0^{\mathrm{t}} \sim \mathcal{N}(m_0, C_0), \quad \xi_j^{\dagger} \sim \mathcal{N}(0, Q) \text{ i.i.d.},$$
 (1b)

The observations are given by

$$y_{j+1}^{\dagger} = h(x_{j+1}^{t}) + \eta_{j+1}^{\dagger},$$
 (2a)
 $\eta_{i}^{\dagger} \sim \mathcal{N}(0, R)$ i.i.d. (2b)

We define, for a fixed integer J,

$$X^{t} = \{x_{0}^{t}, \dots, x_{j}^{t}\}, Y^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}, Y_{j}^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}.$$

We define, for a fixed integer J,

$$X^{t} = \{x_{0}^{t}, \dots, x_{j}^{t}\}, Y^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}, Y_{j}^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}.$$

Three distinct tasks in data assimilation (DA):

 The problem of *smoothing* is to obtain the conditional distribution P(X^t|Y[†]) (e.g., ensemble Kalman smoother).

We define, for a fixed integer J,

$$X^{t} = \{x_{0}^{t}, \dots, x_{j}^{t}\}, Y^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}, Y_{j}^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}.$$

Three distinct tasks in data assimilation (DA):

- The problem of *smoothing* is to obtain the conditional distribution P(X^t|Y[†]) (e.g., ensemble Kalman smoother).
- The problem of *filtering* is to obtain $\mathbb{P}(x_j^{\dagger}|Y_j^{\dagger})$ for any $j \leq J$ (e.g., ensemble Kalman filter).

We define, for a fixed integer J,

$$X^{t} = \{x_{0}^{t}, \dots, x_{j}^{t}\}, Y^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}, Y_{j}^{\dagger} = \{y_{1}^{\dagger}, \dots, y_{j}^{\dagger}\}.$$

Three distinct tasks in data assimilation (DA):

- The problem of *smoothing* is to obtain the conditional distribution P(X^t|Y[†]) (e.g., ensemble Kalman smoother).
- The problem of *filtering* is to obtain $\mathbb{P}(x_j^{\dagger}|Y_j^{\dagger})$ for any $j \leq J$ (e.g., ensemble Kalman filter).
- The problem of state estimation is to obtain a x_j ≈ x^t_j in some norm (e.g., 3DVar).

How to incorporate machine learning (ML) into DA?

• Learning forecast model, learning corrections

How to incorporate machine learning (ML) into DA?

- Learning forecast model, learning corrections
- Using learned forecast model

Other topics (won't discuss):

• Learning filters and state estimators

- Learning filters and state estimators
- End-to-end forecasting and DA using ML, e.g., score-based DA (Rozet and Louppe 2023), DA networks (Boudier et al. 2023)

- Learning filters and state estimators
- End-to-end forecasting and DA using ML, e.g., score-based DA (Rozet and Louppe 2023), DA networks (Boudier et al. 2023)
- Learning smoothers

- Learning filters and state estimators
- End-to-end forecasting and DA using ML, e.g., score-based DA (Rozet and Louppe 2023), DA networks (Boudier et al. 2023)
- Learning smoothers
- Learning observation operators and observation error covariances (Waller, Dance, and Nichols 2016)

- Learning filters and state estimators
- End-to-end forecasting and DA using ML, e.g., score-based DA (Rozet and Louppe 2023), DA networks (Boudier et al. 2023)
- Learning smoothers
- Learning observation operators and observation error covariances (Waller, Dance, and Nichols 2016)
- Transport-based (Spantini, Baptista, and Marzouk 2022) and reinforcement learning-based (Hammoud et al. 2024) approaches to DA

- Learning filters and state estimators
- End-to-end forecasting and DA using ML, e.g., score-based DA (Rozet and Louppe 2023), DA networks (Boudier et al. 2023)
- Learning smoothers
- Learning observation operators and observation error covariances (Waller, Dance, and Nichols 2016)
- Transport-based (Spantini, Baptista, and Marzouk 2022) and reinforcement learning-based (Hammoud et al. 2024) approaches to DA
- Practical considerations

Gradients are often difficult to obtain in closed form for complex cost functions.

Gradients are often difficult to obtain in closed form for complex cost functions.

Finite difference approximations are often inaccurate and expensive for high-dimensional problems.

Gradients are often difficult to obtain in closed form for complex cost functions.

Finite difference approximations are often inaccurate and expensive for high-dimensional problems.

Automatic differentiation (autodiff) involves repeated application of chain rule on elementary operations that enables computing derivatives accurately to working precision. Gradients are often difficult to obtain in closed form for complex cost functions.

Finite difference approximations are often inaccurate and expensive for high-dimensional problems.

Automatic differentiation (autodiff) involves repeated application of chain rule on elementary operations that enables computing derivatives accurately to working precision.

This can allow differentiation through model states (adjoint), model parameters, and DA algorithms, and enable use of gradient-based optimization.

Suppose y = f(x), where $y \in \mathbb{R}^{d_n}$ and $x \in \mathbb{R}^{d_0}$, and the derivative of f is not readily available in closed-form.

Suppose y = f(x), where $y \in \mathbb{R}^{d_n}$ and $x \in \mathbb{R}^{d_0}$, and the derivative of f is not readily available in closed-form.

We assume the implementation of f in computer code is made up of n elementary operations $f_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ (for instance, addition, multiplication, logarithms, etc.), such that

$$f(x) = f_n \circ f_{n-1} \circ \cdots \circ f_1(x), \tag{3}$$

where the Jacobians for these elementary operations, $Df_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i \times d_{i-1}}$, are available in closed form.

Suppose y = f(x), where $y \in \mathbb{R}^{d_n}$ and $x \in \mathbb{R}^{d_0}$, and the derivative of f is not readily available in closed-form.

We assume the implementation of f in computer code is made up of n elementary operations $f_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ (for instance, addition, multiplication, logarithms, etc.), such that

$$f(x) = f_n \circ f_{n-1} \circ \cdots \circ f_1(x), \tag{3}$$

where the Jacobians for these elementary operations, $Df_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i \times d_{i-1}}$, are available in closed form.

Writing the partial evaluations up to $i \leq n$ as

$$g_i = (f_i \circ \cdots \circ f_1)(x),$$

by the chain rule we have that

$$D_{x}f(x) = (D_{g_{n-1}}f_{n}(g_{n-1}))\cdots(D_{g_{1}}f_{2}(g_{1}))(D_{x}f_{1}(x)).$$
(4)

This suggests the following algorithm (*forward mode* automatic differentiation):

1: Input: The functions $\{f_i(\cdot)\}_{i=1}^n$, their corresponding Jacobians $\{Df_i(\cdot)\}_{i=1}^n$, and the function input *x*.

2: Set
$$g_1 = f_1(x)$$
 and $J_1 = Df_1(x)$.

- 3: For i = 2, ..., n: set $g_i = f_i(g_{i-1})$ and $J_i = (Df_i(g_{i-1}))J_{i-1}$.
- 4: **Output**: The function output $y = f(x) = g_n$ and the derivative $Df(x) = J_n$.

This suggests the following algorithm (*forward mode* automatic differentiation):

1: Input: The functions $\{f_i(\cdot)\}_{i=1}^n$, their corresponding Jacobians $\{Df_i(\cdot)\}_{i=1}^n$, and the function input *x*.

2: Set
$$g_1 = f_1(x)$$
 and $J_1 = Df_1(x)$.

- 3: For i = 2, ..., n: set $g_i = f_i(g_{i-1})$ and $J_i = (Df_i(g_{i-1}))J_{i-1}$.
- 4: **Output**: The function output $y = f(x) = g_n$ and the derivative $Df(x) = J_n$.

Reverse mode autodiff requires a backwards pass to compute the derivative, and is more efficient when $d_0 \gg d_n$ (many inputs), whereas forward mode is more efficient when $d_n \gg d_0$ (many outputs).

The dynamical model has parameters ϑ :

$$x_{j+1}^{t} = \mathcal{M}_{\vartheta}(x_{j}^{t}) + \xi_{j}^{\dagger}, \qquad (5a)$$

$$x_0^{t} \sim \mathcal{N}(m_0, C_0), \ \xi_j^{\dagger} \sim \mathcal{N}(0, Q(\vartheta)) \text{ i.i.d.}$$
 (5b)

The dynamical model has parameters ϑ :

$$x_{j+1}^{t} = \mathcal{M}_{\vartheta}(x_{j}^{t}) + \xi_{j}^{\dagger},$$
 (5a)

$$x_0^{t} \sim \mathcal{N}(m_0, C_0), \ \xi_j^{\dagger} \sim \mathcal{N}(0, Q(\vartheta)) \text{ i.i.d.}$$
 (5b)

Example 1 (parameterized dynamics): $\mathcal{M} = \mathcal{M}_{\vartheta^{\dagger}}$ is parameterized, but the true parameter ϑ^{\dagger} is unknown and needs to be estimated.

The dynamical model has parameters ϑ :

$$x_{j+1}^{t} = \mathcal{M}_{\vartheta}(x_{j}^{t}) + \xi_{j}^{\dagger}, \qquad (5a)$$

$$x_0^{t} \sim \mathcal{N}(m_0, C_0), \ \xi_j^{\dagger} \sim \mathcal{N}(0, Q(\vartheta)) \text{ i.i.d.}$$
 (5b)

Example 1 (parameterized dynamics): $\mathcal{M} = \mathcal{M}_{\vartheta^{\dagger}}$ is parameterized, but the true parameter ϑ^{\dagger} is unknown and needs to be estimated.

Example 2 (fully unknown dynamics): \mathcal{M} is fully unknown and ϑ can represent, for example, the parameters of a neural network or Gaussian process surrogate model \mathcal{M}_{ϑ} for \mathcal{M} .

The dynamical model has parameters ϑ :

$$x_{j+1}^{t} = \mathcal{M}_{\vartheta}(x_{j}^{t}) + \xi_{j}^{\dagger},$$
 (5a)

$$x_0^{\mathrm{t}} \sim \mathcal{N}(m_0, C_0), \ \xi_j^{\dagger} \sim \mathcal{N}(0, Q(\vartheta)) \text{ i.i.d.}$$
 (5b)

Example 1 (parameterized dynamics): $\mathcal{M} = \mathcal{M}_{\vartheta^{\dagger}}$ is parameterized, but the true parameter ϑ^{\dagger} is unknown and needs to be estimated.

Example 2 (fully unknown dynamics): \mathcal{M} is fully unknown and ϑ can represent, for example, the parameters of a neural network or Gaussian process surrogate model \mathcal{M}_{ϑ} for \mathcal{M} .

Example 3 (model correction): \mathcal{M} is unknown, but we have access to an inaccurate model $\mathcal{M}^{approx} \approx \mathcal{M}$. The goal is to learn ϑ so that $\mathcal{M}_{\vartheta} = \mathcal{M}^{approx} + \mathcal{M}_{\vartheta}^{correction}$ approximates \mathcal{M} accurately.
We consider learning the parameters from partial, noisy observations.

We consider learning the parameters from partial, noisy observations.

If we have the true (or good enough) model and want to learn an ML surrogate, we can just use supervised learning. We consider learning the parameters from partial, noisy observations.

If we have the true (or good enough) model and want to learn an ML surrogate, we can just use supervised learning.

We would like to learn ϑ by maximizing the likelihood

$$\mathbb{P}(Y^{\dagger}|\vartheta) = \int \mathbb{P}(Y^{\dagger}, X|\vartheta) \, dX.$$
(6)

Here the states X are unobserved latent variables.

Evaluating the preceding integral is intractable. Instead, we use *expectation–maximization* (EM) algorithm for maximum likelihood estimation.

Evaluating the preceding integral is intractable. Instead, we use *expectation–maximization* (EM) algorithm for maximum likelihood estimation.

EM consists of two steps:

1. *Expectation:* Take the expectation of the log-likelihood of the state and observations given the parameters ϑ_{ℓ} :

$$J_{\ell}(\vartheta) = \mathbb{E}^{X \sim \mathbb{P}(X|Y^{\dagger}, \vartheta_{\ell})}[\log \mathbb{P}(X, Y^{\dagger}|\vartheta)]$$
(7)

Evaluating the preceding integral is intractable. Instead, we use *expectation–maximization* (EM) algorithm for maximum likelihood estimation.

EM consists of two steps:

1. *Expectation:* Take the expectation of the log-likelihood of the state and observations given the parameters ϑ_{ℓ} :

$$J_{\ell}(\vartheta) = \mathbb{E}^{X \sim \mathbb{P}(X|Y^{\dagger},\vartheta_{\ell})}[\log \mathbb{P}(X,Y^{\dagger}|\vartheta)]$$
(7)

2. Maximization:

$$\vartheta_{\ell+1} = \arg\max_{\vartheta} J_{\ell}(\vartheta)$$
 (8)

Evaluating the preceding integral is intractable. Instead, we use *expectation–maximization* (EM) algorithm for maximum likelihood estimation.

EM consists of two steps:

1. *Expectation:* Take the expectation of the log-likelihood of the state and observations given the parameters ϑ_{ℓ} :

$$J_{\ell}(\vartheta) = \mathbb{E}^{X \sim \mathbb{P}(X|Y^{\dagger},\vartheta_{\ell})}[\log \mathbb{P}(X,Y^{\dagger}|\vartheta)]$$
(7)

2. Maximization:

$$\vartheta_{\ell+1} = \arg\max_{\vartheta} \mathsf{J}_{\ell}(\vartheta) \tag{8}$$

This algorithm has the property that

$$\mathbb{P}(Y^{\dagger}|\vartheta_{\ell+1}) \ge \mathbb{P}(Y^{\dagger}|\vartheta_{\ell}).$$
⁽⁹⁾

Suppose we have a reanalysis

 $\{x_{j}^{a}\}_{j=1}^{J}$.

Suppose we have a reanalysis

 $\{x_j^a\}_{j=1}^J.$

We can train an ML model to predict

$$x_{j+1}^{a} = f(x_{j}^{a}; \theta),$$

and then use this autoregressively to predict the future. That is,

$$x_{j+2}^{a} = f(f(x_{j}^{a}; \theta); \theta),$$

etc.

Why use reanalysis rather than just observations?

• The reanalysis functions as a physics-based interpolator, fills in information where observations missing.

Why use reanalysis rather than just observations?

- The reanalysis functions as a physics-based interpolator, fills in information where observations missing.
- Lower error compared to observations alone.

Why use reanalysis rather than just observations?

- The reanalysis functions as a physics-based interpolator, fills in information where observations missing.
- Lower error compared to observations alone.
- Provided gridded in space and at regular time intervals. This is simpler for ML to deal with.

Why use reanalysis rather than just observations?

- The reanalysis functions as a physics-based interpolator, fills in information where observations missing.
- Lower error compared to observations alone.
- Provided gridded in space and at regular time intervals. This is simpler for ML to deal with.

Why use reanalysis rather than just observations?

- The reanalysis functions as a physics-based interpolator, fills in information where observations missing.
- Lower error compared to observations alone.
- Provided gridded in space and at regular time intervals. This is simpler for ML to deal with.

Why use reanalysis rather than just model forecasts?

Training on model forecasts can only be as good as the physics-based model. May be faster though.

Why use reanalysis rather than just observations?

- The reanalysis functions as a physics-based interpolator, fills in information where observations missing.
- Lower error compared to observations alone.
- Provided gridded in space and at regular time intervals. This is simpler for ML to deal with.

Why use reanalysis rather than just model forecasts?

Training on model forecasts can only be as good as the physics-based model. May be faster though.

Downsides:

The reanalysis still includes some error due to the model used to generate it, so the ML model is learning this.



Figure 1: By Stephan Rasp

Kalman-based DA algorithms can be written as

$$x^{a} = x^{f} + K(y - h(x^{f})).$$

Kalman-based DA algorithms can be written as

 $x^{a} = x^{f} + K(y - h(x^{f})).$

Observations are usually assumed to be unbiased $(\mathbb{E}[y - h(x^t)] = 0)$. Thus, the bias in the analysis is usually smaller than that of the forecast.

Kalman-based DA algorithms can be written as

 $x^{a} = x^{f} + K(y - h(x^{f})).$

Observations are usually assumed to be unbiased $(\mathbb{E}[y - h(x^t)] = 0)$. Thus, the bias in the analysis is usually smaller than that of the forecast.

Thus, the *analysis increment*, $x^a - x^f$, tends to correct for bias.

Kalman-based DA algorithms can be written as

 $x^{a} = x^{f} + K(y - h(x^{f})).$

Observations are usually assumed to be unbiased $(\mathbb{E}[y - h(x^t)] = 0)$. Thus, the bias in the analysis is usually smaller than that of the forecast.

Thus, the *analysis increment*, $x^{a} - x^{f}$, tends to correct for bias.

Given a history of analysis increments,

$$\{x_j^{\mathrm{a}} - x_j^{\mathrm{f}}\}_{j=1}^J,$$

can we extract systematic patterns?

Kalman-based DA algorithms can be written as

$$x^{\mathrm{a}} = x^{\mathrm{f}} + K(y - h(x^{\mathrm{f}})).$$

Observations are usually assumed to be unbiased $(\mathbb{E}[y - h(x^t)] = 0)$. Thus, the bias in the analysis is usually smaller than that of the forecast.

Thus, the *analysis increment*, $x^{a} - x^{f}$, tends to correct for bias.

Given a history of analysis increments,

$$\{x_j^{\mathrm{a}} - x_j^{\mathrm{f}}\}_{j=1}^J,$$

can we extract systematic patterns?

If so, we would like to be able to predict this correction before gathering observations.

We can train an ML model to predict the analysis increment, by regression on the pairs

$$\{(x_j^{f}, x_j^{a} - x_j^{f})\}_{j=1}^{J}$$

We can train an ML model to predict the analysis increment, by regression on the pairs

$$\{(x_j^{f}, x_j^{a} - x_j^{f})\}_{j=1}^{J}$$

Call the predicted analysis increment $\widetilde{\text{AI}}.$ Then, when assimilating, we update the background,

$$x_j^{\mathrm{a}} = (x_j^{\mathrm{f}} + \widetilde{\mathrm{AI}}_j) + K_j(y_j - h(x_j^{\mathrm{f}})).$$

Methods:

• General: expectation maximization Brajard et al. 2020; Bocquet et al. 2020

- General: expectation maximization Brajard et al. 2020; Bocquet et al. 2020
- General: auto-differentiable EnKF (Chen, Sanz-Alonso, and Willett 2021)

- General: expectation maximization Brajard et al. 2020; Bocquet et al. 2020
- General: auto-differentiable EnKF (Chen, Sanz-Alonso, and Willett 2021)
- General: augmented EnKF (Gottwald and Reich 2021)

- General: expectation maximization Brajard et al. 2020; Bocquet et al. 2020
- General: auto-differentiable EnKF (Chen, Sanz-Alonso, and Willett 2021)
- General: augmented EnKF (Gottwald and Reich 2021)
- Model error correction: analysis increments (Danforth, Kalnay, and Miyoshi 2007; Farchi et al. 2021)

- General: expectation maximization Brajard et al. 2020; Bocquet et al. 2020
- General: auto-differentiable EnKF (Chen, Sanz-Alonso, and Willett 2021)
- General: augmented EnKF (Gottwald and Reich 2021)
- Model error correction: analysis increments (Danforth, Kalnay, and Miyoshi 2007; Farchi et al. 2021)
- Model error covariance estimation: innovation-based methods (Tandeo et al. 2020)

Assume we have access to an approximate forecast model learned using ML, \mathcal{M}^a , which is much faster to evaluate.

Assume we have access to an approximate forecast model learned using ML, \mathcal{M}^a , which is much faster to evaluate. We can then obtain an approximate forecast using \mathcal{M}^a .

- Assume we have access to an approximate forecast model learned using ML, \mathcal{M}^a , which is much faster to evaluate.
- We can then obtain an approximate forecast using \mathcal{M}^a .
- We can state theorems that say that if the models are "close" (in a precise way), the analysis will also be close.

Suppose that we have access to both \mathcal{M} and \mathcal{M}^a . \mathcal{M} is a costly "high-fidelity" model which we will take to be the true one, while \mathcal{M}^a is a cheaper "low-fidelity" model.

Suppose that we have access to both \mathcal{M} and \mathcal{M}^a . \mathcal{M} is a costly "high-fidelity" model which we will take to be the true one, while \mathcal{M}^a is a cheaper "low-fidelity" model.

We would like to use a large ensemble of \mathcal{M}^a to augment a smaller ensemble of \mathcal{M} and improve ensemble filtering.

Suppose that we have access to both \mathcal{M} and \mathcal{M}^a . \mathcal{M} is a costly "high-fidelity" model which we will take to be the true one, while \mathcal{M}^a is a cheaper "low-fidelity" model.

We would like to use a large ensemble of \mathcal{M}^a to augment a smaller ensemble of \mathcal{M} and improve ensemble filtering.

The basic idea of multifidelity Monte Carlo methods is to use a correlated random variable, which may have a different expectation, to reduce the variance in a Monte Carlo estimate.

Suppose that we have access to both \mathcal{M} and \mathcal{M}^a . \mathcal{M} is a costly "high-fidelity" model which we will take to be the true one, while \mathcal{M}^a is a cheaper "low-fidelity" model.

We would like to use a large ensemble of \mathcal{M}^a to augment a smaller ensemble of \mathcal{M} and improve ensemble filtering.

The basic idea of multifidelity Monte Carlo methods is to use a correlated random variable, which may have a different expectation, to reduce the variance in a Monte Carlo estimate.

Alternate approach: multi-model ensemble Kalman filter (Bach and Ghil 2023)
Suppose that we have *K* low-fidelity models $\{\mathcal{M}_k\}_{k=1}^{K}$. \mathcal{M}_{hi} is the high-fidelity model, with respect to which we would like to be unbiased.

Suppose that we have *K* low-fidelity models $\{\mathcal{M}_k\}_{k=1}^{K}$. \mathcal{M}_{hi} is the high-fidelity model, with respect to which we would like to be unbiased.

Assume furthermore that the ensemble sizes for each model are arranged such that $0 < N_{hi} \le N_1 \le \ldots \le N_K$.

Suppose that we have *K* low-fidelity models $\{\mathcal{M}_k\}_{k=1}^{K}$. \mathcal{M}_{hi} is the high-fidelity model, with respect to which we would like to be unbiased.

Assume furthermore that the ensemble sizes for each model are arranged such that $0 < N_{hi} \le N_1 \le \ldots \le N_K$.

Draw random samples $\{v^{(n)}\}_{n=1}^{N_{K}}$. Then compute the following ensemble means:

$$\hat{m}_{hi} = \frac{1}{N_{hi}} \sum_{n=1}^{N_{hi}} \mathcal{M}_{hi}(v^{(n)}), \qquad \hat{m}_{k} = \frac{1}{N_{k}} \sum_{n=1}^{N_{k}} \mathcal{M}_{k}(v^{(n)}),$$
$$\hat{m}'_{k} = \frac{1}{N_{k-1}} \sum_{n=1}^{N_{k-1}} \mathcal{M}_{k}(v^{(n)}).$$

Then we consider a multifidelity estimator given by

$$\hat{m} = \hat{m}_{\rm hi} + \sum_{k=1}^{K} \alpha_k (\hat{m}_k - \hat{m}'_k), \quad \alpha_k = \frac{\rho_k Q_{\rm hi}}{Q_k},$$
 (10)

where ρ_k is the correlation coefficient between \mathcal{M}_{hi} and \mathcal{M}_k , and Q_k^2 , Q_{hi}^2 are the variances of \mathcal{M}_k and \mathcal{M}_{hi} , respectively.

Theorem

The estimator is unbiased with respect to $\hat{m}_{\rm hi},$ and has variance

$$\operatorname{Var}[\hat{m}] = \frac{Q_{\operatorname{hi}}^2}{N_{\operatorname{hi}}} - \sum_{k=1}^{K} \left(N_{k-1}^{-1} - N_k^{-1} \right) \rho_k^2 Q_{\operatorname{hi}}^2. \tag{11}$$

If $\rho_k^2 > 0$ for any k, $Var[\hat{m}] < Var[\hat{m}_{hi}]$.

We can use the multifidelity Monte Carlo estimator in the forecast step of, e.g., an ensemble Kalman filter.

- We can use the multifidelity Monte Carlo estimator in the forecast step of, e.g., an ensemble Kalman filter.
- Given a fixed computational budget, the ensemble sizes for the different models can be optimized to give the lowest variance.

Suppose now that we want to learn the analysis step, or certain unknown parts of the analysis step.

Suppose now that we want to learn the analysis step, or certain unknown parts of the analysis step.

Why? Can do this to get a trajectory closer to the truth.

Suppose now that we want to learn the analysis step, or certain unknown parts of the analysis step.

Why? Can do this to get a trajectory closer to the truth.

Can also learn an existing DA algorithm for computational efficiency using supervised learning.

Suppose now that we want to learn the analysis step, or certain unknown parts of the analysis step.

Why? Can do this to get a trajectory closer to the truth.

Can also learn an existing DA algorithm for computational efficiency using supervised learning.

Example 1: learning inflation and localization in an EnKF.

Suppose now that we want to learn the analysis step, or certain unknown parts of the analysis step.

Why? Can do this to get a trajectory closer to the truth.

Can also learn an existing DA algorithm for computational efficiency using supervised learning.

Example 1: learning inflation and localization in an EnKF.

Example 2: learning a gain matrix in $x_j = \hat{v}_j + K(y_j^{\dagger} - h(\hat{v}_j))$.

Suppose now that we want to learn the analysis step, or certain unknown parts of the analysis step.

Why? Can do this to get a trajectory closer to the truth.

Can also learn an existing DA algorithm for computational efficiency using supervised learning.

Example 1: learning inflation and localization in an EnKF.

Example 2: learning a gain matrix in $x_j = \hat{v}_j + K(y_i^{\dagger} - h(\hat{v}_j))$.

Example 3: learning an analysis step for an ensemble filter, parameterized as a neural network.

Instead of trying to match the filtering distribution, we can instead try to match the true trajectory.

Instead of trying to match the filtering distribution, we can instead try to match the true trajectory.

Assuming we have access to the model, we can simulate a trajectory x^{t} and observations Y^{\dagger} , and $x_{j}(Y_{j}^{\dagger}; \theta)$ to be some estimate of x_{i}^{t} . Then,

$$J(\theta) = \frac{1}{J} \sum_{j=1}^{J} ||x_j(Y_j^{\dagger}; \theta) - x_j^{\dagger}||^2, \qquad (12a)$$
$$\theta^* = \arg\min_{\theta} J(\theta). \qquad (12b)$$

Suppose now that we do not have access to the true model, and hence have to minimize a loss against observations Y^{\dagger} :

$$J(\theta) = \frac{1}{J} \sum_{j=1}^{J} \|h(x_j(Y_j^{\dagger}; \theta)) - y_j^{\dagger}\|^2, \qquad (13a)$$
$$\theta^* = \arg\min_{\theta} |\theta|. \qquad (13b)$$

Suppose now that we do not have access to the true model, and hence have to minimize a loss against observations Y^{\dagger} :

$$J(\theta) = \frac{1}{J} \sum_{j=1}^{J} \|h(x_j(Y_j^{\dagger}; \theta)) - y_j^{\dagger}\|^2,$$
(13a)

$$\theta^* = \arg\min_{\theta} J(\theta). \tag{13b}$$

Since x_j is a function of y_j^{\dagger} , this cost function can cause x_j to be overfit to observations.

Suppose now that we do not have access to the true model, and hence have to minimize a loss against observations Y^{\dagger} :

$$J(\theta) = \frac{1}{J} \sum_{j=1}^{J} \|h(x_j(Y_j^{\dagger}; \theta)) - y_j^{\dagger}\|^2, \qquad (13a)$$
$$\theta^* = \arg\min_{\theta} J(\theta). \qquad (13b)$$

Since x_j is a function of y_j^{\dagger} , this cost function can cause x_j to be overfit to observations.

Example: if h is surjective, one can achieve a perfect score by simply setting $x_j = h^{-1}(y_i^{\dagger})$, where h^{-1} is the right inverse of h.

Taking the expectation over the observation noise realization, $\eta_i^{\dagger} \sim \mathcal{N}(0, R)$, we obtain (Mallia-Parfitt and Bröcker 2016):

$$\mathbb{E}[\|h(x_j(\theta)) - y_j^{\dagger}\|^2] = \mathbb{E}[\|h(x_j) - h(x_j^{\dagger})\|^2] + \operatorname{tr}(R) - 2\mathbb{E}[h(x_j(\theta))^{\top}\eta_j^{\dagger}].$$

Taking the expectation over the observation noise realization, $\eta_i^{\dagger} \sim \mathcal{N}(0, R)$, we obtain (Mallia-Parfitt and Bröcker 2016):

$$\mathbb{E}[\|h(x_j(\theta)) - y_j^{\dagger}\|^2] = \mathbb{E}[\|h(x_j) - h(x_j^{\dagger})\|^2] + \operatorname{tr}(R) - 2\mathbb{E}[h(x_j(\theta))^{\top} \eta_j^{\dagger}].$$

 $\mathbb{E}[h(x_j)^{\top}\eta_j^{\dagger}]$ will generally be positive, since the observation y_j^{\dagger} was used to produce the analysis x_j . The first term on the RHS is what we would like to estimate.

Taking the expectation over the observation noise realization, $\eta_i^{\dagger} \sim \mathcal{N}(0, R)$, we obtain (Mallia-Parfitt and Bröcker 2016):

 $\mathbb{E}[\|h(x_j(\theta)) - y_j^{\dagger}\|^2] = \mathbb{E}[\|h(x_j) - h(x_j^{\dagger})\|^2] + \operatorname{tr}(R) - 2\mathbb{E}[h(x_j(\theta))^{\top} \eta_j^{\dagger}].$

 $\mathbb{E}[h(x_j)^{\top}\eta_j^{\dagger}]$ will generally be positive, since the observation y_j^{\dagger} was used to produce the analysis x_j . The first term on the RHS is what we would like to estimate.

 $\mathbb{E}[\|h(x_j(\theta)) - y_j^{\dagger}\|^2] + 2\mathbb{E}[h(x_j(\theta))^{\top}\eta_j^{\dagger}]$ is thus a better proxy for out-of-sample performance, and should be minimized instead of only $\mathbb{E}[\|h(x_j(\theta)) - y_j^{\dagger}\|^2]$.

References i

Bach, E. and M. Ghil (2023). "A Multi-Model Ensemble Kalman Filter for Data Assimilation and Forecasting". Journal of Advances in Modeling Earth Systems.

- Bocquet, M., J. Brajard, A. Carrassi, and L. Bertino (2020).
 "Bayesian Inference of Chaotic Dynamics by Merging Data Assimilation, Machine Learning and Expectation-Maximization". Foundations of Data Science.
- Boudier, P., A. Fillion, S. Gratton, S. Gürol, and S. Zhang (2023). "Data Assimilation Networks". Journal of Advances in Modeling Earth Systems.
- Brajard, J., A. Carrassi, M. Bocquet, and L. Bertino (2020). "Combining Data Assimilation and Machine Learning to Emulate a Dynamical Model from Sparse and Noisy Observations: A Case Study with the Lorenz 96 Model". Journal of Computational Science.

References ii

- Chen, Y., D. Sanz-Alonso, and R. Willett (2021). "Auto-Differentiable Ensemble Kalman Filters". arXiv: 2107.07687 [cs, stat].
- Danforth, C. M., E. Kalnay, and T. Miyoshi (2007). "Estimating and Correcting Global Weather Model Error". Monthly Weather Review.
- Farchi, A., P. Laloyaux, M. Bonavita, and M. Bocquet (2021).
 "Using Machine Learning to Correct Model Error in Data Assimilation and Forecast Applications". Quarterly Journal of the Royal Meteorological Society.
- Gottwald, G. A. and S. Reich (2021). "Supervised Learning from Noisy Observations: Combining Machine-Learning Techniques with Data Assimilation". Physica D: Nonlinear Phenomena.

References iii

📔 Hammoud, M. A. E. R., N. Raboudi, E. S. Titi, O. Knio, and I. Hoteit (2024). Data Assimilation in Chaotic Systems Using Deep Reinforcement Learning. arXiv: 2401.00916 [physics]. Pre-published.

- Mallia-Parfitt, N. and J. Bröcker (2016). "Assessing the Performance of Data Assimilation Algorithms Which Employ Linear Error Feedback". Chaos: An Interdisciplinary Journal of Nonlinear Science.
- Rozet, F. and G. Louppe (2023). "Score-Based Data Assimilation". Advances in Neural Information Processing Systems.



Spantini, A., R. Baptista, and Y. Marzouk (2022). "Coupling Techniques for Nonlinear Ensemble Filtering". SIAM Review.

References iv

Tandeo, P., P. Ailliot, M. Bocquet, A. Carrassi, T. Miyoshi, M. Pulido, and Y. Zhen (2020). "A Review of Innovation-Based Methods to Jointly Estimate Model and Observation Error Covariance Matrices in Ensemble Data Assimilation". Monthly Weather Review.

Waller, J. A., S. L. Dance, and N. K. Nichols (2016). "Theoretical Insight into Diagnosing Observation Error Correlations Using Observation-Minus-Background and Observation-Minus-Analysis Statistics". Quarterly Journal of the Royal Meteorological Society.