Online model error correction with neural networks
Towards an implementation in the ECMWF data assimilation system

Alban Farchi[†], Marcin Chrust[‡],
Marc Bocquet[†], Patrick Laloyaux[‡], and Massimo Bonavita[‡]

[†] CEREA, joint laboratory École des Ponts ParisTech and EDF R&D, Île-de-France, France
[‡] ECMWF, Shinfield Park, Reading, United Kingdom

Tuesday, 15 November 2022

ECMWF–ESA Workshop on Machine Learning for Earth Observation and Prediction

Weak-constraint 4D-Var: the forcing formulation

- The idea of *weak-constraint 4D-Var* is to relax the perfect model assumption.
- The price to pay is a huge increase in problem dimensionality.
- This can be mitigated by making additional assumption, e.g. the model error $\mathbf{w}$ is constant over the DA window:

$$\mathbf{x}_{k+1} = \boldsymbol{\mathcal{M}}_{k+1:k}\left(\mathbf{x}_k\right) + \mathbf{w} \triangleq \boldsymbol{\mathcal{M}}^{\mathsf{wc}}_{k+1:0}\left(\mathbf{w}, \mathbf{x}_0\right).$$

- The cost function can hence be written

$$\begin{aligned}\mathcal{J}^{\mathsf{wc}}\left(\mathbf{w}, \mathbf{x}_0\right) = {}& \frac{1}{2}\left\|\mathbf{x}_0 - \mathbf{x}_0^{\mathsf{b}}\right\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\left\|\mathbf{w} - \mathbf{w}^{\mathsf{b}}\right\|^2_{\mathbf{Q}^{-1}} \\ & + \frac{1}{2}\sum_{k=0}^{L}\left\|\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k \circ \boldsymbol{\mathcal{M}}^{\mathsf{wc}}_{k:0}\left(\mathbf{w}, \mathbf{x}_0\right)\right\|^2_{\mathbf{R}_k^{-1}}.\end{aligned}$$

- This is called *forcing formulation* of weak-constraint 4D-Var. This is the weak-constraint 4D-Var currently implemented in OOPS (the ECMWF data assimilation system).

Weak-constraint 4D-Var: a neural network formulation

- Now suppose that the dynamical model is *parametrised* by a set of parameters $\mathbf{p}$ constant over the window:
$$\mathbf{x}_k = \boldsymbol{\mathcal{M}}^{\mathsf{nn}}_{k:0}\left(\mathbf{p}, \mathbf{x}_0\right).$$
- Following the same approach, the cost function becomes
$$\begin{aligned}\mathcal{J}^{\mathsf{nn}}\left(\mathbf{p}, \mathbf{x}_0\right) = & \frac{1}{2}\left\|\mathbf{x}_0 - \mathbf{x}_0^{\mathsf{b}}\right\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\left\|\mathbf{p} - \mathbf{p}^{\mathsf{b}}\right\|^2_{\mathbf{P}^{-1}} \\ & + \frac{1}{2}\sum_{k=0}^{L}\left\|\mathbf{y}_k - \boldsymbol{\mathcal{H}}_k \circ \boldsymbol{\mathcal{M}}^{\mathsf{nn}}_{k:0}\left(\mathbf{p}, \mathbf{x}_0\right)\right\|^2_{\mathbf{R}_k^{-1}}.\end{aligned}$$
- This approach can be seen as a *neural network formulation* of weak-constraint 4D-Var when $\mathbf{p}$ is the set of parameters (weights and biases) of a NN.

- In order to merge the two approaches, we consider the case where the *constant model error* $\mathbf{w}$ is *estimated using a neural network*:

$$\mathcal{M}^{\mathsf{nn}}_{k+1:k}\left(\mathbf{p}, \mathbf{x}_k\right) = \mathcal{M}_{k+1:k}\left(\mathbf{x}_k\right) + \mathbf{w}, \quad \mathbf{w} = \mathcal{F}\left(\mathbf{p}, \mathbf{x}_0\right).$$

- This means that the model evolution becomes

$$\mathcal{M}^{\mathsf{nn}}_{k:0}\left(\mathbf{p}, \mathbf{x}_0\right) = \mathcal{M}^{\mathsf{wc}}_{k:0}\left(\mathcal{F}\left(\mathbf{p}, \mathbf{x}_0\right), \mathbf{x}_0\right).$$

- As a consequence, it will be possible to build this simplified method on top of the *currently implemented weak-constraint* 4D-Var, in the *incremental assimilation* framework (with inner and outer loops).

Gradient of the incremental cost function

Input: $\delta\mathbf{p}$ and $\delta\mathbf{x}_0$

$\delta\mathbf{w} \leftarrow \mathbf{F}^{\mathsf{p}}\delta\mathbf{p} + \mathbf{F}^{\mathsf{x}}\delta\mathbf{x}_0$ ▷ TL of the NN $\mathcal{F}$

$\mathbf{z}_0 \leftarrow \mathbf{R}_0^{-1}\left(\mathbf{H}_0\delta\mathbf{x}_0 - \mathbf{d}_0\right)$

for $k = 1$ **to** $L-1$ **do**

  $\delta\mathbf{x}_k \leftarrow \mathbf{M}_{k:k-1}\delta\mathbf{x}_{k-1} + \delta\mathbf{w}$ ▷ TL of the dynamical model $\boldsymbol{\mathcal{M}}_{k:k-1}$

  $\mathbf{z}_k \leftarrow \mathbf{R}_k^{-1}\left(\mathbf{H}_k\delta\mathbf{x}_k - \mathbf{d}_k\right)$

end for

$\delta\tilde{\mathbf{x}}_{L-1} \leftarrow \mathbf{0}$ ▷ AD variable for system state

$\delta\tilde{\mathbf{w}}_{L-1} \leftarrow \mathbf{0}$ ▷ AD variable for model error

for $k = L-1$ **to** 1 **do**

  $\delta\tilde{\mathbf{x}}_k \leftarrow \mathbf{H}_k^{\top}\mathbf{z}_k + \delta\tilde{\mathbf{x}}_k$

  $\delta\tilde{\mathbf{w}}_{k-1} \leftarrow \delta\tilde{\mathbf{x}}_k + \delta\tilde{\mathbf{w}}_k$

  $\delta\tilde{\mathbf{x}}_{k-1} \leftarrow \mathbf{M}_{k:k-1}^{\top}\delta\tilde{\mathbf{x}}_k$ ▷ AD of the dynamical model $\boldsymbol{\mathcal{M}}_{k:k-1}$

end for

$\delta\tilde{\mathbf{x}}_0 \leftarrow \mathbf{H}_0^{\top}\mathbf{z}_0 + \delta\tilde{\mathbf{x}}_0$

$\delta\tilde{\mathbf{x}}_0 \leftarrow \left[\mathbf{F}^{\mathsf{x}}\right]^{\top}\delta\tilde{\mathbf{x}}_0$ ▷ AD of the NN $\mathcal{F}$

$\delta\tilde{\mathbf{p}} \leftarrow \left[\mathbf{F}^{\mathsf{p}}\right]^{\top}\delta\tilde{\mathbf{w}}_0$ ▷ AD of the NN $\mathcal{F}$

$\delta\tilde{\mathbf{x}}_0 \leftarrow \mathbf{B}^{-1}\left(\mathbf{x}_0^{\mathsf{i}} - \mathbf{x}_0^{\mathsf{b}} + \delta\mathbf{x}_0\right) + \delta\tilde{\mathbf{x}}_0$

$\delta\tilde{\mathbf{p}} \leftarrow \mathbf{P}^{-1}\left(\mathbf{p}^{\mathsf{i}} - \mathbf{p}^{\mathsf{b}} + \delta\mathbf{p}\right) + \delta\tilde{\mathbf{p}}$

Output: $\nabla_{\delta\mathbf{p}}\widehat{\mathcal{J}}^{\mathsf{nn}} = \delta\tilde{\mathbf{p}}$ and $\nabla_{\delta\mathbf{x}_0}\widehat{\mathcal{J}}^{\mathsf{nn}} = \delta\tilde{\mathbf{x}}_0$

Gradient of the incremental cost function

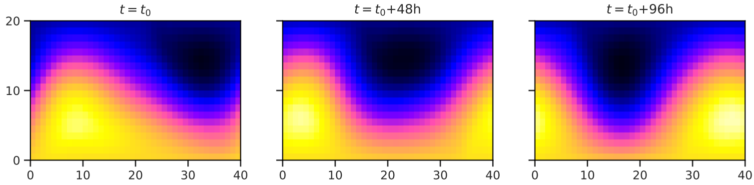
- In order to implement the simplified NN 4D-Var we can reuse most of the framework already in place for WC 4D-Var.
- A few *new bricks* need to be implemented:
  - the forward operator $\mathcal{F}$ of the NN to compute the nonlinear trajectory at the start of each outer iteration;
  - the tangent linear (TL) operators $\mathbf{F}^{\mathrm{x}}$ and $\mathbf{F}^{\mathrm{p}}$ of the NN;
  - the adjoint (AD) operators $[\mathbf{F}^{\mathrm{x}}]^{\top}$ and $[\mathbf{F}^{\mathrm{p}}]^{\top}$ of the NN.
- These operators have to be computed in the model core (where the components of the state are available), which is implemented in Fortran.

- To do so, we have implemented our own *NN library in Fortran*.

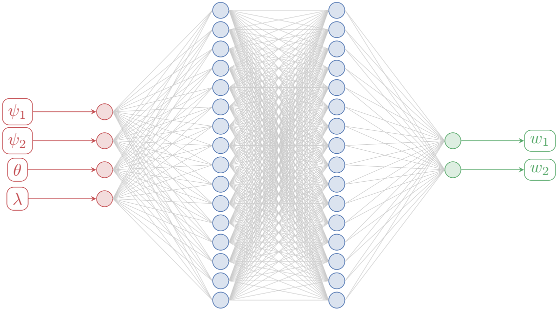
https://github.com/cerea-daml/fnn

- The FNN library has been interfaced and included in OOPS.

- Before using it in operational data assimilation, we would like to illustrate the method with a lower model.
- To do so, we use the *QG model implemented in OOPS*. This is a two-layer, two-dimensional quasi geostrophic model.
- The control vector contains all values of the stream function ψ for both levels for a total of *1600 variables*.
- Model error is introduced by using a perturbed setup, in which layer depths and the integration time steps have been modified.

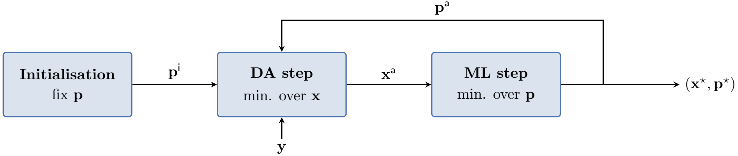


- By construction, NN 4D-Var is very similar to parameter estimation, which is challenging when the number of parameters is high.
- For this reason, it is important to use smart NN architectures to be parameter efficient.
- Taking inspiration from Bonavita & Laloyaux, 2020, we use a *vertical architecture*, with only 386 parameters.



Offline learning: training step

- We start by an *offline learning* step to provide a baseline.
- We combine data assimilation and machine learning to learn from sparse and noisy observations.
- Effectively, data assimilation is used to estimate the state from observations, and machine learning is used to estimate the NN parameters from the estimated states.



- In practice, the NN is trained to predict the *analysis increments*, a proxy for model error.

Offline learning: data assimilation skill

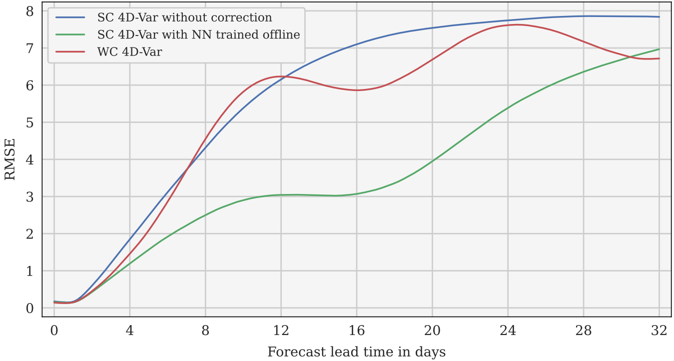
- We evaluate the accuracy of the corrected model in data assimilation experiments.
- To do so, the NN correction is rescaled (from one window to one time step) and used as a constant forcing throughout the window.
- We measure the time-averaged *first-guess* and *analysis RMSE* (vs. the truth).

Variant	constraint	Model error correction	First-guess RMSE	Analysis RMSE
SC 4D-Var	strong	—	0.35	0.16
SC 4D-Var	strong	NN trained offline	0.26	0.14
WC 4D-Var	weak	constant, online estimated	0.27	0.13

- *The NN correction is effective* and indeed reduces both the first-guess and analysis errors.

Offline learning: forecast skill

- Starting from the analysis, we run a long-range forecast.
- The NN correction is updated every day.
- We measure the *forecast RMSE* (compared to the truth) as a function of lead time.



- *The NN correction is effective* up to 10 to 15 days.

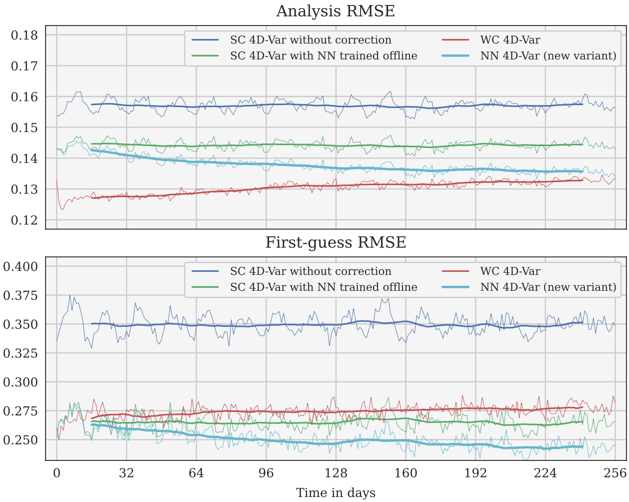
Online learning: setup

- We now train the NN online using the new 4D-Var variant.
- We start from the set of parameters obtained via offline learning (*pre-training*).
- The *background error covariance matrix* for model parameters is set to

$$\mathbf{P} = 0.02^2 \times \mathbf{I}.$$

- The coefficient 0.02 is chosen on empirical grounds. It measures how much information is transferred from one window to the next by the means of the background for model parameters $\mathbf{p}^{\mathrm{b}}$.
- We run a total of 257 windows. For each window, we measure the first-guess, the analysis, and the long-range forecast errors.

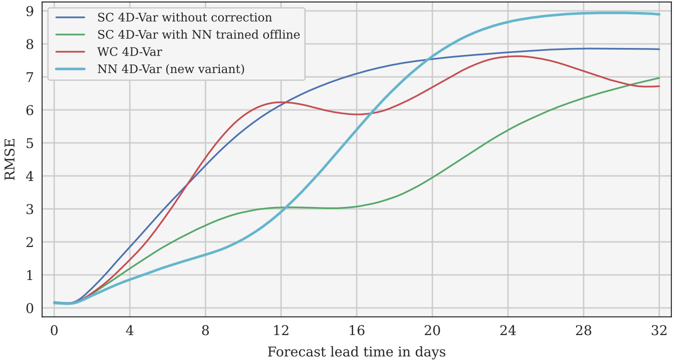
Online learning: first-guess and analysis errors



- As new observations become available, online learning *steadily improves the model*, which results in more accurate first-guess and analysis.

Online learning: forecast errors

- We compute the accuracy of the forecast at the end of the experiment.



- The online trained model *is more accurate*, up to 12 days.
- After 12 days, the forecast error increase accelerate. This is related to the limited predictive power of the NN.

Conclusions

- We have developed a *new variant* of weak-constraint 4D-Var to perform an *online, joint estimation* of the system state and NN parameters.
- The new method is built on top of the existing weak-constraint 4D-Var, in the incremental assimilation framework.
- The new method is *implemented in OOPS*, using a newly developed NN library in Fortran (FNN).
- The new method has been tested using the QG model in OOPS.
- The new method is *compatible with future applications to more realistic models*, for example with the IFS (work in progress).

https://doi.org/10.1002/essoar.10512719.1