Model error correction with data assimilation and machine learning: from theory to the ECMWF forecasting 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, 18 June 2024

19th International EnKF workshop

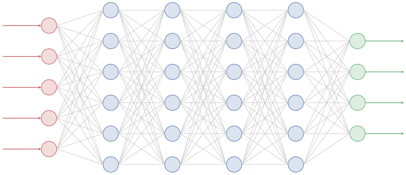
Contents

Machine learning for NWP with dense and perfect observations

- A typical (supervised) machine learning problem: given observations $\mathbf{y}_k$ of a system, derive a *surrogate model* of that system.

$$\mathcal{J}(\mathbf{p}) = \sum_{k=1}^{N_\mathrm{t}} \left\| \mathbf{y}_{k+1} - \mathcal{M}(\mathbf{p}, \mathbf{y}_k) \right\|^2.$$

- $\mathcal{M}$ depends on a *set of coefficients* $\mathbf{p}$ (*e.g.*, the weights and biases of a neural network).
- This requires dense and perfect observations of the system. In NWP, observations are usually *sparse* and *noisy*: we need data assimilation!



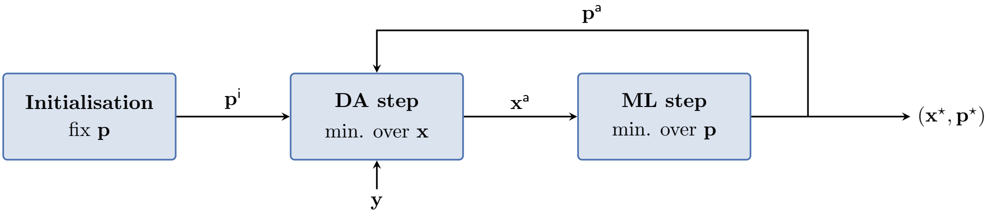
Machine learning for NWP with sparse and noisy observations

- A rigorous Bayesian formalism for this problem[1]:

$$\mathcal{J}(\mathbf{p}, \mathbf{x}_0, \ldots, \mathbf{x}_{N_\mathrm{t}}) = \frac{1}{2} \sum_{k=0}^{N_\mathrm{t}} \left\| \mathbf{y}_k - \mathcal{H}_k(\mathbf{x}_k) \right\|^2_{\mathbf{R}_k^{-1}} + \frac{1}{2} \sum_{k=0}^{N_\mathrm{t}-1} \left\| \mathbf{x}_{k+1} - \mathcal{M}(\mathbf{p}, \mathbf{x}_k) \right\|^2_{\mathbf{Q}_k^{-1}}.$$

- This resembles a typical *weak-constraint 4D-Var* cost function!

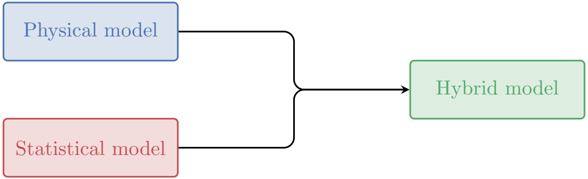
- *DA* is used to estimate the state and then *ML* is used to estimate the model.



[1] Bocquet et al. (2019, 2020), Brajard et al. (2020)

Machine learning for model error correction

- Even though NWP models are not perfect, they are already quite good!
- Instead of building a surrogate model from scratch, we use the DA-ML framework to build a *hybrid* surrogate model, with a physical part and a statistical part[2].



- In practice, the statistical part is trained to learn the *error* of the physical model.
- In general, it is easier to train a correction model than a full model: we can use *smaller NNs* and *less training data*.

[2] Farchi et al. (2021), Brajard et al. (2021)

Typical architecture of a physical model

- The model is defined by a set of ODEs or PDEs which define the *tendencies*:

$$\frac{\partial \mathbf{x}}{\partial t} = \phi(\mathbf{x}). \tag{1}$$

- A numerical scheme is used to integrate the tendencies from time t to $t + \delta t$ (*e.g.*, Runge–Kutta):

$$\mathbf{x}(t + \delta t) = \mathcal{I}\Big(\mathbf{x}(t)\Big). \tag{2}$$

- Several integration steps are composed to define the *resolvent* from one analysis (or window) to the next:

$$\mathcal{M} : \mathbf{x}_k \mapsto \mathbf{x}_{k+1} = \mathcal{I} \circ \cdots \circ \mathcal{I}(\mathbf{x}_k) \tag{3}$$

Resolvent correction

- Physical model and of NN are *independent*.
- NN must predict the analysis increments.
- Resulting hybrid model not suited for short-term predictions.
- For DA, need to assume *linear growth of errors in time* to rescale correction.

Tendency correction

- Physical model and NN are *entangled*.
- Need the adjoint of the physical model to train the NN!
- Resulting hybrid model suited for any prediction.
- Can be used as is for DA.

Illustration with the two-scale Lorenz system: setup

- True model: 2-scale Lorenz (2005-III) system with 36 slow variables and 360 fast variables.
- Physical model (to correct): 1-scale Lorenz (1996) system with 36 variables.

Sources of model error

- the fast variables are not represented;
- the integration step is 0.05 instead of 0.005;
- (the forcing constant is 8 instead of 10).

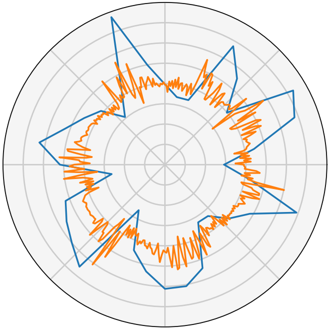
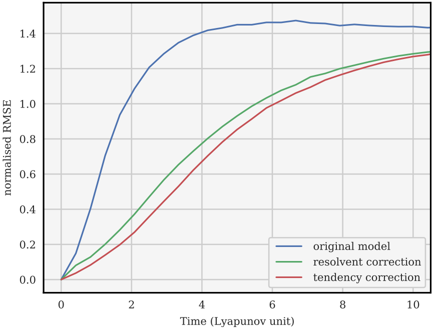


Illustration with the two-scale Lorenz system: results

- Noisy observations are assimilated using strong-constrained *4D-Var*.
- Simple *CNNs* are trained using the 4D-Var analysis dataset to correct model errors.



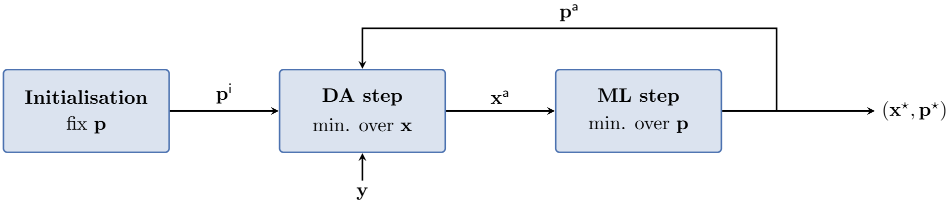
Model	Analysis RMSE
Original model	0.31
Resolvent correction	0.28
Tendency correction	0.24
True model	0.22

- The TC is *more accurate* than the RC, even with smaller NNs and less training data.
- The TC benefits from the *interaction* with the physical model.
- The RC is highly penalised (in DA) by the assumption of linear growth of errors.

Contents

Merging DA and ML for online model error correction

- So far, the model error has been learnt *offline*: the NN is trained only once the entire analysis dataset is available.



- We now investigate the possibility to make *online* learning, *i.e.* improving the NN as new observations become available.
- In practice, we propose to *merge the DA and ML steps*: we want to use the formalism of DA to estimate both the state and the NN parameters at the same time.

A neural network formulation of weak-constraint 4D-Var

- Taking inspiration from *weak-constraint 4D-Var*, we propose to use the following DA cost function:

$$\mathcal{J}(\mathbf{p},\mathbf{x}_0)=\frac{1}{2}\left\|\mathbf{x}_0-\mathbf{x}_0^{\mathsf{b}}\right\|^2_{\mathbf{B}^{-1}}+\frac{1}{2}\left\|\mathbf{p}-\mathbf{p}^{\mathsf{b}}\right\|^2_{\mathbf{P}^{-1}}+\frac{1}{2}\sum_{k=0}^{L}\left\|\mathbf{y}_k-\mathcal{H}_k\circ\mathcal{M}_{k:0}(\mathbf{p},\mathbf{x}_0)\right\|^2_{\mathbf{R}_k^{-1}}.$$

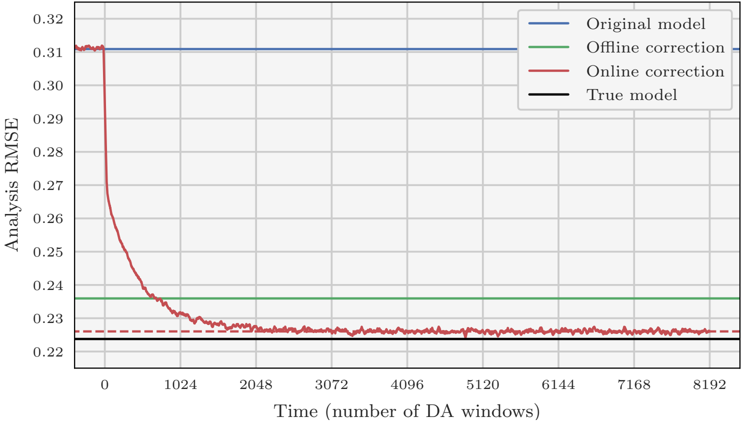
- The parameters $\mathbf{p}$ (e.g., NN weights and biases) are assumed constant over the DA window.
- Information is flowing from one window to the next using the prior $\mathbf{x}_0^{\mathsf{b}}$ and $\mathbf{p}^{\mathsf{b}}$.

- This approach is very similar to classical *parameter estimation* in DA, and it can be seen as a NN formulation of weak-constraint 4D-Var.
- This has been already done in an EnKF context[3].

[3] Bocquet et al. (2020)

Illustration with the two-scale Lorenz system

- We use the tendency correction approach, with the same simple CNN as before.



- The online correction steadily improves the model.
- At some point, the online correction *gets more accurate* than the offline correction.
- Eventually, the improvement saturates. The analysis error is similar to that obtained with the true model!

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:

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

- The DA cost function can hence be written

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

- 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).

A simplified NN 4D-Var built on top of WC 4D-Var

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

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

- This means that the model evolution can be written

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

- 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 $\mathbf{F}^{\mathsf{x}}$ and $\mathbf{F}^{\mathsf{p}}$ of the NN;
  - the adjoint (AD) operators $[\mathbf{F}^{\mathsf{x}}]^\top$ and $[\mathbf{F}^{\mathsf{p}}]^\top$ 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.

Illustration with a quasi-geostrophic model: the model

- 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.

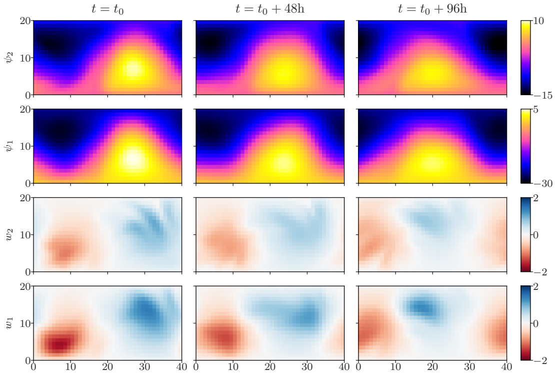
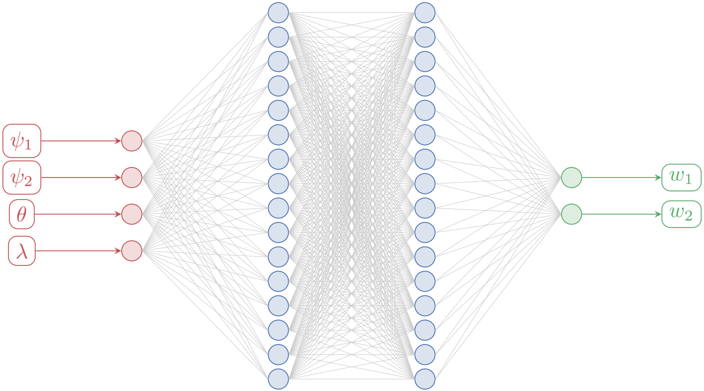
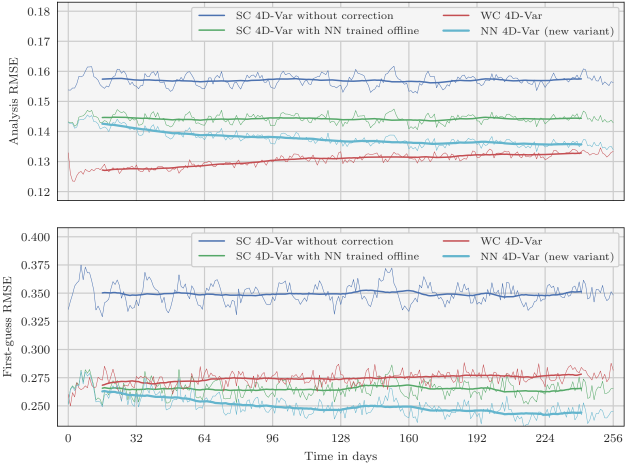


Illustration with a quasi-geostrophic model: NN architecture

- 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.



Online learning: first-guess and analysis errors



- The NN is first trained offline (*pre-training*) then online using the new 4D-Var variant.
- As new observations become available, online learning *steadily improves the model*, resulting in more accurate first-guess and analysis.

Contents

Experiments with the IFS

- We want to develop a model error correction for the operational IFS.
- Following the QG experiments, we use a two-step process:
  - offline learning to *screen potential architectures* and *pre-train the NN*
  - online learning: data assimilation and forecast experiments

- Offline experiments rely on preliminary work by Bonavita & Laloyaux (2020), using the *operational analyses* produced by ECMWF between 2017 and 2021.

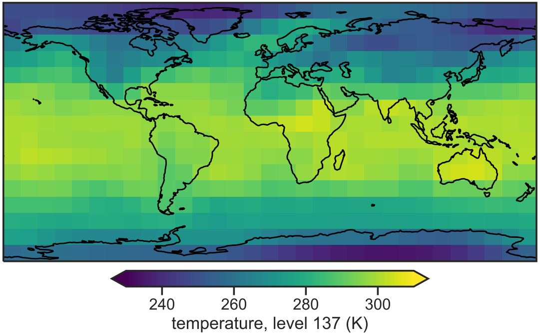
- The NN is trained to predict the analysis increments, which are available every 12 hours.

- Training / validation split:
  - training from 2017-01-01 to 2020-10-01 (IFS cycles 43R1 to 47R1);
  - validation from 2020-10-01 to 2021-10-01 (IFS cycles 47R1 to 47R2).

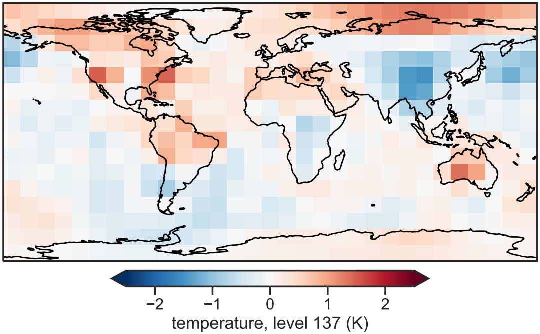
Focus on large-scale model errors

- Focus on *large-scale model errors*: we use the data at a low spectral resolution (T15), interpolated in Gaussian grid with 16×31 nodes.

Input
Analysis at t_0

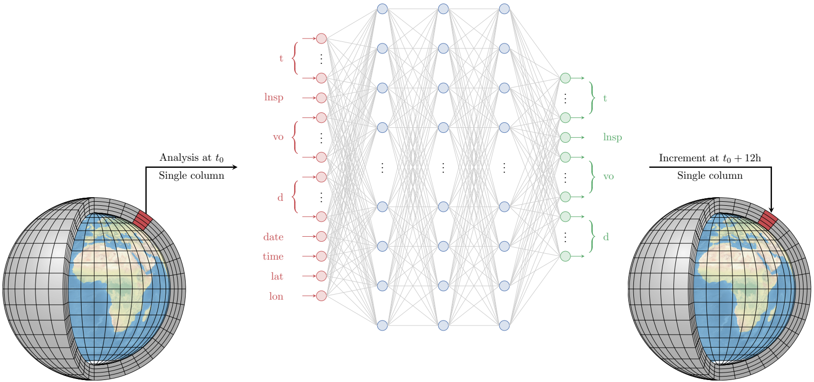


Output
Analysis increment at $t_0 + 12\mathrm{h}$



Neural network architecture

- We compute a correction for 4 variables in the same NN: temperature (t), logarithm of surface pressure (lnsp), vorticity (vo) and divergence (d).
- We keep the same *vertical architecture* as in Bonavita & Laloyaux (2020).

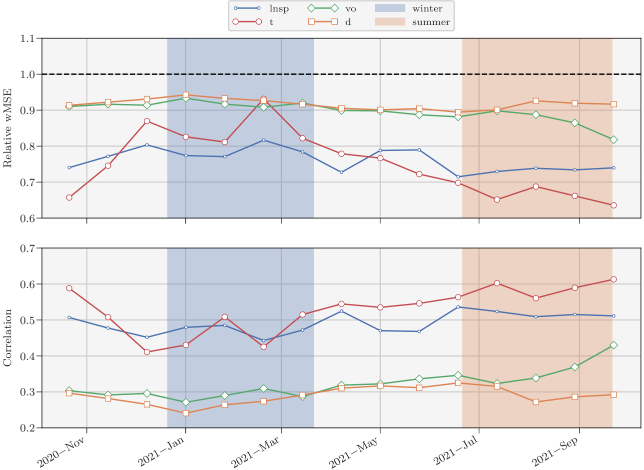


- The NN can be used with any grid.
- The number of parameters is relatively small (approx. 1M) compared to the dimension of the control vector and to the size of the training dataset (approx. 700M).
- Spatial information is partially lost.

Offline performance of the NN

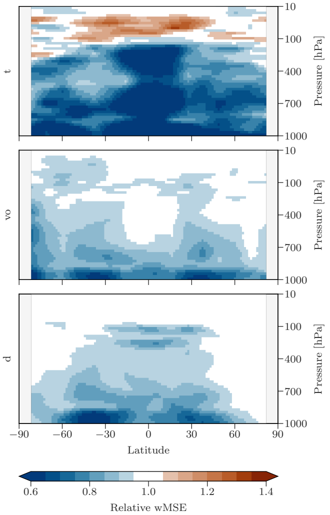
Test MSE (relative)

Model	t	lnsp	vo	d
No correction	1.000	1.000	1.000	1.000
Trained NN	0.760	0.759	0.898	0.919



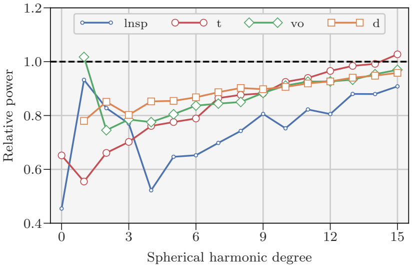
- Overall, the NN predicts approximately *15% of the analysis increments*.
- The increments for *tlnsp* are more predictable than for *vod*.
- The increments are more predictable in summer than in winter.

Offline performance of the NN



- The NN is most accurate *close to the surface*.
- The estimations deteriorate between 10 and 100 hPa, where weak constraint 4D-Var is active in the test set.

- The estimations are more accurate *at larger scales*.



Second step: data assimilation experiments

- The trained NN is inserted *into the IFS*, in a standard research configuration:
  - 12h assimilation window;
  - Latest IFS cycle 48R1;
  - Resolution of the nonlinear model: TCo399;
  - Resolution of the inner loops: TL95, TL159, TL255.

- Three-month experiment in *summer 2022* (outside the offline training and test set).

- First test series *without online learning*.

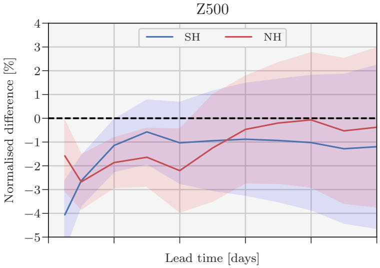
  This is equivalent to using strong-constraint 4D-Var with the corrected model.
- Second test series *with online learning*.

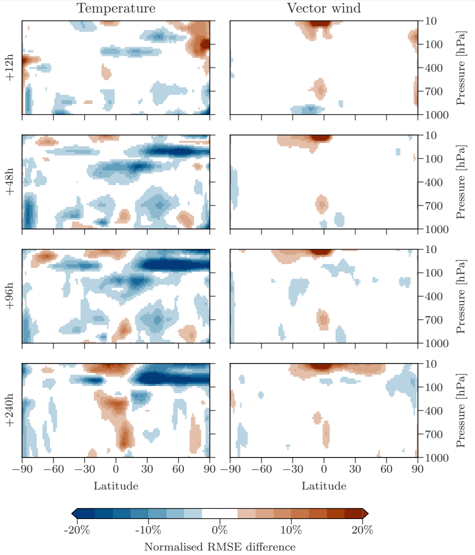
Data assimilation experiments without online training

- Comparison to the *operational analysis*.
- Baseline: standard weak-constraint 4D-Var by Laloyaux et al. (2020).

- Significantly reduced errors above 100 hPa, especially at long lead time.
- Below 100 hPa, the performance in the tropics is degraded.

- For Z500, we see a RMSE reduction of 1 % to 2 %.

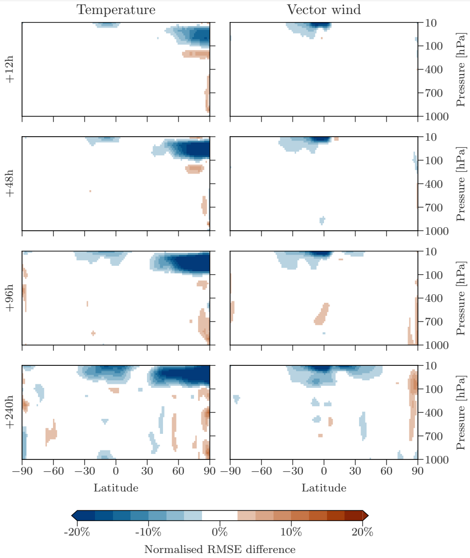




Data assimilation experiments with online training

- Comparison to the *operational analysis*.
- Baseline: experiment without online training.

- Significantly reduced the errors in the stratosphere.
- Especially in the northern hemisphere for temperature and in the tropics for vector winds.

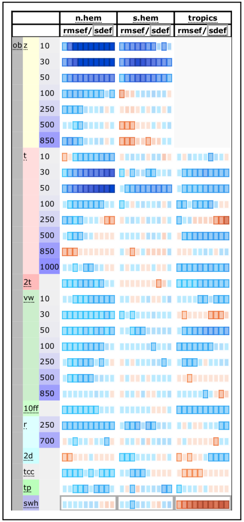


Data assimilation experiments with online training

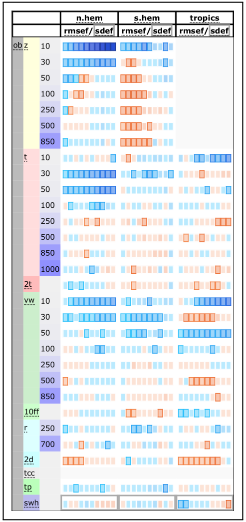
- Comparison to *independent observations*.

- Overall, the impact on forecast RMSE of all variables is positive in the *northern hemisphere* and in the *tropics*.
- Relatively modest impact in the *southern hemisphere* except in the stratosphere.
- On the downside, some score are slightly degraded, e.g. temperature at 850 hPa.

Online - Reference



Online - Offline



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 variant is built on top of the existing weak-constraint 4D-Var, in the incremental assimilation framework.
- The new variant is *implemented in OOPS*, using a newly developed NN library in Fortran (FNN).

- We are testing the method with the operational IFS.
- First results are promising.
- Upcoming challenges:
  - training at *higher resolution*;
  - develop a *time-dependent correction* within the window;
  - improve the consistency between offline and online training.

- More details can be found in our preprint:

https://doi.org/10.48550/arXiv.2403.03702