Data assimilation and machine learning

Alan Geer
European Centre for Medium-range Weather Forecasts
alan.geer@ecmwf.int

Thanks to: Massimo Bonavita, Sam Hatfield, Patricia de Rosnay, Peter Dueben, Peter Lean
An ML example: microwave land surface observation operator

Python, Keras, Tensorflow, Numpy, Matplotlib, Xarray
Datasets

AMSR2 24GHz v-pol observations

10 possible predictors for the brightness temperature

- Skin temperature
- Soil moisture
- Leaf area index
- + orography, snow depth, snow density, integrated water vapour, cloud, rain and snow water contents
Data preparation

```python
obdata = xr.open_dataset('/perm/rd/stg/odb/hkhg/ml_amrs2_chan9.nc')

x0 = np.column_stack([obdata.TSFC, obdata.SOIL_MOISTURE, obdata.SNOW_DEPTH, \n                         obdata.SNOW_DENSITY, obdata.LAI, \n                         obdata.FG_TCWV, obdata.FG_CWP, obdata.FG_RWP, obdata.FG_IWP])
y0 = np.column_stack([obdata.OBSVALUE])

def x_normalise(x_orig):
    x_min = [200.0, 0, 0, 0, 0, 0, 0, 0, 0]
    x_max = [350.0, 0.75, 0.5, 300, 10, 5000, 70, 1, 2, 8]
    x_min = np.outer(np.ones(x_orig.shape[0]), np.array(x_min))
    x_max = np.outer(np.ones(x_orig.shape[0]), np.array(x_max))
    return (x_orig - (x_max + x_min) / 2.0) / (x_max - x_min) * 2.0

x1 = x_normalise(x0)
```

Dataset of 470,000 observations and colocated model data

Prepare numpy arrays of correct shape for Keras

Normalise ‘features’ $x$ to roughly $-1$ to $+1$

And... (not shown) normalise labels $y$ to within $0$ to $1$
Sigmoid activation function \( \sigma() \)

\[
b = \text{np.arange}(-5, 5, 0.01) \\
\text{plt.plot}(b, 1/(1+\text{np.exp}(-b)))
\]
Feedforward neural network - example

1 hidden layer

\[
x' = \sigma(Wx) + b
\]

\[
x_3
\]

\[
+b_1
\]

\[
+w'_{1}
\]

\[
+x_3
\]

\[
+b_2
\]

\[
+w'_{2}
\]

\[
+x_2
\]

\[
+b_3
\]

\[
+w'_{3}
\]

\[
+y = \sigma(W'x') + b'
\]

output layer

x

\[
=Wx + b
\]

1 hidden layer

y
Set up a neural network for the land surface observation operator

```python
In [21]: model = Sequential()
    ...: model.add(Dense(units=10, activation='sigmoid', input_dim=10))
    ...: model.add(Dense(units=6, activation='sigmoid'))
    ...: model.add(Dense(units=1, activation='sigmoid'))
    ...: model.summary()
    ...
    ...: model.compile(loss='mean_squared_error', optimizer='adam')
    ...
Model: "sequential_2"
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_4 (Dense)</td>
<td>(None, 10)</td>
<td>110</td>
</tr>
<tr>
<td>dense_5 (Dense)</td>
<td>(None, 6)</td>
<td>66</td>
</tr>
<tr>
<td>dense_6 (Dense)</td>
<td>(None, 1)</td>
<td>7</td>
</tr>
</tbody>
</table>

Total params: 183
Trainable params: 183
Non-trainable params: 0

Train it (about 25 minutes on a linux workstation)

```python
history = model.fit(x1, y1, epochs=100)
```
Results (ability to fit training dataset)

Physically-based simulation produced by IFS (RTTOV for atmosphere, dynamical emissivity retrieval for surface emissivity)

\[
\text{predict} = y_{\text{unnormalise}}(\text{model.predict}(x1))
\]

Hand-written function to recover TB
Results cont. (ability to fit training dataset)

ML predicted (TB [K])

TBs not low enough
The data assimilation approach: physical models, prior knowledge
Physical forward model

Satellite observations

SSMIS F-17 channel 13 (19 GHz, v)
Microwave brightness temperatures
3rd December 2014

Geophysical variables

- Atmospheric temperature, water vapour, wind, cloud, precipitation
- Skin and substrate temperature and moisture
- Ocean wind, waves, foam
- Sea-ice
- Snowpack
- Ice
- Vegetation
- Soil

\[
y = h \left( \begin{array}{c}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_{\ldots}
\end{array} \right)
\]
Physical forward model

Satellite observations

Geophysical variables

Equations & parameters – where sometimes knowledge is quite uncertain

Gas spectroscopy
Scattering from hydrometeors
Cloud and precipitation micro and macro-structure
Snow / ice grain size and structure

SSMIS F-17 channel 13 (19 GHz, v)
Microwave brightness temperatures
3rd December 2014

\[ y = h \]

\[
\begin{bmatrix}
\chi_1 \\
\chi_2 \\
\chi_3 \\
\chi_4 \\
\chi_5 \\
\chi_{...}
\end{bmatrix},
\begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
w_4 \\
w_5 \\
w_{...}
\end{bmatrix}
\]
The forward and inverse problem

\[ y = h(x, w) \]

Observations

Geophysical state

Model parameters

Forward model

\[ x, w = h^{-1}(y) \]

Inverse model

No unique solution: ill-posed

The best that observations can do is to provide a statistical improvement in our knowledge of x and w
The inverse problem solved by Bayes theorem

\[ P(x, w | y) = K (y, P(y|x, w), P(x, w)) \]

**Observations**

**Geophysical state**

**Model parameters**

(Posterior) Probability of x and w given y

Prior probability of x and w

Probabilistic equivalent of the forward model h()
Cost function for variational DA

Assume Gaussian errors (error standard deviation $\sigma$) and for clarity here simplify to scalar variables and ignore any covariance between observation, model or state error.

$$J(x, w) = \frac{(y - h(x, w))^2}{(\sigma y)^2} + \frac{(x^b - x)^2}{(\sigma x)^2} + \frac{(w^b - w)^2}{(\sigma w)^2}$$

- **DA Cost function**
- **Observation term**
- **Prior knowledge of state**
- **Prior knowledge of model**
Cost / loss function equivalence of ML and variational DA

Assume Gaussian errors (error standard deviation $\sigma$) and for clarity here simplify to scalar variables and ignore any covariance between observation, model or state error.

\[
J(x, w) = \frac{(y - h(x, w))^2}{(\sigma_y)^2} + \frac{(x^b - x)^2}{(\sigma_x)^2} + \frac{(w^b - w)^2}{(\sigma_w)^2}
\]

- **ML**
  - Loss function
  - Basic loss function
  - Feature error?
  - Weights regularisation

- **DA**
  - Cost function
  - Observation term
  - Prior knowledge of state
  - Prior knowledge of model
<table>
<thead>
<tr>
<th>Machine learning (e.g. NN)</th>
<th>Variational data assimilation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Labels</strong></td>
<td><strong>Observations</strong></td>
</tr>
<tr>
<td>$y$</td>
<td>$y^o$</td>
</tr>
<tr>
<td><strong>Features</strong></td>
<td><strong>State</strong></td>
</tr>
<tr>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td><strong>Neural network or other learned models</strong></td>
<td><strong>Physical forward model</strong></td>
</tr>
<tr>
<td>$y' = W(x)$</td>
<td>$y = H(x)$</td>
</tr>
<tr>
<td><strong>Objective or loss function</strong></td>
<td><strong>Cost function</strong></td>
</tr>
<tr>
<td>$(y - y')^2$</td>
<td>$J = J^b + (y^o - H(x))^T R^{-1} (y^o - H(x))$</td>
</tr>
<tr>
<td><strong>Regularisation</strong></td>
<td><strong>Background term</strong></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
<tr>
<td><strong>Iterative gradient descent</strong></td>
<td><strong>Conjugate gradient method (e.g.)</strong></td>
</tr>
<tr>
<td><strong>Back propagation</strong></td>
<td><strong>Adjoint model</strong></td>
</tr>
<tr>
<td></td>
<td>$\frac{\partial J}{\partial x} = H^T \frac{\partial J}{\partial y}$</td>
</tr>
<tr>
<td><strong>Train model and then apply it</strong></td>
<td><strong>Optimise state in an update-forecast cycle</strong></td>
</tr>
</tbody>
</table>
Need for model learning even in current DA approach
4D-Variational DA with a perfect model assumption

Geophysical state

Background model trajectory

Analysis model trajectory

Improved forecast

Observations

DA adjusts initial state to better fit observations
But models are not perfect...

- Within the fluid systems (atmosphere / ocean):
  - Truncation of model resolution and dynamical processes (e.g. 8 km currently) therefore needing parametrisations of diffusion, turbulence, gravity waves etc.
  - The water problem: phase changes
    - Cloud and precipitation processes – microscopic scales
    - Sea-ice, snowpack, soil moisture

- Throughout the earth system – many processes or boundary conditions that are not modeled from physical first principles and/or are strongly heterogeneous in time and space
  - Biosphere, lithosphere, aerosols ...

Wilson Bentley (1902, [http://www.photolib.noaa.gov](http://www.photolib.noaa.gov))
Model learning from observations

**State complexity**

- Parameter estimation in meteorology

- Parameter estimation in hydrogeology

- Chemical source terms

**Model complexity**

- Parameter identification in meteorology

- Use DA to infer ODE representation of Lorenz models (Bocquet et al., 2019)
  https://doi.org/10.5194/npg-26-143-2019

**Completeness of learning**

- New wave of ML and DA

**Simultaneous state and parameter estimation:**
- Autoconversion parameter learnt in a GCM (Kotsuki et al., 2020)
  https://doi.org/10.1029/2019JD031304

- Roughness length learnt in local area model (Ruckstuhl and Janjić, 2020)
  https://doi.org/10.1175/MWR-D-19-0233.1

**Pure parameter estimation, e.g:**
- Groundwater modeling, e.g. hydraulic conductivity (Zhou et al., 2014).
  https://doi.org/10.1016/j.advwatres.2013.10.014

- CO₂ source term estimation (Peylin et al., 2013)
  https://doi.org/10.5194/bg-10-6699-2013

**Model (and sometimes) state estimation in simple models:**
- ML of KS equation (Pathak et al., 2018)
  https://doi.org/10.1103/PhysRevLett.120.024102
Bayesian view
Bayesian equivalence of ML and DA

As a Bayesian network

\[ y = h(x, w) \]

Geer (2021)  
Bocquet et al. (2020)  
Abarbanel et al. (2018)  
Goodfellow et al. (2016)  

https://doi.org/10.21957/7fyj2811r  
https://doi.org/10.1162/neco_a_01094  
https://doi.org/10.1175/1520-0477(1998)079%3C1855:ANNMTP%3E2.0.CO;2  
https://www.deeplearningbook.org
Bayesian networks: representing the factorisation of joint probability distributions

1. Factorise in two different ways using the chain rule of probability

\[ P(y, x, w) = P(x|w, y)P(w|y)P(y) \]
\[ P(y, x, w) = P(y|x, w)P(x|w)P(w) \]

2. Equate the two right hand sides and rewrite

\[ P(x|w, y)P(w|y) = \frac{P(y|x, w)P(x|w)P(w)}{P(y)} \]

3. Rewrite by putting back the joint distributions of x,w: Bayes’ rule

\[ P(x, w|y) = \frac{P(y|x, w)P(x, w)}{P(y)} \]
Illustration (scalar $w$, $x$, $y$)

prior
$P(x, w)$

posterior
$P(x, w | y)$

observation
Adding the time dimension

Assume $x_{t+1}$ depends only on initial state $x_t$ and parameters $w$

$$P(x_{t+1}, w, x_t | y_t) = P(x_{t+1} | w, x_t)P(x_t, w | y_t)$$

Forward model
Bayes’ rule (previous slides)

$$P(x_{t+1}, w | y_t) = \int P(x_{t+1}, w, x_t | y_t) \, dx_t$$

‘Marginalisation’
Time evolution of state – cycled data assimilation
Recursive neural network ≈ cycled data assimilation
Time evolution of state – cycled data assimilation

Time evolving state

Time-constant model (parameters)

Observations
Recursive neural network

Inputs

Hidden state

Constant NN weights

Outputs

Inputs:
- c
- a
- t

Hidden state:
- \( x_1 \)
- \( x_2 \)
- \( x_3 \)
- \( x_4 \)

Outputs:
- \( y_1 \)
- \( y_2 \)
- \( y_3 \)
Inside an atmospheric model & data assimilation timestep

One model time-step
Learning an improved model of cloud physics (ML or DA)

Cloud physics

\[ x_{1.1} \]
\[ x_{1.2} \]

We want to train a model against observations, but we cannot directly observe gridded intermediate states \( x_{1.1} \) and \( x_{1.2} \) ... or more precisely model tendencies ...
Inside an atmospheric model

... so train the model inside the data assimilation system
How ML can help typical DA
Learning model error

\[ x_t \xrightarrow{\text{Physical model}} x_{t+1}^{phys} + \Delta x_{t+1}^{nn} \xrightarrow{\text{Neural network}} x_{t+1} \]


Offline ML – learn weights \( w \) of model for model error tendency

\[
J(x, w) = (y - h(x, w))^2
\]

Weak-constraint 4D-Var, done online (\( w \) is a field of model error tendencies)

\[
J(x, w) = \\
\quad \quad \ldots + \frac{(w^b - w)^2}{(\sigma w)^2}
\]

- Train a NN on IFS non-oographic gravity wave drag scheme (NOGWD)
- Differentiate the NN via the chain rule to get:
  - TL (tangent-linear)
  - Adjoint (== back-propagation)
- Use the NN TL and adjoint in the 4D-Var (but keep the physical nonlinear NOGWD scheme)
  - Successful 2-month run of IFS cycling 4D-Var completed

\[ \varepsilon = \langle |H\delta x - h(x + \delta x) - h(x)| \rangle \]
Satellite bias correction using ML

- Application: SSMIS instrument solar-dependent calibration anomalies
- Offline training of a bias model in observation space
  - Learn bias between observations and model as a function of satellite-solar orbital parameters.
- Problems: many latent variables, orbit changes, model changes

### Daily binned bias

- Training set: 360 days
- Model upgrade

### Static NN model predicted

- Bias [K]
Satellite bias correction

- Dynamic NN approach:
  - Re-train each day on yesterday’s biases
  - Learn bias as a function of orbit angle
- This is essentially an offline, nonlinear version of VarBC, which is a standard bias correction technique for DA
Sequential learning

• Batch learning
  – relearn on everything, i.e. past and new data, each time?
  – take pre-trained network and run training on new data only

• Sequential learning
  – e.g. Online sequential Extreme Learning Machine (OS-ELM, Liang et al., 2006) [10.1109/TNN.2006.880583](https://doi.org/10.1109/TNN.2006.880583)
  – e.g. Forecasting daily streamflow using OSELM (Lima, Cannon, Hsieh, 2016) [10.1016/j.jhydrol.2016.03.017](https://doi.org/10.1016/j.jhydrol.2016.03.017)

• How close are these techniques to the overarching Bayesian framework, i.e. to data assimilation?
Observation operators using ML

- Fast observation operators (e.g. radiative transfer operators RTTOV and CRTM) are already ‘ML’ – just using linear regression or heuristic model components
  - Train on physical reference models
  - Train directly between model background and observations

- E.g. Soil moisture
  - Single hidden layer NN trained between SMOS observations and model soil moisture (H-TESSEL or IFS)
  - A version of this is used in the operational ECMWF forecasting system
How DA can help typical ML
Physically constrained machine learning

```python
def net_u(self, x, t):
    u = self.neural_net(tf.concat([x,t], 1), self.weights, self.biases)
    return u

def net_f(self, x, t):
    u = self.net_u(x, t)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - self.nu*u_xx
    return f

self.loss = tf.reduce_mean(tf.square(self.u_tf - self.u_pred)) + \
            tf.reduce_mean(tf.square(self.f_pred))
```

Neural network

Gradients of the network

Burger’s equation

\[
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - v \frac{\partial^2 u}{\partial x^2} = 0
\]

Custom loss function

https://github.com/maziarraissi/PINNs

But ML and DA are one…