Data assimilation and machine learning

Alan Geer

European Centre for Medium-range Weather Forecasts

alan.geer@ecmwf.int

ECMWF data assimilation training course, March 7, 2024

Thanks to: Matthew Chantry, Marcin Chrust, Massimo Bonavita, Sam Hatfield, Patricia de Rosnay, Peter Dueben

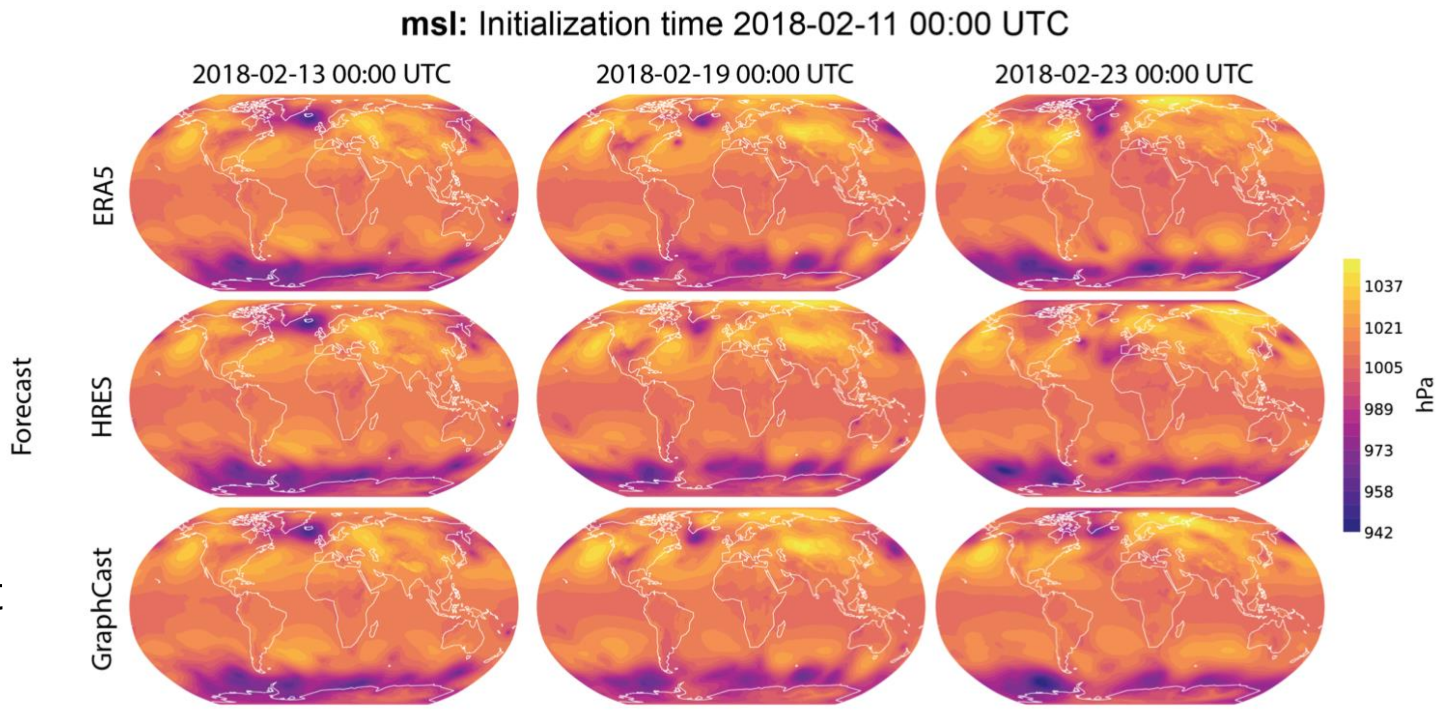
Forecast models based on machine learning are here and they're good!

- Huawei's Pangu-Weather (Bi et al., 2022, *arXiv preprint arXiv:2211.02556*)
- Google DeepMind's GraphCast (Lam et al., 2022, *arXiv preprint arXiv:2212.12794*)

ERA5: reanalysis as **training data (1979-2017)** and validation data (2018)

HRES: ECMWF T1279Co (9 km) 10 day forecast

GraphCast: 10 day forecast at 0.25 degrees (25 km)



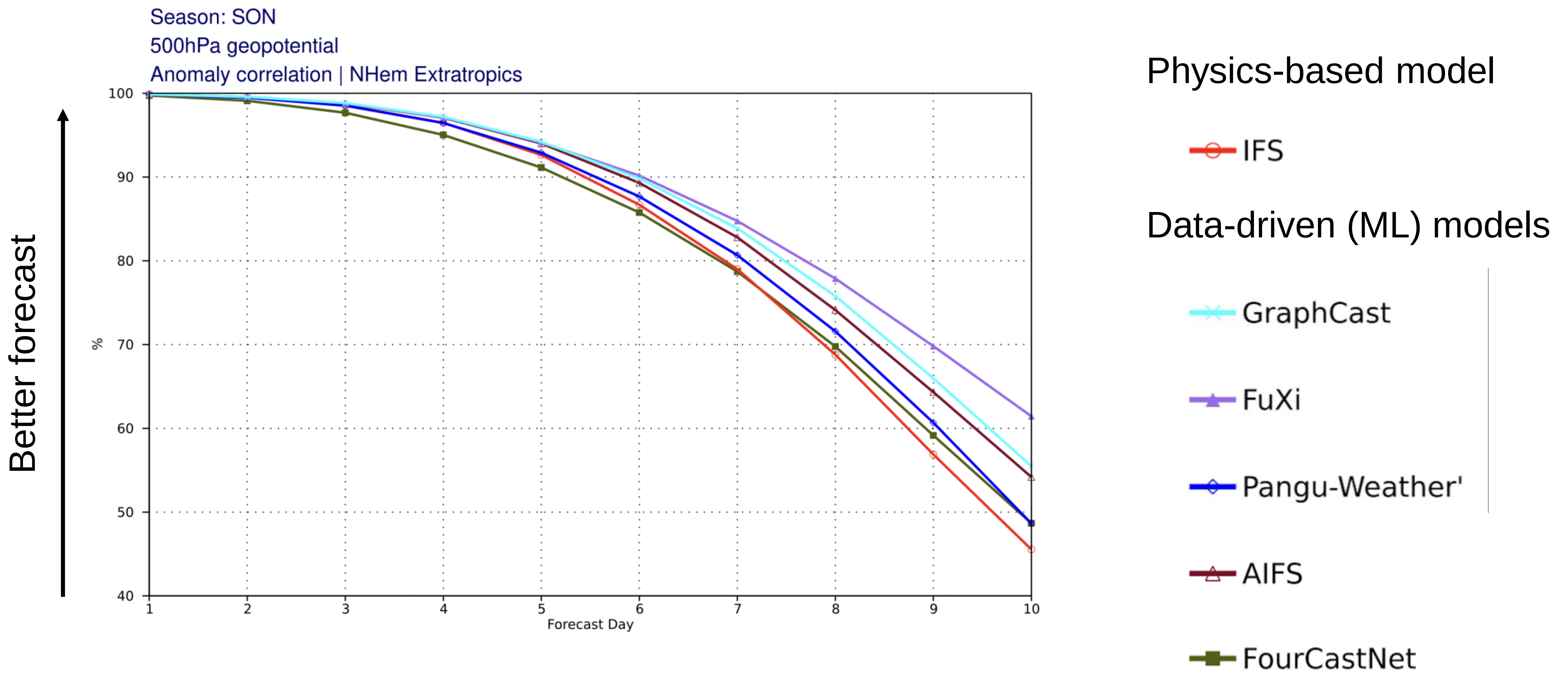
<u>Run time</u>

30 minutes (128 nodes of HPC)

1 minute (1 TPU)

https://arxiv.org/pdf/2212.12794.pdf

Machine learning weather forecasts out-perform* physics-based models



ECMWF charts catalogue - experimental: machine learning models
Ben-Bouallegue et al. (2023) The rise of data-driven weather forecasting - https://doi.org/10.48550/arXiv.2307.10128
Bi et al. (2023) Accurate medium-range global weather forecasting with 3D neural networks - https://doi.org/10.1038/s41586-023-06185-3
Lam et al. (2023) Learning skilful medium-range global weather forecasting - https://doi.org/10.1126/science.adi2336

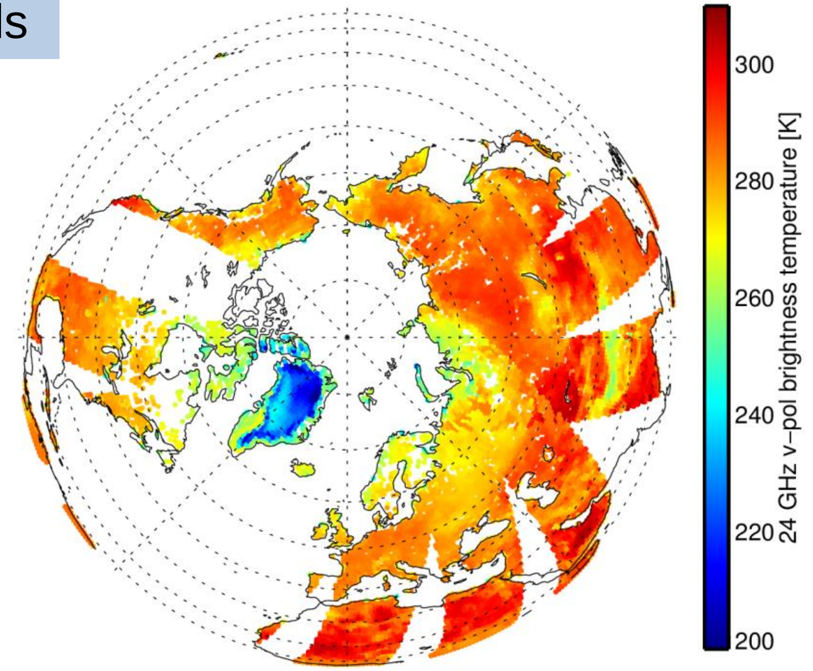
An ML example: microwave land surface observation operator

Python, Keras, Tensorflow, Numpy, Matplotlib, Xarray

Datasets

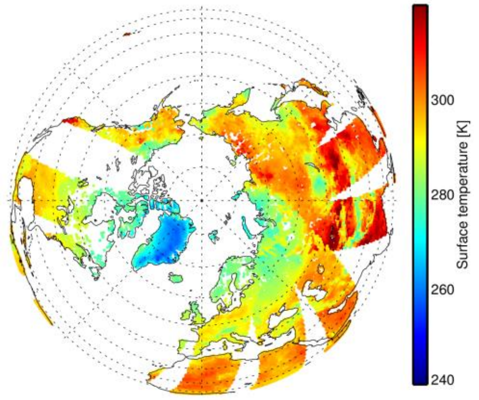
AMSR2 24GHz v-pol observations

Labels

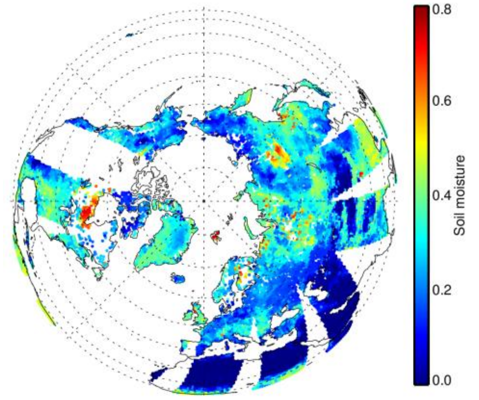


10 possible predictors for the brightness temperature

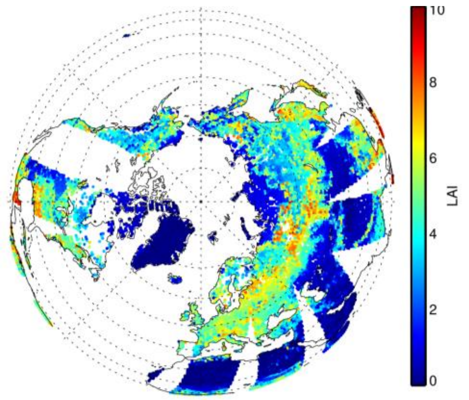
Features



Skin temperature



Soil moisture



Leaf area index

+ orography, snow depth, snow density, integrated water vapour, cloud, rain and snow water contents

Data preparation

Dataset of 470,000 observations and colocated model data

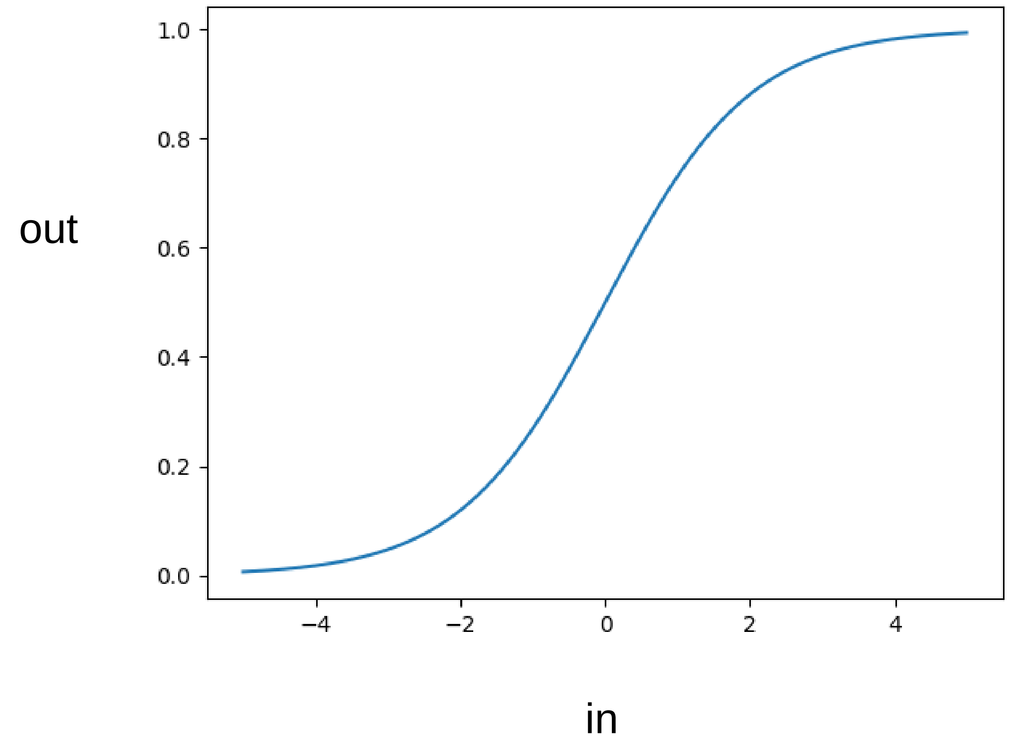
```
obdata = xr.open_dataset('/perm/rd/stg/odb/hkhg/ml_amsr2_chan9.nc')

x0 = np.column_stack([obdata.TSFC,          obdata.SOIL_MOISTURE, obdata.SNOW_DEPTH, \
                      obdata.SNOW_DENSITY, obdata.LAI,            obdata.OROGRAPHY, \
                      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, 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)
```

Prepare numpy arrays of correct shape for Keras

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

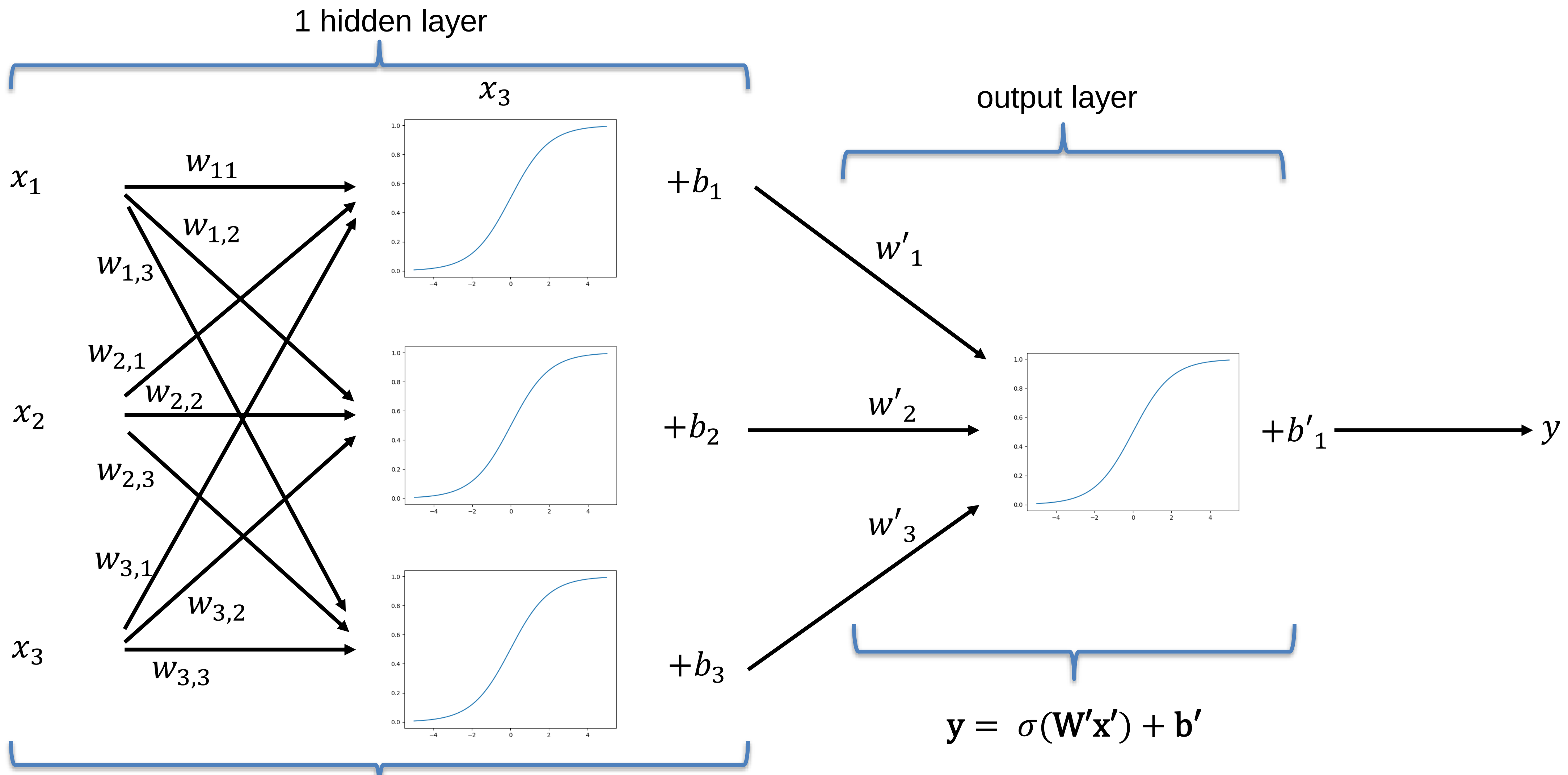
And... (not shown) normalise labels y to within 0 to 1

Sigmoid activation function $\sigma()$



```
b=np.arange(-5,5,0.01)
plt.plot(b,1/(1+np.exp(-b)))
```

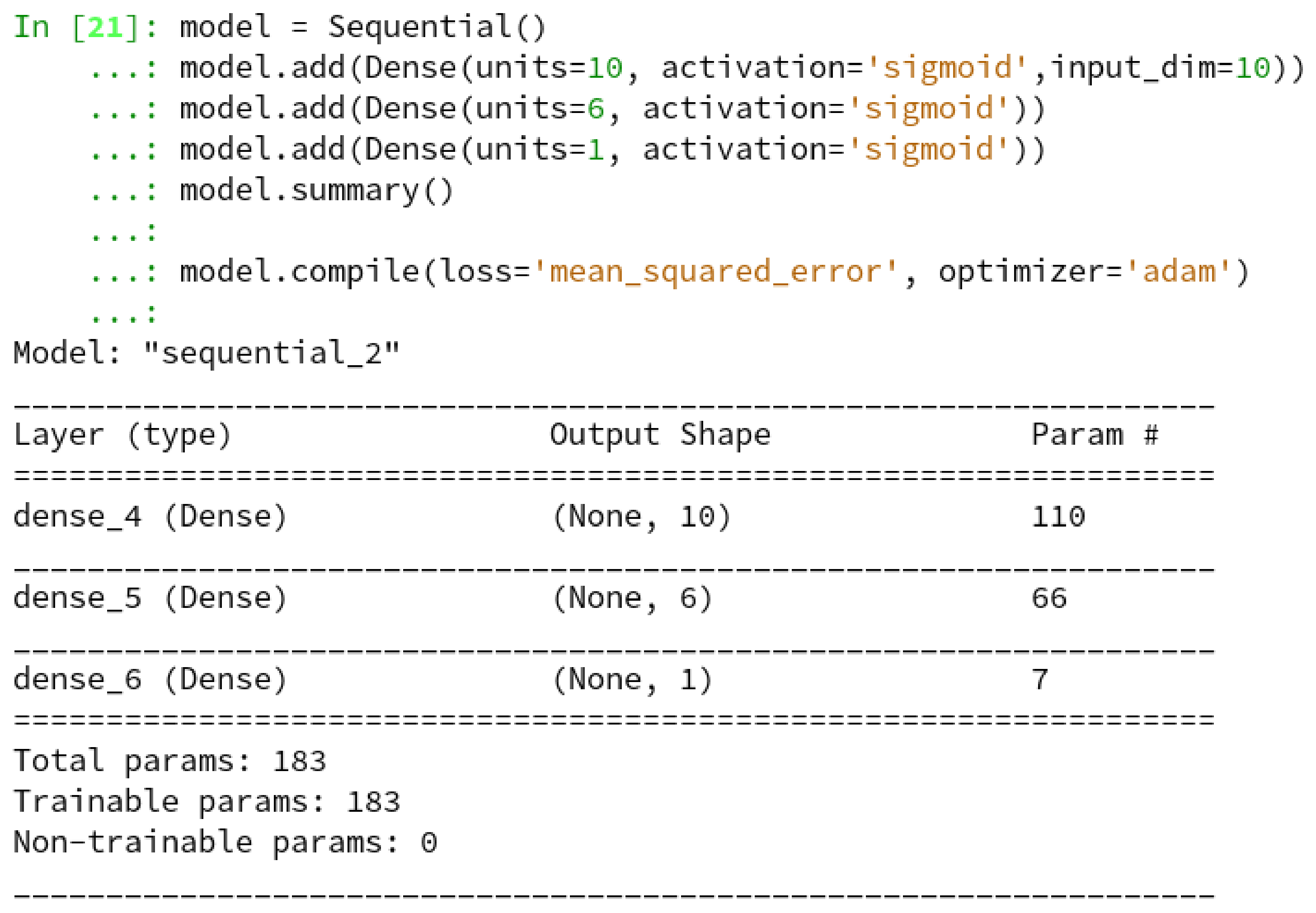
Feedforward neural network - example



$$\mathbf{x}' = \sigma(\mathbf{W}\mathbf{x}) + \mathbf{b}$$

$$\mathbf{y} = \sigma(\mathbf{W}'\mathbf{x}') + \mathbf{b}'$$

Set up a neural network for the land surface observation operator



```
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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 10)                110
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 66
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 7
=================================================================
Total params: 183
Trainable params: 183
Non-trainable params: 0
_________________________________________________________________
```

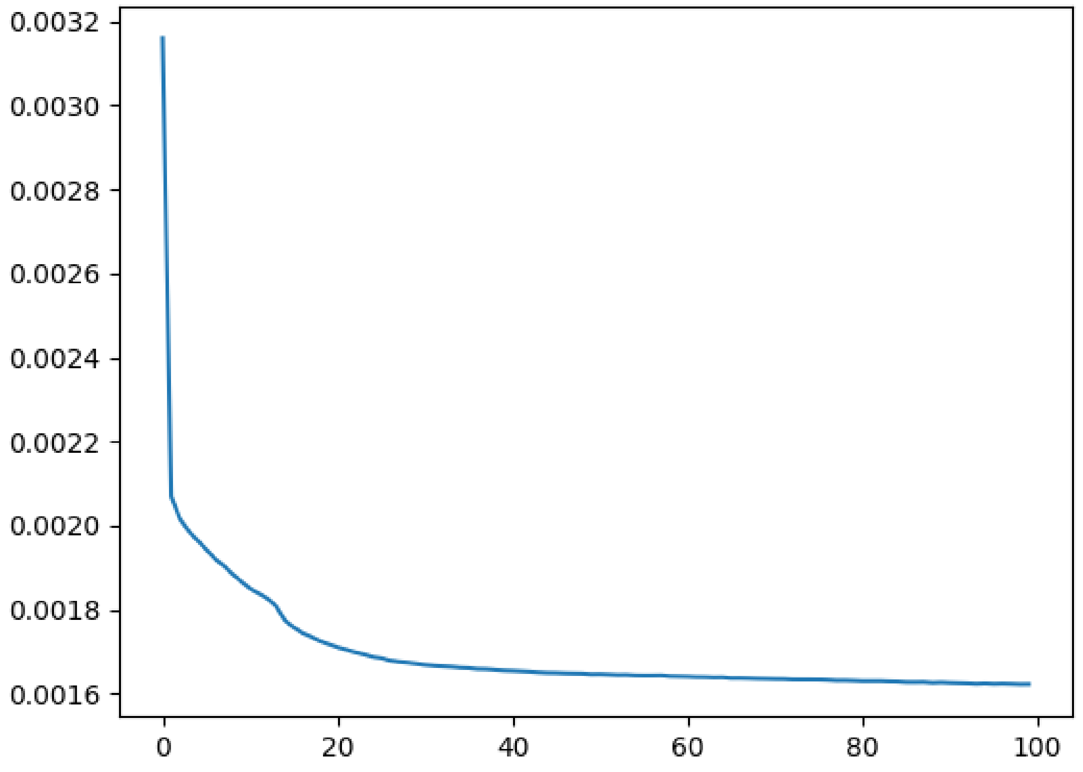
Train it (about 25 minutes on a linux workstation)

```
history = model.fit(x1, y1, epochs=100)
```

Loss function

$$J_{\text{obs}} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{obs},i} - y_{\text{sim},i})^2$$

Default "loss function" is just the 4D-Var Jo without representation of observation error.



epoch

Adam – a sophisticated stochastic gradient descent (SGD) minimiser

"Backpropagation" is ML's term for computing gradients of the cost function with respect to trainable parameters, using calls through the adjoints of each neural network layer.

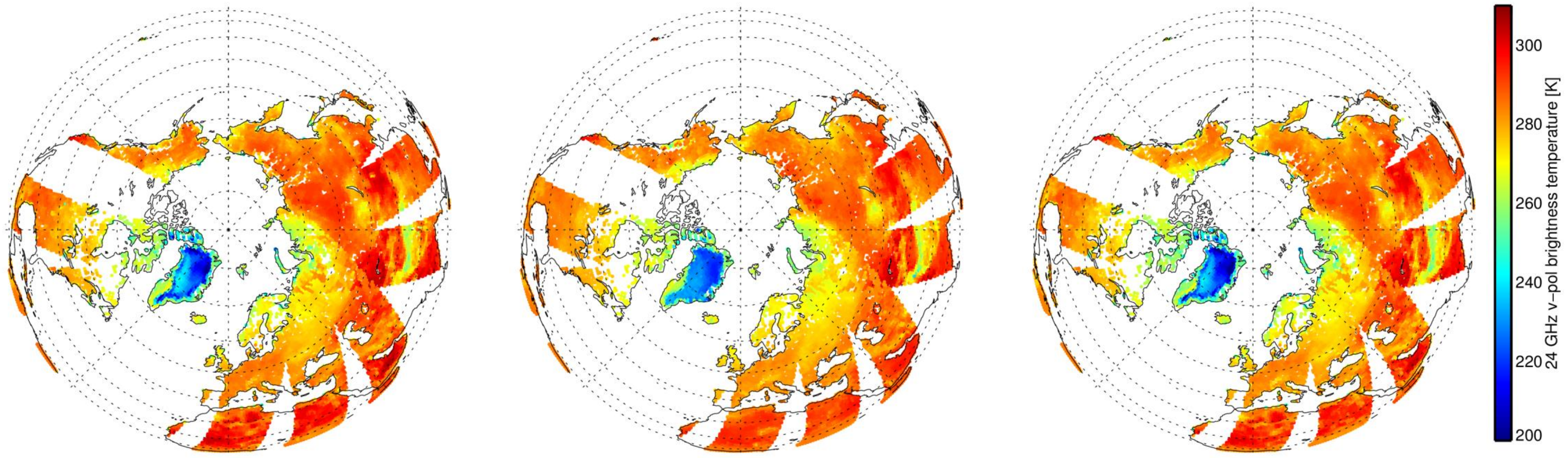
~Variational data assimilation without error representations, without regularisation, without state update

Results (ability to fit training dataset)

Observations

ML predicted

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



```
predict = y_unnormalise(model.predict(x1))
```

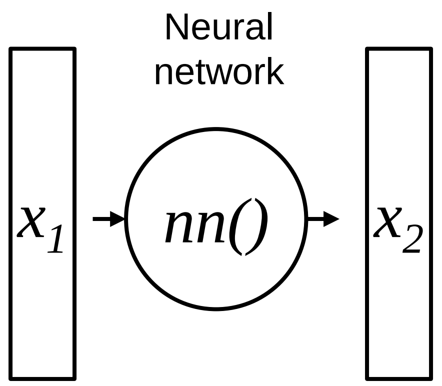
Hand-written function to recover TB

Problems with this toy NN model for 24 GHz radiances

- It’s not as good as the current physical methods
- The input variables are not sufficient to drive the outputs
  - Missing variables – e.g. over Greenland, detailed knowledge of snow and ice microstructure
- One of the driving problems for all-surface data assimilation:
  - Neither the models nor the input state are fully known
  - Chicken and egg problem: can’t train the model if you don’t know the necessary inputs well enough

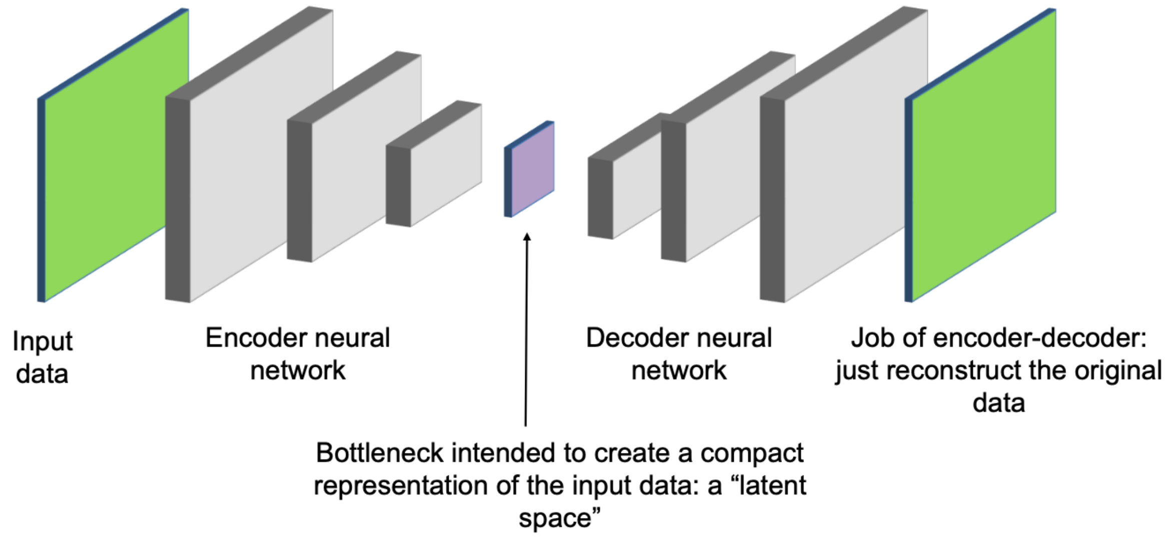
Types of ML

Types of ML – supervised learning



Supervised learning:

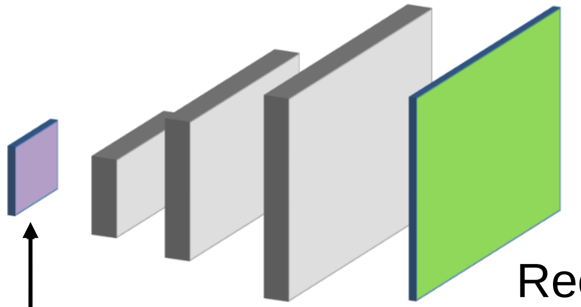
- ML as a "universal function approximator" (Hornik, 1991)
- Both inputs x1 and outputs x2 need to be provided as training data
- An "emulator" / "surrogate" / "empirical model"


Input data
Encoder neural network
Decoder neural network
Job of encoder-decoder: just reconstruct the original data
Bottleneck intended to create a compact representation of the input data: a "latent space"

Encoder-decoder:

- Data compression
- Data assimilation in the space of an autoencoder (Peyron et al., 2021)
- Still needs both inputs and outputs to train the model

Types of ML – unsupervised learning – generative ML

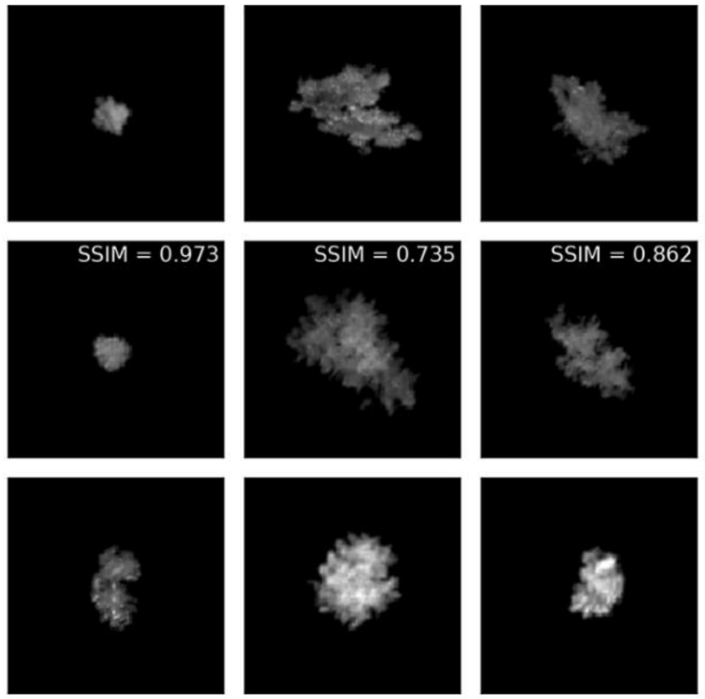


Latent space: a reduced statistical description of a phenomemon

A bit like a set of eigenvalues in a principal component decomposition

What if we could just have the decoder?

- How do we train it?
  - We could train an encoder-decoder on something, and then throw away the encoder.
  - Or find some more clever way...



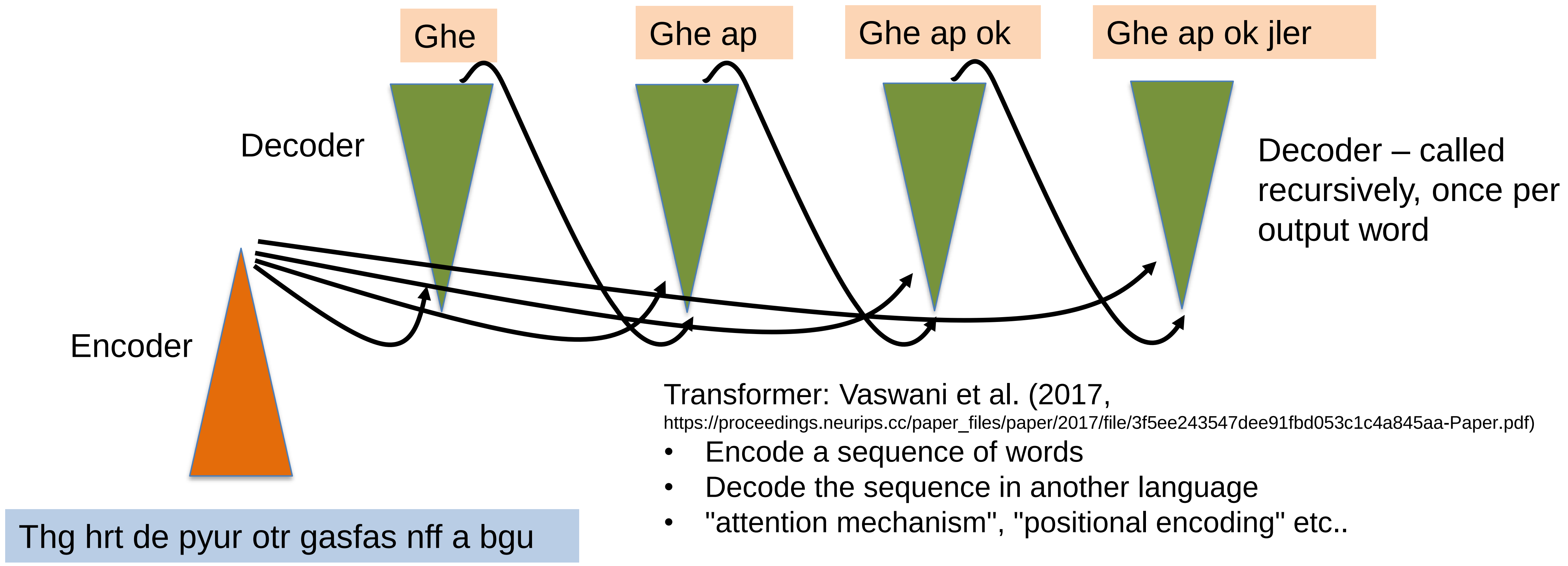
Snowflake images from Leinonen and Berne (2021, https://doi.org/10.5194/amt-13-2949-2020)

Generative Adversarial Network (GAN):

- Generator (~decoder): make an image
- Discriminator (~encoder): given an image, tell if it is real or fake -> drives the loss function

Generative ML - transformers

Outputs – translate to another language (original google application) – or continue in same language (GPT-1)



Decoder – called recursively, once per output word

Inputs – in one language (whole word sequence at once)

Transformer: Vaswani et al. (2017, https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf)

- Encode a sequence of words
- Decode the sequence in another language
- "attention mechanism", "positional encoding" etc..

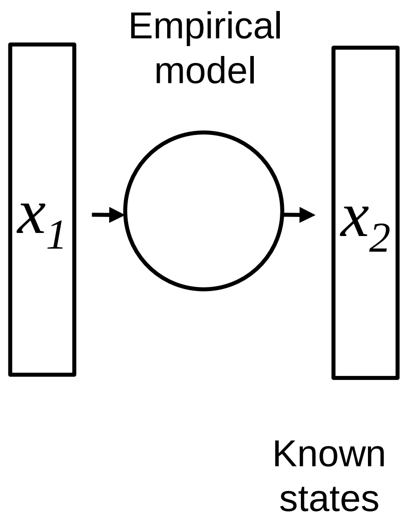
Generative Pre-trained Transformer (GPT)

- GPT-1 is *just* a transformer like this (120 million parameters)
- GPT-2/3/4 architecture is not public but broadly an extension of these concepts (parameters: v3: 175 billion, v4: 1000 billion???)

How ML can benefit DA

What does ML bring for data assimilation? 1) surrogate modelling

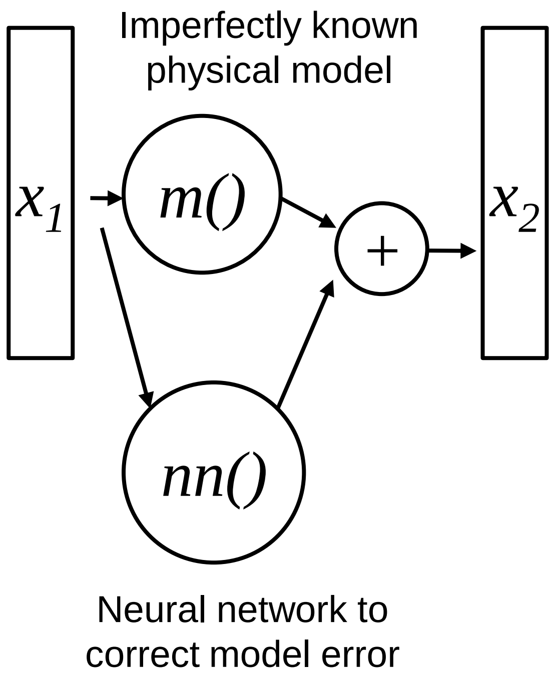
- Train against existing datasets, typically an existing physical model
- Acceleration:
  - E.g. use many more ensemble members, allowing previously unaffordable data assimilation algorithms (Chattopadhyay et al. , 2021, GMDD, https://doi.org/10.5194/gmd-2021-71, generate a 1000-member ensemble)
  - E.g. generate samples of model error from which to derive a model error covariance matrix: Bonavita and Laloyaux, 2022 (https://arxiv.org/abs/2209.11510)
- Space compression:
  - E.g. data assimilation in the latent space of an auto encoder (Peyron et al., 2021, Latent space data assimilation by using deep learning https://arxiv.org/abs/2104.00430)
- Numerical differentiation:
  - E.g. provide a tangent linear and adjoint for variational data assimilation: Gravity wave drag scheme emulated by ML and then ML used to provide TL/adjoint: Hatfield, Chantry et al., 2021(https://doi.org/10.1029/2021MS002521)



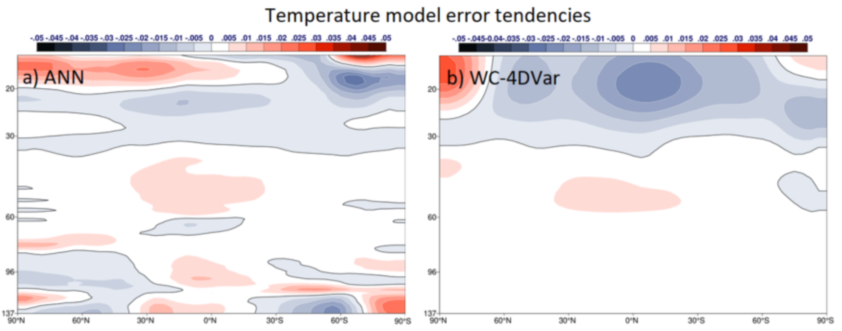
What does ML bring for data assimilation? 2) bias correction

Correct model or observation error:

- Review: Farchi et al. (2021) https://doi.org/10.1016/j.jocs.2021.101468
- Train against historical data assimilation increments or departures
- It is possible to train “online” inside a data assimilation system
    - Separate, iterative approach (e.g. Brajard et al., 2020, https://doi.org/10.1016/j.jocs.2020.101171)
    - Online training inside variational data assimilation is in development
- A nonlinear extension to existing data assimilation bias correction methods
    - Weak constraint data assimilation
    - Parameter estimation
    - Variational bias correction (VarBC)
- Example: model error correction in IFS, Bonavita and (https://doi.org/10.1029/2020MS002232)


Imperfectly known physical model
x_1
$m()$
$+$
x_2
$nn()$
Neural network to correct model error

Neural network estimate of model bias



Weak constraint 4D-Var

What does ML bring for data assimilation? 3) replace entire DA system

The analysis equation

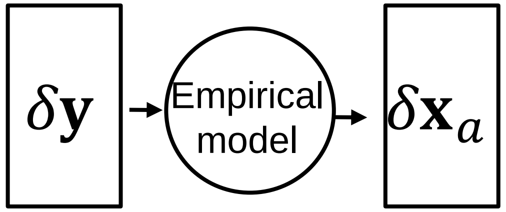
$$\boxed{\mathbf{x}_a = \mathbf{x}_b + \mathbf{K}(\mathbf{y} - H\mathbf{x}_b)}$$

(*model space*)

$$\boxed{\delta\mathbf{x}_a = \mathbf{K}\delta\mathbf{y}}$$

(observation space)

$\mathbf{x}_a$ - analysis vector
$\mathbf{x}_b$ - background vector
$\mathbf{y}$ - observation vector
$H(\mathbf{x}_b)$ - forward observation operator
H - Jacobian or tangent linear approximation of *H*
R – observation error covariance
B – background error covariance
$\mathbf{K} = \mathbf{B}\mathbf{H}^T(\mathbf{H}\mathbf{B}\mathbf{H}^T + \mathbf{R})^{-1}$ Kalman gain matrix
$\delta\mathbf{y} = \mathbf{y} - H\mathbf{x}_b$ is the innovation vector
$\delta\mathbf{x}_a = \mathbf{x}_a - \mathbf{x}_b$ is the analysis increment



- Train an ML model to do the DA step – e.g. take innovations $\delta\mathbf{y}$ and produce increments $\delta\mathbf{x}_a$
  - Cintra et al. (2016, Tracking the model: Data assimilation by artificial neural network. In *2016 International Joint Conference on Neural Networks (IJCNN)* (pp. 403-410))
  - Arcucci et al. (2021, https://doi.org/10.3390/app11031114)
- Problems:
  - Most work so far has been done on simple test systems (e.g. Lorenz '63)
  - Real DA uses H() operator to map diversely in space, time and to observable variables (e.g. observation space)
  - How do we adapt to new observation types? How much training (and retraining) is needed?

What does ML bring for data assimilation? 4) learn new models

Learn new models directly from observations, where physical models do not exist or are not good enough:

- If we train an ML forecasting model against analysis, it learns to map analyses (the combination of forecast and observations) not forecast state to forecast state (if training to a physical forecast model)
  - ML forecasting models like Pangu-Weather or Graphcast may already be encoding some “new physics” – at minimum, online bias corrections
- Train against high resolution models (e.g. train moist physics scheme for a GCM against a convection resolving model)
- Train directly against observations?
  - Irregular and indirect link to observation space makes this hard
  - Solution? **Train empirical components inside a physical DA system**

What does ML bring for data assimilation? 5) new observation operators

Observations

$$\begin{tikzcd}
\boxed{y} \\
h(x,w) \arrow[u] \\
\boxed{x} \arrow[u]
\end{tikzcd}$$

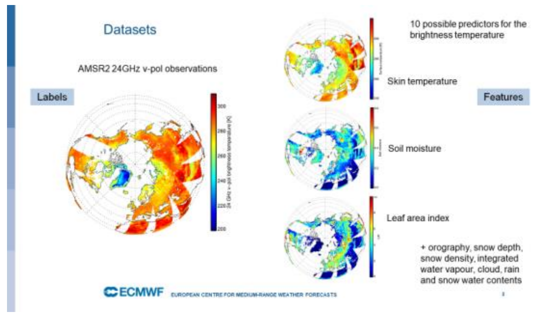
Well-constrained model variables from the DA system

E.g. backscatter triplet from scatterometer

Train a neural network observation operator (w = weights) where a physical model is not available

E.g. ocean surface wind speed

- Example: land surface radiances at microwave frequencies


Datasets
AMSR2 24GHz v-pol observations
Labels
10 possible predictors for the brightness temperature
Skin temperature
Features
Soil moisture
Leaf area index
+ orography, snow depth, snow density, integrated water vapour, cloud, rain and snow water contents
ECMWF

- Example (in retrieval direction) operationally used at ECMWF for soil moisture assimilatiuon from SMOS: Rodriguez-Fernandez et al., 2019, "SMOS Neural Network Soil Moisture Data Assimilation in a Land Surface Model and Atmospheric Impact", https://www.mdpi.com/2072-4292/11/11/1334

Theoretical links between ML and DA

The forward and inverse problem

Forward model

Observations

$$y = h(x, w)$$

Geophysical state

Model parameters

$$\cancel{x, w = h^{-1}(y)}$$

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

as the first lecture of this DA course, but extended with state AND parameters

Observations

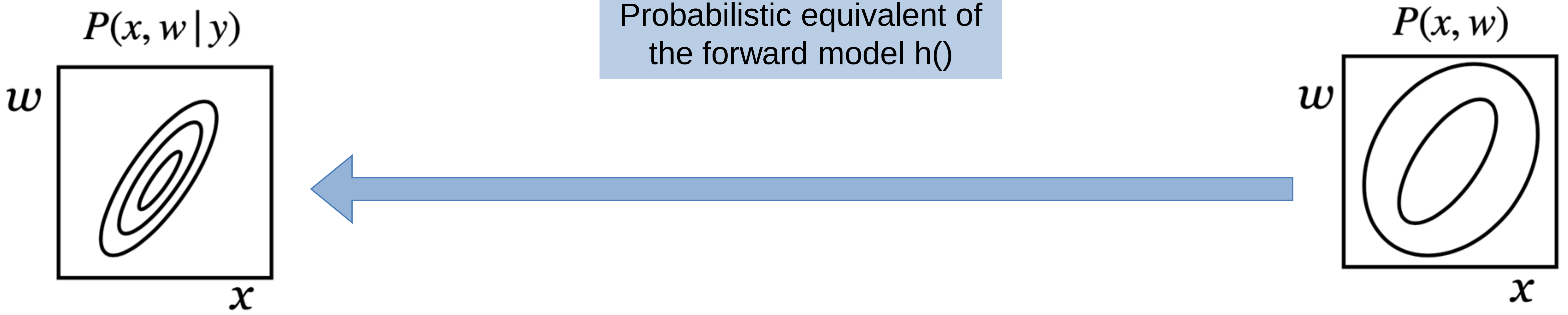
Geophysical state

Model parameters

Bayes theorem

$$P(x,w|y) = K\ (y, P(y|x,w), P(x,w))$$

Probabilistic equivalent of the forward model h()



(Posterior) Probability of x and w given y

Prior probability of x and w

Cost function for variational DA

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

Prior (background)

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

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 σ)
and for clarity here simplify to scalar variables
and ignore any covariance between observation, model or state error

ML | Loss function | Basic loss function | Feature error? | Weights regularisation

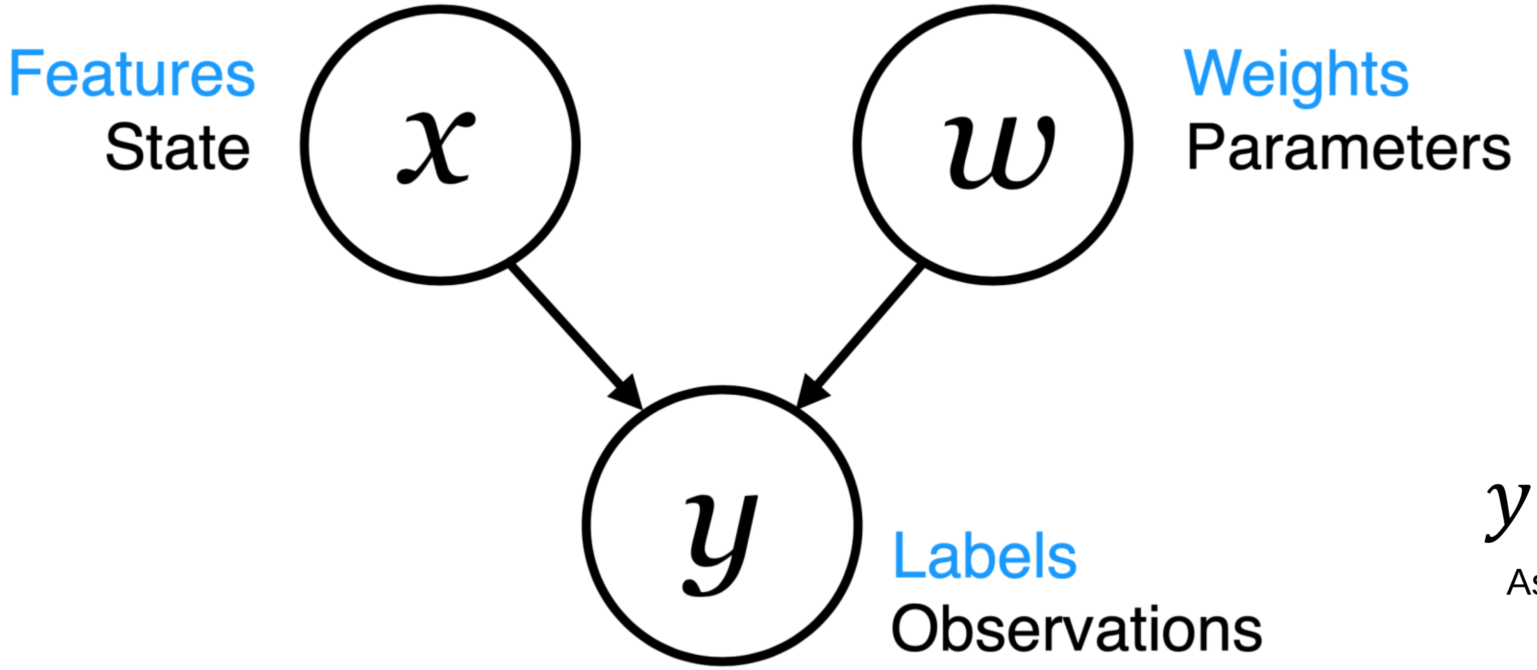
$$J(x,w) = \underbrace{\frac{(y-h(x,w))^2}{(\cancel{\sigma^y})^2}}_{J^y} + \underbrace{\xcancel{\frac{(x^b-x)^2}{(\sigma^x)^2}}}_{J^x} + \underbrace{\frac{(\cancel{w^b}-w)^2}{(\cancel{\sigma^w})^2}}_{J^w}$$

DA | Cost function | Observation term | Prior knowledge of state | Prior knowledge of model

Machine learning (e.g. NN)		Variational data assimilation	
Labels	y	Observations	y^{o}
Features	x	State	x
Neural network or other learned models	$\mathrm{y}' = W(\mathrm{x})$	Physical forward model	$\mathrm{y} = H(\mathrm{x})$
Objective or loss function	$(\mathrm{y} - \mathrm{y}')^2$	Cost function	$J = J^b + \left(\mathrm{y}^{\mathrm{o}} - H(\mathrm{x})\right)^T \mathrm{R}^{-1}\left(\mathrm{y}^{\mathrm{o}} - H(\mathrm{x})\right)$
Regularisation	$\lVert \mathrm{w} \rVert$	Background term	$J^b = \left(\mathrm{x} - \mathrm{x}^{\mathrm{b}}\right)^T \mathrm{B}^{-1}\left(\mathrm{x} - \mathrm{x}^{\mathrm{b}}\right)$
Iterative gradient descent		Conjugate gradient method (e.g.)	
Back propagation		Adjoint model	$\frac{\partial J}{\partial \mathrm{x}} = \mathrm{H}^T \frac{\partial J}{\partial \mathrm{y}}$
Train model and then apply it		Optimise state in an update-forecast cycle	

Boukabara et al. (2021) https://doi.org/10.1175/BAMS-D-20-0031.1

Bayesian equivalence of ML and DA

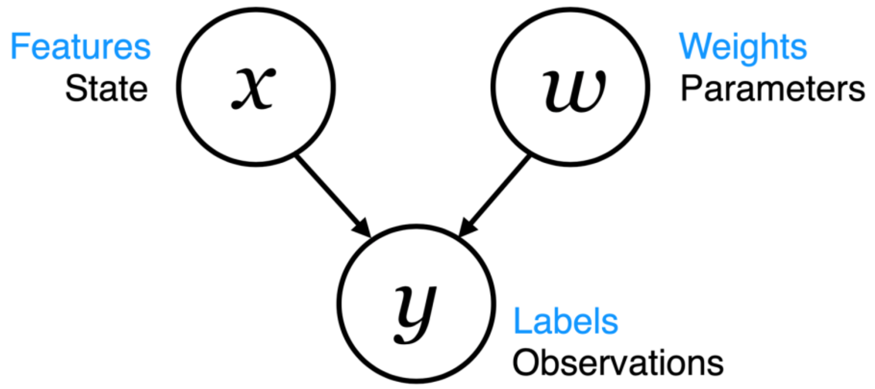


$$y = h(x, w)$$

As a Bayesian network

Geer (2021)	https://doi.org/10.1098/rsta.2020.0089
Bocquet et al. (2020)	https://arxiv.org/abs/2001.06270
Abarbanel et al. (2018)	https://doi.org/10.1162/neco_a_01094
Hsieh and Tang (1998)	https://doi.org/10.1175/1520-0477(1998)079%3C1855:ANNMTP%3E2.0.CO;2
Goodfellow et al. (2016)	https://www.deeplearningbook.org

Bayesian networks: representing the factorisation of joint probability distributions



$$P(y,x,w)$$

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

$$P(y,x,w) = P(x|w,y)P(w|y)P(y)$$

$$P(y,x,w) = P(y|x,w)P(x|w)P(w)$$

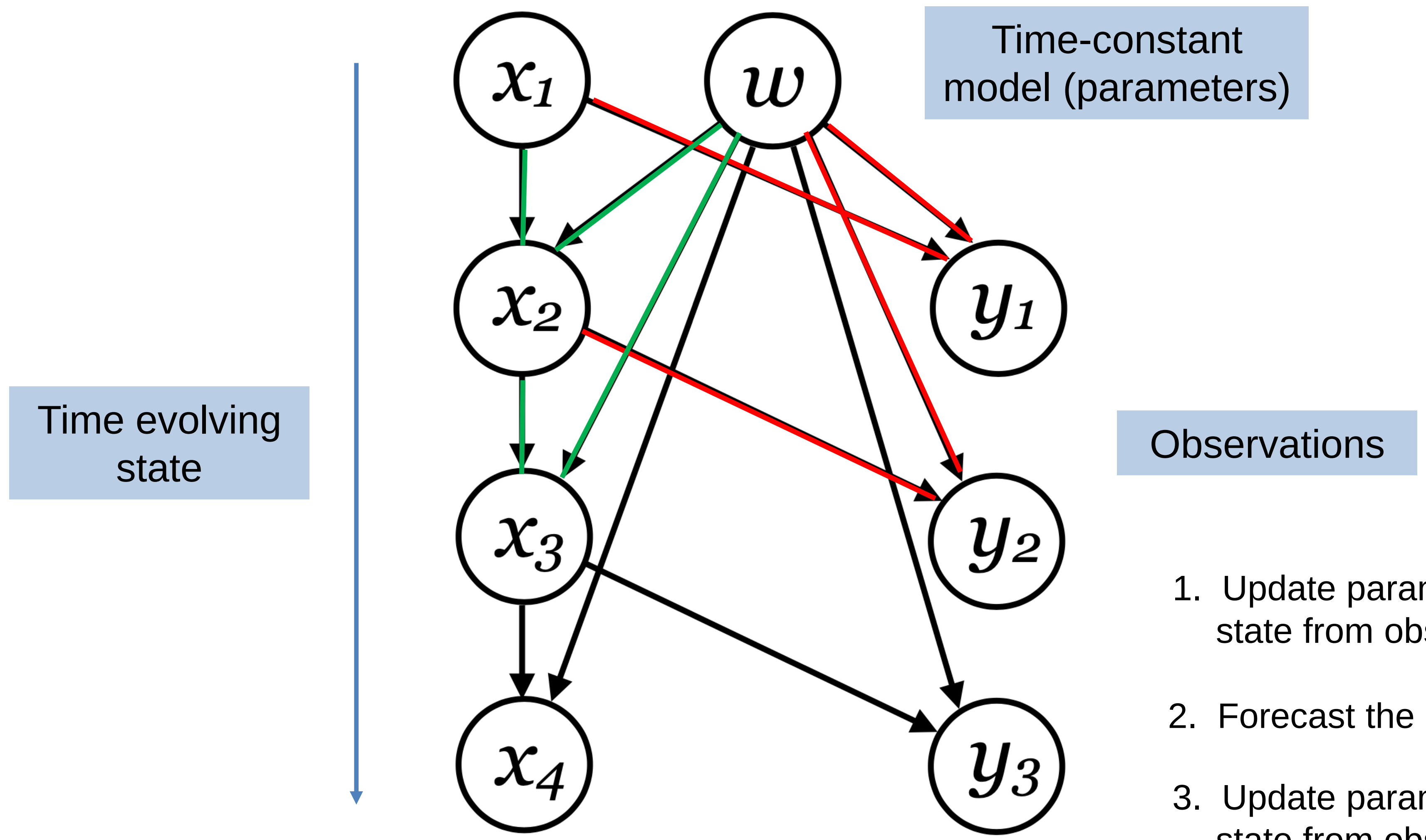
2. Equate the two right hand sides and rewrite

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

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

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

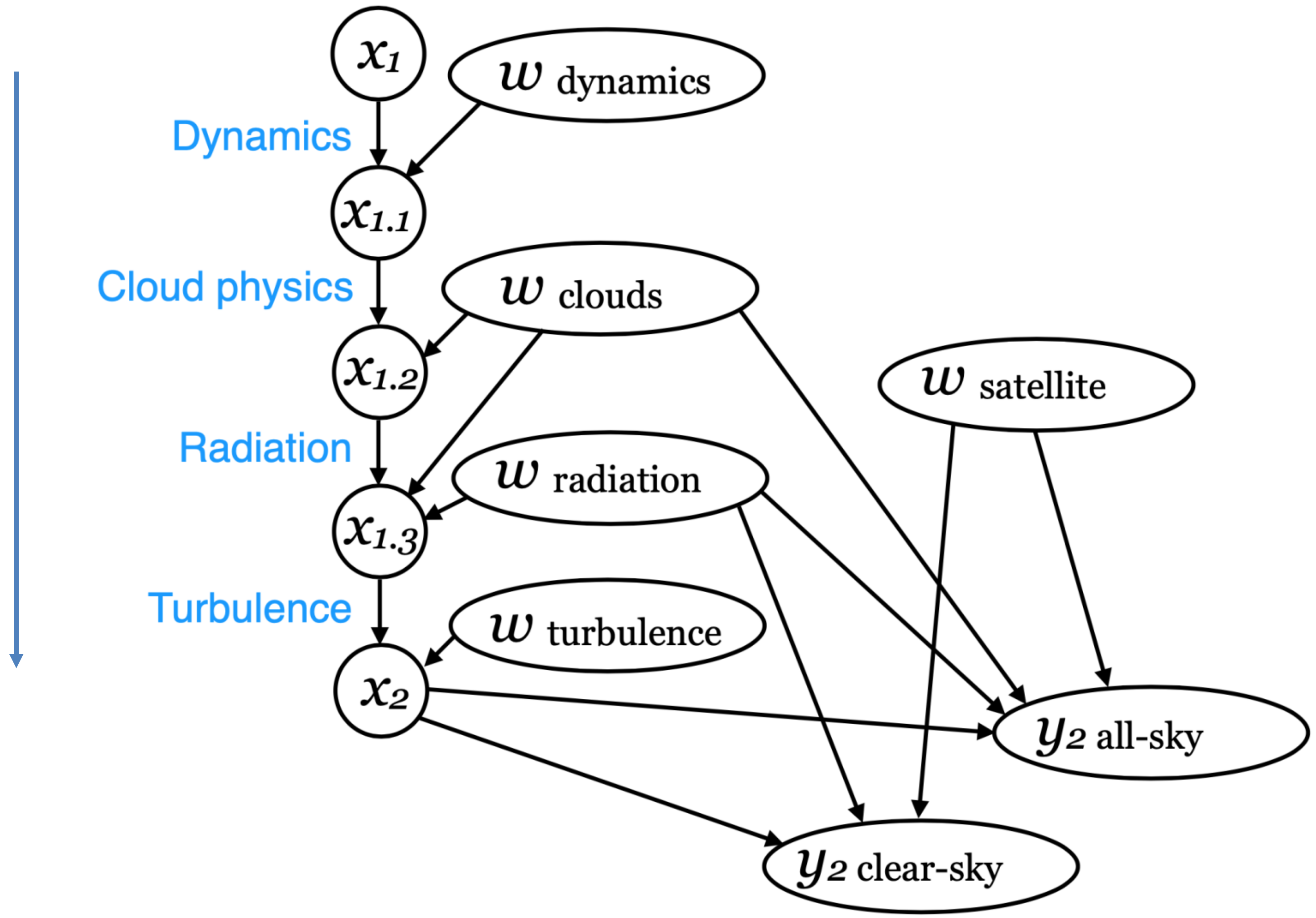
Time evolution of state – cycled data asimilation



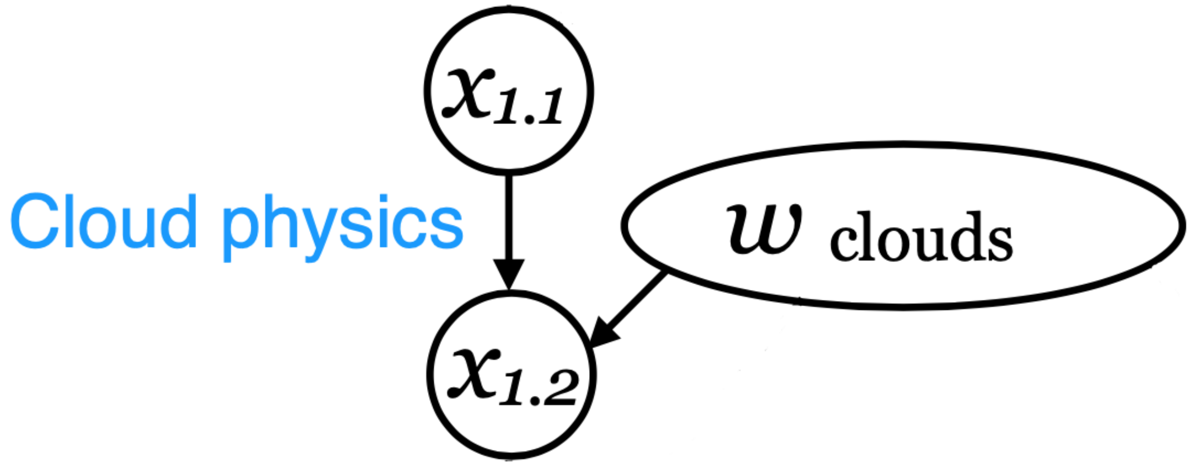
1. Update parameters and state from observations
2. Forecast the next state
3. Update parameters and state from observations
4. Forecast ...

Inside an atmospheric model & data assimilation timestep

One model time-step

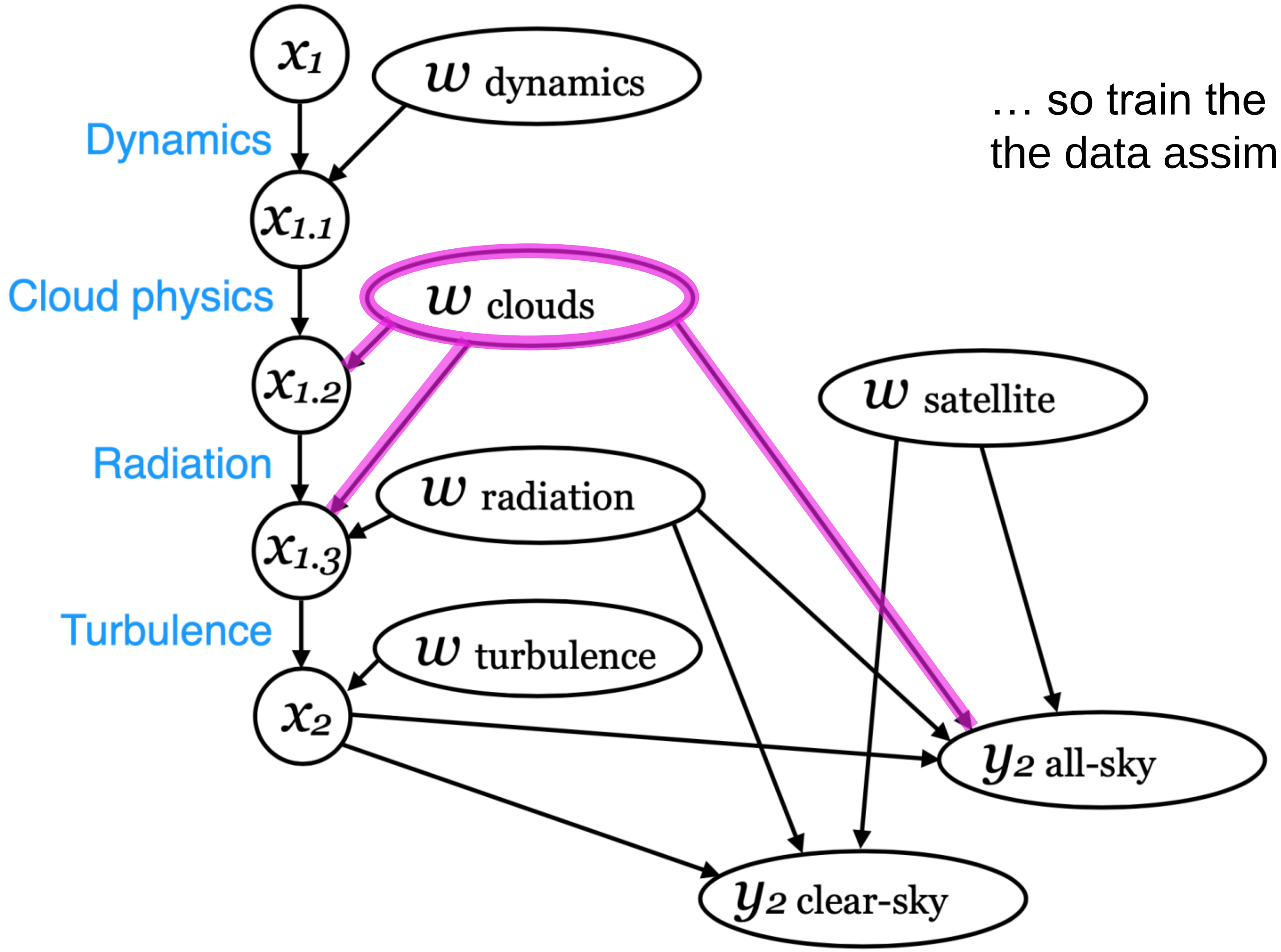


Learning an improved model of cloud physics (ML or DA)



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

Current issues and ideas combining machine learning and DA / physics

A few highlights from a rapidly developing new field

ECMWF

Combine physical and empirical models: Physically constrained ML

```
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

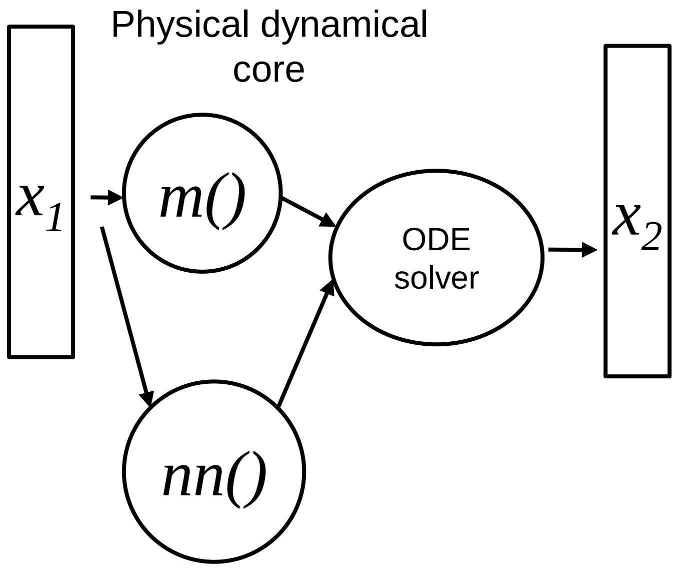
$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - v\frac{\partial^2 u}{\partial x^2} = 0$$

Custom loss function

https://github.com/maziarraissi/PINNs

Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations." arXiv preprint arXiv:1711.10561 (2017)

Hybrid physics – machine learning: "Neural GCM"



Neural network to represent model physics (and correct model error)

Kochkov et al. (2023) Neural General Circulation Models https://doi.org/10.48550/arXiv.2311.07222

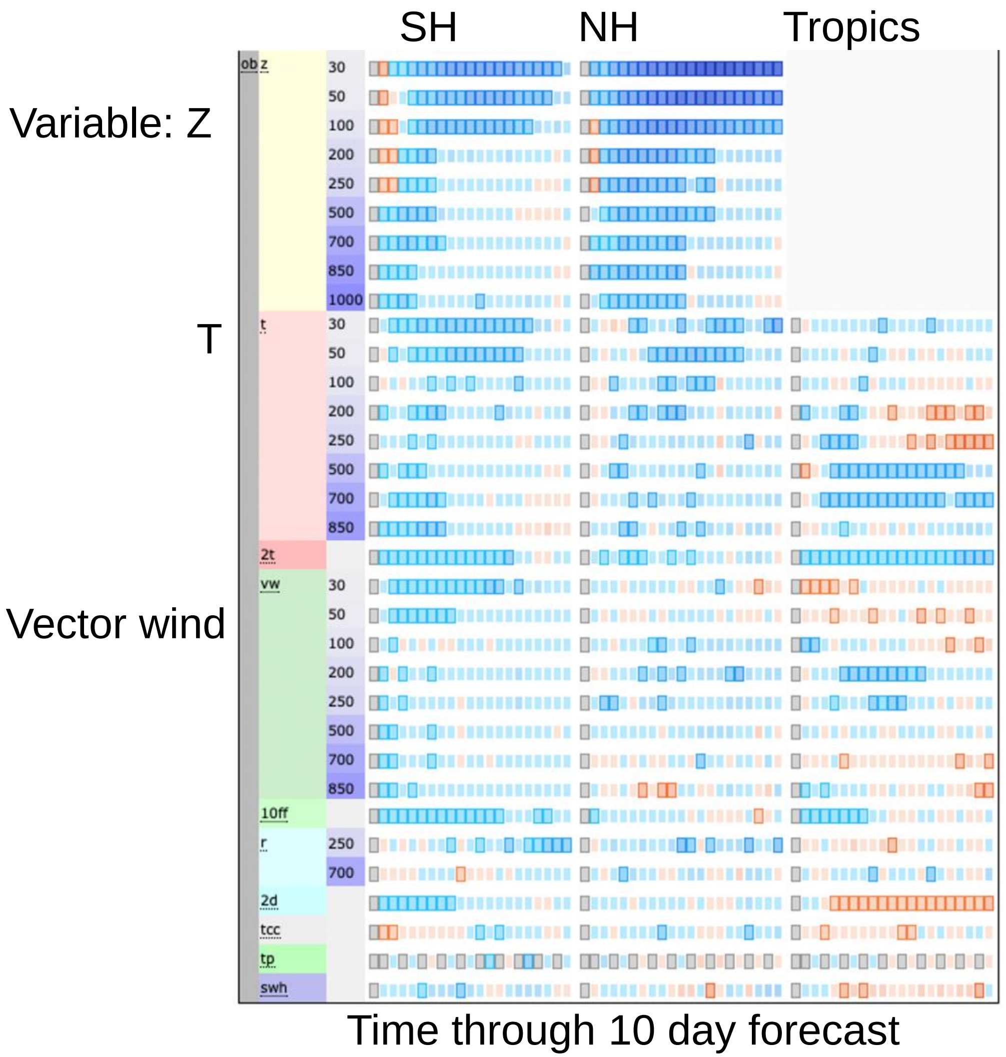
Trained on data assimilation outputs (ERA5)

Hybrid data assimilation and machine learning: train the neural network (forecast model, or bias correction) as part of the data assimilation process

- Simultaneous estimation of the initial conditions, NN parameters and dynamical parameters of a model (e.g. Lorenz '63) using data assimilation (Hsieh and Tang, 2001, https://doi.org/10.1175/1520-0493(2001)129<0818:CNNTID>2.0.CO;2)
- Use iterative cycles of data assimilation followed by neural network training (Brajard et al., 2020, https://doi.org/10.1016/j.jocs.2020.101171)
- In development at ECMWF – train a NN **within 4D-Var –** quasi-geostrophic (QG) model / OOPS
  - "Online model error correction with neural networks in the incremental 4D-Var framework"
  - Alban Farchi, Marcin Chrust, Marc Bocquet, Patrick Laloyaux, Massimo Bonavita (2022, https://doi.org/10.48550/arXiv.2210.13817)
- "Online learning" or sequential learning is a thing in ML too (compared to "train once" approach)
  - e.g. Online sequential Extreme Learning Machine (OS-ELM, Liang et al., 2006) https://doi.org/10.1109/TNN.2006.880583
  - e.g. Forecasting daily streamflow using OSELM (Lima, Cannon, Hsieh, 2016) https://doi.org/10.1016/j.jhydrol.2016.03.017

Real payoff: apply online model bias corrections in the forecast model

Marcin Chrust, Alban Farchi, Patrick Laloyaux, Massimo Bonavita



Score cards (verifications against observation, 3 DEC 2020 – 28 FEB 2021)

Exp: Online NN model error correction (1-hourly) applied in FCLONG (1-hourly) with reduced magnitude:

Control has no online model error correction

Blue = reduced forecast error in experiment compared to control (red = increased)

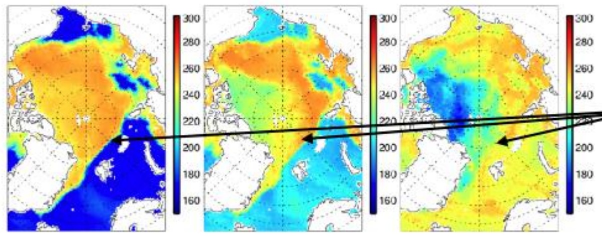
Hybrid data assimilation and machine learning

Sea ice example

ECMWF

A trainable empirical-physical network for sea ice assimilation

AMSR2 observations



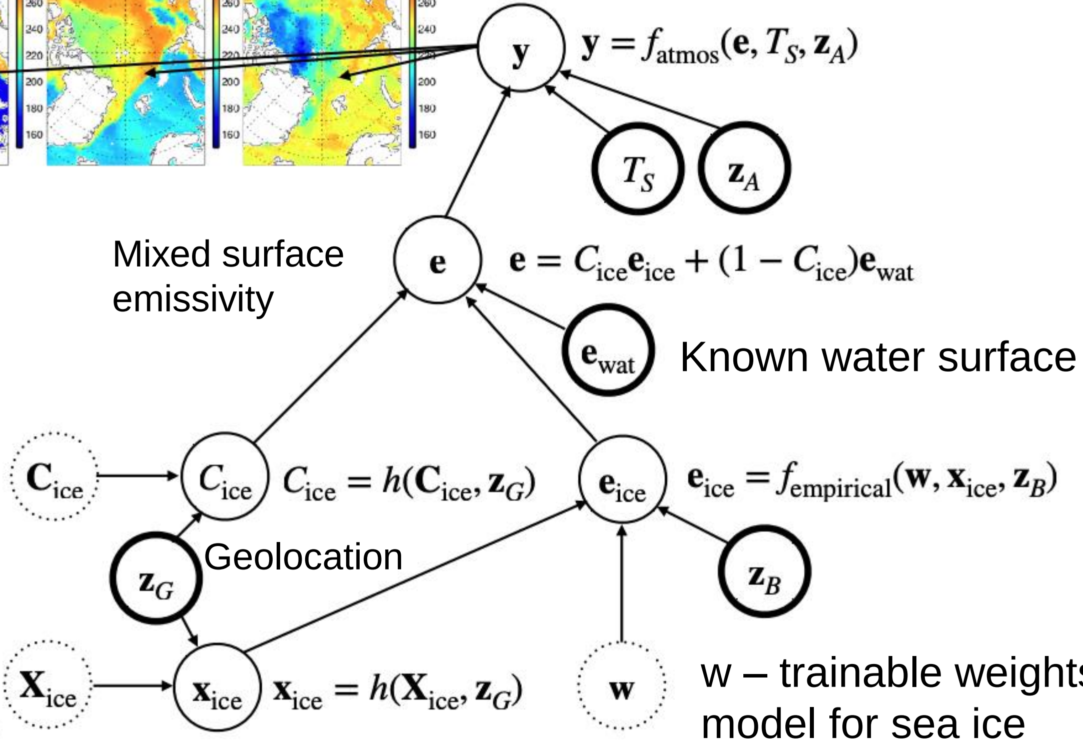
$$J_{\mathrm{obs}} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{(y_{\mathrm{obs},ij} - y_{\mathrm{sim},ij})^2}{r_j^2}$$

Known atmosphere radiative transfer

Fixed parameter

Dependent parameter

Trainable parameter


$\mathbf{y}$
T_S
$\mathbf{z}_A$
$\mathbf{e}$
$\mathbf{e}_{\mathrm{wat}}$
$\mathbf{C}_{\mathrm{ice}}$
C_{ice}
$\mathbf{z}_G$
$\mathbf{X}_{\mathrm{ice}}$
$\mathbf{x}_{\mathrm{ice}}$
$\mathbf{e}_{\mathrm{ice}}$
$\mathbf{w}$
$\mathbf{z}_B$

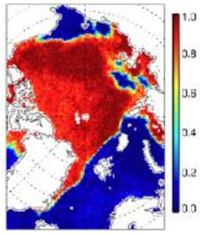
$$\mathbf{y} = f_{\mathrm{atmos}}(\mathbf{e}, T_S, \mathbf{z}_A)$$

Mixed surface emissivity

$$\mathbf{e} = C_{\mathrm{ice}}\mathbf{e}_{\mathrm{ice}} + (1 - C_{\mathrm{ice}})\mathbf{e}_{\mathrm{wat}}$$

Known water surface emissivity

Map of sea ice fraction to be estimated

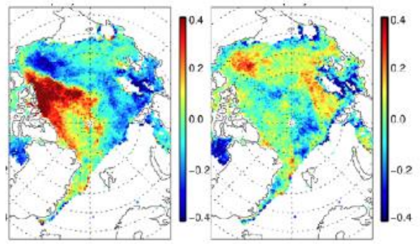


$$C_{\mathrm{ice}} = h(\mathbf{C}_{\mathrm{ice}}, \mathbf{z}_G)$$

$$\mathbf{e}_{\mathrm{ice}} = f_{\mathrm{empirical}}(\mathbf{w}, \mathbf{x}_{\mathrm{ice}}, \mathbf{z}_B)$$

Geolocation

Maps of empirical parameters representing **unknown** sea ice state including microstructure

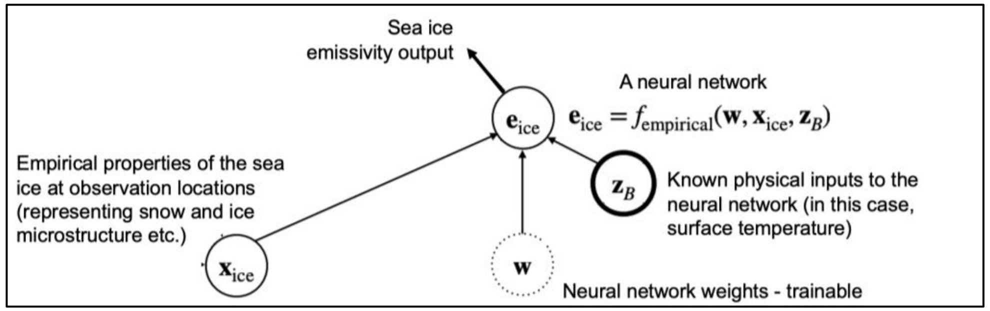


$$\mathbf{x}_{\mathrm{ice}} = h(\mathbf{X}_{\mathrm{ice}}, \mathbf{z}_G)$$

w – trainable weights of NN model for sea ice

h() Interpolation operator: map to observation location in time and space

Built in Python and Tensorflow


Sea ice emissivity output
A neural network
$\mathbf{e}_{\text{ice}} = f_{\text{empirical}}(\mathbf{w}, \mathbf{x}_{\text{ice}}, \mathbf{z}_B)$
Empirical properties of the sea ice at observation locations (representing snow and ice microstructure etc.)
Known physical inputs to the neural network (in this case, surface temperature)
Neural network weights - trainable
$\mathbf{e}_{\text{ice}}$
$\mathbf{z}_B$
$\mathbf{x}_{\text{ice}}$
$\mathbf{w}$

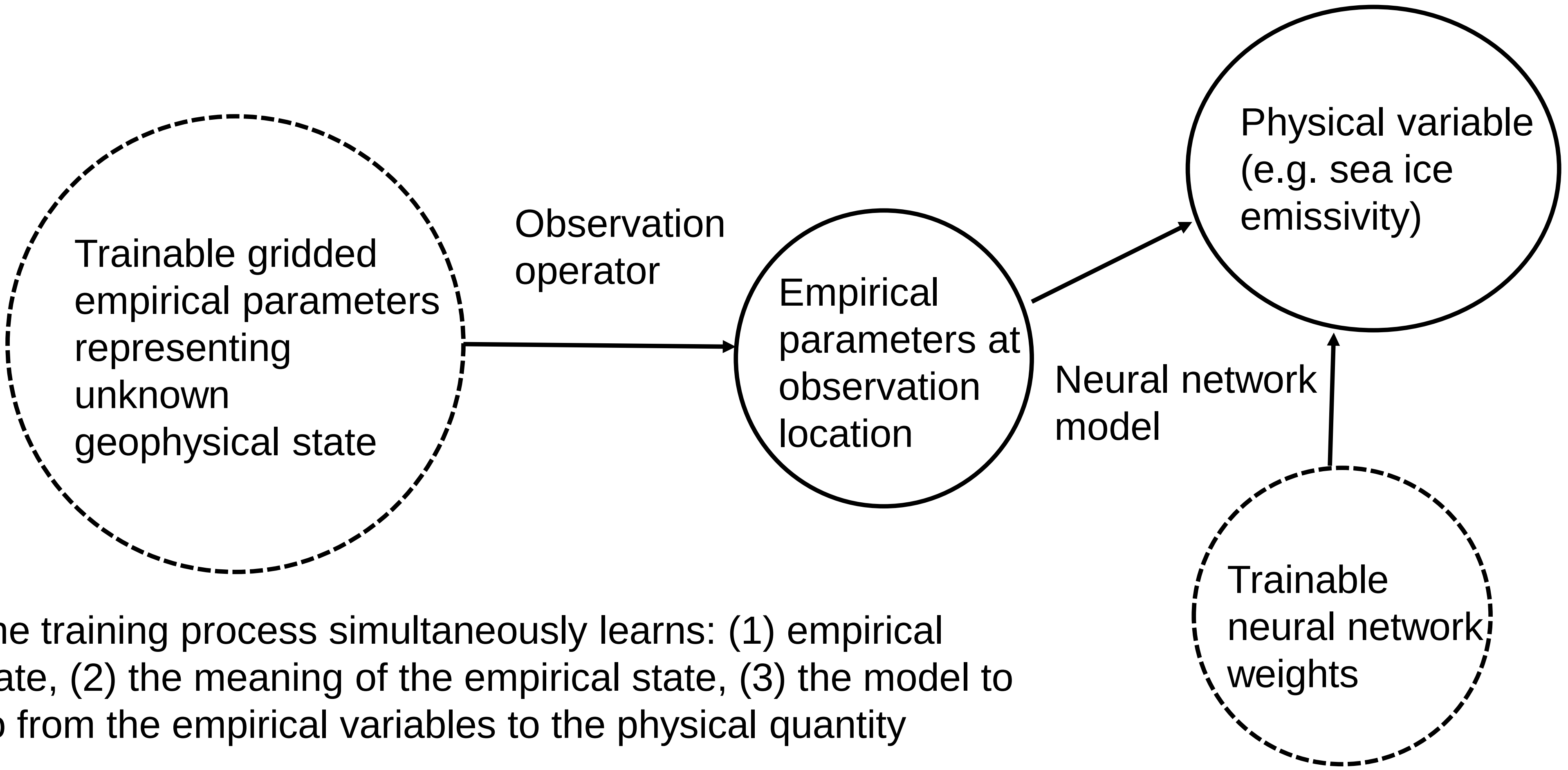
```python
class SeaiceEmis(tf.keras.layers.Layer):
    """
    Linear dense layer representing the sea ice emissivity empirical model.

    The sea ice loss applies to just the first mean emissivity (e.g. channel 10v); it's a single number as required.
    """
    def __init__(self, channels=10, bg_error=0.1, nobs=1, background=0.93):
        super(SeaiceEmis, self).__init__()
        self.dense_1 = tf.keras.layers.Dense(channels,activation='linear',bias_initializer=tf.keras.initializers.Constant(background))
        self.bg_error = bg_error
        self.background = background
        self.nobs = nobs
    def call(self, tsfc, ice_properties):
        inputs = tf.concat([tf.reshape(tsfc,(-1,1)),ice_properties],1)
        ice_emis = self.dense_1(inputs)
        emis_loss = tf.math.squared_difference((self.weights[1])[0],self.background)/tf.square(self.bg_error)/self.nobs
        self.add_loss(emis_loss)
        self.add_metric(emis_loss,name='emis_loss',aggregation='mean')
        return ice_emis
```

A standard dense neural network layer with linear activations

Custom loss functions to regularise / constrain the solution

https://github.com/ecmwf-projects/empirical-state-learning-seaice-emissivity-model/blob/master/seaice_layers.py

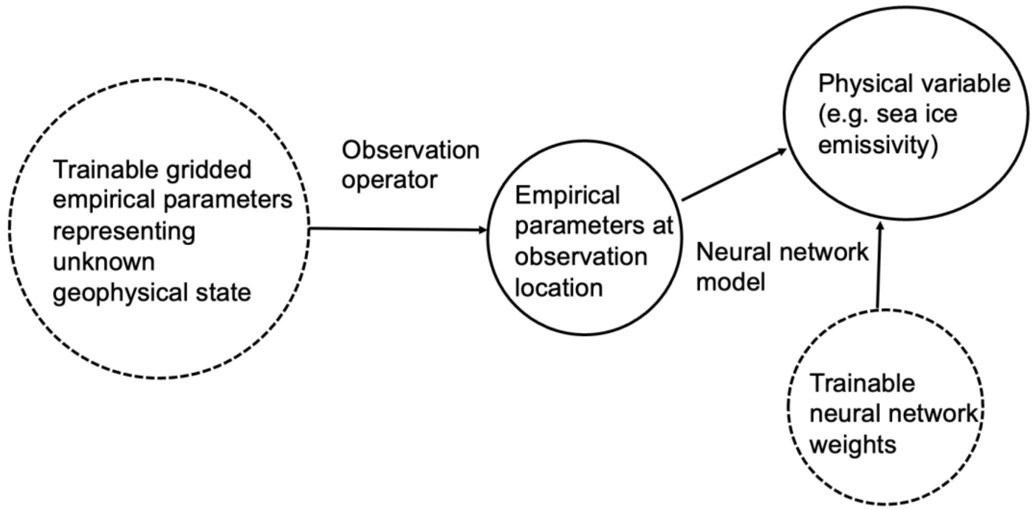
Fundamentals of this “Empirical state” method



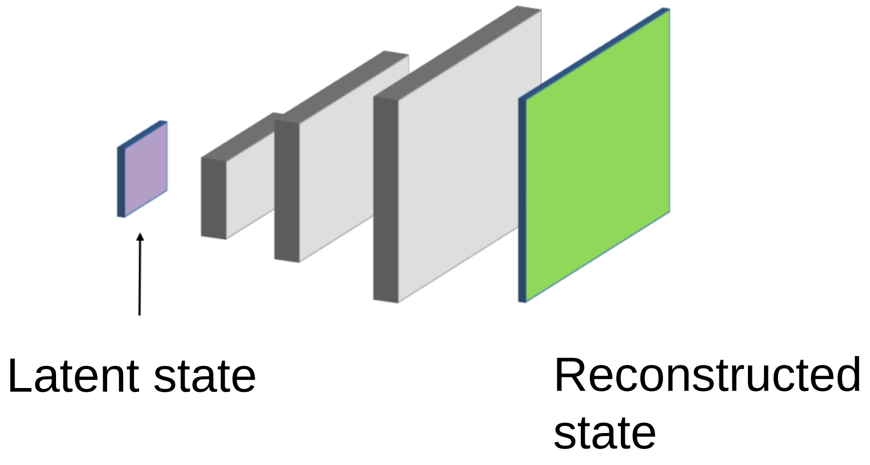
Trainable gridded empirical parameters representing unknown geophysical state
Observation operator
Empirical parameters at observation location
Neural network model
Physical variable (e.g. sea ice emissivity)
Trainable neural network weights

The training process simultaneously learns: (1) empirical state, (2) the meaning of the empirical state, (3) the model to go from the empirical variables to the physical quantity

Fundamentals of this "Empirical state" method

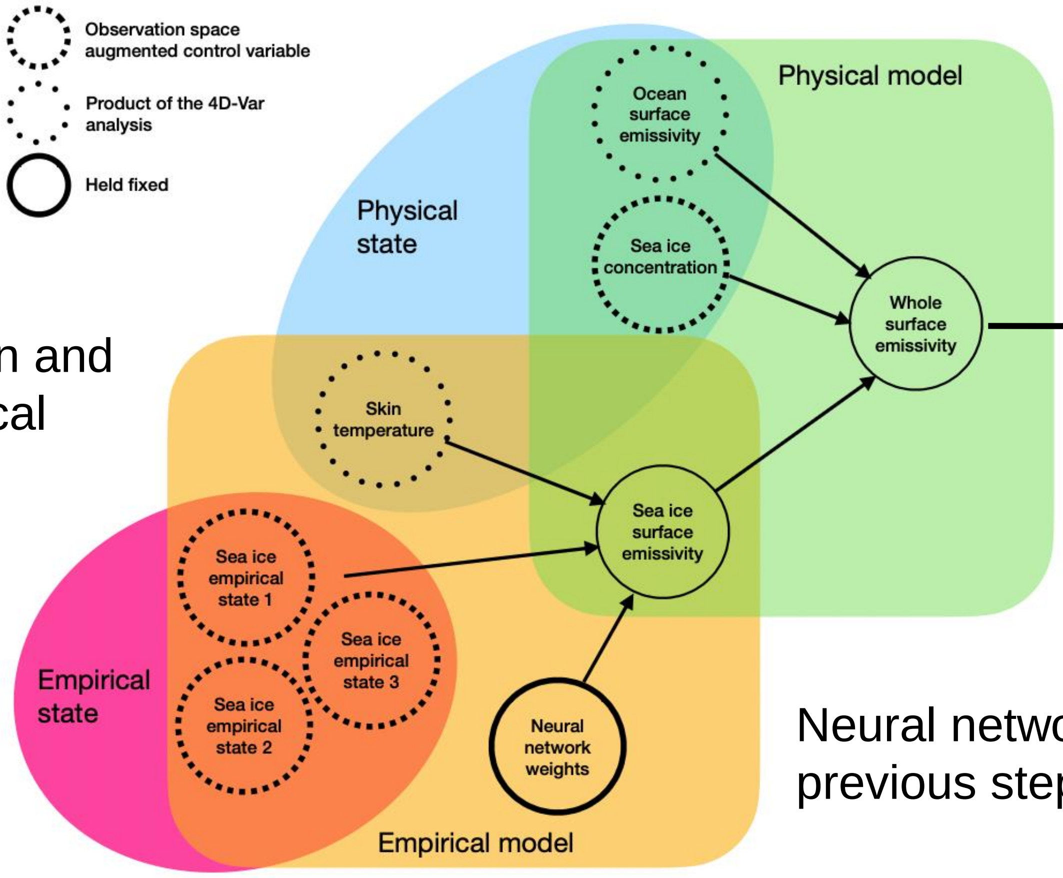

Trainable gridded empirical parameters representing unknown geophysical state
Observation operator
Empirical parameters at observation location
Physical variable (e.g. sea ice emissivity)
Neural network model
Trainable neural network weights

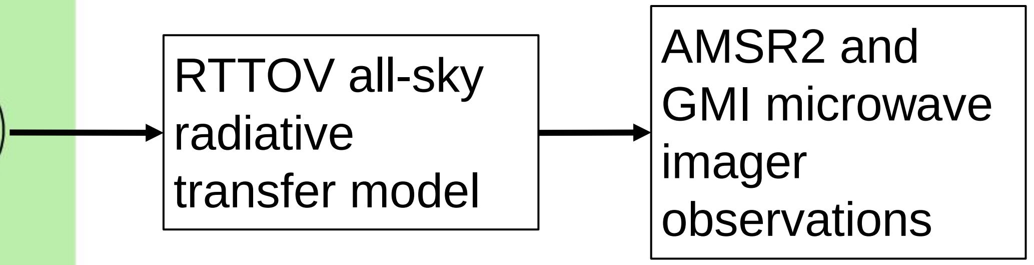
Think back to the decoder in generative ML



Empirical sea ice emissivity model used to retrieve sea ice concentration in atmospheric 4D-Var and to allow radiance assimilation over sea ice

Sea ice concentration and three empirical parameters retrieved at observation locations

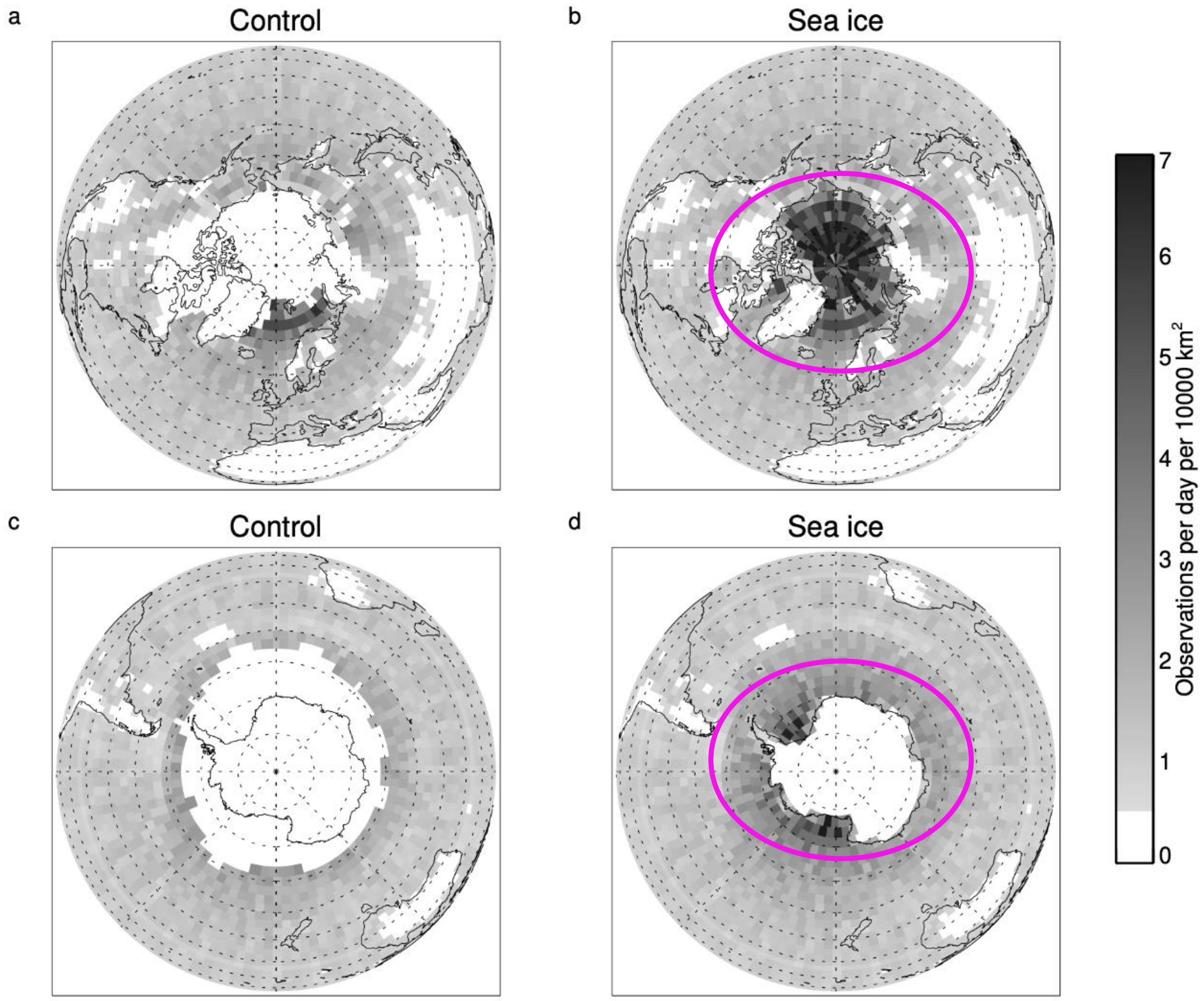

Observation space augmented control variable
Product of the 4D-Var analysis
Held fixed
Physical state
Physical model
Ocean surface emissivity
Sea ice concentration
Whole surface emissivity
Skin temperature
Sea ice surface emissivity
Sea ice empirical state 1
Sea ice empirical state 3
Sea ice empirical state 2
Empirical state
Neural network weights
Empirical model


RTTOV all-sky radiative transfer model
AMSR2 and GMI microwave imager observations

Atmosphere is also updated in 4D-Var, also with the benefit of all other observations, and can be improved by observations over sea ice

Neural network weights trained in previous step are now held fixed

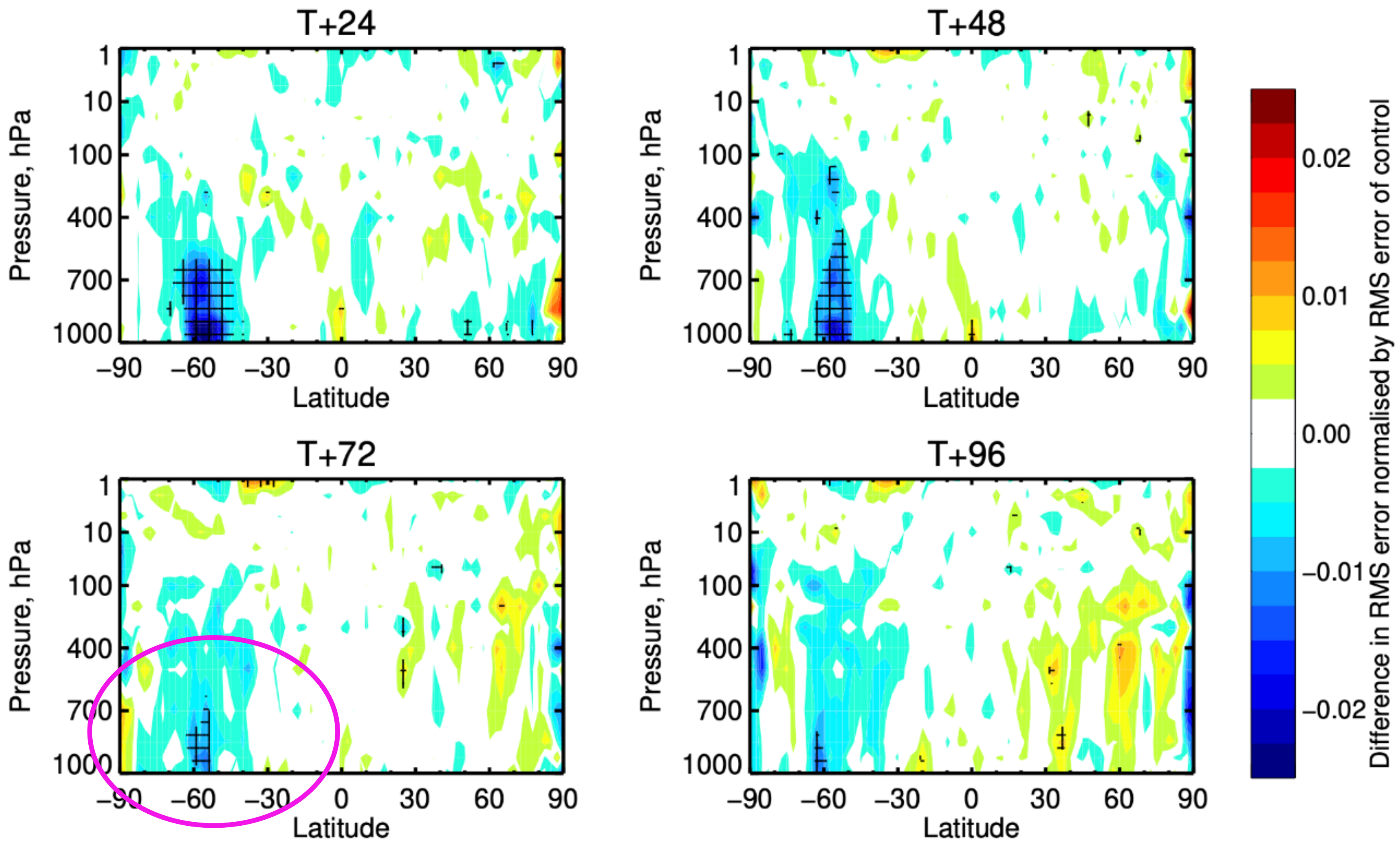
Number of AMSR2 observations added



Up to around 7 observations per day per 10,000 km^2 have been added over sea ice regions

Forecast impact - temperature
(blue = reduced error; +++ = statistical significance)

Improved temperature forecasts out to 72 hours in the Southern Ocean

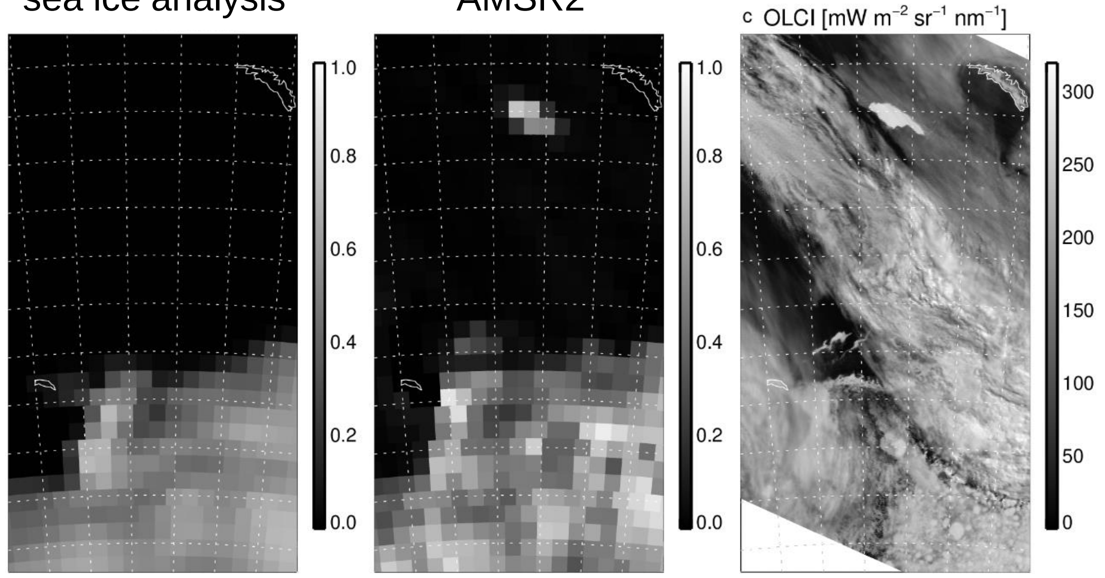


Giant iceberg case studies: accurate retrievals to superob scale (40 km)

Existing OCEAN5 sea ice analysis

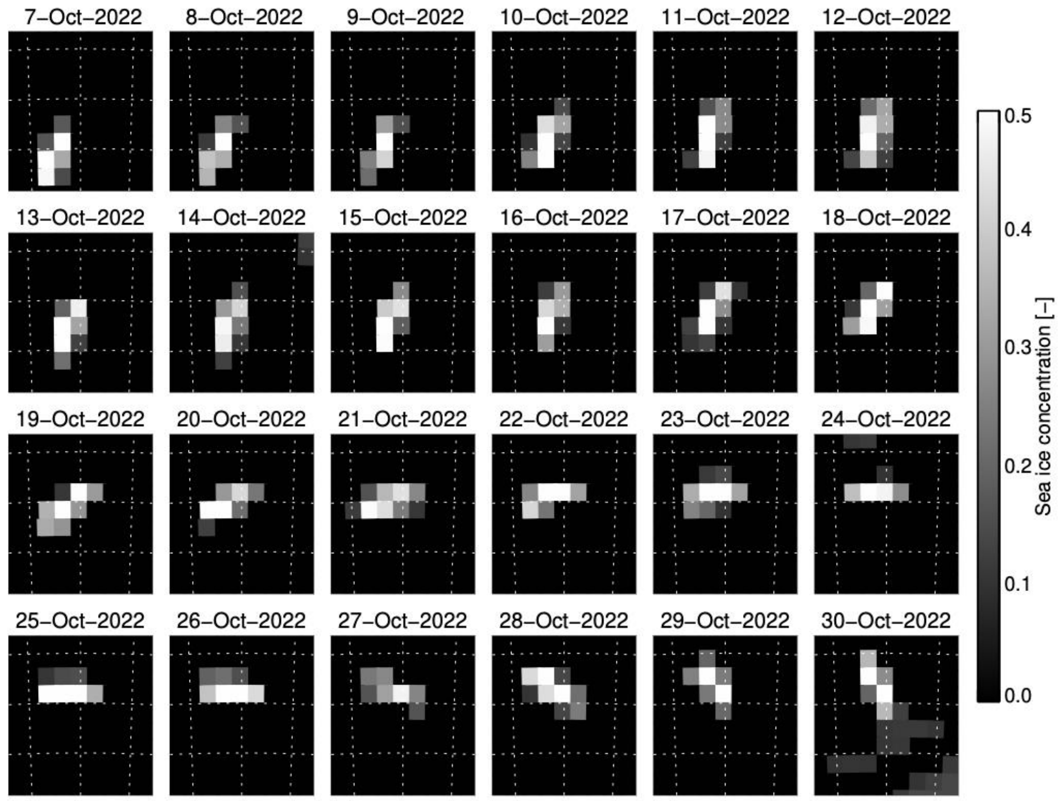
New analysis from AMSR2

OLCI visible image



12 UTC 4th Dec 2020 – A68A (100 km by 60 km)

Iceberg approaching island of South Georgia. Copernicus sentinel data 2020



October 2022 – A76A (135 km by 26 km)

Iceberg moving slowly N and rotating 180º in Drake Passage

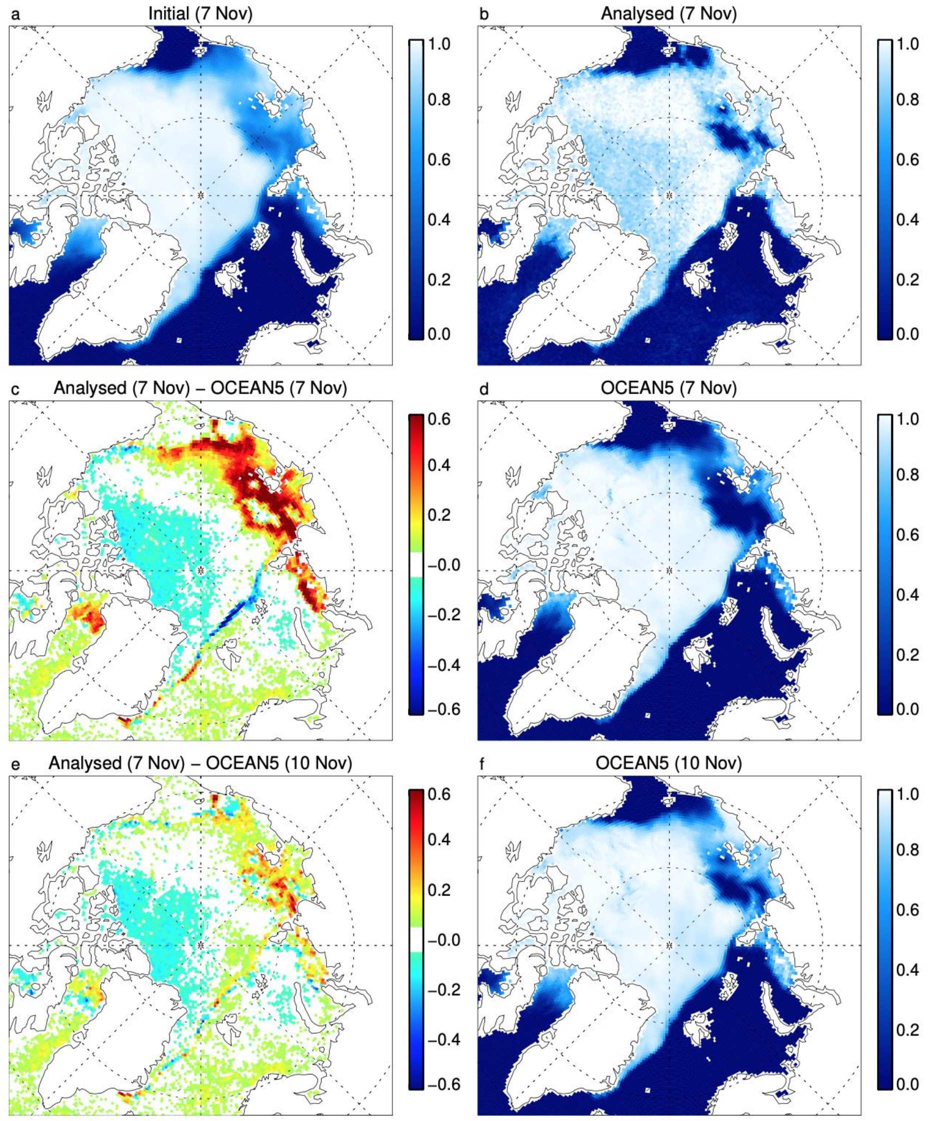
Map grid resolution: 1º latitude, 2º longitude.

Pixel (AMSR2 superob) resolution ~40 km

Sea ice fraction retrieval: rapid freezing 7th Nov 2020

Sea ice concentration

Difference in sea ice concentration



New retrieval from AMSR2

Existing ECMWF sea ice analysis

Existing analysis three days later

Hybrid physical-empirical networks - sea ice example

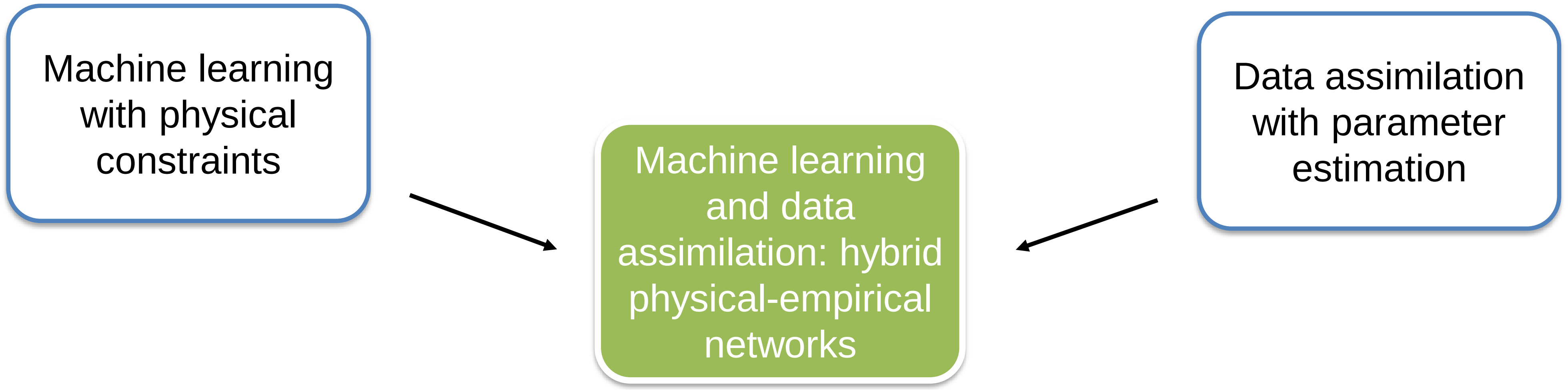
- Sea ice concentration and empirical state estimation will be included in cycle 49r1 of the IFS
  - Model for sea ice emissivity is the simple neural network trained within the hybrid-empirical physical network (and held fixed for now)
  - Operational implementation autumn 2024 – one of the first machine-learned components of the operational IFS
    - Maintainability? Retraining?

- Preprints

Geer (2023) Simultaneous inference of sea ice state and surface emissivity model using machine learning and data assimilation https://doi.org/10.22541/essoar.169945325.51725282/v1

Geer (2024) Joint estimation of sea ice and atmospheric state from microwave imagers in operational weather forecasting https://doi.org/10.22541/essoar.170431213.35796940/v1

Summary: generating new empirical models using ML and DA


Machine learning with physical constraints
Machine learning and data assimilation: hybrid physical-empirical networks
Data assimilation with parameter estimation

- Typical machine learning and variational data assimilation are similar implementations of Bayes’ theorem
- Including known physics into a trainable network is a way of adding prior information in a Bayesian sense
- Existing data assimilation approaches can be very helpful in machine learning:
  - Physically-based loss functions
  - Physically-based observation (label) and background (feature) errors
  - Observation operators to map from grid to irregular and transformed observation space (e.g. satellite radiances)
- Data assimilation frameworks (e.g. weather forecasting) are evolving to be able to train and update empirical models (e.g. neural networks) as part of routine data assimilation activities
  - E.g. model error correction: don’t throw away the physical model – improve it!

ECMWF