Hybrid DA-ML methods (1)

Alan Geer

European Centre for Medium-range Weather Forecasts

alan.geer@ecmwf.int

ECMWF data assimilation and machine learning training course, 19th March 2026

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

ECMWF

The advent of data-driven forecasting

ECMWF

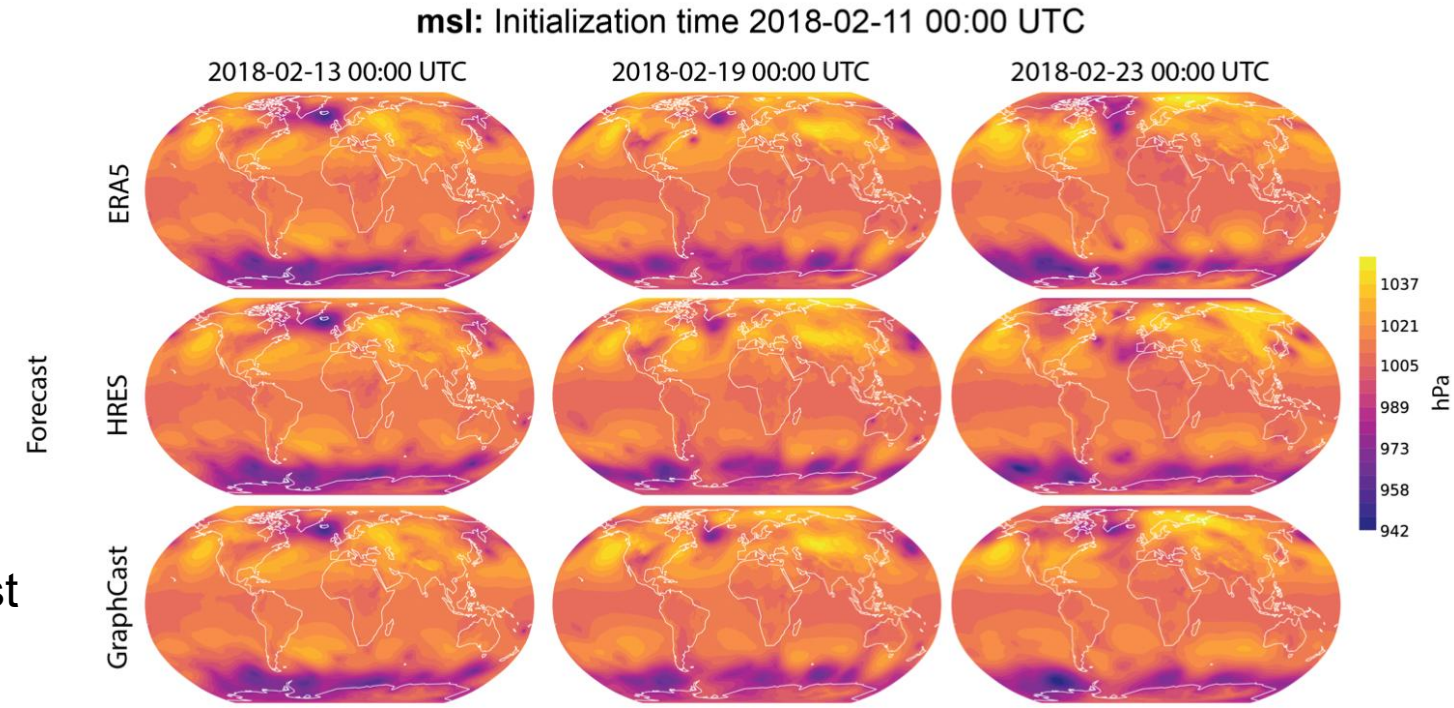
Recent history: The advent of data-driven forecasting

- Keisler, R. (2022). Forecasting global weather with graph neural networks. arXiv preprint arXiv:2202.07575.
- 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)



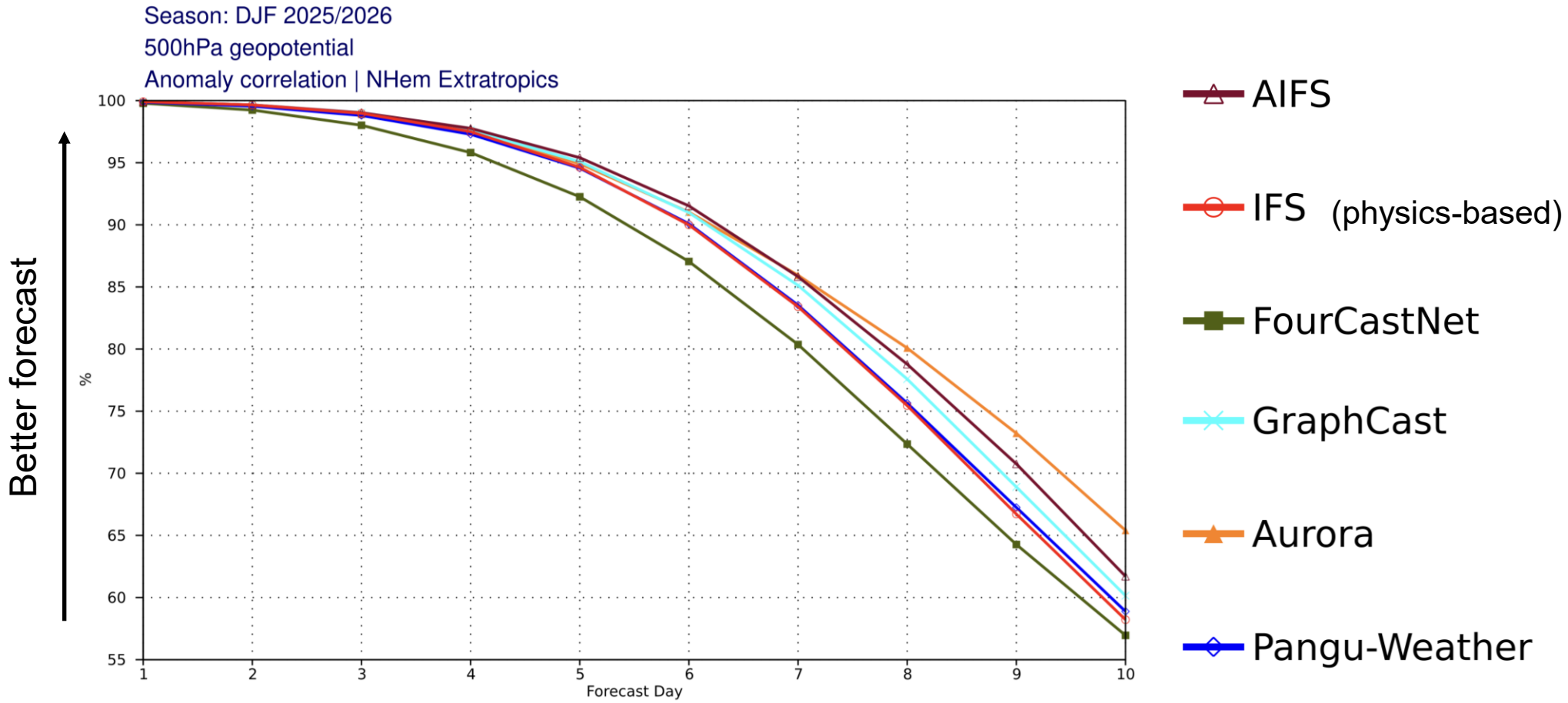
Run time

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



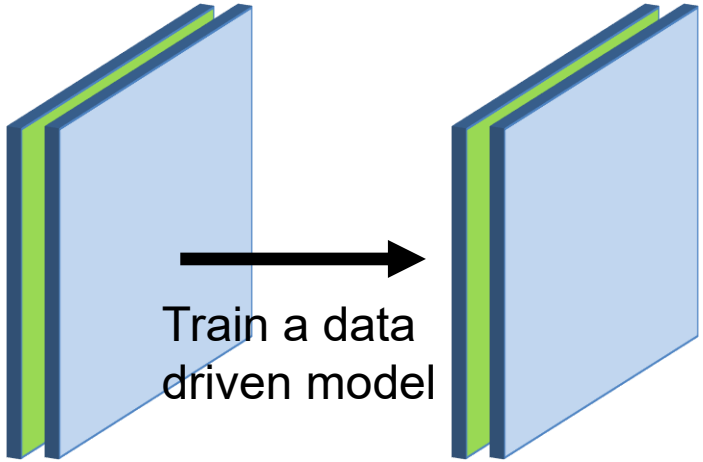
https://charts.ecmwf.int/catalogue/packages/opencharts/products/plwww_3m_fc_aimodels_wp_mean

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

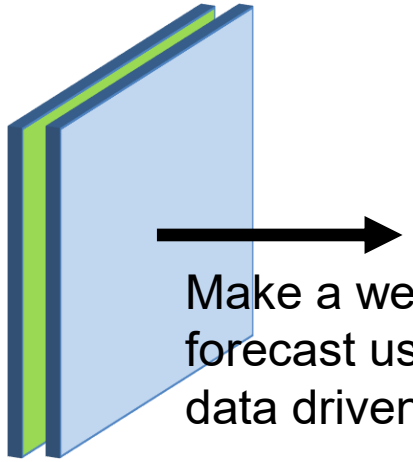
If AI-based forecasting outperforms physical models...



ERA5 analyses

00 UTC 1st Sep 1985

06 UTC 1st Sep 1985



ECMWF operational analyses

12 UTC 22nd Mar 2024

- 1) This great result is >75% due to physical data assimilation!
  - Training data is ERA5 and ECMWF operational analysis
  - Initial conditions are the ECMWF operational analysis.

- 2) Medium range forecasting is possible using lower dimensionality than we thought:

	Horizontal	Vertical	Timestep
IFS	8-9 km	137 levels	7.5 min
AIFS	36 km	13 levels	6 hour

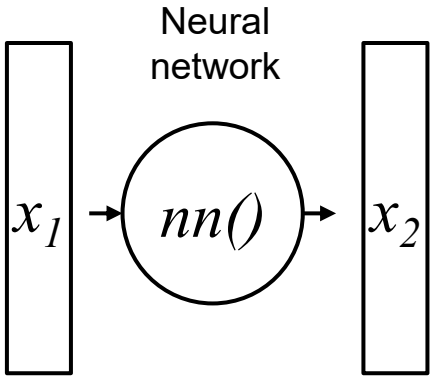
  - Backing up older results eg https://doi.org/10.1002/qj.613
  - Machine learning creates an optimised statistical representation of the atmosphere: “latent space”

- 3) The physical model has significant errors and needs to be improved using observations: **hybrid methods**

ML concepts

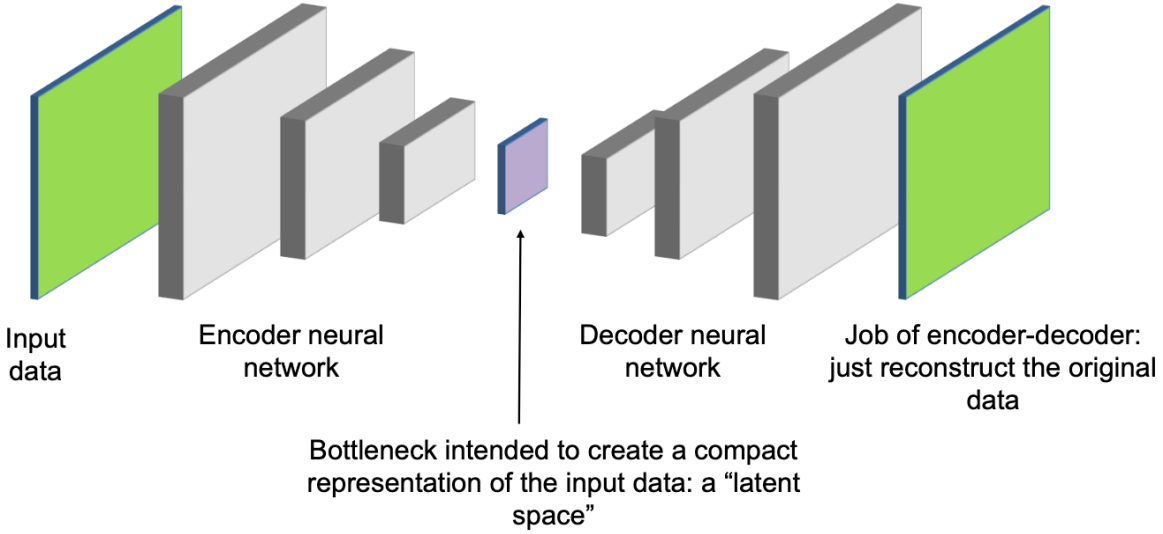
ECMWF

Types of ML – supervised learning



Supervised learning:

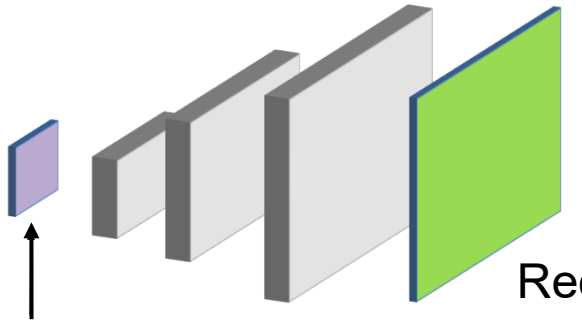
- ML as a "universal function approximator" (Hornik, 1991)
- Both inputs x1 and outputs x2 need to be provided as training data
- An "emulator" / "surrogate" / "empirical model"



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



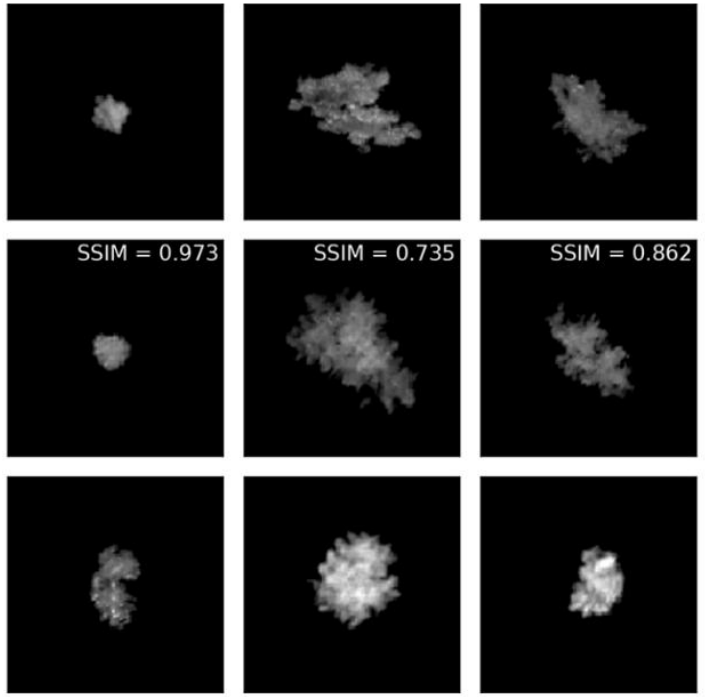
Reconstruction

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):

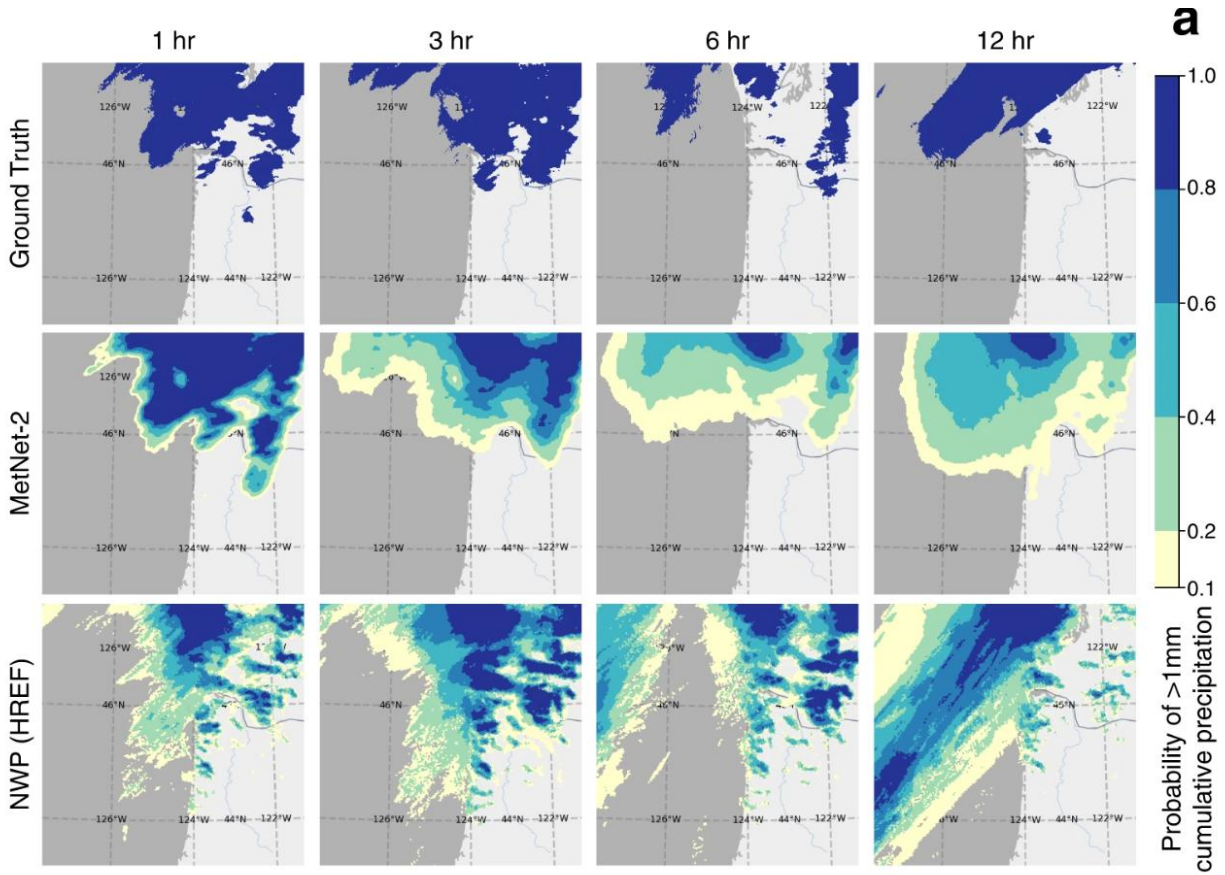
- Generator (~decoder): make an image
- Discriminator (~encoder): given an image, tell if it is real or fake -> drives the loss function

End-to-end methods: do we need physical or hybrid components at all?

ECMWF

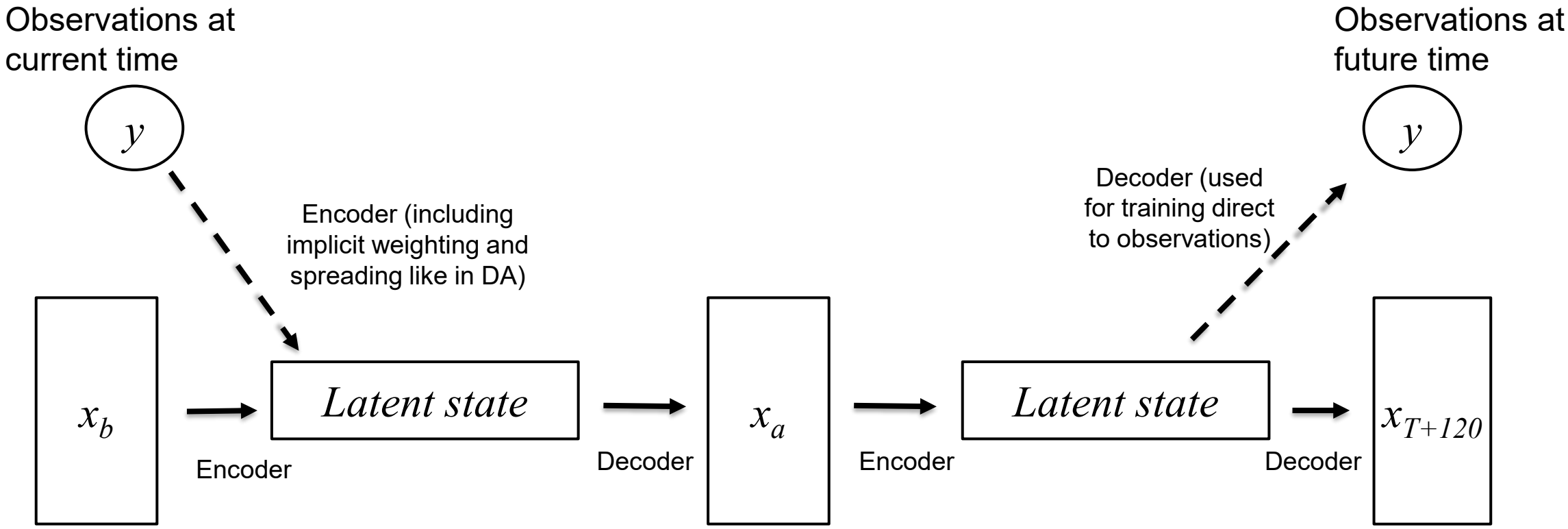
Can we forecast directly from (and to) observations?

- Google MetNet-2 was trained to forecast from/to precipitation “observations” from gauge and radar (although it does use some NWP information for initial conditions)
- Aardvark Weather replaces data assimilation
  - DA emulation is trained on conventional and satellite observations with ERA5 as a target
  - Aardvark Z500 forecasts are about 1 day behind ECMWF physical forecasts
  - Allen et al., 2025, ”End-to-end data-driven weather prediction”, https://doi.org/10.1038/s41586-025-08897-0
- AI-DOP at ECMWF goes from observation to observation with no input from physical NWP
  - https://doi.org/10.48550/arXiv.2412.15687



MetNet-2: CC BY 4.0 reproduction from Espeholt et al. (2022, “Deep learning for twelve hour precipitation forecasts) https://doi.org/10.1038/s41467-022-32483-x)

Weather forecasting completely by ML



Background

Geophysical fields, gridded, at a synoptic time

Analysis

Weather forecast

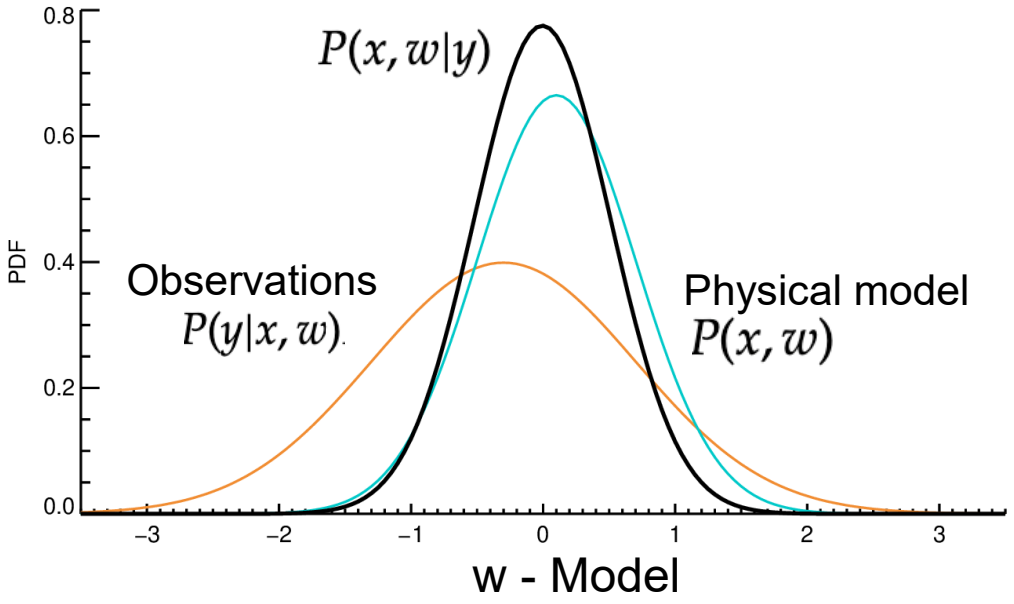
Possibly iterated in forecast mode

Gridded model fields in unobserved variables would still need a physically-based analysis for training

Machine learning vs. DA

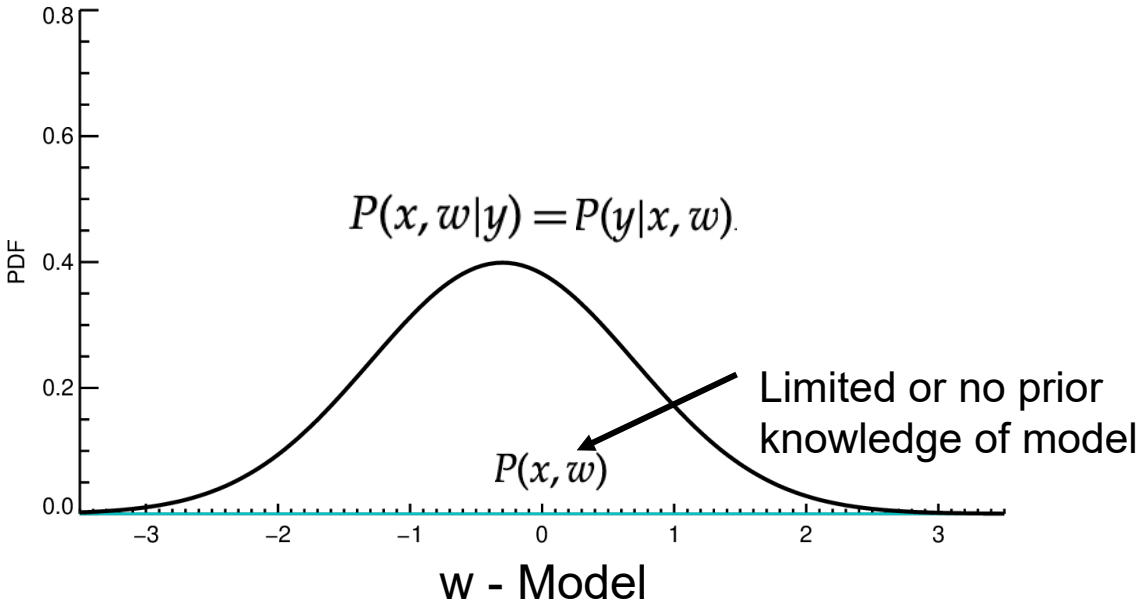
$$P(x, w|y) = \frac{P(y|x, w)P(x, w)}{P(y)}$$

Data assimilation with model parameters w



Our knowledge of the model is better when we combine physics and observations

Machine learning



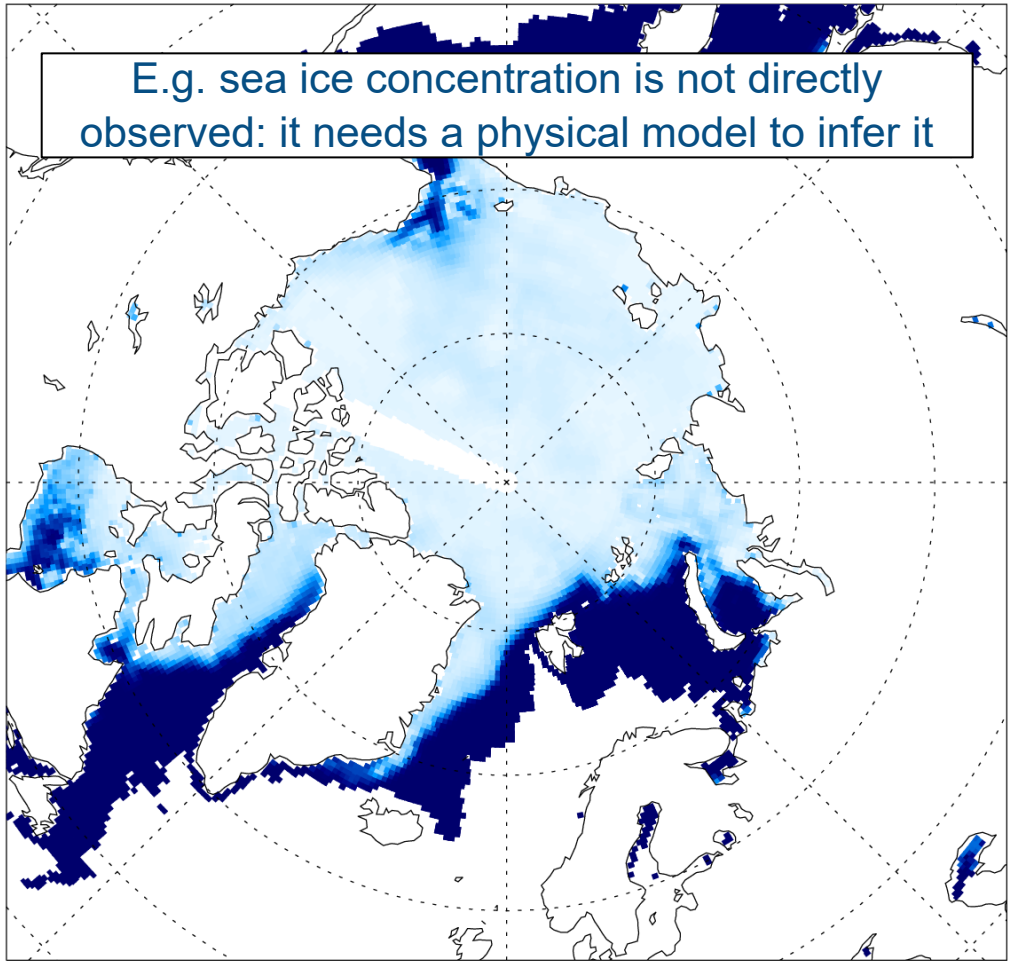
Without a prior, our knowledge of the model is no better than what we know from observations

How to improve this, given this?

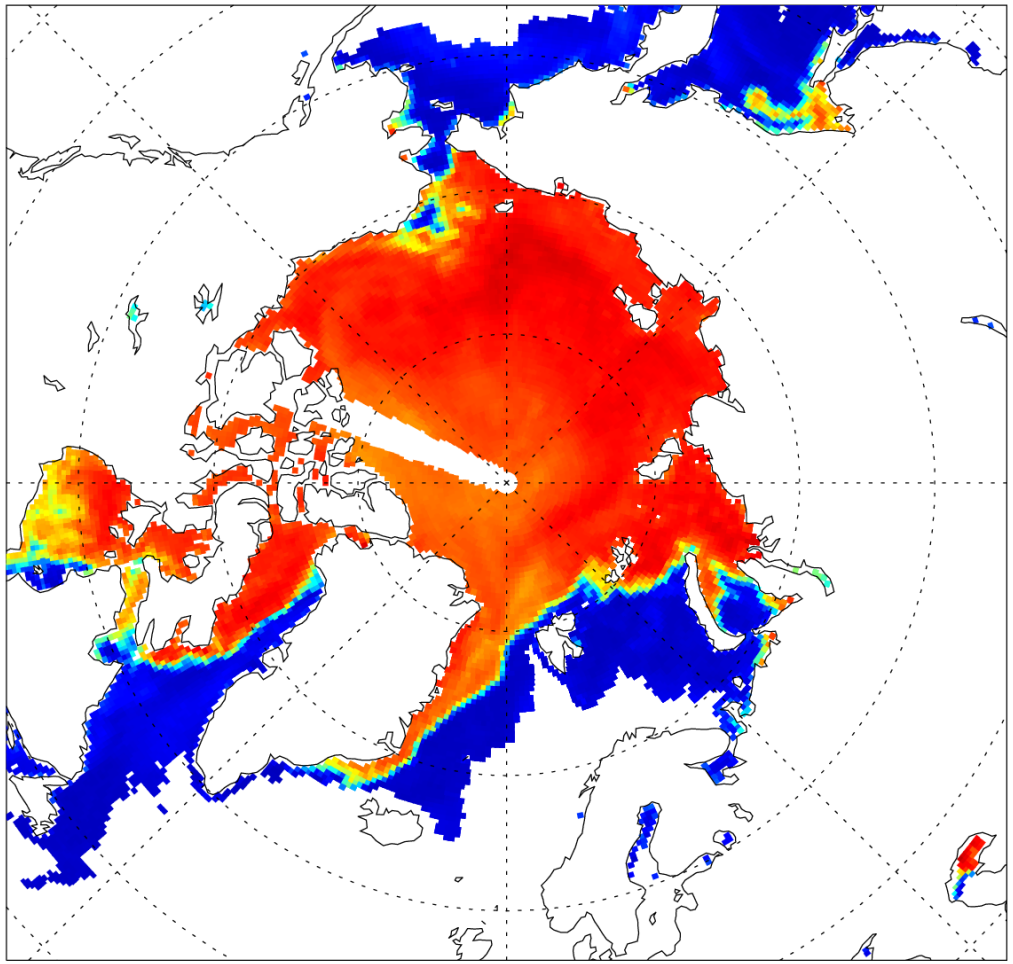
NEMO/SI3 background, 00Z 10th Dec 2022

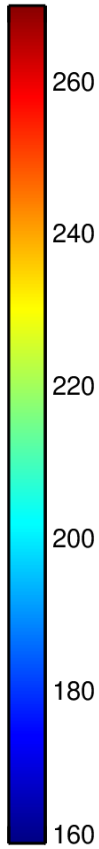
AMSR2 10 GHz observation, 00Z 10th Dec 2022

E.g. sea ice concentration is not directly observed: it needs a physical model to infer it



Sea ice concentration





Brightness temperature [K]

A simple machine learning example: training an observation operator

Tools: Python, Keras, Tensorflow

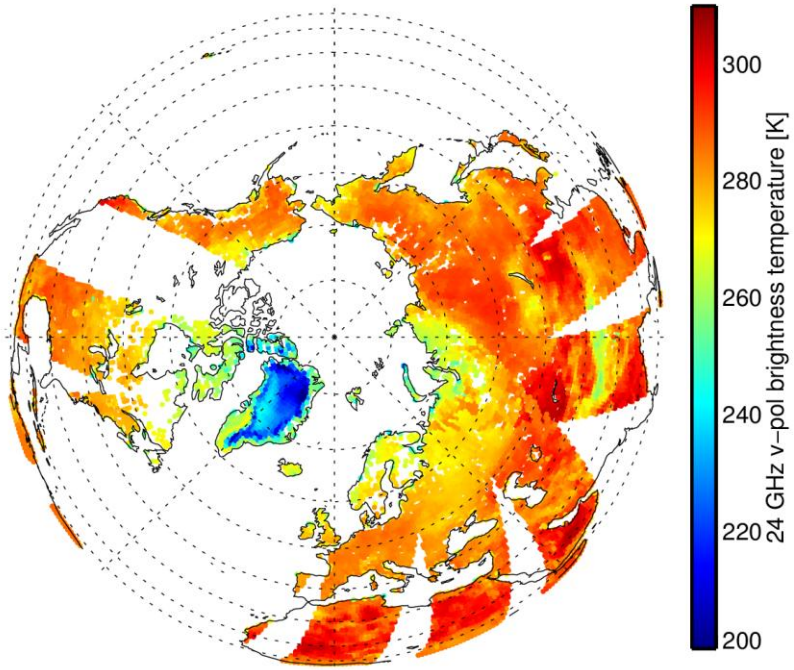
(In this evening's practical led by Alban, we will learn to emulate a forecast model using Pytorch)

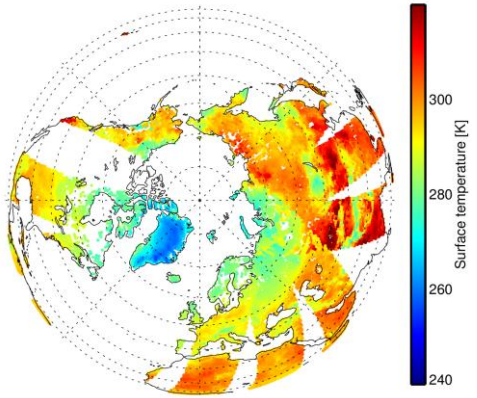
ECMWF

Datasets

y = labels

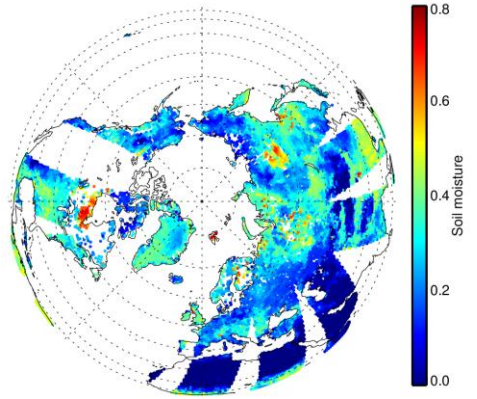
AMSR2 24GHz v-pol observations



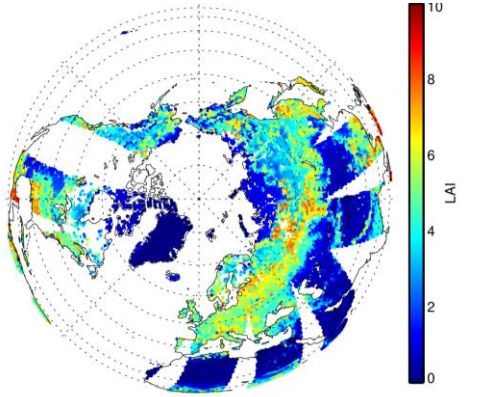


Skin temperature

x = features



Soil moisture

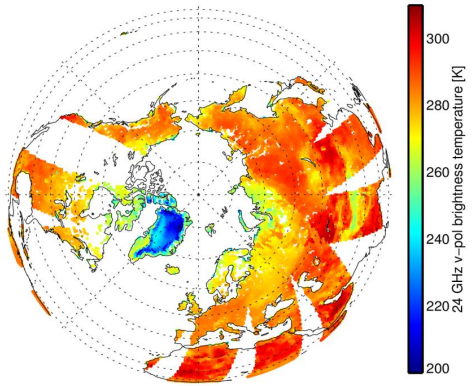


Leaf area index

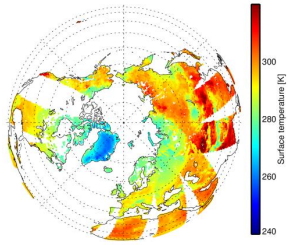
10 possible predictors for the brightness temperature from the IFS 12h forecast

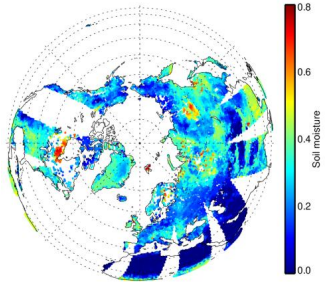
+ orography, snow depth, snow density, integrated water vapour, cloud, rain and snow water contents

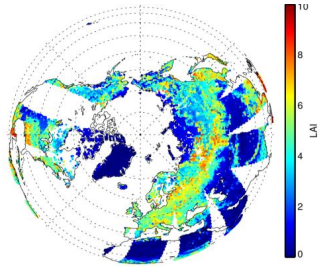
y =
labels



Task of ML:
find y=h(x,w)







x =
features

w =
neural network
weights

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

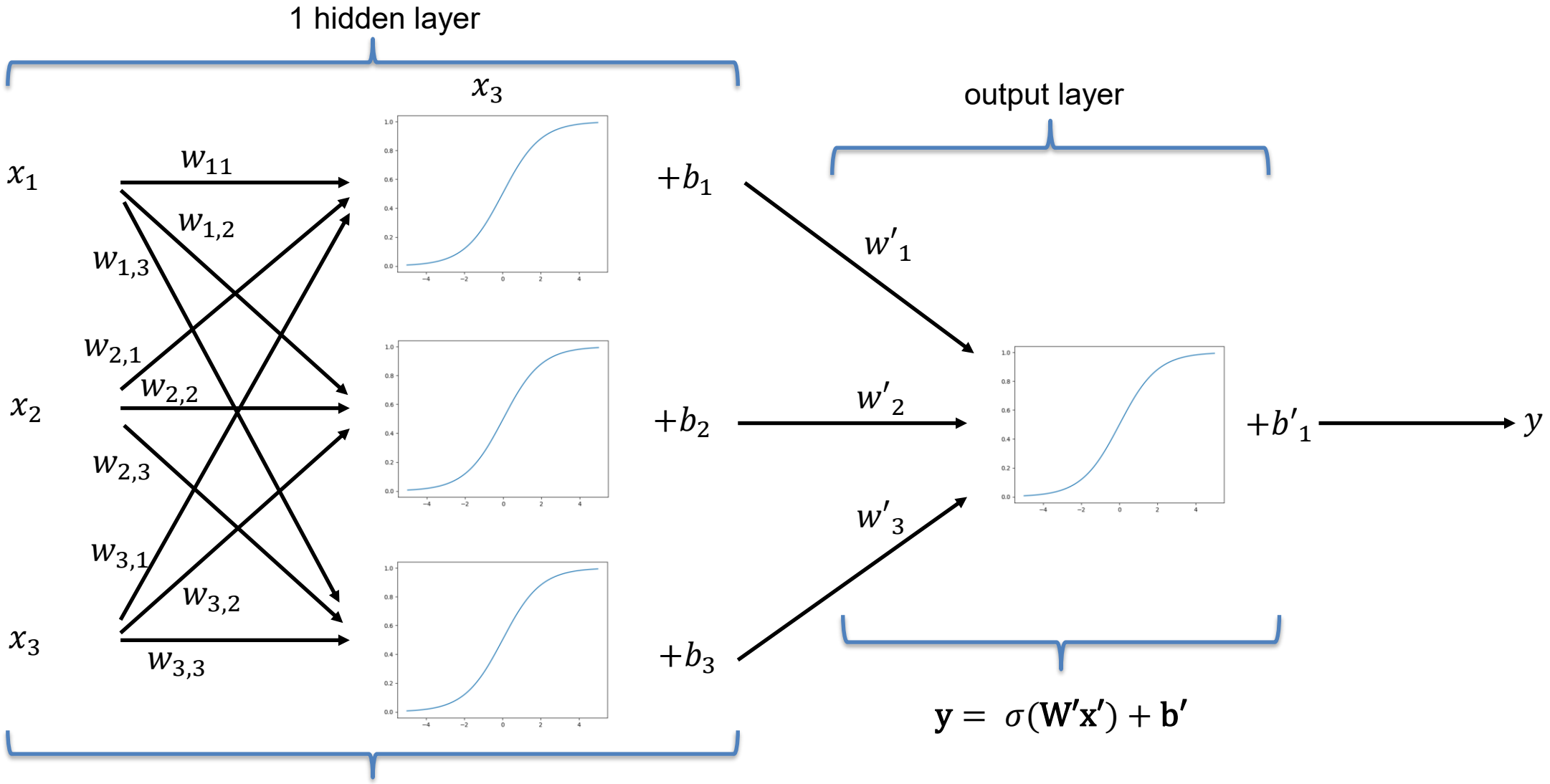
Normalise ‘features’ x to roughly -1 to +1

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

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
_________________________________________________________________
```

Dense neural network - example



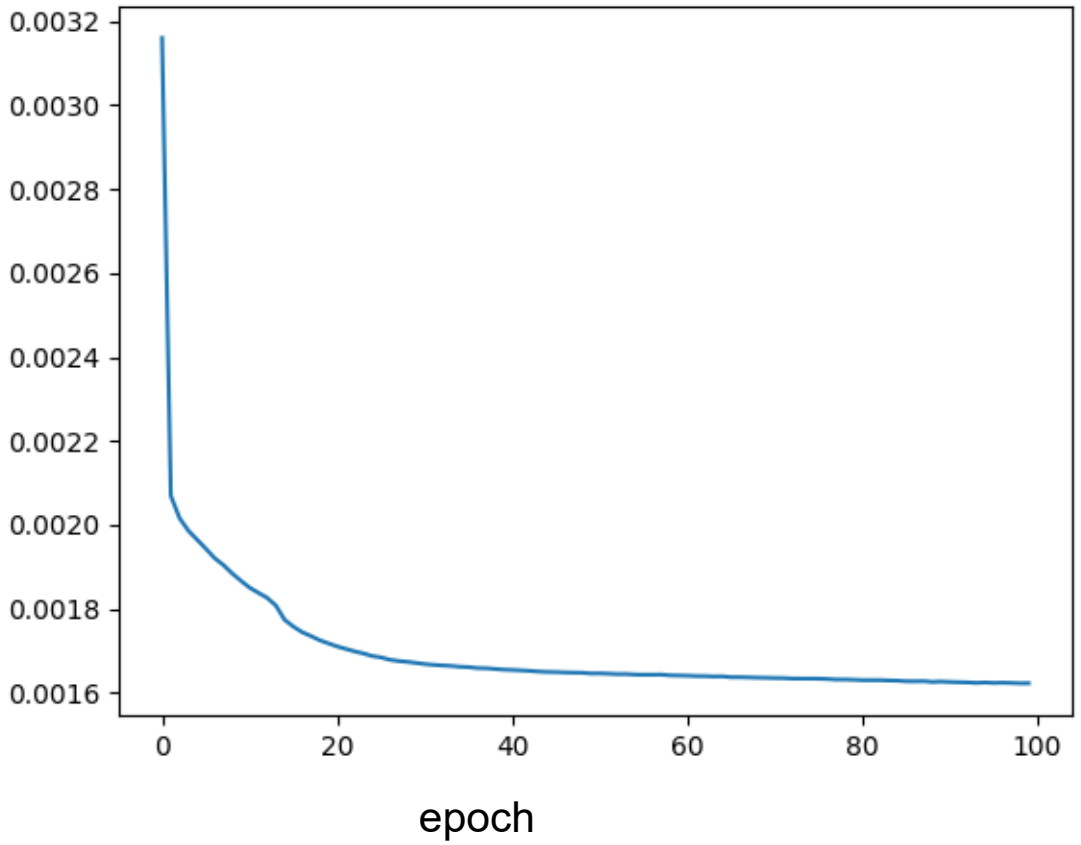
Train it (about 25 minutes on a linux workstation)

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.

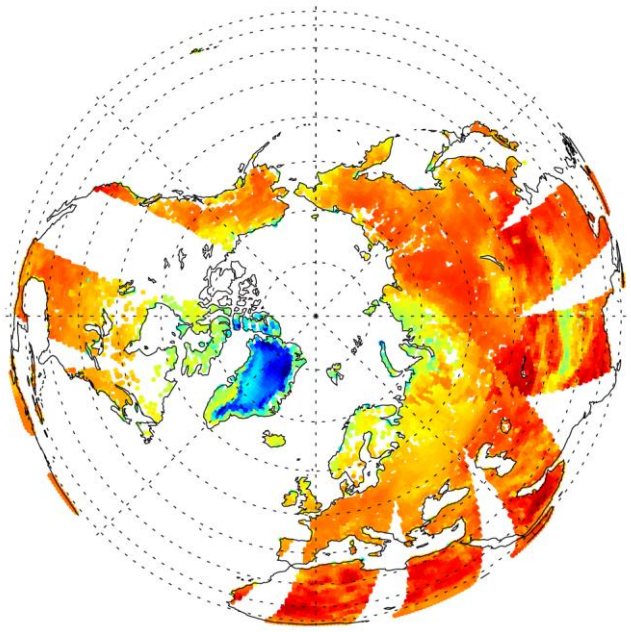
```
history = model.fit(x1, y1, epochs=100)
```



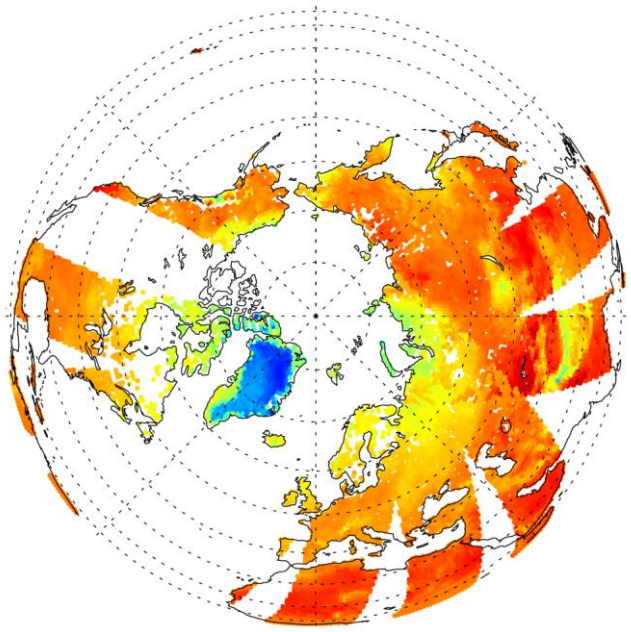
Adam – a sophisticated stochastic gradient descent (SGD) minimiser

Results (ability to fit training dataset)

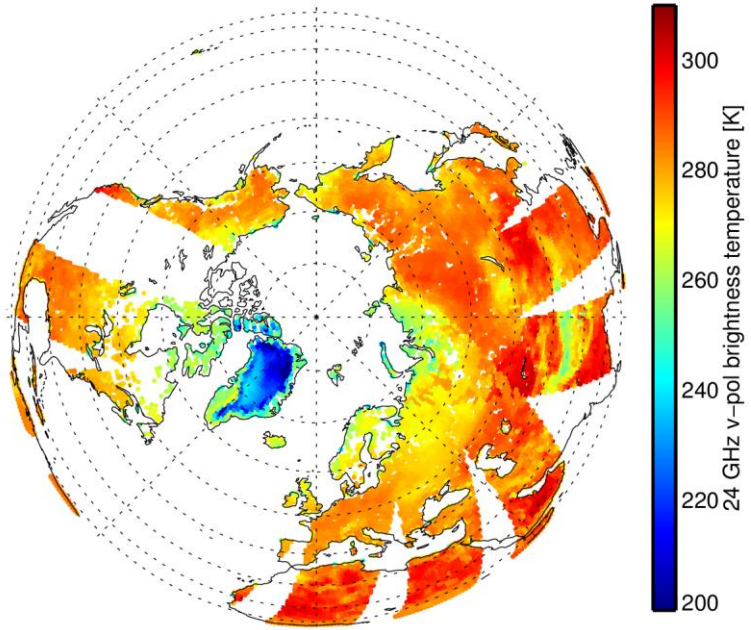
Observations



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
- Some of the fundamental problems for machine learning in the earth system domain:
  - 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

Under the hood of machine learning

Looking at maths that is usually done automatically by modern ML tools

ECMWF

Backpropagation for inputs (compare to earlier TL and adjoint lecture)

- Consider one layer, ignoring the bias weights for simplicity

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

- Differentiate using the chain rule at a given x

$$\left.\frac{d\mathbf{y}}{d\mathbf{x}}\right|_{\mathbf{x}} = \left.\frac{d\sigma}{d\mathbf{x}'}\right|_{\mathbf{W}\mathbf{x}} \left.\frac{d\mathbf{x}'}{d\mathbf{x}}\right|_{\mathbf{x}}$$

- These "Jacobians" can be represented as matrices

$$\left.\frac{d\mathbf{y}}{d\mathbf{x}}\right|_{\mathbf{x}} = \mathbf{G}_{\mathbf{W}\mathbf{x}}\mathbf{W}$$

- The magic of adjoints allows us to compute the gradient of the cost function J with respect to a change in its inputs

$$\left.\frac{dJ}{d\mathbf{x}}\right|_{\mathbf{x}} = \mathbf{W}^T\mathbf{G}^T_{\mathbf{W}\mathbf{x}} \left.\frac{dJ}{d\mathbf{y}}\right|_{\mathbf{x}}$$

Backpropagation for updating weights

For simplicity, differentiate with respect to one weight, w_ij

$$\mathbf{x}' = \mathbf{W}\mathbf{x} \quad \Rightarrow \quad x'_i = \sum_j w_{ij} x_j \quad \Rightarrow \quad \left.\frac{dx'_i}{dw_{ij}}\right|_{x_j} = x_j$$

One output y_i is sensitive to all inputs x_j

$$\mathbf{y} = \sigma(\mathbf{x}') \quad \Rightarrow \quad y_i = \sigma(x'_i) \quad \Rightarrow \quad \left.\frac{dy_i}{dx'_i}\right|_{x'_i} = \left.\frac{d\sigma}{dx'_i}\right|_{x'_i}$$

Gradient of cost function with respect to one weight

$$\left.\frac{dJ}{dw_{ij}}\right|_{x_j} = x_j \left.\frac{d\sigma}{dx'_i}\right|_{x'_i} \left.\frac{dJ}{dy_i}\right|_{x_j}$$

Gradient of cost function with respect to the relevant simulated observation y_i

Gradient depends on x_j (and $x'_i = \sum_j w_{ij} x_j$)

Mini-batch stochastic gradient descent

For a randomly selected batch of paired inputs and outputs update all weights (typical batch size: 32)

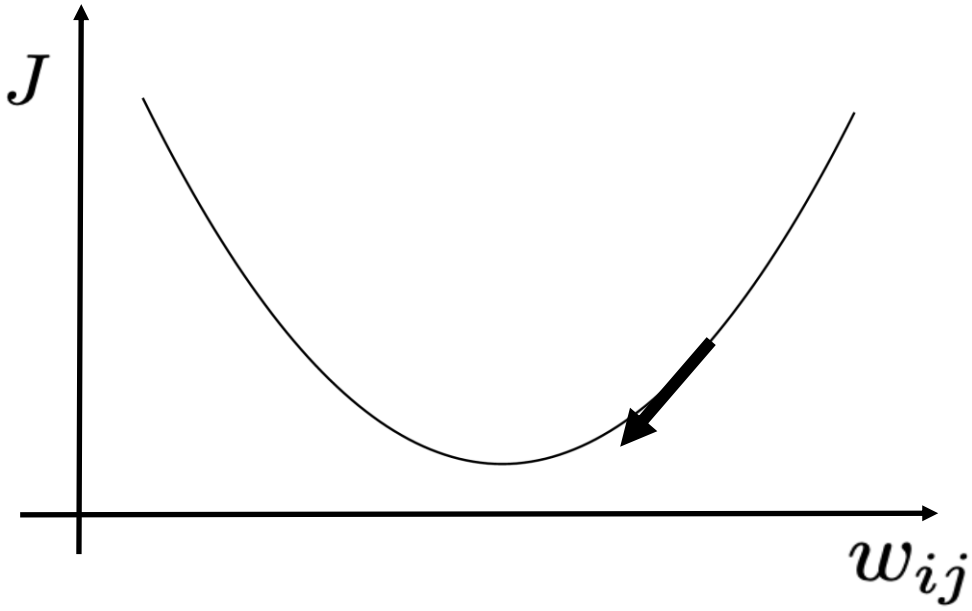
$$w_{ij} = w_{ij} - \eta \frac{dJ}{dw_{ij}}\bigg|_{x_j}$$

New weight setting

Old weight setting

Learning rate

Gradient of loss function with respect to the weight for this x



SGD versus standard gradient descent in DA

- Stochastic gradient descent:
  - Each minimisation is done on one randomly selected mini-batch of data, requiring:
    - One call to the forward model to compute the linearisation state for the gradients
    - One call the the adjoint (backpropagation model)
    - All weights are updated
  - One epoch = one pass through the entire dataset

100 epochs * 15,000 mini batches ->
1,500,000 model runs (forward/adjoint)

- Gradient descent in incremental 4D-Var:
  - Each outer loop needs one call of the nonlinear model to update linearisation state
  - Each minimisation needs 30 – 50 "inner loop" iterations, each needing
    - One call the the TL model
    - One call to the adjoint model

4 outer loops * 40 inner loops ->
approx 160 model runs (forward/adjoint)

Machine learning is (nearly) data assimilation

ECMWF

The forward and inverse problem

Forward model

Observations

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

Geophysical state

Model parameters

The inverse problem solved by Bayes theorem

with state AND parameters

Observations

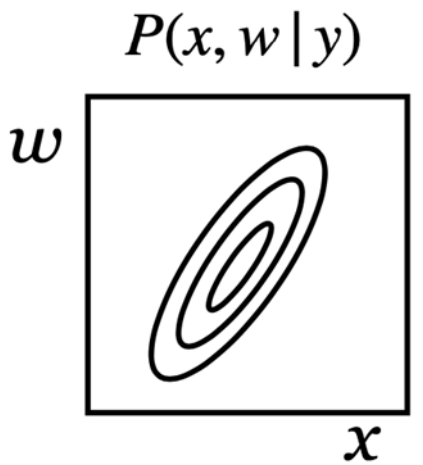
Geophysical state

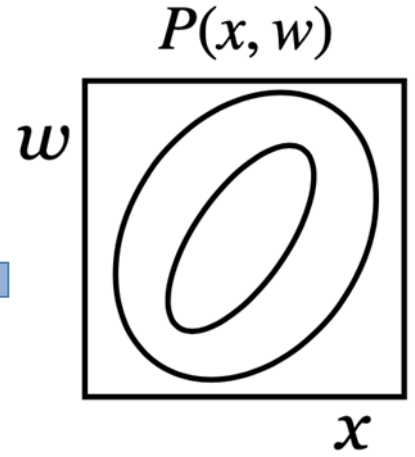
Model parameters

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**

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

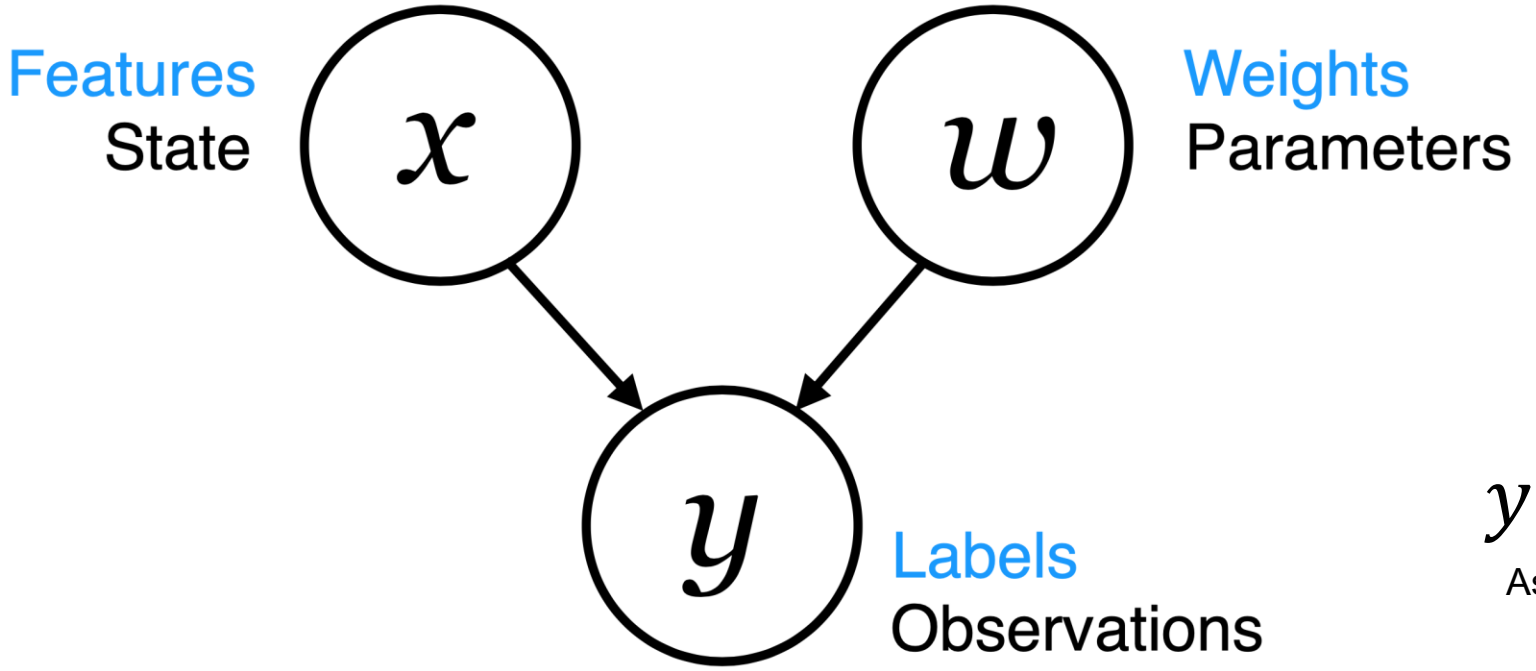
DA **Cost function**

Observation term | **Prior knowledge of state** | **Prior knowledge of model**

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	$\|\mathrm{w}\|$	Background term	$J^b = \left(\mathrm{x} - \mathrm{x}^{\mathrm{b}}\right)^T \mathrm{B}^{-1}\left(\mathrm{x} - \mathrm{x}^{\mathrm{b}}\right)$
Stochastic 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

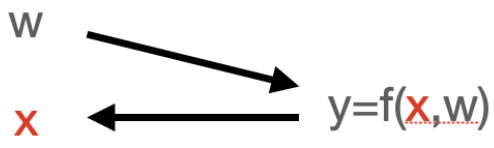
Granular hybrid ML – data assimilation

This is just a generalised state and parameter estimation

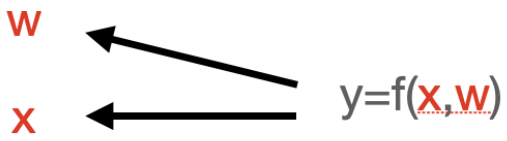
ECMWF

Varieties of state and parameter estimation

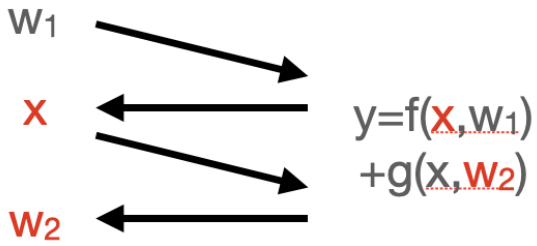
Estimate state: perfect model



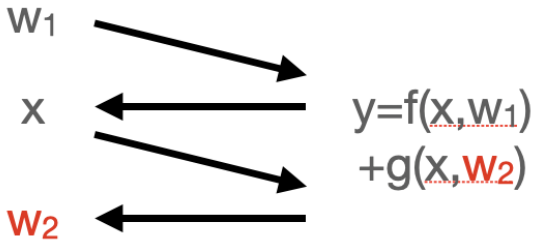
Estimate state and parameters: imperfect model



Estimate state and error correction



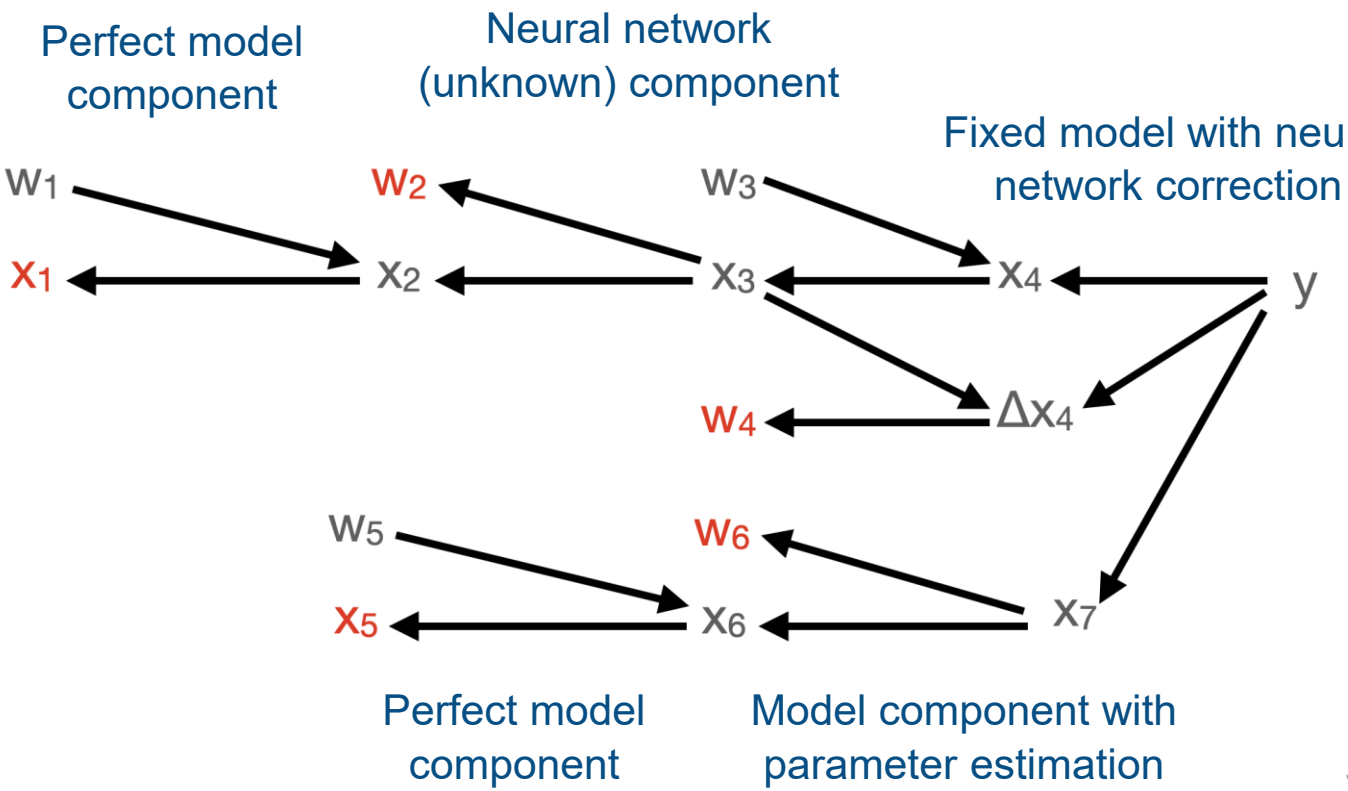
Estimate error correction



Key

Model error correction is covered in Alban's lecture, next

Building a granular hybrid model



$$y=f(x_1,x_5,w_2,w_4,w_6,w_1,w_3,w_5)$$

- Updated geophysical state
- Trainable parameters (ML weights and physical parameters)
- Fixed parameters (perfect model components)

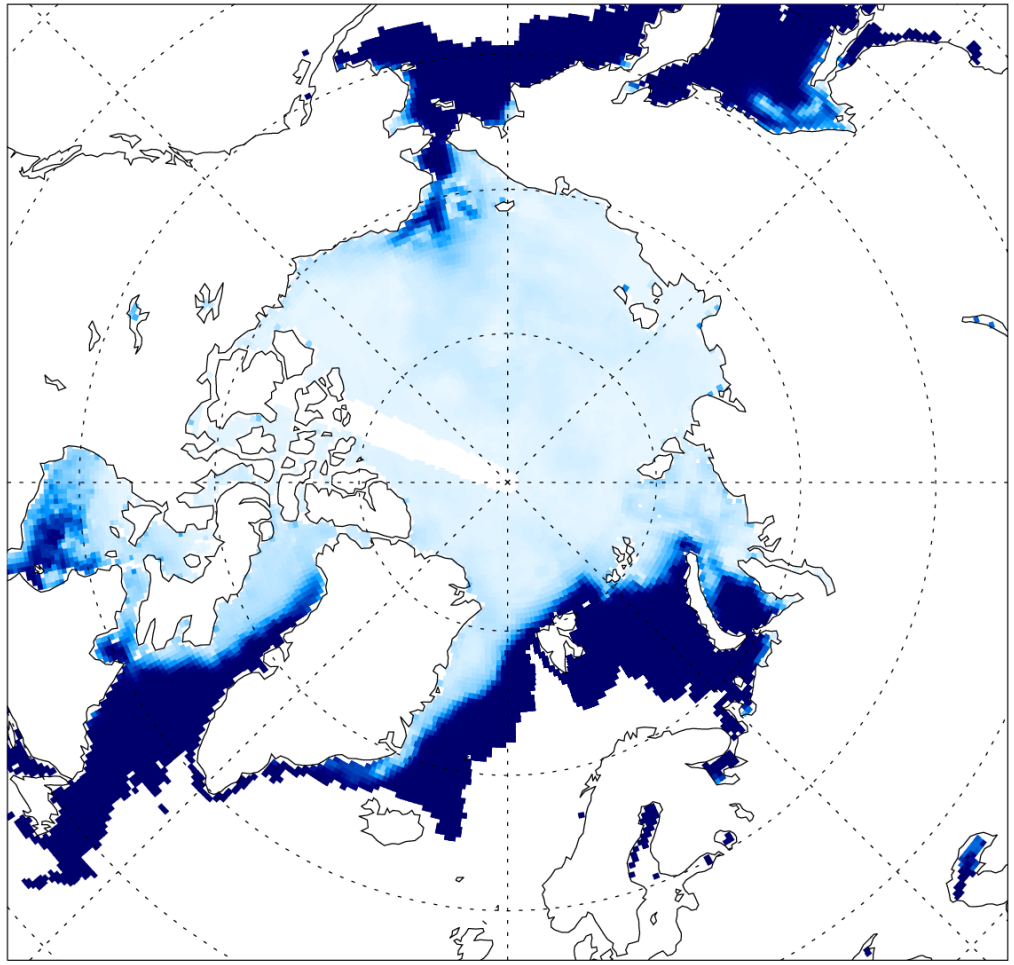
ECMWF

Granular hybrid: sea ice observation operator

ECMWF

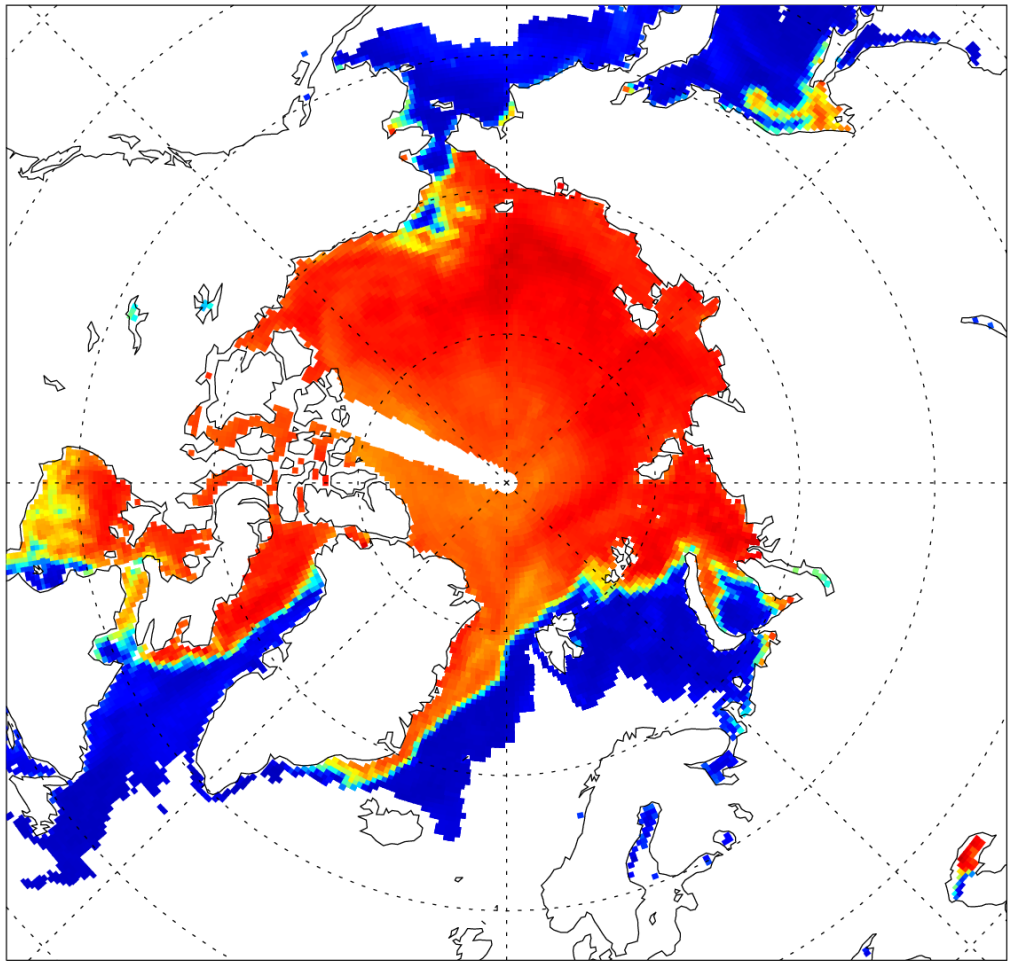
How to improve this, given this?

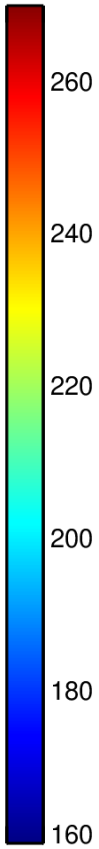
NEMO/SI3 background, 00Z 10th Dec 2022



Sea ice concentration

AMSR2 10 GHz observation, 00Z 10th Dec 2022



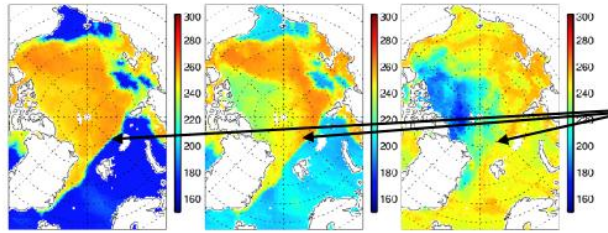


Brightness temperature [K]

A trainable empirical-physical network for sea ice assimilation

AMSR2 observations

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



Known atmosphere radiative transfer

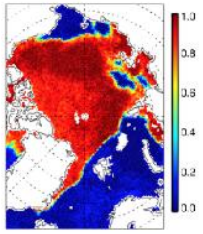
$\mathbf{y} = f_{\text{atmos}}(\mathbf{e}, T_S, \mathbf{z}_A)$

Mixed surface emissivity

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

$\mathbf{e}_{\text{wat}}$ Known water surface emissivity

Map of sea ice fraction to be estimated



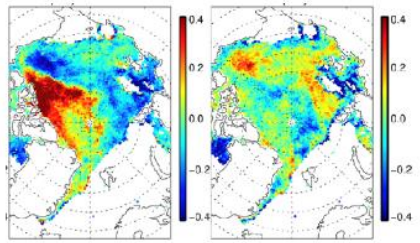
$\mathbf{C}_{\text{ice}}$ C_{ice} $C_{\text{ice}} = h(\mathbf{C}_{\text{ice}}, \mathbf{z}_G)$

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

$\mathbf{z}_G$ Geolocation

$\mathbf{z}_B$

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



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

$\mathbf{w}$ w – trainable weights of NN model for sea ice

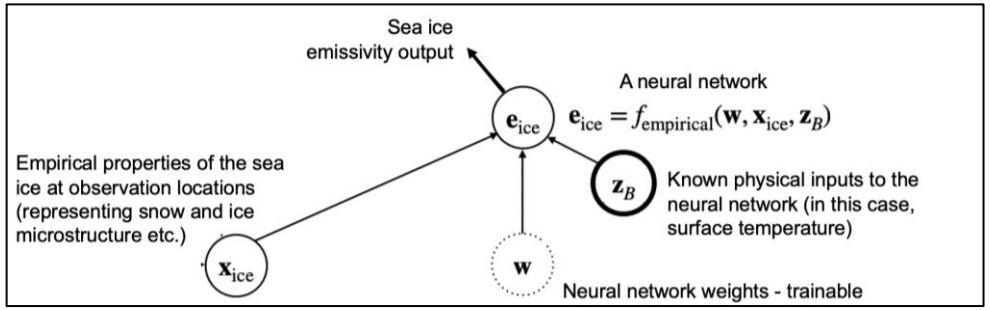
h() Interpolation operator: map to observation location in time and space

Fixed parameter

Dependent parameter

Trainable parameter

Built in Python and Tensorflow




```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
```

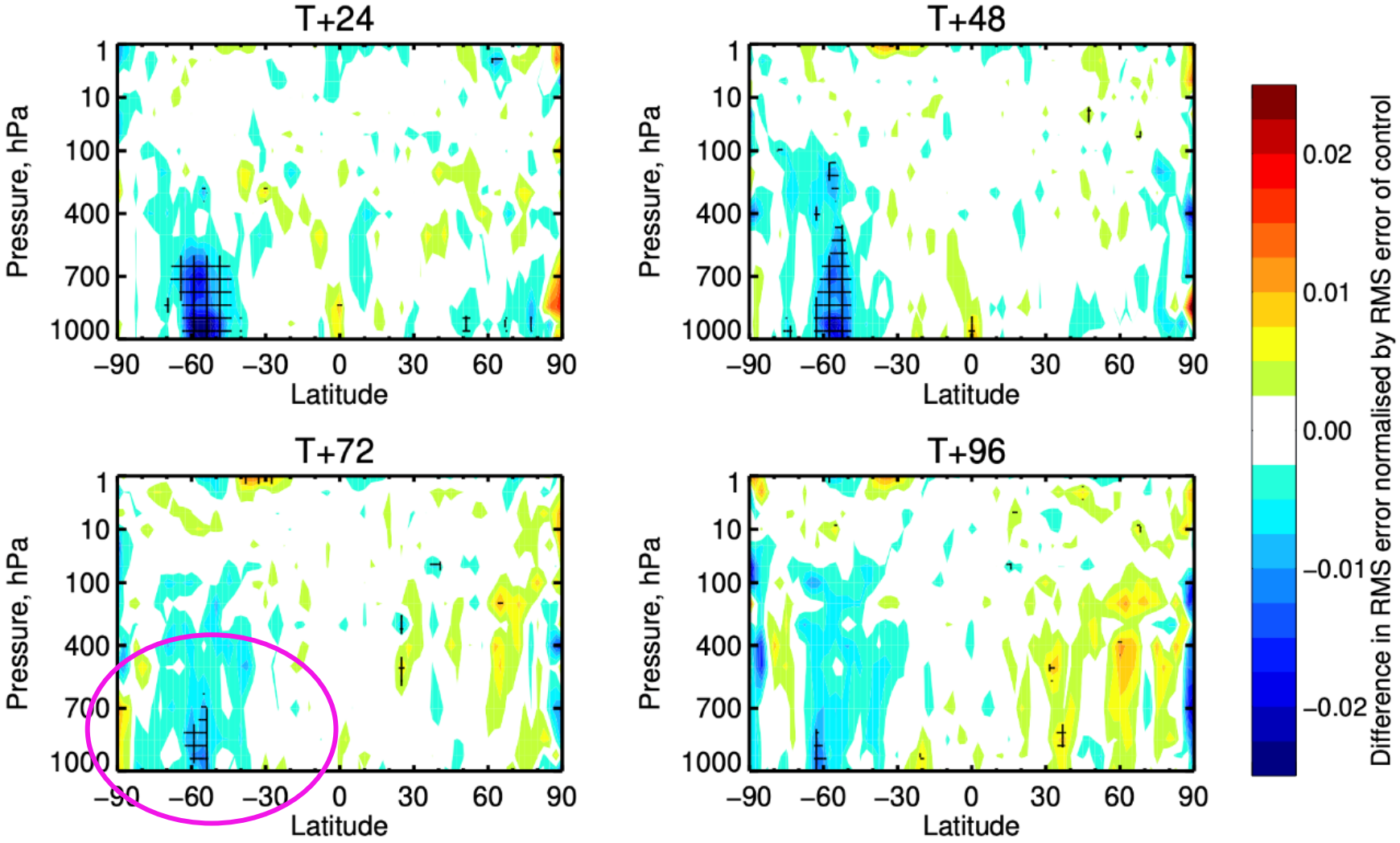
A standard dense neural network layer with linear activations

Custom loss functions to regularise / constrain the solution

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

Forecast impact on temperature from adding observations obs over sea ice regions to 4D-Var
(blue = reduced error; +++ = statistical significance)

Improved temperature forecasts out to 72 hours in the Southern Ocean



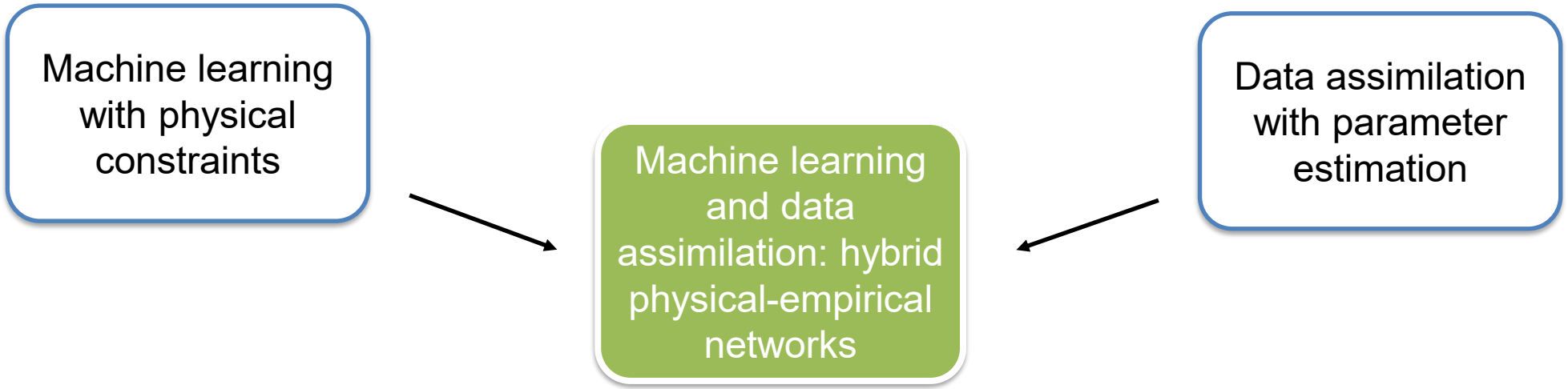
Hybrid physical-empirical networks - sea ice example

- Sea ice concentration and empirical state estimation is 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?
- Sea ice concentration retrievals from this system will be assimilated in the ocean data assimilation component from cycle 50r1 (May 2026)

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



- 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

Don't throw away the physical model – improve it!