Data assimilation and machine learning

Alan Geer

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

Lecture at NERC/NCEO/DARC Training course on data assimilation and its interface with machine learning

13th June 2025

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

ECMWF

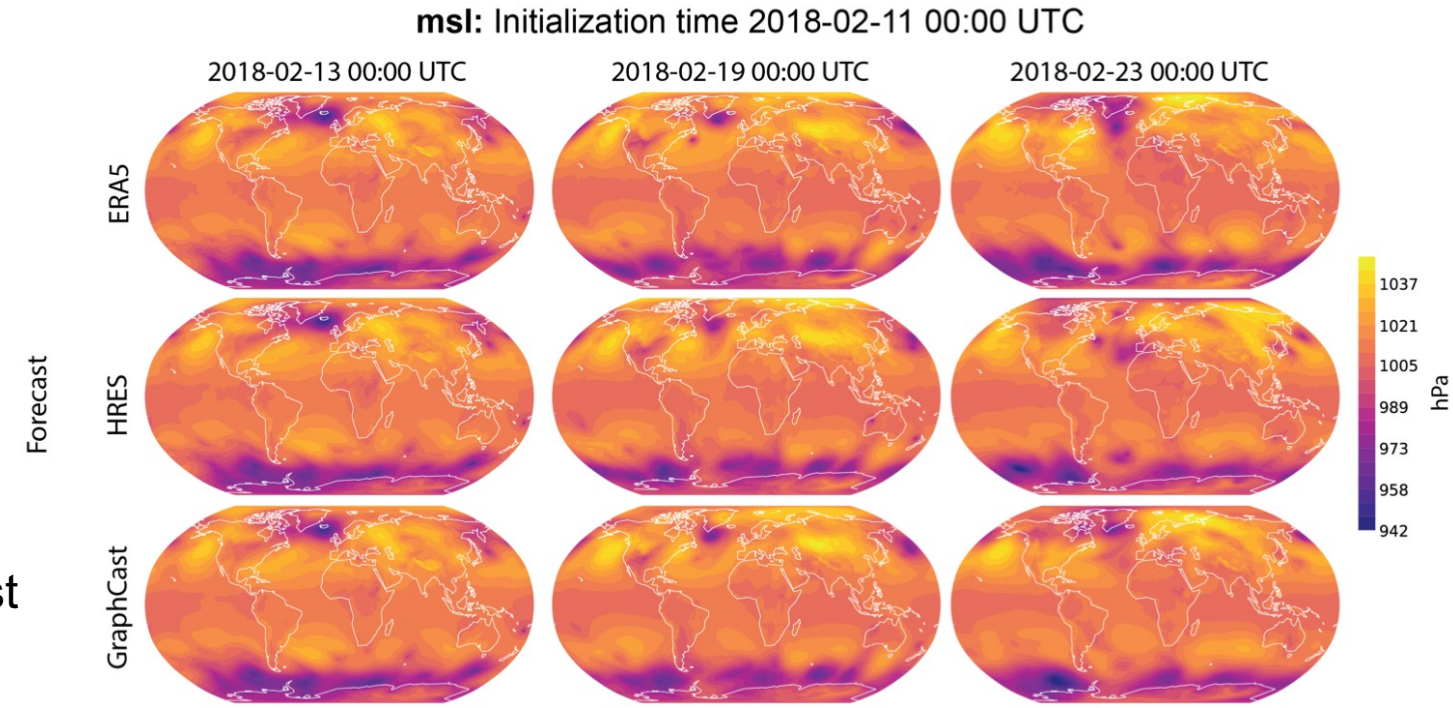
Forecast models based on machine learning are here and they're good!

- 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)



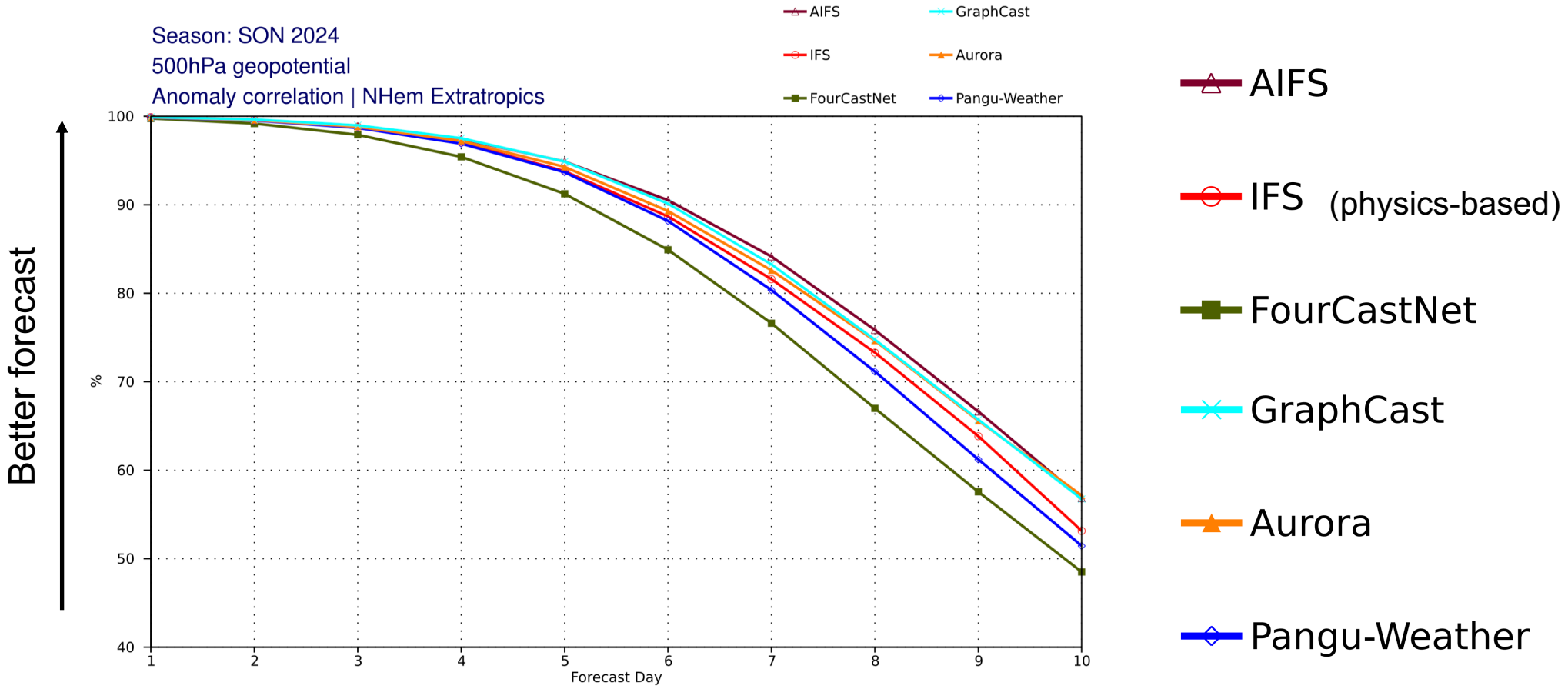
Run time

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



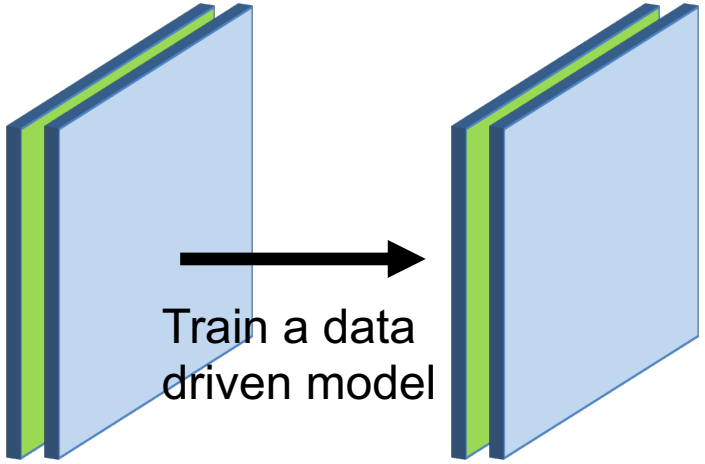
https://charts.ecmwf.int/catalogue/packages/opencharts/products/plwww_3m_fc_aimodels_wp_mean

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

Make a weather forecast using a data driven model

12 UTC 22nd Mar 2024

- 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.

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

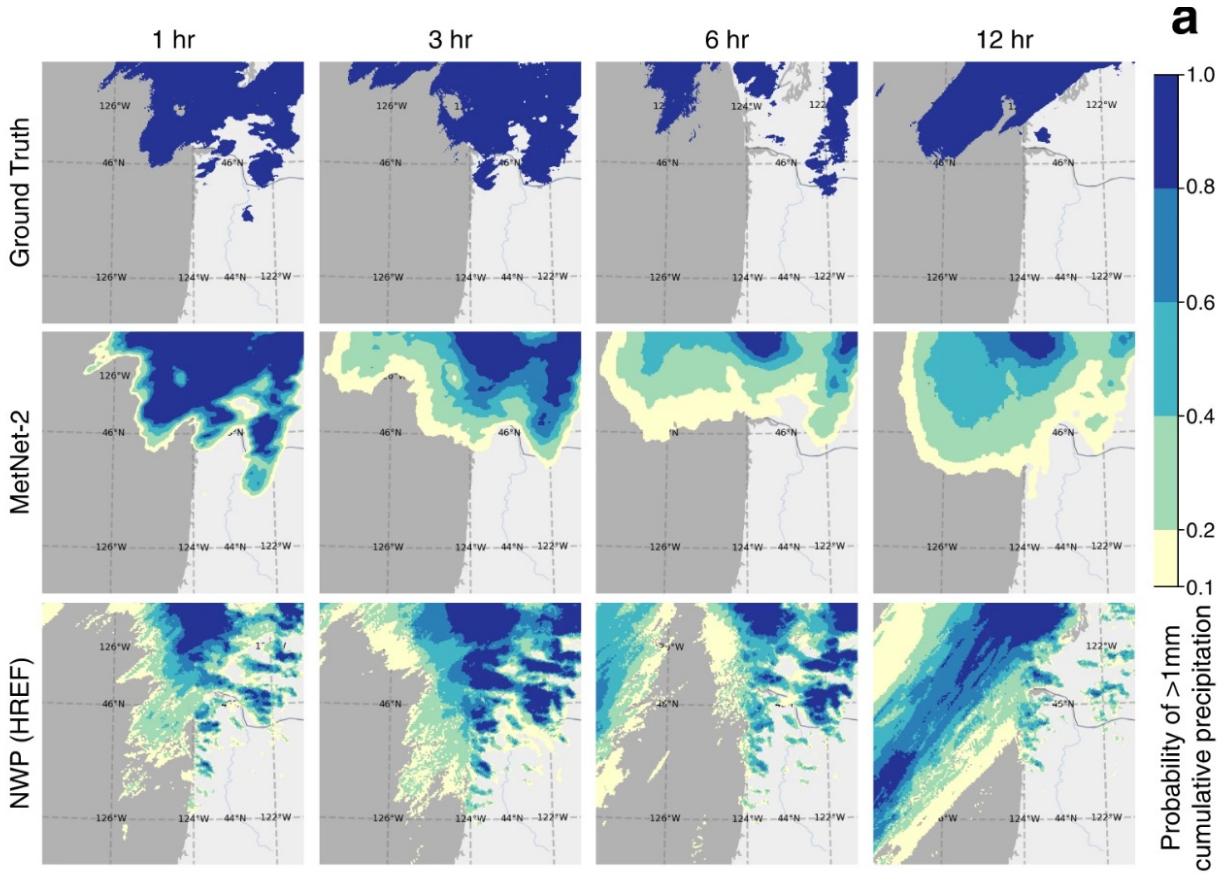
	Horizontal	Vertical	Timestep
IFS	8-9 km	137 levels	7.5 min
AIFS	36 km	13 levels	6 hour

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

- -> **The physical model is not good enough and needs to be improved using observations**

So, can we do without physical model or data assimilation and instead directly forecast from (and to) observations?

- Google MetNet-2 is 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)

An ML example: microwave land surface observation operator

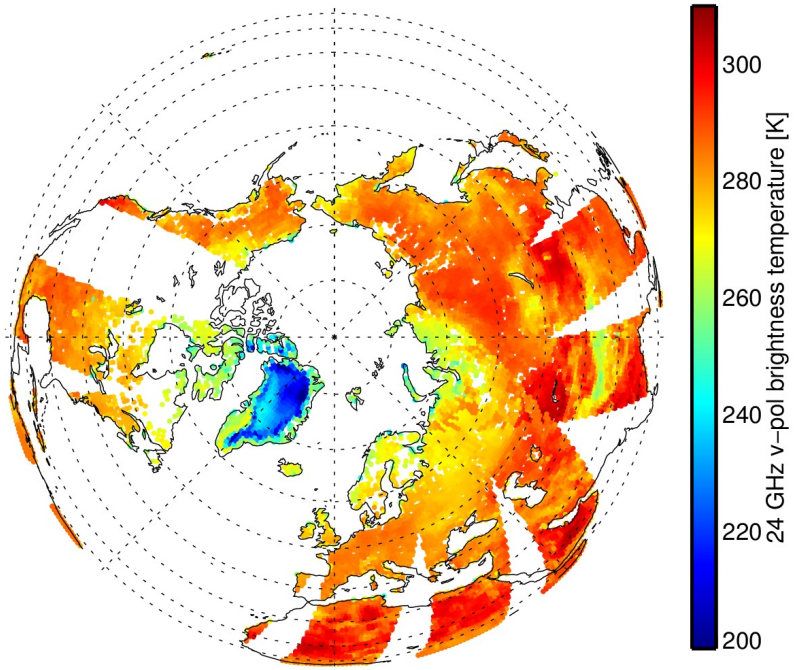
Python, Keras, Tensorflow, Numpy, Matplotlib, Xarray

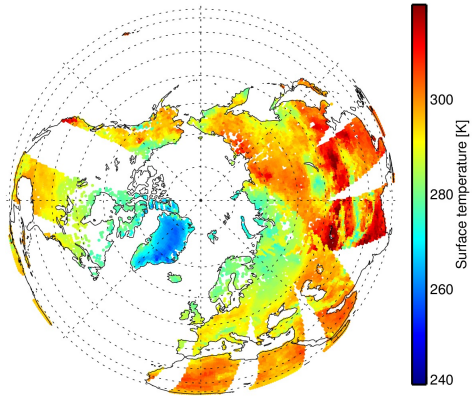
ECMWF

Datasets

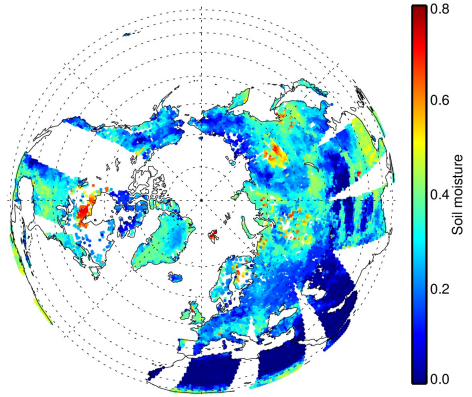
y = labels

AMSR2 24GHz v-pol observations

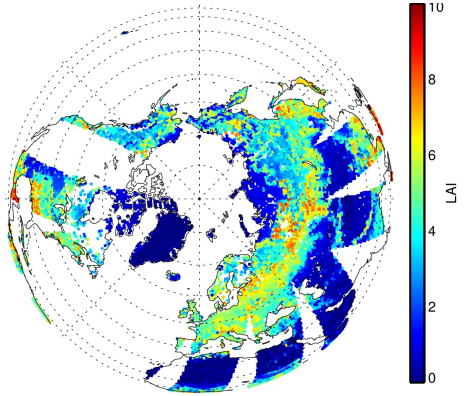




Skin temperature



Soil moisture



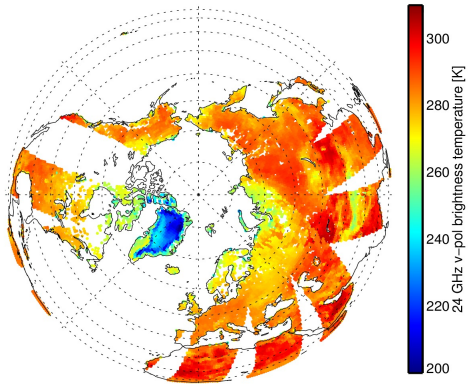
Leaf area index

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

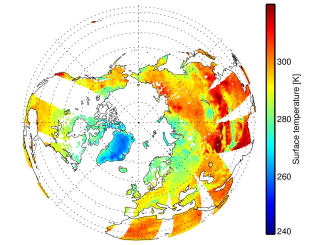
x = features

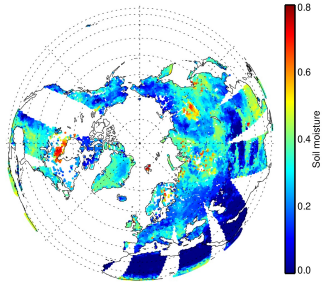
+ orography, snow depth, snow density, integrated water vapour, cloud, rain and snow water contents

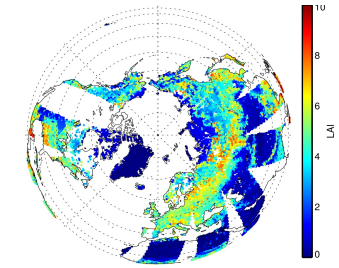
y =
labels



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

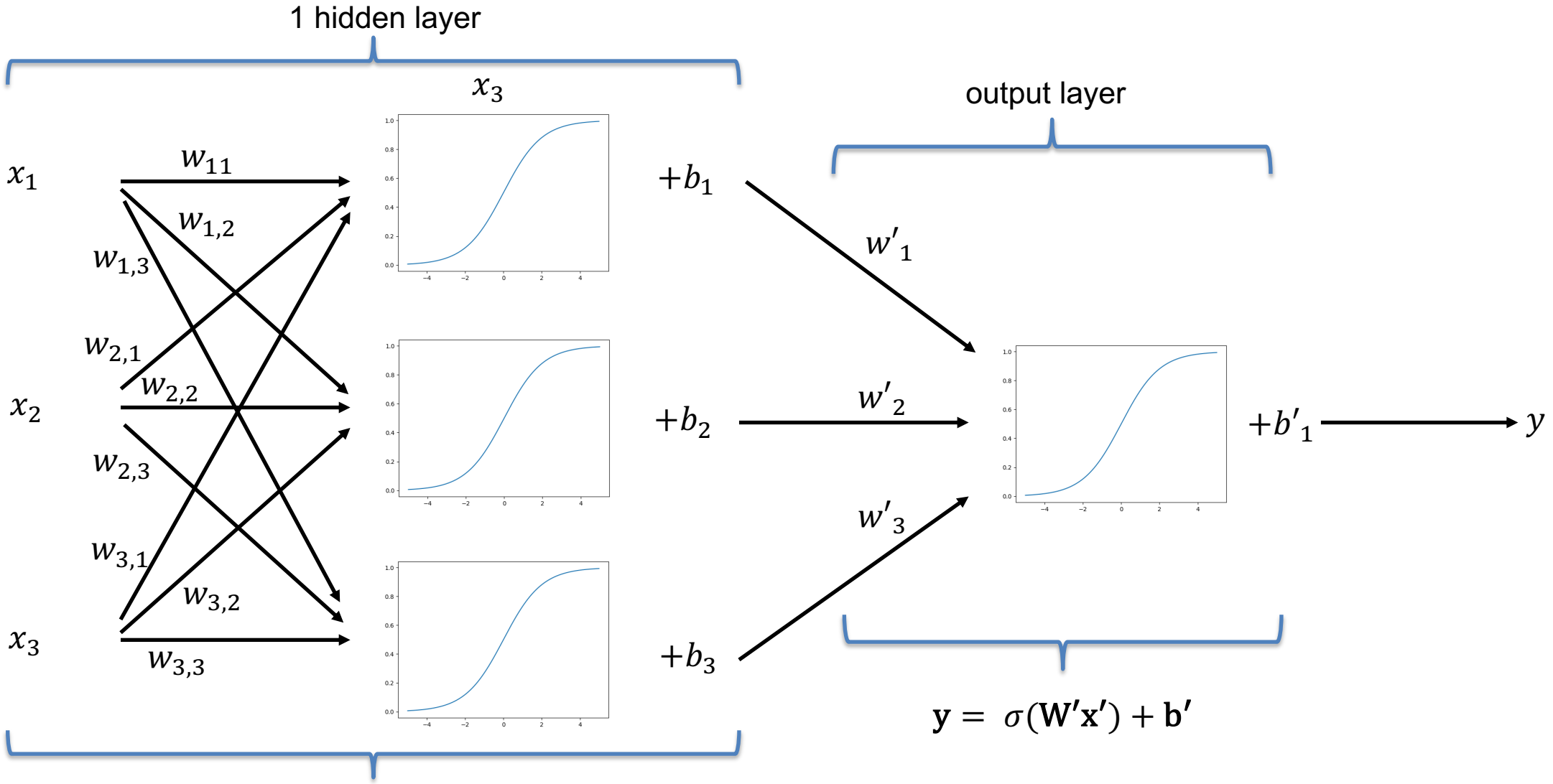
Normalise ‘features’ x to roughly -1 to +1

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
_________________________________________________________________
```

Feedforward neural network - example



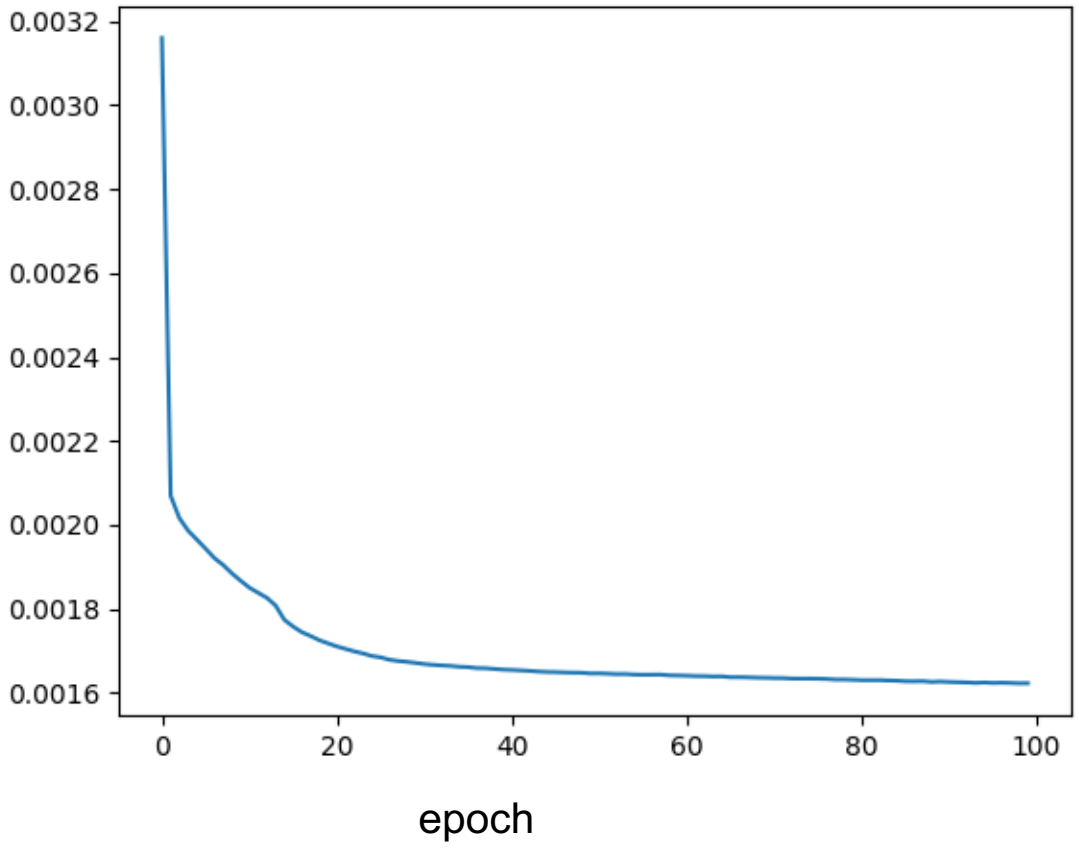
Train it (about 25 minutes on a linux workstation)

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)
```



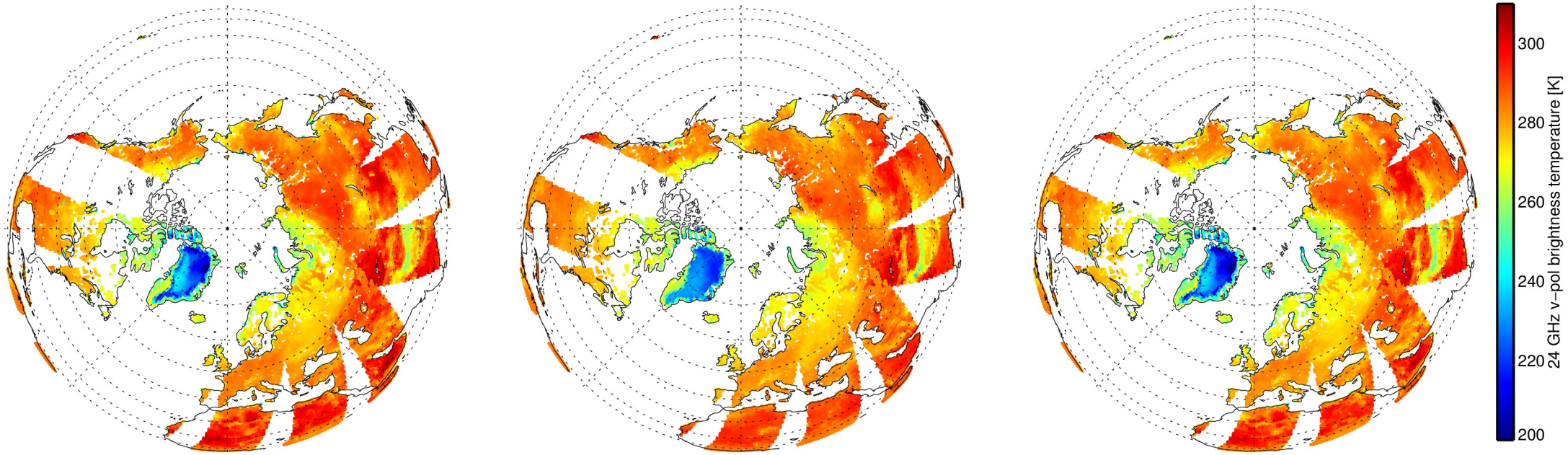
Adam – a sophisticated stochastic gradient descent (SGD) minimiser

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

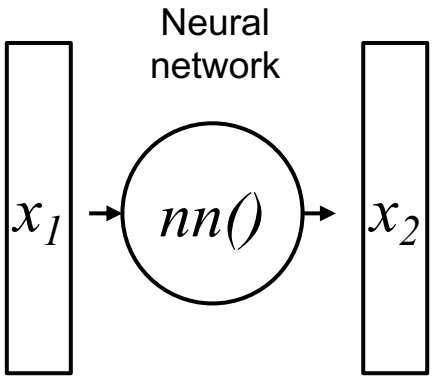
Problems with this toy NN model for 24 GHz radiances

- 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

Types of ML

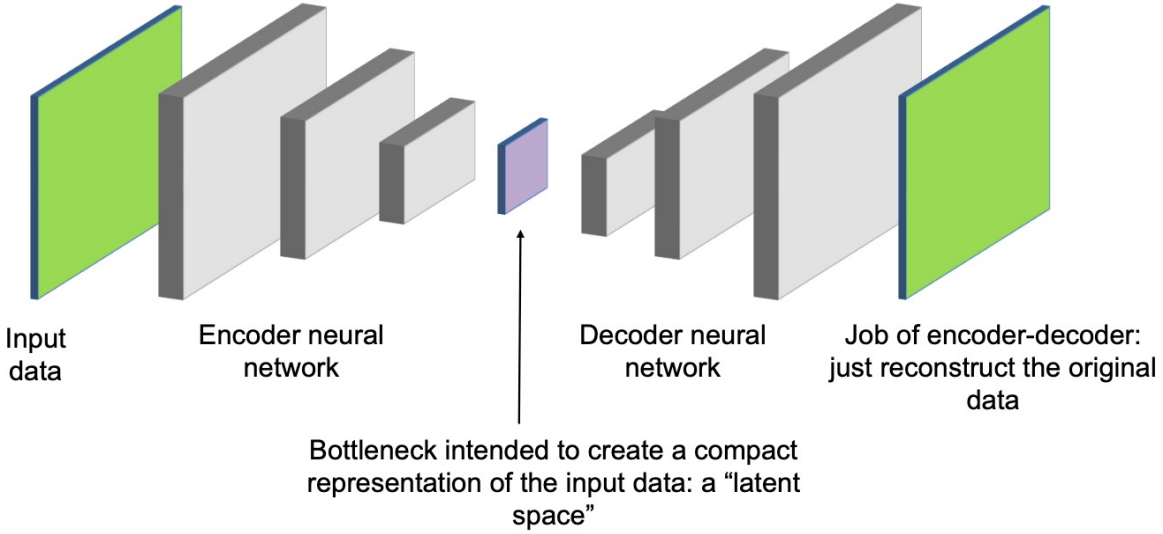
ECMWF

Types of ML – supervised learning



Supervised learning:

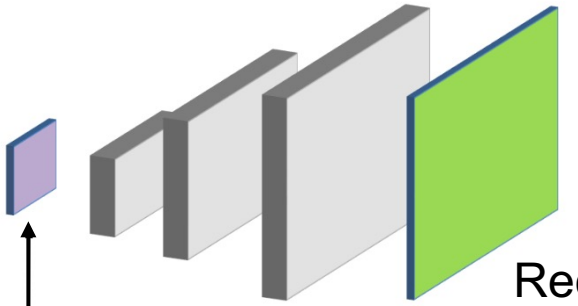
- ML as a "universal function approximator" (Hornik, 1991)
- Both inputs x1 and outputs x2 need to be provided as training data
- An "emulator" / "surrogate" / "empirical model"



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



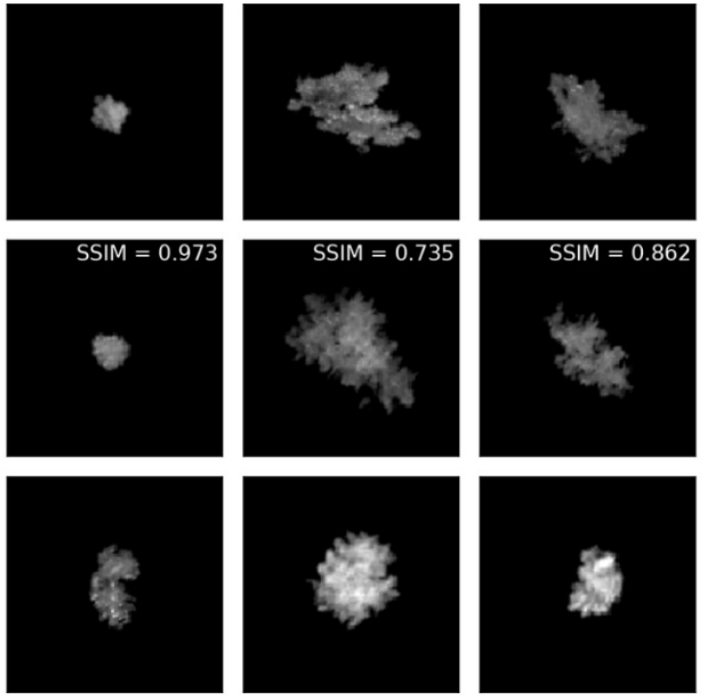
Reconstruction

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...



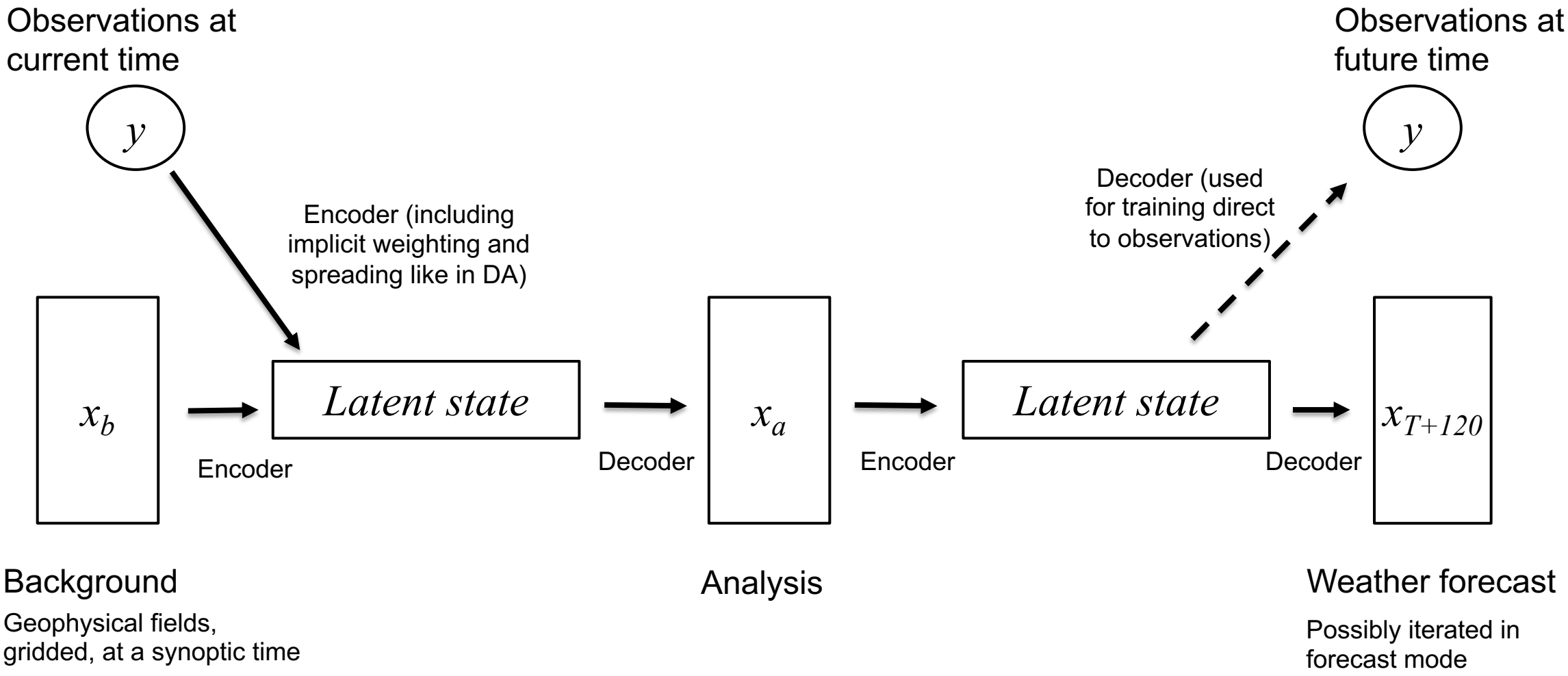
Snowflake images from Leinonen and Berne (2021, https://doi.org/10.5194/amt-13-2949-2020)

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

ECMWF

Weather forecasting completely by ML



Theoretical links between ML and DA

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

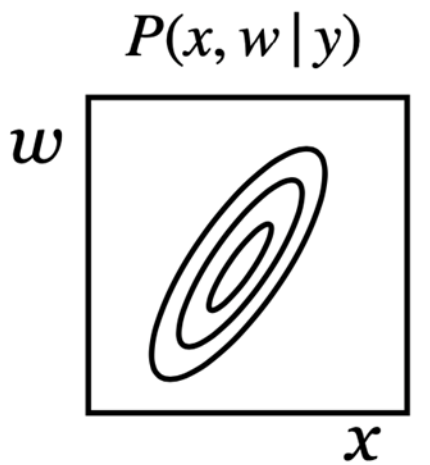
Geophysical state

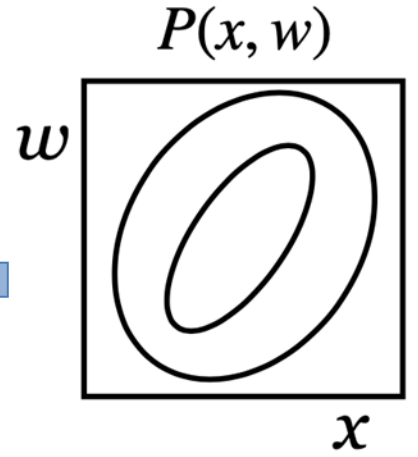
Model parameters

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**

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

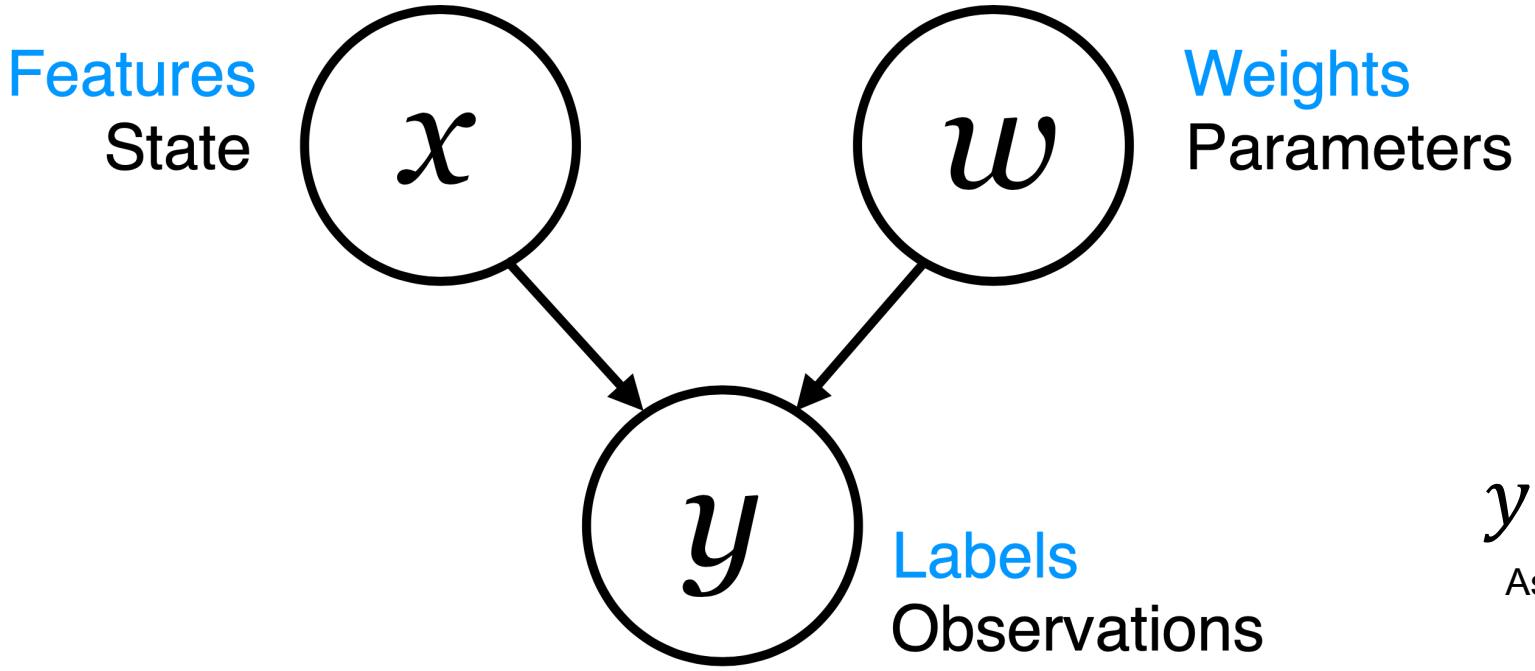
DA **Cost function**

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

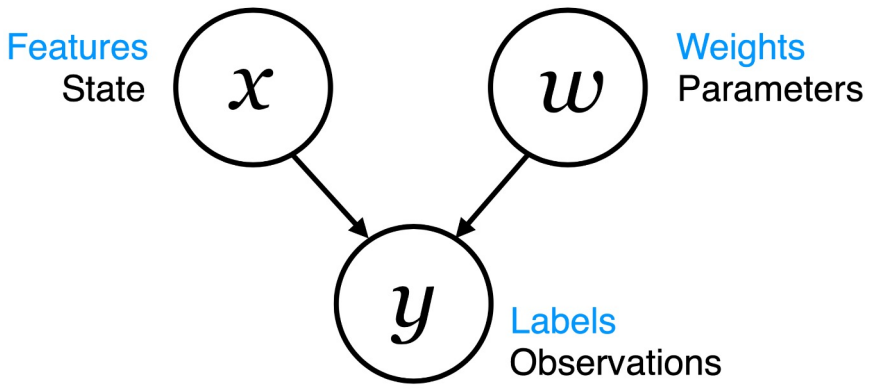
Bocquet et al. (2020) https://arxiv.org/abs/2001.06270

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

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

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

Mini-batch stochastic gradient descent

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

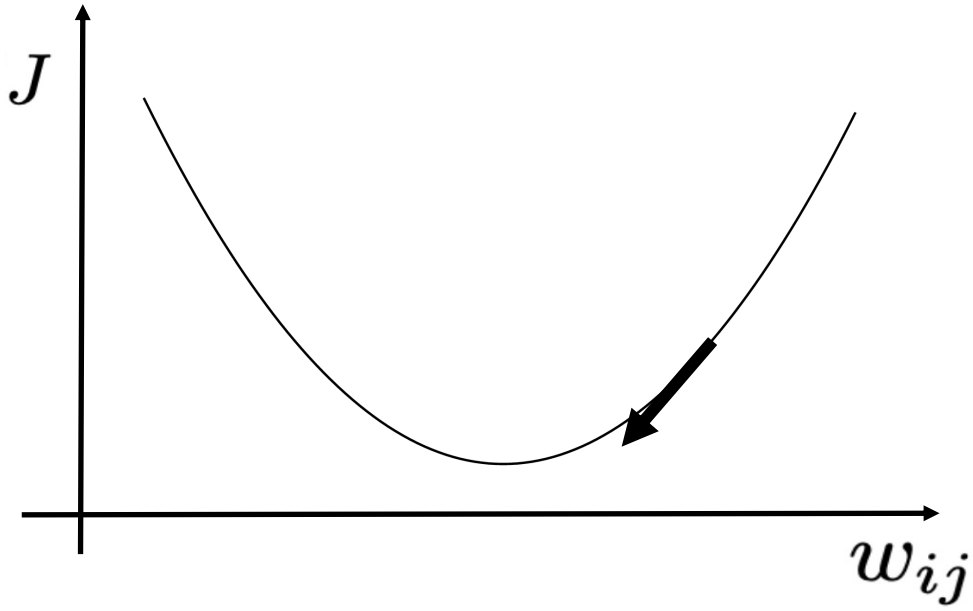
$$w_{ij} = w_{ij} - \eta \frac{dJ}{dw_{ij}}\bigg|_{x_j}$$

New weight setting

Old weight setting

Learning rate

Gradient of loss function with respect to the weight for the current estimate of state 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 (e.g. Gauss-Newton type methods):
  - 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)

Why has machine learning been so successful?

- Many packages can do all this with just a few Python commands
  - Keras, Pytorch, Tensorflow etc.
- Huge amounts of learning material available – it's easy to get started
  - Open source ML models to extend, copy, give inspiration
- Availability of GPUs to perform extremely fast matrix/tensor multiplications
  - Faster, simpler nonlinear activation functions (e.g. relu)
- Vast pools of data to train on
  - No data-driven forecasting without ERA5 reanalyses to train on
- Modern implementations of stochastic gradient descent (e.g. Adam) are incredibly good
- We are surrounded by ever more successful examples of what a "universal approximator" can be applied to...
  - How much can ML achieve?

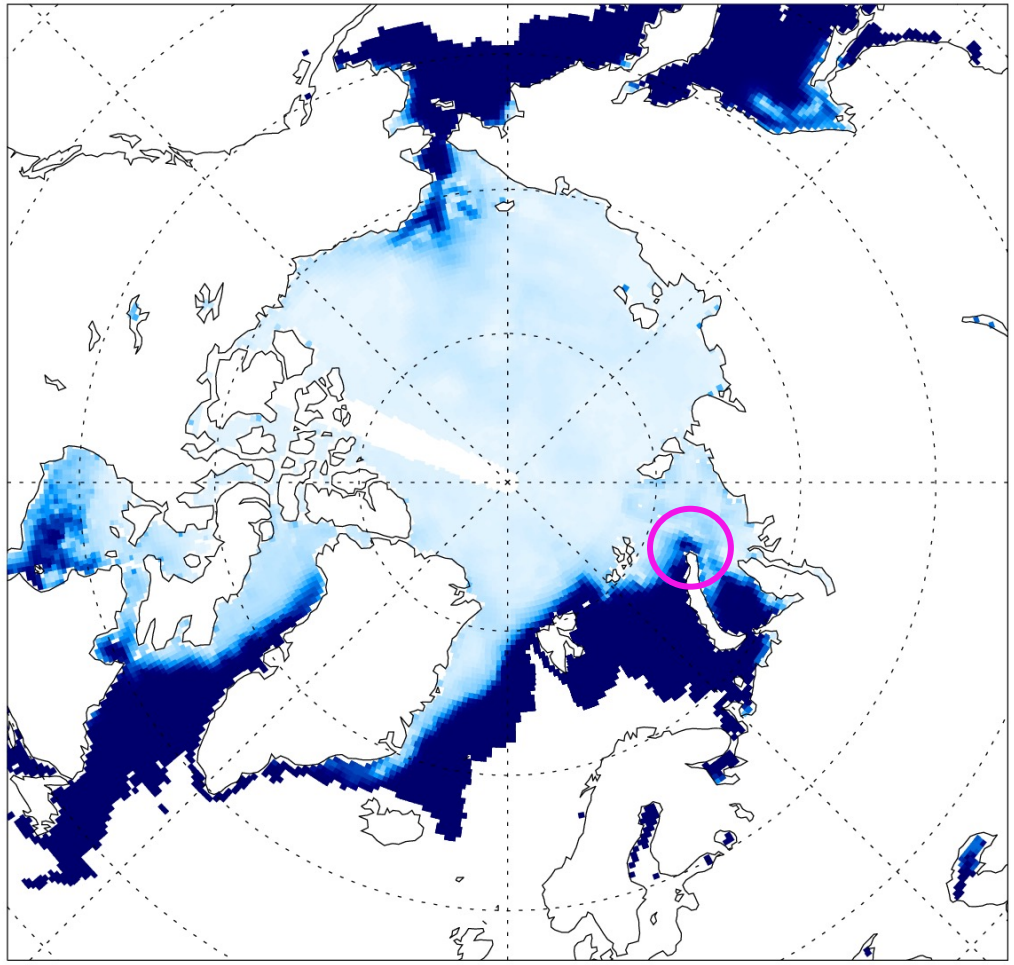
Hybrid data assimilation and machine learning

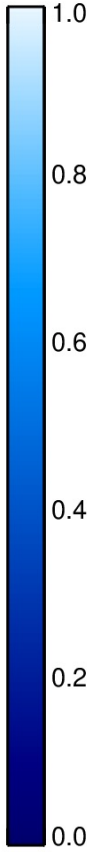
Sea ice observation operator example

ECMWF

How to improve this, given this?

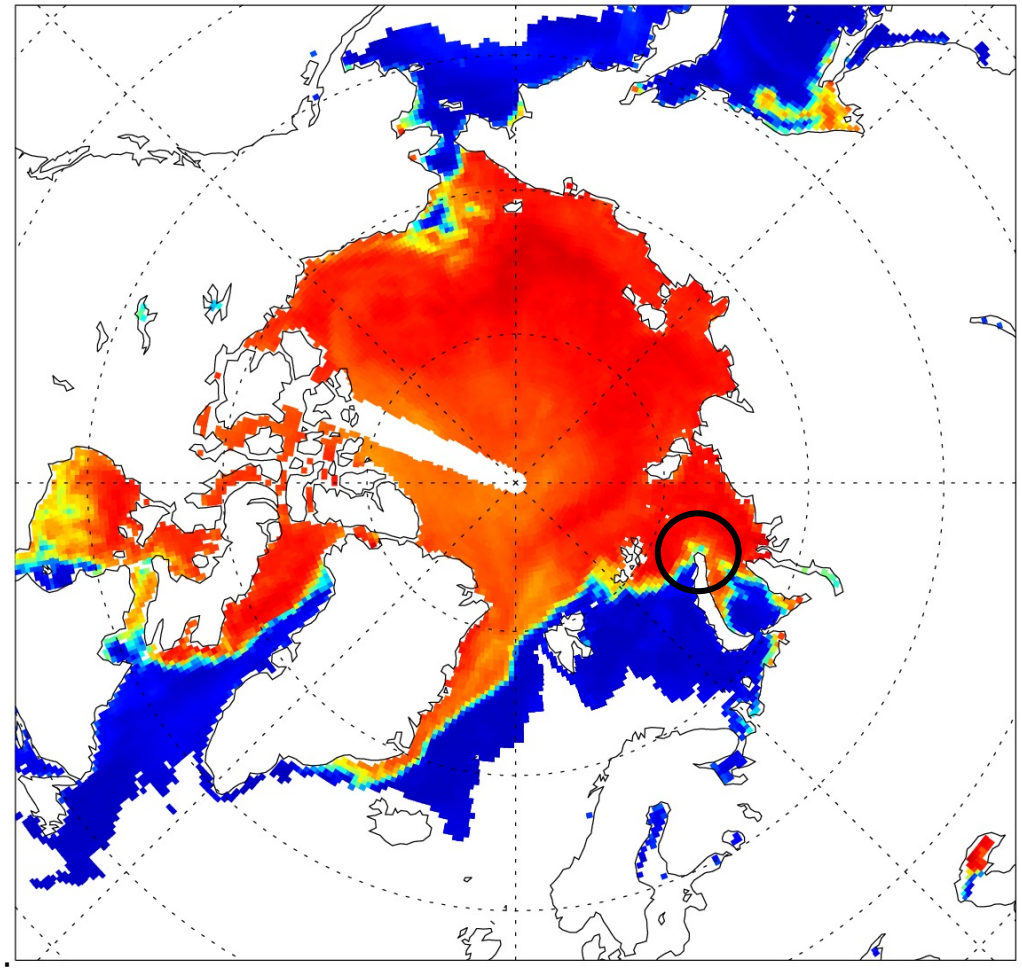
NEMO/SI3 background, 00Z 10th Dec 2022

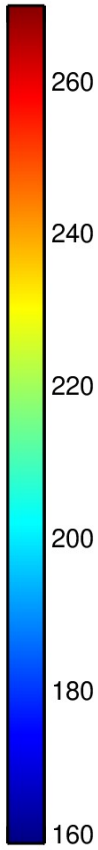




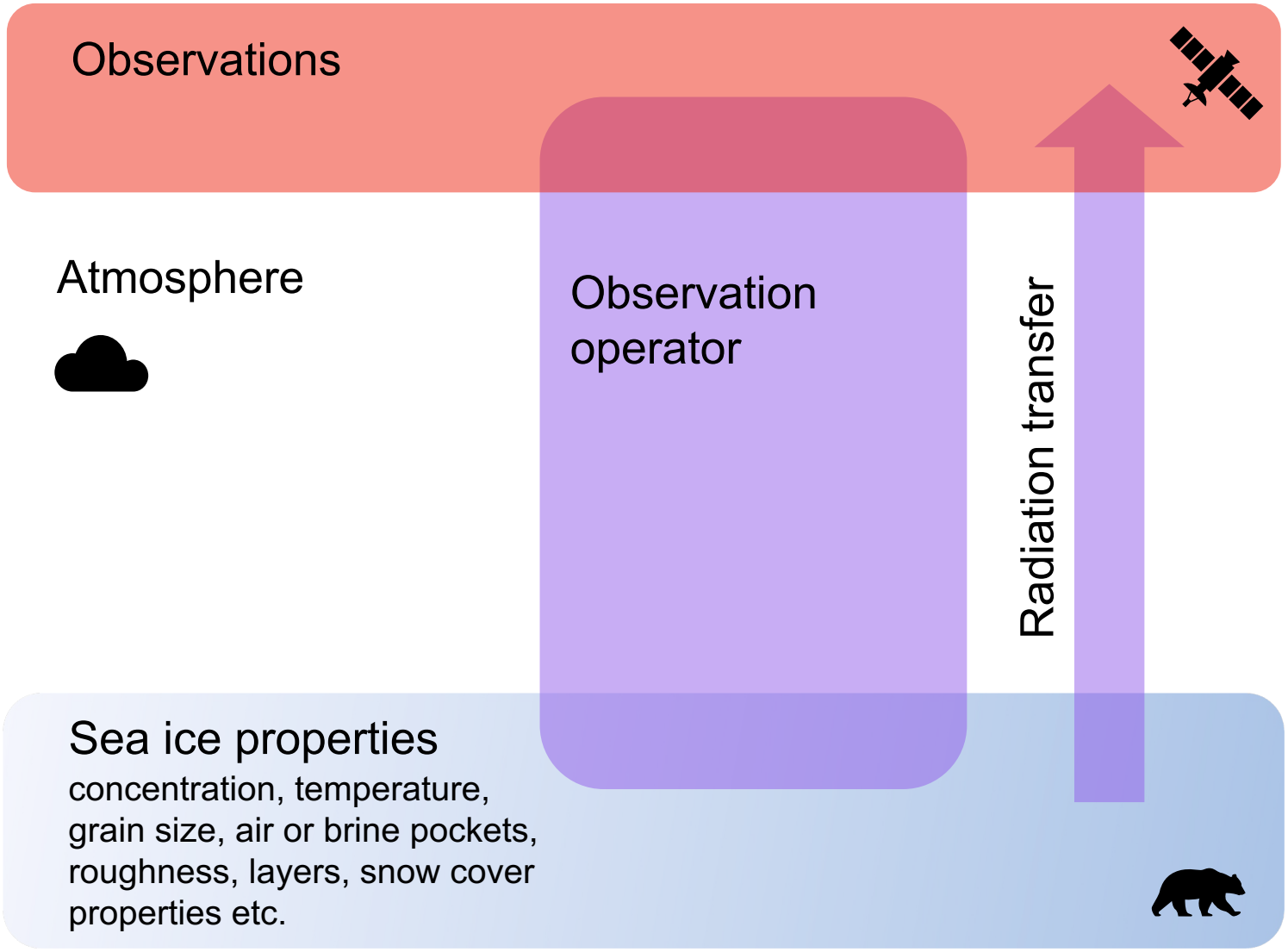
Sea ice concentration

AMSR2 10 GHz observation, 00Z 10th Dec 2022

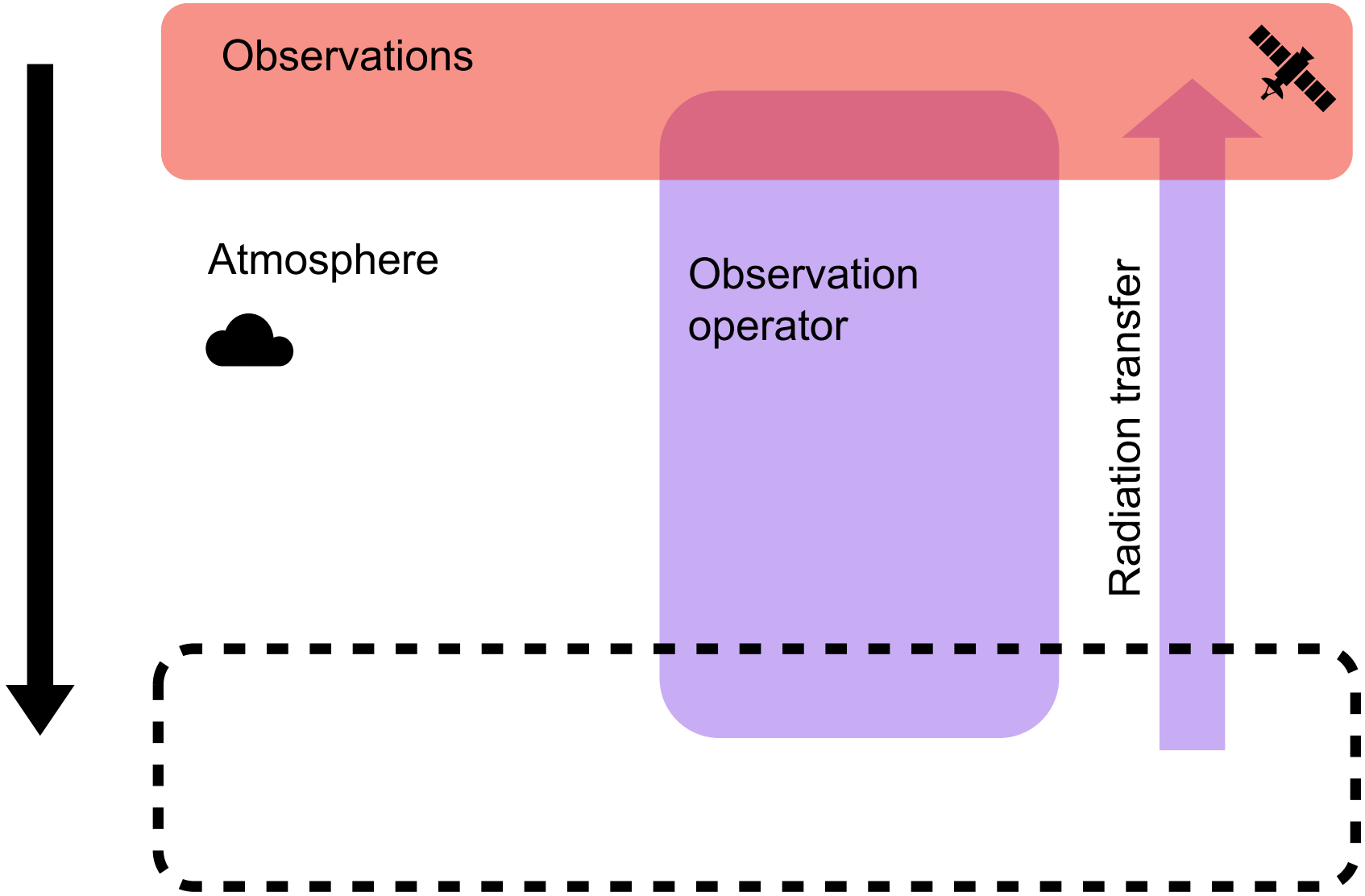




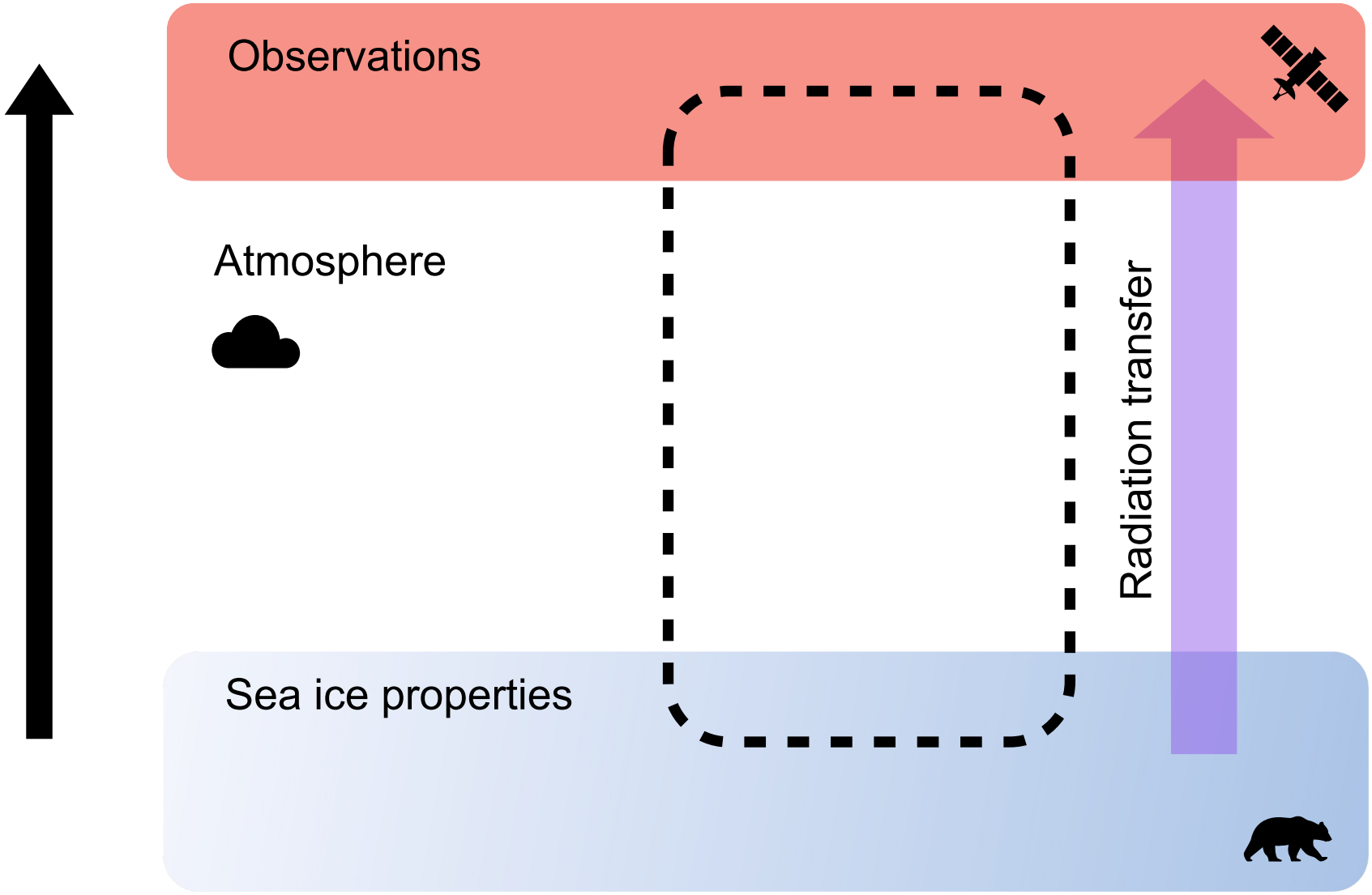
Brightness temperature

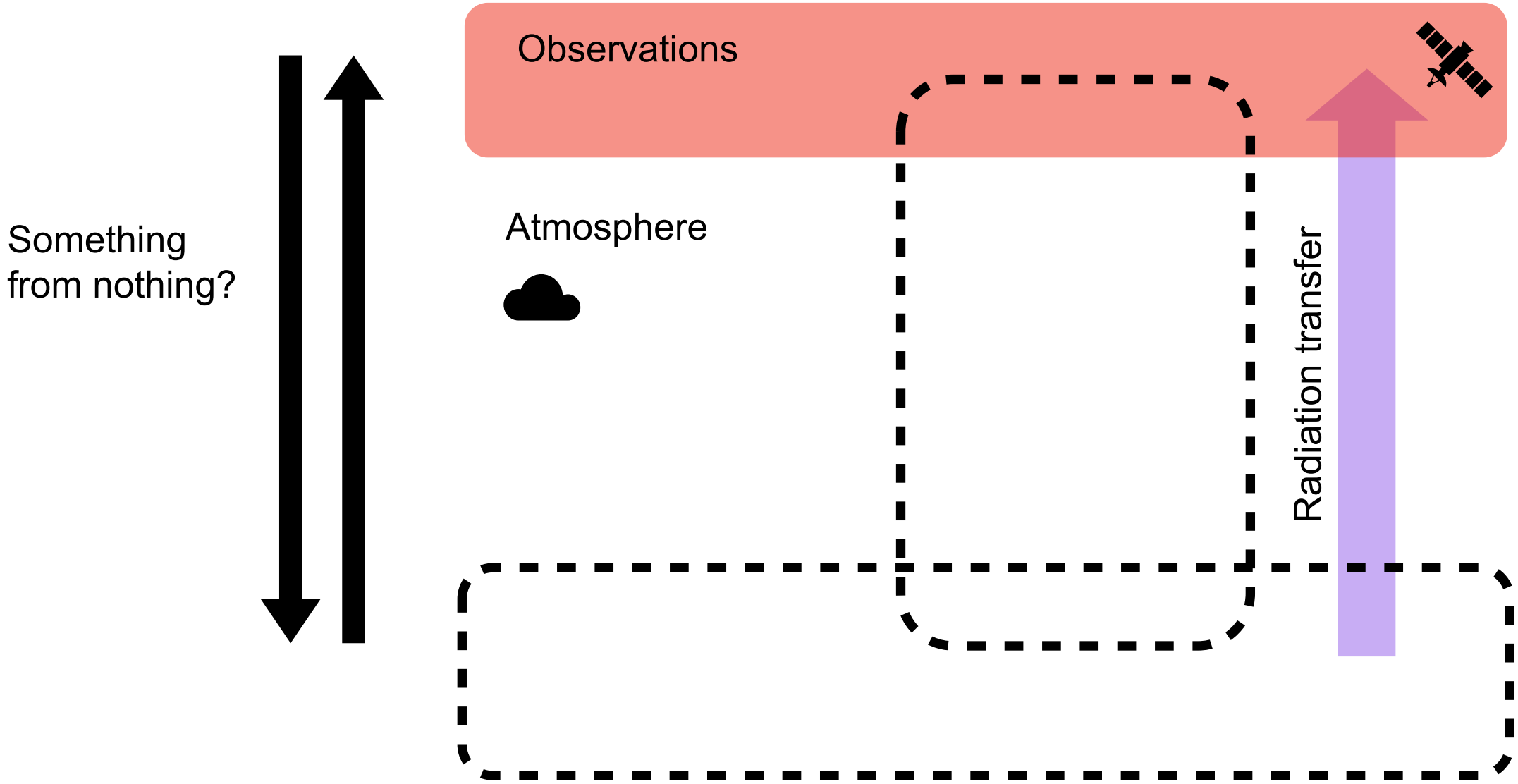

Observations
Atmosphere
Observation operator
Radiation transfer
Sea ice properties
concentration, temperature, grain size, air or brine pockets, roughness, layers, snow cover properties etc.

Inversion (data assimilation) to retrieve sea ice properties

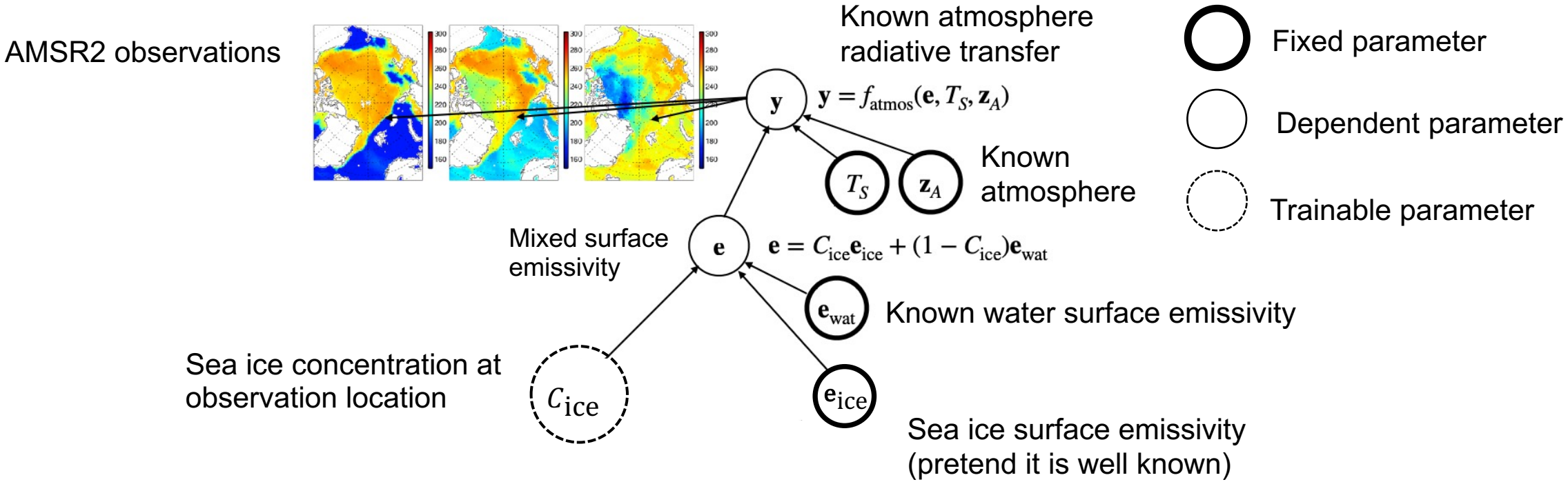


Machine learning to find the observation operator




Observations
Atmosphere
Something
from nothing?
Radiation transfer

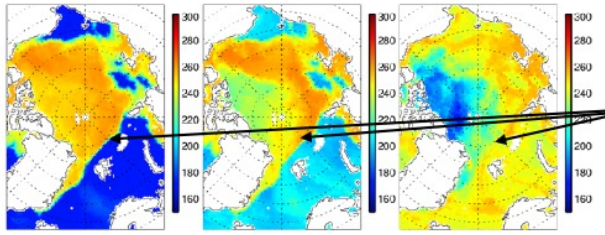
Physical (Bayesian) network representation of sea ice and snow radiative transfer for variational data assimilation



Physical (Bayesian) network representation of sea ice and snow radiative transfer for variational data 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

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

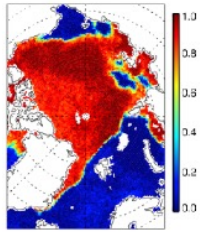
$\mathbf{y}$, T_S, $\mathbf{z}_A$

Mixed surface emissivity

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

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

Map of sea ice fraction to be estimated



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

$\mathbf{e}_{\text{ice}}$

Sea ice emissivity is not actually well known

$\mathbf{z}_G$ Geolocation

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

Fixed parameter

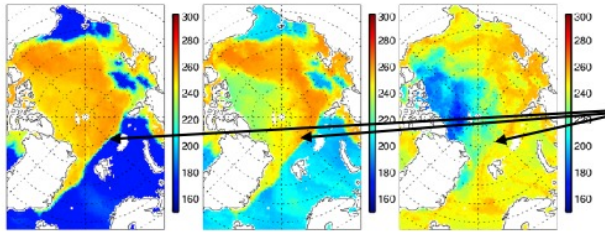
Dependent parameter

Trainable parameter

The whole trainable empirical-physical network

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

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

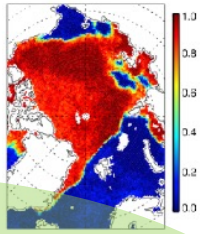
T_S $\mathbf{z}_A$

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



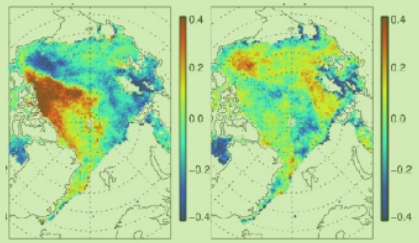
$\mathbf{C}_{\text{ice}}$ C_{ice} $C_{\text{ice}} = h(\mathbf{C}_{\text{ice}}, \mathbf{z}_G)$

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

$\mathbf{z}_G$ 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

Fixed parameter

Dependent parameter

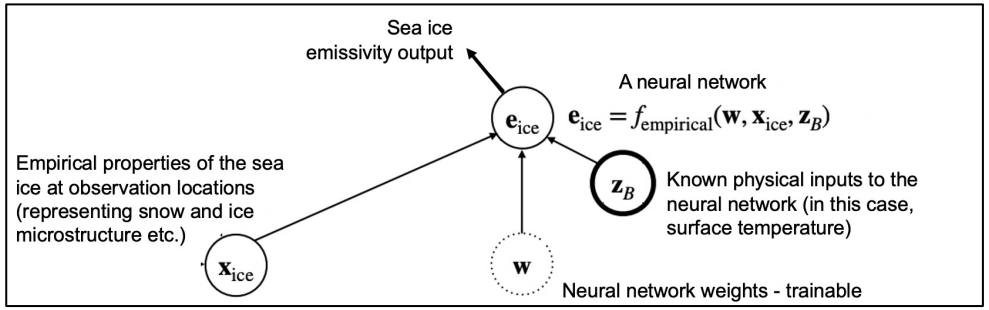
Trainable parameter

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

Latent space

Empirical model

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
```

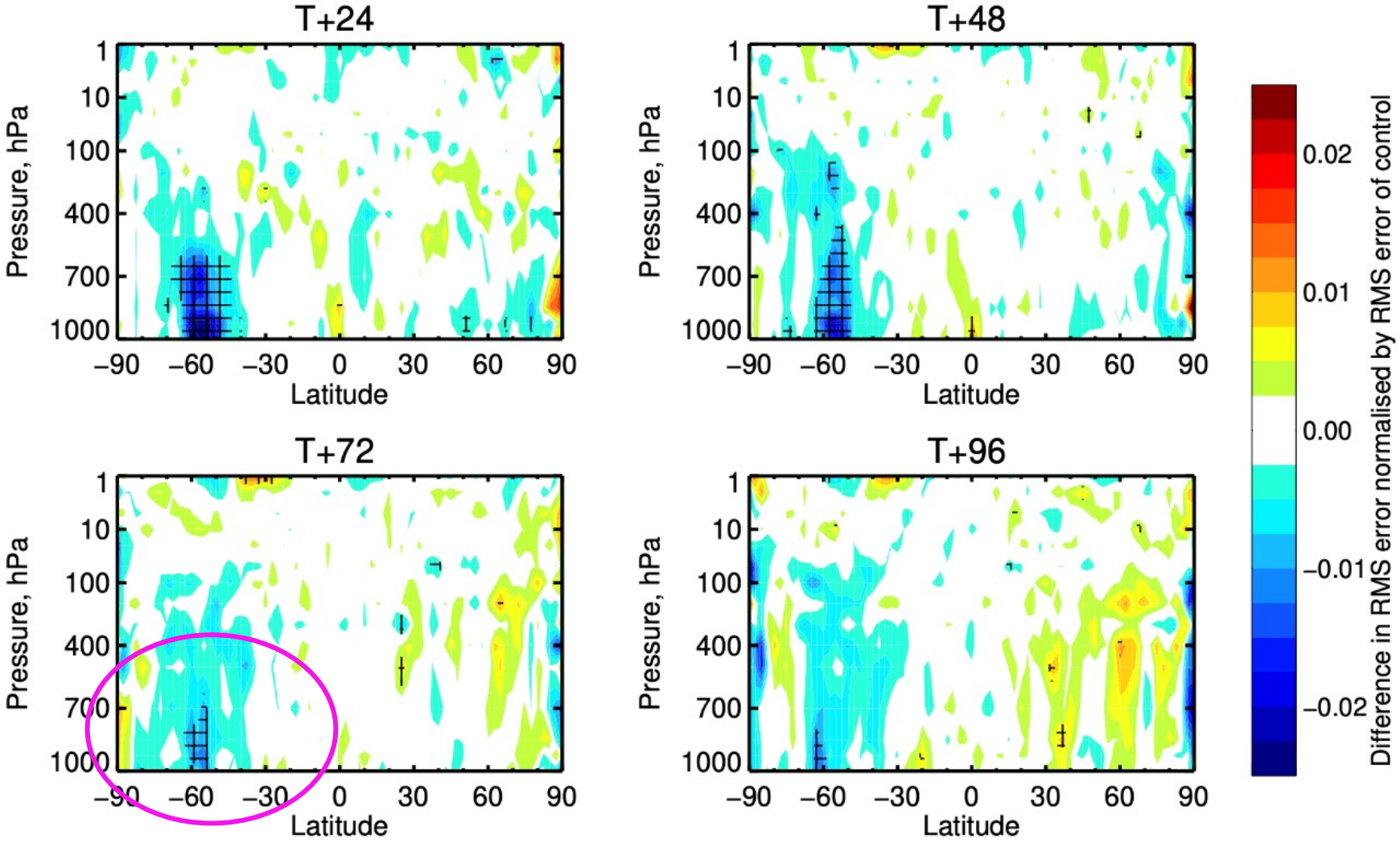
A standard dense neural network layer with linear activations

Custom loss functions to regularise / constrain the solution

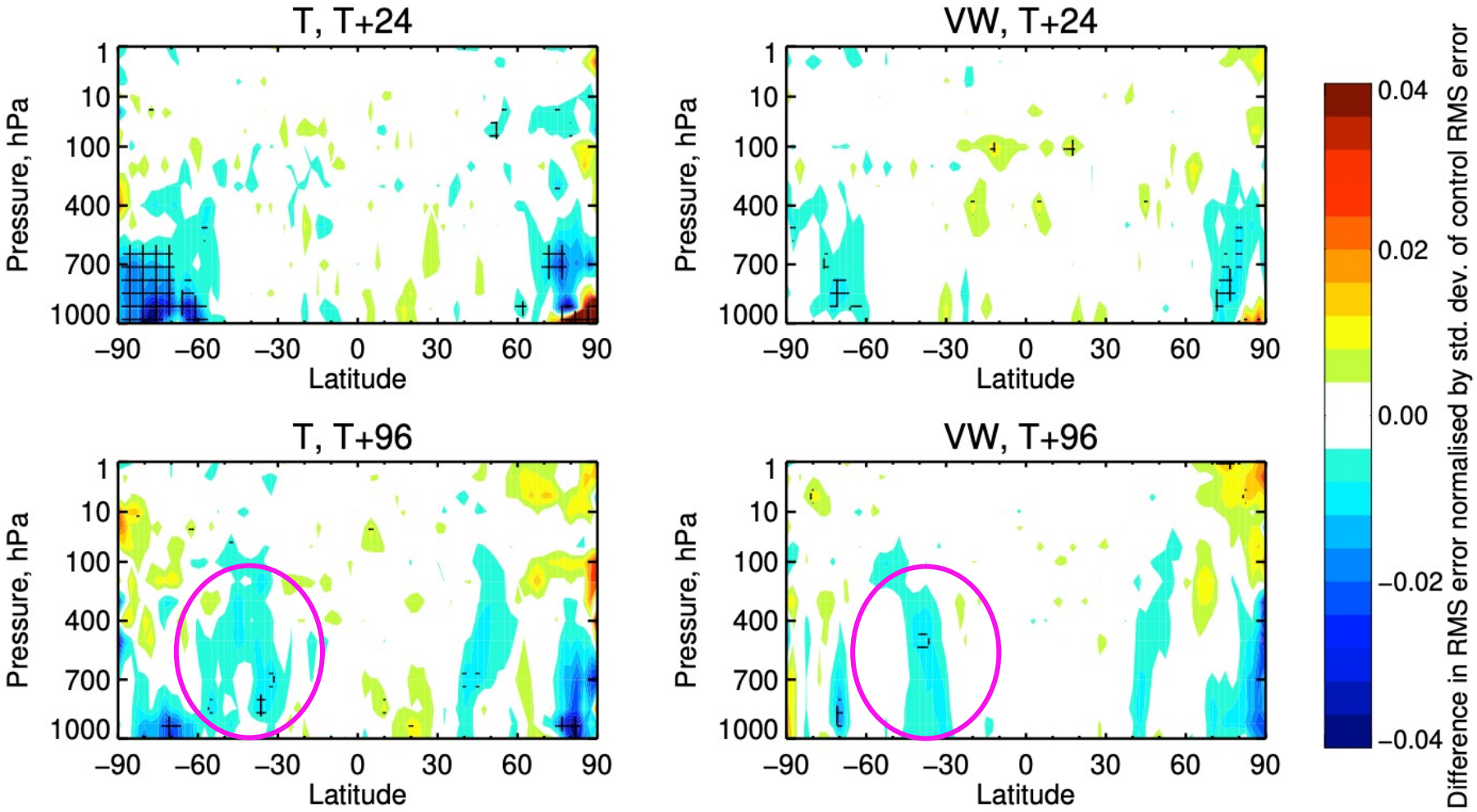
https://github.com/ecmwf-projects/empirical-state-learning-seaice-emissivity-model/blob/master/seaice_layers.py

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



Additional forecast impact of then assimilating pseudo-observations of sea ice concentration in the NEMO sea ice model



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 (autumn 2025)
- Further reading:
  - https://doi.org/10.1002/qj.4797
  - https://doi.org/10.1029/2023MS004080

How does machine learning affect DA in the future?

- Will end-to-end data-driven forecasting provide a complete replacement for DA and physical forecast models?
  - It can only forecast what is observed (e.g. radiances, not sea ice concentration)
  - It discards the idea of prior knowledge expressed in physical equations
- Does traditional physically-based DA continue unaffected?
  - Physical DA systems remain to provide training data / reference (e.g. ERA5)
    - Use this to train faster ML-based DA and forecast model replacements?
- Or hybrid data assimilation – machine learning?
  - ML for error correction of the physical model
  - ML for acceleration of slower tasks within a physical DA framework (e.g. ensemble members, background error covariances, observation operators, TL/AD ...)
  - Mixing physical and empirical components at a more granular scale (e.g. sea ice example)
- Even if ML takes over, it is likely to take on increasingly DA-like concepts
  - E.g. Physical constraints; observation errors