

Machine Learning applications in meteorological forecasting - classics and where federated learning could be useful

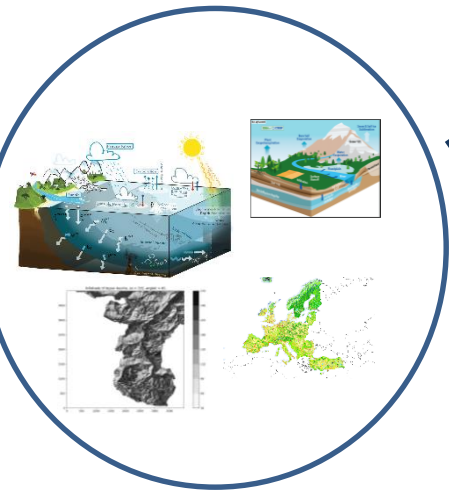
I. Schicker ZAMG



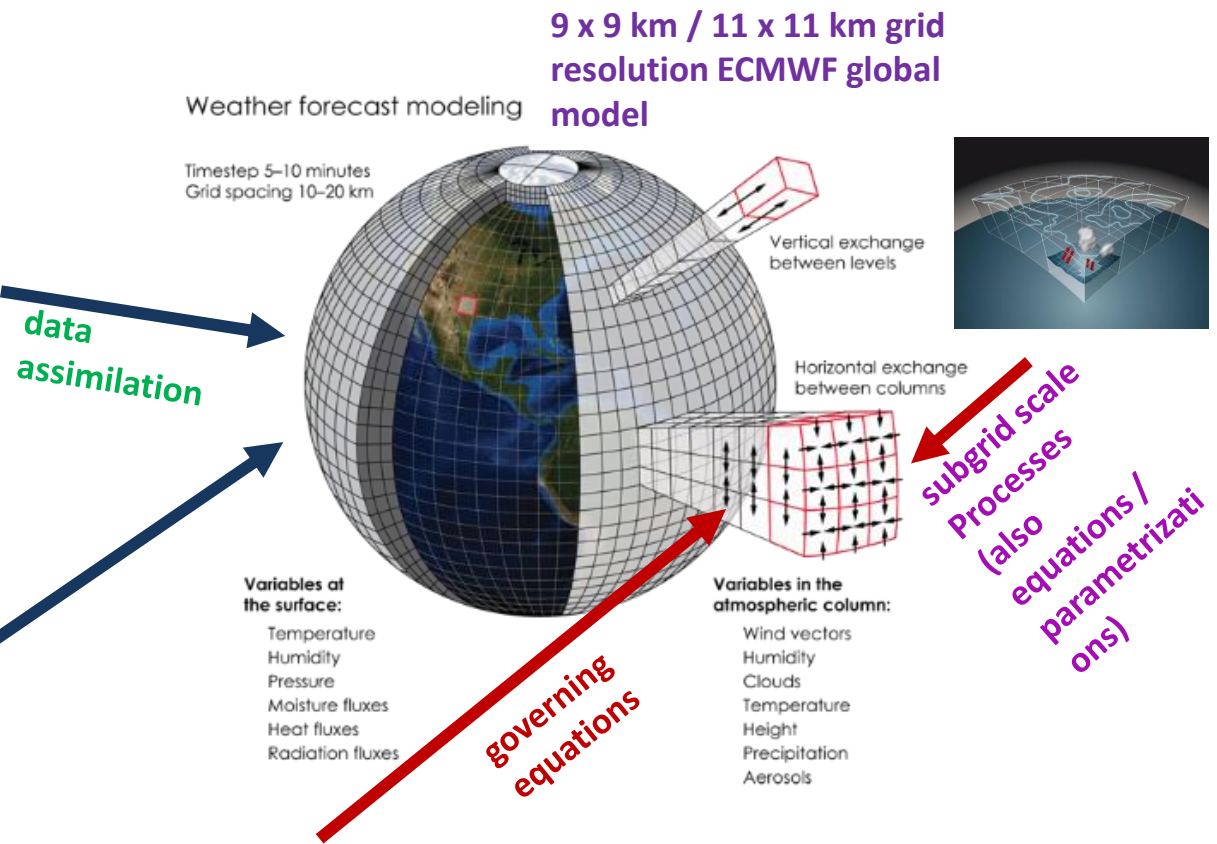
ZAMG
Zentralanstalt für
Meteorologie und
Geodynamik

Numerical weather prediction and weather forecasting

Observations



Ocean and surface model Static fields



"Primitive" Weather Forecasting Equations

$p = \rho R T$ Ideal Gas Law (Equation of State)

$\vec{a}_h = \sum \left(\frac{\vec{F}_h}{m} \right)$ Newton's Second Law of Motion $\Delta p = -\rho g \Delta z$ (PGA) $\vec{v} = g$

$\vec{a}_v = \sum \left(\frac{\vec{F}_v}{m} \right) = (\vec{P}\vec{G}\vec{A})_v - \vec{g}$

Hydrostatic Law (Obtained from the Equation of Vertical Motion)

$\Delta T = \Delta q c_p + (1/\rho) \Delta p$ First Law of Thermodynamics

Conservation of Mass Applied to the Atmosphere (Equation of Continuity)

Zonal wind: $\frac{\partial u}{\partial t} = \eta \frac{\partial u}{\partial x} - \frac{\partial \phi}{\partial x} \frac{\partial u}{\partial x} - c_p \frac{\partial \phi}{\partial x} \frac{\partial u}{\partial x} - \frac{\partial u}{\partial x} \frac{\partial (u^2 + v^2)}{\partial x}$

Meridional wind: $\frac{\partial v}{\partial t} = -\eta \frac{\partial v}{\partial y} - \frac{\partial \phi}{\partial y} \frac{\partial v}{\partial x} - c_p \frac{\partial \phi}{\partial y} \frac{\partial v}{\partial x} - \frac{\partial v}{\partial x} \frac{\partial (u^2 + v^2)}{\partial y}$

Temperature: $\frac{\partial T}{\partial t} = \eta \frac{\partial T}{\partial x} + \frac{\partial T}{\partial x} \frac{\partial T}{\partial y} + v \frac{\partial T}{\partial y} + w \frac{\partial T}{\partial z}$

Precipitation rate: $\frac{\partial W}{\partial t} = u \frac{\partial W}{\partial x} + v \frac{\partial W}{\partial y} + w \frac{\partial W}{\partial z}$

Pressure thickness: $\frac{\partial \phi}{\partial t} = u \frac{\partial \phi}{\partial x} + v \frac{\partial \phi}{\partial y} + w \frac{\partial \phi}{\partial z}$

$\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + \omega \left(\frac{\partial T}{\partial p} + \frac{RT}{pc_p} \right) = \frac{J}{c_p} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial \omega}{\partial p} = 0 \quad 0 = -\frac{\partial \phi}{\partial p} - \frac{RT}{p}$

Weer- en klimaatmodellen



Goed informeren en waarschuwen over weer en klimaat kan niet zonder goede computermodellen. Zij vormen het onmisbare gereedschap waarmee weersverwachtingen en klimaatscenario's worden gemaakt. Het KNMI werkt voortdurend aan het verbeteren van deze modellen, aangepast aan de nieuwste inzichten en technologie. Maar hoe werkt zo'n model?

1 Wat is een model?

De zon verwarmt de aarde. Rond de evenaar wordt het warmer dan aan de polen. Dit veroorzaakt grootschalige luchtstromen en verplaatsing van vocht en warmte in de atmosfeer. Deze weer- en klimaatprocessen worden nagebootst in numerieke modellen.

2 Berekenen

In het model is de atmosfeer opgedeeld in gridcellen



Per gridcel worden grootheden bijgehouden als:

- wind
- temperatuur
- straling
- druk
- vocht
- etc.

De waarden veranderen voortdurend: straling wordt gereflecteerd, water verdampt, turbulentie zorgt voor menging enz. Deze veranderingen worden in het model berekend met modules die de fysische processen beschrijven.



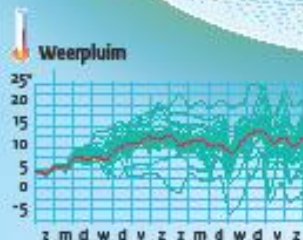
Het model doet zo'n berekening in stapjes van 60 sec



3 Weersverwachting

Om een weersverwachting te maken worden de beginwaarden van de grootheden in elke gridcel afgeleid uit waarnemingen van weersatellieten, grondstations, weerballonnen en andere metingen. De verwachting bestaat uit een weerbeeld en het moment waarop dat optreedt.

Metingen en model zijn niet perfect. Een kleine afwijking van de begintoestand leidt tot de berekening van een ander weerbeeld. Door de begintoestand en de fysische parameters steeds iets te wijzigen ontstaat een weerpluim.



Smalle pluim: redelijk zekere weersverwachting
Gewaaierde pluim: verwachting onzeker

© KNMI 2022
Meer informatie: www.knmi.nl

In een kolom gridcellen zitten rekenmodules voor condensatie, neerslag, straling, turbulentie, verdamping en oppervlakteprocessen.

4 Klimaatscenario's

Voor klimaatsimulaties rekent het model ver vooruit, tientallen tot honderden jaren. Hiervoor zijn ook externe factoren van belang, zoals de toename van de hoeveelheid broeikasgassen in de atmosfeer. Bij klimaatscenario's gaat het om de veranderingen van de kars op een weerbeeld en niet om het precieze tijdstip ervan. Daarom kan veel verder in de toekomst worden gekeken.



Modellen die het KNMI gebruikt

EC-Earth
Wereldwijd model gebaseerd op de natuurkunde van het ECMWF-model. Voor klimaatsimulaties van eeuwen vooruit wordt een grid van 80 x 80 km gebruikt.

RACMO
Fijnmazig regionaal model (grid 12 x 12 km) voor doorvertaling van klimaatsimulaties naar het Europa, waarvan de waarden worden berekend met EC-Earth.

ECMWF
Wereldwijd model van het Europese Weercentrum in Reading (GB). Voor de verwachting van twee weken vooruit wordt een grid van 9 x 9 km gebruikt (ca. 600 vakjes voor Nederland).



Harmonie
Model voor Nederland en omgeving. Sinds 2012 ingezet voor de verwachting van twee dagen vooruit met vakjes van 2,5 x 2,5 km (=10.000 voor Nederland).

Supercomputer

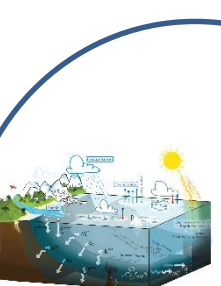
Harmonie vraagt een zeer groot aantal berekeningen in korte tijd. Om 8 x per dag een verwachting te maken beschikken het KNMI en de Deense, Ierse en Amerikaanse meteorologische diensten over een gezamenlijke supercomputer in Reykjavik met een rekenkracht van 4000 biljoen berekeningen/sec. (4 petaflop). Ook voor klimaatberekeningen worden supercomputers gebruikt.



Observations



...



Ocean and surface
Static fields

Numerical weather prediction and weather forecasting

Observations

- The Earth is huge and ranges from flat to rugged
- We cannot resolve every process explicitly
- The system is chaotic
- Some processes are not well understood
- All components are connected in a non-trivial way
- We have a HUGE number of observations to deal with and even more NWP/climate model data

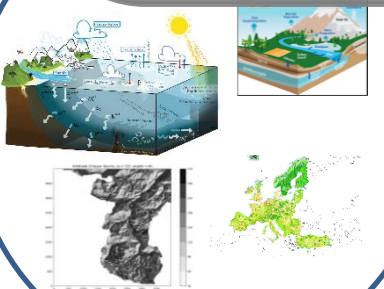
9 x 9 km / 11 x 11 km grid resolution ECMWF global

Vertical exchange between levels

Horizontal exchange between columns

sub-grid scale processes (also equations / parametrizations)

governing equations



Ocean and surface model
Static fields

"Primitive" Weather Forecasting Equations

$p = \rho R T$ Ideal Gas Law (Equation of State)

$\vec{a}_h = \sum \left(\frac{\vec{F}_h}{m} \right)$ Newton's Second Law of Motion $\Delta p = -\rho g \Delta z$ (PGA) $\vec{v} = \vec{g}$

$\vec{a}_v = \sum \left(\frac{\vec{F}_v}{m} \right) = (\vec{P}\vec{G}\vec{A})_v - \vec{g}$

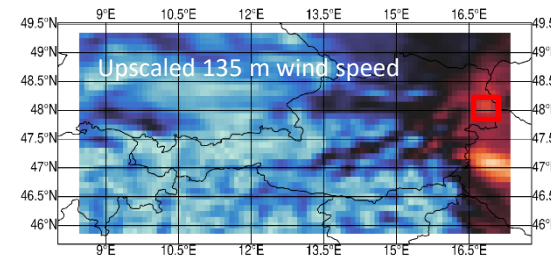
Hydrostatic Law (Obtained from the Equation of Vertical Motion)

$\Delta T = \Delta q c_p + (1/\rho) \Delta p$ First Law of Thermodynamics

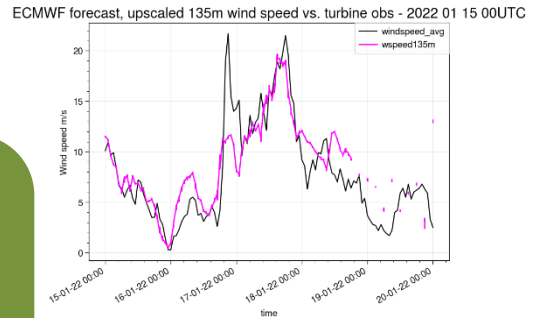
Conservation of Mass Applied to the Atmosphere (Equation of Continuity)

$$\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + \omega \left(\frac{\partial T}{\partial p} + \frac{RT}{pc_p} \right) = \frac{J}{c_p} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial \omega}{\partial p} = 0 \quad 0 = -\frac{\partial \phi}{\partial p} - \frac{RT}{p}$$

Machine Learning (?)

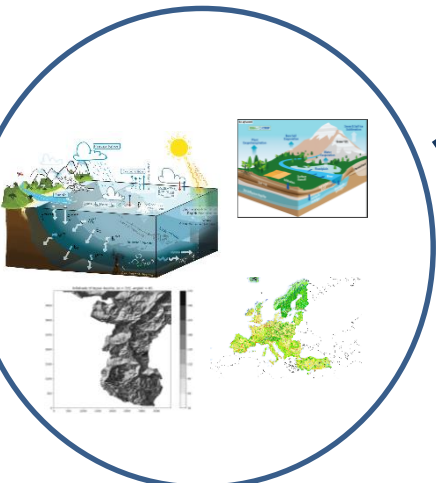


Post-processing



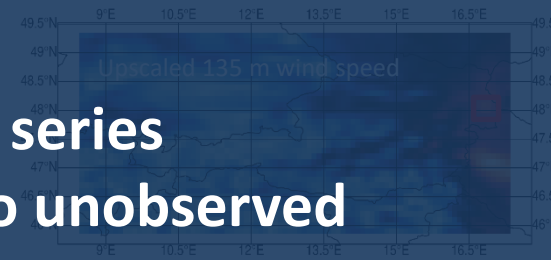
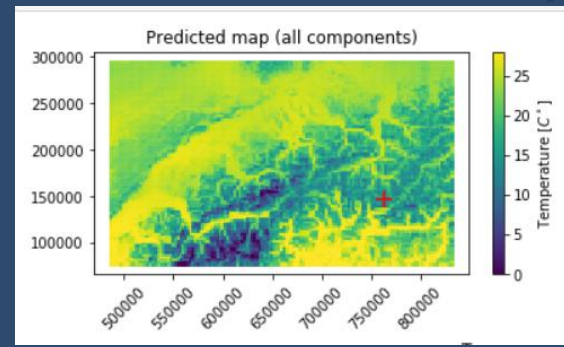
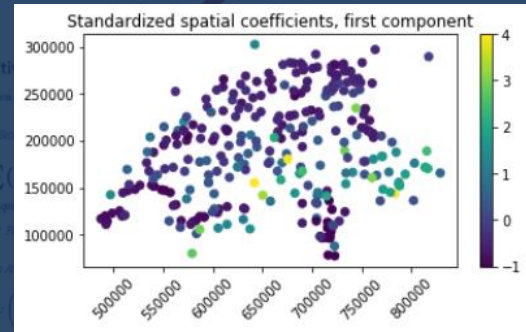
Numerical weather prediction and weather forecasting – and machine learning (?)

Observations



Ocean and surface model
Static fields

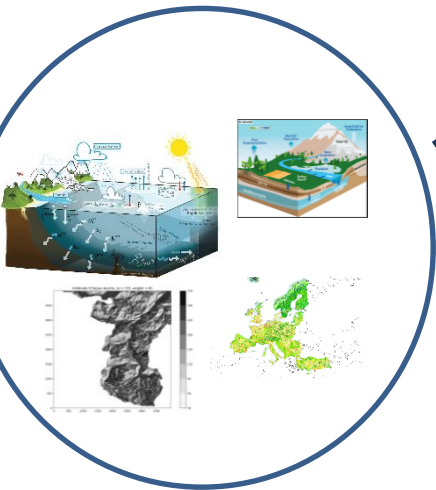
- Data monitoring
- Real time quality control
- Anomaly detection
- Data cleaning and filtering for longer, historic time series
- Observation spatial interpolation / interpolation to unobserved areas
- Data fusion of different sources
- Guided decision making
- Correction of observation error
- Filling of missing values in time series



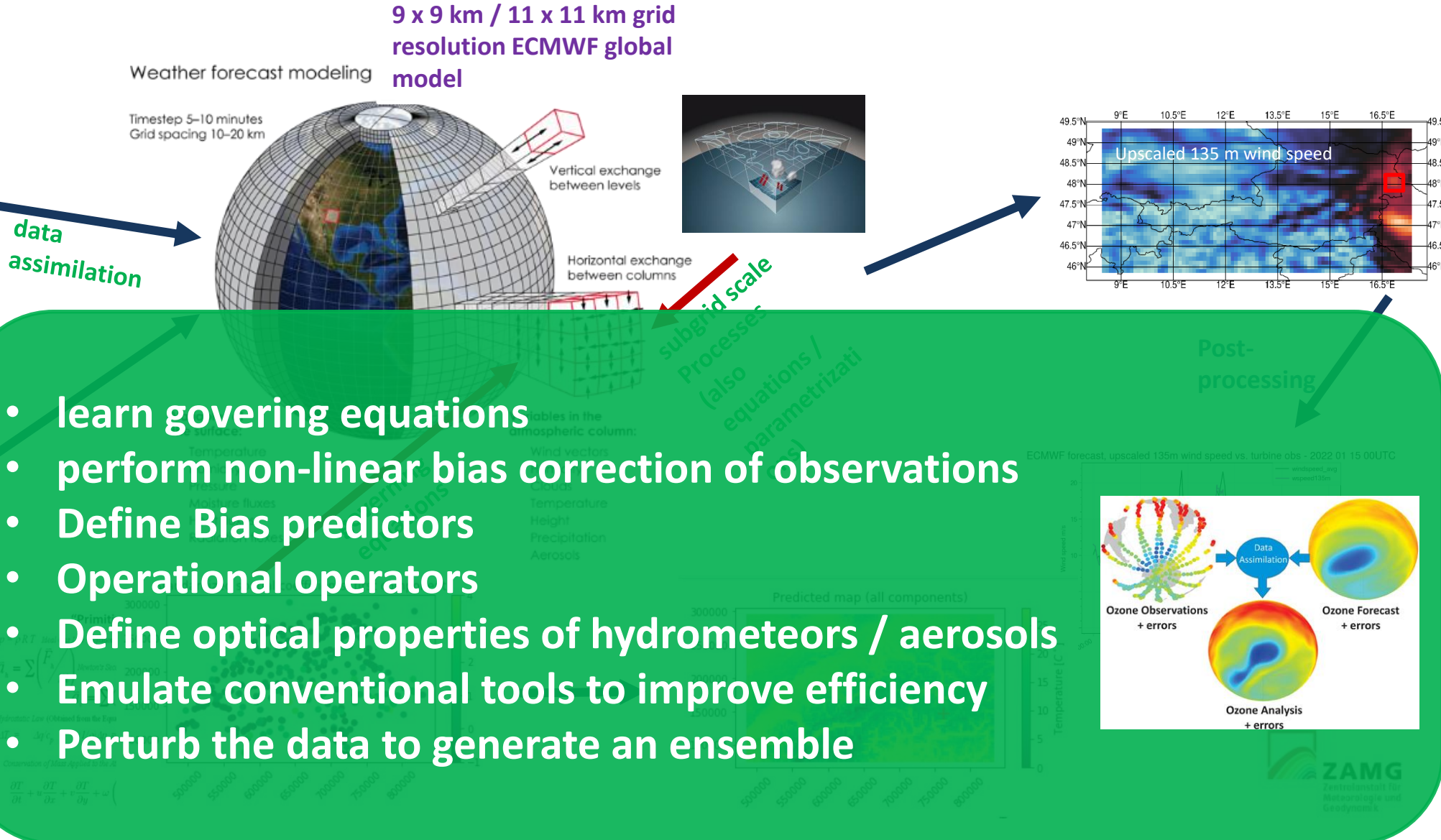
Numerical weather prediction and weather forecasting – and machine learning (?)



Observations



Ocean and surface model
Static fields



- learn governing equations
- perform non-linear bias correction of observations
- Define Bias predictors
- Operational operators
- Define optical properties of hydrometeors / aerosols
- Emulate conventional tools to improve efficiency
- Perturb the data to generate an ensemble

Numerical weather prediction and weather forecasting – and machine learning (?)

Observations

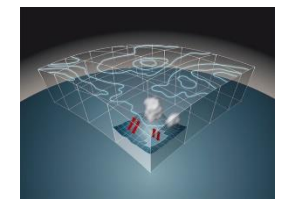
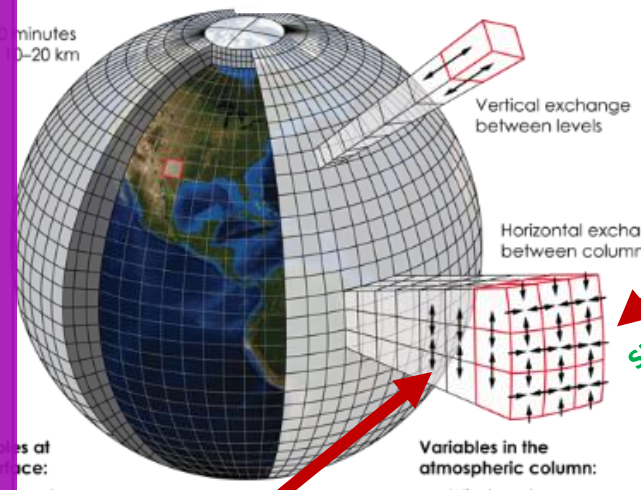
- Emulate model components
- Improve existing parametrizations
- Replace parts of parametrizations
- Better error models
- Learn equations of motion
- Generate conjoined code

data assimilation

Weather forecast modeling model

Timestep 5-10 minutes
Grid spacing 10-20 km

9 x 9 km / 11 x 11 km grid resolution ECMWF global model



subgrid scale Processes (also equations / parametrizations)

Variables at the surface:
Temperature
Humidity
Pressure
Moisture fluxes
Heat fluxes
Radiation fluxes

Variables in the atmospheric column:
Wind vectors
Humidity
Clouds
Temperature
Height
Precipitation
Aerosols

Governing equations

"Primitive" Weather Forecasting Equations

Zonal wind:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - c_y \frac{\partial v}{\partial x} - \frac{\partial u}{\partial z} \frac{\partial (v^2 + u^2)}{\partial z}$$

Meridional wind:

$$\frac{\partial v}{\partial t} - u \frac{\partial v}{\partial x} - c_x \frac{\partial u}{\partial y} - \frac{\partial v}{\partial z} \frac{\partial (v^2 + u^2)}{\partial z}$$

Temperature:

$$\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + w \frac{\partial T}{\partial z}$$

Precipitation rate:

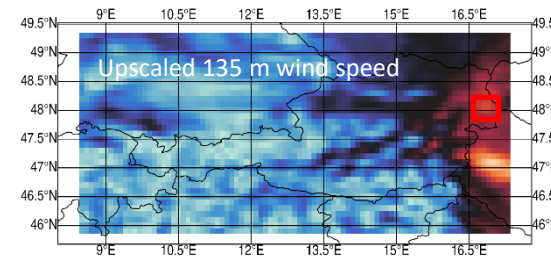
$$\frac{\partial W}{\partial t} = u \frac{\partial W}{\partial x} + v \frac{\partial W}{\partial y} + w \frac{\partial W}{\partial z}$$

Pressure thickness:

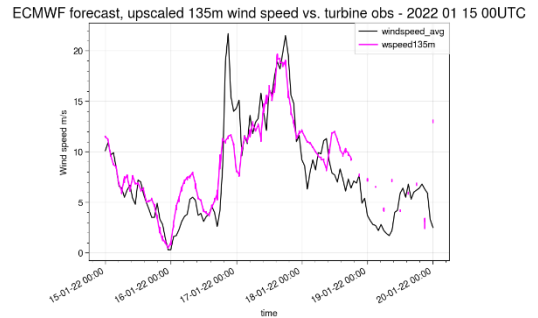
$$\frac{\partial \sigma}{\partial t} + u \frac{\partial \sigma}{\partial x} + v \frac{\partial \sigma}{\partial y} + w \frac{\partial \sigma}{\partial z} = -\frac{\partial \sigma}{\partial p} \frac{\partial p}{\partial t}$$

Continuity of Mass applied to the atmosphere (Equation of Continuity):

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \quad 0 = -\frac{\partial \phi}{\partial p} - \frac{RT}{p}$$



Post-processing



Ocean and surface model
Static fields

Numerical weather prediction and weather forecasting – and machine learning (?)



Observations

- Adjustments of forecast products for renewable energy applications, nowcasting/forecasting of severe weather
- Improvements of forecast products for sub-seasonal to seasonal prediction
- Feature detection
- Uncertainty quantification and „cheap“ ensembling
- Low complexity models for research purposes
- Data driven forecasting
- Generation of synthetic data / data augmentation for algorithm training
- Increasing of spatial-temporal resolution (< 1 km, < hourly)

Weather forecast modeling

9 x 9 km / 11 x 11 km grid resolution ECMWF global model

Timestep 5–10 minutes

Vertical exchange between levels

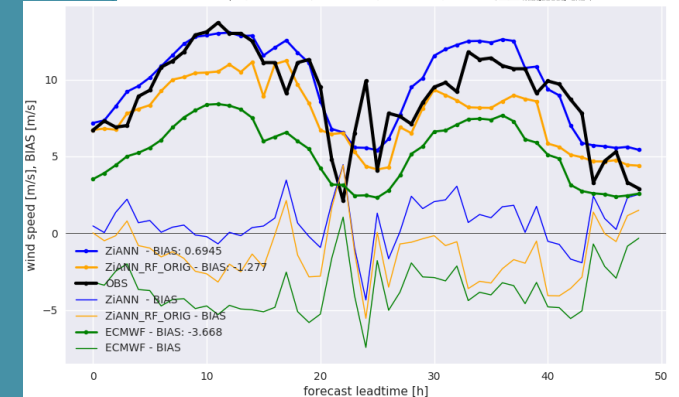
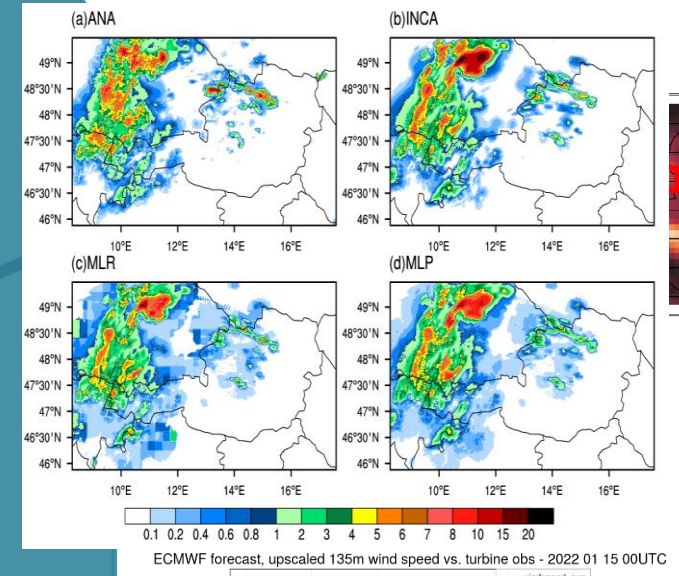
between columns

subgrid scale processes (also equations / parametrizations)

Governing equations

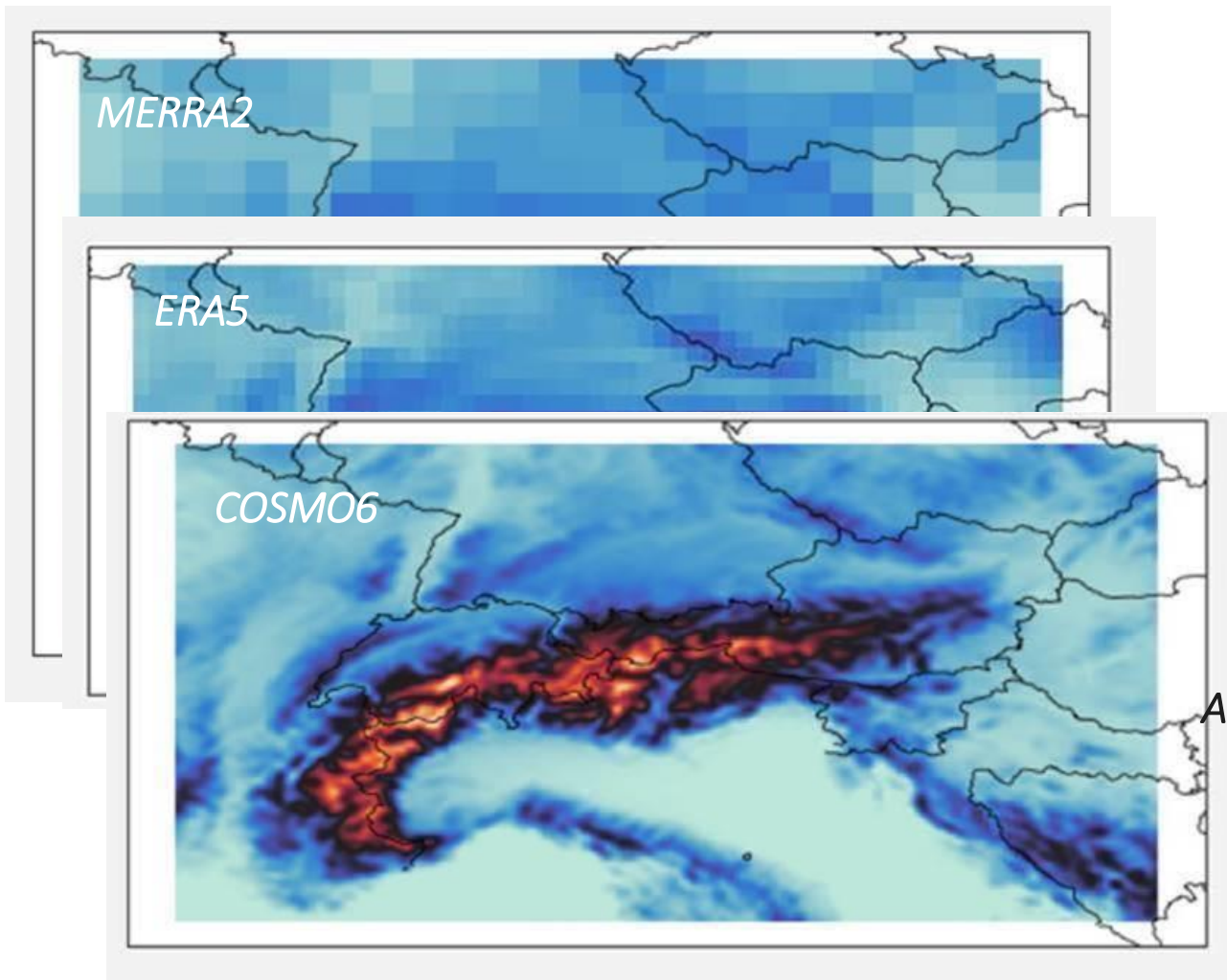
“Primitive” Weather Forecasting Equations

$$\frac{\partial T}{\partial t} + u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} + w \left(\frac{\partial T}{\partial p} + \frac{RT}{p^2} \right) = \frac{1}{\rho} \left(\frac{\partial \sigma}{\partial x} + \frac{\partial \tau}{\partial y} + \frac{\partial \omega}{\partial p} \right) + \frac{RT}{p^2} \left(\frac{\partial \sigma}{\partial x} + \frac{\partial \tau}{\partial y} + \frac{\partial \omega}{\partial p} \right)$$
$$\frac{\partial \sigma}{\partial t} + u \frac{\partial \sigma}{\partial x} + v \frac{\partial \sigma}{\partial y} + w \left(\frac{\partial \sigma}{\partial p} + \frac{RT}{p^2} \right) = \frac{1}{\rho} \left(\frac{\partial \sigma}{\partial x} + \frac{\partial \tau}{\partial y} + \frac{\partial \omega}{\partial p} \right) + \frac{RT}{p^2} \left(\frac{\partial \sigma}{\partial x} + \frac{\partial \tau}{\partial y} + \frac{\partial \omega}{\partial p} \right)$$

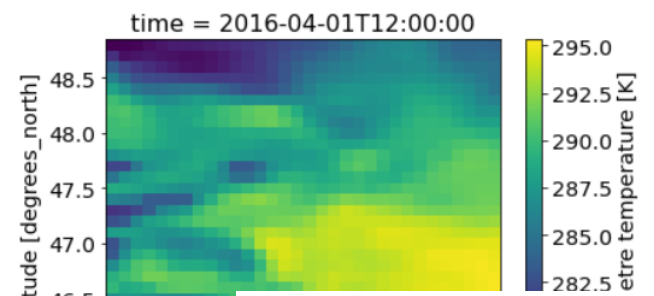


Ocean and surface model
Static fields

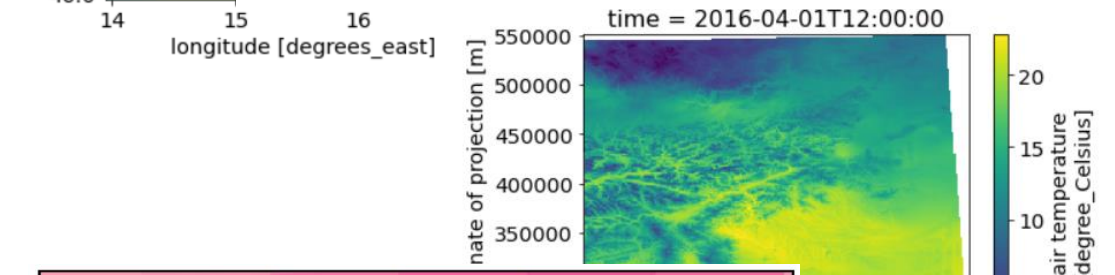
Post-processing – why is it important



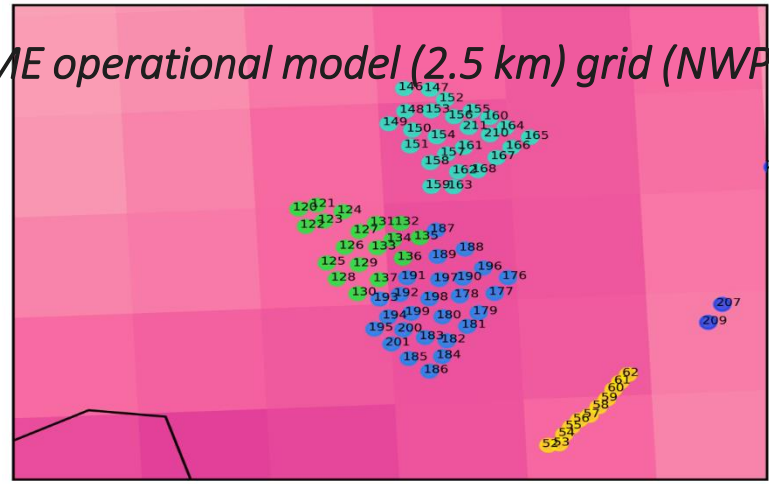
ECMWF operational global model (~ 9 km) – Eastern Austria



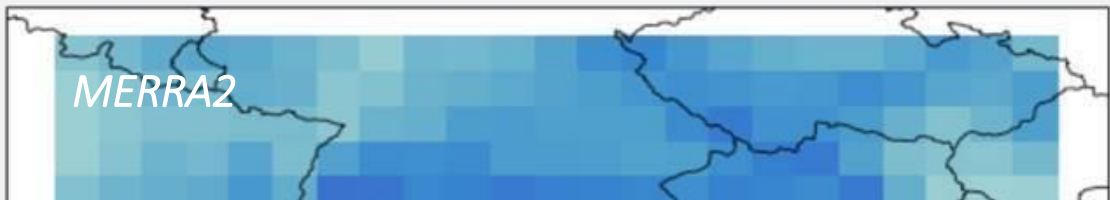
INCA operational analysis 1 km – Eastern Austria



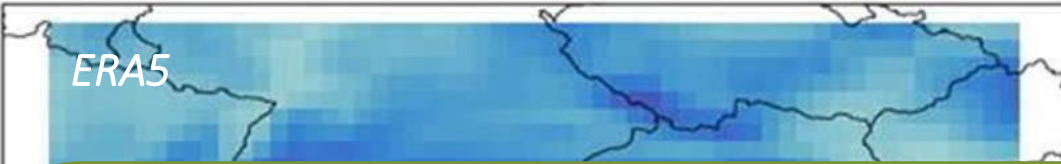
AROME operational model (2.5 km) grid (NWP model)



Post-processing – why is it important



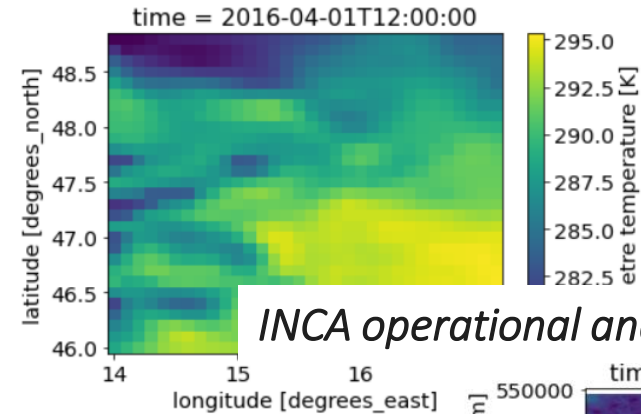
MERRA2



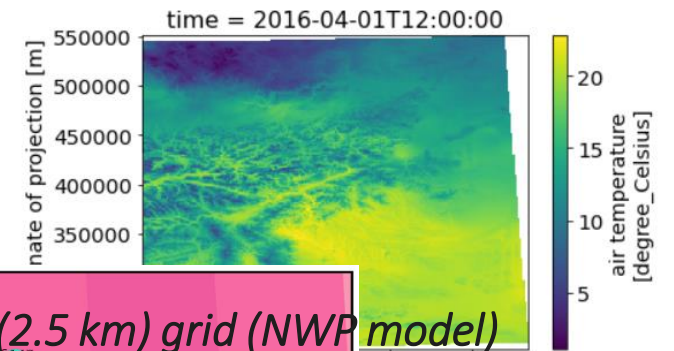
ERA5

- Grid resolution too coarse
- Temporal resolution for some applications too coarse
- Temporal horizon not long enough
- NWP models prone to different error sources (data, assimilation, parametrizations), bias / spread need to be corrected

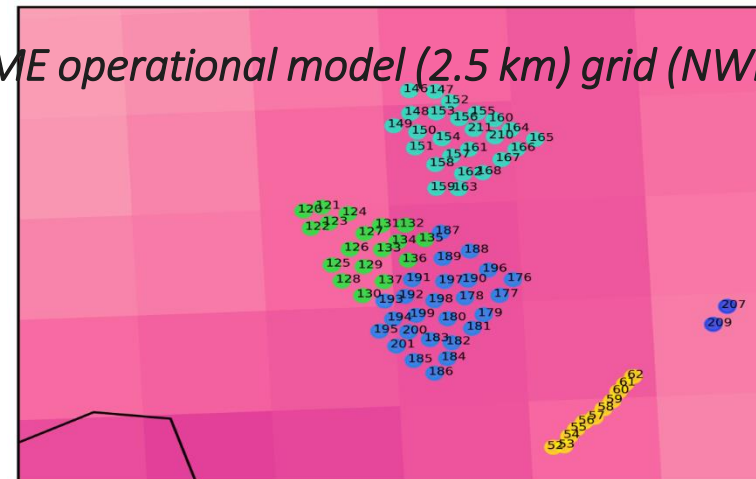
ECMWF operational global model (~ 9 km) – Eastern Austria



INCA operational analysis 1 km – Eastern Austria



AROME operational model (2.5 km) grid (NWP model)





Post-processing – „classical“ methods

- „classical“ as age/idea of methods is >= 50 years dating back to the 1960ies
- Are based on (current) observations and/or recent weather conditions and don't use numerical models ('data driven')
- Observations of the initial (predictor) and resultant (predictands) weather conditions are a must have
- Example: forecast the temperature for tomorrow with input consisting of observational data available at the time the forecast was issued:

$$\hat{Y}_t = \sum f_C(X_0) \dots\dots\dots (1)$$

Y_t = predictand (dependent variable) at time 't'

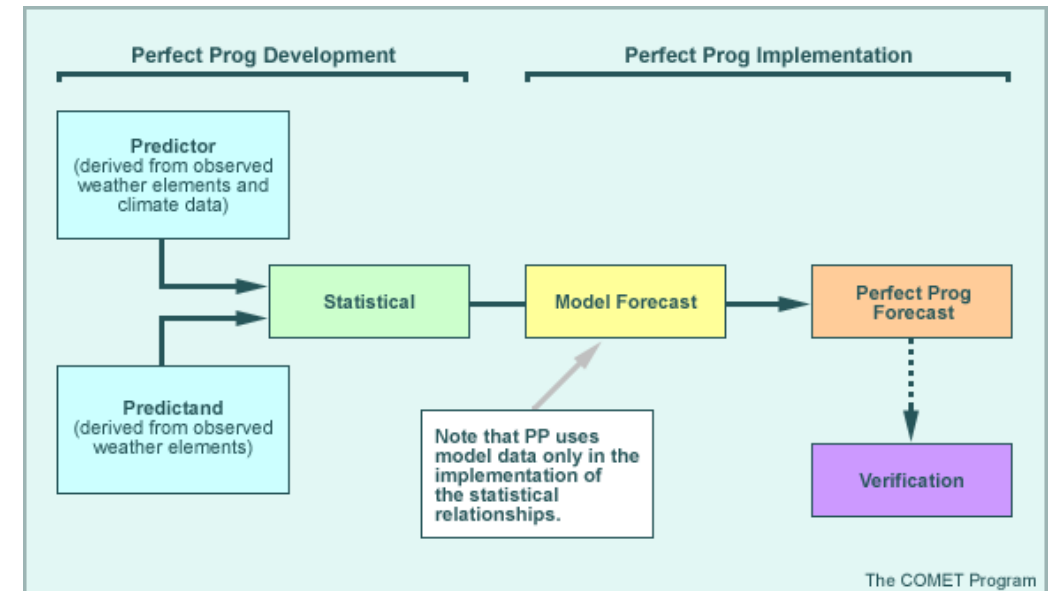
X_0 = predictor vector of observational data (independent variables) available at initial time 0

- Works good for short ranges and very long ranges, no skill in medium range (~24h to 10 days)
- Best under persistend weather conditions with low variability

Post-processing – „classical“ methods

Perfect prognosis / perfect prog

- The need to accurately predict surface weather elements, led to the development of Perfect Prognosis Method (PPM) (Klien et al., 1959).
- objective method, in which, a concurrent relation is developed between the parameter to be predicted and the observed circulation around the location of interest, using several years of data
- based on the assumption that numerical model forecasts are “perfect”
- numerical models are not perfect but this approach gives an estimate of what to expect, if numerical models are correct
- does not require numerical model data for development, uses numerical output when equations are applied operationally
- make sure that variables used as ‘Predictors’ in development of Perfect Prognosis Equations will be available from NWP models for operational purposes



Post-processing – „classical“ methods

Analog-based methods - AnEn

- the current state of the atmosphere is compared with a repository of other, historical states of the atmosphere to determine the most similar scenario in the past (an analog) (Van den Dool, 1989; Hamill and Whitaker, 2006; Delle Monache et al., 2011; Delle Monache et al., 2013).
- Lorenz (1969): analogues refer to “two states of the atmosphere which resemble each other rather closely” and “Each state may then be looked upon as equivalent to the other state plus a reasonably small ‘error’.”
- Are used in meteorology, analogs primarily for pre- and post- processing of NWP forecasts (Hamill and Whitaker, 2006).

$$\|F_t, A_{t'}\| = \sum_{i=1}^{N_v} \frac{w_i}{\sigma_{fi}} \sqrt{\sum_{j=-\tilde{t}}^{\tilde{t}} (F_{i,t+j} - A_{i,t'+j})^2}$$

F_t = forecast to be corrected at a given time *t* and specific station location;

A_{t'} = analog forecast at time *t'* before *F_t* is issued and at the same location.

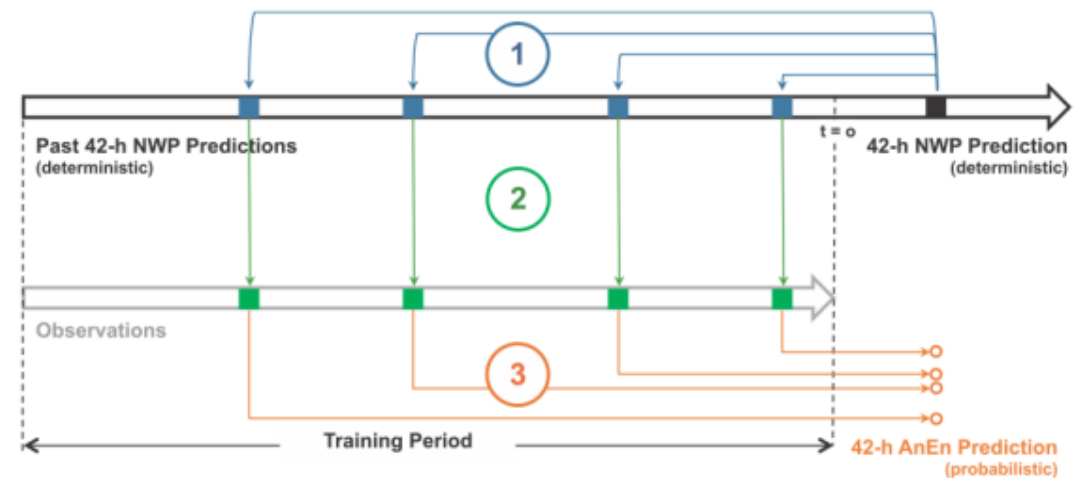
N_v, **w_i** are the number of predictors and their weights, respectively;

σ_{fi} = standard deviation of the time series of past forecasts of a given variable at the same location

ṡ = an integer equal to half the width of the time window over which the metric is computed.

F_{i,t+j} and **A_{i,t'+j}** = values of the prediction and the analog in the time window for a given variable.

This metric describes the quality of the analog chosen and is based upon the similarity of the current forecast window to the past forecast time windows available in the historical dataset. E.g., for a three-hour forecast the window would consist of three points, *t*-3hr, *t*, and *t*+3hr.



Delle Monache et al., 2013

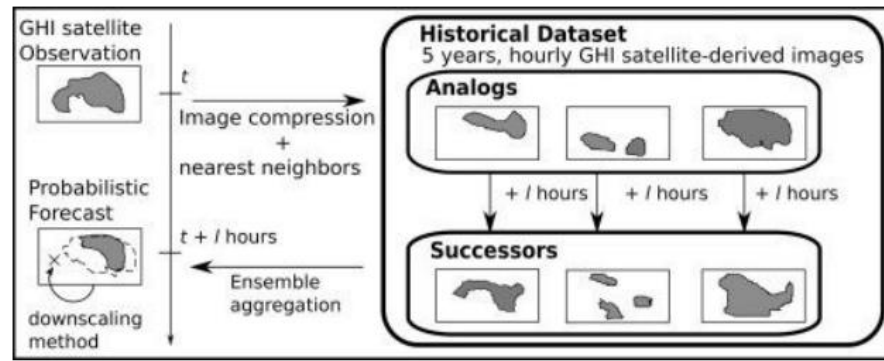
Hybrid analogs – neural networks (e.g. CapsNet)

Post-processing – „classical“ methods

Example application solar energy

Analog-based methods – „data-driven“ AnEn with spatial search field (satellite/radar/analysis field)

- Given a, e.g., satellite image of the current state of the atmosphere (the *observation* or *truth*), → search in a historical database N images that resemble the observation (the *analog*s)
- analog is found running a k-nearest neighbors algorithm on compressed images (into four *features*)

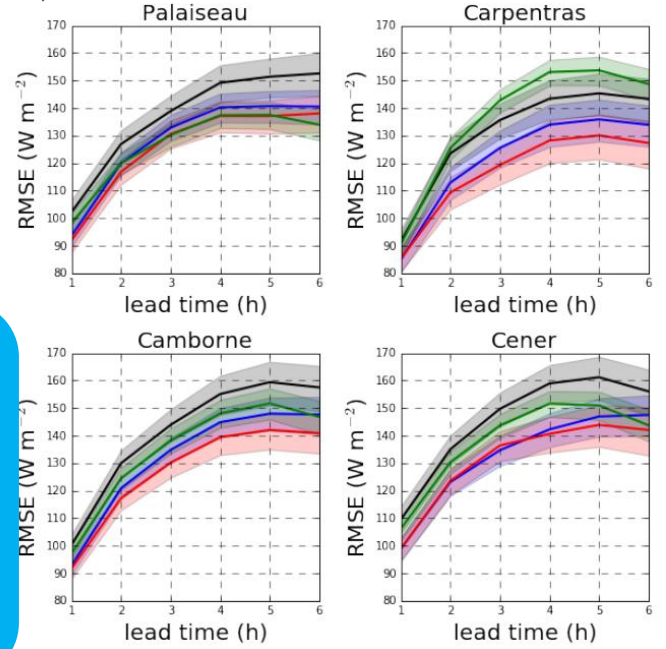
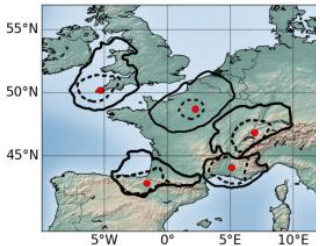


Ayet and Tandeo, 2018

```

219 def knn_analogs(self, obs, analog_zones, k=80):
220     """
221     Returns the k nearest neighbors of the observation obs from the dataset analog_zones. The results are cleaned to not include more t
222     :param obs: observation data 4D feature representation
223     :param analog_zones: analog neighbor candidates 4D feature representation
224     :param k: num of neighbors
225     :return: k nearest neighbors
226     """
227     neigh = NearestNeighbors(n_neighbors=k, metric='euclidean')
228     neigh = neigh.fit(analog_zones.to_array().to_numpy().transpose())
229     dists, indices = neigh.kneighbors(
230         obs.to_array().to_numpy().transpose(), 80, True)
231     analog_time = analog_zones.coords['time']
232     analogs = analog_time[indices.flatten()]
233     analogs = self.check_successors(analogs)
234     analogs_filter = list()
235     t_delta = np.timedelta64(12, 'h')
236     for d in analogs.values:
237         d_min = d - t_delta
238         d_max = d + t_delta
239         if not len(analogs_filter):
240             analogs_filter.append(d)
241         else:
242             tmp = True
243             for t in analogs_filter:
244                 if d_min <= t <= d_max:
245                     tmp = False
246             if tmp:
247                 analogs_filter.append(d)
248     return self.ds_lim.sel(
249         time=self.ds_lim.time.isin(analogs_filter)).load()

```



Not shown:
 - MOS
 - BMA
 - ECC /
 Schaake
 Shuffle
 - ...

Figure 13: Normalized "ground" RMSE and corresponding 95% bootstrap confidence interval as a function of lead time for the analog method (blue), the post-processed analog method (red), the persistence method (black), and the adaptive VAR(1) model (green).

Implemented in test version, currently adapted for operational purposes

Post-processing – hybrid methods



EMOS – ensemble model output statistics on point and grid

Uses:

- numerical weather prediction data, deterministic/probabilistic
- Observations of official weather obs site
- OR: gridded analysis fields
- Based on non-homogeneous gaussian regression
- Originally implemented in Fortran, rewritten in R and python

Boosting: rather machine learning than pure statistics

EMOS:

EMOS boost:

$$y \sim N(\mu, \sigma)$$

$$y \sim N(\mu, \sigma)$$

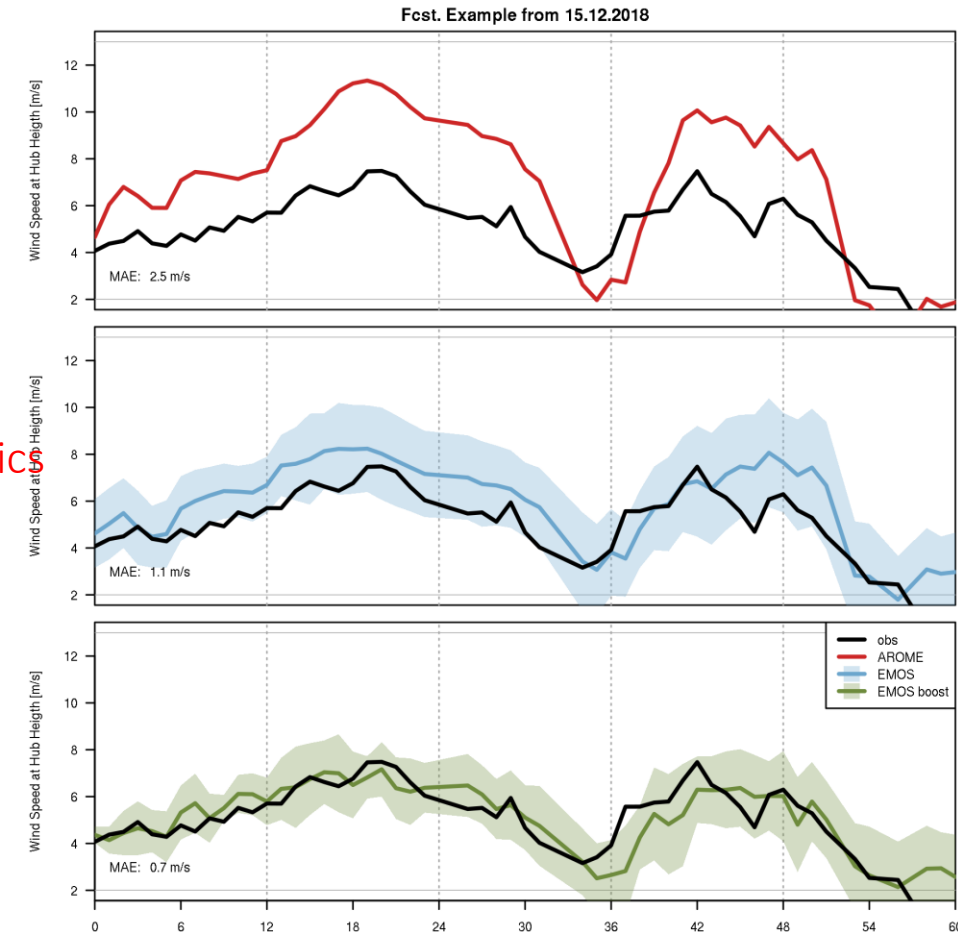
$$\mu = b_0 + b_1 m$$

$$\mu = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots$$

$$\sigma = c_0 + c_1 s$$

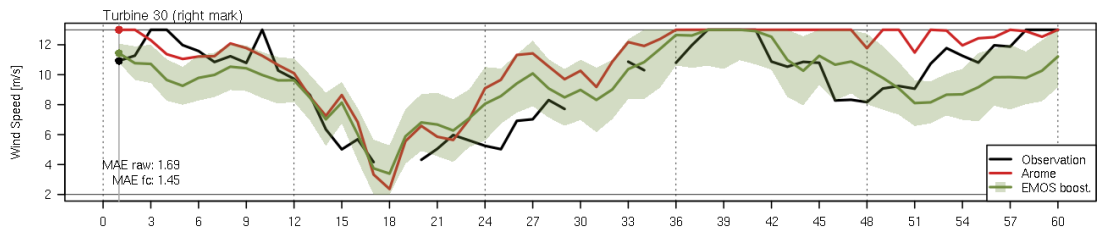
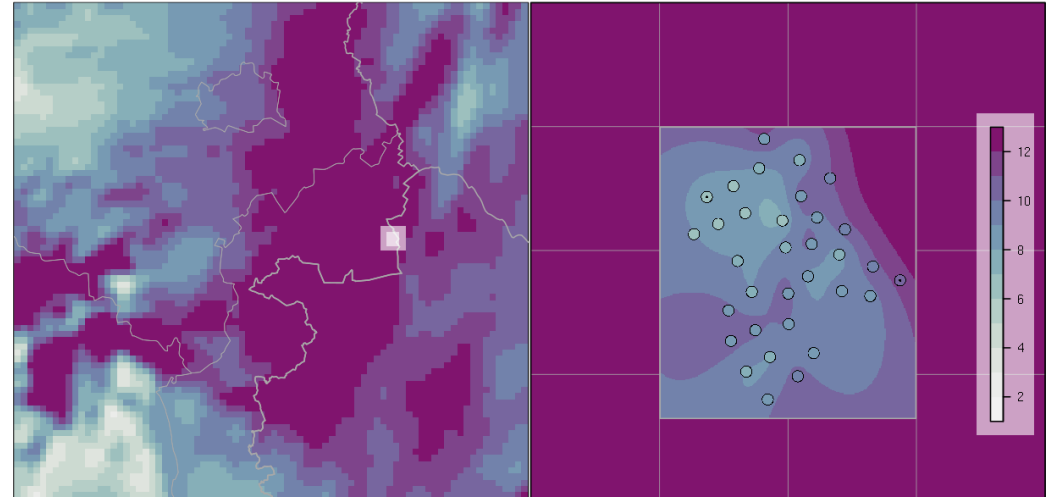
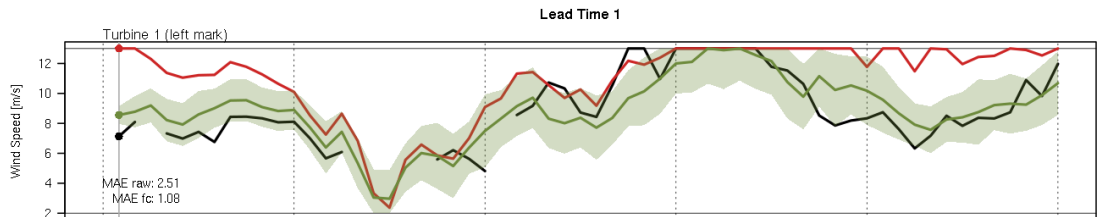
$$\sigma = c_0 + c_1 z_1 + c_2 z_2 + c_3 z_3 + \dots$$

Advantage: we get an uncertainty estimation on-the-fly with the statistical-based method



Post-processing – hybrid methods

```
974 DO istation = 1, numberofstations
975 ! loop over leadtimes
976 DO itime = 1, numberofleadtimes
977 ! if coefficients are missing values
978 IF (ANY(COEFFICIENTS(itime,istation,:) == missing_val .OR. RMSE(itime,istation) == missing_val .OR. EPSMEAN(itime,istation) == missing_val .OR. EPSSPREAD(itime,istation) == missing_val .OR. Fcst_eps_clibrated(:,itime,istation) == missing_val) .AND. .NOT. (FCST_EPS_CALIBRATED(:,itime,istation) == missing_val))
979 ELSE
980 ! assign coefficients
981 a = COEFFICIENTS(itime,istation,1)
982 b = COEFFICIENTS(itime,istation,2)
983 c = COEFFICIENTS(itime,istation,3)
984 d = COEFFICIENTS(itime,istation,4)
985 chosen_area = 0.9
986 tmpspread = 9999. ! Initialisation
987 ! calculate temporary eps mean
988 SELECT CASE (calibration_method)
989 CASE ('NGR')
990 tmpmean = (a+b*EPSMEAN(itime,istation))
991 CASE ('CORGR')
992 z2 = (a + b*EPSMEAN(itime,istation))/EPSSPREAD(itime,istation)
993 pdf_z2 = (1./SQRT(c*PI))*exp(-1.)*(z2**2./2.)
994 CALL NORCDF(z2, cpdf_z2)
995 tmpmean = (a+b*EPSMEAN(itime,istation))*cpdf_z2 + EPSSPREAD(itime,istation)*pdf_z2
996 END SELECT
997 ! do while (chosen_area.GE.0.01.AND.tmpspread.GE.(RMSE(itime,istation)*Fresc))
998 tmpspread = 0.
999 ! loop over ensemble members
1000 DO IMEM = 1, MEM_MAX
1001 ! probability = (1+IMEM/(1.*MEM_MAX+1))
1002 probability = (1+IMEM/(1.*MEM_MAX+1))
1003 probability = 0.5-chosen_area/2. + (float(IMEM-1))*(chosen_area/(float(MEM_MAX-1)))
1004 XTMP1(IMEM) = dinvnorm_standard(probability)
1005 ! end loop over ensemble members
1006 ENDDO
1007 ! calculate temporary version of calibrated ensemble
1008 XTMP1 = XTMP1*(c+d*EPSSPREAD(itime,istation)**2.)+(a+b*EPSMEAN(itime,istation))
1009 XTMP1 = XTMP1*(c+d*EPSSPREAD(itime,istation)**2.)+tmpmean
1010 ! calculate temporary spread
1011 tmpspread = SQRT(SUM(XTMP1-tmpmean)**2.)/FLOAT(MEM_MAX-1)
1012 chosen_area=chosen_area-0.01
1013 ENDDO
1014 ! assign final calibrated ensemble
1015 SELECT CASE (calibration_method)
1016 CASE ('NGR')
```



Python:
https://github.com/slerch/ppnn/blob/master/nn_postprocessing/n_src/emos_network_theano.py

R package:
[ensembleMOS: EMOS modeling in ensembleMOS: Ensemble Model Output Statistics \(rdrr.io\)](https://github.com/slerch/ppnn/blob/master/nn_postprocessing/n_src/emos_network_theano.py)

Post-processing – hybrid methods

SAMOS – standardized anomalies based model output statistics

$$y \sim N(\mu, \sigma),$$

$$\mu = \beta_0 + f_1(\text{doy}),$$

$$\log(\sigma) = \gamma_0 + g_1(\text{doy}),$$



$$y^* = \frac{y - \mu_y}{\sigma_y},$$

$$m^* = \text{mean} \left(\frac{\text{ens} - \mu_{\text{ens}}}{\sigma_{\text{ens}}} \right),$$

$$s^* = \text{std dev} \left(\frac{\text{ens} - \mu_{\text{ens}}}{\sigma_{\text{ens}}} \right),$$

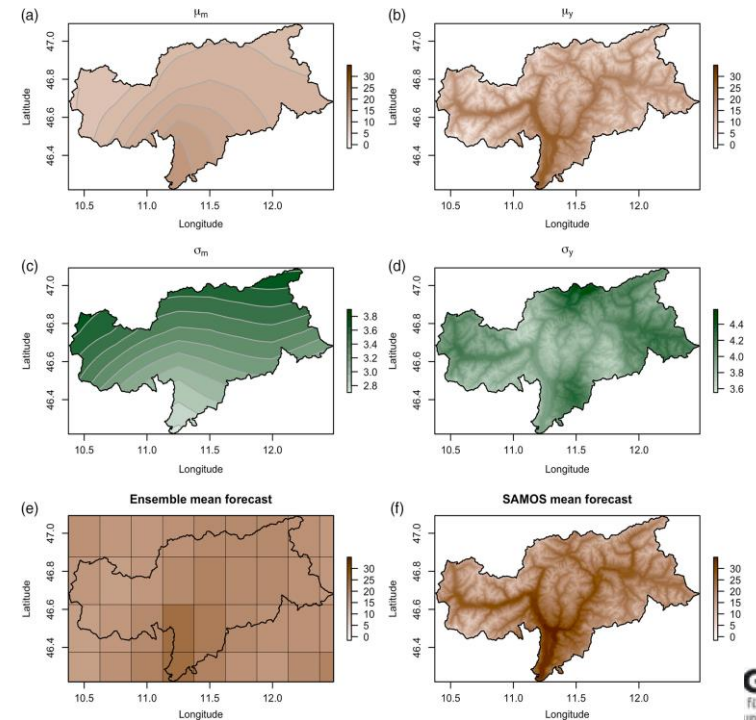
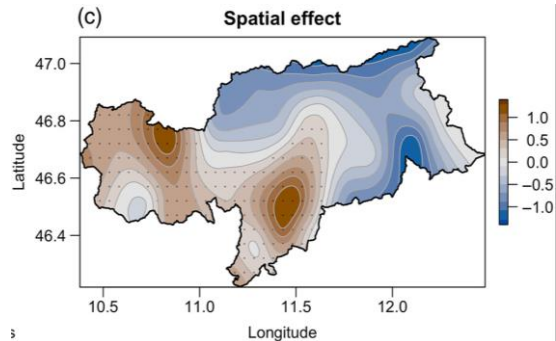
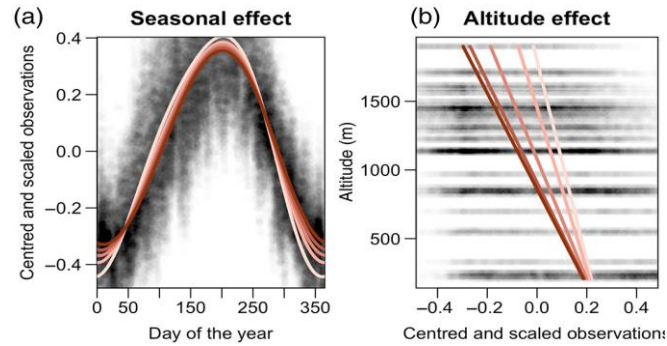
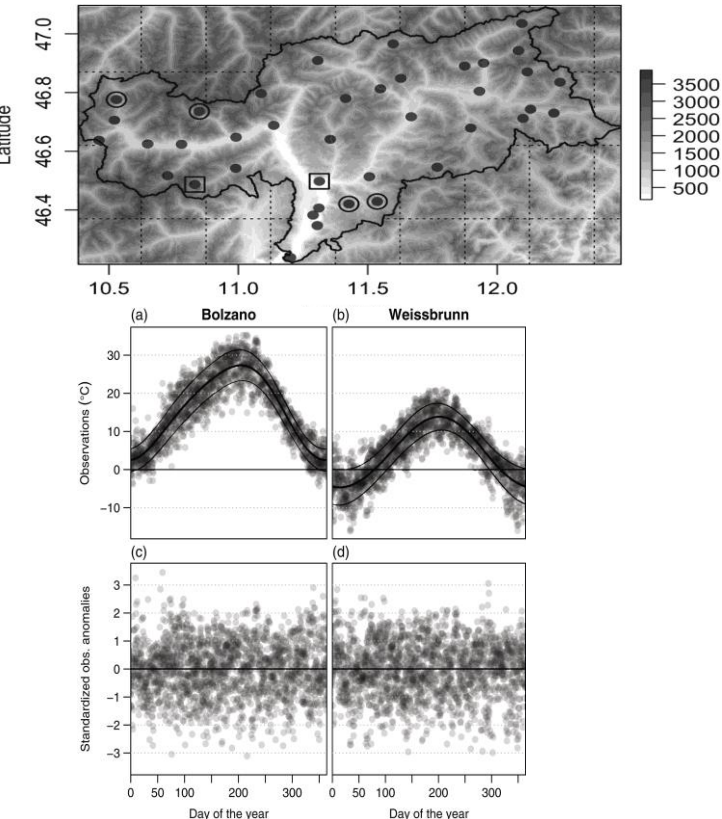


+ Boosting

$$y^* \sim N(\mu^*, \sigma^*),$$

$$\mu^* = b_0 + b_1 m^*,$$

$$\log(\sigma^*) = c_0 + c_1 \log(s^*),$$



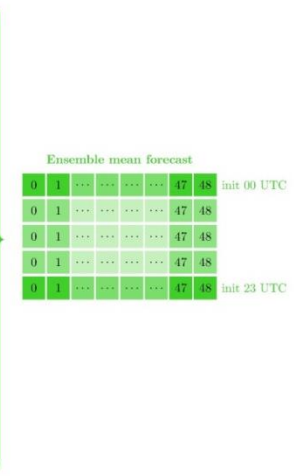
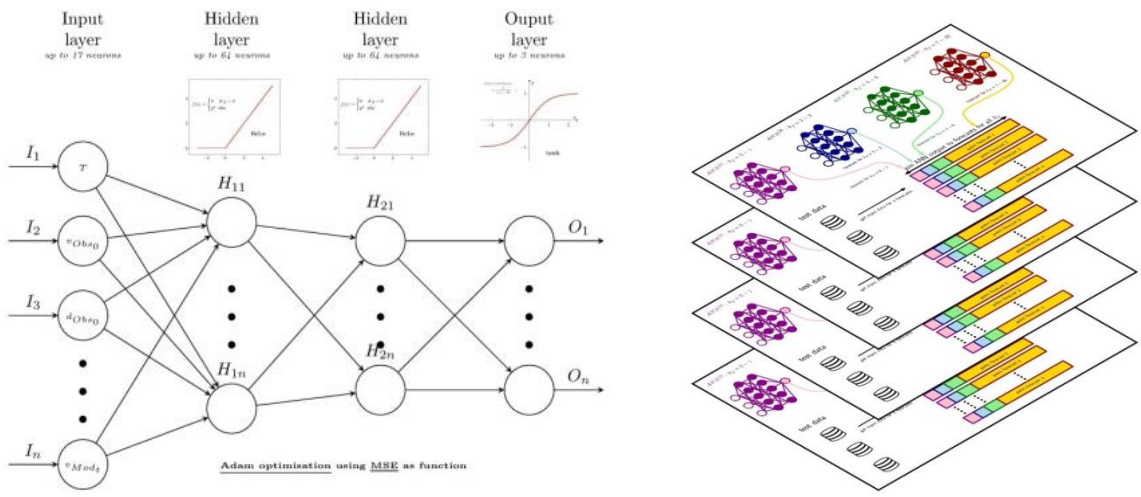


ANN / CNN / ConvLSTM

- Different applications:
 - meteorological forecasting grid/point
 - Downstream applications: renewables, agriculture, transportation, mobility, logistics, road maintenance...
- Different types of data
 - Observations (standard WMO)
 - Satellite
 - Radar, lidar data
 - NWP models with varying quality, domain, grid size,...
 - IoT: private weather stations, GPRSS, microlink data, mobile devices
- Different types of AI methods
 - Simpler: MLP, Random Forest, SVM
 - Complex: CNN, ConvLSTM, Bernstein Quantiles
 - Rather novel: NODEs, Graph (C)NN, SDEs/differential equations, physics-aware/inspired, GANs,...

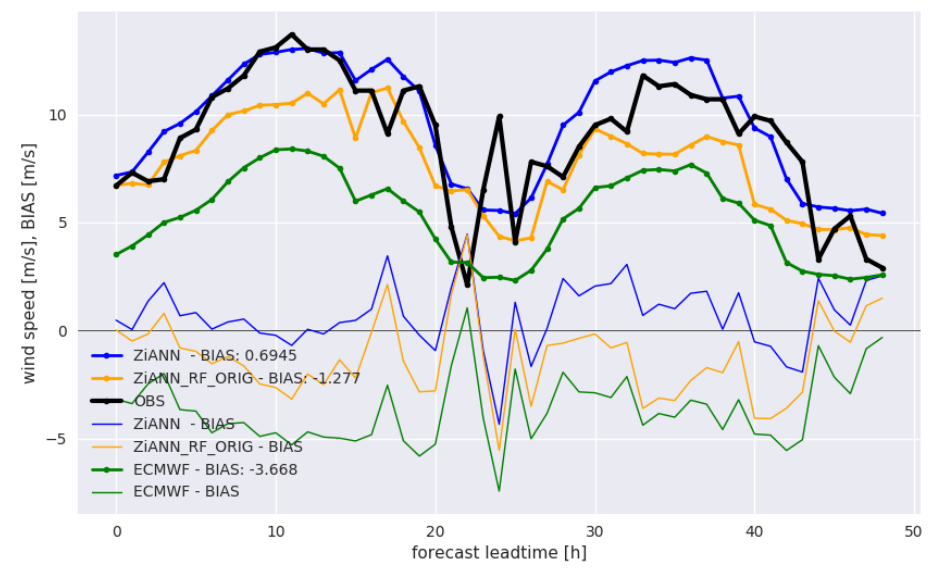
Post-processing – machine learning methods we use

Input data: NWP(point/grid), TAWES/SCADA



- Hourly forecasts for the next 48 hours ahead
- Uses a neural network in “ensemble mode” (deterministic forecast) but can also switch to random forest forecast (future: good to have both)
- Subhourly added
- RF + LSTM component added
- Needed adjustments in pre-processing (scaling + transformation)

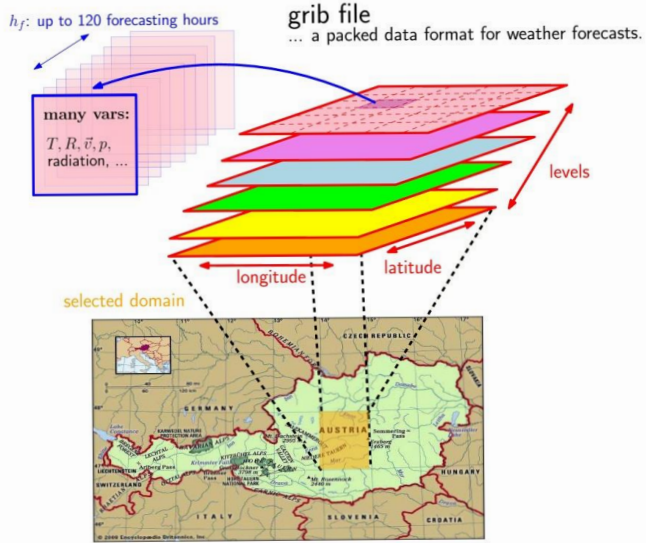
- Skills:**
- Direct access to “online” SCADA data
 - In-built QC
 - Adjustable forecast intervals, neurons, layers, etc.
 - Adjustable training length depending on data availability



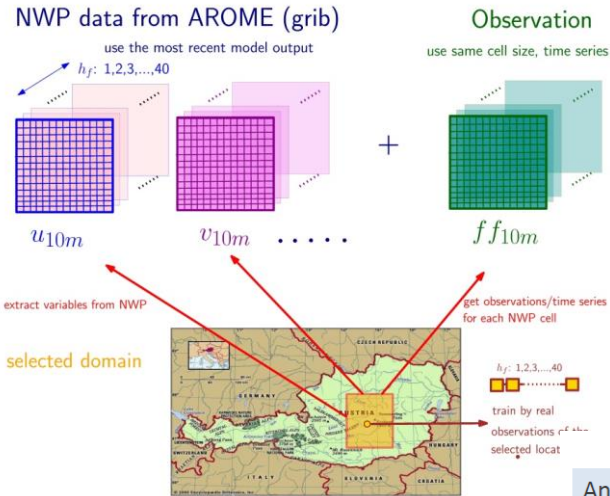
- Challenges:**
- “our” obs data available every 10-minutes
 - NWP data so far with a large delay
 - Non-convection permitting models are easy to learn of, don’t need long time series of data – convection permitting models not, need lots of data
 - Changes in the NWP model – how to deal with them? After 3 – 4 years a model changes nearly completely

Post-processing – machine learning methods we use

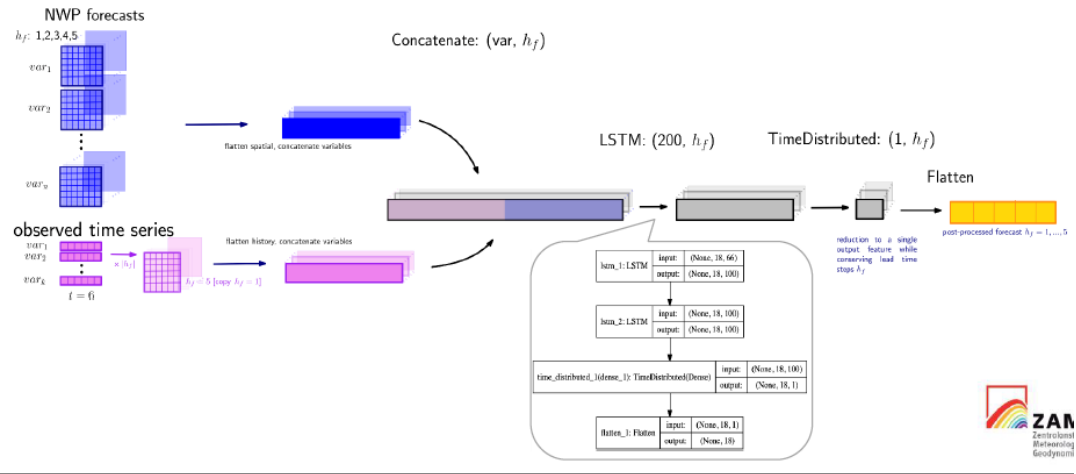
Point forecast using complex neural network setup and multiple data sources (PhD project, AWaKE):



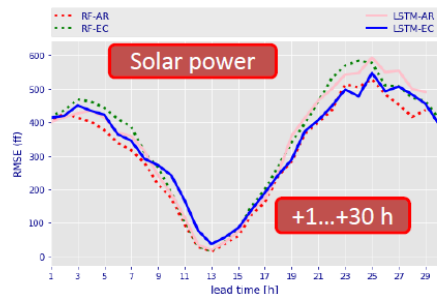
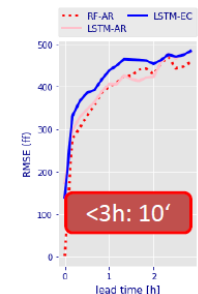
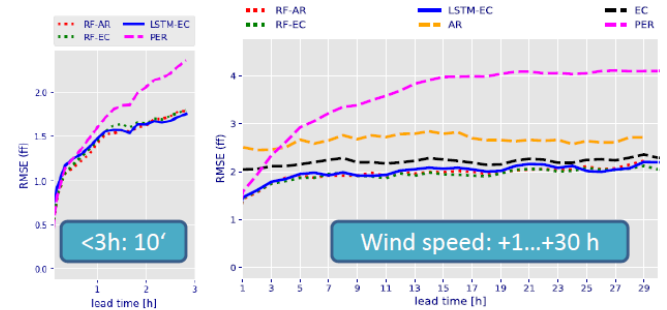
4-DIMENSIONAL DATA MODEL (\rightarrow CNN)



Semi-operational for solar and wind



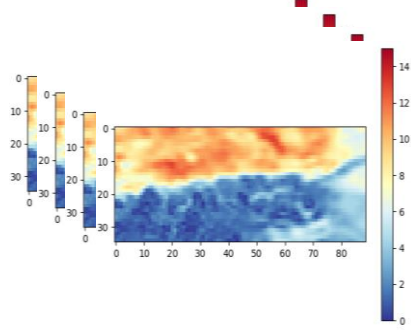
Andau, case study 2020:
 - mean of 38 turbines
 - SAMOS archive for training (= AROME / EC at 00:00 UTC)
 - runs each full hour (24 times a day) or suntime hours



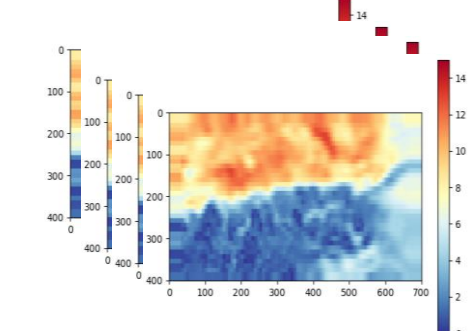
- best setup found used for training from SAMOS archive (2018-2020)
 - AI Models migrated to vmlearn
 - queries of AROME, TAWES, Energie Burgenland etc. in real-time

Post-processing – machine learning methods

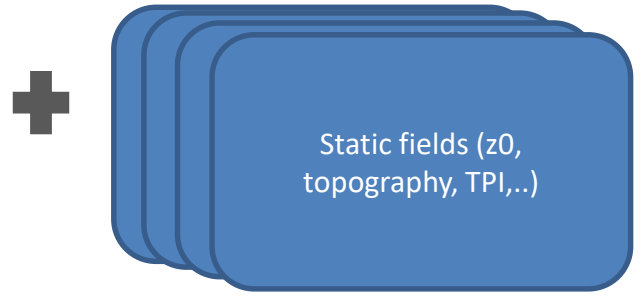
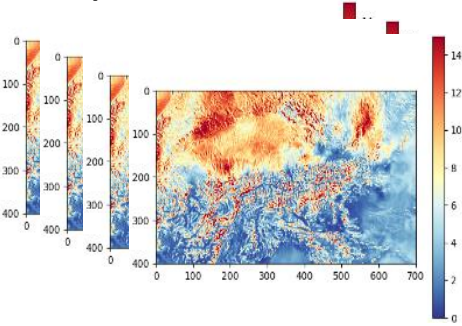
ECMWF raw parameters



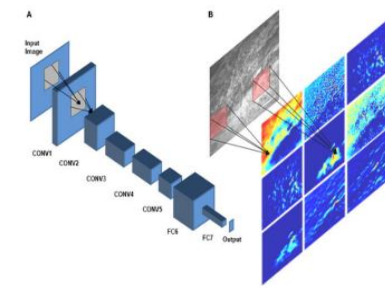
ECMWF downscaled



INCA parameters



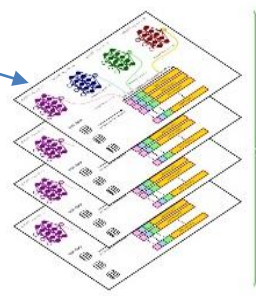
Core being replaced by:



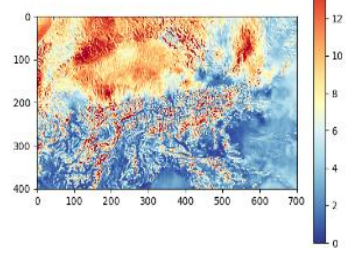
Convolutional neural network

https://miro.medium.com/max/1214/1*AUBtwV3xkXW41Bbh2Uh1hw.png

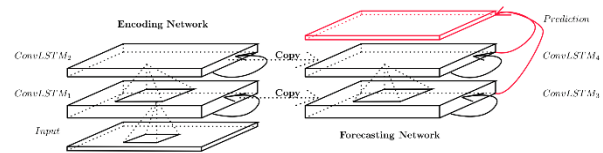
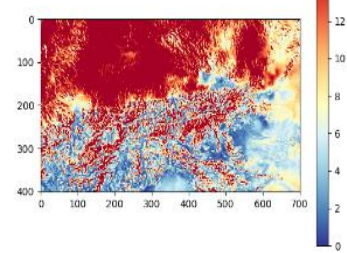
(Ensemble) neural network
2 input „branches“



10 m wind speed

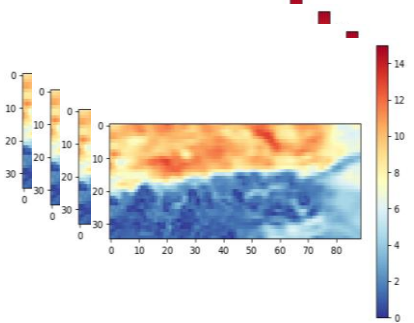


100 m wind speed

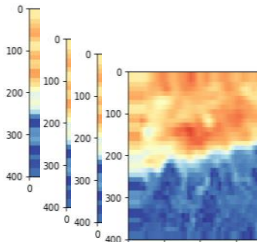


Post-processing – machine learning methods

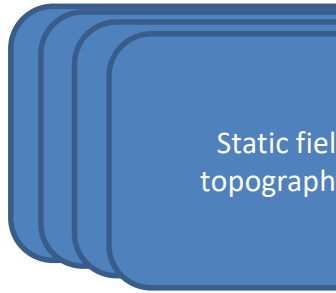
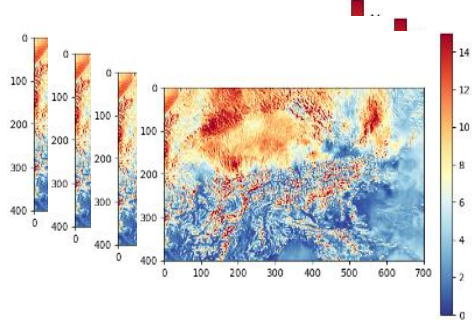
ECMWF raw parameters



ECMWF downscaled



INCA parameters



Static field
topography

```
#####complex model setup#####
def ML_two_branches_samedims(algo, Adim, Bdim, objective, lr, decay):

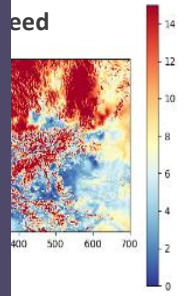
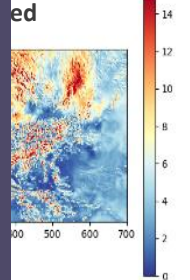
    if algo == 'Adam':
        algo = Adam(lr = lr, beta_1 = 0.9, beta_2 = 0.99, epsilon = 1e-08, decay = decay)
        lookback = 1

    print('shapes input: ', Adim, Bdim)
    # define two sets of inputs
    inputA = Input(shape=Adim)
    inputB = Input(shape=Bdim)
    # the first branch operates on the first input
    x = Dense(64, activation="relu", kernel_initializer="he_normal")(inputA)
    x = Dropout(0.2)(x)
    #x = Dense(64, activation="relu", kernel_initializer="he_normal")(x)
    #x = BatchNormalization()(x)
    x = GaussianNoise( 0.01 )(x)
    x = Model(inputs=inputA, outputs=x)

    # the second branch operates on the second input
    y = Dense(64, activation="relu", kernel_initializer="he_normal")(inputB)
    #y = BatchNormalization()(y)
    y = Dropout(0.2)(y)
    #y = Dense(64, activation="relu", kernel_initializer="he_normal")(y)
    #y = BatchNormalization()(y)
    #y = Dense(64, activation="relu", kernel_initializer="he_normal")(y)
    #y = BatchNormalization()(y)
    y = GaussianNoise( 0.01 )(y)
    y = Model(inputs=inputB, outputs=y)

    # combine the output of the two branches
    combined = concatenate([x.output, y.output])
    # apply a FC layer and then a regression prediction on the
    # combined outputs
    #z = Dense(32, activation="tanh")(combined)
    z = Dense(1, activation="relu")(combined)

    # our model will accept the inputs of the two branches and
    # then output a single value
    model = Model(inputs=[inputA, inputB], outputs=z)
    model.compile(loss=objective, optimizer=algo, metrics=['mae',rmse])
    #plot_model(model, to_file='model_2dims.png', show_shapes=True, show_layer_names=True)
```

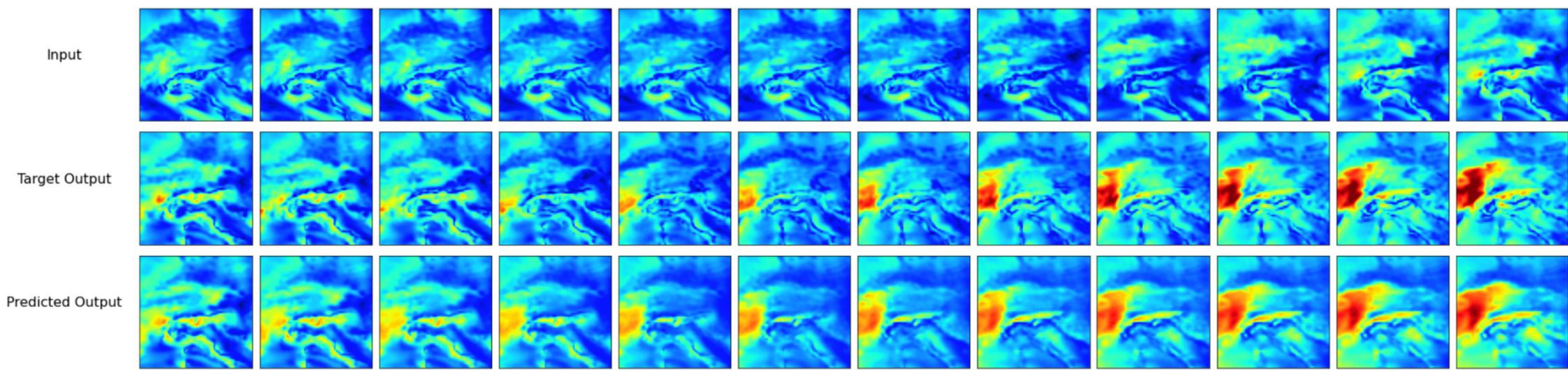
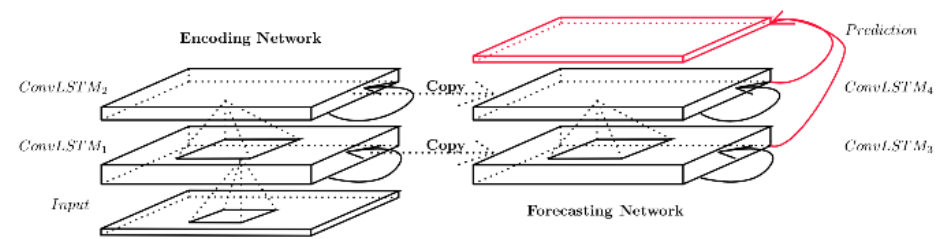




Post-processing – machine learning methods

ConvLSTM based model with adapted weighted loss function for different categories of wind speed

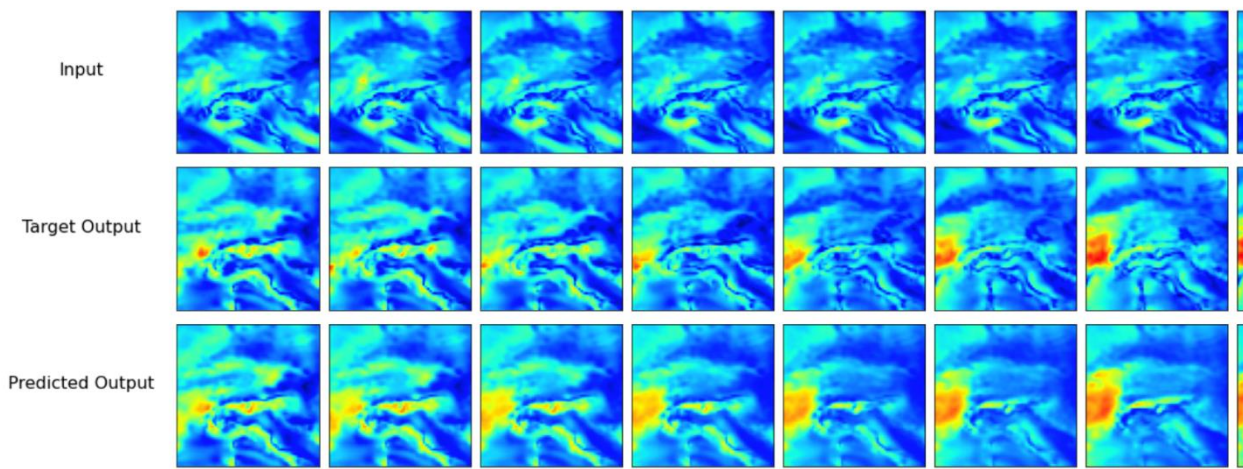
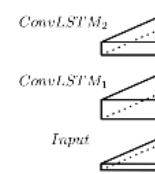
- Some sort of basic physics aware network
- Weighting of less frequent cases of wind speed („extremes“)
- Adapted metric function
- Data-driven using ERA5 as input



Post-processing – machine learning methods

ConvLSTM based model with adapted weighted loss function

- Some sort of basic physics aware network
- Weighting of less frequent cases of wind speed („extremes“)
- Adapted metric function
- Data-driven using ERA5 as input



```

72 class CLSTM_cell(nn.Module):
73     """ConvLSTMCell
74     """
75     def __init__(self, shape, input_channels, filter_size, num_features, seq_len):
76         super(CLSTM_cell, self).__init__()
77
78         self.shape = shape # H, W
79         self.input_channels = input_channels
80         self.filter_size = filter_size
81         self.num_features = num_features
82         self.seq_len = seq_len
83         # in this way the output has the same size
84         self.padding = (filter_size - 1) // 2
85         self.conv = nn.Sequential(
86             nn.Conv2d(self.input_channels + self.num_features,
87                       4 * self.num_features, self.filter_size, 1,
88                       self.padding),
89             nn.GroupNorm(4 * self.num_features // 32, 4 * self.num_features))
90
91     def forward(self, inputs=None, hidden_state=None, seq_len=None):
92         seq_len=self.seq_len
93         if hidden_state is None:
94             hx = torch.zeros(inputs.size(1), self.num_features, self.shape[0],
95                               self.shape[1]).cuda()
96             cx = torch.zeros(inputs.size(1), self.num_features, self.shape[0],
97                               self.shape[1]).cuda()
98         else:
99             hx, cx = hidden_state
100         output_inner = []
101         for index in range(seq_len):
102             if inputs is None:
103                 x = torch.zeros(hx.size(0), self.input_channels, self.shape[0],
104                                   self.shape[1]).cuda()
105             else:
106                 x = inputs[index, ...]
107
108             combined = torch.cat((x, hx), 1)
109             gates = self.conv(combined) # gates: S, num_features*4, H, W
110             # it should return 4 tensors: i,f,g,o
111             ingate, forgetgate, cellgate, outgate = torch.split(
112                 gates, self.num_features, dim=1)
113             ingate = torch.sigmoid(ingate)
114             forgetgate = torch.sigmoid(forgetgate)
115             cellgate = torch.tanh(cellgate)
116             outgate = torch.sigmoid(outgate)
117
118             cy = (forgetgate * cx) + (ingate * cellgate)
119             hy = outgate * torch.tanh(cy)
120             output_inner.append(hy)
121             hx = hy
122             cx = cy
123         return torch.stack(output_inner), (hy, cx)
124
125

```


Post-processing – machine learning methods setup for a case study

- growing **renewable energy** source, can yield very different output for each location of interest
- effective integration to **power grid**: need **forecasts** of the **expected power curve** (e.g.: serves for grid stability, energy trading, scheduling of maintenance / energy transfer, ...)
- various **data sources available**: power generated, met. site observation, satellite, numeric prediction (NWP)
- strong **seasonal** and **diurnal variation** in the data → want these variations in the nowcasts



https://commons.wikimedia.org/wiki/File:Solar_PV_Austrian_Alps.jpg



→ investigate machine learning/ML such as **Artificial Neural Nets**, **Random Forest** as efficient forecast tool

Data for CASE STUDY 2021

We optimize **site specific models** and select data for each site from:

INPUT:

- **AROME:**
forecasts in various p/z levels of solar radiation related parameters (e.g.: short-wave radiation, cloud cover, ...)
- **CAMS – site interpolated radiation timeseries:**
radiation related parameters
- **Observation site:**
observed solar power
- **TAWES/INCA** – closest observation/analysis at surface level:
global radiation, temperature, wind, humidity

CASESTUDY

Training:

- ✓ 2015-2020 (incl. artificial)
- ✓ 2020 (real only)

Testing:

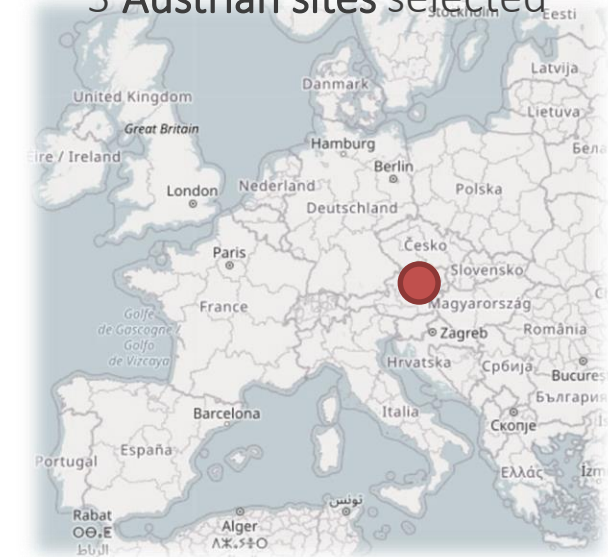
- ✓ 2021

+ computed **climatology**

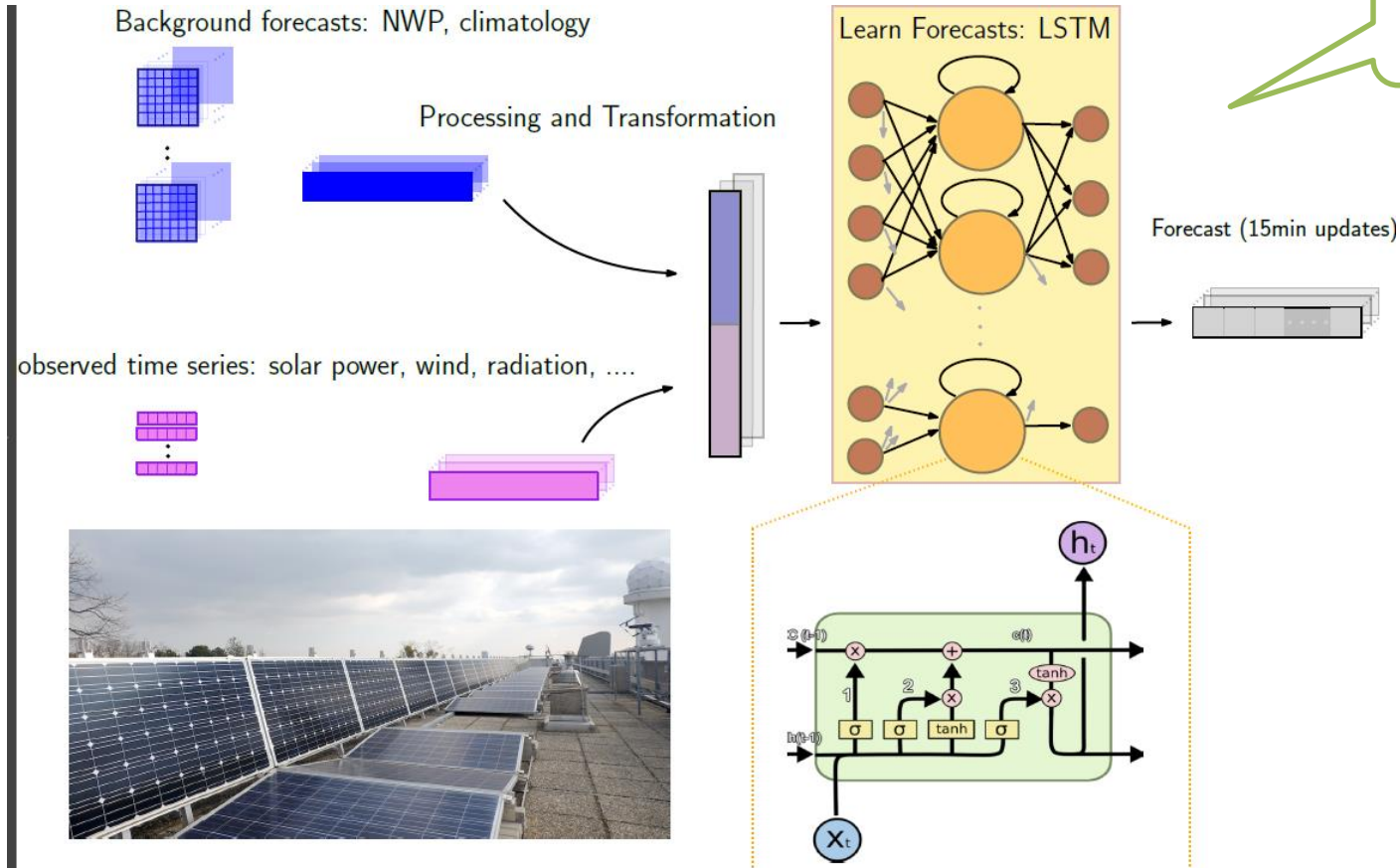
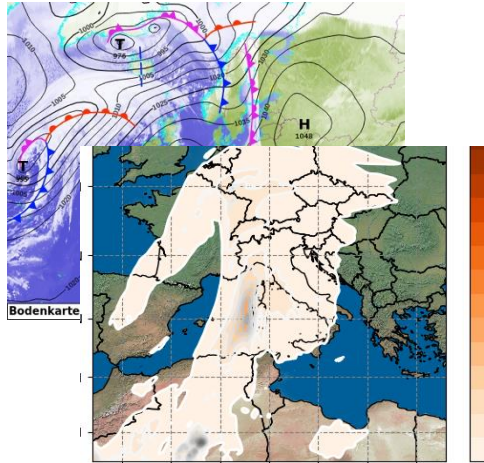
Check missing, normalize, etc.

OUTPUT: solar power forecasts
in 15 min. resolution
+6 hours, hourly runs

3 Austrian sites selected



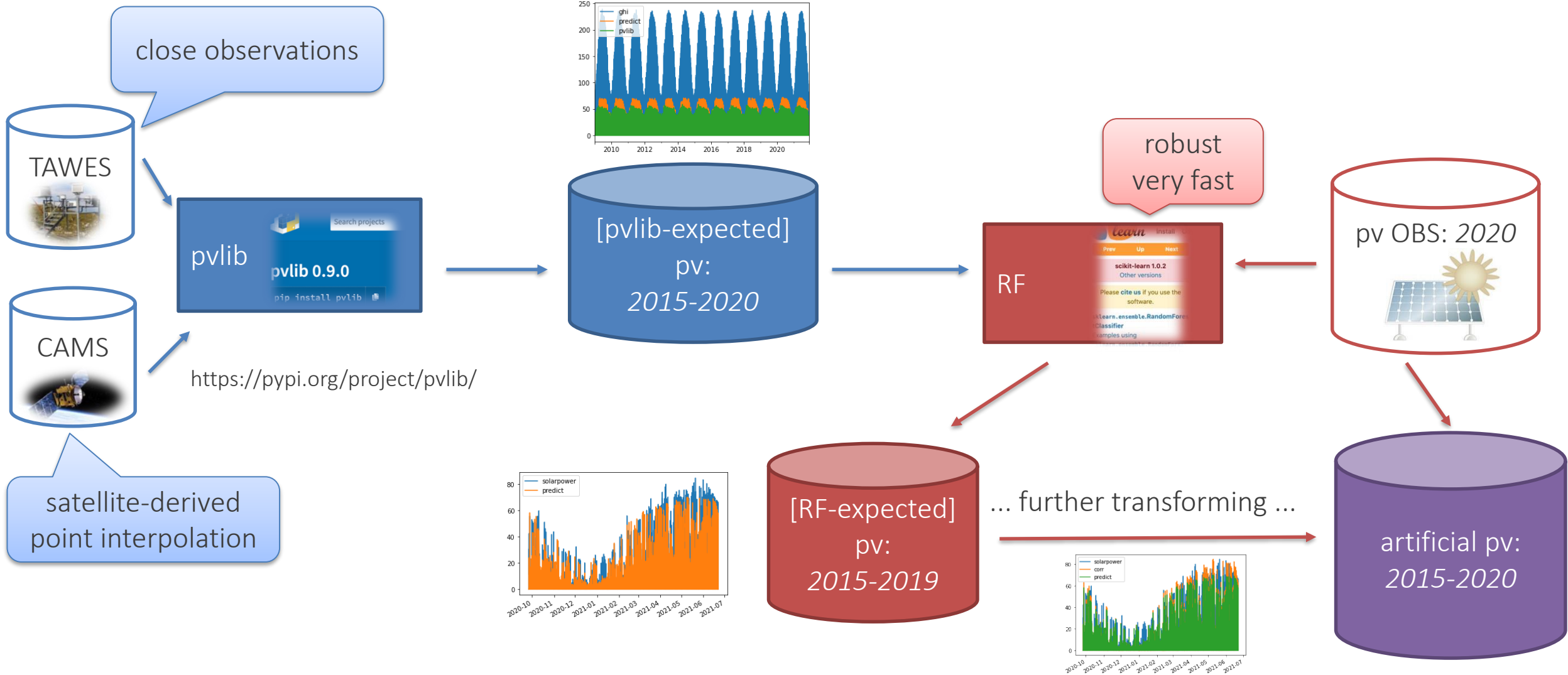
Post-processing Methodology: update a Background Model(s) by ML



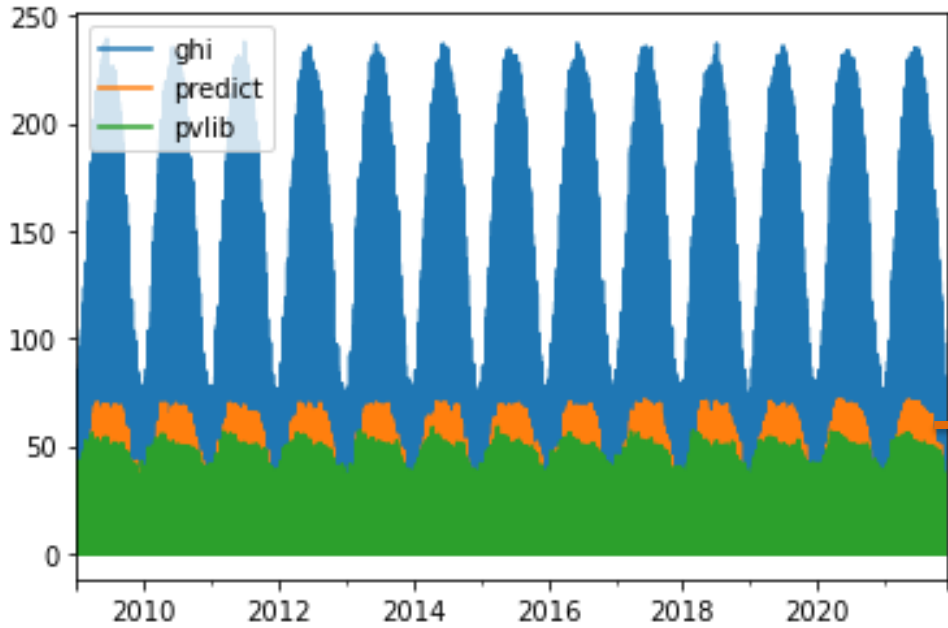
Alternative:
Random Forest

Issue: Short Observation Time-series of Power Plants

→ generation of **artificial training data**, as more is needed for complex models (LSTM etc.)



Data: Obtaining Artificial Time-series

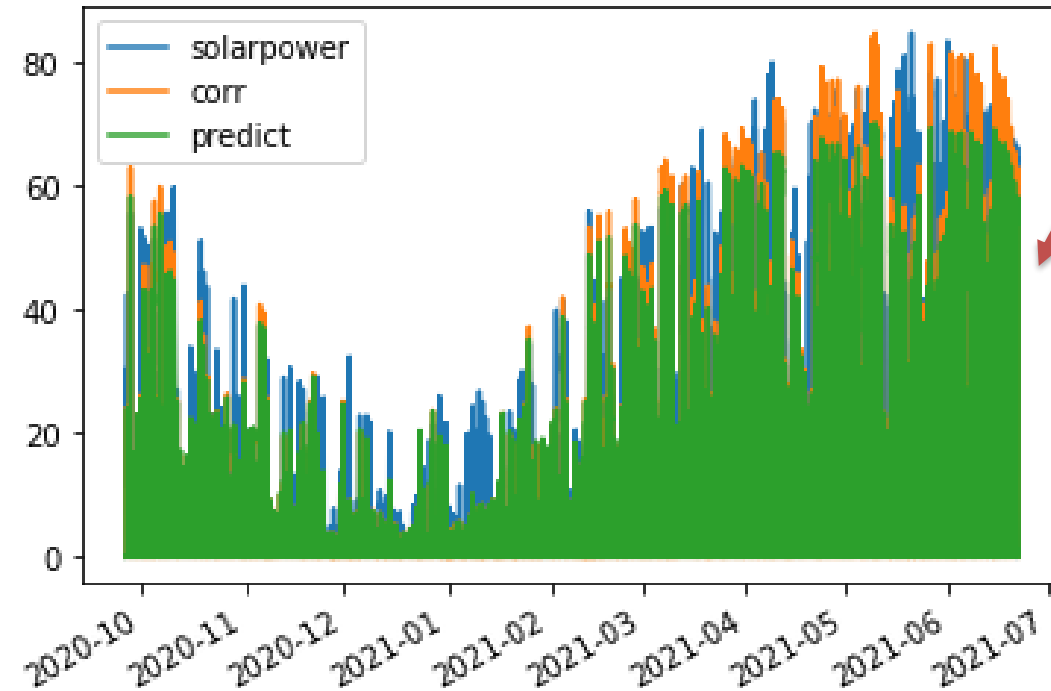
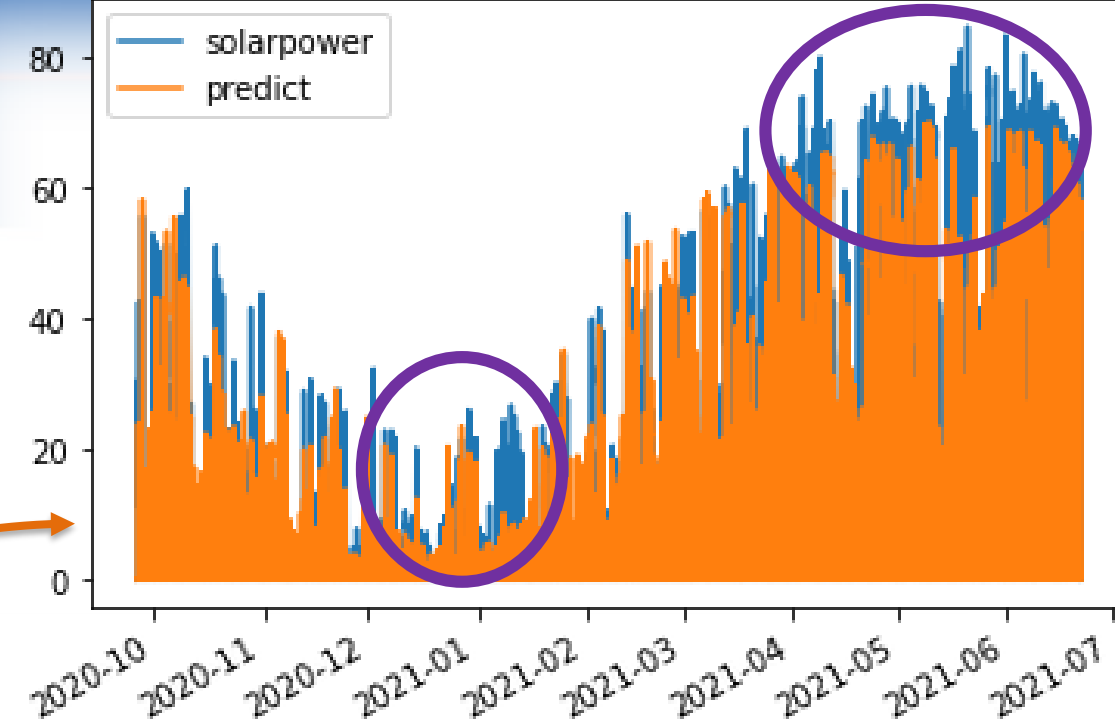


Long time series obtained: ghi, pvlib
... deviate in scale for our specific site



predict :=RF output based on CAMS-radiation + TAWES observation + pvlib

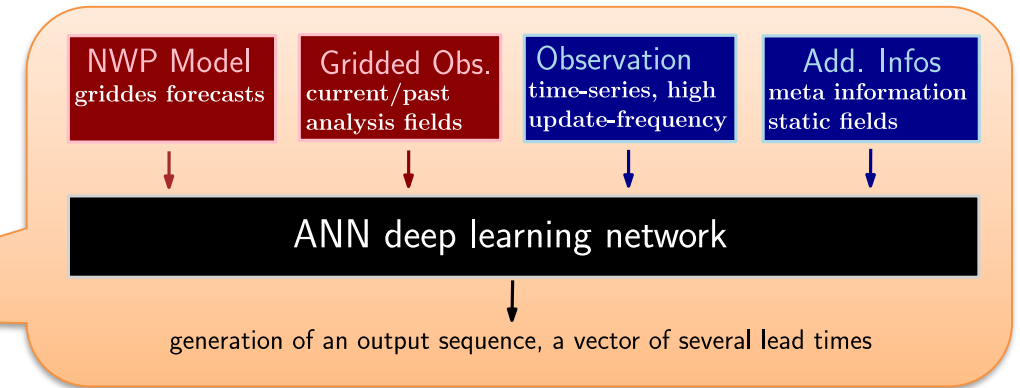
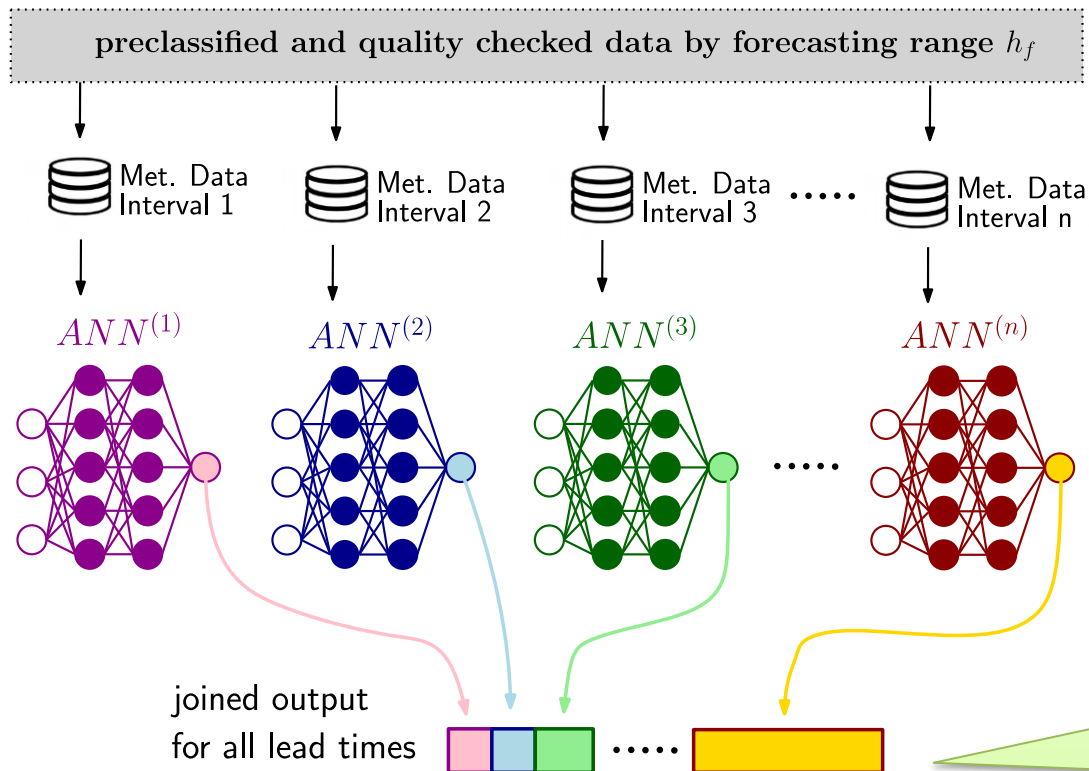
RF+pvlib good but often seems to underestimate real PV



1st opti. approach:
transform by percentiles of PV-OBS

Selection and Transformation of Inputs

1. **input feature X** selection: simple methods such as RF weights, Target Y: solar power
2. replace / remove **missing** values, check quality
3. 0-1 **normalization**, using (here hourly) climatological standards
4. for longer vectors / sequences: **intervalization** by lead time steps



Basic climatological transformation from normalized X

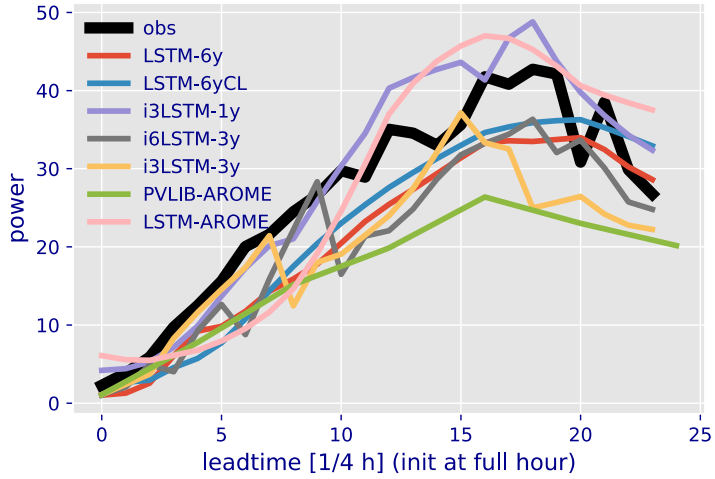
$$\Delta x_i(t) := \frac{1 + \text{norm}(x_{i\{OBS\}}(t)) - \text{norm}(x_{i\{CLIM\}}(\text{hour}(t)))}{2}$$

$$pv = \text{denorm}(2\Delta pv - 1 + \text{norm}(pv_{clim}))$$

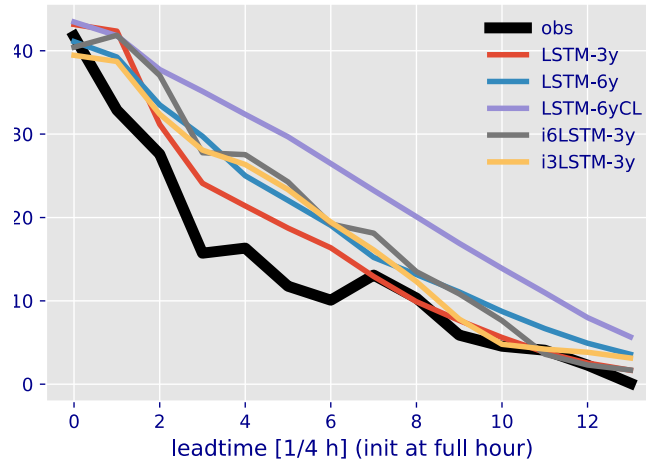
$i = pv, rad, ff, u, v, T, CC, TOA, \dots \quad t = 1, 2, \dots, 24$

Case Study Results – Sample Forecasts

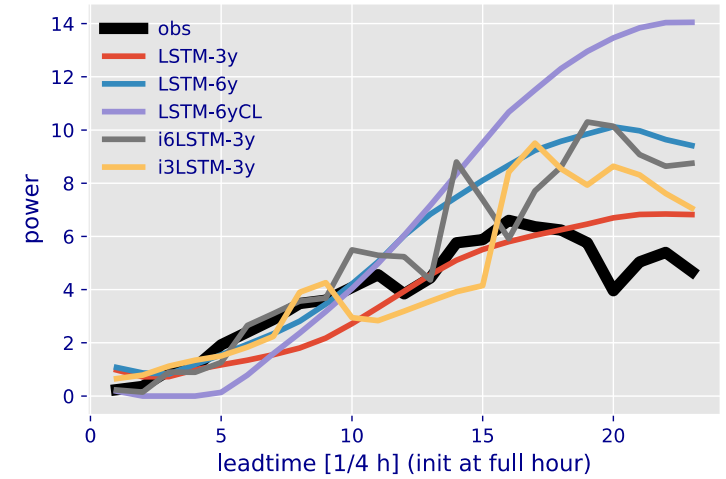
2021-02-01 07:00:00



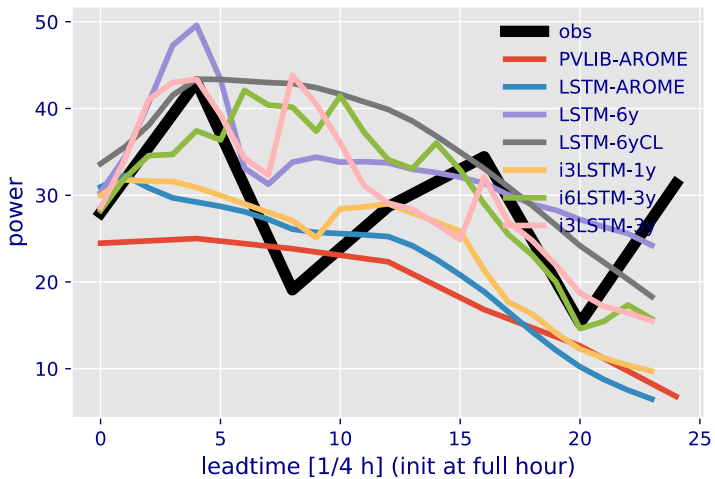
2021-05-01 15:00:00



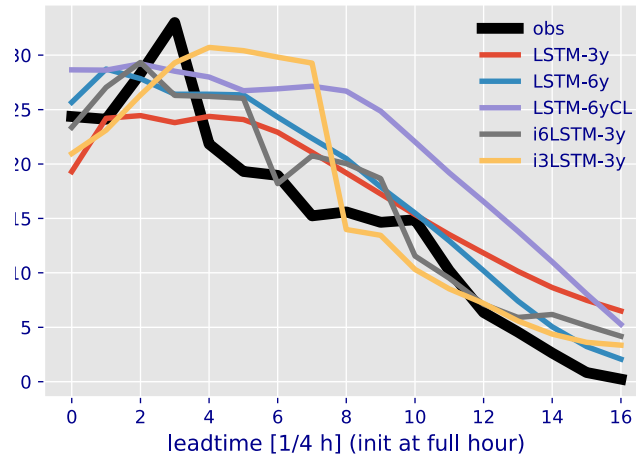
2021-01-01 07:00:00



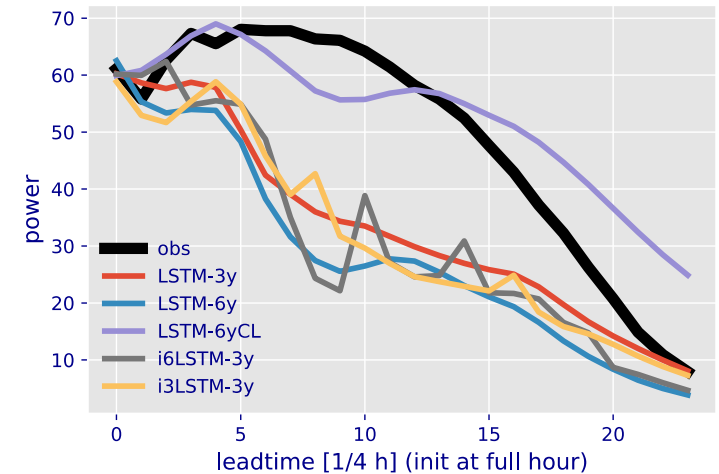
2021-04-15 10:00:00



2021-02-01 12:00:00

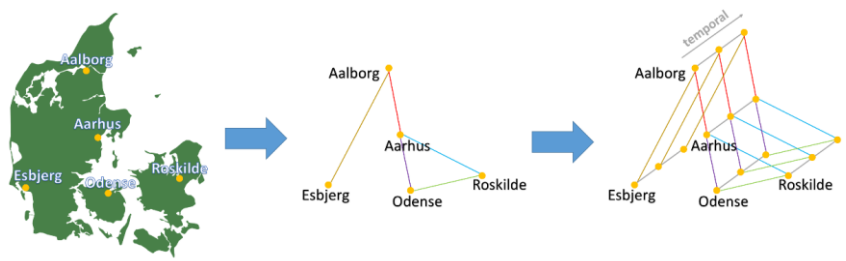


2021-03-01 10:00:00

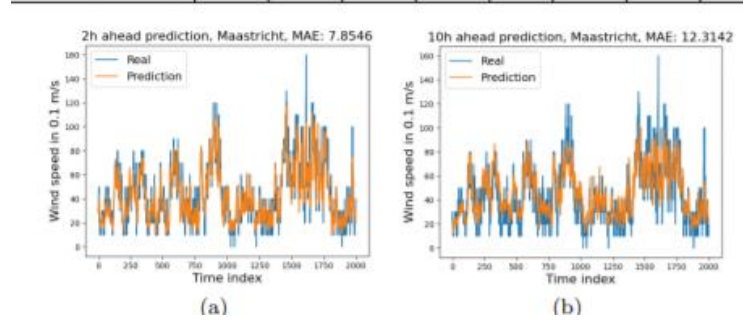
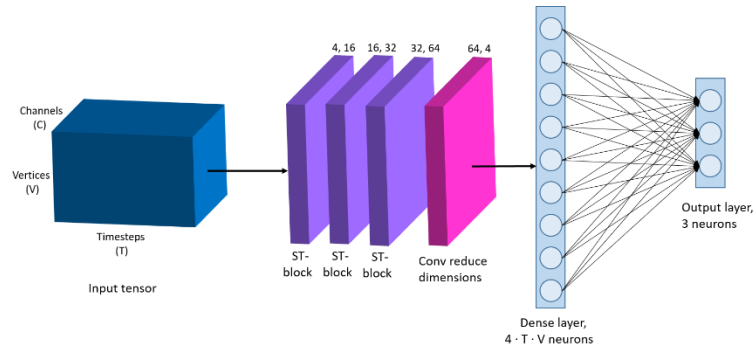


Post-processing – machine learning methods

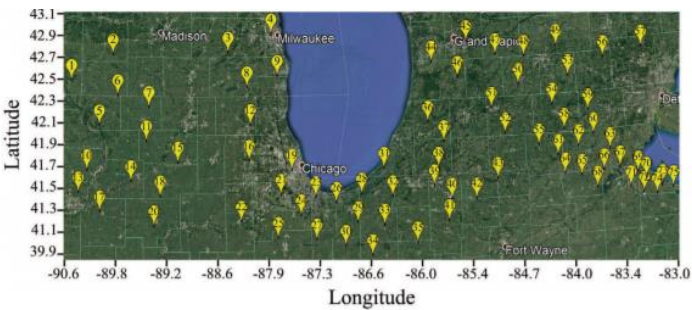
Graph networks – for wind/solar energy prediction



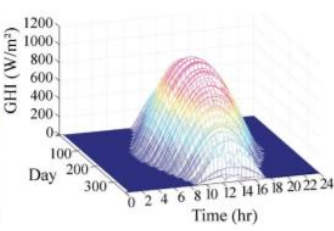
<https://www.esann.org/sites/default/files/proceedings/2021/ES2021-25.pdf>



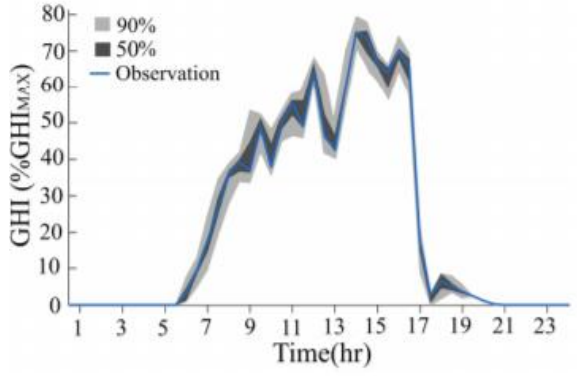
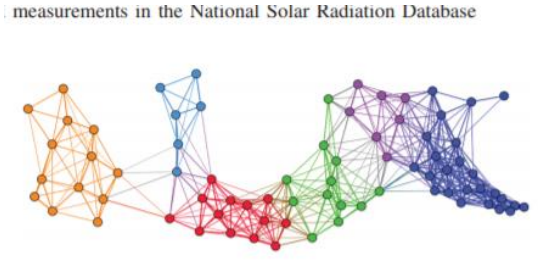
Something similar being implemented right now



(a) Latitude-Longitude map of 75 solar sites in the NSRDB



(b) Solar Irradiance of 2015 at solar site 14

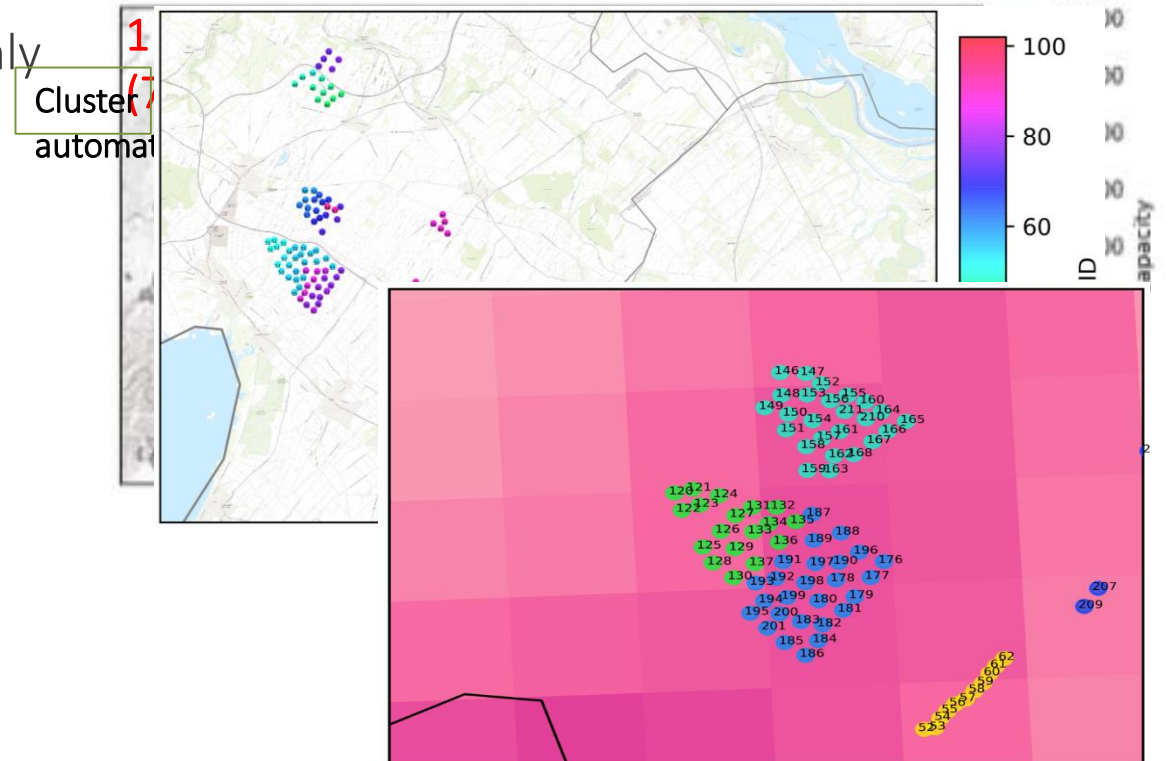


<https://ieeexplore.ieee.org/ielam/5165391/9043622/8663347-aam.pdf>

Post-processing – federated learning

Application fields for federated learning

- wind / solar energy: given the data policies of providers, TSOs, traders, etc. → distributed / network federated learning would definitely help improving forecasts
- Meteorology: forecasts for obs sites/sites using not only e.g. Austrian data but combine European observation network or even PWS sites (after quality control)
- Mobility: combine different sources, even car measurements
- Agriculture
- ...



Post-processing – machine learning methods replacing gridded observations

Idea: use machine learning methods and/or statistics to “interpolate” in-situ observations of wind speed to a specified grid

Results: 100 m and 1 km analysis fields of wind speed using a different methodology. Add on: depending on used background fields (DEM etc.) resolution could be changed to higher/lower.

→ Can we use Graphs here? Would the work better? Can federated learning improve here

Fig. 5 Example of f_i computed at different (differences of Gauss

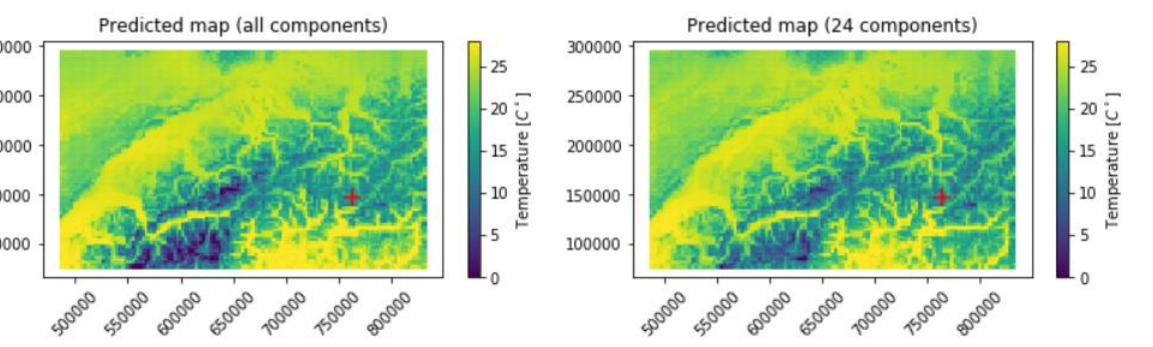
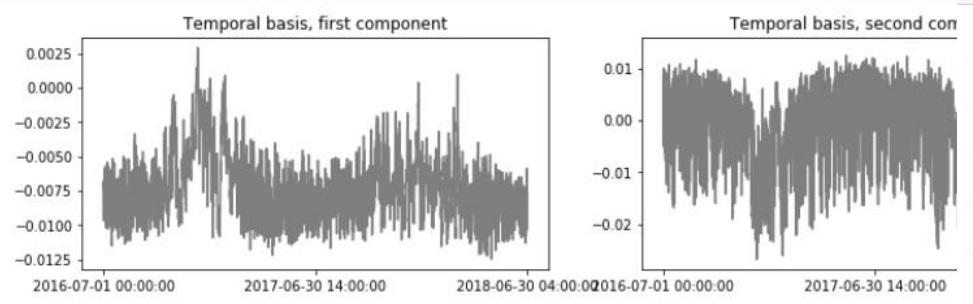
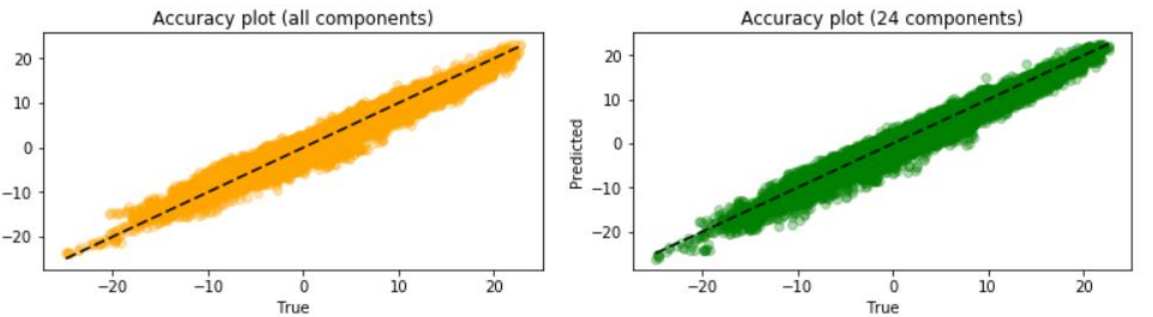
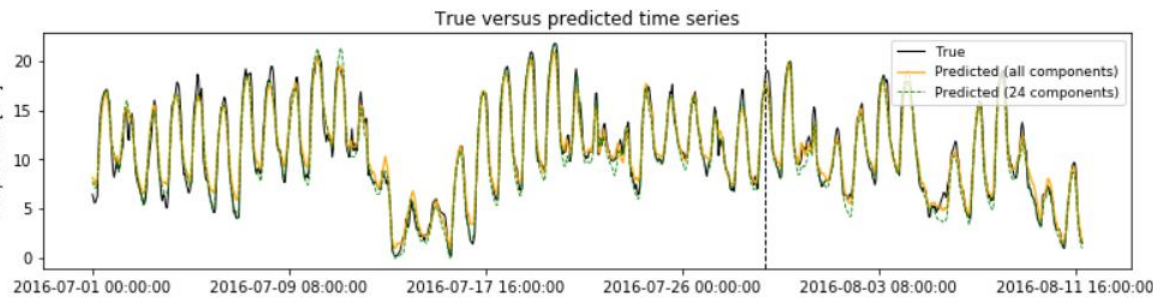
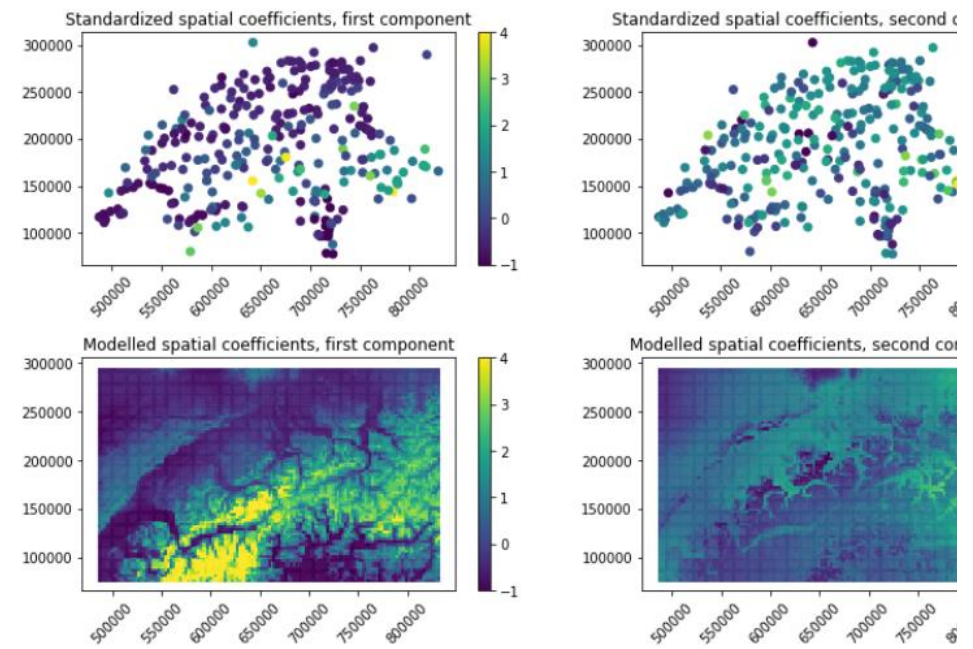


Fig. 6 Three topogra features that are com build the target functi (DoG, f_1), top right (bottom left (direction derivative, f_3), Bottom Digital elevation moc country boundaries





Questions?

Recommendations?

Comments?