Royal Netherlands
Meteorological Institute
Ministry of Infrastructure
and Water Management

Short-Term Forecasting of High-Impact Weather with Physics-Guided Machine Learning : Technical Report

C. A. Severijns & G. A. Pagani

De Bilt, 2023 | Technical report; TR-403

Short-Term Forecasting of High-Impact Weather with Physics-Guided Machine Learning

Technical Report

C. A. Severijns & G. A. Pagani

April 17, 2023

Introduction

Extreme weather events such as heavy storms and heavy precipitation have a large impact on our society. Due to both societal developments and climate change, it is expected that their impact will increase in the future. Especially in the case of rapidly developing extreme weather phenomena, society would benefit from forecasts at short time intervals, i.e., nowcasting. In principle, such forecasts can be made with an NWP model. However, an NWP model is expensive to run both in computational resources and in run-time, so that in practice the highest rate at which forecasts can be produced is about once per hour with today's technologies. Machine learning (ML) techniques have been used as a much more computational efficient replacement for, e.g., parameterizations in NWP models [2, 4, 5]. Therefore, using state-of-the-art ML techniques, it might also be possible to develop a system with which forecasts can be produced efficiently many times per hour and even on-demand.

The aim of this MSO project was to investigate whether it is possible to design a short-term forecasting system that updates the most recent forecast from the NWP model with the latest observational data using physics-guided ML techniques. Such a system fits well in the requirements for Early Warning System as described by the World Meteorological Organization [3] and it will be a prototype for making nowcast products for the KNMI Early Warning Center (EWC) [1].

In section 1 we consider the unique characteristics of weather forecasting data and their implications for the application of ML techniques. In section 2 we describe constraints that current ML techniques have in so far these are relevant to their application to weather forecasting. Hereafter, we describe how we designed the prototype and the results that we obtained with it in sections 3 and 4, and we conclude with a number of recommendations for an operational system based on our prototype design in section 5.

1 Characteristics of Weather Forecast Data

Weather forecasting data have two characteristics in which they differ from the data in other application areas of ML. First of all, weather forecast data is correlated in space and time. The consequence is that the data cannot be subdivided in indepedent training samples that cover less than the spatiotemporal scales at which correlations occur. The batches of training data consist of a relatively small number ($\approx O(10^4)$) of large samples ($\geq O(10^5)$ spatiotemporal data points) of which only a small number concerns cases of high-impact weather (a few cases per year). Second, the forecast produced by the NWP model has no noise[1] although each forecast is uncertain due to the incomplete knowledge of the state of the atmosphere at any point in time. The measurement errors are the source of uncertainty in the observational data. These instrumental errors can be both systematic or random.

The NWP model produces gridded data while observational data is in general not measured at those grid points. In addition, relevant features occur at all scales in the data from the full domain to close to the grid resolution in the output of the NWP model. Not all ML techniques are suited for this type of data. And last but not least, the weather forecasting data should satisfy a number of conditions to be physically consistent. Since the Navier-Stokes equations contain derivatives, the ML technique should support differentiation. We will discuss the implications of these characteristics in the next section.

[1] Noise here means features that are not relevant to the relevant information contained in the data.

1.1 Requirements For an ML-based Forecasting Systems

In most applications of ML the training data is partitioned in batches. In our type of application, this partitioning should be done in such a way that correlations are preserved. Therefore, the batches should consist of samples that are spaced sufficiently far apart in time so that they are uncorrelated. Furthermore, each sample should contain data from a single NWP forecast and observations from the period of the forecast to maintain the correlations within the data.

In addition to preserving correlations, the training data should be well balanced to ensure that the ML system learns about all relevant cases, not just about those that occur most frequently. Given the relatively small amount of data on the most relevant, i.e., high-impact, weather situations, weather forecast data is strongly imbalanced. In general, an imbalance in the data can be corrected by means of data augmentation, e.g, by using rotational, translational, and reflectional symmetry in the case of images. However, weather forecast data lacks the symmetries that can be used for this purpose.[2] Even with data augmentation, the small number of high-impact weather cases means that the system would be biased towards these specific cases.

Since the locations at which observational data are obtained do not coincide with points of the grid used in the NWP model, it is advantageous if the ML technique is mesh-free so that it can deal with both types of data. This excludes ML techniques such as convolutional neural network (CNN) which require data on a regularly spaced grid.[3]

Many ML techniques are better able to approximate low frequency, large scale features in the data than high frequency, small scale features. Since weather data contains features on all scales, this so-called spectral bias should be corrected or a ML technique without such a bias should be used.

Finally, including laws of physics such as the Navier-Stokes equations, is possible in most but not all ML techniques.

2 Technical Limitations And Constraints

Since the amount of data in a single training sample (that is sufficiently large to preserve spatiotemporal correlations) is much larger than the sample size (e.g, an image) in typical ML applications, training on batches of many samples requires a lot of computing resources. In the ML system that we have designed, the sample size is even an order of magnitude larger than the number of model weights. This means that transferring training samples to the GPUs takes longer than transferring the model weights. And the manner in which memory is allocated on the GPUs, limits the number of samples that can be stored in the memory of a single GPU to only a few. Although the loss function, which is a sum of individual loss terms, allows us to use many GPUs to train on a single batch of data, the number of GPUs required would be rather large.

Another aspect that needs to taken into account is that automatic differentiation is slow compared to all other numerical operations that are required to train an ML system. Since many physical laws involve (partial) derivatives, computing loss terms that account for these laws takes much more time than other loss terms, such as the mean squared error (MSE). One way to remedy this is by training on data and physics in parallel on separate GPUs. Computing higher order derivatives is even more expensive than first order derivatives. Higher order derivatives can be replaced by first order derivatives by introducing extra model variables (at the expense of more model weights).

Many ML systems are implemented in Python using a ML package such as Tensorflow or Pytorch. These packages support (and are optimized for) a number of common software architectures for ML systems including ones using multiple GPUs.[4] As long as the ML application matches one of these

[2]The dynamical equations have such symmetries but these are broken by the Earth's rotation and orography.

[3]Unless the observational data are interpolated to the NWP model grid first. This is something we prefer not to do in this work to prevent interpolation errors.

[4]Tensorflow supports both data parallel and model parallel execution modes. However, in model parallel execution mode, the GPUs are used *sequentially*. For true model parallellism, a Python package wrapping the MPI library can be used but this requires custom Python code.

architectures, performance is in general good.[5,6] However, if this is not the case, the implementation of the desired architecture in pure Python leads to performance degradation.[7]

3 Design of The Prototype

The three key components of any ML system using neural networks (NNs) are the network architecture, the loss function, and the optimization algorithm. All the tests we did used forecast and analysis data from the HARMONIE40 model from 2019, 2020 and 2021 on a domain of 40×40 horizontal grid points centered at the Netherlands. The number of model levels used varied between 5 and 30 from the 65 levels available and the forecast length varied between 9 and 12 hours. The observational data consisted of data from RobuKIS [8] and from the MARS archive at ECMWF. The variables that we included in our ML architecture are the air temperature, T, specific humidity, Q, the zonal, meridional and vertical wind components, (U, V, W), and the geopotential height, Z.

The two types of NN that are able to process mesh-free data are multi-layer perceptron (MLP) networks and graph CNNs. We use MLP networks since these are computationally cheaper. Each of the variables is modelled by a MLP network that takes time, t, pressure, P, and horizontal position, (x, y), as input and produces the value of the variable at any time and location within the domain. We tested many architectures, not only standard MLP networks but also networks with extra feature extraction layers and residual connections. We tested with ReLU, swish, tanh() and sin() activation functions. We obtained the best results, i.e., fastest reduction of the loss function, with standard MLP networks with the sin() activation function.

As to be expected, all the tested networks had a significant spectral bias. This was corrected by using a kernel MSE loss instead of the commonly used MSE loss for the NWP data. The kernel MSE loss is given by:

$$\mathcal{L}(f, \hat{f}) = \frac{1}{N} \sum_{i=1}^{N} \Big(\int k(x' - x_i) \big(f(x') - \hat{f}(x') \big) dx' \Big)^2 \tag{1}$$

where x_i are the N locations with data, $f(x')$ are the values predicted by the ML system, $\hat{f}(x')$ are the true values, and $k(x' - x_i)$ is the kernel. The advantage of the kernel MSE loss is that it enables one to control the weight of high-frequency errors in the loss function which results in a faster convergence during training for functions with small scale features. Convergence was also improved by using profiles to capture features at the size of the domain for most of the variables next to the NN. For example, a mean logarithmic vertical profile was added to the NN for the geopotential height, and for zonal and meridional wind components a linear vertical profile was added.

Next to the MSE and kernel MSE loss terms, loss terms for the Navier-Stokes equations is included in the loss function. This ensures that the model satisfies these equations as good as possible.[9] These physics loss terms are evaluated on a set of quasi-randomly distributed residual points instead of the grid points at which the NWP data is available to ensure maximum independence of the loss terms. The amount of residual points can be varied to constrain the model more or less strongly by the physical laws. We performed experiments in which the test dataset contained data from a time interval immediately following the train dataset. These experiments showed that in all cases in which the physics loss was included, the MSE with respect to the test dataset was significantly smaller than in the cases without the physics loss.

NNs are commonly trained with some type of stochastic gradient descent (SGD) optimizer. We experimented with various types of SGD-based optimizer such as standard optimizers like SGD and

[5] Our experience is that in general Tensorflow performs better than Pytorch.

[6] This is not always the case. For example, after loading a Tensorflow model that was trained before, computing time increases by 50% when the training is continued for reasons unknown to us.

[7] Typically, an algorithm implemented in C++ or Fortran is between 10^2 and 10^3 times faster than the equivalent implementation in Python.

[8] Although Robukis provides a large amount of observations, we found later that there are many duplications which reduces the actual amount by a factor ≈ 10.

[9] This of course within the limits allowed by the loss terms for the data.

Variable	Hidden Layers	Neurons/ Layer	Vertical Profile	Initialization	
				Weights	Biases
T	4	128	Linear	Glorot	Uniform
Q	4	128	Quartic	Orthogonal	Uniform
U	4	128	Linear	Orthogonal	Uniform
V	4	128	Linear	Orthogonal	Uniform
W	4	128	-	Orthogonal	Uniform
Z	4	32	Logarithmic	Glorot	Glorot

Table 1: The architecture of the MLP networks for all variables implemented in the prototype. The vertical profile specifies the dependency of the profile on pressure. The uniform distribution has values in the interval $(-\pi/2, \pi/2)$.

Adam, and more advanced optimizers like AdaBelief, AdamP, and SuperAdam. Our tests showed that the quasi-Newton L-BFGS method converged faster than any of these SGD-based optimizers. This is most likely due to the fact that in our application most of the data is free of noise.[10]

The results of all the tests were used to design a prototype of a ML system that can be used to update the weather forecast. Our prototype system is designed to locally approximate the trajectory of the atmosphere in state space. An overview of the architectures of all MLP networks used in the prototype is presented in Table 1 (see also Figure 2(b)). This gives a total of 338513 parameters. All training and test data were normalized in such a way that they were in the interval $(-1, 1)$. The loss function consists of an MSE loss for the NWP forecast and observational data, a kernel MSE that puts more weight in the first and second numerical derivatives[11] of the NWP forecast data, and three physics loss terms, one for each of the (non-hydrostatic) Navier-Stokes equations.

The training and test data sets that we used are shown schematically in Figure 1. The training data consists of the hourly data from a single forecast made with NWP model and the observational data that was measured after this forecast was produced upto the moment that the moment that the forecast is going to be updated. The test data consists of the analysis from later forecasts that are produced at 3 hourly intervals and observational data made during that time. The physics loss terms were evaluated at 50000 residual points. These indicated in Figure 1 as well.

The training procedure used in our prototype is illustrated in Figure 2(a). The positional data from the training data and the residual points are passed as input to the NNs which return predicted values for each variable. These predicted values are used to compute the MSE and kernel MSE loss terms with respect to the true values of the variables from the training data. They are also used to compute the physics loss terms. At the end of each training epoch, the gradient of the combined loss terms is used to update the weights of the NNs using the L-BFGS algorithm as implemented in Tensorflow Probability with the tolerance set to 10^{-16}, the number of correction pairs to 500, and the maximum number of line search steps to 50.

The very first time, that the NNs are trained to update the forecast, the weights and biases of the first layer are initialized as specified in the last two columns of Table 1. The other layers are initialized such that they act as the identity. A total of 50 such models are trained for 50 epochs while all weights and biases are kept fixed except for those of the first layer. The best of these models is selected for further training. This model is trained for 100 epochs more. Then the weights and biases of the second layer are allowed to be trained as well and an additional 100 epochs of training are applied. This is repeated until all weights and biases are trained and the model has been initialized. When a previously trained model is available this initialization procedure is not necessary and the weights and biases from that model are used instead. It takes $O(10^4)$ epochs to obtain an trained system that is sufficiently accurate. Once this accurracy is reached $O(10^3)$ epochs of training suffice to incorporate new observational data.

Note that in our prototype we don't include closures for unresolved processes similar to the physical parameterizations used in weather and climate models. Also note that our prototype has no separate

[10] Or in our application it is better to state that none of the data is irrelevant.

[11] More precisely, the first and second order numerical differences.

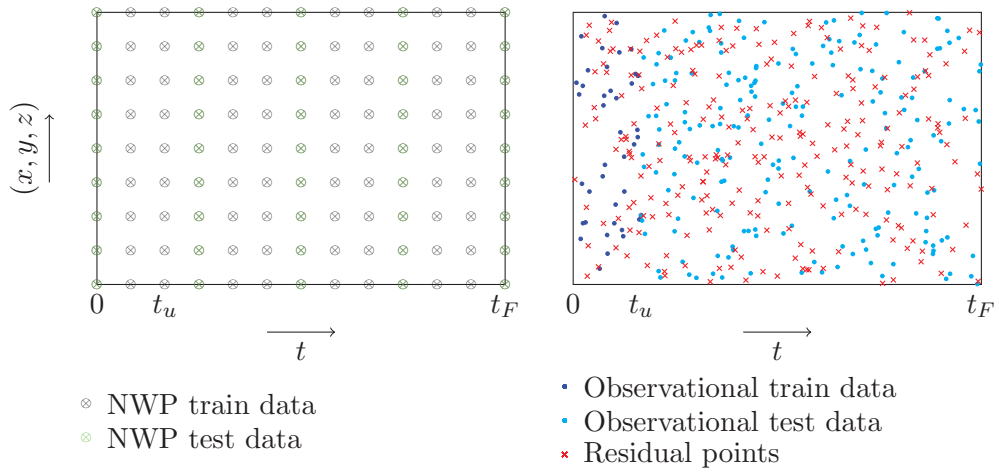

Figure 1: Schematic overview of the data used to evaluate the prototype. The horizontal axis represents the time t and the vertical axis the spatial dimensions x, y and z. The data of the current NWP, that is used for training, is available on hourly intervals as indicated by the crossed circles. The analysis of later forecasts, that is available every three hours (green crossed circles), is used for testing. The observational data used for training is indicated by the blue dots and those used for testing by the cyan dots. The residual points used to impose laws from physics are indicated by the red crosses.

training phase due to the the small amount of data on high-impact weather. Due to these two facts, re-training is not needed when the NWP model is updated or when observational data additional sources are added. Even when sufficient training data would be available, a separate training phase would still be computationally expensive and/or time consuming and would delay the use of an updated ML system after each new version of the NWP model.[12]

4 Results

To evaluate our prototype ML system we used the same 40×40 grid as described in the previous section with every second model level starting from model level 5, a total of 30 model levels, and a forecast length of 9 hours. All the tests were done using 3 GPUs on a single node of the snellius system at SURF.[13] The first GPU was used to perform all computations on NWP forecast data, the second one for all computations involving physics, and the third for the observational data and running the optimizer. In this configuration, the computational performance of the prototype is such that 9 hours long forecasts can be updated every 15 minutes of wall clock time with about 1000 epochs of training which is sufficiently fast to warrant practical application of this system.[14] For the first update the system was initialzed according to the procedure described in the previous section and thereafter was trained for 5000 epochs.

Forecasts were updated each time with 10 minutes worth of new observational data. At the start of each interval, the observational data of the last interval was added to the training data (and removed from the observational test data). Every three hours, the training and test data from the NWP model were updated. At all times, only observational data measured after the start of the current forecast was used as train and test data. One of the test cases is the storm Darcy that passed over the Netherlands in the first week of February, 2021. We ran the prototype using the data from February 4 until February 10. The total loss and individual loss terms after pretraining the initialized system and for the first hour of updating the NWP forecasts are shown in Tables 2, 3, and 4. After the pretraining the total loss terms both for the training and the test data keep decreasing. This is also the case for

[12] And requires many (re)forecasts made with the new version of the NWP model.

[13] Suitable hardware on AWS was hard to get so we have not been able to do tests there. Note that at AWS the ratio of CPUs per GPU is very large compared to nodes on snellius and much more than needed for our prototype.

[14] Technical improvements, such as using a more efficient programming language, should lead to better performance.

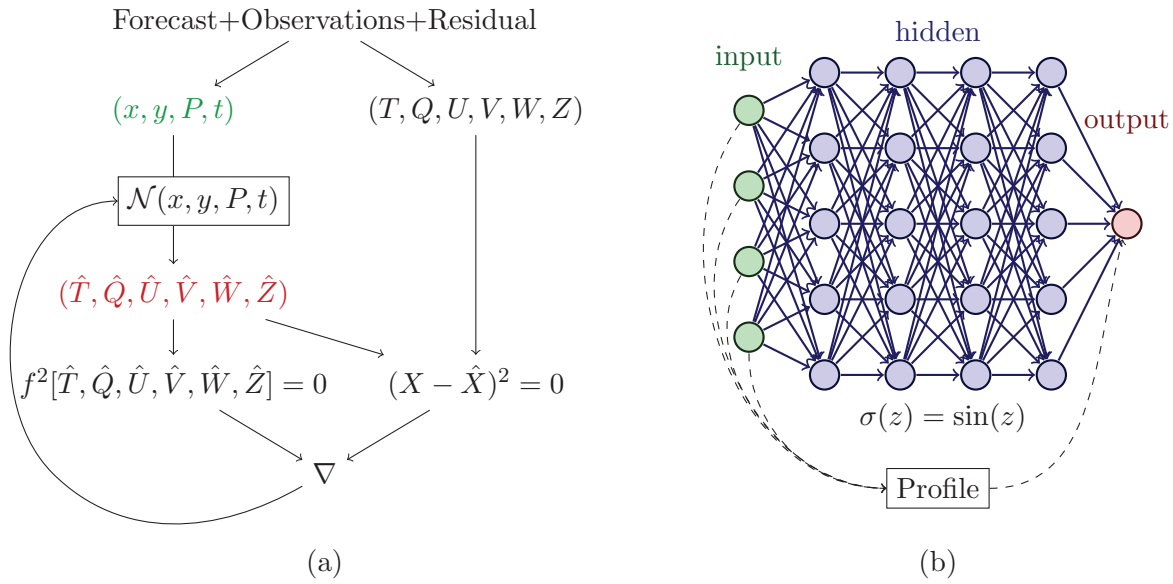

Figure 2: (a) Schematic overview of the training procedure used in the prototype, and (b) the MLP network architecture with the (optional) profile used for each variable. Note that the number of neurons in the hidden layers is only illustrative. The activation function, $\sigma(z)$, is used in all hidden layers.

the train and test loss terms for the NWP data. The loss terms for the observational data fluctuate a bit but are more or less constant. The physics loss terms have a tendency to increase slightly when more observational data is added. These results are in agreement with the results of all our earlier tests and experiments.

Stage	Data	Loss						
		T	Q	U	V	W	Z	Total
Pretraining	Train	1.117e-01	1.472e-01	1.746e-01	1.657e-01	5.917e-01	1.200e-02	1.218e+00
	Test	1.161e-01	1.069e-01	1.413e-01	2.806e-01	4.121e-01	1.766e-02	1.134e+00
00h00m	Train	5.809e-03	1.943e-02	1.717e-02	2.937e-02	1.045e-01	8.219e-06	1.765e-01
	Test	1.065e-02	2.210e-02	4.779e-02	1.668e-01	1.692e-01	2.331e-05	4.399e-01
00h10m	Train	5.318e-03	1.750e-02	1.526e-02	2.545e-02	8.793e-02	8.298e-06	1.517e-01
	Test	1.014e-02	2.178e-02	4.691e-02	1.654e-01	1.641e-01	2.175e-05	4.322e-01
00h20m	Train	4.884e-03	1.595e-02	1.379e-02	2.234e-02	7.607e-02	7.294e-06	1.333e-01
	Test	1.010e-02	2.184e-02	4.706e-02	1.641e-01	1.605e-01	3.297e-05	4.279e-01
00h30m	Train	4.345e-03	1.466e-02	1.250e-02	1.983e-02	6.747e-02	7.467e-06	1.190e-01
	Test	9.437e-03	2.103e-02	4.632e-02	1.630e-01	1.576e-01	2.325e-05	4.226e-01
00h40m	Train	4.027e-03	1.346e-02	1.142e-02	1.766e-02	6.029e-02	7.211e-06	1.071e-01
	Test	7.946e-03	2.115e-02	4.591e-02	1.623e-01	1.553e-01	2.111e-05	4.187e-01
00h50m	Train	3.804e-03	1.241e-02	1.064e-02	1.607e-02	5.479e-02	6.295e-06	9.802e-02
	Test	8.686e-03	2.098e-02	4.550e-02	1.614e-01	1.535e-01	2.346e-05	4.144e-01
01h00m	Train	3.579e-03	1.159e-02	1.068e-02	1.546e-02	5.040e-02	5.769e-06	9.212e-02
	Test	8.396e-03	2.091e-02	4.547e-02	1.606e-01	1.515e-01	2.363e-05	4.080e-01

Table 2: Loss terms for the NWP train and test data after the pretraining and for the first hour of updated forecasts for the Darcy case. The last column gives the total loss.

During the first 8 hours the system keeps updating the forecast in the same manner. But after 9 hours, the total loss increases stepwise by a factor of ≈ 5 (see Tables 5, 6, and 7 and Figure 3). Similar increases occasionally occur later on and take some time to disappear. Note that a small increase is to be expected because the time window is shifted by 3 hours leading to greater change in the train and test data than at all other 10 minute intervals. Due to this greater change in data, the previously trained model will in general not fit as well to the new data.

When we study the loss terms in more detail we see that both the physics and observational loss

Stage	Data	Nr. Obs.	Loss				
			T	Q	U	V	Z
Pretraining	Train	7	4.052e-04	7.527e-04	4.741e-03	6.100e-04	7.406e-04
	Test	610	1.430e-03	4.564e-02	7.967e-03	1.011e-03	2.972e-03
00h00m	Train	7	2.260e-05	1.474e-06	1.709e-05	1.276e-04	8.250e-07
	Test	610	8.837e-04	5.303e-03	1.205e-02	5.057e-03	1.661e-05
00h10m	Train	17	2.032e-05	2.181e-05	2.629e-05	1.334e-04	7.524e-06
	Test	593	6.537e-04	5.083e-03	1.294e-02	5.136e-03	1.421e-05
00h20m	Train	27	2.953e-05	2.882e-05	3.123e-05	9.233e-05	9.355e-06
	Test	583	7.902e-04	5.440e-03	1.257e-02	5.517e-03	1.561e-05
00h30m	Train	37	2.851e-05	4.811e-05	3.910e-05	7.468e-05	9.030e-06
	Test	473	8.046e-04	5.542e-03	1.309e-02	5.852e-03	1.236e-05
00h40m	Train	47	1.931e-05	5.602e-05	6.760e-05	7.377e-05	9.503e-06
	Test	463	1.030e-03	6.202e-03	1.277e-02	6.151e-03	1.315e-05
00h50m	Train	57	2.460e-05	5.415e-05	8.017e-05	9.603e-05	9.739e-06
	Test	463	8.771e-04	5.973e-03	1.160e-02	5.876e-03	1.248e-05
01h00m	Train	74	1.831e-05	4.201e-05	1.178e-04	1.818e-04	9.177e-06
	Test	543	9.831e-04	6.136e-03	9.521e-03	4.343e-03	1.487e-05

Table 3: Loss terms for the observational train and test data after the pretraining and for the first hour of updated forecasts for the Darcy case.

Stage	Loss		
	Continuity	Momentum	Energy
Pretraining	3.176e-05	7.478e-03	1.064e-06
00h00m	1.566e-05	1.256e-05	4.028e-07
00h10m	1.636e-05	1.249e-05	4.169e-07
00h20m	1.765e-05	1.150e-05	4.456e-07
00h30m	1.855e-05	1.075e-05	4.648e-07
00h40m	1.981e-05	1.165e-05	5.011e-07
00h50m	2.067e-05	1.054e-05	5.235e-07
01h00m	2.204e-05	1.002e-05	5.681e-07

Table 4: Loss terms for the physics after the pretraining and for the first hour of updated forecasts for the Darcy case.

terms remain small. This is also the case for the T and Z loss terms for NWP data. The other loss terms for the train data increase by a factor ≈ 5 for Q, ≈ 4 for U, ≈ 5 for V, and ≈ 6 for W. Remarkable is that the loss terms for the test data for these variables don't change this much. Since the test data consists of analysis data from future NWP forecasts, this suggests that the current NWP forecast data is somehow related to this phenomenon. Similar behavior occured when we applied the prototype to other cases of extreme weather. A further detailed investigation is required to determine whether this is indeed the case.

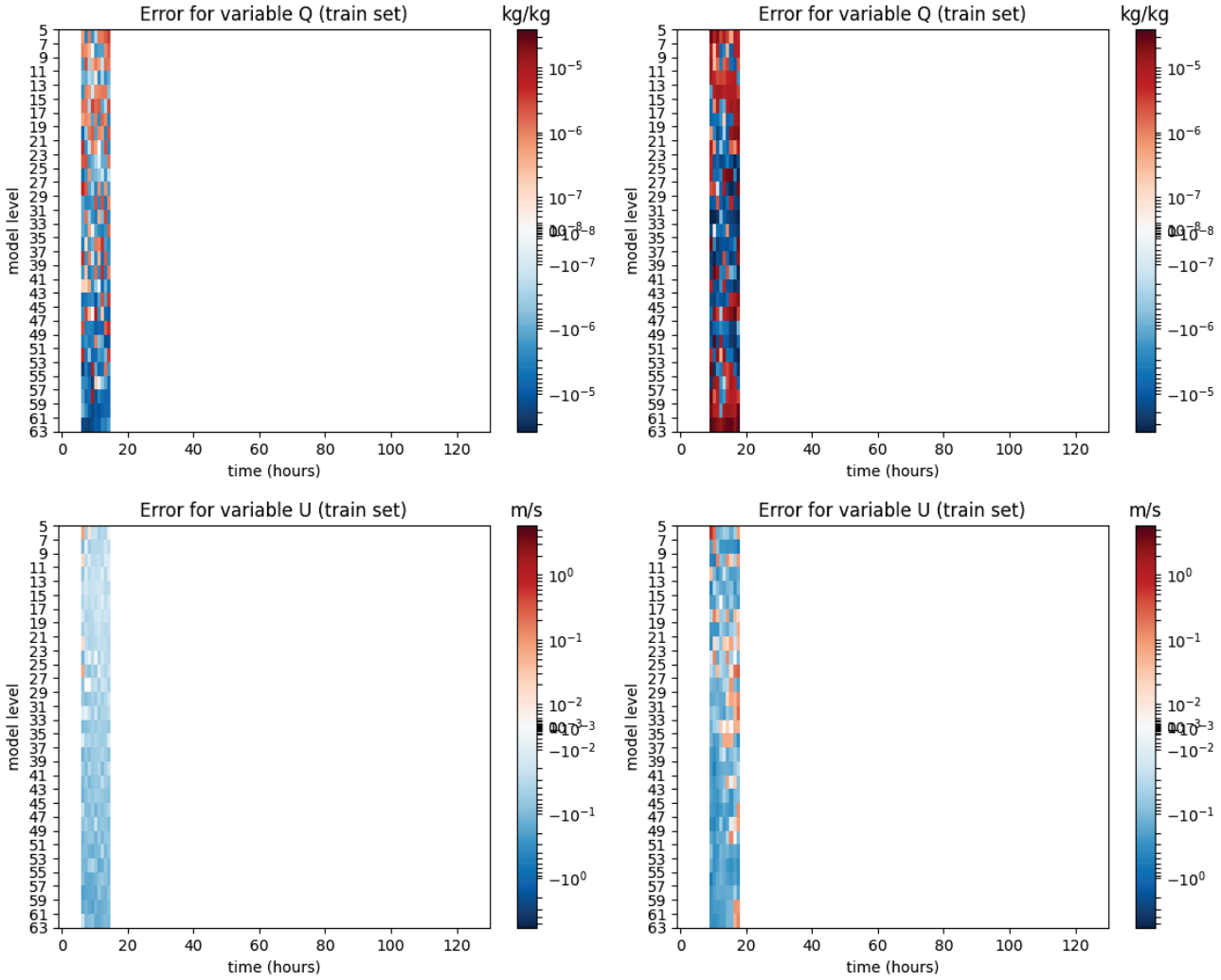

Figure 3: The error in the specific humidity and the zonal velocity with respect to the NWP training data averaged over the model levels. The left plot is for the updated forecast of 8h50m and the right one for 9h00m.

Stage	Data	Loss						
		T	Q	U	V	W	Z	Total
08h50m	Train	2.444e-03	3.972e-03	2.021e-03	5.515e-03	2.537e-02	3.511e-06	3.981e-02
	Test	6.040e-03	4.789e-02	1.932e-02	9.622e-02	4.192e-01	6.556e-06	5.992e-01
09h00m	Train	2.810e-03	1.898e-02	7.906e-03	2.651e-02	1.340e-01	6.790e-06	1.903e-01
	Test	6.459e-03	7.329e-02	2.198e-02	1.529e-01	3.265e-01	1.741e-05	5.921e-01

Table 5: The change in the loss terms for the NWP train and test data from 08h50m to 09h00m for the Darcy case. The last column gives the total loss. Note the large increase in the train loss terms for Q, U, V, and W and the absense of such a large change in the corresponding test loss terms.

Stage	Data	Nr. Obs.	Loss				
			T	Q	U	V	Z
08h50m	Train	191	2.029e-04	1.874e-04	2.064e-05	4.893e-05	9.096e-06
	Test	419	4.393e-04	2.132e-03	5.497e-03	2.533e-03	1.335e-05
09h00m	Train	7	1.478e-05	7.961e-06	1.456e-05	2.546e-05	1.710e-06
	Test	610	5.481e-04	3.617e-03	4.673e-03	2.108e-03	1.442e-05

Table 6: The loss terms for the observational train and test data from 08h50m to 09h00m for the Darcy case. Note that there is no large change in any of these terms.

Stage	Loss		
	Continuity	Momentum	Energy
08h50m	1.495e-05	5.838e-06	4.004e-07
09h00m	1.098e-05	5.589e-06	2.399e-07

Table 7: The loss terms for the physics from 08h50m to 09h00m for the Darcy case. Note that there is not large change.

5 Recommendations For Operationalization

Besides the investigation of the occasional large increase in the loss terms described in the previous section, there are several issues of a technical nature that need attention before a physics-guided ML system can be used to update the NWP forecasts with new observational data. First, the performance should be improved. This would shorten the amount of time it takes to spinup the system and would allow the forecast to be updated with an increased number of training epochs if an better accuracy is desired. Since the prototype doesn't use any of the standard architectures from Tensorflow, much of the code is in Python. An implementation in a programming language like C++ would increase computational speed significantly. Second, a larger number of GPUs can be used so that all of the Netherlands can be covered by the system. This can be done by running one larger system or by running several smaller systems in parallel. Third, it might be possible to speed up the evaluation of the physics loss by explicitly modelling first order derivatives. This will increase the amount of GPU memory needed for the model weights. This should be investigated only when the total runtime of the system is determined by the physics loss. Finally, the input and output of the ML system needs to be adapted to the operational data stream.

Further points for fine tuning such a system are using more residual points and changing the relative weights of loss terms and training data. An operational version can (and should) also use more observational data such as WoW, Mode-S, and radar data to update the forecast. The system could also be extended with more physical variables and processes. It might also be worth to check whether running the system on TPUs instead of GPUs can increase performance.

References

[1] Factsheet KNMI EWC, Bouwen aan een Early Warning Center t.b.v. een duurzame toekomst voor het KNMI, 2019.

[2] F. Chevallier, F. Chéruy, N. A. Scott, and A. Chédin. A neural network approach for a fast and accurate computation of a longwave radiative budget. *Journal of Applied Meteorology*, 37:1385–1397, 1998.

[3] E. Jacks, J. Davidson, and H. G. Wai. Guidelines on early warning systems and application of nowcasting and warning operations. World Meteorological Organization, 2010.

[4] V. M. Krasnopolsky, M. S. Fox-Rabinovitz, and D. V. Chalibov. New approach to calculation of atmospheric model physics: Accurate and fast neural network emulation of longwave radiation in a climate model. *Mon. Weather Rev.*, 133:1370–1383, 2004.

[5] P. A. O'Gorman and J. G. Dwyer. Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *Journal of Advances in Modelling Earth Systems*, 10:2548–2563, 2018.

Royal Netherlands Meteorological Institute

PO Box 201 | NL-3730 AE De Bilt
Netherlands | www.knmi.nl