Using modern machine learning techniques for improved post-processing of probabilistic NWP forecasts

Stephan Rasp and Sebastian Lerch

December 4, 2017

This will be the abstract.

1 Introduction

Numerical weather forecasts continue to be unreliable and biased (**?**) making post-processing necessary. The general premise of post-processing takes many forecast-observation pairs from the past and estimates error statistics to correct future forecasts. Probabilistic forecasts in NWP are usually achieved using ensemble techniques where the same of different models are run many times with slight changes to the initialization or representation of the physical processes in the model. ECMWF, for example, produces a 50-member ensemble. Ideally, a probabilistic forecast is sharp and reliable. Sharp means that the spread of the ensemble members, i.e. the uncertainty, should be small. But this is only valuable if the forecasts are reliable: The forecast distribution has to match the conditional true distribution (**?**). Typically, NWP models are underdispersive which means that their ensemble spread does not reflect the real uncertainty and often the corresponding observation falls outside of the ensemble spread. These forecast are therefore not reliable.

The goal of

1.1 Aims

The main aim of this study is to test modern machine learning techniques for the problem of post-processing probabilistic forecasts. Our aim is not to get the best possible score. For this reason we did not conduct a lot of hyper-parameter tuning.

1.2 Notation

We now introduce the notation we are using throughout the paper. $x_{m,s,t}$ is the 48 h-forecast for a variable, or in machine learning terms a *feature*, x for an ensemble member $m \in \{1, \ldots, M\}$, a station $s \in \{1, \ldots, S\}$ and a verification time $t \in \{1, \ldots, T\}$. In the simplest case the feature vector $\mathbf{x} = (x_1, \ldots, x_{N_{\text{features}}})$ contains only the temperature forecasts, either all members or reduced to the mean and the standard deviation. In this study however we also add further variables to $\mathbf{x}$. $y_{s,t}$ is the corresponding observation of 2 m temperature, or the *target*. The post-processed forecasts,

or *predictions*, made by our models is denoted by $\hat{y}_{s,t}$. Since we are dealing with probabilistic predictions, our $\hat{y}$ can take the form of a parametric PDF or binned probabilities. In case of temperature a normal distribution is a good approximation. In this case $\hat{y} = \mathcal{N}(\mu, \sigma^2)$ and we are predicting two values, the mean μ and the standard deviationσ. In case of binned probabilities $\hat{y} = (p_1, \ldots, p_B)$ for (not necessarily uniform) bins $b \in \{1, \ldots, B\}$ under the constraint that $\sum_{b=1}^{B} p_b = 1$. Lastly, we are merging the station and time dimensions into a single sample dimension $i \in \{1, \ldots, N_{\text{sample}}\}$.

2 Dataset

For this study, we focused on 2-meter temperature forecasts at synop stations in Germany at a forecast lead time of 48 h. Data is available from 1 January 2008 to 31 December 2016 every day. For details on the data used for training and verification, see Section 5

2.1 ECWMF TIGGE forecast dataset

The forecasts for this study are taken from the TIGGE dataset (**?**). In particular, we used the ECMWF 50-member ensemble forecasts started at 00UTC every day. The data in the TIGGE archive is upscaled to a 40 km horizontal grid. For comparison with the station observations, the gridded data were interpolated to the observation locations using some sort of algorithm.

In addition to the forecasts for the target variable, we have several auxiliary variables.

How are 2m temperatures interpolated?

2.2 DWD SYNOP data

Observations from 537 (is this correct) weather stations in Germany are used (see Fig. ???). Some data points are missing. These data were omitted.

Where did we get it from?

We are not taking into account observation error. Reference for how large that is.

2.3 Data augmentation

3 Traditional post-processing techniques

3.1 EMOS

3.2 Boosting

4 Modern machine learning techniques

4.1 (Neural) networks[1]

Rather than explaining all the basics of neural networks, we refer to two great resources on neural networks (**??**) and keep our discussion brief and visual (Fig???). The parameters of a neural network

[1] A note on terminology: Many of our simpler models are not neural networks or even networks in the traditional sense since they linearly connect the features and variables. In this regard we are basically doing linear regression using stochastic gradient descent. We will nevertheless use the term "network" to describe all models, but only use the term "neural" if there is a hidden layer and a non-linear activation function in the model.

are trained using stochastic gradient descent (SGD). In particular we are using a version of SGD called Adam (**?**).

The traditional EMOS technique can also be seen from a network perspective (Fig???a). In this case not all the nodes in the input and output layers are connected. It is simple to extend this approach to a fully connected linear network (Fig???b) Which allows to incorporate arbitrarily many input and output nodes. A neural network includes one or more hidden layers with non-linear activation functions. For this study we use the relu activation function $g(z) = \max(0, z)$.

If we want probability bins as an output, we use the Softmax function as an activation for the output layer:

$$g_b(\mathbf{z}) = \frac{e^{z_b}}{\sum_k e^{z_k}} \tag{1}$$

This function squashes an input vector $\mathbf{z}$ to a vector of probabilities between zero and one.

4.1.1 CRPS loss function

As a loss function for our probabilistic forecasts we use the Continuous Ranked Probability Score (CRPS; **?**), which for one forecast-observation pair is defined as

$$\mathrm{crps}(F, y) = \int_{-\infty}^{\infty} [F(t) - H(t - y)]^2 \mathrm{d}t, \tag{2}$$

where $F(t)$ is the forecast CDF and $H(t-y)$ is the Heaviside function. For a Gaussian forecast PDF it is possible to write down an analytic version of the CRPS (**?**):

$$\mathrm{crps}[\mathcal{N}(\mu, \sigma^2), y] = \sigma \left\{ \frac{y-\mu}{\sigma} \left[2\Phi\left(\frac{y-\mu}{\sigma}\right) - 1 \right] + 2\varphi\left(\frac{y-\mu}{\sigma}\right) - \frac{1}{\sqrt{\pi}} \right\}, \tag{3}$$

where $\Phi\left(y-\mu/\sigma\right)$ and $\varphi\left(y-\mu/\sigma\right)$ are the CDF and PDF of $\mathcal{N}(0,1)$ evaluated at the normalized prediction error $(y-\mu)/\sigma$.

For probability bins the CRPS can be written as a step-wise combination of uniform distributions within the bins. A step-for-step guide of this method is illustrated in the notebook (???).

@Sebastian: Do you think it would be interesting for people to write down in detail how to calculate the binned CRPS? If not I would just refer to the Rmarkdown or Jupyter notebook script.

Since the exact computation of the discrete CRPS involves some computation steps which cannot be vectorized, we are using an approximate version as a loss function for the neural networks.

4.1.2 Embeddings

An embedding is a mapping from a discrete object to a vector of real numbers. In our case each station id is mapped to a 3-dimensional vector. The dimensions of the vector represent some property of the station. The values of the embedding vectors, also called latent features, are concatenated to the usual input features. During training, the latent features are learned along with the weights and biases of the network.

4.1.3 Auxiliary variables

As with the traditional boosting methods we can add auxiliary variables as features. A list of variables is given in Table???. For each variable we compute the mean and standard deviation.

4.1.4 Using previous errors as features

If forecast errors are persistent in time, we can use the error of the forecast verifying at time t to post-process the forecast for time $t + 48\,\mathrm{h}$. For this we are adding the ensemble mean forecast, the observation and the ensemble mean error at time t as input features.

4.2 Discrete forecasts

Explain how data was converted to bins.

4.3 Gradient boosted trees

We are using XGBoost

5 Results

5.1 CRPS for 2016

Our validation metric is the mean CRPS for all stations and days of 2016. Ignoring missing values this amounts to 182,218 samples in the validation set. Table??? summarizes the score for all of our network configurations. We did some basic manual hyper-parameter tuning which is noted down in the table as well.

5.2 Computational cost

Table??? shows the training time for all of the algorithms. All models were trained on the same workstation, no GPU was used. Most certainly, significant speed-ups could be achieved by optimizing each algorithm. Here, we simply wanted to show the huge difference between the traditional approaches, which take many hours, to the network approaches, which take seconds.

5.3 Feature importance

Simple linear neural networks and XGBoost allow us to get an idea about how important each feature is.

6 Things we tried but didn't work

6.1 Recurrent neural nets

7 Discussion and conclusion