August 2019

Attempts at using Bayesian neural networks for uncertainty assessments of temperature forecasts

Joel Lundqvist

Department of Physics, Division of Combustion Physics, Lund University

Bachelor thesis supervised by Fredrik Karlsson and Elna Heimdal Nilsson

LUND
UNIVERSITY

(This page is intentionally left blank)

Abstract

This thesis describes attempts at estimating the uncertainty of the 2-metre temperature forecast error from a probabilistic point of view, utilizing Bayesian neural networks. Bayesian neural networks are a type of machine-learning algorithms used to find patterns in data and make probabilistic predictions. Multiple fields of output data from the ECMWF IFS global model, along with temperature measurements from two meteorological observation stations for a period of six years are used for training of the networks. Attempts are made to assess probability distributions for the error as a continuous variable and through approaching the task as a binary classification problem. None of the attempts described were successful in terms of producing useful predictions, but may serve as a starting point for further investigations.

Contents

1 Introduction

Modern weather forecasting relies on numerical weather prediction (NWP), which is computer simulations of the atmosphere. They build upon differential equations whose boundary conditions are determined through observation, and then integrated forwards in time. Weather forecasts are of importance for many branches of society. The 2 metre temperature, that is, the temperature measured two metres above the ground, is a weather model output parameter to which most of us ground dwelling creatures have an everyday relationship. Small differences in this variable may result in vastly different outcomes. For example, when the temperature is close to 0 °C, it can be a matter of fractions of degrees that determine whether or not ice forms on roads, which may pose a major traffic hazard. Weather forecasts are however always, to some extent, uncertain. Some of the sources of uncertainty include measurement errors when specifying initial conditions for the differential equations, and errors introduced as the equations are discretized to the finite resolution of the models. To improve the accuracy, the models are continuously being updated, increasing spatial and temporal resolution as computer power becomes cheaper. But as the atmosphere is a non-linear system, a perfect forecast will never be made.

The error may be dealt with in various ways. A direct method is through improving the forecasts of a certain parameter at a specific site using Kalman-filters, a statistical method for improvement of predictions using previous observations and forecasts of that parameter. The full uncertainty can be assessed through the use of ensemble methods, where multiple models are run with deliberately introduced small observation errors. The simulations will diverge from one another, and their spread may be taken as an indication of the uncertainty of the current weather situation. Some problems with quantifying the uncertainty using the ensemble methods for early time-steps in the simulations is that the variation among its members has not yet grown enough to describe the variation of possible outcomes, and that they are computationally heavy and takes a long time before they are available to forecasters.

Artificial neural networks are a group of machine-learning algorithms that has been applied in numerous fields where great amounts of data are available, such as speech and image recognition and machine translation (Vasilev et al., 2019). Bayesian neural networks are a subgroup of these algorithms that can be used to make probabilistic predictions.

The aim of this thesis is to construct Bayesian neural networks that can give estimates of probability distributions for the error in the 2 metre temperature forecast. To achieve this, output data for a selection of parameters from a global NWP-model, along with observations of the 2 metre temperature from two meteorological stations will be processed by the networks. If relations between the output data and the observed errors can be identified by the networks it may provide a useful complement to the forecast of the 2 metre temperature for that specific location.

2 Background

2.1 Numerical weather prediction

2.1.1 Historical background

The idea of treating weather prediction as an initial value problem of atmospheric physics dates back to the early 20th century (Bauer et al., 2015). At that time, routine observations were sparse and no computers existed. But as of today this approach forms the basis of weather prediction. Attempts of solving the governing equations by hand to produce a so called *hindcast* were conducted in the 1920s (Inness & Dorling, 2013). A hindcast is a method of testing a weather model where historical data is used to initialize the model and then compare the solutions with observations. The calculations took a very long time and went horrifically wrong, forecasting a pressure drop of 145 hPa during six hours (Lynch, 2008). For comparison, very rapidly deepening low-pressure systems are referred to as *cyclone bombs* if their central pressure sinks by 1 hPa/h during a 24 hour period (Ahrens & Henson, 2013). At that time the error was blamed on poor observations, but more important was actually the lack of filtering which left computationally unstable solutions possible, allowing waves that propagate faster than over one grid length per time-step (Holton & Hakim, 2013). After the second world war, with the advent of early computers and the newly developed equations of the quasi-geostrophic approximation which filtered out too quickly propagating waves, the first successful hindcasts were made (Charney, 1948; Lynch, 2008). Modern forecasting centers have been able to go back to the full set of equations tried in twenties, as computational power has become cheaper (Kalnay, 2003). These equations are introduced and briefly described in section 2.1.2.

But no matter how well the models describe the physics, eventually the path predicted by a model will deviate from that of reality (Lorenz, 1963). As the atmosphere is a non-linear system, any perturbations to initial conditions will eventually grow through feedback mechanisms, and cause a completely different evolution of the system. The sensitivity to perturbations are much smaller than the accuracy of the instruments used to make observations (Inness & Dorling, 2013). However, as the sensitivity to perturbations varies from one situation to another, the resulting uncertainty is difficult to assess directly. Today, and since the 1990s, the main method for quantification of the uncertainty are through the method of ensembles. The idea is quite simple. Instead of running only one simulation of the atmosphere, many models are run in parallel, but with slightly different perturbations added to their initial conditions. The resulting spread may taken as possible outcomes. This is useful when looking at large scale phenomena, like prediction of storm tracks. But as it takes time before the non-linear effects dominate the evolution of the simulations, the uncertainty regarding early time-steps are not always well caught by this method.

2.1.2 Primitive equations

Equations (1) to (4) are defined as in *An Introduction to Dynamic Meteorology* (Holton & Hakim, 2013), and Eq. (5) as in *Atmospheric modeling, data assimilation and predictibility* (Kalnay, 2003).

The momentum equation that describes the three-dimensional acceleration for a parcel of air in a rotating coordinate-system can be expressed as

$$\frac{D\boldsymbol{U}}{Dt} = -\alpha\nabla p + \boldsymbol{g} + \boldsymbol{F}_F - 2\boldsymbol{\Omega} \times \boldsymbol{U}, \tag{1}$$

where $\boldsymbol{U} \equiv (u, v, w)$ is the three-dimensional velocity (where in turn u, v and w are the zonal, meridional and vertical velocities, respectively), α is the specific volume ($\alpha \equiv \frac{1}{\rho}$), p is pressure, $\boldsymbol{g} \equiv (0, 0, -g^*)$ is the gravitational acceleration (where in turn g^* is referred to as the effective gravitational acceleration, and is the sum of the gravitational and the opposing centrifugal acceleration due to the Earth's rotation), $\boldsymbol{F_F}$ is the friction acting on the parcel and $-2\boldsymbol{\Omega} \times \boldsymbol{U}$ is referred to as the coriolis acceleration, which is due to an apparent deflecting force imposed by the Earth's rotation at an angular velocity of $\boldsymbol{\Omega}$.

The continuity equation expresses conservation of mass for an air parcel, and may be written as

$$\frac{1}{\rho}\frac{D\rho}{Dt} + \nabla \cdot \boldsymbol{U} = 0. \tag{2}$$

$\nabla \cdot \boldsymbol{U}$ is the divergence of the wind field.

The equation of state for the dry atmosphere relates the pressure, density and temperature as

$$p = \rho R T \tag{3}$$

where R is the gas constant for dry air.

The first law of thermodynamics is a conservation law for energy, which states that

$$J = c_v \frac{DT}{Dt} + p\frac{D\alpha}{Dt}, \tag{4}$$

where J is the rate of heating per unit mass and c_v is the specific heat at constant volume.

Any change in the local water vapour content must be due to either evaporation (E, a source term), condensation (C, a sink) or transport of water vapour by the wind. This can be expressed mathematically as:

$$\frac{\partial q}{\partial t} = -\left(\boldsymbol{U} \cdot \nabla\right) q + E - C, \tag{5}$$

where $(\boldsymbol{U} \cdot \nabla)$ is the advection operator.

The European Center for Medium-range Weather Forecasting (ECMWF), that runs the global model used in this thesis, make use of the *hydrostatic* approximation (ECMWF, 2018). The hydrostatic approximation assumes hydrostatic equilibrium, an exact balance between the vertical component of the pressure gradient force and gravity, so that that the vertical acceleration is always zero. The vertical component of Eq. (1) then simplifies to

$$0 = -\alpha \frac{\partial p}{\partial z} - g \implies \frac{\partial p}{\partial z} = \rho g. \tag{6}$$

This assumption is valid for large scale processes, as these two forces are orders of magnitude stronger than the other forces acting along the vertical axis, but not always for smaller scale systems, such as convective clouds (Holton & Hakim, 2013).

Equations (1) to (5) form an (almost) closed set of differential equations. If boundary conditions are specified from observations, they may be integrated forwards in time and describe the future evolution of the atmosphere.

2.1.3 Discretization and parametrization

In order for numerical methods to be applied to the equations, the continuous fields of the atmosphere must be discretized, both spatially and temporally. In the ECMWF global model, these fields are represented in a three-dimensional grid. Transformations into a spectral space are used for some computations (ECMWF, 2018).

As the resolution on which the equations are resolved is finite, and some important processes are not even described, the method of parametrization is used. Some examples among these processes are radiative heating, latent heat release and turbulence. The processes are simplified, their effects are approximately quantified, and added as sources or sinks to the set of equations described in the previous section. The 2 metre temperature is sensitive to many of these processes, and the accuracy of the prognosis consequently also influenced. Also, even if some parametrization works well accounting for the influence of some process on the larger scale, it might not be locally accurate.

2.2 Artificial neural networks

Artificial neural networks are algorithms for data-processing. They are used for generation of models that can link observations $\boldsymbol{X}$ to outcomes y, and make predictions $\tilde{y}$ given new observations (Dreyfus, 2005). Initially, inspiration may have been taken from how biological brains once were thought to function, but the field has since then developed independently of neurology (Vasilev et al., 2019). To explain the functioning of an artificial network it may be easiest to begin with describing the basic building block of which they consist, the neuron. A neuron is a function that takes vector input and produces a single scalar output. For an input vector $\boldsymbol{X}$, a neuron generates output $\tilde{y}$ as

$$\tilde{y} = f(\boldsymbol{W} \cdot \boldsymbol{X} + b), \tag{7}$$

where f is referred to as its activation function, the elements of $\boldsymbol{W}$ as its weights and b as its bias. These neurons may then be connected in networks, through having the outputs of some neurons be the input of others. A network that only lets information travel in one direction is called a feedforward network, and is the only type of network applied in this thesis. An example of the structure of such a network is shown in Fig. 1

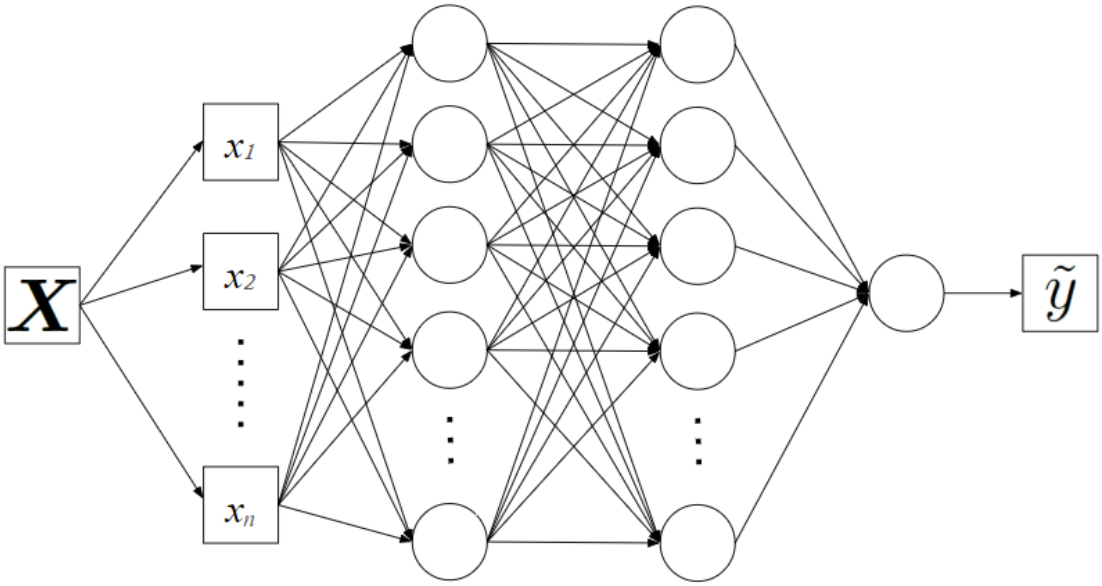


Figure 1: An example of a feedforward neural network structure. Each circle represents a neuron, where the activation function acts on the sum of the bias and the dot product of its weights and input values, see Eq. (7). A column of neurons is referred to as a layer, those not directly connected to input or output are called "hidden" layers. The number of neurons in a layer is referred to as its width.

It can be shown, by universal approximation theorems, that feedforward networks with only one hidden layer can model any continuous non-linear function, with arbitrary accuracy,

given that the activation functions used in the neurons are bounded, continuous and non-constant (Hornik, 1991), or rectified linear unit functions (Lu et al., 2017), given that the modelled functions are within the range of the activation functions. If the modelled function is beyond the range of the activation functions a linear layer may be added as the last layer of the network, but it should be noted that it is not a bounded function and cannot be used as the only activation function in a network (Dreyfus, 2005). Even though the universal approximation theorems state that any continuous function may be modelled, it should be noted that they do not state that the performance of the networks, in terms of computational efficiency, are independent of the choice of activation functions (Hornik, 1991). In Table 1, some common activation functions applied in this thesis are presented.

Table 1: Activation functions used in the neural networks of this thesis.

Name	**Definition**	**Range**
Linear/direct/identity	$f(x) = x$	$(-\infty, \infty)$
Rectified linear unit (ReLU)	$f(x) = max(0, x)$	[0,∞)
Softplus	$f(x) = \ln(1 + e^x)$	$[0, \infty)$
Hyperbolic tangent (tanh)	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1,1)
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	(0,1)

After being set up, the structure of a network is static in terms of how the neurons are connected and their activation functions. In order to find the desired mapping between the input vector $\boldsymbol{X}$ and the output y, the values of its parameters (the weights and biases) are adjusted.

The process of adjusting the weights and biases of the neurons to reach the desired output is called training, and the most commonly used method is called backpropagation (Vasilev et al., 2019). To evaluate how well a network performs, some cost function is used. One of the most common cost functions is the mean squared error function (Dreyfus, 2005), which is defined as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2 . \tag{8}$$

When doing backpropagation, one tries to approximate the mean-squared-error as a function of the parameters of the network. Then its gradient is approximated as well, and is then used to update the weights and biases in the direction of the cost function's minimum. The sources of the errors are traced back to their origins in the parameters of the network. It should be noted though, that the cost functions are typically not smooth and continuous, and several local minima may exist. Updating the parameters using the gradient is thus not a trivial task. For each iteration of the updating process the value of the cost function is saved. During the first cycles, the value of the cost function usually varies quite a lot due to being poorly approximated. Eventually though, if the approximation does get better, the network starts producing more consistent predictions. The fluctuations in the cost function's value between iterations are reduced in magnitude, and in addition to this, the

value shows an overall decreasing tendency. After even more iterations the predictions typically plateau, where further iterations do not improve the fit anymore. This is called convergence, and it is an important sign of a successful network. A network that does not stabilize in this way does not produce consistent predictions, and needs to be redefined in some way before its output is analyzed.

But just because a network seems to have minimized the error as much as possible, it is not for certain that such a network can produce meaningful predictions. When fitting a model to data there is a risk for overfitting. This occurs when the model fits too well to the data trained upon, but fails to generalize on the patterns and produce reasonable predictions when encountering new data. To make investigation of whether or not this is the case possible, the data one wishes to model can be split into training and validation datasets. The training dataset is then used for the process of minimizing the cost-function. The performance can the be evaluated through running the validation dataset through the network without adjusting its parameters. If the performance differs significantly between the training and evaluation datasets, the case might be that the network is overfitted.

2.2.1 Bayesian neural networks

Bayesian neural networks are a subgroup of artificial neural networks that make use of Bayesian inference. This in turns builds upon Bayes theorem, which provides a way to relate the probability of an outcome to other related events (Kruschke, 2015). Most people are intuitively able to apply some basic principles of Bayesian inference, even if they do not formulate it in mathematical terms. To stay within the topic of meteorology, let us make an informal weather-related example of how the basic principles can be applied. For a location where it on average rains every other day, we may make a guess that the probability of rain for any given day is about one half. If we are also given the information that on this particular day, the sky is cloudy, any person who knows that rain falls from clouds will update their belief in the probability of rain for that particular day to a larger value.

Put in mathematical terms, for two discrete variables A and B representing two events, we have their respective probabilities denoted $p(A)$ and $p(B)$. The intersection probability, the probability that the events both occur, are denoted $p(A \cap B)$. These can be used to calculate the conditional probability

$$p(A|B) = \frac{p(A \cap B)}{p(B)}, \tag{9}$$

which is interpreted as "the probability of A given that B has been observed". By symmetry,

$$p(B|A) = \frac{p(B \cap A)}{p(A)} \tag{10}$$

must hold too. As $p(A \cap B) = p(B \cap A)$, by definition, equations (9) and (10) combined give Bayes theorem:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}. \tag{11}$$

It can be applied to relate probabilities for both discrete and continuous variables.

Bayesian neural networks may be structured as feedforward networks, as illustrated in Fig. 1. The weights and biases used here are not scalars, instead they are described by probability distributions. Probability distributions are propagated through the networks, and the uncertainties related to the predictions are quantified. Training such networks is done using either sampling algorithms or variational inference. Most common in Bayesian inference contexts is to use some Markov-chain Monte Carlo method (MCMC), which is a group of sampling algorithms. However, that kind of algorithms become very computationally heavy when doing inference with many parameters. In this thesis, automatic differentiation variational inference (ADVI) is applied. With this method the probability distributions are analytically approximated instead of generated through sampling. This is more computationally efficient, but gained on the cost of accuracy (Kucukelbir et al., 2017).

2.2.2 Two probability distributions

The normal probability distribution will be used for prediction of continuous variables. It is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2\right), \tag{12}$$

where μ determines the position of the distribution along the x-axis, and σ is the standard deviation, and describes the variation around μ.

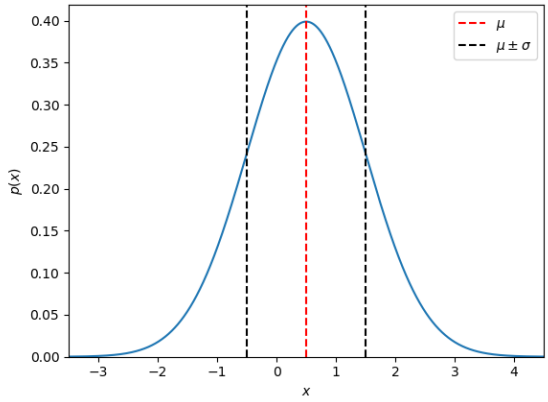


Figure 2: Example of normal distribution. Here $\mu = 0.5$ and $\sigma = 1$.

For binary classification into categories $\gamma \in \{0, 1\}$, the Bernouilli distribution can be used. It is defined as

$$p(\gamma|\theta) = \theta^{\gamma}(1-\theta)^{(1-\gamma)}, \tag{13}$$

where θ is the probability of $\gamma = 1$.

3 Method

3.1 Data

The programming language Python was used for all data processing throughout the project. The data used for training the networks was provided by SMHI, the Swedish Meteorological and Hydrological Institute. The data consists of observations of the 2 metre temperature from two meteorological stations, Östersund-Frösön and Nordkoster, along with output data of the ECMWF Integrated Forecasting System global model for the grid-point nearest to each station, during a period of six years, 2010-2015. The forecasts used are issued 12 hours before their valid time, and the forecast for the 2 metre temperature is compared to the observed value on two occasions each day, 00 and 12 UTC.

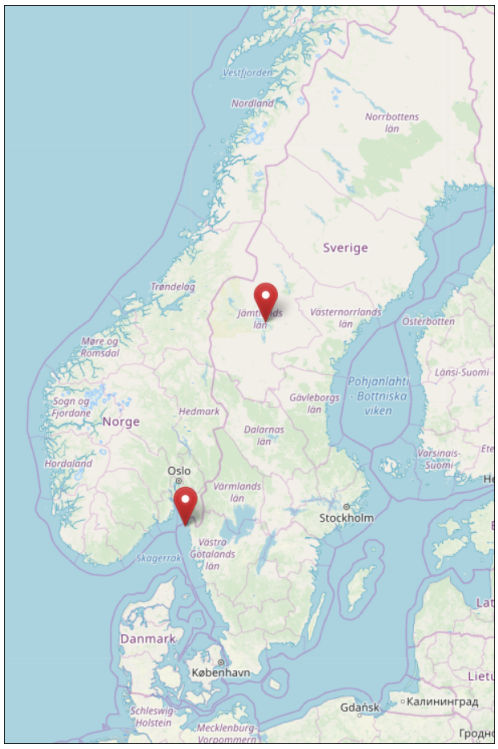


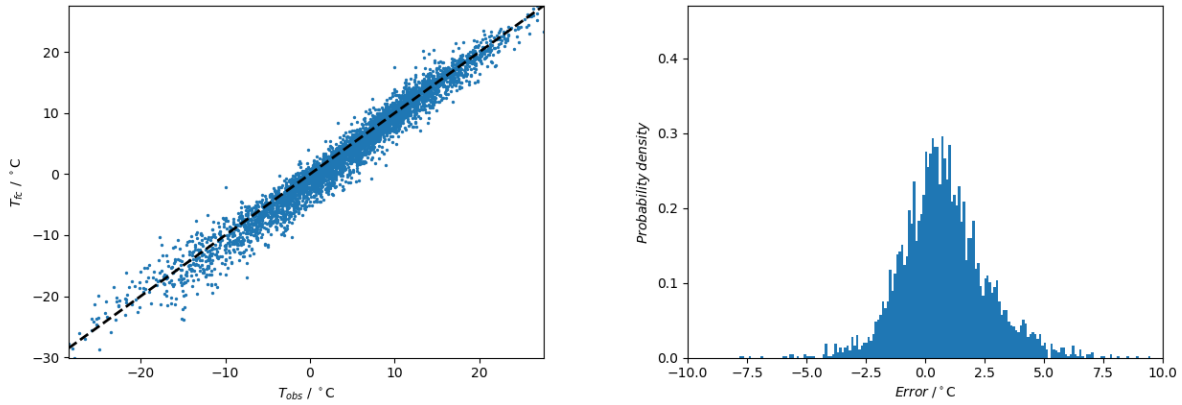
Figure 3: Map of Sweden showing the locations of the measurement stations, where Östersund-Frösön is at the northernmost pin symbol and Nordkoster is at the southernmost one. The map was created using tools provided by the OpenStreetMap project under the CC-BY-SA license (OpenStreetMap contributors, 2017).

The choice of analyzing observations from specifically Östersund-Frösön and Nordkoster was based upon that they differ from one another in various ways, both in terms of their different climatological conditions and how the observations relate to the forecasts. The Östersund-Frösön station is located in inland Sweden, in a mountainous region, in the vicinity of Storsjön, a large lake. The Nordkoster station is at a maritime location on

an island outside the Swedish west coast, in the Skagerrak strait. Defining the error as the difference between the 2 metre temperature, T_{obs}, observed at the station, and the forecasted value, T_{fc}, at its nearest grid-point, as

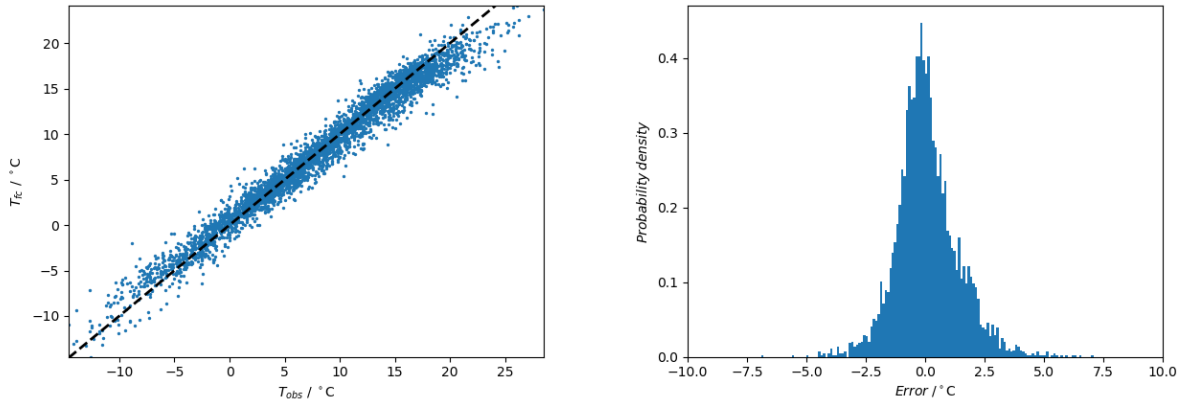
$$Error \equiv T_{obs} - T_{fc}, \tag{14}$$

one finds that the error observed at Östersund-Frösön is generally greater than that for Nordkoster. The mean and median values of the error (0.8 °C and 0.7 °C, respectively) for Östersund-Frösön also indicates that there is a bias for the forecasted temperature to be lower than the observed temperature. The error distribution for Nordkoster is centered closer to zero, with mean and median values of 0.08 °C and −0.04 °C. The errors for Östersund-Frösön being larger might be due to more complex orographic features and the nearby water-surfaces of sub-grid size, that are taken into account through parametrizations rather than being explicitly resolved. The thermal inertia of the ocean probably stabilizes the temperature for the Nordkoster station.



(a) **(b)**

Figure 4: Observations, forecasts and errors of the 2 metre temperature for Östersund-Frösön. (a) shows observed temperatures T_{obs} against forecasted temperature T_{fc}. (b) shows the normed error distribution.



(a) **(b)**

Figure 5: Observations, forecasts and errors of the 2 metre temperature for Nordkoster. (a) shows observed temperatures T_{obs} against forecasted temperature T_{fc}. (b) shows the normed error distribution.

For Östersund-Frösön, at 19 occasions observations of the 2 metre temperature were missing. As this is such a small fraction of the total number of observations these were simply removed from the dataset. There were no missing observations in Nordkoster. The data were split into training and validation data sets. Data from five years, 2010 - 2014, were used for training the networks, and the data for 2015 were saved for validation. The following output fields from the weather model was chosen as input parameters for the networks:

- 2 metre temperature
- Solar elevation
- Surface pressure
- Temperature 850 hPa
- Total cloud cover
- 10 metre u-wind component
- 10 metre v-wind component

Before feeding this data into the networks, the fields in the training dataset were normalized to zero-centered distributions with standard deviations of unity. The same transformations were applied to the fields of the validation dataset.

A similar structure was used for all networks regardless of task: seven input nodes (one for each input field), two hidden layers with 100 and 50 nodes, respectively, and a single output node. The distributions for the weights and biases were also similarly intitialized for all networks. As nothing was known in prior of which distributions for the parameters would produce desired output, the generic choice of using zero-centered normal distributions in all nodes was made. In the nodes of the input and first and second hidden layer all parameters were assigned standard deviations of unity, and the parameters of the output layer were assigned wider standard deviations of ten, to possibly enable greater flexibility for the parameters of that last node. Analysis of output data from any network was not conducted unless convergence was reached.

For networks that did converge, the weights and biases were approximated through sampling of 3000 values from their distributions. A histogram showing a sample distribution for one of the weights after training is provided in appendix, section B, Fig. B.1. These approximate distributions were then used to create "mirrors" of the networks with the same structure and activation functions. Instead of producing a single scalar output as described in Eq. 7, each neuron now produces an array of 3000 scalars, one for each set of sample values of its parameters. For each input vector, corresponding to the model output fields at a specific time-step, one thus gets 3000 different predictions in the output that form an approximation to a probability distribution. For analysis of the output distribution, the 10th, 50th (the median value) and 90th percentiles are computed for each set of 3000 values.

The networks were all constructed and run using the library PyMC3 (Salvatier et al., 2016) with Theano (Al-Rfou et al., 2016) as backend.

3.2 Regression

The aim is to make predictions in the form of probability distributions for the error as a continuous variable, given values from the input parameters. This could give a very direct assessment of the uncertainty of the forecast.

To make evaluations of the performance of these networks, the proportion of the observed errors that fell within the 10th and 90th percentile of the probability distributions of the predictions for each forecast in the validation dataset was calculated. If this proportion is close to 0.8, the performance of the network is good. In a case where the proportion is close to one, may be a signal of perfect fit, or that the probability distributions are too wide to give meaningful information. Such cases should thus be carefully investigated before the predictions are trusted. A proportion significantly lower than 0.8 means that the predictions do not catch the observed error. To check for signals of overfitting, the performance by this measure can also be evaluated for the predictions for the training dataset. If a significantly higher proportion of the training dataset predictions catch the error, it can be taken as a sign for overfitting.

3.2.1 Fixed standard deviation

Here, the aim is to find a normal distribution for the error through predictions only of μ, while letting the standard deviation of the output probability distribution be fixed to some value. The variation of the error may then be caught in the variation of the predictions for μ. The hyperbolic tangent function was used as activation function for the two first layers, and for the last layer the linear function was used. This setup of activation functions produced converging networks for this task. Two attempts were made, with different values for the fixed standard deviation. First, the standard deviation was fixed to an unrealistically small value of 0.01. The second attempt was conducted by fixing the standard deviation to the value of the standard deviation of the error in the training dataset.

For implementation of these attempts in code, see appendix, section A.2.

3.2.2 Simultaneous determination of standard deviation and mean value

The aim for this approach is also to generate normal distributions for the error, but now through prediction of values for both mean and standard deviation simultaneously, using parallel structures in the same network. Thus, the predictions have associated uncertainty regarding both the mean and the standard deviation. However, constructing such a network proved a challenge that was not overcome. As it was difficult to find a network that did not give nan-errors, and the networks that were found stable in this sense did not converge, all attempts were eventually abandoned. Several functions were tested in the output layer: linear, softplus and exponential. With the linear function, the algorithm found only infinite values in the cost function, and with the softplus and exponential function convergence was never reached.

Some example code for implemenating a network that does not give infinity errors, but neither shows convergence, is found in appendix, section A.3.

3.3 Binary classification

Instead of assessing the error as a continuous variable, a classification approach may be taken. Here, two classification approaches to the problem of uncertainty are presented, classification based on the absolute value of the error, or on the observed temperature directly. The networks used for these tasks were similar in structure, only differing in the data trained against. The ReLU function was used in the first two layers of the network. For the output node, the sigmoid function was used. These activation functions produced networks that could converge. I could not come up with any analogue to the performance evaluation method used for the regression approach to make comparisons between different networks used for binary classification.

3.3.1 Error size

The observed errors can be split into two categories γ defined as

$$\gamma = \begin{cases} 0, & \text{if } |Error| \geq 2\,^{\circ}\text{C} \\ 1, & \text{otherwise.} \end{cases} \tag{15}$$

The network then tries to make predictions for a value of θ, as defined in Eq. (13), that maximizes the value of $p(\theta|\boldsymbol{X})$. So, if most predictions for θ are close to 1 the network assigns the greatest probability for the error being of category $\gamma = 0$, and is thus an indication of a highly uncertain forecast. Setting the limit to 2 °C agrees with what SMHI consider an accurate prognosis for temperature (SMHI, 2017). For the Östersund-Frösön validation dataset, this condition was fulfilled for 81 % of the forecasts. For Nordkoster, 91 % of the forecasts fulfilled the same condition.

The code used may be found in the appendix, section A.4.

3.3.2 Absolute temperature

Binary classification can also be used directly for absolute temperature, to predict the likelihood for the observed temperature to exceed/subceed a certain value. The observations were assigned to categories, γ, according to

$$\gamma = \begin{cases} 0, & \text{if } T_{obs} \leq 0\,^{\circ}\text{C} \\ 1, & \text{otherwise.} \end{cases} \tag{16}$$

Similarly to the other classification approach, the network strives towards maximizing $p(\theta|\boldsymbol{X})$, where θ is defined as in Eq. (13).

Code can be found in the appendix, section A.5.

4 Results and discussion

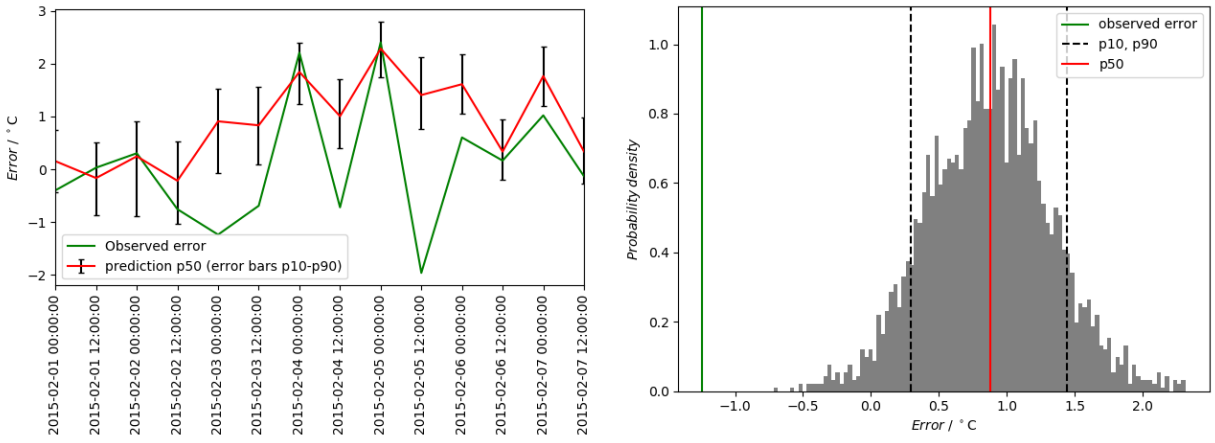
All networks that did converge, were found to do so within 30000 iterations.

4.1 Regression

4.1.1 Fixed standard deviation

The results of the two approaches to fixing the standard deviation are here presented separately. Fixing to the unrealistically small value is referred to as ”small”, fixing to the value from the full error distribution is referred to as ”larger”.

Small standard deviation
The performance, in terms of catching the error, was found to be poor. For the Östersund-Frösön validation dataset only $\sim 30\,\%$ of the observed errors were caught between the 10th and 90th percentiles of the μ predictions. The poor performance is not due to overfitting on the training dataset. The performance by the same measure, but applied to predictions on the training dataset, is only marginally better: $\sim 35\,\%$. Predictions for a short time-series and a sample from that time-series is plotted in Fig. 6, where the observed error is also included.



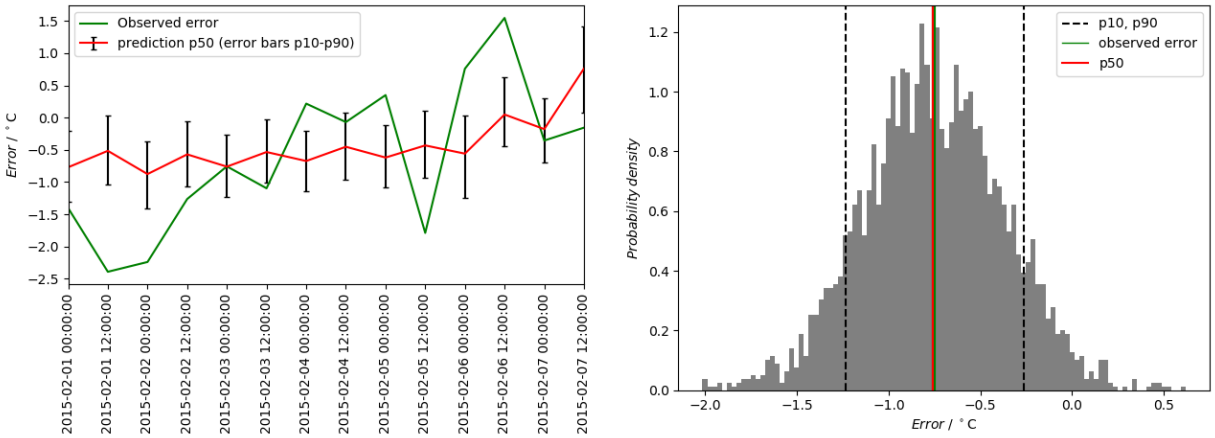
(a) **(b)**

Figure 6: Samples of error predictions for Östersund-Frösön. (a) shows a time series with the observed error compared to the median value for the predictions of μ, along with error bars corresponding to the 10th and 90th percentiles. (b) is a normed histogram of all predictions of μ for Februrary 3rd, 2015 00 UTC.

The observed error shows larger variation than the predictions. The standard deviation of the observed error in the validation dataset is 1.7 °C, which can be compared to the standard deviation for the median prediction of 1.0 °C. As the error is approximately normally distributed, there are much more observations of errors corresponding to the

neighbourhood of its mean and median values than for the larger errors. This might cause the network to favour predictions closer to the median of the error than it should, as there is not so much data pointing towards the greater errors. To check whether this might be the case, the absolute value of the difference between the median of each prediction and the median of the error was compared to the absolute value of the difference between each observed error and the median of the error. Then these values were compared, and the fraction of occasions where the median prediction was closer to the median of the error was computed. It was found to be true for 59 % of the occasions in the validation dataset.

Regarding the predictions for Nordkoster the performance, as evaluated in terms of observed error caught between 10th and 90th percentile, was much better: $\sim$ 48 %. The better performance is likely due to, at least to some extent, that the error is generally smaller for this station. In Fig. 7, a plot of predictions valid for a sample time-series and a sample from the time-series, along with the observed error, are shown.



(a) **(b)**

Figure 7: Samples of error predictions for Nordkoster. (a) shows a time series with the observed error compared to the median value for the predictions of μ, along with error bars corresponding to the 10th and 90th percentiles. (b) is a normed histogram of all predictions of μ for 00 UTC, Februrary 3rd, 2015.

Comparing standard deviations for median prediction and observed error shows a similar relation as for Östersund-Frösön, with a ratio of 1.7. The preference of the network for values closer to the median value than the observed value were stronger than for the Östersund-Frösön network, and was found to be true at 68 % of the cases.

Larger standard deviation
The performance of these network turned out to be worse. For Östersund-Frösön, the drop was not a significant though, $\sim 28\,\%$ of the errors were caught. For Nordkoster on the other hand, the performance dropped to $\sim 36\,\%$, which is clearly lower than for the fixed standard deviation. It does not seem like fixing the standard deviation to a larger value causes greater variation in the predictions of μ. Rather, the opposite occurs and the predictions showed less variation. It might be that if the value for μ is less restricted, more predictions close to zero (for which many more observations exist) are also allowed.

4.1.2 Simultaneous determination of standard deviation and mean value

As no network was found to converge, no analysis of output data was conducted. It is probably a too complicated task for the algorithms available in the program to automatically optimize the fit with two free parameters.

4.2 Binary classification

4.2.1 Error size

The predictions produced by the networks trained towards classification of the error size are not particularly useful for any of the stations. Fig. 8 shows the median predictions of θ for Nordkoster. A similar plot for Östersund-Frösön can be found in the appendix, section B.

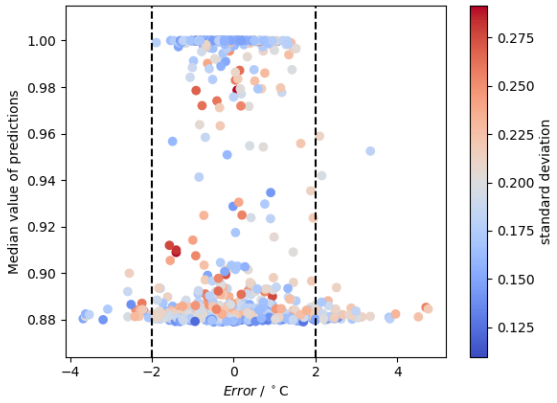
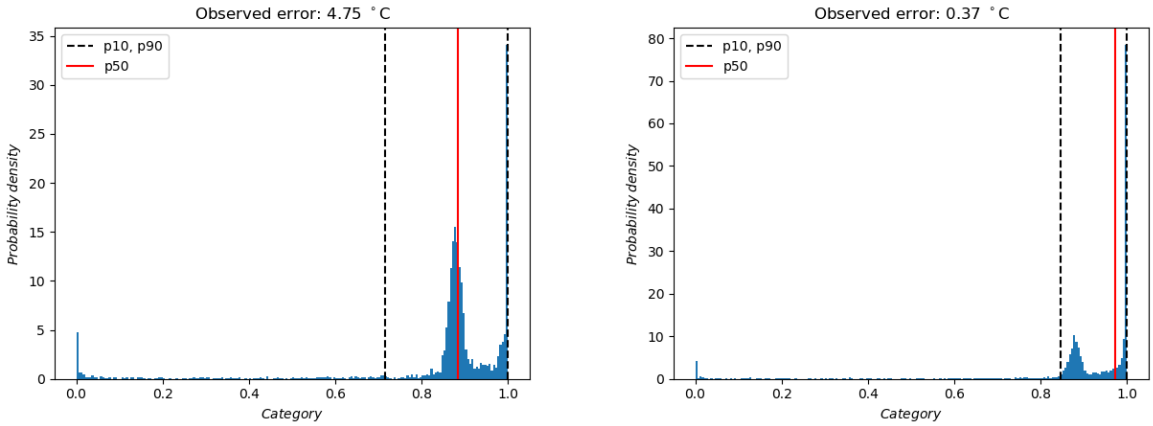


Figure 8: A scatter plot of all predictions for classification of error size, Nordkoster. The colour of the dots represent the standard deviation of the predictions, which corresponds to their uncertainty.

Almost all predictions for θ are confined to a very narrow range of values between 0.88 and 1, instead of spanning the full range between 0 and 1. In the validation dataset for Nordkoster, no predictions where the median was assigned values very close to 1 where found to be wrongly classified (if the median were to be trusted completely on its own, which of course

never was the intention). For the Östersund-Frösön validation dataset however, the median predictions of values close to 1 do occur also for observations corresponding to the category of larger errors. Thus, the predictions belonging to the upper part of the span can not be trusted, despite being correct for all observations in the Nordkoster validation dataset. The median predictions belonging to the lower part of the span are made for observations belonging to both classes, and thus say nothing of the error. Furthermore, the standard deviation, indicated by colour in the figure, does not seem to be a strong indicator of a poor median prediction. If they would have been, the median predictions for wrong category would be coloured red. Fig. 9 shows two samples of full predictions, one for each class of observations. Despite corresponding to very different observations, the predictions are quite similar.



(a) 12 UTC, 5th of July, 2015 **(b)** 00 UTC, 4th of January, 2015

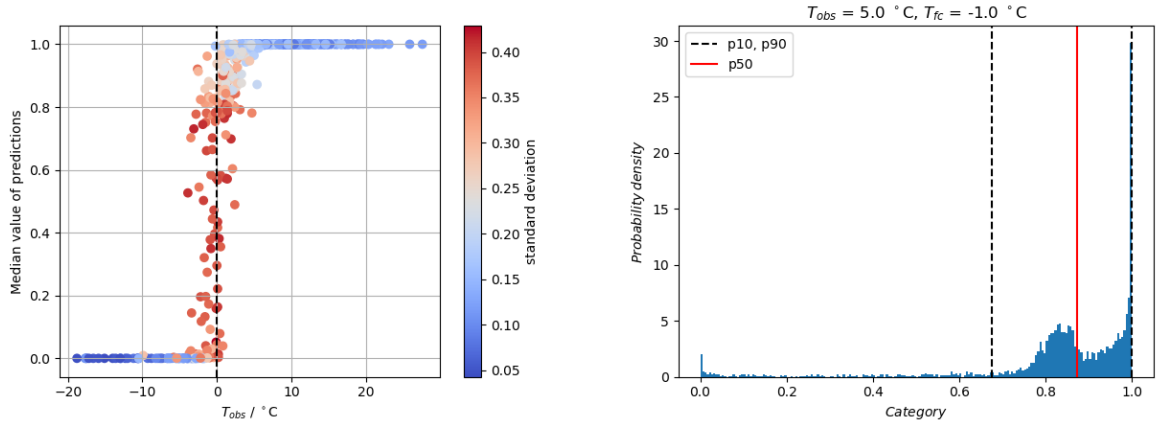
Figure 9: Two sample predictions for Nordkoster. (a) shows an occasion where the error observed is relatively large. (b) shows an occasion for a smaller observed error. Note that both (a) and (b) has a peak in the probability density around 0.9, which corresponds to the proportion of the errors belonging to class $\gamma = 1$, as defined in Eq. (15).

An attempt at improving the narrow span of the predictions were made through adjusting the critical absolute error down to 1 °C. For Östersund-Frösön this classification splits the dataset into two more equally sized categories, with 43 % of the absolute errors being smaller than 1 °C.

Again, the span of the median predictions were narrow, but now even more so. They were centered around 0.43, which corresponds to the amount of observations of errors in the new accurate forecasts class. The standard deviations of the predictions do not seem to give any indication of there being any information in the network of the predictions being poor. It seems to be that the network is underfitted, and have simply identified the proportion of observations belonging to the class, and more or less assigns each observation that probability.

4.2.2 Absolute temperature

The networks for classification of absolute temperature do work more accurately, in some sense. As the correlation between forecast and observations is very strong (see Figures 4a and 5a), anything else should come as a surprise. The predictions for the Östersund-Frösön validation dataset are plotted in Fig. 10, along with full predictions for a sample occasion.



(a) **(b)**

Figure 10: (a) is scatter plot of all predictions for classification absolute temperature, Östersund-Frösön. The colour of the dots represent the standard deviation of the predictions, which corresponds to their uncertainty. (b) shows a plotted sample for 00 UTC, 20th of February, 2015.

Fig. 10a shows that observations of temperatures far from 0 °C are correctly predicted by the median values, along with low standard deviations that express high degrees of certainty. When the observed temperature gets within about 5 °C from zero, the predictions are much more uncertain. The median of the predictions seems to take on almost any value between 0 and 1. But as the corresponding standard deviation also gets larger for these predictions, it seems as the associated uncertainty is caught by the network. That the predictions are so uncertain for observed temperatures around zero, which is the only situation where it could have been of any use, makes it quite useless for uncertainty assessments though. Other attempts were made for classification with other temperatures, but the results were very similar and they do not add any additional information. Training a network towards a specific temperature is difficult for stations in mid-latitudes, such as Östersund-Frösön and Nordkoster, where the temperature shows great seasonal variation. Thus, there may not be as many observations of the temperature trained against to identify the forecasts that are more likely to be erroneous.

5 Conclusions and outlook

The goal of estimating the probability distribution for the temperature error was not achieved using the methods described in this thesis. Treating the error as a continuous parameter would be the most useful approach, if it had been successful. It would not only point out that the forecast is uncertain, but also provide additional information on the probable sign. But the best network I constructed still seems to be underfitted and not able to identify uncertain forecasts to a significant degree. Inability to identify uncertain forecasts applies to the attempts for binary classification of error size too. The issue is propably, again, underfitting. The absolute temperature classification predictions did show some resemblance to observations. However, at such a coarse level no meaningful information is added to the forecast. There is simply no use of a tool that tells a forecaster that there is a high probability of a temperature above 0 °C, when the model forecast for the area is at 15 °C.

Regarding predictions of the error as a continuous parameter, further investigations could be conducted using longer records of data to see if more accurate predictions are reached. An issue with using longer records is that consistent patterns for the error may not exist, due to models continuously being updated. Yet another way of getting more relevant data could be through taking output field from other nearby grid-points too.

For further investigation of the binary classification approach, a measure of how well the networks perform on such tasks needs to be defined. Then, improvements of the predictions can be sought by similar strategies as for the continuous case described above.

In this thesis, no evaluation of how the networks architecture affects the accuracy of the prediction. A systematic approach to the question of structure would be interesting to make, where different depths and/or widths for the layers are tested and their performance are compared to one another. A more systematic approach to the choice of output fields of the models would also be relevant.

References

Ahrens, D., & Henson, R. (2013). *Meteorology today* (11th ed.). Cengage learning.

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., ... Zhang, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, *abs/1605.02688*. Retrieved from `http://arxiv.org/abs/1605.02688`

Bauer, P., Thorpe, A., & Brunet, G. (2015). The quiet revolution of numerical weather prediction. *Nature*, *525*(7567), 47 - 55.

Charney, J. G. (1948). *On the scale of atmospheric motions.* Oslo: Grøndahl & søns boktr., i kommission hos Cammermeyer.

Dreyfus, G. (2005). *Neural networks. methodology and applications.* New York: Springer.

ECMWF. (2018). Part III: Dynamics and numerical procedures. In *IFS Documentation CY45R1.* Retrieved 2019-08-14, from `https://www.ecmwf.int/node/18713`

Holton, J., & Hakim, G. (2013). *An Introduction to Dynamic Meteorology* (Fifth ed.). Amsterdam: Elsevier Science.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251.

Inness, P., & Dorling, S. (2013). *Operational weather forecasting.* Chichester, West Sussex, UK: Wiley-Blackwell.

Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability.* Cambridge University Press.

Kruschke, J. K. (2015). *Doing bayesian data analysis: a tutorial with r, jags, and stan.* Amsterdam : Academic Press, cop. 2015.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M. (2017). Automatic differentiation variational inference. *Journal of Machine Learning Research*, *18*(9-17), 1 - 45.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, *20*(2), 130–141.

Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017). The expressive power of neural networks: A view from the width. In I. Guyon et al. (Eds.), *Advances in neural information processing systems 30* (pp. 6231–6239). Curran Associates, Inc.

Lynch, P. (2008). The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, *227*(Predicting weather, climate and extreme events), 3431 - 3444.

OpenStreetMap contributors. (2017). *Planet dump retrieved 2019-08-05, from* ***`https://planet.osm.org`***.

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*. doi: 10.7287/PEERJ.PREPRINTS.1686V1

SMHI. (2017). *Hur mäts prognosers träffsäkerhet?* [How is accuracy of forecasts measured?]. Retrieved 2019-08-21, from `https://www.smhi.se/kunskapsbanken/meteorologi/hur-mats-prognosers-traffsakerhet-1.17383`

Vasilev, I., Slater, D., Spacagna, G., Roelants, P., & Zocca, V. (2019). *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow* (Second ed.). Birmingham: Packt Publishing.

Acknowledgements

First and foremost I want to thank my supervisors, Fredrik Karlsson at SMHI and Elna Heimdal-Nilsson at Lund University for supervising this thesis.

I want to thank Tove Fast as well, for all the good stötande and blötande of difficult concepts throughout our studies.

I am also grateful to all my friends and family for emotional support and encouragement during the course of this project.

A Code excerpts

A.1 Code common to all programs

Importing libraries and data, splitting training and validation datasets and deciding input parameters:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pymc3 as pm
import theano
import theano.tensor as T
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle as shfl


osd = pd.read_csv('ostersund_froson.csv',sep=';',
                      parse_dates=['validDateTime'])
osd = osd.dropna()
osd['Error'] = osd['stn t2m'] - osd['2 metre temperature']
osd_valid = osd[osd.validDateTime.dt.year == 2015]
osd = osd[osd.validDateTime.dt.year < 2015]


nkr = pd.read_csv('nordkoster.csv',sep=';',
                      parse_dates=['validDateTime'])
nkr = nkr.dropna()
nkr['Error'] = nkr['stn t2m'] - nkr['2 metre temperature']
nkr_valid = nkr[nkr.validDateTime.dt.year == 2015]
nkr = nkr[nkr.validDateTime.dt.year < 2015]


parameters = (['2 metre temperature',
               'Solar elevation',
               'Surface pressure',
               'Temperature 850 hPa',
               'Total cloud cover',
               '10 metre U wind component',
               '10 metre V wind component'])
```

For a succesful network, the weigths may be saved for future predictions, without having to go through the process of training the network again. This code snippet saves and reloads weights traced from the network.

```
qW_0S = trace['W_0']
qb_0S = trace['b_0']

qW_1S = trace['W_1']
qb_1S =trace['b_1']

qW_2S =  trace['W_2']
qb_2S = trace['b_2']

np.savez('weights.npz',
         qW_0S = qW_0S, qb_0S = qb_0S,
         qW_1S = qW_1S, qb_1S = qb_1S,
         qW_2S = qW_2S, qb_2S = qb_2S)

weights = np.load('weights.npz')

qW_0S = weights['qW_0S']
qb_0S = weights['qb_0S']

qW_1S = weights['qW_1S']
qb_1S = weights['qb_1S']

qW_2S = weights['qW_2S']
qb_2S = weights['qb_2S']
```

A.2 Code for regression with fixed standard deviation

This code snippet makes arrays for input and output data, scales the input, makes the arrays accessible for Theano, builds the neural network, and then samples weights and biases from it. It then defines a numpy function that contains the weights and biases from the network to be able to make predictions. Predictions are made on scaled parameter values from the validation dataset and the function for performance evaluation is defined.

```
    def arrays_regression(df,parameters):
    """
    make training data for regression, return tuple containing two numpy
    arrays where index [0] of return tuple is the input to the neural net,
    index [1] is the observations to which the network is fitted

    df is a the dataframe containing the observations and forecasts for the
    station

    parameters is a list of parameters to be included in the input training
    data
    """

    df = shfl(df)

    y = np.array(df['Error'])
    y = np.transpose(y.reshape(1,-1))

    df = df[parameters]
    x = np.array(df)

    return x, y


x_train, y_train = arrays_regression(station,parameters)


ss=StandardScaler()
ss.fit(x_train)
x_train=ss.transform(x_train)


ann_input = theano.shared(x_train)
ann_output = theano.shared(y_train)


```

```
Ni = np.shape(x_train)[1]
NL0 = 100
NL1 = 50
No = 1

with pm.Model() as nn:
    W_0 = pm.Normal('W_0', mu = 0.0, sigma = 1.0, shape = (Ni, NL0),
                        testval = np.random.randn(Ni,NL0))

    W_1 = pm.Normal('W_1', mu = 0.0, sigma = 1.0,shape = (NL0, NL1),
                        testval= np.random.randn(NL0, NL1))


   W_2 = pm.Normal('W_2', mu = 0.0, sigma = 10.0, shape = (NL1, No),
                        testval = np.random.randn(NL1, No))

    b_0 = pm.Normal('b_0', mu = 0.0, sigma = 1.0, shape=(1,NL0),
                        testval = np.random.randn(1, NL0))

    b_1 = pm.Normal('b_1', mu = 0.0, sigma = 1.0, shape = (1,NL1),
                        testval = np.random.randn(1, NL1))

    b_2 = pm.Normal('b_2', mu = 0.0, sigma = 10.0, shape=(1,No),
                        testval = np.random.randn(1, No))

    mu = pm.math.tanh(pm.math.dot(ann_input, W_0) + b_0)
    mu = pm.math.tanh(pm.math.dot(mu, W_1) + b_1)
    mu = pm.math.dot(mu, W_2) + b_2

    Y = pm.Normal('Y',
                  mu = mu,
                  sigma = preset_value,
                  observed = ann_output,
                  total_size = y_train.shape[0])


with nn:
    inference = pm.ADVI()
    approx = pm.fit(n = 30000, method = inference)


Nsamp = 3000
trace = approx.sample(draws = Nsamp)
```

```
80
81  qW_0S = trace['W_0']
82  qb_0S = trace['b_0']
83
84  qW_1S = trace['W_1']
85  qb_1S = trace['b_1']
86
87  qW_2S = trace['W_2']
88  qb_2S = trace['b_2']
89
90
91  def np_nn(x, W_0, W_1, W_2, b_0, b_1, b_2):
92      """
93      make distribution of predictions for mu given parameter values x,
94      using weights W_0,1,2 and bias terms b_0,1,2 generated from network
95      """
96      mu = np.tanh(np.dot(x, W_0) + b_0)
97      mu = np.tanh(np.dot(mu, W_1) + b_1)
98      mu = np.dot(mu, W_2) + b_2
99      return mu
100
101
102 prediction_mu = np_nn(
103     x_valid, qW_0S[0], qW_1S[0], qW_2S[0], qb_0S[0], qb_1S[0], qb_2S[0])
104
105
106 for i in range(1,Nsamp):
107     prediction_mu = np.hstack((
108         prediction_mu,
109         np_nn(x_valid, qW_0S[i], qW_1S[i],  qW_2S[i],
110         qb_0S[i], qb_1S[i], qb_2S[i])))
111
112
113 def performance_score():
114      p90 = np.percentile(prediction_mu, 90, axis=1)
115      p10 = np.percentile(prediction_mu, 10, axis=1)
116      a = np.zeros((len(p90), 1))
117
118      for i in range(len(p90)):
119          if p10[i] < y_valid[i][0] < p90[i]:
120              a[i][0] = 1
121
122      return np.mean(a)
```

Row 68 differs between the tiny standard deviation approach and the large standard deviation approach, where `preset_value` is either replaced with `0.01` for the small standard deviation approach, or with the value of the standard deviation for the error of the station currently being fitted to.

A.3 Code for determining both bias and standard deviation

The following code is an example of a network structure used for determining both bias (`mu`) and standard deviation (`sigma`). As none of the networks converged, no code for post-processing the predictions are supplied. The pre-processing of the data is similar to the fixed-standard deviation regression attempts.

```
with pm.Model() as nn:
    W_0 = pm.Normal('W_0', mu = 0.0, sigma = 1.0, shape=(Ni, NL0),
                        testval = np.random.randn(Ni, NL0))

    Ws_0 = pm.Normal('Ws_0', mu = 0.0, sigma = 1.0, shape=(Ni, NL0),
                        testval = np.random.randn(Ni, NL0))

    W_1 = pm.Normal('W_1', mu = 0.0, sigma = 1.0, shape=(NL0, NL1),
                        testval = np.random.randn(NL0,  NL1))

    Ws_1 = pm.Normal('Ws_1', mu = 0.0, sigma = 1.0, shape=(NL0, NL1),
                        testval = np.random.randn(NL0,  NL1))

    W_2 = pm.Normal('W_2', mu=0.0, sigma = 10.0, shape=(NL1,  No),
                        testval = np.random.randn(NL1, No))

    Ws_2 = pm.Normal('Ws_2', mu = 0.0, sigma = 10.0, shape=(NL1, No),
                        testval= np.random.randn(NL1, No))


    b_0 = pm.Normal('b_0', mu = 0.0, sigma = 1.0, shape=(1, NL0),
                        testval = np.random.randn(1, NL0))

    bs_0 = pm.Normal('bs_0', mu = 0.0,  sigma = 1.0, shape=(1, NL0),
                        testval = np.random.randn(1, NL0))

    b_1 = pm.Normal('b_1', mu = 0.0, sigma = 1.0, shape=(1, NL1),
                        testval = np.random.randn(1, NL1))

    bs_1 = pm.Normal('bs_1', mu = 0.0, sigma = 1.0, shape=(1, NL1),
                        testval = np.random.randn(1, NL1))
```

```
32
33      b_2 = pm.Normal('b_2', mu = 0.0, sigma = 10.0, shape=(1, No),
34                          testval = np.random.randn(1, No))
35
36      bs_2 = pm.Normal('bs_2', mu = 0.0,  sigma = 10.0, shape=(1, No),
37                          testval = np.random.randn(1, No))
38
39      mu = pm.math.tanh(pm.math.dot(ann_input,W_0)+b_0)
40      mu = pm.math.tanh(pm.math.dot(mu,W_1)+b_1)
41      mu = pm.math.dot(mu,W_2)+b_2
42
43      sigma = pm.math.tanh(pm.math.dot(ann_input,Ws_0)+bs_0)
44      sigma = pm.math.tanh(pm.math.dot(sigma,Ws_1)+bs_1)
45      sigma = pm.math.dot(sigma,Ws_2)+bs_2
46      sigma = T.nnet.softplus(sigma)
47
48      Y = pm.Normal('Y',
49                    mu = mu,
50                    sigma = sigma,
51                    observed = ann_output,
52                    total_size=y_train.shape[0])
```

A.4 Code for binary classification of error size

This code snippet accomplishes almost the same things as the previous one, but differs at a few important points:

- The output array consists of boolean elements
- Other activation functions are used in the network
- The target distribution in the output of the neural network is a Bernouilli distribution
- There is no function corresponding to the `performance_score`-function for evaluating the performance of the predictions.

```
1  def arrays_class(df,parameters,critical_error):
2      """
3      make training data for classification of error size, return tuple
4      containing two numpy arrays where index [0] of return tuple is the input
5      to the neural net, index [1] is the observations to which the network
6      is fitted
7
8      df is a dataframe containing the observations and forecasts for the
```

```
    station

    parameters is a list of parameters to be included in the input training
    data

    critical_error is the upper limit for the absolute error, in units of K, for whic
    """

    df = shfl(df)

    y_train = np.array(np.abs(df['Error']) < crit_error)
    y_train = np.transpose(y_train.reshape(1,-1))

    df = df[parameters]
    x_train = np.array(df)

    return x_train,y_train


parameters= (
        ['2 metre temperature',
         'Solar elevation',
         'Surface pressure',
         'Temperature 850 hPa',
         'Total cloud cover',
         '10 metre U wind component',
         '10 metre V wind component'
         ])


critical_error = 2.

x_train_class, y_train_class = arrays_class(
                               station,parameters,critical_error)


ss=StandardScaler()
ss.fit(x_train_class)
x_train_class=ss.transform(x_train_class)


Ni = np.shape(x_train_class)[1]
NL0 = 100
```

```
NL1 = 50
No = 1


nn_class_input = theano.shared(x_train_class)
nn_class_output = theano.shared(y_train_class)
with pm.Model() as nn_class:
    W_0 = pm.Normal('W_0', mu = 0.0, sigma = 1.0,shape=(Ni, NL0),
                        testval = np.random.randn(Ni, NL0))

    W_1 = pm.Normal('W_1', mu = 0.0, sigma = 1.0,shape=(NL0, NL1),
                        testval = np.random.randn(NL0, NL1))

    W_2 = pm.Normal('W_2', mu = 0.0, sigma = 10.0, shape=(NL1, No),
                        testval = np.random.randn(NL1, No))


    b_0 = pm.Normal('b_0', mu = 0.0, sigma = 1.0, shape=(1, NL0),
                        testval = np.random.randn(1, NL0))


    b_1 = pm.Normal('b_1', mu = 0.0, sigma = 1.0, shape=(1, NL1),
                        testval = np.random.randn(1, NL1))


    b_2 = pm.Normal('b_2', mu = 0.0, sigma = 10.0, shape=(1, No),
                        testval = np.random.randn(1, No))

    act_1 = T.nnet.relu(pm.math.dot(nn_class_input, W_0) + b_0)
    act_2 = T.nnet.relu(pm.math.dot(act_1, W_1) + b_1)
    act_out = pm.math.sigmoid(pm.math.dot(act_2, W_2) + b_2)

    Y = pm.Bernoulli('Y',
                        p = act_out,
                        observed = nn_class_output,
                        total_size = y_train_class.shape[0])
```

The inference, tracing and sampling is similar to the regression code. The mirror of the network is adjusted to the proper activation functions, which can be defined using numpy.

```
def sigmoid(x):
    """
    sigmoid activation function
    """
    return 1/(1+np.exp(-1*x))




def relu(x):
    """
    rectified linear unit activation function
    """
    return np.maximum(x,0)

def np_nn_errorsize_class(x, W_0, W_1, W_2, b_0, b_1, b_2):
    act_1 = relu(np.dot(x, W_0) + b_0)
    act_2 = relu(np.dot(act_1, W_1) + b_1)
    act_out = sigmoid(np.dot(act_2, W_2) + b_2)
    return act_out
```

A.5 Code for binary classification of absolute temperature

The only difference between the absolute temperature classification and error size classification code is in the creation of the boolean output array. Here, the truth value depends on whether the observed temperature exceeds a critical temperature.

```
def arrays_class_temp(df,parameters,critical_temperature):
    """
    make training data for classification of absolute temperature, return
    tuple containing two numpy arrays where index [0] of return tuple
    is the input to the neural net, index [1] is the observations
    to which the network is fitted

    df is a dataframe containing the observations and forecasts for the station

    parameters is a list of parameters to be included in the input training
    data

    critical_temperature is the lower limit for the absolute temperature, in
    units of K, for which the the outcome of the forecast is labeled True
    """

```

```
17      df = shfl(df)
18
19      y_train = np.array(np.abs(df['stn t2m']) > critical_temperature)
20      y_train = np.transpose(y_train.reshape(1,-1))
21
22      df = df[parameters]
23      x_train = np.array(df)
24
25      return x_train, y_train
```

B Additional figures

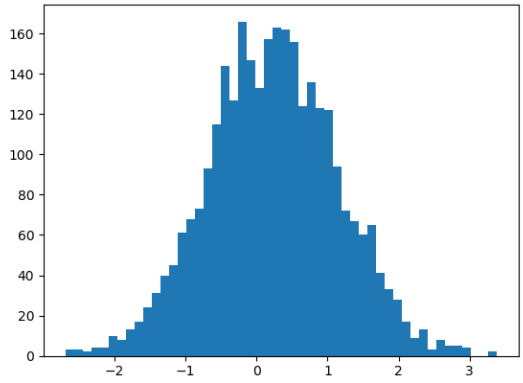


Figure B.1: An example distribution of samples from one of the weights in a network after training.

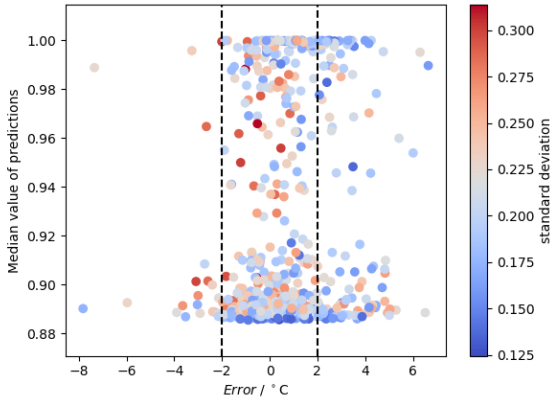
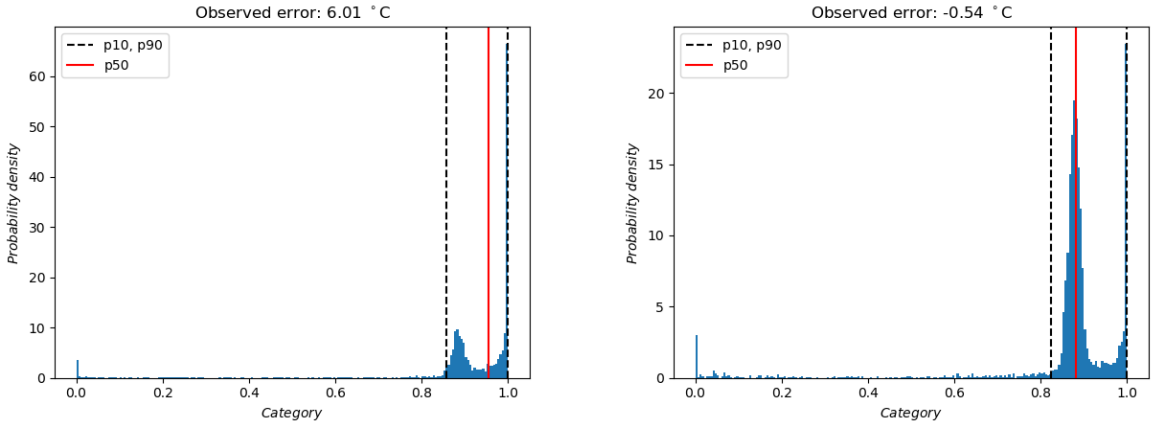


Figure B.2: A scatter plot of all predictions for classification of error size, Östersund-Frösön. The colour of the dots represent the standard deviation of the predictions, which corresponds to their uncertainty.



(a) 00 UTC, 22nd of February

(b) 12 UTC, 30th of May.

Figure B.3: Two samples of single time-step predictions for binary classification of error size, Östersund-Frösön. (a) shows predictions for an occasion where the observed absolute error was significantly greater than 2 °C. (b) shows predictions for a smaller observed error.

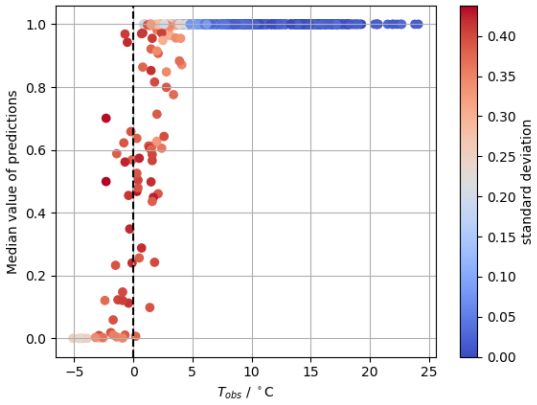
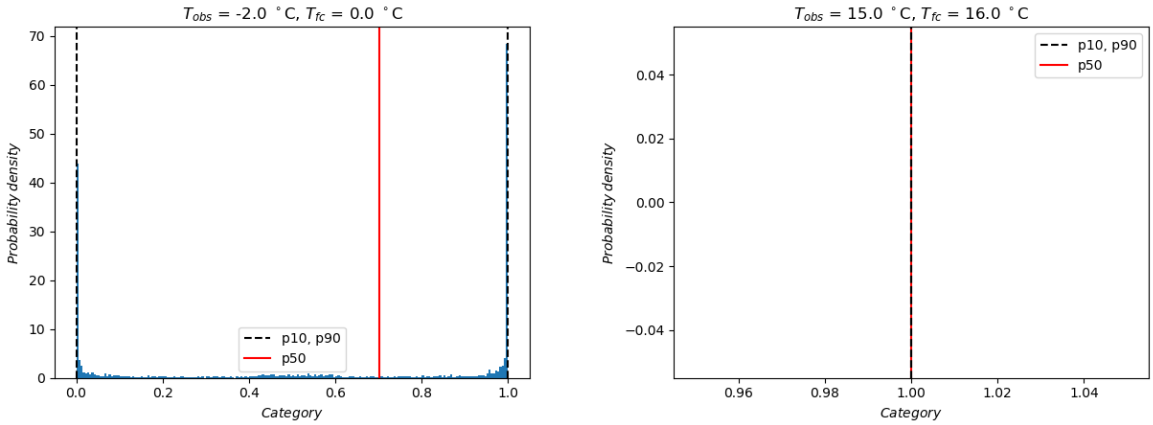


Figure B.4: A scatter plot of all predictions for classification of temperature, Nordkoster. The colour of the dots represent the standard deviation of the predictions, which corresponds to their uncertainty.



(a) 28 December, 00 UTC

(b) 20th July, 00 UTC

Figure B.5: Sample predictions of temperature classification for Nordkoster. (a) shows a highly uncertain prediction for a day where both the forecast and observed temperature where close to 0 °C. (b) shows a prediction of high certainty that the temperature will exceed 0 °C.