Machine Learning in Python for Weather Forecast based on Freely Available Weather Data

E. B. Abrahamsen, O. M. Brastein, B. Lie*

Department of Electrical Engineering, Information Technology and Cybernetics
University of South-Eastern Norway, N-3918 Porsgrunn,
*(Bernt.Lie@usn.no)

Abstract

Forecasting weather conditions is important for, e.g., operation of hydro power plants and for flood management. Mechanistic models are known to be computationally demanding. Hence, it is of interest to develop models that can predict weather conditions faster than traditional meteorological models. The field of machine learning has received much interest from the scientific community. Due to its applicability in a variety of fields, it is of interest to study whether an artificial neural network can be a good candidate for prediction of weather conditions in combination with large data sets. The availability of meteorological data from multiple online sources is an advantage. In order to simplify the retrieval of data, a Python API to read meteorological data has been developed, and ANN models have been developed using TensorFlow.

Keywords: Weather prediction, Auto-regressive neural networks, Meteorological data

1 Introduction

1.1 Background

The forecasting of weather conditions and in particular the prediction of precipitation is important for hydro-power operation and flood management. Mechanistic meteorology prediction models based on 3D CFD/Navier Stokes equations (Thibault and Senocak, 2009) is extremely demanding wrt. computing power. Generating a 14 day weather forecast can easily take 12 hours even on fast computers. Machine Learning (ML), Big Data, and use of Internet of Things (IoT) are receiving increased interest from the industry. It is well known that large amounts of data coupled with novel ML methods can produce results on par with traditional physics based models.

Due to an interest in weather monitoring in the general public, today a large number of weather stations are connected to the internet, and are thus available as cheap, distributed sensors. Additionally, several organizations that are involved in collection of meteorological data offer online data servers with accessible Application Programming Interfaces (API) such as the `HTTP` based `GET/REST` protocols. In order to simplify experimentation with several sources of meteorological data it is of interest to develop a unified API, hence facilitating the extraction of data from different sources. With large quantities of data, both historical and current measurements, it is an attractive solution to use machine learning in order to predict weather conditions based on these relatively simple data sources. Using a large amount of data together with novel machine learning algorithms can then compensate for lack of complex meteorological models and yield usable forecasts with less computing time.

Simple ML models would base predictions on auto regressive (AR) structures, where, say the current temperature in a location is correlated with several past temperatures in the same location. In a slightly more advanced auto regressive structure, a set of properties, e.g., the tuple (temperature, humidity, and precipitation) could be correlated with several past values of the same tuple. An even more advanced structure is the auto regressive structure with exogenous input (ARX). In such a model, the current (local) set of properties is correlated with both past values of the same (local) set, but also with other values from the same location or values of the same properties from other locations at *current time*. Finally, in ARMAX structures, exogenous inputs at *different times* (= moving average) are used in the correlation.

1.2 Previous Work

(Hayati and Mohebi, 2007) studied multi layer perceptron (MLP) neural networks trained and tested on ten years of meteorological data (1996-2006). The network structure consisted of three layers with a logistic sigmoid activation function in hidden layers and linear functions in the output layer. Seven weather variables were used in the study: dry temperature, wet temperature, wind speed, humidity, pressure, sunshine, and radiation. The inputs were normalized and used to predict dry air temperature in intervals of 3 hours for a total of 8 predictions pr day. The error was calculated using mean absolute error (MAE). In (Smith et al., 2006), the authors focused on developing artificial neural network (ANN) models to forecast air temperature at hourly intervals from one to 12 hours ahead. Thirty models were calibrated for each interval, in order to study the effect of randomized initial weights on test set prediction accuracy. The network structure consisted of three fully connected hidden layers that used Gaussian, Gaussian complement, and hyperbolic tangent activation functions. The input data was linearly transformed to the range $[0.1, 0.9]$ and consisted of five weather variables: temperature, relative humidity, wind speed, solar radiation and rainfall. Later, seasonal variables were introduced as inputs which improved model accuracy. A recent machine learning (ML) approach, based on a hybrid model including both ANNs, decision trees (DT), and Gaussian process modeling (GP) is presented in (Grover et al., 2015). They concluded that while previous attempts at weather modeling using ML have had limited success, their hybrid model approach surpasses the NOAA[1] benchmarks. A review on the use of machine learning methods for weather prediction is presented in (Chauhan and Thakur, 2014).

Meteorological data from a number of sources are available today, e.g., from the Norwegian Meteorological Institute data service `frost.met.no`, and from *Netatmo*[2]. These and others are potential "Big Data" sources.

A number of high quality ML tools have become available the last decade, also as packages in free computer languages such as Python. One possible ML tool which runs in Python is Google's TensorFlow. AR, ARX, and ARMAX models for linear systems are routinely used in system identification, e.g. (Ljung, 2002, 1999; Johansson, 1993).

1.3 Overview of Paper

Weather prediction is a convenient case for studying machine learning. By developing APIs for accessing available data from meteorological institutes and other weather stations, this gives access to an abundance of data. Weather data is something that most people can relate to in their daily life, but is also important for energy systems, flood prediction, etc. Good physical based meteorological models are available, which makes it easy to compare the quality of machine learning models. In this paper, we have focused on a new Python API for collecting weather data, and given simple, introductory examples of how such data can be used in machine learning. Weather data from `frost.met.no` have been collected using a newly developed Python API. These data have been used to train and tune several auto-regressive artificial neural networks (AR-ANN) by using TensorFlow from Python. The resulting models have been used to predict the temperature in Porsgrunn with prediction horizons of 1, 3, 6, and 12 hours. The example ANN is then extended with precipitation data and compared to the initial AR-ANN..

This paper is organized as follows; Section 1 provides the necessary background information, and a short review of previous work relevant to the project. Section 2 gives some theoretical details regarding ANNs and the developed APIs for collection of weather data. Section 3 discusses the obtained results, before the work is concluded in Section 4 together with suggestions for future work.

[1]National Oceanic and Atmospheric Administration, `http://www.noaa.gov`

[2]`https://www.netatmo.com`

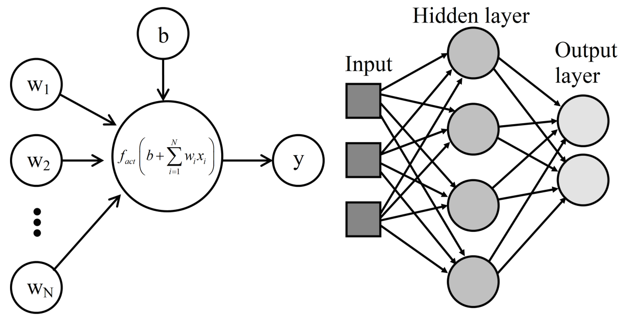

Figure 1. Illustration of a single neuron (left) and an example of an artificial neural network (right).

2 Materials and Methods

2.1 Artificial Neural Networks

Artificial neural networks (ANN) have existed in various forms since the 1940s (McCulloch and Pitts, 1943; Goodfellow et al., 2016), but have received renewed interest in recent years (Goodfellow et al., 2016). An ANN is a collection of neurons, which are small computational units that superficially mimic the way neurons work in nature. A single neuron is simply a weighted sum of a set of inputs, plus a bias, with an applied activation function, Fig. 1 (left).

A non-linear activation function $f_{\text{act}}(\cdot)$ is important for success in applying ANNs, otherwise the resulting model output is simply a linear combination of the inputs. The equation for a single neuron can be written as:

$$y_k = f_{\text{act}}(b + x_i w_i) \tag{1}$$

The power of ANNs comes from connecting many neurons together in a network. The simplest network structure is a feed forward network, as shown in Figure 1 (right). Neurons are connected in simple layered structures where the inputs of each neuron are connected to all the outputs of the previous layer. If we describe the inputs x_i and weights w_i in matrix form, we can write a whole layer of neurons as:

$$y = f_{\text{act}}(Wx) \tag{2}$$

where the bias is included as $w_0 = b$ by adding an artificial constant input $x_0 = 1$,.

A feed forward ANN is built by connecting multiple layers together. The inputs to the network are connected to the inputs of the first hidden layer. The first hidden layer can then be connected to more hidden layers. The last hidden layer connects to the output layer. The output of the ANN is given by this output layer. We can then write a single non-linear matrix equation for the whole network. An example equation for an ANN with three hidden layers is:

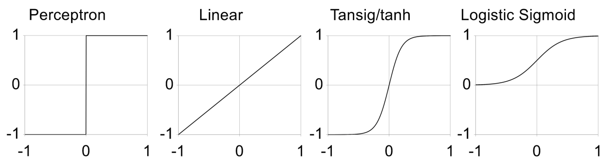

Figure 2. Typical activation functions used in ANNs.

$$y^{\text{out}} = f_{\text{act}}^{\text{out}}\left(W^{\text{out}} f_{\text{act}}^{(2)}\left(W^{(2)} f_{\text{act}}^{(1)}\left(W^{(1)} x\right)\right)\right) \tag{3}$$

Equation (3) shows that an ANN is simply a non-linear matrix equation with a large number of coefficients. Each $W^{(j)}$ matrix can be large, thus allowing the ANN model to fit complex non-linear systems. The descriptive power that comes from this complexity is the reason why ANN models is able to adapt to such a large variety of systems.

Many choices for activation function are possible, as shown in Figure 2. Initially in the history of ANNs, a simple sign operator $y = \text{sgn}(x)$ was often used as the activation function. These simple neurons were called *perceptrons*. The perceptron term survives to this day in the ANN community; indeed, a deep learning network is often referred to as a Multi Layer Perceptron (MLP). In this work, both tanh and logistic sigmoid $\frac{1}{1+\exp(-x)}$ activation functions (Goodfellow et al., 2016) are used. Training of an ANN is essentially a parameter optimization of the network weights such that the output of the network minimizes a chosen loss function. However, since an ANN model has a high number of parameters to optimize, there are a large number of local solutions. The optimization of the training weights is performed iteratively, where each training iteration is called an *epoch* (Goodfellow et al., 2016).

Globally optimal solutions for ANN training are in general considered very hard to find (Goodfellow et al., 2016; Bishop, 2013). Instead, the focus is on finding a solution that is "good enough". During training, there is always a risk that a particular training session can get stuck in a local minimum which is far from optimal. If the same ANN hyper-parameter configuration is trained multiple times, it is usually clear if one or more iterations are indeed giving sub-optimal performance due to this local minima problem. It is also important to note that there has been much research in improving training performance in the presence of local minima. Hence, there exists a large number of training algorithms which seek to improve ANN training performance. For more details on the development of ANNs see, e.g., (Goodfellow et al., 2016).

A model with high descriptive power is prone to over-fitting. The term over-fitting is used in empirical modeling to describe what happens when a model adapts to random variations in the training set which does not generalize well to new data. This effect is apparent in all forms of empirical modeling, from simple curve fitting to complex ANNs. Since the ANNs have such a large number of coefficients, the over-fitting problem is particularly important. The simplest way to reduce the risk of over-fitting is to increase the amount of training data, either by collecting more data or artificially creating more training data through some form of transformation on the original data (Ciresan et al., 2010). Another way to reduce the risk of over-fitting is to apply a regularization method (Goodfellow et al., 2016; Kuhn and Johnson, 2013). The subject of regularization is a research field in itself, which involves methods that prevents the training algorithms for ANNs from adapting to random variations in the training data. The simplest form of regularization is to have a large amount of data. Since this work is based on retrieving weather data from online databases, the cost of obtaining data is relatively low, hence a large amount of training data is readily available. One common regularization method is the use of weight decay (Goodfellow et al., 2016). Weight decay adds a penalty to the loss function which is proportional to the training weights w_i themselves. This forces the weights to stay "small" (Goodfellow et al., 2016):

$$J_{\min} = \alpha \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 + (1-\alpha) \sum_{i=1}^{M} w_i^T w_i \tag{4}$$

In Eq. (4) the loss measure is the mean square error (mse) between the predictions $\hat{y}_i$ and the references/observations y_i. The L_2 (2-norm) weight decay penalty is the last term in the $J_{\min}$ loss function. A hyper parameter α is used to decide how strong the weight decay regularization should be.

2.2 Timeseries modeling

A common method for modeling discrete timeseries data is the use of auto-regressive models with exogenous inputs (ARX). Using the time-shift operator q^{-k} to indicate a quantity being shifted k time-steps back in time, these models can be expressed on the form

$$y_k\left(1 + a_1 q^{-1} + \cdots + a_n q^{-n}\right) = u_k\left(b_1 q^{-1} + \cdots + b_m q^{-m}\right) \tag{5}$$

That is, the output at time k is a function of both the inputs and the output at previous times. If all b_i coefficients are zero, e.g., there are no exogenous inputs, the model is called an AR model (Ljung, 1999). A nonlinear ARX model can be formulated as

$$y_k = f\left(y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots, u_{k-m}\right) \tag{6}$$

Traditional ARX models are linear models as illustrated in Eq. (5) , thus $f(\cdot)$ in Eq. (6) forms a linear combination of past inputs and outputs. When ANNs are applied to timeseries modeling, function $f(\cdot)$ in Eq. (6) is replaced by the ANN, such that:

$$y_k = \text{ANN}\left(y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots, u_{k-m}\right) \tag{7}$$

Hence, the term auto-regressive neural network (AR-ANN) is used to denote an ANN which predicts a time-series variable based on previous measurements of the same variable, e.g. the inputs and outputs to the network are the same variables but at different times. Similarly, an ARX-NN is a network which in addition to previous measurements of the output variable also has additional measurements as its inputs.

2.3 Python API for Data Collection

A Python API wrapper is an easy way to obtain free weather data from APIs and open data. A wrapper was designed to support multiple weather data suppliers, so it is possible to add more suppliers in the future. The API does not support the use of multiple suppliers at the same time. Currently the Norwegian Meteorological Institute data service `frost.met.no` and *Netatmo* are supported. The API will request hourly data for a given date, either at the station nearest to the specified latitude and longitude coordinate or within a specified rectangle as specified in kilometers centered on a given latitude and longitude coordinate. The wrapper uses `HTTP GET` requests to obtain the data from the data suppliers and returns a list where each element is a 3 item list with `stationID`, timestamp, and measured value. The returned data can then be saved to a file or database.

2.4 Experimental Data

The data used in this work was collected from `frost.met.no` using the mentioned Python API. The data consists of hourly temperature and precipitation measurements in the period `01.01.2016 T00:00` to `31.12.2017 T23:00` from weather station `SN30255` at latitude: `59.091` and longitude: `9.66` in Porsgrunn, Norway. Due to downtime on the station the first month of 2017, data starts at 01.02 in the 2017 part of the data set, hence for consistency the first month of 2016 was also removed. For the first experiment only temperature data was used. In the second experiment, temperature data and precipitation data was used. For all experiments, the data was split into three independent sets: 60% used for training, 20% for hyper parameter tuning (validation), and 20% for testing the prediction accuracy of the models.

3 Results and Discussion

The goal of this work is to predict the temperature using an artificial neural network (ANN). Four cases have been studied in both experiments, using prediction horizons of 1, 3, 6 and 12 hours. In each experiment, four separate models were created, one for each case. The input data was normalized, and the output was denormalized to get predictions in degrees Celsius. To test the different models, 48 consecutive hours is set to be predicted by the different models.

All ANNs use rectified linear unit (ReLU) as the activation function in the hidden layers and a linear activation function in the output layer.

3.1 Experiment 1 - AR Neural Network

The first experiment used only temperature data. This constitutes an auto-regressive neural network (AR-NN) model. Figure 3 shows the results of predicting the test set using each of the four models, together with the reference measurements.

There is a sudden change in measured temperature in the time interval 36 to 42 in the test set. The models with prediction horizons 1 and 3 hours show an oscillating response to this rapid change. However, they are still closer to measured data than the models with longer prediction horizons. At time 8 to 20, the 12 hour model is significantly poorer than the 1, 3, and 6 hour models. A plausible reason for this might be that the models respond to data given and the 12 hour model uses data that is 12 hours prior to the measurement. So, at time 7 the 12 hour model started increasing, and at time 15 it flattens out, probably because the algorithm used the data at time 3 and measured that the data started to turn, thus a prediction model should "slow down". All the models show significant deterioration in predictions when the temperature changes rapidly.

Table 1 shows the hyper-parameters with test error summarized. The error is calculated from normalized test data, hence the error is *not* presented in units of Celsius. The number of layers for the 12 hour prediction model is two times that in the other models. Further, the regression horizon (i.e., the amount of past data points used in the predictions) is 169 hours (7 days) for the 12 hour model. Compared to a regression horizon of 48 hours for the 6 hour prediction model, this is a significant difference. The learning rate is also significantly lower for the 12 hour prediction model. The low number of epochs and low learning rate was found necessary for the 6 and 12 hour prediction models to generalize properly and avoid over-fitting. With these hyper-parameters, training is slowed down.

Figure 4 shows the errors in degree Celsius for the predictions on the test set. These results are calculated as the difference between measured value and predicted value. Hence, the error that is negative is overshooting and error that is positive is undershooting. Observe from Figure 4, that due to the rapid change in measured data in time interval 36 to 42, the error is significantly higher for all the models.

Table 2 shows a selection of tuning of the ANN hyper-parameters for 1 hour prediction. At first, a 3 hour regression horizon was used on the assumption that a human observer would likely be able to predict the temperature one hour ahead of time based on a small amount of data. Later, a 24 hour regression horizon was tested against the same structure. The longer regression horizon was found to improve prediction performance on the validation set. Selecting a good neural network structure is important. The choice of network structure depends on the type of application. According to (Heaton, 2017), one hidden layer can approximate any continuous function, while two hidden

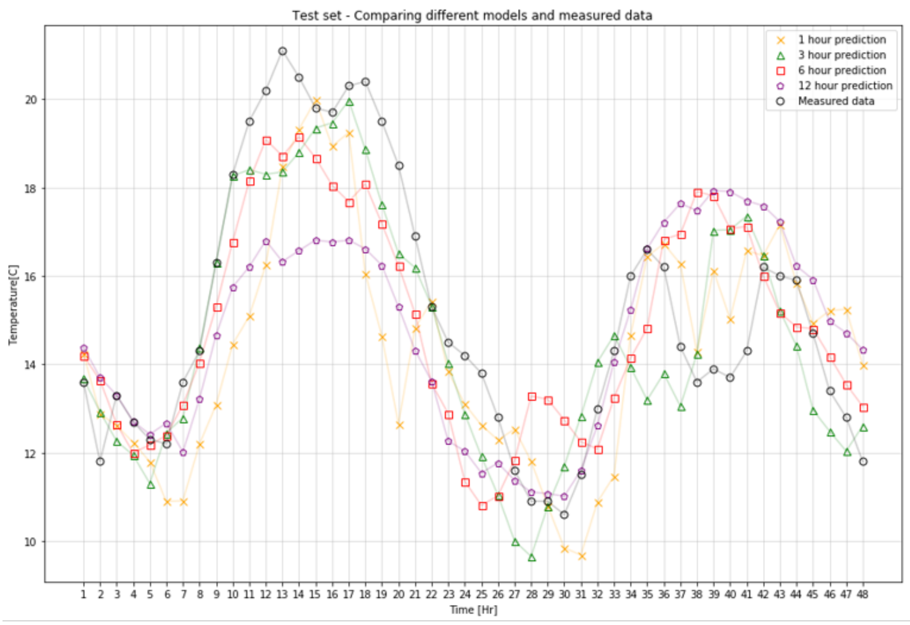

Figure 3. Shows the measurements and outputs of the 4 different models.

Table 1. Hyper-parameters for each model.

Model	*Test error*	*Epochs*	*Layer structure*	*Regression horizon*	*Learning rate*
1 hour	0.0101	810	17, 12	24	0.01
3 hour	0.0318	150	30, 20	48	0.01
6 hour	0.0608	1000	38, 24	48	0.001
12 hour	0.0894	500	112, 75, 50, 34	169	0.0001

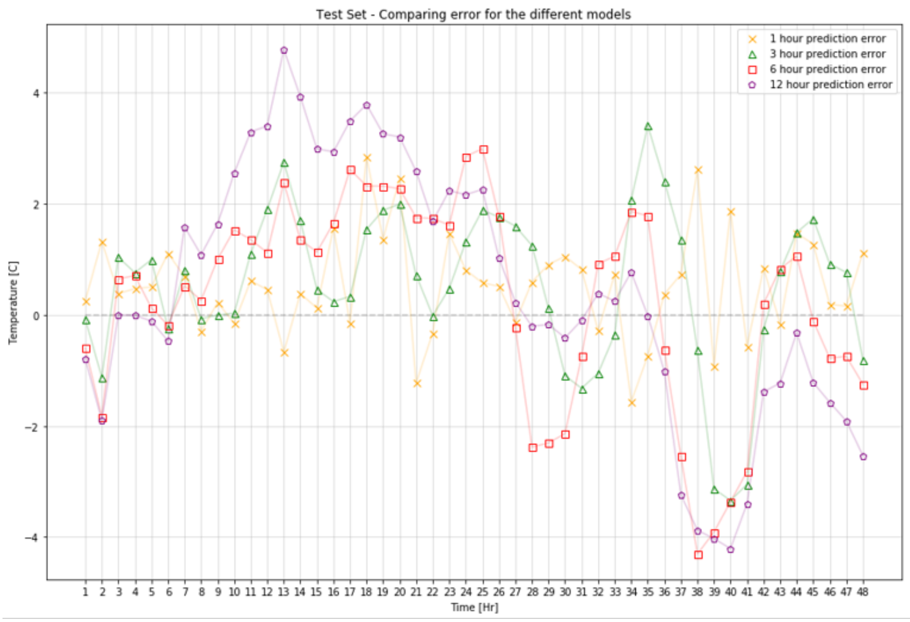

Figure 4. Deviation in degrees Celsius each point for each AR model was off for the 48 hours predicted. The deviation is calculated as the difference between measured value and predicted value, thus a negative error implies overshooting and a positive error implies undershooting.

layers can approximate any arbitrary function. Due to having more descriptive power, the two layer networks also tends to adapt faster to the patterns in the training data, thus learning the input-output relationships faster. Hence, from one to three hidden layers were tested as shown in Table 2. Three layers give approximately the same loss as two layers, hence a two-layer model is chosen. The width of each layer was chosen according to the following three rules suggested by (Heaton, 2017):

1. The number of hidden units in each layer should be between the number of inputs and number of outputs.
2. The number of hidden units in each layer should be $\frac{2}{3}\times$(number of inputs + number of outputs).
3. The total number of hidden neurons in all layers should be less than $2\times$ (number of inputs).

3.2 Experiment 2 - ARX Neural Network

In Experiment 2, the neural network input is extended to include precipitation data. Observe that the error in the time interval 36 to 42 is reduced for the shorter models (1 and 3 hour). The 1 hour model predicts the test set with satisfactory accuracy. However the 3 hour model still has significant prediction errors, in particular at higher temperatures. Observe that, as expected, the prediction accuracy of each model deteriorates with longer prediction horizons. This is particularly apparent in the time interval 8 to 20.

Table 3 shows the hyper-parameters with test set prediction errors. The test errors on the ARX models are higher than on the AR models. This is unexpected and the reason is most likely poor tuning of the models. Two hidden layers were used for all four models, as suggested by (Heaton, 2017). The hidden structures depend largely on the choice of regression horizon, hence the structures for each model is similar, except for the 6 hour model, which achieved better tuning with hidden layer structure (33,12) instead of (17,12).

The error for each model predicted on the 48 hour test set is shown in Figure 5. Observe that the error due to the mentioned rapid change in measured data at time 36 to 42 in the 1 hour prediction model is reduced.

4 Conclusions and Future Work

In this work, artificial neural networks are used to predict temperature. Four separate models were trained to predict the temperature 1, 3, 6, and 12 hours ahead. In the first experiment, only temperature was used as input to the networks. This constitutes an auto-regressive neural network (AR-NN). In the second experiment, precipitation data was introduced into the network, forming an auto-regressive neural network with exogenous input (ARX-NN). After extensive tuning of hyper parameters for all eight models, the prediction results of the models were compared. Introducing precipitation as an input in the ARX model was shown to slightly improve the prediction performance. Hence, it may be interesting to extend the model with other inputs. Mainly, it is of interest to study whether introduction of data from other geographical locations can improve the prediction results. Based on knowledge of how the jet stream moves and influences the weather, together with local pressure variations, it would be natural to add weather information from, e.g., Kristiansand, Oslo, etc. as exogenous inputs. This will be a topic for future research.

References

C.M. Bishop. *Pattern Recognition and Machine Learning: All "just the Facts 101" Material*. Information science and statistics. Springer, 2013. ISBN 9788132209065. URL `https://books.google.no/books?id=HL4HrgEACAAJ`.

Divya Chauhan and Jawahar Thakur. Data mining techniques for weather prediction: A review. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(8):2184–2189, 2014.

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition, 2010. *Cited on*, 80, 2010.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Aditya Grover, Ashish Kapoor, and Eric Horvitz. A deep hybrid model for weather forecasting. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 379–386. ACM, 2015.

Mohsen Hayati and Zahra Mohebi. Application of artificial neural networks for temperature forecasting. *World Academy of Science, Engineering and Technology*, 28(2):275–279, 2007.

Jeff Heaton. The number of hidden layers, 2017. URL `http://www.heatonresearch.com/2017/06/01/hidden-layers.html`.

R. Johansson. *System Modeling and Identification*. Information and system sciences series. Prentice Hall, 1993. ISBN 9780134823089. URL `https://books.google.no/books?id=FZ7gAAAAMAAJ`.

Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

L. Ljung. *System Identification: Theory for the User*. Prentice Hall information and system sciences series. Prentice Hall PTR, 1999. ISBN 9780136566953. URL `https://books.google.no/books?id=nHFoQgAACAAJ`.

Lennart Ljung. Prediction error estimation methods. *Circuits, Systems and Signal Processing*, 21(1):11–21, Jan 2002. ISSN 1531-5878. doi:10.1007/BF01211648. URL `https://doi.org/10.1007/BF01211648`.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Table 2. 1 hour prediction, selected hyper-parameter search.

Exp. No.	*Epochs*	*Regression horizon*	*Layer structure*	*Learning rate*	*Training set loss*	*Validation set loss*
1	1000	3	3	0.001	0.0144	0.0109
2	1000	24	3	0.001	0.0125	0.0105
3	1000	24	17	0.01	0.0116	0.0101
4	1000	24	17,12	0.01	0.0113	0.0100
5	1000	24	17,12,9	0.01	0.0113	0.0101
6	810	24	17,12	0.01	0.0113	0.0100

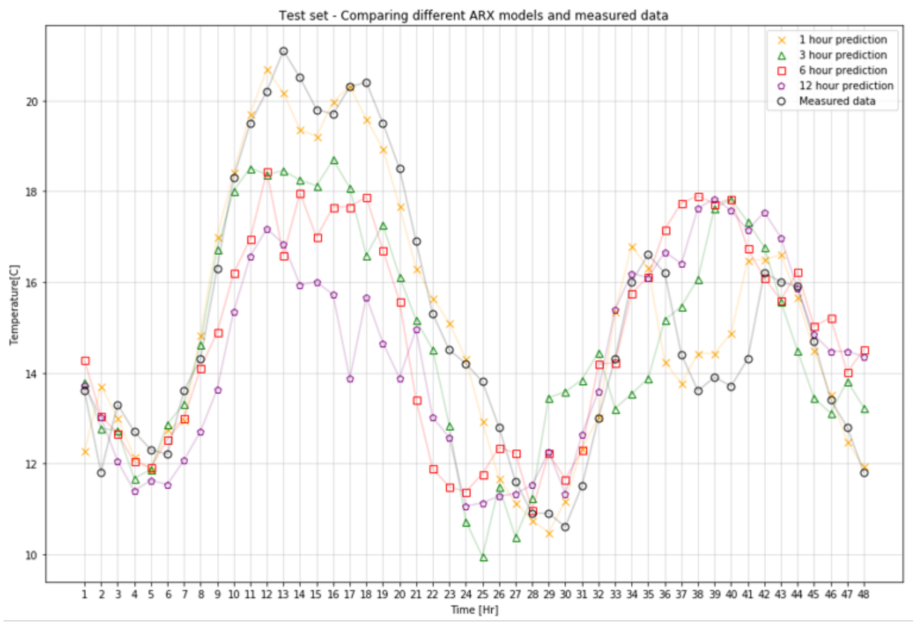

Figure 5. Measured value and the outputs of 4 different ARX models.

Table 3. Hyper-parameters for each ARX model.

Model	*Test error*	*Epochs*	*Layer structure*	*Regression horizon*	*Learning rate*
1 hour	0.0133	500	17, 12	24	0.01
3 hour	0.0653	500	33, 23	48	0.001
6 hour	0.0946	500	17, 12	48	0.01
12 hour	0.0991	500	17, 12	24	0.001

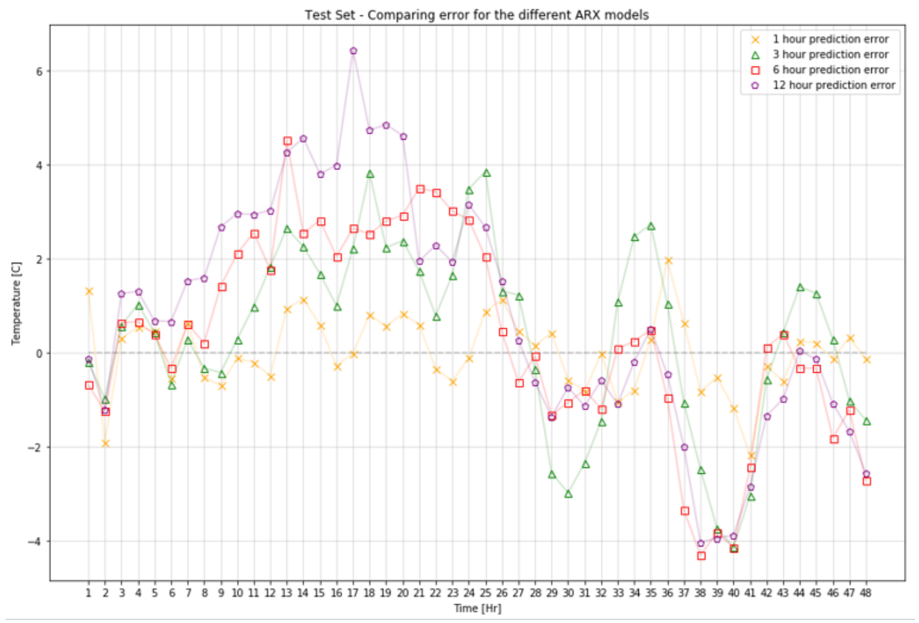

Figure 6. Error in Celsius for the ARX model over the 48 hour test set. The error is calculated as the difference between measured value and predicted value.

Brian A Smith, Ronald W McClendon, and Gerrit Hoogenboom. Improving air temperature prediction with artificial neural networks. *International Journal of Computational Intelligence*, 3(3):179–186, 2006.

Julien Thibault and Inanc Senocak. Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows. In *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, page 758, 2009.