***2020 IEEE** 26th International Symposium for Design and Technology in Electronic Packaging (**SIITME**)*

A Neuro Model for Weather Forecasting

T.G. Predună, V.A. Rusu, and D.I. Năstac
Faculty of Electronics, Telecommunications and Information Technology
POLITEHNICA University of Bucharest, Romania
nastac@ieee.org

***Abstract*—Traditional weather forecasting, using physical methods, requires a lot of computational power and is generally slow. Modern developments in machine learning have led us to experiment with weather prediction using those newly available tools. In this article, we will present the capabilities, advantages and shortcomings of neural networks in the context of weather prediction. The tools consist of open source machine learning libraries, freely available data and two off the shelf computers for all the experiments, in order to show that weather forecasting can be done in a cost-efficient manner.**

Keywords—weather forecasting; neural networks; machine learning

I. Introduction

The problem we are trying to solve is to predict "weather" with the aid of machine learning.

But what is weather? Weather consists of many different phenomena that humans experience, such as wind, temperature, humidity, sunlight and other meteorological effects, linked together by the laws of physics. As an example, it's always cloudy when it's raining, but it's not always raining when it's cloudy. Therefore, weather can be characterized as a complex system of random interdependent variables, which are all time dependent. Everything weather event happens at the same time or after another one.

And what is machine learning? Machine learning is an application of artificial intelligence where a 'machine' learns by itself how to solve a problem. Most of the problems in the field are either classification or regression problems. This project started using three different algorithms, specifically, decision-trees, support vector machines and neural networks, but the first two were dropped due to bad results. Weather forecasting, by nature, is a regression problem, which is very well suited for neural networks. Traditionally, weather forecasting works by collecting as much data as possible about the atmosphere in the present moment and then inputting the data into computer programs that simulate the physical processes which take place in nature, in order to attempt to predict what the weather will be like at a certain location and time. Most resources and time are spent in those simulations, because you have to insert the current state of the atmosphere and then wait each time for the simulation to happen. The 'machine learning' approach is to collect as much 'past' data as possible about the weather at a location, train a neural network on a part of it, test the neural network on the remaining part and modify the network until a good enough accuracy is reached. Most of the resources and time are spent on the training and tuning part, but after this part, every prediction is almost instantaneous, after the current state of the atmosphere is inserted.

Simple and obvious questions can then be asked: Is it possible that such a technique may predict accurately complex weather phenomena and events? What about just some of them? What is the most time efficient way to achieve accurate "enough" predictions in the context of machine learning? and Why should someone use machine learning and not the physical models that are already in use?

So, can complex phenomena be predicted with machine learning? As of now, no. Can at least "some" weather phenomena be predicted? Yes. What's the most efficient way? Multiple neural networks for each weather phenomena, all trained on the same data consisting of different phenomena. Why machine learning? It's very good for making localized predictions, as long as there is enough data for the point on the map for which the predictions are made.

But why should someone switch from physical models? Ideally, both physical models and neural networks would be used. An article that came out this June compares the two [1]. The authors used LSTMs (long short-term memory neural networks) trained on 5 months of data in a similar fashion to us and managed to get better results for up to twelve hours.

A more basic approach is also possible, using linear regression to achieve some degree of success [2] or using support vector machines [3]. Another option is the "Google Approach", which uses radar images and convolutional neural networks [4]. But the main advantages of neural networks and machine learning in general are their speed and ability to be tailored to a specific area. This flexibility makes them a better choice in places where using physical models is not justified, being too resource intensive or expensive. Neural networks can be trained in a similar fashion to how social media sites predict users' behavior, based on what they like and their interactions on the website, only this time the user is a specific area, the area where we want to forecast the weather. Traditional Numerical Weather Forecasting models require a large amount of computing power, which is expensive, whereas a neural network only needs to be trained once (or periodically), after which the predictions are instantaneous. They can also be linked to real time sensors to predict real time phenomena, as described in [5].

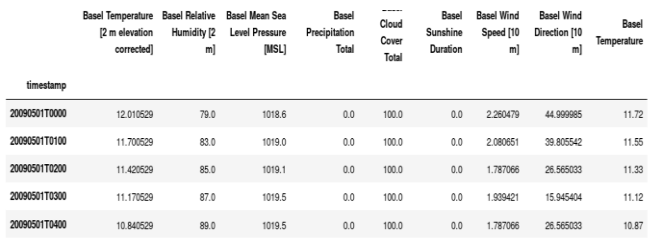
This article will therefore show just how easy and fast it is to train such a neural network, on inexpensive hardware, making accurate weather forecasting accessible even in remote locations.

II. Methods

A. Data Preparation

The data must be prepared such that a neural network can be trained for each weather feature.

First, Neural Networks require a lot of data. Our dataset was obtained from [6], for the city of Basel, Switzerland. More specifically, it contains hourly data for the last 11 years for parameters like Temperature, Wind, Sunlight. The full list of parameters is presented in Fig. 1.



timestamp	Basel Temperature [2 m elevation corrected]	Basel Relative Humidity [2 m]	Basel Mean Sea Level Pressure [MSL]	Basel Precipitation Total	Cloud Cover Total	Basel Sunshine Duration	Basel Wind Speed [10 m]	Basel Wind Direction [10 m]	Basel Temperature
20090501T0000	12.010529	79.0	1018.6	0.0	100.0	0.0	2.260479	44.999985	11.72
20090501T0100	11.700529	83.0	1019.0	0.0	100.0	0.0	2.080651	39.805542	11.55
20090501T0200	11.420529	85.0	1019.1	0.0	100.0	0.0	1.787066	26.565033	11.33
20090501T0300	11.170529	87.0	1019.5	0.0	100.0	0.0	1.939421	15.945404	11.12
20090501T0400	10.840529	89.0	1019.5	0.0	100.0	0.0	1.787066	26.565033	10.87

Fig. 1. Sample from the dataset

In total, the dataset contains 96456 samples with 9 features. Missing or erroneous entries were removed from the dataset.

Looking back on it, we could have dropped some of the features, for example, there are two instances of 'temperature', which may have led to some degree of over-fitting. However, the effect is minimal.

Second, the weather features are signed values and have different orders of magnitude. Therefore, standardization was applied by subtracting the mean and dividing by the standard deviation. This also has the effect of making computations much easier.

Third, the data was split, 80% for training and 20% for testing the neural networks.

Next, the data is treated as a time series [7][8]. Therefore, separate delay vectors are created for each timestep in the future to be predicted. So, to create the delay vector for the temperature three hours into the future, the following steps were taken:

Firstly, we need training and testing vectors, let's call them X_train, Y_train, X_test, Y_test. X_train consists of the first 80% entries and X_test the other 20% of all features. This is simple enough. Y_train and Y_test consist only of the 'temperature' entry delayed by three, since it's hourly data and we want the temperature, one hour, two hours and three hours later. So, those vectors are made by selecting all entries, starting from the second up to the fourth, in order to 'move' three hours into the future. Keep in mind that the 'X' vectors are shorter than the 'Y' vectors by three, since the numbering started three entries later, therefore, the last three entries are dropped from the 'X' vectors in order for them to be the same length. This applies to any kind of delay we want to add, so you have to keep it in mind. For our paper, a 24 and 48 hour 'future' was used.

Finally, a sliding window [9] will be used for the training data. This is the way in which we use multiple hours to predict the future. In the situation above, it was assumed that only one hour will be used. The program will work such that it will use 48 and 192 hours of history for the prediction. So, if we want to predict the 'temperature' in three hours with a 48-hour history window, the data will be arranged such that we feed all the 48 hours of data in the neural net. Therefore, our X_train has a shape of (77093, 48, 9) and our Y_train_temp has a shape of (77093, 24, 1). The order of the items in parenthesis means entry size, sample size, feature size. Remember, there are 77093 entries on the training variables because only 80% of the initial set will be used.

B. Working Principles

As of now, we have X_train and X_test, which consist of entries of all the features, and different Y_train and Y_test for each feature we want to predict. For temperature specifically let us call them Y_train_temp and Y_test_temp.

In the following paragraphs, a very quick explanation of neural networks will be given. Let x_n be an input vector, containing numbers, let's assume natural numbers for simplicity. Let w_n be a vector of scalars, called the weight vector, which contains other natural numbers. Fig. 2 illustrates an extremely simple neural network, consisting of an input layer of n neurons, where all items in x_n are placed, and an output layer, consisting only of one neuron.

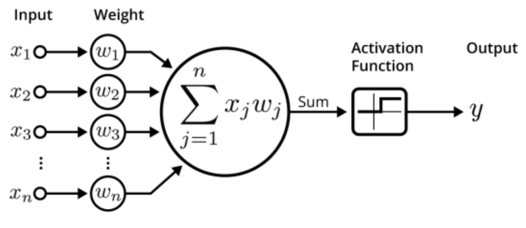


Fig. 2. Basic Neural Network Representation [10]

What is referred to as 'feeding forward' is the process by which, in this case, each value of the neurons of the input layer is multiplied by the corresponding value of the weight vector (x_1 is multiplied by w_1) and then, after this multiplication, all these values are added together. The number resulted after the summation is passed through an 'activation function' which has the sole purpose of translating the number in a limited domain of values, usually between –1 and 1 or 0 and one. After the activation function has done its job, the last neuron in the network represents what the network 'thinks' those inputs should output.

Consider a neural network which needs to learn how to predict the variable, or, vector of variables, Y in relation to the variable or vector of variables X. Datasets are usually given containing both X and Y with values for all of them. Some of the variables contained by our dataset (see the samples in Fig. 1) for example are Temperature [2 m elevation corrected], Relative Humidity [2 m] and Mean Sea Level Pressure. As it can be observed, the dataset has numerical entries for those variables for 10 years. If a toy neural network were to be trained to predict Temperature, the Y variable would become Temperature and the other variables would become the X variable. What is referred to as 'training' is the process where the X variables are 'fed forward' from the input of the neural network, the neural network then outputs a result, which represents its best estimation about what value the solution should have and then this result is compared to the actual Y value corresponding to the X values fed at the input (remember, the dataset is 'bijective' because all the columns are linked by a unique time of measurement'). To measure how different the neural network result is from the actual value, the 'error' function is used, and again, a multitude of error functions can be used. This error function computes the difference, and then, in order to adjust the weight vector, the error function is derived in regard to each weight in the weight vector. This derivative is later used to update the weights. This whole algorithm, in its most basic form is using Fermat's theorem, taking small steps to find the minimum of the error function. The most basic algorithm for this is called 'gradient descent' and the whole process of weight updating is called 'backpropagation'.

Lastly, the 'testing' process consists of using a portion of the dataset which was not used for training. Here, the inputs are fed through and again the outputs are compared to the actual values. This step exists to see if the neural network either 'generalized' the problem-solving or it just learned how to predict the training set very well, which is called 'overfitting'. We hope now it is clearer why we processed the data in the way that we did.

For the temperature example, X_train is fed at the input and Y_train at the output in order for it to train. Then the network is tested by 'feeding' it X_test and making it make predictions. We then compare those predictions with Y_test and see how close it was. Different error and activation functions [11] were tested. We settled on using the Mean Squared Error as our main error function, and the hyperbolic tangent and relu for our activation.

For the Sklearn implementation, the standard perceptron learning algorithm was used, better described here [12], and LSTMs [13] were used in Tensorflow. Keep in mind the different architectures, in Sklearn is feed-forward and in Tensorflow is recurrent.

Now, the best neural network architecture has to be found. In reality, neural networks are more complex than the example described above, with more neurons each layer and more layers, but the basic processes of training, backpropagation and testing are the same. Our project branches out in two directions, one using Sklearn and the other using Keras, both open-source and very well documented machine learning libraries that allowed us to quickly build neural nets. Both work on the same principle, that is, a separate network for each parameter. We decided to use every other input parameter to predict only one, for example, one network predicts 'temperature' using all other columns as inputs, one network predicts the 'humidity' using all the other columns all the other and so on. Why? Firstly, it's faster to train and secondly, it's harder for the model to over-fit.

C. Sklearn

On the Sklearn [14] part of the project, a simple grid-search type of program was implemented, so that it finds the 'best' neural net from a list of given hyperparameters. For the temperature example, the program functions as described in section B. It builds neural networks using different combinations of hyper parameters, trains them, tests them and picks the most accurate one. For the backpropagation, the program can either use the ADAM algorithm [15] or the stochastic gradient descent algorithm [16].

Regarding the number of layers and neurons per layer, here is a list of the neurons combinations (10, 10), (30, 10), (30, 15), (20, 20), (10, 15), all mixed with either an adaptive or constant learning rate, and a maximum number of 400 iterations, as this is a relatively small dataset. All the networks have the number of input neurons equal to the sample size multiplied by the feature size.

D. Tensorflow

Keras is the high-level API of Tensorflow 2.0, written in Python, designed with a focus on enabling fast experimentation [17]. Keras contains essential building blocks for creating complex neural network solutions. To enhance the speed at which different neural network architectures can be tested, Keras also supports GPU and TPU hardware acceleration.

We based our research on a previous multilayer perceptron network designed for energy usage forecasting. The results proved that reasonably accurate weather forecasting was possible and led us to continue our research in order to find networks better suited for this task. Various MLP, CNN and RNN network architectures were tested. We spent most of the time testing RNN networks with LSTM layers, as they're the optimal fit for this task.

The best results were achieved using a simple network with an LSTM input layer with 30 units and a fully connected output layer. We use 8 days of history (192 timesteps) to predict 4 days into the future (96 timesteps). The Adam optimiser [15] is used, with MSE as the loss function. To prevent overfitting, we used L1 and L2 kernel regularisation in the LSTM layer and implemented Early Stopping. The architecture is presented in Fig. 3.

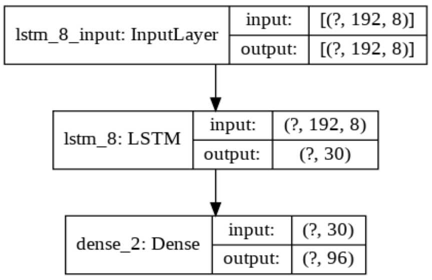


Fig. 3. Architecture of the Keras Network

III. Results

The obtained results can be most easily visualized graphically. However, as there were no meteorologists involved in this project, the precision could only be measured for the Temperature. As none of the authors are familiar with the scale or measurement units of different meteorological phenomena, the only thing that could be judged was temperature, were even for 24 hour predictions, the error is mostly within a 0.5 margin, which is highlighted by the blue 'tube' around the red prediction line.

Fig. 4 represents the predictions for a 96-hour span made by the Tensorflow implementation. Figures 5 and 6 represent the predictions made by the Sklearn implementations on a 1 hour and 24 hours span. As it can be easily observed, the Tensorflow implementation not only offers more accurate results, but on a longer time span. The, Sklearn implementation, while marginally more inaccurate, is still very usable.

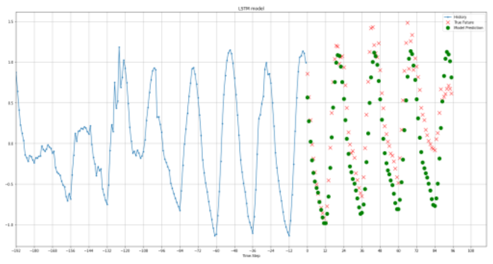


Fig. 4. Tensorflow Results for 96 hour future

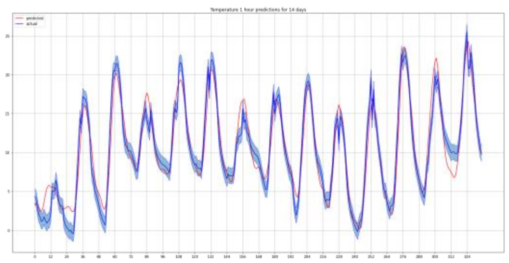


Fig. 5. Sklearn Results for 1 hour future

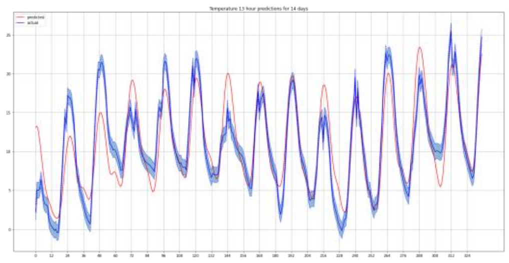


Fig. 6. Sklearn Results for 24-hour future

IV. Conclusion

While it is obvious that the Tensorflow implementation provides more accurate overall results, it must be noted that it is harder to use, takes more time to train and is harder to implement. It is very flexible though.

The Sklearn implementation, while not as accurate and not able to make predictions that far into the future, it is a lot simpler to use, train, understand and reuse. It only provides the user to select the number of basic neurons on each layer, it does not have other architectures or types of like Tensorflow, just dense layers.

Finally, for small, very localised applications of forecasting, we recommend the usage of Sklearn. For larger areas, where a lot more data is available and accuracy is needed, Tensorflow is a better choice. Overall, for local forecasting, both are adequate.

References

[1] P. Hewage, M. Trovati, E. Pereira, A. Behera, “Deep learning-based effective fine-grained weather forecasting model,” in Pattern Analysis and Applications, 2020.

[2] M. Holmstrom, D. Liu., “Machine Learning Applied to Weather Forecasting.”, Stanford, 2016.

[3] J. Hickey, “Using Machine Learning to “Nowcast” Precipitation in High Resolution”, Google, 2020.

[4] N. Singh, S. Chaturvedi and S. Akhter, "Weather Forecasting Using Machine Learning Algorithm," in 2019 International Conference on Signal Processing and Communication (ICSC), 2019, NOIDA, India.

[5] N.O. Jaiswal, “Weather Forecasting using Linear Regression In Machine Learning”, 2020.

[6] MeteoBlue, https://www.meteoblue.com/en/weather/, 2020.

[7] E. Dobrescu, D.I. Nastac, and E. Pelinescu, ”Short-term financial forecasting using ANN adaptive predictors in cascade” in International Journal of Process Management and Benchmarking, vol. 4, 2014, pp. 376-405.

[8] Tensorflow, “Time series forecasting”, in Tensorflow Documentaion, 2020, https://www.tensorflow.org/tutorials/structured_data/time_series.

[9] J. Brownlee, “Time Series Forecasting as Supervised Learning”, 2016.

[10] S. Saxena, “Artificial Neuron Networks (Basics) | Introduction to Neural Networks,” in Becoming Human, https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c, 2017.

[11] S. Bhardwaj, “Activation Function Basics,” in Kaggle, https://www.kaggle.com/general/187778, 2020.

[12] Y. Freund, R.E. Schapire, “Large Margin Classification Using the Perceptron Algorithm”. in Machine Learning, 1999, vol 37, pp.277–296.

[13] S. Hochreiter and J. Schmidhuber, “Long Short-term Memory. Neural computation,” in MIT Press, 1997, vol. 9, pp. 1735–1780.

[14] Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python” in JMLR 12, 2011, pp. 2825-2830.

[15] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in International Conference on Learning Representations, 2014.

[16] J. Kiefer, J. Wolfowitz, “Stochastic Estimation of the Maximum of a Regression Function,” in Ann. Math. Statist., 1952, vol 23, pp. 462-466.

[17] F. Chollet, et al., “Keras”, https://keras.io.