10/28/14
CS74: Machine Learning
Project Milestone Write-up

Location Independent Weather Forecasting

Andrew Pillsbury and Rebecca Leong

Overview:

Weather forecasting has traditionally been treated as a well-studied physics-based phenomenon for a specific location. Even so, weather still exhibits data patterns that can potentially be utilized as a basis for future prediction. By gathering data from a variety of different locations in the continental United States, we are attempting to create a location-neutral weather forecaster. For any given location, the model will predict 24 hours of future weather based on data from the past 72 hours in that location. This will offer significant benefits over current methods of weather forecasting that require region-specific models.

Data:

The data we are using is quality controlled local climatological data from the National Climatic Data Center (http://cdo.ncdc.noaa.gov/qclcd/QCLCD?prior=N). We have gathered hourly weather data from every day in 2013 from nine different cities, one for each climate region in the United States. These cities are: Seattle, WA, San Francisco, CA, Cortez, CO, Bismarck, ND, Dallas, TX, Atlanta, GA, Indianapolis, IN, Minneapolis, MN, Boston, MA. The weather stations record data roughly 40 times per day at varying intervals, so to normalize the time between recordings we only take the first reading each hour, which gives us 9 cities * 365 days/city * 24 hours/day * 1 reading/hour = 78,840 readings. Each reading has six features: visibility, temperature, dew point, wind speed, wind direction, and pressure. Each input vector consists of a reading of these six weather features from every hour in the 72 hours before T, where T is the beginning of the 24-hour period for which we wish to predict the weather. The output vector consists of readings of those same six weather features at every hour from T to 23 hours after T.

Algorithm:

We have implemented a basic feed forward backpropagation neural network with the option of zero, one or two hidden layers. The number of neurons per layer is currently fixed to the number of features in the data set, but will later be set by a hyper-parameter. Using stochastic steepest descent, the error from the final output is backpropagated to adjust the weights into each neuron. The neurons on each of the hidden layers perform a non-linearity soft-threshold function (*tanh*). The output layer contains neurons equal to the number of output features. These neurons do not perform the threshold function since we want continuous output values.

The forward pass through the network functions by taking the input to a layer and then scaling by its respective weights to generate a signal. This signal is then passed through the non-linearity function to generate the input for the next layer (or the output). [1]

$$x_j^l = g\left(\sum_{i=0}^{d^{l-1}} w_{ij}^l x_i^{l-1}\right) \text{ where } g(x) = \tanh(x)$$

[1] "Lecture 10 – Neural Networks (Professor Yaser Abu-Mostafa)." *YouTube*. YouTube, 6 May 2012. 26 Oct 2014.

$l = current\ layer$
$j = number\ identifying\ output\ neuron$
$i = number\ indentifying\ input\ from\ neuron\ i$

Once the forward pass through all layers of the network is completed, the error is calculated using the predicted and actual values. This error is used to calculate a delta value which indicates how the weights for each layer should be adjusted. This final error is a function of the weight matrix that adjusts values into the neuron. Thus in order to compute the gradient, we want to derive in respect the weights of the current layer. This derivative can be solved using the chain rule for the two functions involved in the layer – the scaling by weights and the neuron function (which is just the linearity transfer function).

$$\nabla e(\boldsymbol{w}) = \frac{\delta\, e(\boldsymbol{w})}{\boldsymbol{\delta} w_{ij}^{l}} = \frac{\delta e(\boldsymbol{w})}{\delta s_j^l} \times \frac{\delta s_j^l}{\boldsymbol{\delta} w_{ij}^{l}} \qquad (final\ layer)$$

$$= 2(x_i^l - y_n) \times x_i^{l-1}$$

$where\ s_j^l\ indicates\ the\ signal\ (\boldsymbol{x} \cdot \boldsymbol{w}) entering\ the\ neuron$

For the hidden layers, we cannot directly calculate the output error for that layer, but we can infer the error using the error gradient of the layer after it. Thus we can take the gradient value for the final layer and recursively calculate the gradient values for all the prior layers.

$$\nabla e(\boldsymbol{w})_j^{l-1} = \frac{\delta\, e(\boldsymbol{w})}{ds_j^{l-1}} \times \frac{\delta s_j^l}{\boldsymbol{\delta} w_{ij}^{l}} = \left(\frac{\delta e(\boldsymbol{w})}{\delta s_j^l} \times \frac{\delta s_j^l}{\delta x_i^{l-1}} \times \frac{\delta x_i^{l-1}}{\delta x_i^{l-1}}\right) \times \frac{\delta s_j^{l-1}}{\delta w_{ij}^{l-1}} \qquad (Hidden\ Layers)$$

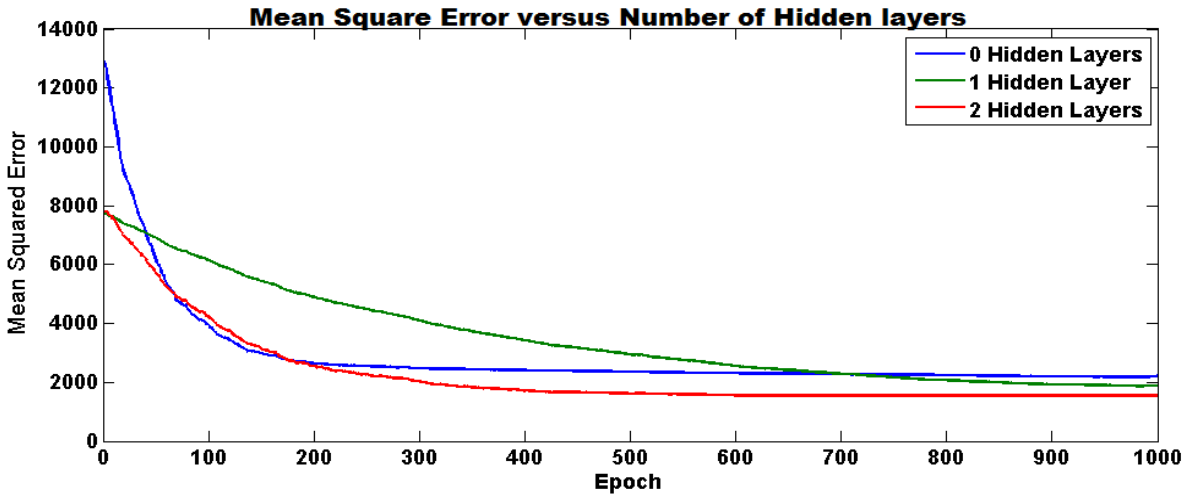
$$= x_i^{l-2} \times \sum_{j=1}^{d^l} \left(\frac{\delta e(\boldsymbol{w})}{\delta s_j^l} \times \boldsymbol{w}_{ij}^{l} \times g'(s_i^{l-1})\right)$$

Our algorithm starts by loading the randomized data set and allocating subsets for various purposes. Half of the data (Train set) is used for stochastically training the network. A quarter of the data (validation set) is used to evaluate the current network error to determine convergence and overtraining. Currently we do not have convergence checks in the algorithm and are just running on a set number of iterations. The final quarter of data (test set) is used to evaluate the network error after the training is complete. The error in the network is calculated as mean square error per sample per feature.

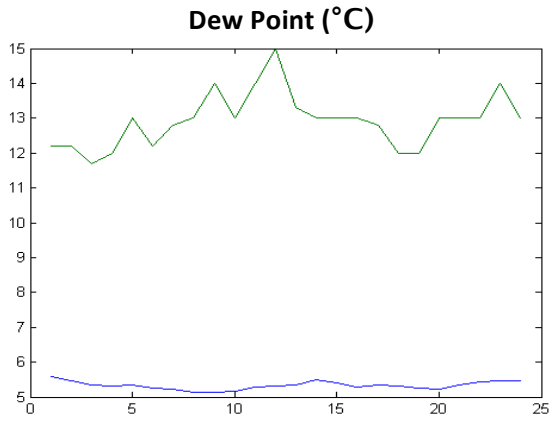
Results:

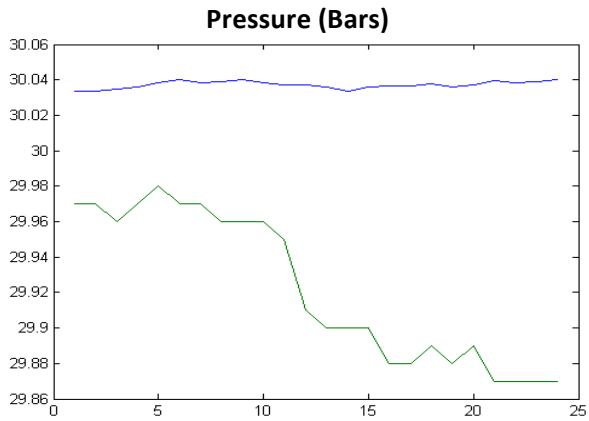
We trained the network using no hidden layers, one hidden layer, and two hidden layers for 1000

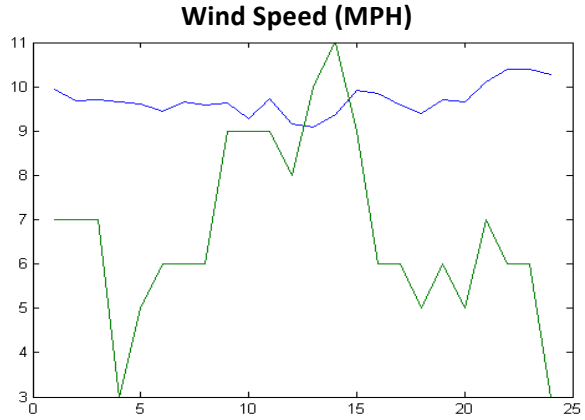
epochs. The graph below shows the error decreasing for each network type (zero, one, or two hidden layers) as we train the network.

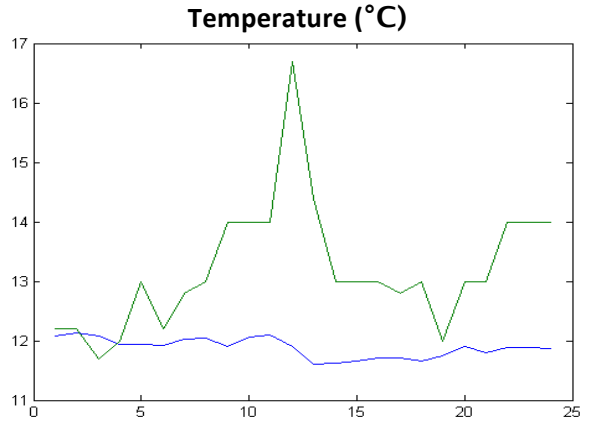


The following graphs show our predicted values for a specific 24-hour period and the actual values for the same period. The prediction was made using a network with two hidden layers that was trained for 500 epochs. In each graph, the blue line is the predicted values, the green line is the actual values, the y-axis is the value of the data, and the x-axis is time in hours.









Future Steps:

We met part of our proposal milestone by having a working neural network, but did not completely implement the dynamic portion of it. We've read through a lot of literature about dynamic neural networks but are still working on implementing it. Based on our results, our model isn't accurately modeling the variance and complexity of our data. A dynamic neural network, which will take into account the time dependent nature of our data, will hopefully allow us to treat the three days of input data as a pattern over time rather than a single linear combination.

Another improvement we need to make is to normalize the error. Currently we just compute the mean square error across all of the features in all the samples. But the ranges of each feature vary from values between 0 and 10 and values between 0 and 360. So certain features are weighted more heavily in our error calculations.

After making the neural network dynamic and adding convergence checking conditions, we still need to tweak the number of hidden layers, number of neurons per layer and learning rate to minimize network error. We may use an adaptive learning rate to improve the speed of convergence if necessary. We will choose these hyperparameters by running a cross validation over a set of values. Then we can train the model on the entire training set and test on U.S. cities not in the training set. Also, we will test on weather data from cities outside of the U.S. to see how generalizable the model is.

References:

Bontempi, Gianluca, Souhaib Ben Taieb, and Yann-Aël Le Borgne. "Machine learning strategies for time series forecasting." *Business Intelligence*. Springer Berlin Heidelberg, 2013. 62-77.

de Kock, Matthew. "Weather Forecasting Using Bayesian Networks." University of Cape Town. (2008)

Haerter, Fabricio P., and Haroldo Fraga de Campos Velho. "New approach to applying neural network in nonlinear dynamic model." *Applied Mathematical Modelling* 32.12 (2008): 2621-2633.

Lai, Loi Lei, et al. "Intelligent weather forecast." *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*. Vol. 7. IEEE, 2004.

Rasouli, Kabir, William W. Hsieh, and Alex J. Cannon. "Daily streamflow forecasting by machine learning methods with weather and climate inputs." *Journal of Hydrology* 414 (2012): 284-293.

Shrivastava, Gyanesh, et al. "Application of Artificial Neural Networks in Weather Forecasting: A Comprehensive Literature Review." *International Journal of Computer Applications* 51.18 (2012): 0975-8887.

Sinha, N. K., M. M. Gupta, and D. H. Rao. "Dynamic neural networks: An overview." *Industrial Technology 2000. Proceedings of IEEE International Conference on*. Vol. 1. IEEE, 2000.

Soderland, Stephen. "Learning information extraction rules for semi-structured and free text." *Machine learning* 34.1-3 (1999): 233-272.

Williams, John K., et al. "A machine learning approach to finding weather regimes and skillful predictor combinations for short-term storm forecasting." *AMS 6th Conference on Artificial Intelligence Applications to Environmental Science and 13th Conference on Aviation, Range and Aerospace Meteorology*. 2008.

Gupta, Madan M., Liang Jin, and Noriyasu Homma. *Static and dynamic neural networks from fundamentals to advanced theory*. New York: Wiley, 2003.