**Guy Dor**

**CSC 578 Section 701**

**Final Project -- class Kaggle competition**

**Kaggle user name: guydor11 (appears as Guy Dor), 13th place in the private leaderboard  
Video:** [**http://youtu.be/\_uiBEZeYF9o?hd=1**](http://youtu.be/_uiBEZeYF9o?hd=1)

**Kaggle Submission Model:**

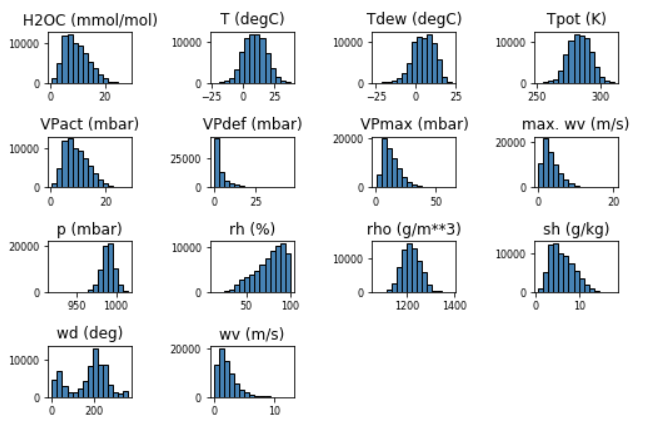
My best performing model was, in fact, a very simple LSTM model:

* 1 LSTM initial layer of 240 nodes.
* 1 densely connected layer of 1 output node.

This yields 245,041 trainable parameters.

While this may seem like a “lazy model” to construct, the results speak for themselves. I will go into further detail below.

**Data Preparation:**I would say that I had spent about 85% of the time for this project working on how to prepare this data properly for the model. The toughest part was restructuring the data into the time step format discussed in class. As well, if the data wasn’t sliced into the right dimensions, I needed to look at the original csv file line by line to find the correct index values. Beyond that, I scaled the data (both the x and y values) using a MinMax scaler from 0 to 1. Looking at the figure below, I knew needed to normalize the data across the board:



Originally, I also used -1 to 1 scaling, but found that 0 to 1 yielded better performance.

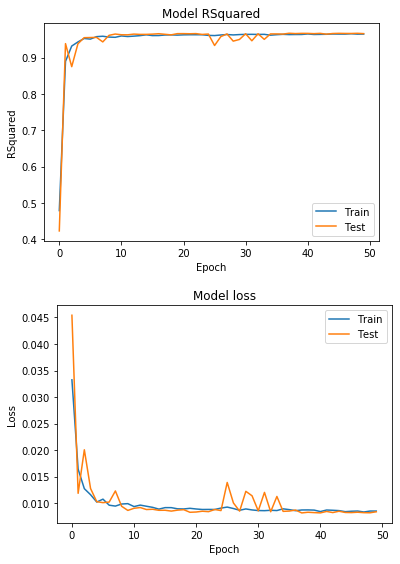
**Parameter Tuning:**Hyperparameter tuning was an arduous task as I experimented with multiple types of models with different layers (multiple LSTM layers, dense Layers, ReLU activation layers, and Repeat Vectors), applying dropout and other regularizations such as L1. However, each of these methodologies either yielded a poorer learning performance than a simple single LSTM layer or tended to overfit very easily. The closest performance I yielded from the single layer model was with:

* Two 6-node LSTM layers,
* a 240-node Dense layer,
* a ReLU activation layer ,
* a 24-node RepeatVector layer,
* a Flatten layer,
* and a 1 node Dense output layer.

Unfortunately, this model was still nowhere near as accurate as the simpler model.

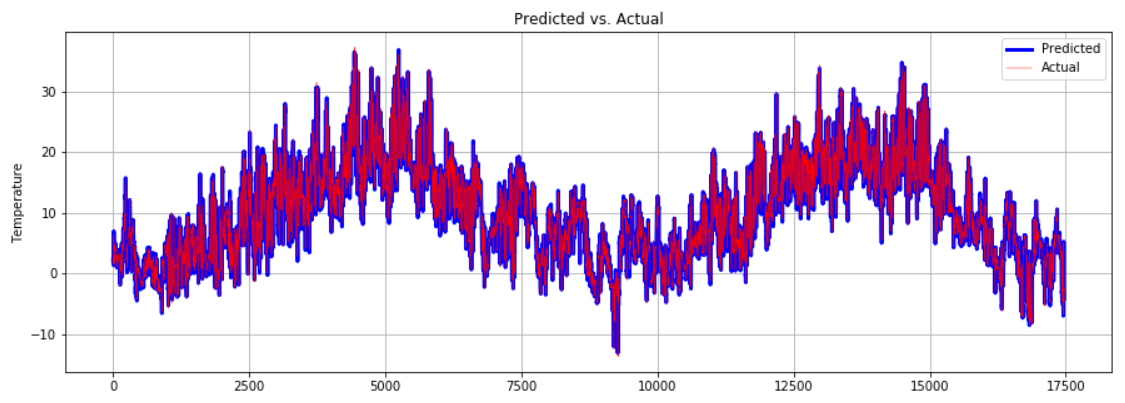
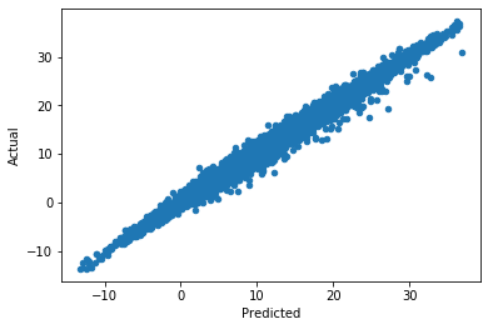
As well, configuring these models to more complex layers only slowed down the training, each epoch running for 3-4 minutes at a time. If I wanted to get better results, I needed to run through more models and at a faster rate.

Once I decided to stick to one LSTM layer, I found that a lot of the tuning was based on two things; making sure the training data is not randomly shuffled and setting the batch size. When removing random shuffling, the model actually “sees” the training data as a time series (i.e. the training data is treated as a stochastic process rather than a set of data that can be broken out randomly) , so I decided to set Shuffle to False when fitting the model. With batch sizing I started at 50, which performed very slowly and did not show much promise after 50 epochs. I then decided to add a zero for a 500 batch size. This model learned faster, but also learned very erratically and I wanted to smoothen the performance. My happy medium was setting the batch size to the number of nodes in the layer, 240. This model not only learned very quickly but also had very little sign of overfitting at 50 epochs (See below).



**Estimation and Prediction:**

My model would generate (before inverse scaling) a 96% R-squared value and about .008 in mean absolute error. After rescaling, my best mean absolute error (as shown in Kaggle) was about .492. I also visualized the forecasted and actual results shown below.



**Conclusion:**

While this was a great exercise in constructing a recurrent neural network model, it was also very time consuming. Training time of each model I designed alone took at least half an hour each on CC Labs using a CPU. And in between each model I would experiment with another way to tune the hyperparameters which took time to research and explore other options.

This was overall a fun project; it gave me opportunity to explore time series analysis from a different perspective. Though I am not necessarily a fan of a “black box” model for time dependent multivariate data, since I would like to better understand what features were determining the weights more than others, it was still interesting to see how well the model predicted the temperature at such a granular level. I hope to use these techniques in the future and would like to further explore more into the realm of Artificial Intelligence.