Robert Kaszubski

CSC 578 – Final Project

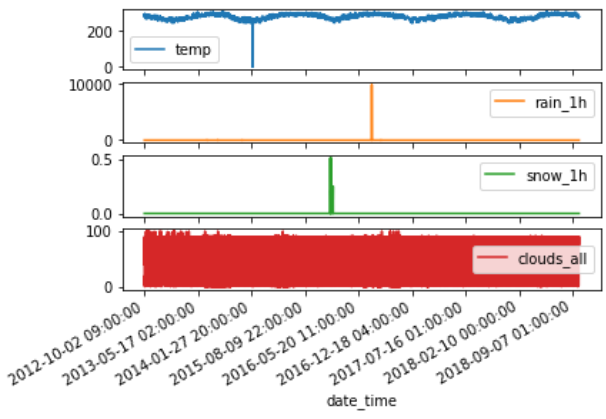
Kaggle Name: **Robert**

Kaggle Rank (11/21) : **9th place: 297.75**

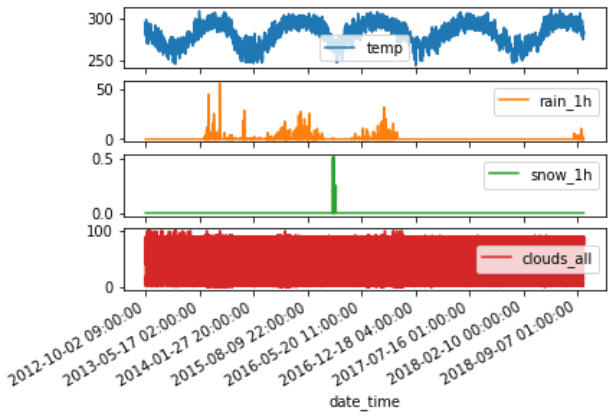
Exploratory Data Analysis

The exploratory phase began with examining the first few rows of the dataset. I noticed that there was a datetime datatype that would need to be converted, as well as three categorical variables (holiday, weather description, weather_main). Using code from the TF tutorial, I performed a similar transformation (feature engineering) by reading the datetime object and creating variables corresponding to time of day and time of year. The next step was to convert the categorical variables into dummy variables or one hot encoding. This means that for each of the categories in each categorical variable, a separate column was formed with either a 0 or 1. Only one value in each data row would have a 1 indicating that the original category it belonged to hence the name one hot. This allows the categorical variables to be used in the neural network.

The next step was to look at the distributions of the data via a statistical summary as well as a plot of all numeric data excluding the target variable in proportion to time. This showed a few errors or anomalies in the data that would have to be corrected. The first was an extreme max value in the rain_1h variable of 9831mm. I found that the most rain ever recorded in the span of an hour was about 300mm, so this value had to be altered. Any value above 300 was simply set to 0. For the temperature variable, the minimum was 0 Kelvin. That is absolute zero and impossible. This value was adjusted to the mean of the temperature variable. This fixed the egregious anomalies in the data that could have thrown off the network.



Before Cleaning



After Cleaning

I also checked the distributions of the numerical variables by creating histograms. Temperature has a near normal or symmetric distribution with most of the data being in the middle range of temperatures with fewer data toward the upper and lower extremes. The cloud percentage variable was the most interesting as the two most frequent values were either 0% or 100% cloud cover with much

fewer data points along the middle. As far as rain or snow, most of the data set saw neither during most hours. This can be confirmed looking at the statistical summary with the 3rd quartile being at 0 for both. Likewise, this is the case with most of the dummies created, there's only a few instances were there are holidays or there is some specific weather condition.

Splitting, Window Generator, & Final Preparation

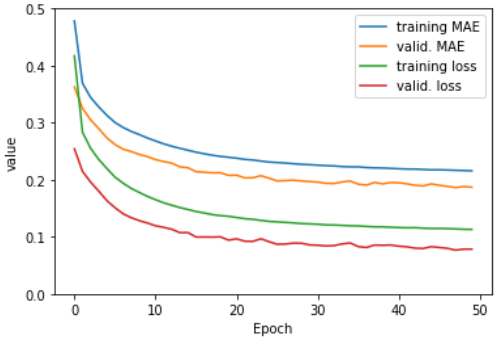
At this point I was able to split the dataset. The last 5000 became the testing data, the subsequent 7000 became the validation data, and the rest became the training data. The data was then normalized using z score normalization. This did conflict with one of the dummy variables created as it had only 1 instance where it was true. This variable was dropped as its influence was not substantial. Dummy variables or one hot vectors don't have to be standardized but it doesn't change the outcome. All the 0's remain 0's and only the one hot category remains greater than 0. I did make attempts to standardize only the numeric variables by splitting the data frame, but this created errors with the dataset upon joining it back together.

All the window generator code was copied from the TF tutorial. I created a window that had an input width of 7, label width of 1, and shift of 3. This is because we are using 7 hours of data, reporting only 1 hour, and that 1 hour is 3 hours after the end of our input. The total window size is therefore 10. I used the TF tutorial plot code to create a sample plot ensuring my window was correct.

I added in the code for making datasets from the TF tutorial except with a slight modification. I didn't want the test and validation data to be shuffled. I made a separate function to make the training dataset and a separate function for the testing and validation datasets. Then the code to compile and fit a model was added. I also modified this slightly. I added restore best weights to the early stopping callback, this made sure the weights from the best epoch were used. I also parametrized the max epoch variable. This made training a lot easier. Following this, I wrote a function that would make predictions using the testing data, denormalize it, and then export it to a csv for submission to Kaggle. I also wrote a function that would plot the MAE and loss for the training and validation data to visualize model performance.

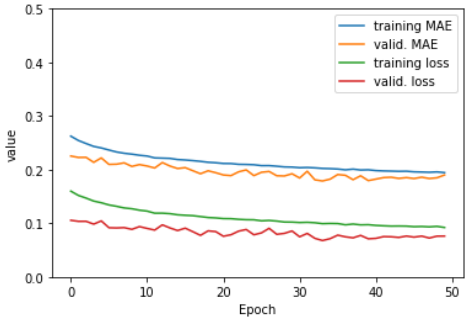
Baseline Model

My baseline model has one LSTM layer with 7 units, followed by a single dense output layer, with a single output. This was trained over 50 epochs, with an early callback patience of 10. The validation loss was .0816, and the validation mean absolute error was .1925. I made this my baseline as I wanted one recurrent unit per time series input. I trained over 50 epochs, as this took about 5 to 6 minutes per run, keeping training reasonable for all the subsequent models and variations. It is fairly similar to the TF tutorial single step RNN model except it has fewer units. The baseline that was used in the TF tutorial was only returning the last input value. I didn't want to use this as it wasn't learning or doing any training just returning an existing value. I didn't think that this would be useful. used this as a baseline because it was simple but already returning decent results. I submitted this to Kaggle as a test and it scored a 371.42.



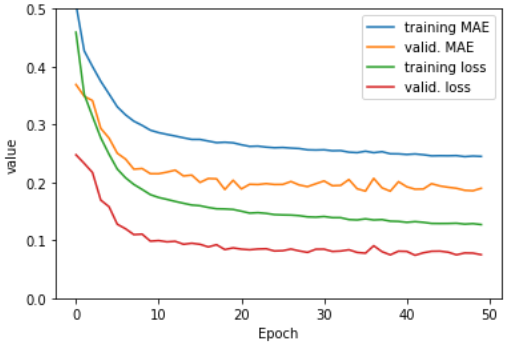
Best Model

My best model had 4 stacked LSTM recurrent layers. The number of units per layer was 9, 9, 9, and 3 respectively. This was followed by a dense or fully connected layer with 10 neurons that fed into the single neuron output. The model was trained over 50 epochs and resulted in a Kaggle score of 297.75 which placed me just below the Kaggle baseline. During validation the model had a loss of only .0646 and a mean absolute error of .1735. I was a little surprised that this ended up being the best model. My runner up model and even my model without anything but the 4 stacked LSTM layers got slightly lower validation results but when submitting to Kaggle, scored 20 to 30 points higher.



Runner Up Model

My runner up model was fairly similar to my best model. When I was experimenting with additional dense layers and dropout, I had a model that performed well but inconsistently. It consisted of the same 4 stacked LSTM recurrent layers with the number of units per layer being 9, 9, 9, and 3 respectively. This was followed by a dropout layer with dropout set to .1. The dropout layer was connected to the first fully connected or dense layer which had 9 nodes, this connected to another fully connected layer with 3 nodes before then connecting to the singular output node. I also submitted this model to Kaggle and it netted a score of 334.25, not quite as good as the best model but still an improvement from the baseline.



Hyperparameter Tuning and Model Development

I wanted to attempt and explore several different features and changes to the model architecture in hopes of finding a better model. I started by changing the number of recurrent units in the single layer I had. I expected that a model with a larger number of units would work better. I wanted to test a large range of numbers both low and high as well as the number used in the TF tutorial which was 32.

Changing number of units:

Units	Loss	MAE	Time
7 (Baseline)	.0816	.1925	6s
4	.0861	.2010	6s
8	.0836	.1944	6s
9	.0716	.1825	6s
10	.0792	.1898	6s
11	.0753	.1882	6s
12	.0919	.2032	6s
14	.0961	.2018	6s
20	.0943	.2028	6s

25	.0934	.2058	6s
30	.0880	.2056	6s
32	.1016	.2165	6s
50	.0930	.2091	6s
100	.1016	.2175	6s

It seems like with only a single layer, the model is too complex and likely overfitting if I'm using too many units. The best results so far have come from using 9 units. However, I'll keep 9 units going forward but I'll come back to this after adjusting other hyperparameters. It is interesting to note that odd numbers of units seem to work best.

Changing number of recurrent layers:

Layer Count (units)	Loss	MAE	Time
2 (9,9)	.0812	.1873	7s
3 (9,9,9)	.0940	.2037	9s
4 (9,9,9,9)	.0752	.1793	10s
5 (9,9,9,9,9)	.0875	.1973	12s

I expected that adding more layers would marginally improve the MAE. I used the same number of units for each layer which likely hindered the model. Using 4 layers netted the best results, however I'd like to try adjusting the number of units in each layer.

Changing number of recurrent layers and number of units:

Layer Count (units)	Loss	MAE	Time
2 (9,1)	.0815	.1967	7
2 (9,7)	.0833	.1905	7
2(18,9)	.0840	.1896	7
4(9,9,9,3)	.0673	.1713	10
4(9,9,3,3)	.0799	.1810	10
4(32,9,3,3)	.1032	.2057	10
4(12,9,9,3)	.0781	.1790	10
4(9,9,6,3)	.0802	.1822	10
4(9,9,9,7)	.0717	.1738	10
4(9,9,9,1)	.0780	.1852	10
4(9,9,9,5)	.0761	.1833	11

I tested this extensively and tried several different layer combinations. I stuck with using either two or four stacked recurrent layers. This showed some improvements using 9,9,9,3 units for each respective LSTM layer. I attempted to increase the number of units in the early layers, this did not improve performance. I suspect the model becomes a little too complicated then and might be overfitting.

Adding additional dense layers with traditional dropout:

I was curious to see if adding more fully connected layers would change results. Currently the model only had a single output layer with only one output unit.

Dense Layers (units)	Dropout	Loss	MAE	Time
2 (10,1)	0	.0646	.1735	8
2(10,1)	.1	.0764	.1860	8
2 (10,1)	.2	.0831	.2096	8
2(3,1)	0	.0745	.1761	8
2(3,1)	.1	.0684	.1772	8
2(3,1)	.2	.0781	.2005	8
2(9,1)	0	.0723	.1751	8
2(9,1)	.1	.0776	.1794	8
3(9,9,1)	0	.0745	.1828	8
3(9,3,1)	0	.0707	.1776	8
3(9,3,1)	.1	.0651	.1709	8
3(9,3,1)	.2	.0909	.2088	8

It seems like adding more fully connected layers did slightly improve performance when compared to the model without them. I also tried to add dropout to the fully connected layers, and it seems like the loss and mean absolute error increased the more that was added. It looks like the best option is keeping the model simple with just one additional dense layer with 10 units and no dropout – this became my best model. The one exception to this was using three fully connected layers with .1 dropout. This netted comparable results to the single dense layer but may be overcomplicating the model. I submitted both to Kaggle and using the layer with 10 units netted better results.

Bidirectional Layers:

I wanted to see if the model would perform better using Bidirectional LSTM layers with the same units as the best model from the previous iteration (9, 9, 9, 3). I expected that there wouldn't be too much of a change. I don't think this application is ideal for bidirectional. The results confirmed my expectations, there was slight decrease in performance with each configuration I tried. However, the time it took to run almost doubled as each layer was essentially running two LSTMs.

Configuration	Loss	MAE	Time
(9,9,9,3) without added Dense layers	.0776	.1825	13
(9,9,9,3) with added Dense layers (9,3,1)	.0758	.1800	13
(9,9,9,3) with added Dense layer (10)	.0820	.1859	13

Recurrent Dropout:

I attempted to add recurrent dropout. However, this slowed the run time because the cuDNN kernals do not support it, so it defaults to a generic GPU kernel. The run time was about 25 seconds per epoch which was not reasonable for training. The recurrent dropout added was .5 to the first LSTM layer. This

led to a loss of .0761 and MAE of .1896. The results weren't bad, but the performance was too much of a hinderance to continue with this approach. I did try adding recurrent dropout to multiple layers, but this just compounded the runtime issue leading to a run time of over a minute per epoch.

Other Attempts:

I attempted a few other methods that ended up with either poor performance or errors. I tried using different batch sizes. To do so I adjusted the make dataset function that creates the training, testing, and validation sets. This didn't result in any significant changes but did increase the performance time when I decreased the batch count. I also tried to implement a CNN + RNN model but I couldn't get it to work. I'm not sure if it would have improved performance but it would have been fun to get working.

Conclusions

This was a fun project just as Homework 7 was albeit a lot more challenging. Overall, I think my final best model was adequate although I am not fully satisfied with it. I am also disappointed that I was just short of passing the baseline model on Kaggle. I noticed that training my models was also a lot more inconsistent that before. Sometime, my models would train well, then I would train them again and get different results. This is normal with neural networks and especially with the shuffled training data, but I felt as though it was more noticeable now than when we did CNNs in the previous assignment. I wish I was able to do a little bit more with recurrent dropout. Having more time, I would have also liked to mess around with some of the other recurrent neural network layers that Keras offers such as Gru and simpleRNN. Nonetheless, I am still mostly pleased with how this assignment turned out and I'm glad that I was able to hit the top 10 on the leaderboard, at least at the time of writing this.

I thought that this was a great class although it did go a little quickly particularly at the end. I wish DePaul didn't have 10-week quarters so that we could spend a little more time on each topic we covered. I also wish that there was a bit of a larger gap between assignment 7 and the final project just to give us a little bit more time. I learned a lot during the class and I hope to continue learning about deep learning, AI, and neural networks in the future.