Definition

Project overview

Space Weather prediction is becoming an increasingly interesting subject of study due to the large numbers of complex systems that are in use on a daily basis that are sensitive to the "space weather" environment. At it's core, the space weather environment is driven by the sun and high-speed plasma that is constantly being ejected from the sun. The vernacular term for this plasma is the "solar wind". In scientific communities the solar wind is called the "Interplanetary Magnetic Field" (IMF). The IMF interacts with the earth's magnetic field (i.e. the "magnetosphere") and can cause observable and sometimes detrimental effects on earth and in near earth space. Particles in near earth space can be energized during periods of prolonged southward oriented IMF (i.e. when the solar wind plasma's magnetic field is pointing southward) or when the IMF pressure on the magnetosphere suddenly increases. These excited particles tend to drift in the same direction and the drift direction is dependent on their charge (protons drift westward, electrons drift eastward). These drifting particles create the ring current. Enhanced ring current is the metric which is used define "geomagnetic storms". This is important for a number of reasons:

1. Satellite electronics are sensitive to high energy electrons that can be produced during strong geomagnetic storms.
2. The ring current can create magnetic field perturbations on the earth. Those perturbations can (among other things) destroy transformers that the power grid relies on.

Understanding and predicting geomagnetic storms and their severity would be very useful to people in these and other industries.

There are a few indexes used to measure the severity of a magnetic storm. The most common is the Disturbed Storm Time (DST) index which is calculated from ground based perturbations at low latitudes. The values are computed using hour long averages. This index is not calculated in real-time because it also relies on subtracting off a "quiet" baseline that is determined very infrequently. A provisional version is available for use where the official DST has not been computed yet. Another version of the DST that is calculated in real-time and with a much higher time resolution is the Sym-H index. The Sym-H index will be the subject of this report.

For those interested, a list of geomagnetic storms can be found here.

Problem statement

As stated in the overview, the goal of this project is to be able to predict future values of the Sym-H index based readily available data sources. In general, the magnetosphere's response to IMF is not only dependent on the current state of the IMF, but also previous states of the IMF. If it helps, you can think of the magnetosphere as a flywheel. Once it gets spun up, it tends to keep going but it takes significant effort to get it moving in the first place. Because of this, we need to use IMF data from the preceding hours in order to make an effective prediction. It is also likely that we won't be able to make predictions for more than an hour or two in advance because the IMF could suddenly change which would alter the prediction. To keep things reasonable for a first pass, the goal of this project will be to construct a model that can predict the next hour of Sym-H given IMF parameters that are available at the current timestamp.

The dataset that will be used is the OMNI dataset which can be downloaded from ftp://spdf.gsfc.nasa.gov/pub/data/omni/high_res_omni/. The 5 minute averages are more than adequate for this project. The OMNI dataset has solar wind parameters (and conveniently Sym-H) starting in 1981 and continuing to the present. While OMNI data is not available in real-time, the source satellite data (ACE) is available and could be used instead of the omni data with little effort.

To do the prediction, I will use a Recurrent Neural Network (RNN). Specifically, I will construct a network that has some LSTM or GRU layers. RNNs are great for time-series predictions since they can understand and account for the sequence dependence of the inputs that many other machine learning algorithms would neglect.

Metrics

Automated performance measurement of this model should be relatively straight forward. Using Mean Squared Error (MSE) to compare model predictions to actual observations should suffice. Physically, we can think of that as minimizing the distance between the predicted Sym-H and the observed Sym-H which seems like exactly the right thing to do.

While MSE might be a good metric to use for automated model tuning, nothing substitutes for actually spot-checking results against things which you actually want to predict.

Analysis

Data Exploration

The OMNI dataset is a timeseries dataset that has many different variables. The data comes from satellites that are sitting in the solar wind and taking measurements. Of interest for our study, the dataset contains:

1. Magnetic field vector components (B_x, B_y, B_z)
2. Solar wind speed components (V_x, V_y, V_z)
3. Solar wind density (particles per cubic centimeter)

4. Solar wind temperature
5. Solar wind dynamic pressure (computed from velocity, density and temperature)
6. Sym-H

The dataset is only available for times when there was a satellite with measuring capabilities in the solar wind. Prior to August 1995, there are many missing values (due to lack of satellites). After August 1995, the data has much higher fidelity. If we use the 5-min Omni data between August 1995 and Jan 1, 2017, there are about 2.25 million datapoints per time series. Past research and experience has taught us that B_y and B_z play an important role in geomagnetic storm generation. Average values for both are approximately 0 which shows that there is no preferred magnetic field orientation in the dataset. The standard deviations of the magnetic field components are also less than 5nT. Extreme values can be around 50nT (10 sigma) which implies that our distribution might have a longer tail than a typical gaussian. Pressure also plays an important role in storm generation because high pressure solar wind can cause the magnetosphere to shrink rapidly which energizes the magnetospheric plasma. Average pressure values are around 2 nPa.

	SymH	**B_y**	**B_z**	**DynamicPressure**
mean	-1.224470e+01	7.962645e-02	-9.993451e-03	2.127520e+00
std	1.957057e+01	4.103176e+00	3.258576e+00	1.691478e+00
min	-4.880000e+02	-4.554000e+01	-5.151000e+01	1.000000e-02
25%	-1.900000e+01	-2.520000e+00	-1.600000e+00	1.230000e+00
50%	-9.000000e+00	7.962645e-02	-9.993451e-03	1.820000e+00
75%	-1.000000e+00	2.660000e+00	1.570000e+00	2.430000e+00
max	1.360000e+02	5.660000e+01	5.410000e+01	9.777000e+01

Table 1: Dataset statistics

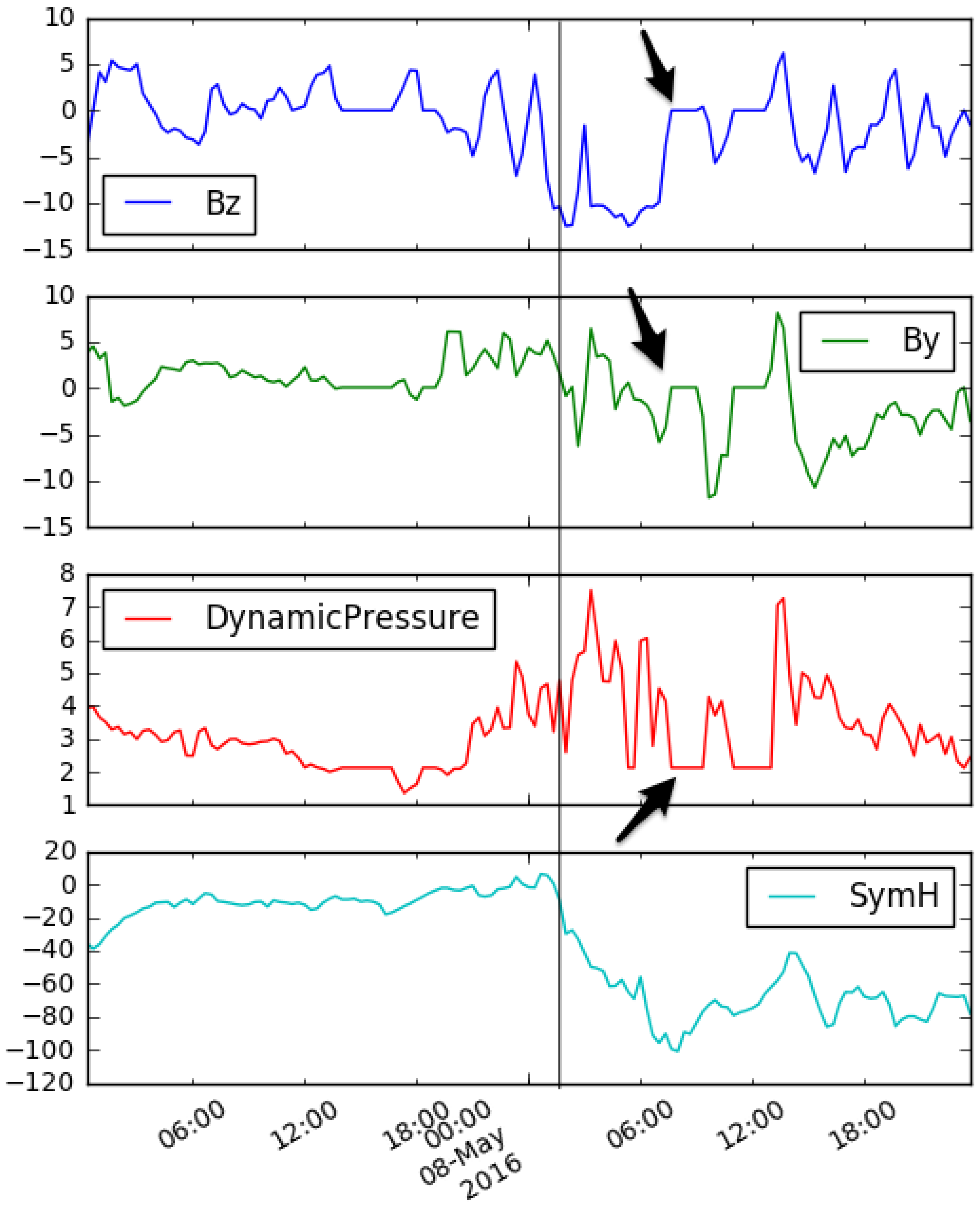


Figure 1: Geomagnetic storm on 2016-05-08

Figure 1 shows a plot of a single storm that started on 2016-05-08. At the time marked in the figure, you can see a sharp drop in the SymH index. This drop is accompanied by an enhancement in the magnitude of B_z and B_y and also an enhancement in the dynamic pressure. Careful inspection of the figure also shows some periods where the data is abnormally horizontal (e.g. at around 08:00 on the 8th of May). These are places where missing data has been filled with the dataset mean values which will be discussed in greater detail in the preprocessing section.

Geomagnetic storms (defined as times when SymH < -50nT) are relatively infrequent in the dataset (only about 3.5%). A storm can last from about 1 day to 3 days and generally has a very repeatable structure. The structure's sequence is:

1. a quick bump in the SymH (right before the vertical line). This is known as the "sudden storm commencement" or SSC
2. A rapid decrease in SymH over the next ~12 hours
3. Levelling and slow recovery to nominal values over the next 24 hours to 60 hours.

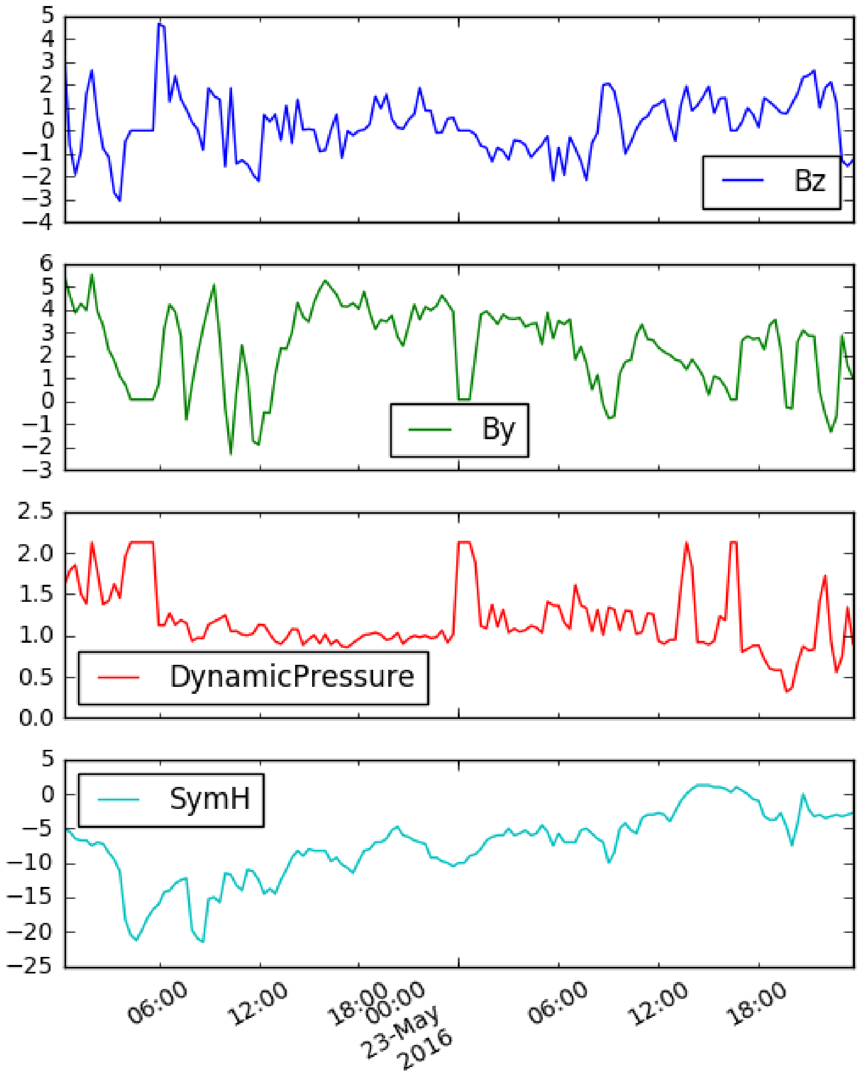


Figure 2: Example of nominal solar wind parameters from 2016-05-22 until 2016-05-24

In contrast to Figure 1, Figure 2 shows nominal solar wind conditions from a couple days chosen randomly from the dataset. We see that the solar wind fluctuates slightly almost all the time but generally fluctuates between +/-1σ of the mean.

Methodology

Data Preprocessing

As noted in the data exploration section, prior to August 1995, a large percentage of the data is missing due to lack of a monitoring satellite. In principle, that data could be masked off and we could still use some of it for training/testing. However, those observations were made with much older satellites so it's conceivable that there is a systematic offset in the values that they report. Additionally the dataset is large enough now that it takes a while to train on my laptop so I decided to just drop that data rather than to deal with the complexity of adding it. Future work could probably be done to add that data back in and see if results could be improved with it.

After I dropped all data before August 1995, I filled all missing values still present in the dataset with the mean value of the dataset. There were other methods that I considered (filling with the previously available value, filling with the next available value, interpolating between available values, removing them from the dataset entirely). Removing them from the dataset seems promising though it would probably force me to scrap some entire input sequences and the data is more likely to be bad during storms. Since storms are what is most interesting in this dataset and only 3.5% of the data is during storm time I didn't want to do that. Also, operationally, there might be missing data sometimes too and if this model were ever to become operational it would have to do *something* when the satellite data fails to get downloaded in time or is otherwise unavailable. Interpolation seems like an OK bet for short durations of missing values, but it wasn't clear to me how to define "short" (1 hour? 2 hours? 10 hours?). It also has problems when considering operational constraints (since the future value to interpolate to is unknown). Rather than deal with that complexity, I choose to assume again that the amount of missing data left would be insignificant enough that the model should be able to cope if I just replaced missing values with nominal (average) values.

Finally, I reduced the number of datapoints by a factor of 4 by resampling down to 20 minute intervals instead of using the dataset's 5 minute intervals. Physically, small temporal fluctuations in the IMF parameters shouldn't be enough to dramatically influence a global magnetic configuration so there isn't a compelling reason to avoid down-sampling. This also reduces the size of the input (and output) spaces which helps to avoid the curse of dimensionality.

I chose to use 5 hours (15 points per input variable) of data as the input space. The number of hours of data ahead of the prediction time is a free parameter that could easily be tuned in my implementation. Five hours seemed like a good amount of time based on my prior understanding of the problem. For example, global fluid simulations of the magnetosphere often require 3 to 5 hours of solar wind data input to set up a realistic magnetosphere which implies that the magnetosphere's memory may be near those timescales.

Implementation

The first thing that I had to do after exploring the dataset was figure out how to split it up into vectors for training and testing.

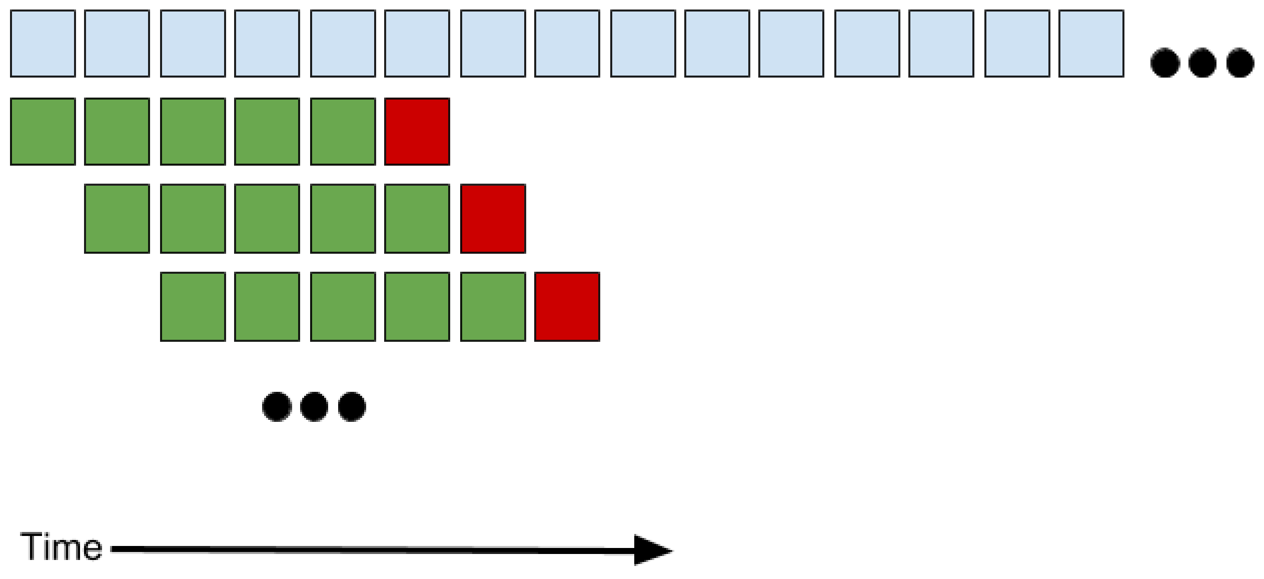


Figure 3: Representation of splitting the input time series. The blue boxes represent 1 hour of time in input series. The green boxes each represent 1 hour of training data copied from the corresponding data in the input time series. The red boxes represent 1 hour of Sym-H data from the input series at the corresponding time.

Figure 3 shows the sliding window method that was used for dataset splitting. Basically, I split my input data series of N points into `N - N_LOOK_BACK_POINTS - N_PREDICT` input series. Then I had to decide how to pick which sequences to use for testing and which to use for train. For non-RNN methods, I would have just randomly chosen some percentage of the vectors to be train data and the rest to be test data. However, that probably doesn't work so well here. Consider if the first and third green rows in Fig. 3 were chosen as training data and the second row was chosen as testing data. Since this is an RNN, it should notice if it sees something that it's seen before only shifted a little bit. Picking vectors that way would likely lead to overfitting that would be hard to diagnose with just the training and testing datasets. Indeed, I originally saw this in my validation dataset because I was letting Keras randomly pick the vectors to use for validation. Instead, I split my dataset (70% train, 30% test) before I generated the vectors to use. This guaranteed that there was no overlap between the test data and the training data. The split was chosen on a specific date (2010-07-29T19:00:00) because it was

70% of the way through the dataset. There are distinct advantages to picking it this way when thinking about the physical constraints of the problem:

1. There is more than 11 years of sequential data so we should get training data for an entire sunspot cycle (sunspots are correlated with geomagnetic activity)
2. Sequential data ensures that we do not introduce seasonal bias (due to the earth's dipole tilt)

The DNN that I am working with has the following simple format:

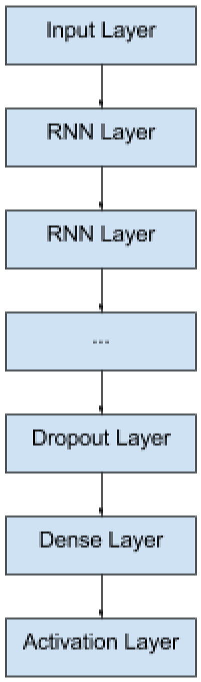


Figure 4: Basic structure of my neural network

This model has a number of free parameters that can be tuned. I'll enumerate some of them here:

1. The number of RNN layers. Using keras, we can be flexible about the number of RNN layers that we include.
2. The type of RNN layer (e.g. LSTM, GRU)
3. The number of units for the RNN layers.
4. The activation layer. There are a number of choices that we could use here ranging from "linear" to "relu" to no activation layer at all.
5. The value for the dropout layer.
    a. We can also apply the dropout to the RNN layers themselves instead of adding a dedicated layer as described in this article.
6. The size of the input and output layers.
    a. The input layers size is dependent on the sampling frequency (see pre-processing) and the number of hours of previous data that we want to

include. For example, if we sample the data at a 20min frequency and want to try to predict the next hour's value using the previous 5 hours' data, we'll need to use an input vector of size 15.

7. The optimizer. Keras documentation states that the RMSprop optimizer works well with RNNs and that the only parameter that should be tuned is the learning rate.

This model is adapted from a github gist by Dmitry Lukovkin.

The other thing that can be tuned that I haven't spoken much about is the input sequence choices. As mentioned in the data section, we have a lot of choices of sequences. I have done tests using the following data input sequences:

- B_z, DynamicPressure, clock_angle (angle between the magnetic field vector and the z-axis)
- B_z, DynamicPressure, B_y
- B_z, DynamicPressure, clock_angle, SymH

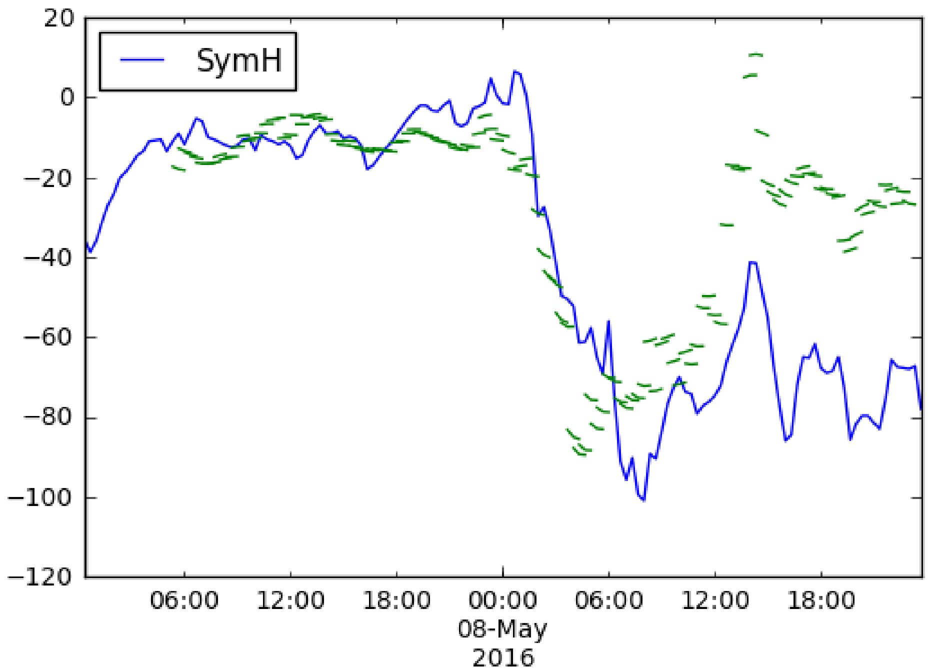


Figure 5: Predictions from my initial neural network for a geomagnetic storm without using Sym-H as an input. The blue line is the measured data, the green lines are hour-by-hour predictions that the model would have made in operation at that time.

Training with SymH generally leads to really good results, however it is not clear to me that it would be available operationally so I have chosen to leave it out of the dataset for subsequent analysis. After training for 10 epochs, My initial model's loss (Mean Square Error) was

209.8 nT^2. Can we do better?

Refinement

To answer this question, I trained 18 different models with slightly different parameters. Half of the models used B_y as input data and the other half used the IMF clock angle. On average, after training through 10 epochs, the models that used B_y had a lower MSE on the training data, however the loss on the validation data is very similar. This might be because information is lost when going from B_y, B_z -> clock angle, B_z. That additional information might be more than the model needs and is causing some overfitting.

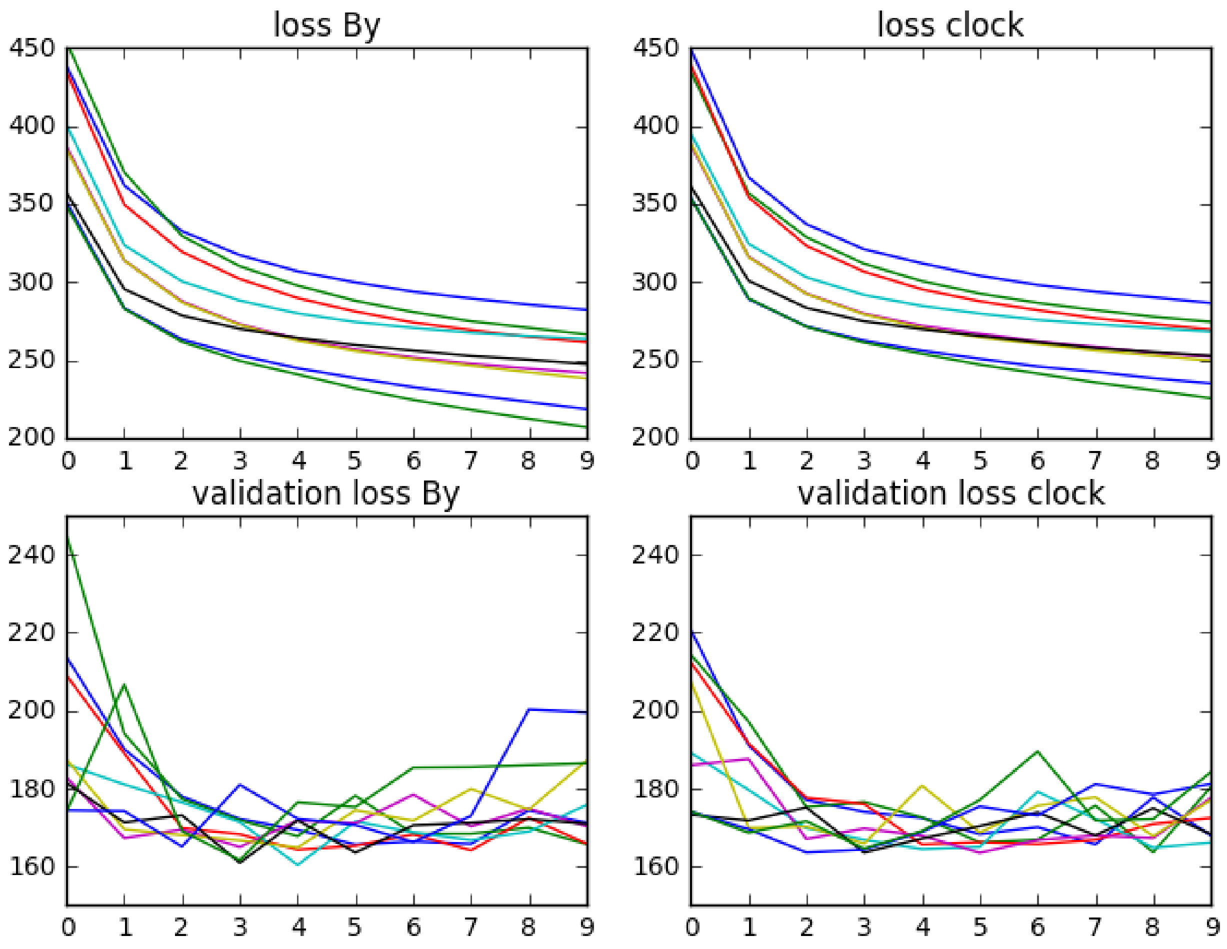


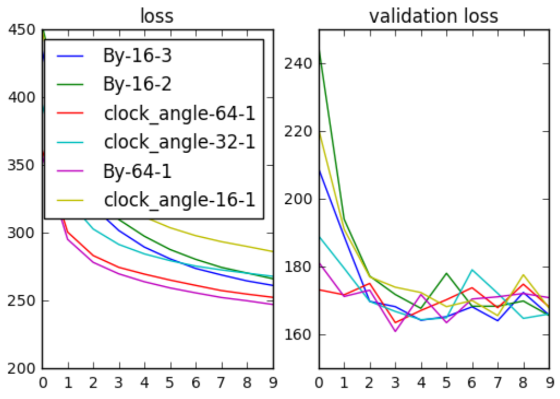
Figure 6: Loss function (MSE) vs. epoch number. The top left panels show the average loss function for the dataset with B_y and clock angle respectively. The different lines represent different model parameterizations. The bottom panels show the loss function for the B_y and clock angle models at the end of each epoch when evaluating the validation dataset.

I also varied the number of LSTM layers from 1 to 3 and the size of the LSTM layers from 16 to 64. Below is a table of the MSE on the test dataset for the top 6 runs:

Optional Data Source	LSTM size	LSTM layer count	Mean Square Error
By	16	3	184
By	16	2	185
clock_angle	64	1	185
clock_angle	32	1	185
By	64	1	185
clock_angle	16	1	187

Looking at the data, we can see that some of the simpler models (smaller LSTMs and/or fewer LSTM layers) performed really well. It is probable that these simple models did a better job of avoiding overfitting. We can see this by comparing the loss/validation loss of the 6 best models with the same quantities for the worst models:

Best models



Worst models

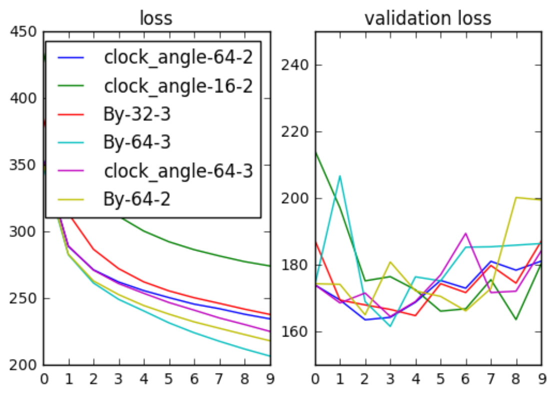


Figure 7: Losses and validation losses for the 6 best and 6 worst performing models on the test dataset.

There are much larger oscillations in the validation losses for the more complex models which indicates that the models are trying to overfit the data.

In contrast, the validation and test error of the top 6 models are almost identical.

Results

Justification

In order to compare my results with what others have done previously, I searched the literature for others who have worked on similar problems. One paper that is very similar is a paper called “Prediction of SYM-H index by NARX neural network from IMF and solar wind data” by Lei et al, 2009. In their paper, they use 2 hours lead-time in their input vectors (as opposed to my 5 hours) and they trained and validated only on storm data instead of the entire dataset. They had MSE values between about 100 and over 500 for different events.

While DST and SymH are distinct indices, the primary difference between them is that DST is reported hourly and SymH is available on a much higher time cadence. This means that DST forecasting algorithms can be directly compared with SymH forecasting algorithms. O’Brien and McPherron [1999] showed the comparisons of a number of different empirical DST prediction models. These predictions include both storm and quiet times. They reported the error results as a distribution of absolute errors:

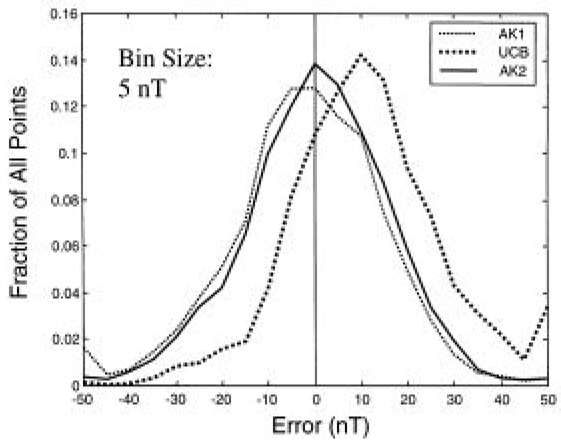


Figure 8: Error frequency in prediction from empirical models from O’Brien and McPherron [1999]

Doing the same thing for our 6 best models, we see that we get a sharper peak around the 0 nT bin and narrower distribution in general. This indicates that any one of our top models is out-performing simple empirical models having less error on average.

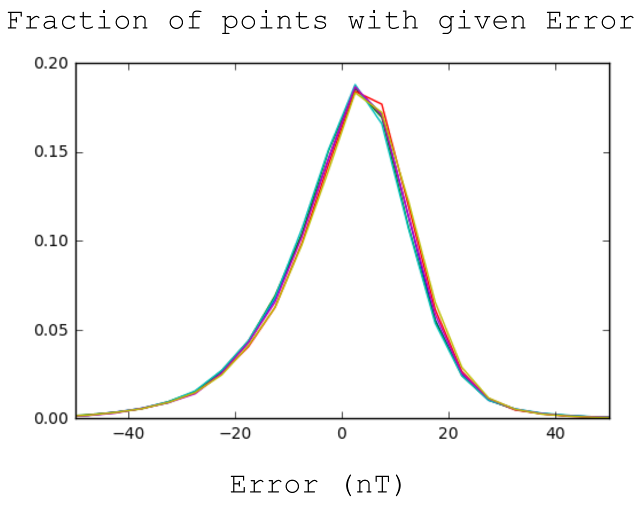


Figure 9: Error frequency in prediction from our LSTM based models on the testing dataset.

Additionally, the O’Brien and McPherron models require DST at the current timestep whereas our model only requires the solar wind inputs.

Conclusion

As seen in Figure 9, it is clear that our model is out-performing simple empirical models for overall prediction efficiency. We also do a good job predicting *some* storms.

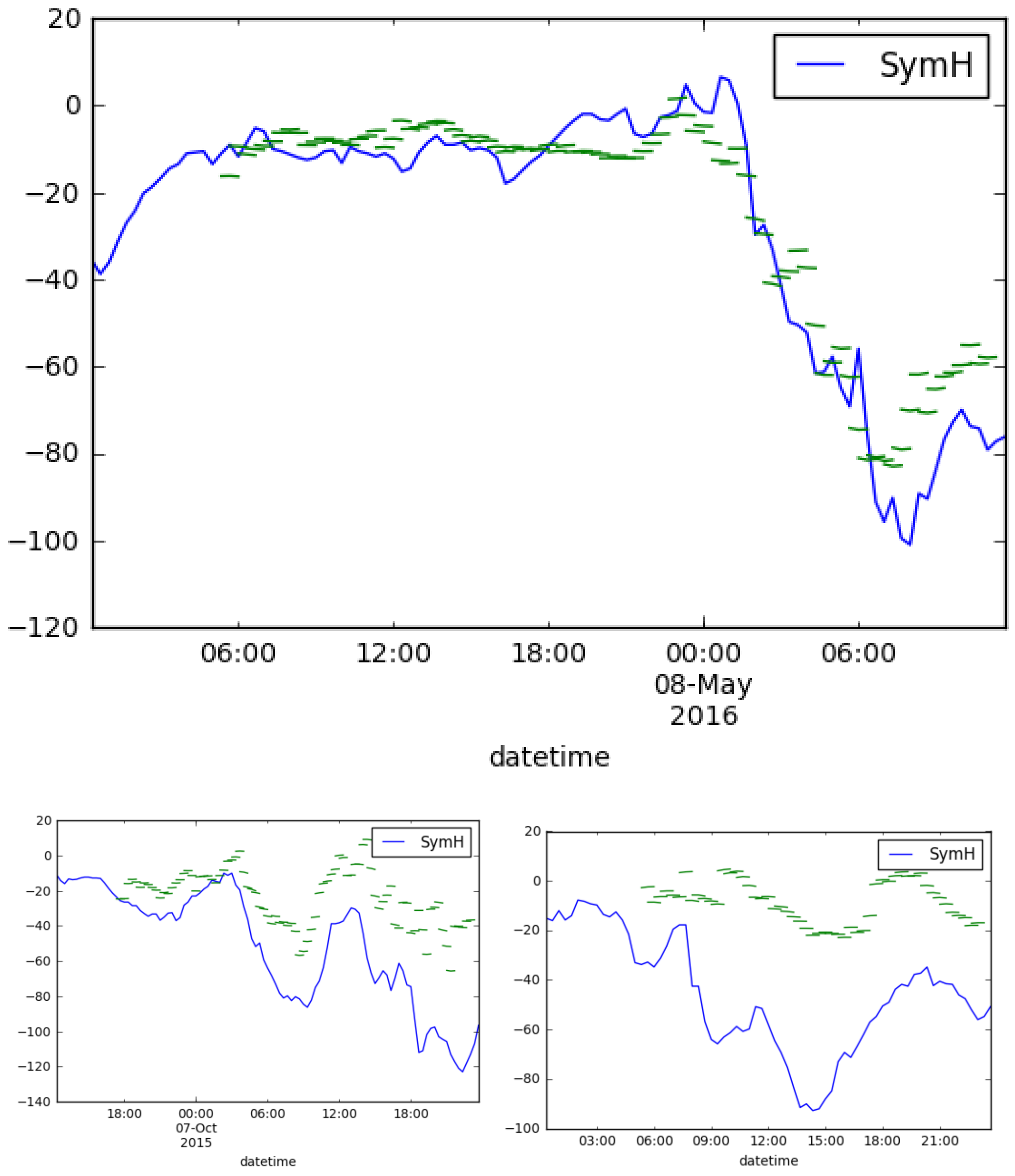


Figure 10: Predictions (green) overlayed onto actual data (blue) for 3 storms in the testing interval. In some events, we do a good job getting the large-scale trends in the data, but in some events we do a remarkably bad job. In all cases, the network tends to do “flat” predictions.

Reflection

This dataset proved to be very interesting to work on. At first, I was worried about the quantity of data and the unbalanced nature of the dataset with only 3.5% storm time representation. That doesn't seem to have been too problematic for our model however.

The other challenge that I faced while trying to make this prediction was to figure out how to pick a modelling strategy. At first, my research into time-series predictions made me want to try to train using an Autoregressive Integrated Moving Average (ARIMA) model. However, I am particularly interested in Deep Neural Networks (DNN) at the moment so I decided to keep looking for something that I might be able to solve using keras and/or Tensorflow. In the end, I remembered a talk that I had seen recently where the presenter used a character-by-character representation of Shakespeare to teach a computer to write a new play. It seemed that was not too different than what I wanted to do with my time-series representation of the data so I was able to use that as an inspiration for my network. I also found some discussions online about modelling stock market predictions using multivariate timeseries. Those code snippets were able to be borrowed to seed my network ideas. It's pretty cool how readily techniques transfer from one field to another with DNNs and machine learning algorithms in general.

To summarize the entire project, I first downloaded the OMNI dataset and removed much of the missing data by truncating the dataset prior to August 1995. Next, I replaced any missing data values with the mean values of the dataset. I split the remaining dataset into training and test datasets (70% train, 30% test) and I hypertuned the parameters for a RNN based neural network. The parameters that I tuned were:

1. Data input sources
2. Number of units in the RNN layers
3. Number of RNN layers
4. Type of RNN layers

In the end, it seems that simple networks (fewer units, fewer RNN layers) tended to work slightly better because they avoided overfitting. The RNN layer type (LSTMs and GRUs) didn't make much of a difference though LSTMs might have had a slight edge. The data input sources that I tuned against didn't make a whole lot of a difference either. This probably indicates that only the Z component of the magnetic field makes a huge impact on the SymH index. This could be further tested by not training against clock-angle or B_y at all. Finally, training with historical SymH values also made a significant difference in the overall model performance but that avenue wasn't explored further because there are probably operational difficulties with getting real-time SymH.

Improvement

Due to limited computational power, my tuning of hyperparameters was limited. I could have tried other network configurations. For example, I could try more complex models and/or I could have added dropout on the RNN layers themselves rather than (or in addition to) using a completely separate dropout layer. Supposedly this can help decrease overfitting.

I also could have done better validation if I had more time/computational power. For example, with my current setup, I used the same data for training and validation for each of the models. A more accurate approach would have been to train multiple identical models with the training/validation data split up differently. The averages of the losses of those models would have given me a better understanding of the model's overall performance rather than needing to rely on the validation provided by only a single train/validate dataset.

It would also be really interesting to tune the input state vector length. For example, would it be better to look at the last 7 hours when trying to predict the next 1 hour? Would a shorter duration like 3 hours work better? In principle, these are probably questions that could be explored independent of the network design to gain insight on the actual dataset.

It is also reasonable to expect that a fill-forward strategy to handle missing data could potentially do better than our fill-with-mean strategy while still maintaining operational viability. It is expected that filling with recent values would probably be closer to what is seen in the solar wind than filling with mean values (particularly during interesting storm intervals which are definitely not average).

One more preprocessing step that could produce better results is re-scaling the data so that the total input dataset has 0 mean and unit variance. Supposedly that can help some networks avoid numerical roundoff errors.

More data is always nice -- I removed a lot of the data from the dataset because it was slightly more complicated to deal with due to the high occurrence of missing values. This isn't ideal since more data is always better. Exploring that data for systematic offsets and then including it in the overall dataset would likely help to decrease overfitting as well.

I might have done better if I only tried to predict a single point 1 hour from "now" rather than predicting 3 points (20 min, 40 min and 1 hour from "now"). In the end, it seemed that the models ended up making "static" (flat) predictions anyway so we didn't really gain much from trying to break up the prediction into a few different values.

Finally, if SymH were operationally available, using it in my simple networks performed *really* well compared to the other models dropping the MSE to about 6 nT^2 (compared to ~185 nT^2 without it). For example, the event that did really poorly in the other models is shown in figure 12.

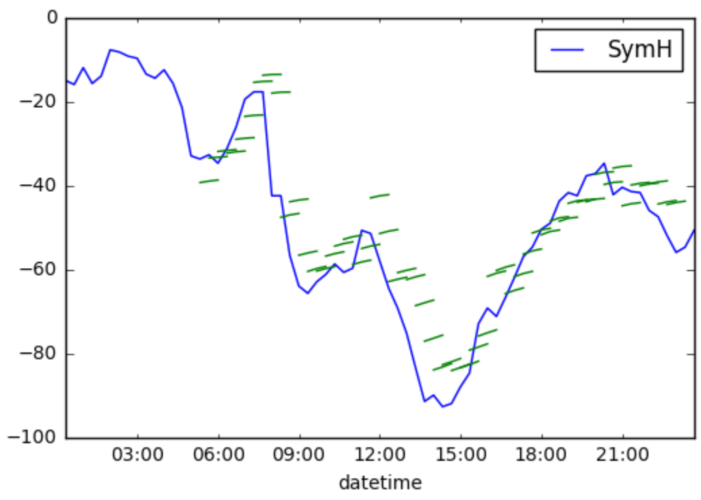


Figure 12: Adding historical SymH as input data to the model even gives reasonable performance to the events that we failed to predict well in the other models.