David Kang and Sally Qizhen He
6/3/2013

Neural Network Analysis on the MISO Electricity Market

Introduction:

With this project, we sought to explore the application of neural network analysis to a complex dataset. Focusing on the MISO electricity market, our goal was to forecast future price changes based on a variety of current-time inputs – including current day prices, relationships between locations, and time. Our analysis took several stages. In making our initial data exploration, we built a neural network in Python that produced sample results from a subset of the data at our disposal. To our surprise, our initial results looked promising. In fact, a basic trading scheme derived from our price forecasts could consistently generate profits in this market.

At this stage, we re-implemented our program in C++ to improve on the performance from our prototype. We were then able to apply our program to the entire dataset. We evaluated the results based on the price forecast accuracy as well as the associated profit/loss derived from our trading strategy. Overall, our program showed an ability to reliably produce profits based on our forecasts. Lastly, we ran a parameter sweep on our neural network model, harassing AWS parallelization to collect a wide range of data on the choices in model specification.

Goals:

The aim of this project was to apply neural network analysis to a large dataset. The use of neural networks is best applicable when the underlying dataset is difficult to model and the relationships between the inputs are highly unclear. For this reason, we chose to look at the MISO electricity dataset. We assumed very little background in the knowledge of electricity markets, and our goal was simply to evaluate the effectiveness of a custom-build neural network in making price forecasts.

In particular, we were interested in exploring the following questions: In what ways, if any, are the government's forecasts systematically biased? What are potential relationships between the prices at different locations, across different hours, or between different days of the week?

Dataset Description:

The Midwest Independent Transmission System Operator (MISO) electricity market serves the Midwest and southern region of Canada. In this market, a government backed transmission provider (MISO) establishes the electricity price for each power generation site (energy node) for each hour of the day based on a variety of economic factors.

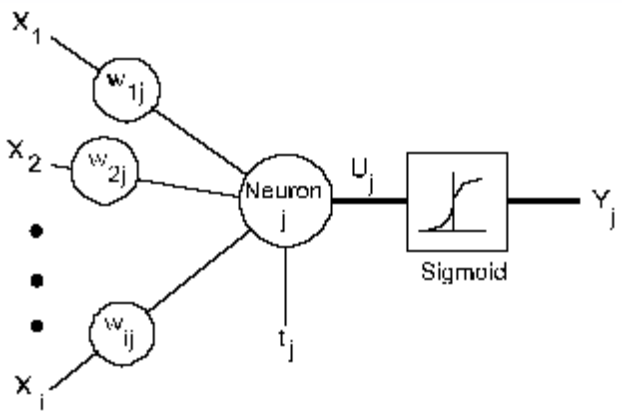
For our analysis, there are two different relevant prices at each location: the *real time price* and the *day ahead price*. The real time price at each location is simply a function of market supply and demand. Market participants bid on the current price at a particular location to establish price, much like the manner in which financial products are traded at a typical exchange. The day ahead price is the government's forecast price for the following day (which is given for each location at each hour). This allows market participants (energy providers and users) to lock in prices for the next day. Both the historical real time prices and day ahead prices are provided in csv files on the MISO website.

Individuals can trade the market in the following way. At any given location for a particular hour, one can buy electricity at the day ahead price with the hope that the actual, real time price the next day will be higher. If this prediction materializes, he or she can then sell the electricity back to the market at the higher price, thus generating a profit. Likewise, if one thinks the day ahead price is too high, he or she can sell electricity at the day ahead price with the hope of buying it back the next day for cheaper. In either case, the goal is to perform a financial arbitrage between the government forecast and the real time price. Indeed, this is the trading scheme we employed in producing our results further below.

* We did make a simplifying assumption. In our model, the neural network is given access to the day ahead prices and is able to make a forecast prior to forming a trade. However, in practice, one does not know the day ahead price until after he or she has placed the trade. However, an individual is allowed to set strict conditions (e.g. buy electricity at the day ahead price if and only if the price is between x and y). Thus, while our model cannot be directly applied to trade the market, it is also not far off. *

Background on Neural Networks:

Abstractly, artificial neural networks are an attempt to mimic the computational process of the human brain. In the brain, thoughts are processed through a network of neurons governed by activation triggers from synapses. In an artificial neural network, inputs are fed in parallel through 'nodes' that transform these inputs into outputs. The neural network 'learns' by making mistakes and adjusting its nodes in step with the magnitude of the error.



More specifically, our neural network uses the following implementation. The neural network accepts an input vector of floats. Each node in the hidden layer then takes a weighted sum of the components in this vector (the weights are initially set randomly), and transforms the result under the nonlinear logistic function. Finally, at each output node, the return value from each node is then combined into a new

weighted sum that represents an output vector. This entire process described above is known as the feedforward loop.

To train the network, this output vector can be compared to a target output. Through a learning algorithm known as error backpropagation, the weights at each node are then adjusted based on the extent of the error and as well as a user-defined learning rate. For the output layer, the change to each weight is simply the output error multiplied by the weight and the learning rate. However, for the hidden layer, we must also calculate the error associated with each node since there are no target outputs for this layer. Once we have done so, we once again apply the weight adjustment that we used in the output layer.

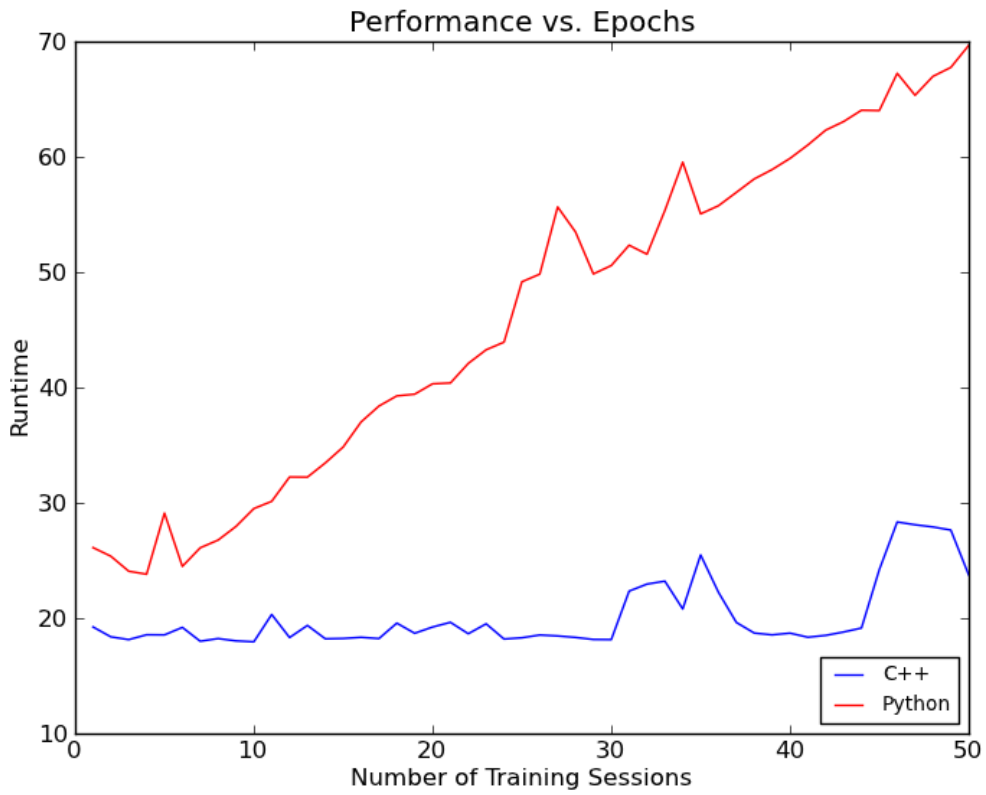
Implementation:

Our Python prototype is divided into two components: datautil.py and neuralnet.py. Neuralnet.py is essentially the code implementation of the neural network model described above. We took an object-oriented approach in which nodes and networks each represented their own class. The datautil.py module is responsible for processing the data from the raw MISO csv files and producing readable output and visualization. The dataset class transforms csv data into a list format readable by the neural network, and stores this in memory.

The output class assumes two roles. First, it calculates profit/loss based on the price forecasts from the neural network, e.g. if the forecast price is higher than the day ahead (input price), the model purchases the day ahead price and sells the next day real time price (and calculates the associated profit/loss). Second, the output class processes this data back into a csv file and graphs the profit/loss results for each energy node. We did not use advanced Python libraries – we relied primarily on matplotlib, scripy, and built-in python functions.

In our second implementation, this code was reprogrammed into the language of C++, since we were interested in the performance gains. Due to the added

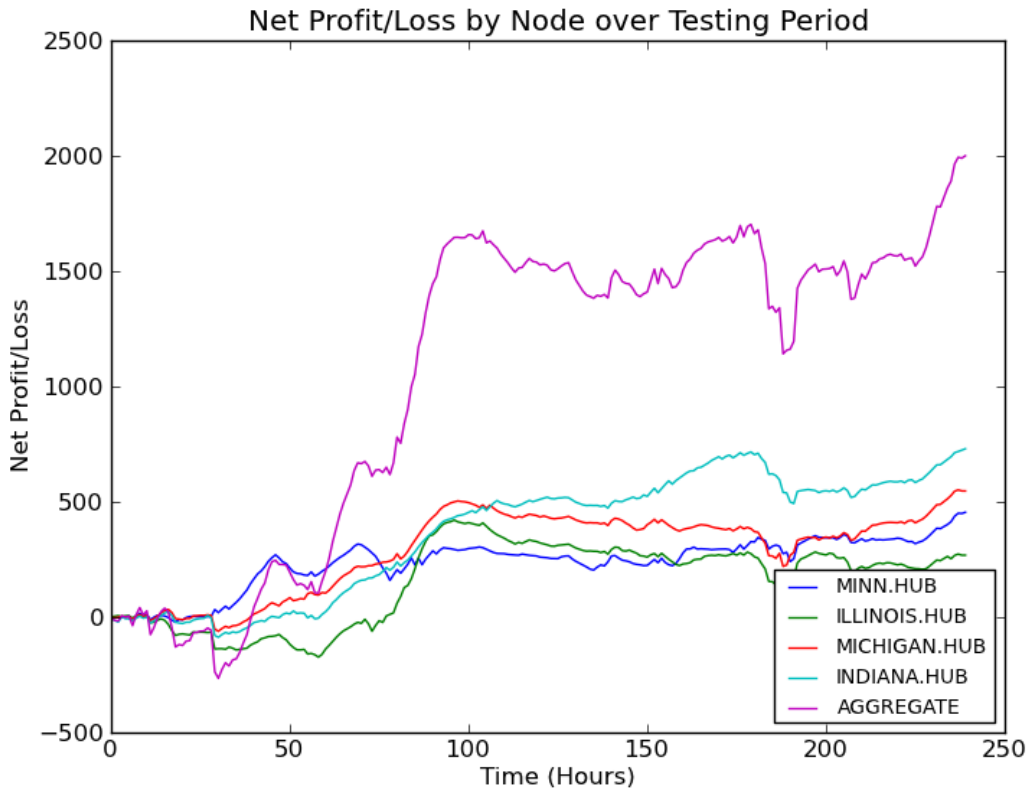
complexity of programming in C++, our code was divided into several more components than with our python implementation. However, the underlying architecture remained the same. Each class from neuralnet.py (e.g. hiddenNeuron, Network, etc.) was separated into its own header and cpp file, but the relationship between these classes is the same as it was in our earlier implementation. Datautil.py was also divided into several components, but its role remains the same. Unfortunately, C++ does not have the same visualization libraries as Python. As a result, we deferred the visualization of our output back to Python in the script visualization.py.



We tested the C++ implementation against the Python prototype on the sample data (as we varied the number of training sessions). As we see, the C++ implementation performed at linear time (it performed each training session so quickly the changes were undetectable) throughout the test, while the Python implementation varies linearly with the epoch count. The main overhead with the C++ version was in the data processing data.

Experiment and Analysis:

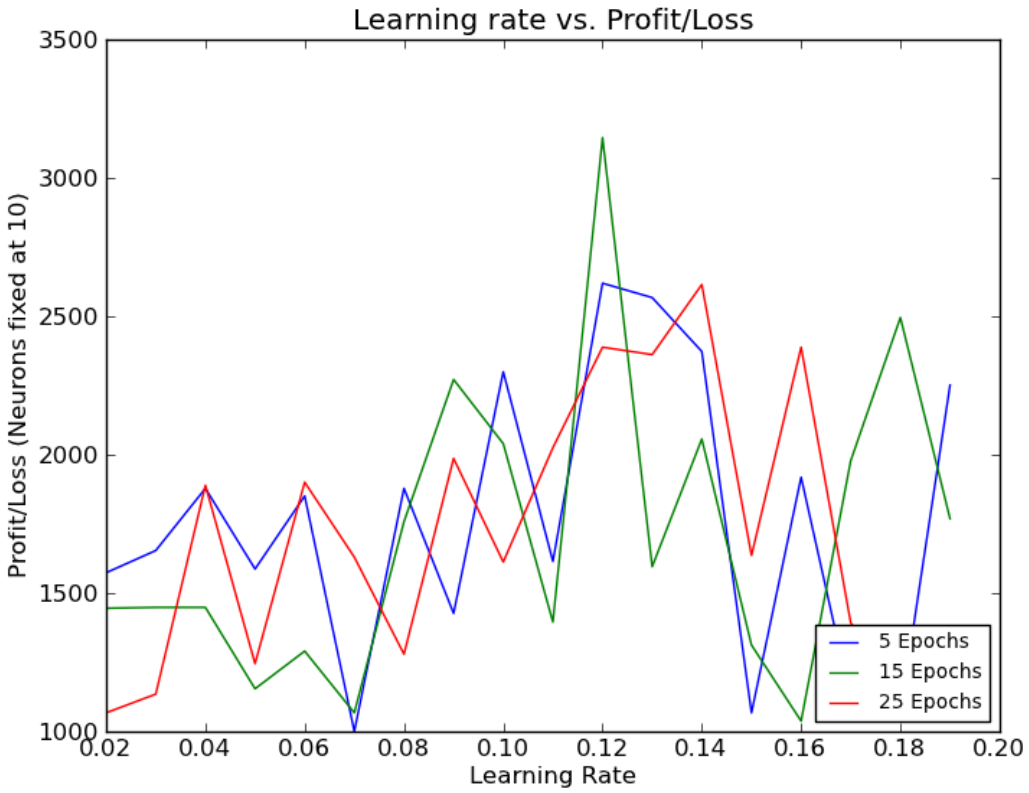
We divided the MISO data into training and testing sets. The most recent data (5/1/13 to 5/29/13) was used as testing data while the neural network was trained using the historical data prior to May. Here is the graph showing our profit/loss results by location (neurons: 10, training sessions: 10, learning rate: .05):



Our results show that trades made based on the price forecasts for the month of May generated $2000.35 in profit. As it turned out, we were able to achieve a profit almost regardless of the initial parameters we specified (such as learning rate, number of neurons, number of training sessions, etc.) This suggests the existence of significant mispricing in the government model that has remained largely unexploited by market participants.

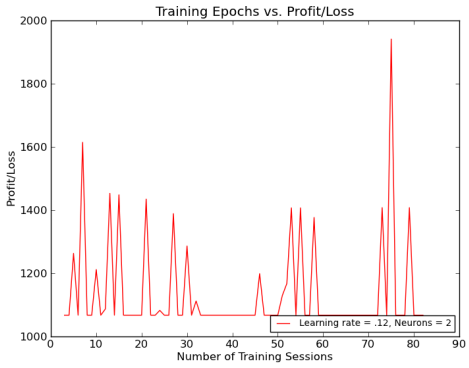
This finding was surprising. Our initial hypothesis was that the electricity market would be highly efficient, and that any forecast gains made by the neural network would be small and fleeting. However, this does not appear to be the case. Our mean squared error (in the comparison between our output prices and the actual, real time prices) of 0.558 shows that our neural network predictions are largely correct.

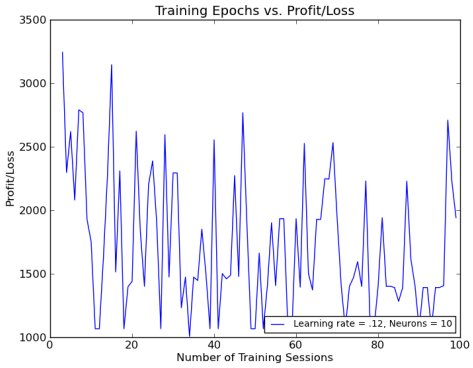
Although we were able to generate profits almost regardless of parameter choices, the amount of profit could vary considerably based on such choices. As a result, we sought to run a parameter sweep to identify factors that might strengthen the model. For this portion of our analysis, we utilized parallelization with AWS and aggregated the results later. Here are some of our findings:



In the above sweep, we held the number of neurons fixed and varied the learning rate. It appears the optimal learning rate hovers around 0.12. Based on our study of the literature, we expected the graph to take this shape. Generally, a very high

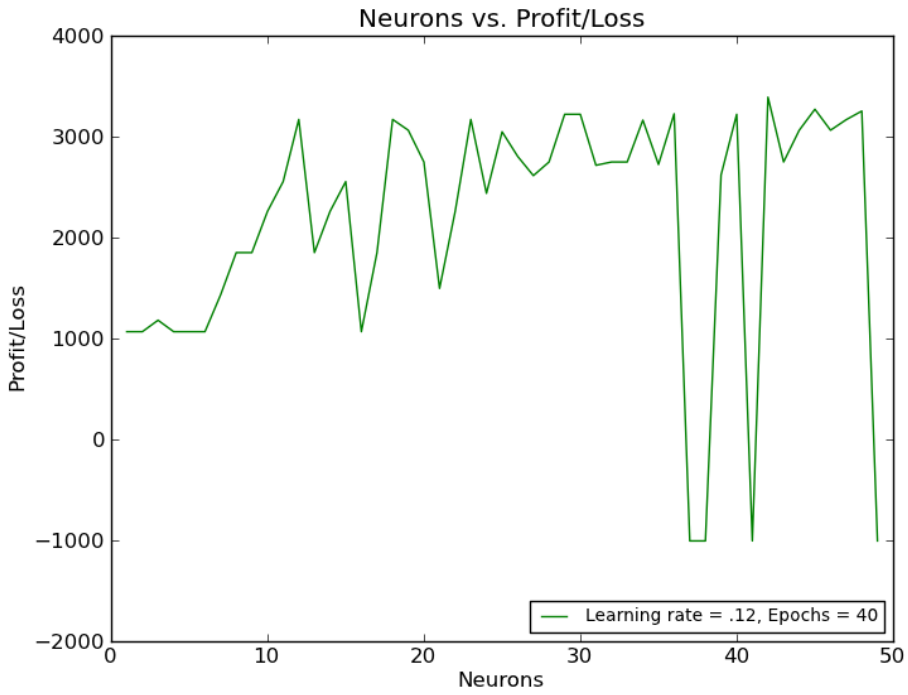
learning rate can inhibit the ability for the network to converge to a solution, while a very low learning rate can slow the learning process to a crawl. As a result, there should be an optimal learning rate somewhere in between, which our parameter sweep appears to have identified as 0.12.





In this experiment, we fixed the learning rate at the optimal level (0.12) while we varied the number of training sessions (epochs). The left graph shows the result when neurons = 2 and the right graph when neurons = 10. As one can see, the relationship appears to be random. This is counterintuitive – we expected the results to steadily improve as we increased the number of training sessions. It is unclear what causes the sudden spikes or dips in the profit/loss.

Finally, we varied the number of neurons and held the other variables fixed. For small values 1 – 15, increasing the number of neurons offered a marked improvement in results. However, the gains quickly level off. In fact, at very high neuron counts (>35), the neural network can lose stability and can occasionally slip into trading loss of -$1000. This is intuitive because including a larger number of neurons introduces more degrees of freedom, and it becomes harder for the neural network to converge to a solution.



Conclusion:

Overall, our neural network was able to outperform the government price forecasts and produce a hypothetical trading edge in the market. However, the primary drawback to neural network analysis is the difficulty in backtracking its underlying logic. That is, it is extremely challenging to understand why our neural network produces the predictions that it does.

However, based on our choice of inputs for the model, we have collected some insight that could warrant further investigation. For one, adding additional electricity nodes tends to increase profits (but not reliably), which suggests that the neural network can exploit price relationships between the various locations. Moreover, when we add the date associated with each price to the input vector, profits generally rise. This suggests that time-related information (e.g. seasonality, day of the week, etc.) would likely improve the price forecasts. These are all topics to consider in a further analysis.