Prediction of Air Pollution using Neural Networks

Nicholas P. Hirning nhirning@stanford.edu
Andy S. Chen asjchen@stanford.edu
Stanford University

Abstract

This study attempted to use simple neural network models (feed-forward and Elman recurrent) to predict pollutant concentrations. The neural net performance was better than linear regression when predicting more than several hours in the future, but not for the immediate future. Thus, neural networks show promise as components of prediction tools, but not necessarily as standalone algorithms.

I. Introduction

With the onset of globalization, industrialization, and rapid population growth, air pollution is increasingly becoming a universal issue. The World Health Organization found that from 2008 to 2013, air pollution rose 8% globally.[9] This increasing danger motivates the study of systems for predicting air pollutant concentrations ahead of time. Ideally, we would like to predict which pollutants will be at dangerous levels a significant number of hours in advance, so cities can better respond to the certain pollutants. In this prediction, we want to fit the general trend over the next several hours while minimizing the error at each hour. This project studies the application of neural networks for this objective.

II. Task Definition

To define the problem concretely, we used a data set of air pollution data taken from northern Taiwan in 2015.[4] This contained roughly 200,000 data points in total, split (roughly) equally among 25 different observation stations. Each data point represented the data for one hour of weather conditions and pollutant concentrations. The following variables were all recorded in the inital dataset.

1. Sulfur Dioxide (SO_2)
2. Carbon Monoxide (CO)
3. Ozone (O_3)
4. Particulate Matter $\leq 10\mu$m in diameter (PM10)
5. Particulate Matter $\leq 2.5\mu$m in diameter (PM2.5)
6. Nitrogen Oxides (NOx)
7. Nitric Oxide (NO)
8. Nitrogen Dioxide (NO_2)
9. Total Hydrocarbons (THC)
10. Non-Methane Hydrocarbons (NMHC)
11. Methane (CH_4)
12. Ultraviolet Index (UVB)
13. Ambient Temperature
14. Rainfall
15. Relative Humidity (RH)
16. Wind Speed (hour average and last 10 minutes average)
17. Wind Direction (hour average and last 10 minutes average)
18. Rain PH
19. Rain Conductivity

However, due to missing data in some observation stations, we removed variables $9, 10, 11, 12, 18, 19$. This resulted in around $60,000$ usable data points. We then partitioned the data into three sets: training (roughly 60%), validation (roughly 20%), and test (roughly 20%). Then, explicitly, we aimed to develop a model that processes several hours of past data at a given time t and outputs the concentrations of the 8 pollutants (variables 1 through 8) in the next x hours of pollutant concentrations (corresponding to $t+1, t+2, ..., t+x$), where $1 \leq x \leq 24$ was a reasonable bound for x.

In order to test the difficulty of this task and provide initial error bounds for later comparison, we implemented simple baseline and oracle algorithms. The baseline algorithm was linear regression on the past n hours to predict the next hour's pollutant concentrations. This could be repeated x times to predict the next x hours of concentrations (where each consequent prediction relied on the accuracy of the previous prediction). Though this approach worked quite well for small x and small n (local approximation of the next data point), this yielded invariably bad results for nontrivial values of x. An example of this is illustrated below.

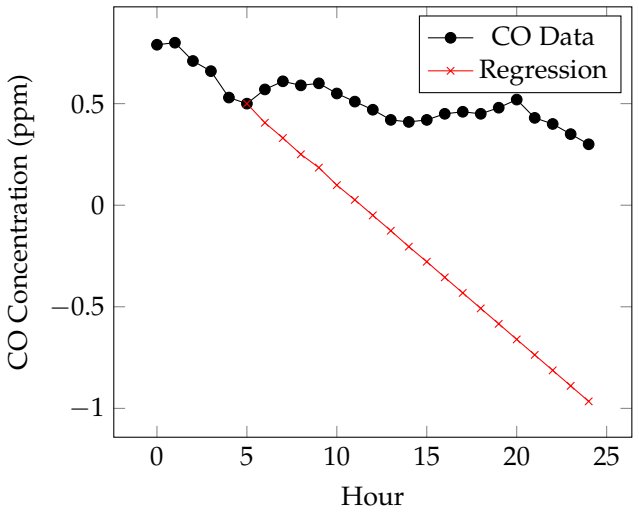

Figure 1: *Linear regression prediction for* 20 *hours ahead with* 5 *hours of history*

The oracle algorithm provided a lower bound for the error on any algorithm. The oracle predicted the level of a pollutant at an hour t by using quadratic regression on the n hours before x and the n hours after x. To "predict" the next x hours, we repeated this regression for each of those x hours. In the section on loss functions we give several tables on the error of the baseline and oracle algorithms.

III. Literature Review

There have been a number of papers published relating to predictive models for pollutants. A paper by Pérez et al used a three-layer neural network with a logistic activation function to predict the concentrations of PM2.5 in the next 24 hours.[6] The inputs were the previous 24 hours of data (on only PM2.5). Predictions were improved by using noise reduction, data rearrangement, and explicit consideration of meteorological variables. The resulting errors ranged from 30% to 60% depending on how far in the future the predictions were made. Overall, the neural network outperformed linear regression. Another paper, by Abdul-Wahab et al., analyzed prediction of ozone concentrations using neural networks.[7] The inputs to the neural net included other pollutants and weather data, and the paper concluded that (in the trained neural net) around 35% of the variation was due to meteorology and 65% due to the levels of other pollutants. The work by Pérez et al. similarly found limited improvement by incorporating meteorological variables.

A study by Atakan et al. attempted to predict air pollution indicator levels (1, 2, or 3) for three different pollutants using neural networks (inputs are the previous concentrations).[1] The indicator levels for days further in the future were predicted in several different ways, but cumulatively using predictions for the ith day to predict the $(i+1)$-th day was the most effective. The final model used 3 to 15 past days as input and also used the day of the week as an input parameter. An older article by De Vito et al. uses two weeks of on-field data recording and neural regression systems.[3]

There is another class of studies which utilize neuro-fuzzy networks. A paper by Nunnari et al. attempted to use these networks to predict pollution levels, but found that, as implemented in the paper, neuro-fuzzy predictors were less accurate than multilayer perceptrons.[5] Another paper by Aparna Soni applied similar ideas to urban areas in India.[8]

IV. Approach

We implemented two models to achieve the problem defined in the previous section: a three-layer feed-forward neural network and a three-layer Elman recurrent neural network (RNN). These were both implemented from scratch in Python (using solely the NumPy library). These models were relatively simple, but since most neural networks are more complex combinations of these structures, these should have been able to demonstrate the ability or inability of neural networks in predicting pollutant concentrations.

Due to the computational cost of training a neural network, we made one important simplification of the problem. Instead of trying to train separate neural networks for prediction of a variable number of hours in the future, we trained the neural networks to predict accurately just the next hour, and then we extrapolated to predict the next x hours by applying the neural network x times. If a neural net was trained to predict x hours in the future directly, it would undoubtedly be more accurate than this model. However, the model predicting only one hour in the future was more flexible; if one were to train a neural network for prediction of pollutants exactly 6 hours in the future and later it was necessary to predict pollution only 4 hours in the future, the only solution would be to train a new neural network. This approach allows one model to predict any number of hours in the future.

A. Feed-Forward Neural Network

Our basic feed-forward neural network had three layers, where each layer was an operation on the output data of the previous layer. Mathematically, if the input vector was x, then the operations were defined as follows:

$$z_1 = W_1 \cdot x + b_1 \tag{1}$$
$$z_2 = F(z_1) \tag{2}$$
$$z_3 = W_2 \cdot z_2 + b_2 \tag{3}$$

where F was the activation function of the neural network (which was designed to add nonlinearity to the data). We visualized these operations by defining three layers: input, hidden, and output, in that order. The vector z_1 was the input to the hidden layer, the vector z_2 was the output of the hidden layer, and the vector z_3 was the input to the output layer (equivalently, z_3 was the output of the network). If $x \in \mathbb{R}^n$ (corresponded to having n features) and we chose $z_2 \in \mathbb{R}^d$ (this is also stated by saying that there were d hidden neurons or d was the dimension of the hidden layer), then $W_1 \in \mathbb{R}^{d \times n}$, $b_1 \in \mathbb{R}^d$, $W_2 \in \mathbb{R}^{p \times d}$, and $b_2 \in \mathbb{R}^p$ (where p was the dimension of the output vector). The parameters W_1, W_2, b_1, b_2 were determined via training. Visually, these operations and layers are depicted in the following figure.

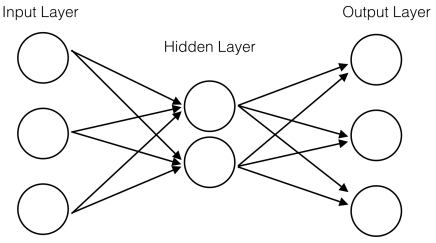

Figure 2: *Visual depiction of a three-layer feed-forward neural network*

In the case of time-series prediction, a standard method was to use a sliding-window approach. This involved predicting one hour of pollutant data at a time, using the previous k hours as input to the neural network (thus we were sliding a "window" of width k along the data to make predictions — hence the name). Specifically, we concatenated the data from the previous k hours (15 features per hour) into one vector of length $15k$, which we input directly into the network. The output was then the prediction of the 8 pollutants for this hour, represented as a vector in $\mathbb{R}^8$. We yielded a prediction for the next x hours into the future by repeating this approach x times, so for this network structure, we simplified this problem to predicting just the next hour. The hyperparameter k must be determined by cross-validation, along with d, the dimension of the hidden layer. There were also several different options for the activation function F, which contributed to the extensive parameter determination necessary for optimal performance.

B. Elman Recurrent Neural Network

One drawback to using feed-forward neural networks was that there was no notion of current state in the model. That is, every data point was treated separately from the previous ones, even though meaningful information may have been extracted at the last time step. Our recurrent neural networks solved this by passing information from the $(t-1)$-th timestep to the t-th timestep. In particular, we chose to implement an Elman recurrent neural network, which was very similar to a three-layer feed-forward neural network except the output of the hidden layer at timestep $t-1$ was used as input to the hidden layer at timestep t. Thus, we could maintain an internal. The following equations governed the Elman neural network:

$$z_{1,t} = W_1 \cdot x_t + b_1 + U \cdot z_{2,t-1} \tag{4}$$
$$z_{2,t} = F(z_{1,t}) \tag{5}$$
$$z_{3,t} = W_2 \cdot z_{2,t} + b_2 \tag{6}$$

Similar to before, at timestep t, x_t was the input vector, $z_{1,t}$ was the input to the hidden layer, $z_{2,t}$ was the output of the hidden layer, and $z_{3,t}$ was the output of the network. In this case, we had to train an additional parameter, U, which intuitively weighted the contribution of the output of the hidden layer in the previous timestep to the input to the hidden layer in the current timestep.

The input was identical to the feed-forward network, but the results of hyperparameter optimization for k, d (the number of past hours used as input and the dimension of the hidden layer) may have been different due to the slightly modified structure. Visually, the Elman recurrent network was similar to the three-layer feed-forward network:

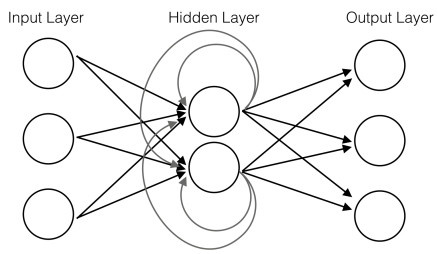

Figure 3: *Visual depiction of an Elman recurrent neural network*

V. Loss and Activation Functions

Determining the appropriate loss function was crucial to generating an accurate predictive model. In the case of this problem, there were a number of limitations that led to two loss functions in particular.

Much of the data were in units that were incompatible with each other, so an absolute difference (or standard L2 loss) would not suffice. It was natural to use percentage error instead, but the low concentration of some pollutants in certain hours led to arbitrarily large losses (and in some cases, infinite loss when the concentration was zero). Instead, we used a percentage loss where the percentage was taken off the average value of a given pollutant. Thus, if the pollutant concentrations in some hour t were enumerated $\{p_{1,t}, ..., p_{8,t}\}$, $\hat{p}_{i,t}$ was the predicted concentration of $p_{i,t}$, and $\overline{p}_i$ was the average value of $\{p_i\}$ over all hours contained in the training data set D. Then we defined the training loss as

$$\text{TrainLoss}_1 = \sum_{t \in D} \sum_{i=1}^{n} \frac{1}{\overline{p}_i} |p_{i,t} - \hat{p}_{i,t}|. \tag{7}$$

where n was the number of pollutants. This could be interpreted as a modified L1 loss, and this naturally suggested the modified L2 loss:

$$\text{TrainLoss}_2 = \sum_{t \in D} \sum_{i=1}^{n} \frac{1}{\overline{p}_i} (p_{i,t} - \hat{p}_{i,t})^2. \tag{8}$$

From now on, we will refer to these as L1 and L2 loss, respectively. Note that these two loss functions were naturally more robust to dangerous levels than standard percentage error losses in the sense that when actual concentrations were significantly above the global average, the error function was proportionally larger for similar percentage error in predictions than it would be if the actual concentrations were average. Under these two loss functions, we could calculate the losses for our baseline and oracle algorithms. The following table provides some errors on the training set for baseline linear regression for different values of x (hours in the future to predict) and n (past hours to use for prediction) using TrainLoss_1.

n	$x = 1$	$x = 6$	$x = 12$	$x = 24$
2	2.269	7.356	13.136	23.833
3	2.367	6.596	11.218	19.725
6	2.877	6.126	9.404	15.421
12	3.379	5.966	8.093	11.847
24	3.850	4.973	5.915	7.221

Figure 4: *Linear regression errors on the training set using different values for n (number of past hours) and predicting x hours ahead*

These errors were reasonable for low x, but increase rapidly with x. For the oracle, to "predict" the next x hours, we ran regression for each individual hour and averaged the resulting errors. The following table summarizes these errors:

n	$x = 1$	$x = 6$	$x = 12$	$x = 24$
2	1.238	1.188	1.089	1.150
3	1.240	1.196	1.144	1.226
6	1.701	1.548	1.727	1.737
12	2.841	2.964	2.868	2.634
24	3.043	3.053	3.097	3.142

Figure 5: *Oracle errors on the training set using different values for n (number of past hours) and predicting x hours ahead*

Once the neural networks were trained on these loss functions, they could be evaluated relative to the baseline and oracle algorithms using these two loss functions. To train the neural networks on these loss functions, we used stochastic gradient descent. For the feed-forward network and these loss functions, closed-form calculations of the gradients were simple. Under $L = \text{TrainLoss}_1$ (with $n, W_1, W_2, b_1, b_2, z_1, z_2$ defined as in equations (1) to (3)), the gradients for timestep t were

$$\frac{\partial L}{\partial W_{1,j,k}} = \sum_{t \in D} \sum_{i=1}^{n} \frac{1}{\overline{p}_i} \text{sgn}(\hat{p}_{i,t} - p_{i,t}) W_{2,i,j} (1 - z_{2,j}^2) x_k \quad (9)$$

$$\frac{\partial L}{\partial b_{1,j}} = \sum_{t \in D} \sum_{i=1}^{n} \frac{1}{\overline{p}_i} \text{sgn}(\hat{p}_{i,t} - p_{i,t}) W_{2,i,j} (1 - z_{2,j}^2) \quad (10)$$

$$\frac{\partial L}{\partial W_{2,j,k}} = \sum_{t \in D} \frac{1}{\overline{p}_j} \text{sgn}(\hat{p}_{j,t} - p_{j,t}) z_{2,k} \quad (11)$$

$$\frac{\partial L}{\partial b_{2,j}} = \sum_{t \in D} \frac{1}{\overline{p}_j} \text{sgn}(\hat{p}_{j,t} - p_{j,t}), \quad (12)$$

where $\text{sgn}(x)$ denotes the sign of x. For the recurrent neural network, we calculated the gradients via backpropagation. For example, again under $L = \text{TrainLoss}_1$ with definitions as above, the recursive equations for the gradients of b_1, W_1 were as follows:

$$\frac{\partial L}{\partial W_{1,j,k}} = \sum_{t \in D} \sum_{i=1}^{n} \frac{1}{\overline{p}_i} \text{sgn}(\hat{p}_{i,t} - p_{i,t}) W_{2,i,j} \frac{\partial z_{2,t,j}}{\partial W_{1,j,k}} \quad (13)$$

$$\frac{\partial z_{2,t,j}}{\partial W_{1,j,k}} = (1 - z_{2,t,j}^2) \left(x_{t,k} + U_{j,j} \frac{\partial z_{2,t-1,j}}{\partial W_{1,j,k}} \right) \quad (14)$$

$$\frac{\partial L}{\partial b_{1,j}} = \sum_{t \in D} \sum_{i=1}^{n} \frac{1}{\overline{p}_i} \text{sgn}(\hat{p}_{i,t} - p_{i,t}) W_{2,i,j} \frac{\partial z_{2,t,j}}{\partial b_{1,j}} \quad (15)$$

$$\frac{\partial z_{2,t,j}}{\partial b_{1,j}} = (1 - z_{2,t,j}^2) \left(1 + U_{j,j} \frac{\partial z_{2,t-1,j}}{\partial b_{1,j}} \right). \quad (16)$$

The backpropagation for W_2, b_2, U were similar. Choosing the activation function for our neural networks was not a simple task. In general, activation functions of different shapes resulted in vastly different fits (e.g., a logistic activation function would result in a neural network that entirely encompasses logistic regression, but would not be able to fully perform basic linear regression). For our networks, we chose to examine three separate activation functions: hyperbolic tangent ($f(x) = \tanh(x)$), the linear rectifier ReLU ($f(x) = \max(0, x)$), and SoftPlus ($f(x) = \ln(1 + e^x)$). In particular, these three functions allowed comparison of bounded and unbounded, as well as linear and nonlinear activation functions. Graphs of these activation functions are shown below.

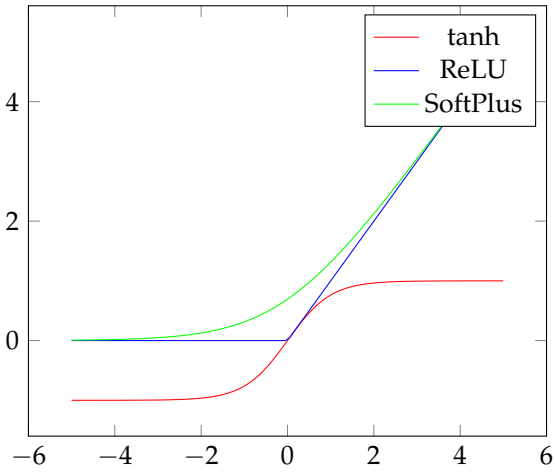

Figure 6: *Plots of three activation functions chosen for the neural networks*

VI. Hyperparameter Optimization

With the numerous different hyperparameters involved in the neural networks described, determination of the optimal values was very important for optimal performance. Specifically, there were different hyperparameters we had to vary: activation function F, number of hours k used for input, number of hidden neurons d, and loss function. For each choice of activation function, loss function, and neural net type, we collected

data on the errors for $k = 2, 3, 6, 12, 24$ and $d = 25, 50, 100, 200$. (We limited d to 200 for computational reasons.) To estimate the errors, we used two-fold cross-validation (with 20 full iterations through the training set), where the appropriate step size (for stochastic gradient descent) was based on several test runs on validation sets (the step sizes were all of the form $A/\sqrt{n}$, where n was the number of updates and A is determined empirically). For brevity, we only show two of these training tables.

k	$d = 25$	$d = 50$	$d = 100$	$d = 200$
2	4.464	4.435	4.353	3.967
3	4.454	4.157	3.750	3.194
6	4.445	3.993	3.970	3.363
12	4.420	4.397	4.156	3.788
24	4.429	4.146	4.443	4.084

Figure 7: *Cross-validation for feed-forward network with* tanh *activation function and L1 loss (k is the number of past hours to input and d is the number of hidden neurons)*

k	$d = 25$	$d = 50$	$d = 100$	$d = 200$
2	3.853	2.301	2.290	2.309
3	2.962	2.754	2.614	2.256
6	3.848	2.851	2.744	2.512
12	3.805	3.788	3.816	2.769
24	7.080	3.781	3.777	3.762

Figure 8: *Cross-validation for feed-forward network with Softplus activation function and L2 loss (k is the number of past hours to input and d is the number of hidden neurons)*

These optimal hyperparameters are summarized in the following table:

Network Description	k	d
Feed-Forward, tanh, L1	3	200
Feed-Forward, ReLU, L1	12	50
Feed-Forward, SoftPlus, L1	3	200
Feed-Forward, tanh, L2	3	50
Feed-Forward, ReLU, L2	3	50
Feed-Forward, SoftPlus, L2	3	200
Elman, tanh, L1	3	50
Elman, ReLU, L1	12	100
Elman, SoftPlus, L1	3	100
Elman, tanh, L2	3	50
Elman, ReLU, L2	3	100
Elman, SoftPlus, L2	3	100

Figure 9: *Optimal hyperparameters based on cross-validation results (k is the number of past hours to input and d is the number of hidden neurons)*

In general, it appeared that the optimal number of past hours to input was 3, while the optimal numbers of hidden neurons varied significantly from model to model. For reference, below is a plot of the learning curve of the feed-forward network with L1 loss and SoftPlus activation function with the optimal hyperparameters.

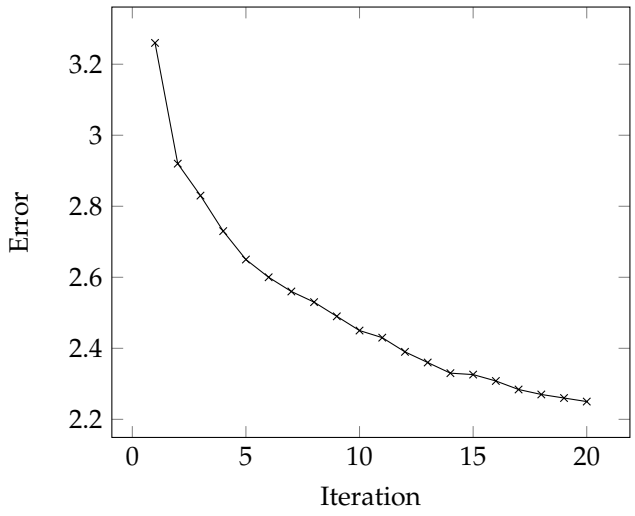

Figure 10: *Learning curve for 20 iterations of feed-foreard neural network with SoftPlus activation function and L1 loss*

Finally, to reduce overfitting we implemented early stopping and L2 regularization. Due to the time-intensive process of cross-validation, we only tested several different options for early stopping. In the end we settled on 20 iterations for the feed-forward network and 10 iterations for the Elman RNN. Regularization kept the parameters W_1, W_2, b_1, b_2, U from becoming too large by adding their norms to the loss function. We used the Frobenius norm and divided by the number of entries in the matrix to normalize for the differing parameter sizes. For example, for W_1 with the L1 loss function and regularization constant λ we had the new regularized loss function $\text{TrainLoss}_1'$ (with similar variable definitions as before) as

$$\text{TrainLoss}_1' = \left(\sum_{t\in D}\sum_{i=1}^{n}\frac{1}{\overline{p}_i}|p_{i,t} - \hat{p}_{i,t}|\right) + \sum_{P=W_1,W_2,b_1,b_2}\frac{\lambda}{2}\frac{||P||^2}{|P|} \quad (17)$$

$$= \left(\sum_{t\in D}\sum_{i=1}^{n}\frac{1}{\overline{p}_i}|p_{i,t} - \hat{p}_{i,t}|\right) + \sum_{P=W_1,W_2,b_1,b_2}\frac{\lambda}{2}\frac{\sum_i\sum_j|P_{i,j}|^2}{|P|} \quad (18)$$

where $|P|$ denotes the number of elements in P. Though we implemented L2 regularization, the difference between test and train errors (on validation sets) did not significantly decrease and the overall error increased massively, indicating that the overfitting was not necessarily a significant problem. The hyperparameter determination table for the regularization constant λ is shown below. We imposed the regularization constraint on all of the parameters W_1, b_1, W_2, and b_2 (as well as U for the Elman network).

λ	Train Error	Test Error
0	10.401	9.602
0.25	35.857	38.259
0.5	60.571	63.678
0.75	82.597	86.047
1.0	108.186	110.877

Figure 11: *Cross-validation for regularization constant λ on feed-forward network with softplus activation function and L2 loss (20 iterations)*

VII. Results and Error Analysis

Using the results from figure (9) and utilizing early stopping, we trained the 12 different models (determined by activation functions, neural network type, and loss function) and then calculated the loss on the test set. The following table summarizes the results

Network Description	Train Error	Test Error	L1 Test Error
Feed-Forward, tanh, L1	2.798	2.835	2.835
Feed-Forward, ReLU, L1	4.029	3.941	3.941
Feed-Forward, SoftPlus, L1	2.307	2.335	2.335
Feed-Forward, tanh, L2	27.526	26.853	7.192
Feed-Forward, ReLU, L2	12.960	12.393	5.799
Feed-Forward, SoftPlus, L2	10.401	9.602	4.328
Elman, tanh, L1	3.816	3.787	3.787
Elman, ReLU, L1	3.960	4.383	4.383
Elman, SoftPlus, L1	2.490	2.770	2.770
Elman, tanh, L2	33.329	32.297	8.072
Elman, ReLU, L2	15.957	19.489	4.762
Elman, SoftPlus, L2	18.043	24.083	4.725

Figure 12: *Final train/test error determined utilizing early stopping*

Note that these results were for predicting the next hour utilizing the optimal numbers for hidden neurons and past input hours. We have included a column of the L1 test error which allows comparison of models trained with the L2 loss function to other models (including the baseline and oracle).

There were a number of conclusions to draw from this. First, in general, the L1 loss performed better than the L2 loss in every test with the same neural network type and the same activation function. This may have been due to instability encountered with the L2 loss. Slight changes in step size resulted in large variation during stochastic gradient descent (though repeated runs with the certain step size choices were consistently stable). If we were not careful in choosing the step size, overflow became a problem with the L2 loss. Furthermore, the best activation function was SoftPlus in all cases. Finally, though it appeared that the feed-forward network was superior to Elman in all but one case, this may just be due to the extra computational cost associated with Elman (which forced us to run fewer iterations in that case). It appeared that in any case, the benefit of the Elman RNN was outweighed by the computational cost. Across the types of neural networks we examined, the feed-forward network with SoftPlus activation, trained to minimize L1 loss, had the lowest L1 test error.

Secondly, we saw that these models generalize quite well and overfitting was not a significant problem. In particular, the maximum discrepancy between train and test error for L1 error was 0.423 and occurred in the case with Elman and ReLU. For the L2 cases, the train error was greater than the test error in all but two cases (though in these cases there did appear to be some overfitting).

In general, these results were not better than the average results from local linear regression. However, for predicting more than several hours in the future, the neural network models tended to outperform repeated linear regression. This comparison was made in the below table.

n	Oracle Error	Baseline Error	Neural Net Error
1	1.238	2.269	3.051
6	1.188	4.973	4.272
12	1.089	5.915	4.824

Figure 13: *Comparison of neural net (feed-forward, SoftPlus, L1) predictions for predicting the point n hours in the future*

The following graph depicts this failure of local linear regression in comparison to the neural network.

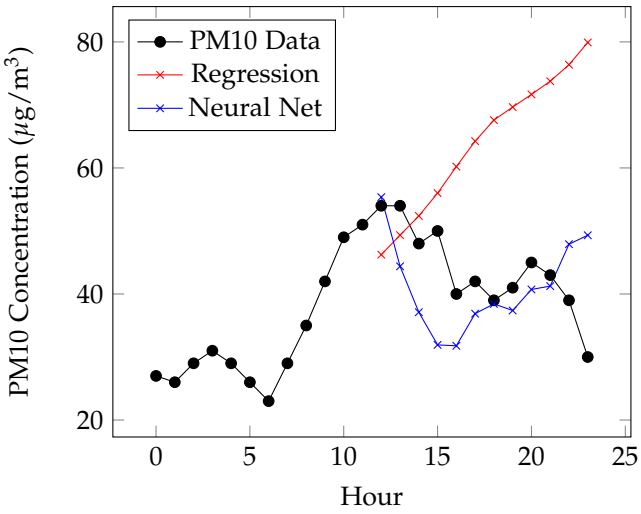

Figure 14: *Comparison of linear regression and neural network (feed-forward, ReLU, L1) for predicting 12 hours in the future of PM10*

VIII. Conclusion and Future Directions

In general, the neural network models, when compared to the baseline linear regression model, were better at predicting the

pollutant concentrations more than several hours in the future, but not significantly better for predicting the immediate future. This was due to the globally nonlinear nature of the data, and this observation implied that the neural network models may have been better at prediction when the pollution levels were changing rapidly as linear regression would likely fail in that case as well. Thus, neural network prediction models could be potentially more useful in a very dynamic environment (or one in which you simply need to predict many hours in the future). This observation indicated that the neural nets may, in general, deviate less from the average value than linear regression. A potential future investigation could be to compare the neural nets to a baseline algorithm that simply tracks a running average and returns that value at each point.

There were a number of other ways that the performance of the predictor could have been improved. Due to the time-intensive nature of cross-validation, we were unable to thoroughly test regularization constants and early stopping constants for every model represented in table (9). It could have potentially improved performance (in particular, overfitting) to improve the values of these hyperparameters. We were also limited computationally in the number of hidden neurons that could be tested via cross-validation. Ideally, we would test models with greater than 200 hidden neurons, even though this is likely to cause overfitting. Furthermore, cross-validation was 2-fold, and it would be significantly more comprehensive if k-fold cross-validation were used for greater k.

We noted earlier that the loss functions as defined were fairly robust and motivated. However, we could potentially further improve the loss function by taking into account the variance in the signals. For example, if there were a pollutant with an average concentration of 10 ppm and a very small standard deviation (say 0.001 ppm), then a fluxuation to 11 ppm is a drastic change that isn't accounted by the loss functions defined in this paper. Essentially, we implicitly assumed that the variance was proportional to the average value, which was not necessarily true. Fixing this could potentially yield a more accurate predictor, and in particular one that is more sensitive to pollutants with low levels variation.

Another potential improvement could be made in the quality of the data used to train the predictor. Choosing a larger data set and, in particular, one with longer contiguous sequences of measurements, would likely yield more accurate predictions. Furthermore, the data used in this project was collected from different locations in northern Taiwan, so it would be potentially more accurate if a larger set of data were extracted from a single location for training the predictor.

Finally, this paper covered only the simplest neural network models for this task (albeit, with a large number of hyperparameters). It would be a natural extension to test performance with more complex deep learning models, such as high-layer feed-forward models, LSTM recurrent neural networks, etc. Combining these models with this project one may be able to design predictors that are significantly better at predicting certain aspects of the data, if not all of the data.

IX. Acknowledgements

We thank Percy Liang and Kratarth Goel from Stanford University for their guidance for the direction of this project. We also used the Stanford Corn computer cluster for running our cross-validation.

References

[1] Atakan Kurt, Betul Gulbagci, Ferhat Karaca, Omar Alagha, An online air pollution forecasting system using neural networks, Environment International, Volume 34, Issue 5, July 2008, Pages 592-598, ISSN 0160-4120, http://dx.doi.org/10.1016/j.envint.2007.12.020.

[2] Britz, Denny. "Implementing a Neural Network From Scratch in Python âĂŞ An Introduction." *WildML: AI, Deep Learning, NLP*, 3 Sept. 2015. Web. 15 Nov. 2016.

[3] De Vito, Saverio et al. "CO, NO2 and NOx urban pollution monitoring with on-field calibrated electronic nose by automatic bayesian regularization." *Sensors and Actuators B: Chemical*, vol. 143, no. 1, 2009, pp. 182-191. Web. 26 October 2016.

[4] Environmental Protection Administration Taiwan. "Annual Hourly Data: Air Quality Monitoring Network." Kaggle (Nelson Chu) [distributor], 2016. Web. 23 October 2016.

[5] Nunnari, Giuseppe et al. "The application of neural techniques to the modelling of time-series of atmospheric pollution data." *Ecological Modelling*, vol. 111, no. 2-3, 1998, pp. 187-205. Web. 25 October 2016.

[6] Pérez, Patricio, Alex Trier, and Jorge Reyes. "Prediction of PM2.5 Concentrations Several Hours in Advance Using Neural Networks in Santiago, Chile." Atmospheric Environment 34.8 (2000): 1189-196. Web.

[7] S.A Abdul-Wahab, S.M Al-Alawi, Assessment and prediction of tropospheric ozone concentration levels using artificial neural networks, Environmental Modelling & Software, Volume 17, Issue 3, 2002, Pages 219-228, ISSN 1364-8152, http://dx.doi.org/10.1016/S1364-8152(01)00077-9.

[8] Soni, Aparna. "Application of Neuro-Fuzzy in Prediction of Air Pollution in Urban Areas." *IOSR Journal of Engineering* 02.05 (2012): 1182-187. Web.

[9] WHO. "Air Pollution Levels Rising in Many of the World's Poorest Cities." World Health Organization. World Health Organization, 12 May 2016. Web. 1 Dec. 2016.