Definition

Project Overview

DiDi Chuxing is a ride-hailing company with a business model similar to Uber and focused on the Chinese domestic market. The company collects and analyses a large amount of data in order to, among other things, ensure a sufficient supply of cars available for those looking for rides. The DiTech Challenge is an opportunity for machine learning engineers and researchers to help improve the predictive capability of these algorithms using their particular knowledge and skills.

DiDi Chuxing has provided a plethora of data for our consideration. In addition to historical ride order data for much of January 2016, we are provided with traffic and weather data for the same period, as well as points of interest (POI) data which details, in a cryptic manner, the sights and attractions in all 66 districts of the undisclosed city. It is up to those challenged to determine which features out of all provided are key to making predictions about the gap, i.e., the difference between the number of customers requesting a ride in a given district in a given 10-minute interval and the number of cars available to supply rides. As the gap is always a integer greater than or equal to zero, this problem is most naturally treated as one of regression.

I have chosen to pursue a solution to this problem using artificial neural networks (ANNs) primarily because I want to learn more about how artificial neural networks function. My personal gal in this project is to learn as much as I can about neural networks. I would like to perform well on the DiTech challenge, but if this choice prevents me from doing so then so be it. Learning is its own reward.

Problem Statement

The DiTech Challenge is a regression problem, as a single numerical output--the gap--is to be predicted from an understanding of previous supply and demand data, traffic and weather data, and POI data for each district in the city. The time frame is limited to the three previous 10-minute time intervals in the test data provided by DiDi Chuxing. Scoring is done using mean absolute percentage error (MAPE).

The goal of this project is to construct an artificial neural network that yields a model of the training data that results in the lowest possible MAPE. As I cannot judge the performance of the model on the test data by myself (and the competition now being closed, my best efforts far too inferior to have made it through to the final round), I will be forced to split the given training data into a training set and a test set. Thus the test data insofar as this project is concerned is actually a chunk of the training data provided by DiDi Chuxing.

As I discussed above I will build an artificial neural network to do this. This ANN will be a purely feed-forward neural network (i.e., there will be no loops within the network). I will make use of cross validation and grid search to find ideal hyper-parameters.

Metrics

The scoring metric provided by DiDi Chuxing is mean absolute percentage error, or MAPE. MAPE is defined as

$$C=\frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i-y_i|}{y_i},\forall\, y_i\neq 0 \qquad (1)$$

where $\hat{y}_i$ is the predicted value for sample *i*, y_i is the target value for sample *i* and *N* is the number of data points, If $y_i=0$ then the MAPE contribution from that data point is automatically considered to be *0* no matter how far out the prediction.

MAPE is less commonly used than the more familiar mean squared error (MSE) cost function. One solid reason for this is that MAPE is not a smooth function. There's a kink whenever $\hat{y}_{i=}y_i$. It's possible to perform gradient descent on a MAPE scoring function, but it is more challenging than minimizing more well-behaved functions. A well-behaved function such as MSE has a gradient that decreases smoothly toward zero near its minimum. That doesn't happen with MAPE; the curve approaches its minimum with some non-zero steepness. Overshooting during gradient descent becomes a difficult problem to contend with.

Unlike MSE, MAPE punishes errors more when the absolute value of y_i is small. Thus if one is attempting to minimize MAPE, one would want to ensure that one gets the predictions for small values right. In fact, if one knew nothing about the factors that gave rise to some set of observations, one would minimize MAPE by predicting some small value at the low end of the distribution [Gneiting]. For example, suppose one was dealing with a discrete exponential distribution. Without knowing the factors that gave rise to the distribution, the best strategy would be to predict *1* every time. Predicting *0* would not work as true zeros do not contribute toward MAPE. Later on I will invoke this 'guess-1' strategy as a benchmark against which to compare the performance of the ANN I will construct.

Analysis

Data Exploration

Substantial effort went into converting the data as provided into a form usable for learning. The code for converting and transforming the raw data is provided here, without much verbosity. The raw training data has not been uploaded to GitHub, but is (hopefully still) available here. The manipulated training data is available, so when the is run, these files will be read relatively quickly (see the `retrieve` function). Do please take notice that for the purposes of this investigation I am focusing on the training data, as I will require data for which known values of gap exist.

Order Data

We are provided with order data from a certain city on the majority of days in January 2016. These data are in the form of individual orders. We are told when they were placed, in which district they were placed, by whom, at what price, and by whom they were fulfilled (if at all). This level of granularity is too detailed for our requirements. I have summarized these data in the following manner

- District_ID: the region of the city as determined by DiDi Chuxing
- UTC: universal time coordinate, an ordered 10-minute-long time interval, referenced to Jan 1st 2016 at 00:00.
- Demand: number of requests for cars in a district in a UTC
- Supply: number of fulfilled requests in a district in a UTC, always less than or equal to demand. (We have no way of knowing if there are other drivers out there who would have responded had not some other driver responded first)
- Med Price: the median price of a ride in that district in that UTC
- Gap: the arithmetic difference between demand and supply
- Weekday: ordinal value of the day of the week (Jan 1st --> 0 by default)

An example of the summarized order data is given in Table 1. It provides data on the first five 10-minute time periods (UTC) for the first day of January in District 1. For example, in UTC 1 in District 1 there were 187 requests for rides, 178 of which were fulfilled. The 9 rides unfulfilled make up the gap.

Table 1: Summarized order data example

district_ID	UTC	demand	supply	med_price	gap	weekday
1	1	187	178	12	9	0
1	2	198	191	10	7	0
1	3	192	182	11	10	0
1	4	172	167	11.5	5	0
1	5	153	152	11	1	0

The median supply, demand and gap values from the training data are summarized in Table 2. The median demand is 7, supply 6 and thus the median gap is 1.

Table 2: Median supply, demand, gap and price from provided training data

	median
demand	7
supply	6
med_price	13
gap	1

POI Data

POI stands for points of interest. These data are intentionally cryptic; they refer to the number of points of interest of various undisclosed type, per district in the city. Points of interest may include things such as restaurants, theaters, historical sites, stadiums etc. There are 64 categories and sub-categories. A number of POI data columns are linear combinations of other columns. Columns have been deleted as necessary in order to achieve linear independence.

It seems that the numbers in most (but not all) columns are multiples of 83. This may be a means of further disguising the true nature of the points of interest. It does not appear to be relevant to the task at hand. NaN values are assumed to correspond to 0 points of interest of that type, as zeros are absent from the database.

An small sample of the summarized POI data is given in Table 3. It provides data on five districts, whose order is determined by the order in which they appeared in the raw POI data. POI-1 would appear to be a super-category, with POI-1#1, POI-1#2 apparently being subcategories of POI-1. But the values in the POI-1 column are not simply a sums of the values of its subcategories; the relationship is more complex. You can see that all values appear to be multiples of 83.

Table 3: Exemplary POI data

district_ID	poi-1	poi-1#1	poi-1#2	poi-1#3	poi-1#4	poi-1#5	poi-1#6	poi-1#7	poi-1#8
50	83	83	249	83	0	581	83	0	166
43	83	332	83	0	166	0	249	83	332
63	83	0	83	0	83	415	0	83	0
46	2075	166	1328	249	166	7304	0	83	3154
56	0	0	0	83	0	249	0	0	0

Traffic Data

We are provided with information on traffic congestion in each district at numerous instances. The information is slightly criptic, but the gist of it is that we are given the number of roads per district with certain levels of traffic congestion, labeled 1,2,3 and 4 with 1 being the least bad congestion and 4 being the worst. Uncongested roads appear to be unreported.

There is no traffic data available for District 54. A solution to this missing data problem is required. In terms of supply, demand and gap, District 54 is perhaps most similar to Districts 2, 36 and 57 (see Table 4). I've taken the mean traffic in these three districts in a given time interval as a proxy for the traffic in District 54 during that same time interval.

Table 4: Median supply, demand and gap in districts similar to District 54

district_ID	median demand	median supply	median gap
2	17	16	0
36	16	15	0
54	14	13	0
57	11	11	0
35	9	8	0
13	9	9	0
53	8.5	8	0

In addition to those of District 54 there are other time intervals for which we are provided no traffic data. In cases when traffic data for a district/time interval is missing, I have interpolated a value based upon the nearest two time slots in that district for which data exist. When that is not possible, I have taken the traffic value of the chronologically nearest available measurement in the district in question.

An example of the summarized training traffic data is given in Table 5. Traffic data is reported approximately once every 10 minutes for each district (except District 54 as mentioned above). An example is provided below for the first five UTCs on January 1st in District 1. A summary of the training traffic data is provided in Table 6.

Table 5: Example of traffic data summarized by district and time interval

district_ID	UTC	traffic-1	traffic-2	traffic-3	traffic-4
1	1	1399	318	102	94
1	2	1399	318	102	94
1	3	1491	322	99	64
1	4	1490	287	98	78
1	5	1425	302	95	51

Table 6: Median levels of traffic congestion for all districts and time intervals

	median
traffic-1	327
traffic-2	50
traffic-3	15
traffic-4	10

Weather Data

We are given three pieces of weather information collected at various intervals for the entire city: temperature (in Kelvin), PM25 (a measure of air pollution, with high values being worse) and weather, which appears to be a categorical value describing the weather at the time.

The training data contains weather category 9, which is not present in the test data. Likewise, the test data contains weather category 6, not present in the test data. We will consider these two categories to be 'weather: other', and lump them together as such.

There are multiple occasions when more than one weather reading is take per time interval, in such cases, I have taken the average temperature and PM25, and the first reported weather category.

In case of missing data, I have interpolated values of temperature and PM25 whenever possible. When that is not possible, I have taken the traffic value of the chronologically nearest available measurement in the district in question. Missing categorical weather data is always filled in by copying the nearest chronological value.

An example of the summarized training weather data is given in Table 7. Weather data is reported approximately once every 10 minutes for the entire city. Exemplary data is provided below for the first five time intervals of January 1, just to give you an idea of what these data look like. A summary of the weather data is provided in Table 8. The weather category is not reported here as its categorical an a median of a categorical value is meaningless.

Table 7: Example of weather data by time interval

UTC	weather	pm25	temp
1	1	177	276.65
2	1	177	276.15
3	1	177	276.15
4	1	177	276.15
5	1	177	276.15

Table 8: Median pollution levels and temperatures over the time period of the training data

	median
`pm25`	114
`temp`	279.15

Exploratory Visualization

A plot of count vs gap is shown in Figure 1; count here being defined as the number of data points in the training set that have the corresponding gap is shown on the y-axis with logarithmic scale. The inset shows a close-up of the data up to a gap of 250 to give you a better idea of the shape of what comprises the vast majority of the data. This choice of axes can be explained as follows. One might expect the gap to be distributed exponentially--by definition it cannot be negative and one's intuition expects small values to occur more often than large values. Were the gap distributed exponentially, then a plot of log(count) vs gap would result in an approximately straight line. This is not the case, but it is clear that the probability of observing a particular gap has a strong tendency to decrease as the value of the gap increases.

A bar plot of the median gap in each district vs district ID is shown in Figure 2. The median gap by district extends from a low of 0 for multiple districts to a high of 24 for district 51. Eventually I would like to see the MAPE score on a district level. Will MAPE for district 51 be lower or higher than for district with median gap of 0? It's interesting to speculate: district 51 stands furthest away from the "average" district, thus one would expect predictions for its gaps to be less accurate. But on the other hand MAPE punishes absolute errors less when the true value is large, as is the case for district 51. We shall have to wait and see which of these two factors (or perhaps some other) is the greatest influence on district-level MAPE.

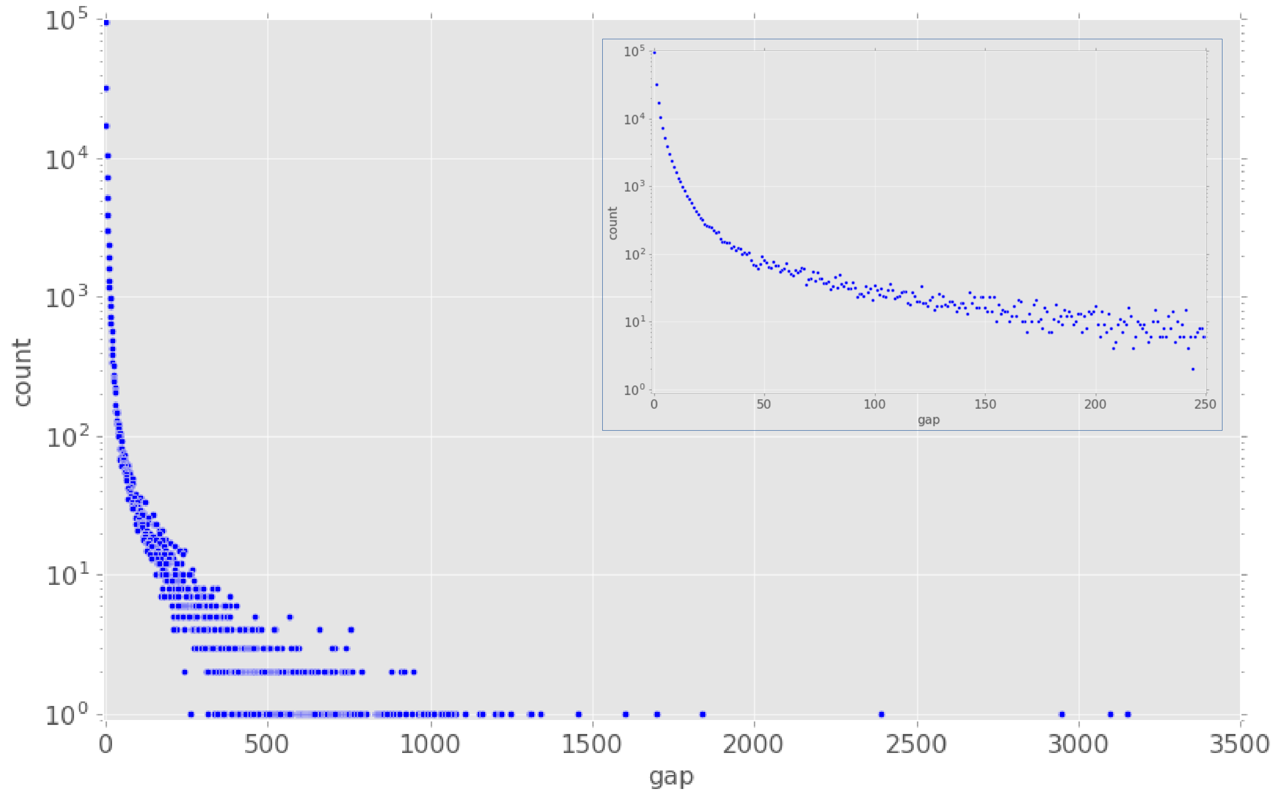

Figure 1: Gap gap count versus gap. Gap count is the number of district/interval combinations with a given gap. The inset shows a close-up for gaps below 250.

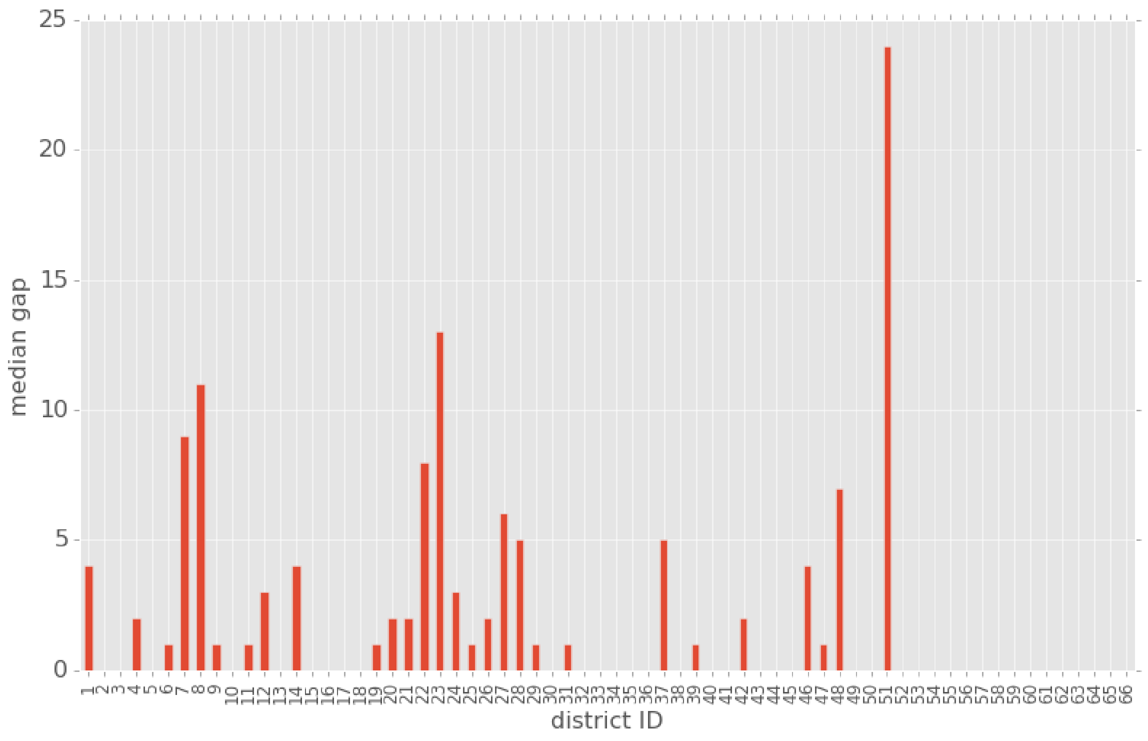

Figure 2: Median gap by district.

Algorithms and Techniques

Feed-Forward Prediction by Neural Network

Let's consider a two-hidden-layer network with *K* input nodes, *M* nodes in the first hidden layer, *N* nodes in the second hidden layer, and a single output node. The *N* nodes in the first hidden layer are all of the same non-linear functional type; call that function f_1. Function f_1 could be a sigmoid function, rectifying function, exponential function, etc., as long as it is non-linear. The nodes in hidden layer *2* are also all of the same non-linear functional type; call that function f_2. This rule will be strictly enforced for any neural network considered in this study; all nodes within a given layer will always share the same non-linear function. The key components of this exemplary neural network are compiled in Table 9.

Table 9: Key Components of an Exemplary Neural Network

Layer	Num Nodes	Function	Weights (incoming)	Biases	Layer Input	Layer Output
Input	J	NA	NA	NA	NA	X_{ij}
Hidden 1	K	f_1	W^1_{jk}	B^1_k	X_{ij}	$f_1(\sum_j X_{ij} W^1_{jk} + B^1_k) = a^1_{ik}$
Hidden 2	M	f_2	W^2_{km}	B^2_m	a^1_{ik}	$f_2(\sum_k a^1_{ik} W^2_{km} + B^2_m) = a^2_{im}$
Output	1	f_3	W^3_m	B^3	a^2_{im}	$f_3(\sum_m a^2_{im} W^3_m + B^3) = \hat{y}$

We can see from the table above that prediction by neural network is an iterative process. We take the output matrix from the previous layer, multiply it by the weights matrix, add the bias vector in element-wise fashion, and apply the current layer function to the whole thing. This output is called the activation matrix, or the prediction if we're dealing with the output layer. The simple iterative nature of the neural network lends itself nicely to functional encoding. Supposing we know the input matrix X_{ij} and all the weights, biases and functions of the various layers of nodes, we can easily calculate the activation at any layer.

Back Propagation

A moderately complex neural network can easily have many thousands of weights and biases to update. This can be easy if we chose the right updating algorithm, or it can be slow and painful if we don't choose wisely. The wise choice is back propagation, the best known algorithm (to this author) for implementing gradient descent atop a neural network. Let's spend some time to understand from whence back propagation comes and what makes it so desirable.

Back propagation is an algorithm that allows gradient descent to proceed rapidly. The most difficult part of gradient descent is finding all the partial derivatives that are the components of the gradient. The key to back propagation's rapidity lies in finding certain partial derivatives that are simple functions of stuff we already know, and then using those partial derivatives to find the precise partial derivatives we need, i.e., the components of the gradient.

We'll use the exemplary neural network from Table 1 to derive the back propagation algorithm. The conceptual model we'll keep in mind is that of errors propagating backwards through the network. Well, not errors exactly; more like the sensitivity of error to certain small changes. But as a conceptual model, back propagation of error-like quantities works well enough. So in thinking along these lines, we ask ourselves what's the simplest meaningful error-like quantity we can compute. Much of the discussion below is inspired by, if not directly taken from, Neural Networks and Deep Learning, in particular Chapter 2.

We defined the cost in terms of the prediction vector $\hat{y}_i$, so let's begin by calculating the partial derivative of the cost function with respect to a particular $\hat{y}_i$

$$\delta_i^3 = \frac{\partial C}{\partial \hat{y}_i} \qquad (2)$$

As stated Eq(2) is the change in the cost function with the change in network output of a given input X_i. Once we recognize that the output of the network is nothing but the the activation of the output node, we can envision calculating the change in the cost function with respect to the change in activation at any and all nodes in the network for any given input X_i. We do this by propagating bthe error-like quantity ackward through the network That would give us a cost function gradient with respect to activation at all nodes for all input samples. This is not quite what we want, but it is exactly what we need in order to calculate a cost function gradient with respect to all weights and biases of the network.

Let's now propagate the calculation of these error-like quantities farther back into the network, to the second hidden layer.

$$\delta_{im}^2 = \frac{\partial C}{\partial a_{im}^2} = \frac{\partial C}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial a_{im}^2} = \delta_i^3 \frac{\partial \hat{y}_i}{\partial a_{im}^2} \qquad (3)$$

From Table 1 we see that we can write the output from the output node as

$$\hat{y}_i = f_3\left(\sum_m a_{im}^2 W_m^3 + B^3\right) = f_3(z) \qquad (4)$$

We clearly get an idea of how to take the partial derivatives of $\hat{y}_i$ with respect to a^2_{im}, Using this knowledge and Eq(3) we obtain

$$\delta_{im}^2 = \delta_i^3 \frac{\partial \hat{y}_i}{\partial a_{im}^2} = \delta_i^3 \frac{\partial \hat{y}_i}{\partial z}\frac{\partial z}{\partial a_{im}^2} = \delta_i^3\, \hat{y}_i{}'\, W_m^3 \qquad (5)$$

where $\hat{y}_i'$ is the derivative of the function $\hat{y}_i = f_3(z)$ with respect to z.

Let's take one more back-propagation step to Hidden Layer 1

$$\delta_{ik}^1 = \frac{\partial C}{\partial a_{ik}^1} = \sum_m \frac{\partial C}{\partial a_{im}^2}\frac{\partial a_{im}^2}{\partial a_{ik}^1} = \sum_m \delta_{im}^2 \frac{\partial a_{im}^2}{\partial a_{ik}^1} \qquad (6)$$

From Table 1 we see that we can write the relationship between a^2_{im} and a^1_{ik} as

$$a^2_{im} = f_2\left(\sum_k a^1_{ik} W^2_{km} + B^2_m\right) = f_2(z) \qquad (7)$$

Taking appropriate partial derivatives of Eq(7), we can rewrite Eq(6) as

$$\delta^1_{ik} = \sum_m \delta^2_{im} \frac{\partial a^2_{im}}{\partial a^1_{ik}} = \sum_m \delta^2_{im} \frac{\partial a^2_{im}}{\partial z} \frac{\partial z}{\partial a^1_{ik}} = \sum_m \delta^2_{im} (a^2_{im})' W^2_{km} \qquad (8)$$

where now $(a^2_{im})'$ is the derivative of $a^2_{im}=f_2(z)$ with respect to z.

The benefits of choosing a two-hidden-layer network for our example are apparent when we consider Eq(5) and Eq(8) in conjunction. These two equations, considered together, allow us to generalize a back-propagation rule for the error-like quantities.

$$\delta^L_{i\alpha} = \sum_\beta \delta^{L+1}_{i\beta} (a^{L+1}_{i\beta})' W^{L+1}_{\alpha\beta} \qquad (9)$$

Eq(9) states that the error-like quantity at each node in layer L for input X_i is calculated from the sum of error-like quantities, activations and connected weights over all nodes from one layer downstream. We have just demonstrated the general rule for propagation of "error" back up through the network. We need only to have an initial, most downstream set of error-like quantities from Eq(2) to feed into Eq(9) to get the procedure started.

Now that we can calculate these error-like quantities at each node in the network for each input X_i, we can find the partial derivatives with respect to the weights and biases that are the components of the gradient. Using Eq(4), we see clearly how to take the partial derivatives of $\hat{y}_i$ with respect to W^3_m and B^3. Using this knowledge and Eq(1) we obtain

$$\frac{\partial C}{\partial B^3} = \sum_i \frac{\partial C}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z} \frac{\partial z}{\partial B^3} = \sum_i \delta^3_i y'_i \qquad (10)$$

Let's do the same for W^3_m

$$\frac{\partial C}{\partial W^3_m} = \sum_i \frac{\partial C}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z} \frac{\partial z}{\partial W^3_m} = \sum_i \delta^3_i y'_i a^2_{im} \qquad (11)$$

Going on to Hidden Layer 2, we see that Eq(7) shows us the way to find the partial derivatives of the cost function with respect to W^2_{km} and B^2_m

$$\frac{\partial C}{\partial B^2_m} = \sum_i \frac{\partial C}{\partial a^2_{im}} \frac{\partial a^2_{im}}{\partial z} \frac{\partial z}{\partial B^2_m} = \sum_i \delta^2_{im} (a^2_{im})' \qquad (12)$$

For W^2_{km} we have

$$\frac{\partial C}{\partial W^2_{km}} = \sum_i \frac{\partial C}{\partial a^2_{im}} \frac{\partial a^2_{im}}{\partial z} \frac{\partial z}{\partial W^2_{km}} = \sum_i \delta^2_{im} (a^2_{im})' a^1_{ik} \qquad (13)$$

Finally, we do the same thing for $W^1{}_{jk}$ and $B^1{}_k$. We see from Table 1 that

$$a_{ik}^1 = f_1\left(\sum_j X_{ij} W_{jk}^1 + B_k^1\right) \qquad (14)$$

So we find for $B^1{}_k$ that

$$\frac{\partial C}{\partial B_k^1} = \sum_i \frac{\partial C}{\partial a_{ik}^1} \frac{\partial a_{ik}^1}{\partial z_{ik}^1} \frac{\partial z_{ik}^1}{\partial B_k^1} = \sum_i \delta_{ik}^1 (a_{ik}^1)' \qquad (15)$$

For $W^1{}_{jk}$ we find

$$\frac{\partial C}{\partial W_{jk}^1} = \sum_i \frac{\partial C}{\partial a_{ik}^1} \frac{\partial a_{ik}^1}{\partial z_{ik}^1} \frac{\partial z_{ik}^1}{\partial W_{jk}^1} = \sum_i \delta_{ik}^1 (a_{ik}^1)' X_{ij} \qquad (16)$$

The final step is to generalize these rules, which we are now in a position to do.

$$\frac{\partial C}{\partial B_\beta^L} = \sum_i \delta_{i\beta}^L (a_{i\beta}^L)' \qquad (17)$$

$$\frac{\partial C}{\partial W_{\alpha\beta}^L} = \sum_i \delta_{i\beta}^L (a_{i\beta}^L)' a_{i\alpha}^{L-1} \qquad (18)$$

Provided we know the ultimate value of δ_i from Eq(1) we can calculate using Eq(9), Eq(17) and Eq(18) all the partial derivatives we need to construct the gradient of the cost function with respect to all the weights and biases of any arbitrarily large feed forward network.

Activation Function Selection

We need to choose activation functions for each of our hidden layers and the output layer. There's a wide variety of hidden layer activation functions to choose from, including perceptrons, sigmoid units, rectifying linear units (ReLU) and exponential units. Perhaps even softmax. Virtually any non-linear function will suffice, but we would like that non-linear function to have a simple derivative. Our options for the output function are more limited, as we need a function whose range is all non-negative real numbers ($\boldsymbol{R}_+$). That eliminates perceptron, sigmoid and softmax immediately. Now, how do we decide?

We could conceivably include activation function type for each layer individually in grid search, and optimize over the full spectrum of possibilities. Perhaps the globally optimal network has a sigmoid first hidden layer, an exponential second hidden layer, a ReLU third hidden layer and so on. But searching over such an elaborate grid is impractical considering the realities of the computer hardware available for this investigation and all the other parameters (layer widths, learning rates, learning rate decays, number of epochs) we wish to optimize. So we are taking a much simpler course: we choose ReLUs for all hidden layers and the output layer for no other reason that ReLUs are the activation functions of choice in Udacity's Deep Learning course and seem to work.

A rectifying linear unit does as its name implies and rectifies the inputs into it. It passes the input on unchanged if the input is a positive number, and passes *0* on if the input is negative. It can be stated as

$$a_{i\beta}^{L+1} = max\left(\sum_{\alpha} a_{i\alpha}^{L} W_{\alpha\beta}^{L+1} + B_{\beta}^{L+1}, 0\right) \qquad (19)$$

where *a* is the activation of a particular node for a particular input, *W* is the weight of the connection between two nodes, *B* is the bias of the node at which the activation is calculated, the superscripts reference the layer, and the subscripts are referenced by *i*: the data point, *α*: the node from the previous layer upstream and *β*: the node from the layer at which the activation is calculated.

The derivative of Eq(19) is easy to determine:

$$\left(a_{i\beta}^{L+1}\right)' = sign\left(max\left(\sum_{\alpha} a_{i\alpha}^{L} W_{\alpha\beta}^{L+1} + B_{\beta}^{L+1}, 0\right)\right) \qquad (20)$$

where *sign(z=0)=0* and *sign(z>0)=1*.

Benchmark

I will use the 'guess-1' strategy inspired by Gneiting as a benchmark for comparison. The 'guess-1' strategy is the optimal strategy for minimizing MAPE for exponentially-distributed data if one knew nothing about the factors that gave rise to the distribution. While the gap in the DiTech Challenge data is not truly exponentially distributed, it does have some of the key features of an exponential distribution that makes the 'guess-1' strategy the optimal choice, namely that *0* is the most commonly observed gap, and the probability of observing higher gaps 'strictly' decreases with increasing gap.

The 'guess-1' approach works because MAPE punishes errors more severely for examples for which the true gap is small. Thus one can expect to minimize MAPE if one ensures that correct prediction of small gap examples is prioritized over prediction of large-gap examples. Note that a 'guess-0' strategy is sub-optimal, as any observation with true gap of *0* contributes nothing to the MAPE (see Eq(1)). If the data were not exponentially distributed, for instance, it they followed a normal distribution with mean *100* and standard deviation *10*, then the 'guess-1' strategy—and quite possibly any 'guess-min' strategy—would be sub-optimal as far as pure guessing strategies go.

Methodology

Data Preprocessing

As mentioned previously, all missing data were filled in by interpolation if possible, and by backfilling if not. The exception to this general rule was the missing traffic data from District 54, which was averaged over the three most similar districts at any given time interval.

No outlier filtering was performed on the data.

All time-dependent data were replicated for the three previous time intervals. In the data I will use for training and prediction, there are three columns of prior-interval traffic data, weather data and supply-demand data.

All original columns X_j of numerical data were duplicated and logarithms taken (technically $log(X_j + 1)$ to account for zero values in the data). Thus all original numerical feature columns are available in both raw and log form.

One-hot encoding was performed on all categorical data, such as weather and weekday.

All data were normalized to mean *0* and standard deviation *1*, including one-hot encoded data. I had a lot of misgivings about this. On one hand, normalizing OHE data seems like an odd thing to suggest. On the other hand, I don't know if normalizing categorical data without normalizing OHE data might lead to some unanticipated glitch in the model (the un-normalized OHE data will have mean greater than *0*; would that be a problem? I think it would for the correlation analysis used to order the columns by importance). So it's been done.

Implementation

The training data as provided by DiDi Chuxing was split 80/20 into training and test data for the model. The original test data provided by DiDi Chuxing was not considered as part of this analysis Why? Well, the DiTech challenge is closed as of this writing, and I have no way to assess predictions of the original test data since test labels are proprietary and hidden to me. Thus the need to split a usable test set off the original training data.

The constant ε represents some very small value, such as the machine precision number. Thus it is not too difficult to see that the dMAPE function returns *0* whenever the value of the gap in the *target* array is *0*. This approach is very fast due to vectorization; using something much more closely resembling the MAPE function with conditions for target gap of *0* would be considerably slower.

ANNs are trained using three-fold cross validation. The training data is split into three folds, and three models, each with the same hyper-parameters, are trained with three two-fold combinations, with the remaining fold being used as validation data. Test data is fit to each of the three models as well. At the end of the cross validation, the three sets of validation predictions are concatenated vertically, and the MAPE for this full set of validation data is calculated. Simultaneously, the three sets of test predictions are concatenated horizontally and averaged, and this average test prediction is used to determine the test MAPE for the hyper-parameter set.

During my participation in the DiTech challenge, I discovered the benefits of a perpetual search procedure. I created an algorithm that performs a randomized search around the best known set of hyper-parameters, updating the hyper-parameters when a new best-known set, as determined by the validation MAPE, are discovered. The grid search algorithm records the hyper-parameters and MAPE for both validation and test predictions for each model after 3-fold cross validation. If the validation MAPE is a new record then the full training data is used to train a model with the newly discovered best known hyper-parameters, and test predictions are made and fed to the MAPE function to obtain what I expect to be the best model possible with the best known hyper-parameters. My thinking here is that a model trained with the complete training data would produce test predictions superior to the average of three similar models trained with three overlapping subsets of training data, such as we encounter with three-fold cross validation. We shall soon see that my thinking is erroneous.

The variable hyper-parameters explored by grid search are

- number of feature columns

- number of nodes per each of the four hidden layers (varied independently)
- η, the learning rate
- α, the learning rate decay
- number of epochs, where an epoch consists of one complete round of training data having been fed into the model

The variable number of feature columns requires some explication. Due to hardware limitations, I am unable to feed all 274 feature columns into the model. It simply takes too long to train. So I created a function that orders the feature columns by (the absolute value of) correlation coefficient. Feature columns are selected in order of importance, up to the number determined by the randomized grid search.

By the way, the perpetual search uses a random normal distribution with mean equivalent to the best known value of a parameter and standard deviation of 10% of that best known value. The randomization for the *α* parameter works slightly differently due to the fact that it must never exceed *1*.

Pseudocode for the MAPE function is shown below.

```
def MAPE(prediction::Nx1 array, target::Nx1 array):
    I = [set of all indices for which target > 0]
    target = target[I]
    prediction = prediction[I]
    return sum(abs(prediction – target) / target) / N
```

While the encoded MAPE function is pretty much as you would expect from Eq(1), the derivative of the MAPE function, which is required for back propagation, is rather more complicated. Because the backpropagatio algorithm demands the change in MAPE with the change in each weight/bias for each data point, we cannot 'skip' the calculation when the true gap is zero as one can for the MAPE function. Pseudocode for the derivative MAPE function is shown below

```
def dMAPE(prediction::Nx1 array, target::Nx1 array):
    return (target * sign(prediction – target)) / (N*target**2 + ε)
```

Refinement

I began the process of identifying the optimal hyper-parameters for a three-hidden-layer ANN using grid search. The grid search incorporated four parameters, with values given below

- nodes: 10, 20, 50, 100
- learning rate (η): 0.1, 0.5, 1
- learning rate decay (α): 0.9, 0.99, 0.999
- epochs: 100, 200, 300, 400

In this grid search procedure, the number of nodes per input and hidden layers were not varied independently, i.e., an ANN with 10 input nodes had 10 nodes in each hidden layer as well.

Figure 3 attempts to show the results of the grid search procedure. I attempted to fit as much information onto this graph as possible, so please allow me to explain. The graph plots the validation MAPE on the vertical axis versus $\frac{\eta}{1-\alpha}$, an measure of the "intensity" of learning by the ANN, on the horizontal axis. The individual points are color coded by the complexity of the model, specifically, $\log_{10}(N_i N_{h1} N_{h2} N_{h3})$ —N representing the number of nodes per input or hidden layer—with yellow points indicating 10 nodes and blue indicating 100. The size of the points is proportional to the number of training epochs undertaken. It also shows the benchmark result for the Guess-1 strategy discussed earlier. The graph will be discussed in detail in the next section.

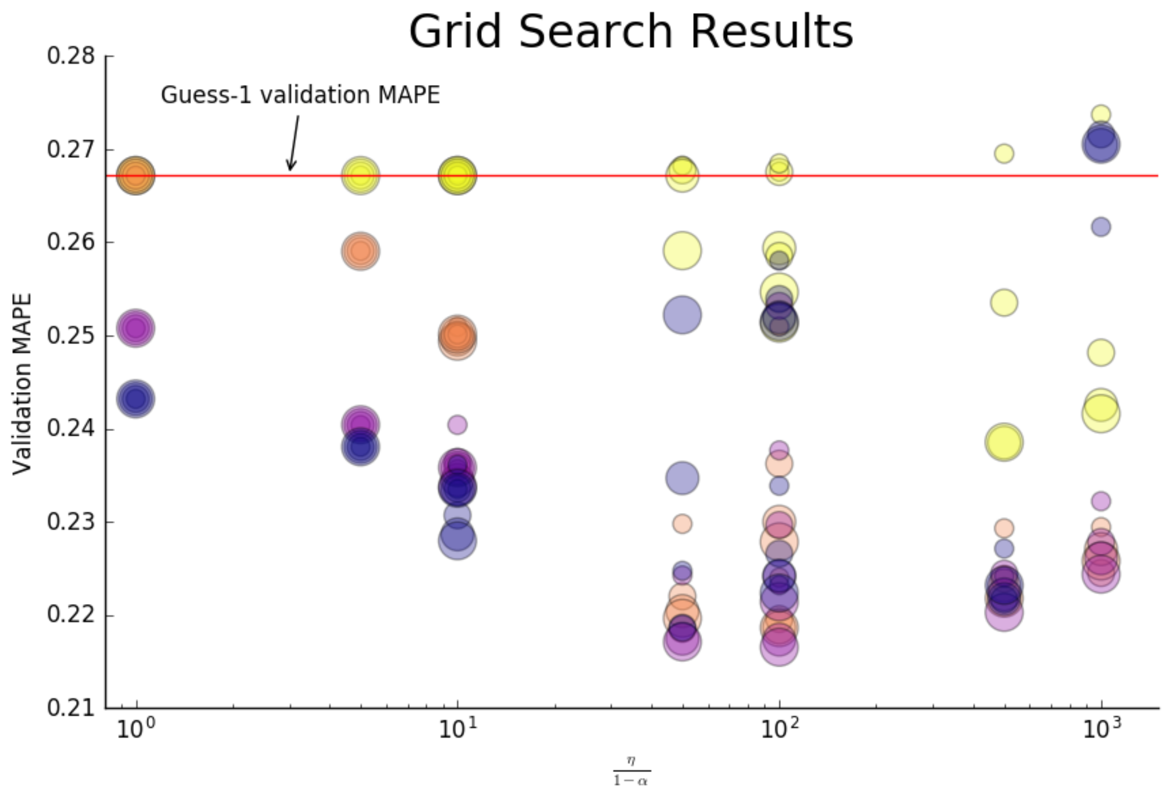

Figure 3: Validation MAPE *obtained during grid search* versus *η/(1-α).* Points are color coded by *log($N_iN_{h1}N_{h2}N_{h3}$)* with yellow indicating fewer and blue greater number of nodes. Points are sizes by the number of training epochs.

The best model found by grid search had the parameters and MAPEs shown in Table 10.

Table 10: Best hyper-parameters and MAPEs after grid search

Nodes	50
η	1
α	0.99
Epochs	400
Val MAPE	0.2165
Pseudo Test MAPE	0.2170
Full Test MAPE	0.2158

Pesudo test MAPE is the MAPE for the test data based on the average of the three models generated during the three-fold cross validation. The full test MAPE is the MAPE for the test data for a single model with the best known hyper-parameters trained on the full complement of the training data.

Now that we have a good starting point, we can perform perpetual search around it. Perpetual search randomizes all hyper parameters independently in the neighborhood of the currently best known hyper-parameters, and then fits the randomized model. It keeps going until you say stop. Here I stopped after 50 iterations. The grid search results graph updated with the results from perpetual search is shown in Figure 4. I only found one set of hyper-parameters that led to validation MAPE superior to the best obtained from grid search. The best known hyper-parameters for a three hidden layer ANN and the associated MAPE values are shown in Table 11.

Results

Model Evaluation and Validation

It's clear from Figure 4 that most combinations of hyper-parameters lead to better results that the Guess-1 approach. But not always. In fact, a lot of the time the hyper-parameters induce the ANN to learn the Guess-1 strategy (and on a few occasions do worse). This is quite interesting, as at first glance it seems that such models have minds of their own and learn the optimal guessing strategy without learning from the data. But recall that the backpropagation algorithm makes use of gradient descent, and that gradient descent will seek out a local minimum in parameter space. If a model is to learn from the data, it must be able to “escape” from the trough surrounding the Guess-1 local minimum. This seems to require a certain minimum amount of complexity. Simple models cannot escape, and thus gradient descent draws them into the Guess-1 minimum. The models that under perform the Guess-1 strategy likely haven't reached the minimum, due to too few epochs or too steep of a learning rate decay, or perhaps bounce around the minimum due to too shallow of a learning rate decay.

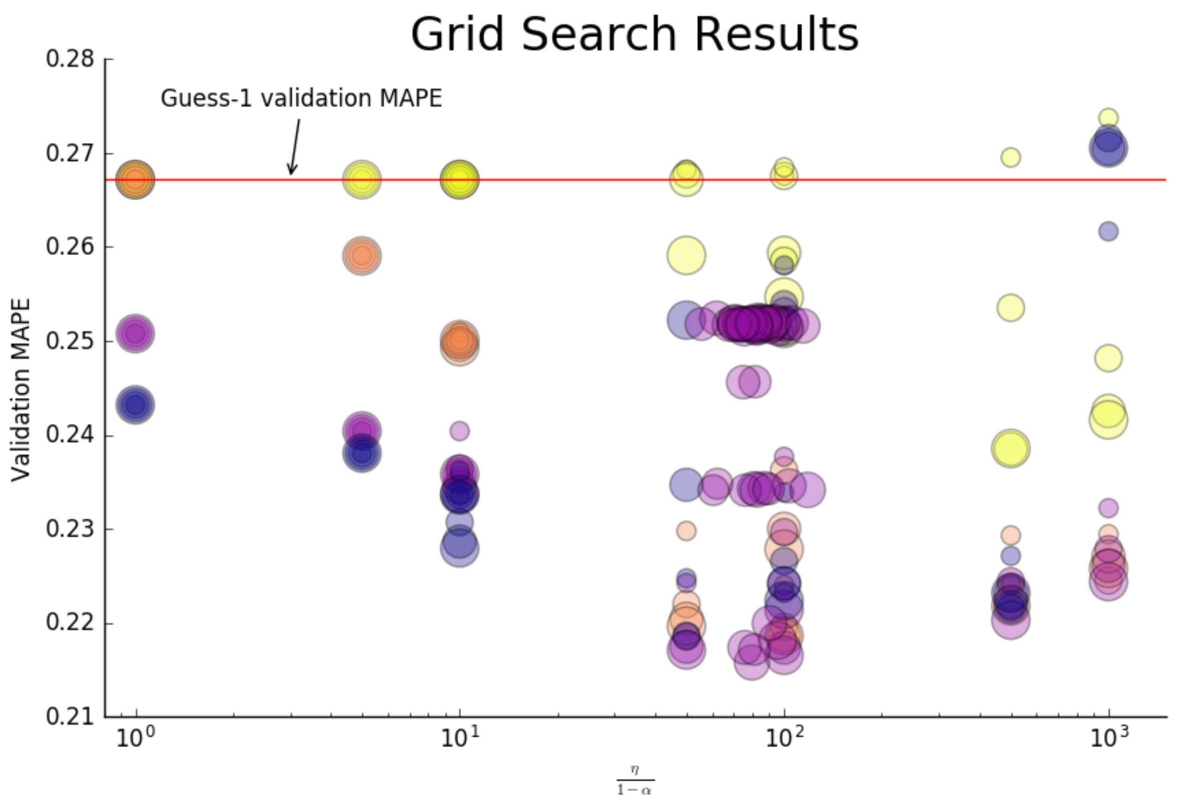

Figure 4: Validation MAPE obtained during grid search and perpetual search versus *η/(1-α)* . Points are color coded by *log($N_iN_{h1}N_{h2}N_{h3}$)* with yellow indicating fewer and blue greater number of nodes. Points are sizes by the number of training epochs.

Table 11: Best hyper-parameters and MAPEs after perpetual search

Input Nodes	64
Hidden Layer 1 Nodes	40
Hidden Layer 2 Nodes	62
Hidden Layer 3 Nodes	34
η	0.898
α	0.989
Epochs	337
Val MAPE	0.2156
Pseudo Test MAPE	0.2164
Full Test MAPE	0.2155

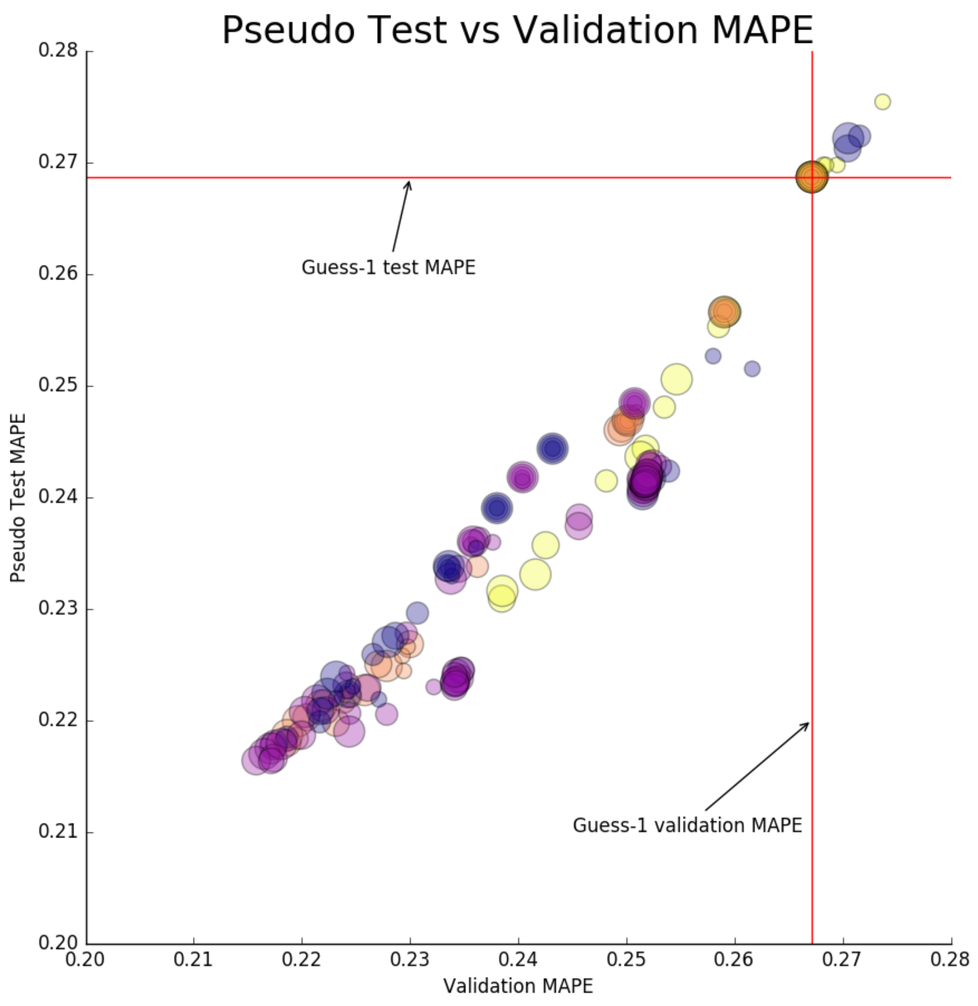

Figure 5: Pseudo test MAPE versus Validation MAPE obtained during grid search and perpetual search . Points are color coded by *log($N_iN_{h1}N_{h2}N_{h3}$)* with yellow indicating fewer and blue greater number of nodes. Points are sizes by the number of training epochs.

A graph of pseudo test MAPE versus validation MAPE for all hyper-parameter combinations investigated in both grid and perpetual searches is provided in Figure 5. The curve has some noise to it, but there is a strong correlation between pseudo test MAPE and validation MAPE. This implies that for any of these models validation MAPE is a good predictor of MAPE for an arbitrary test data set. That's good for the DiTech challenge in which you have a limited number of test data prediction submissions per day: you can have confidence that the test MAPE for your submission will be in the ballpark of the known validation MAPE (provided the training and test data are drawn from the same population).

One thing that is not apparent from Figure 4 is any clear explanation of how combinations of parameters will perform. This is especially apparent when you look at the results from perpetual search: models will seemingly very similar hyper-parameters result in substantially different MAPE values. That brings us to the question of model stability. The experiment is as follows. Run the best model on 15 different train-test splits (excluding the one form which the optimal model was established as optimal). and determine the validation MAPE. The results are shown in Figure 6.

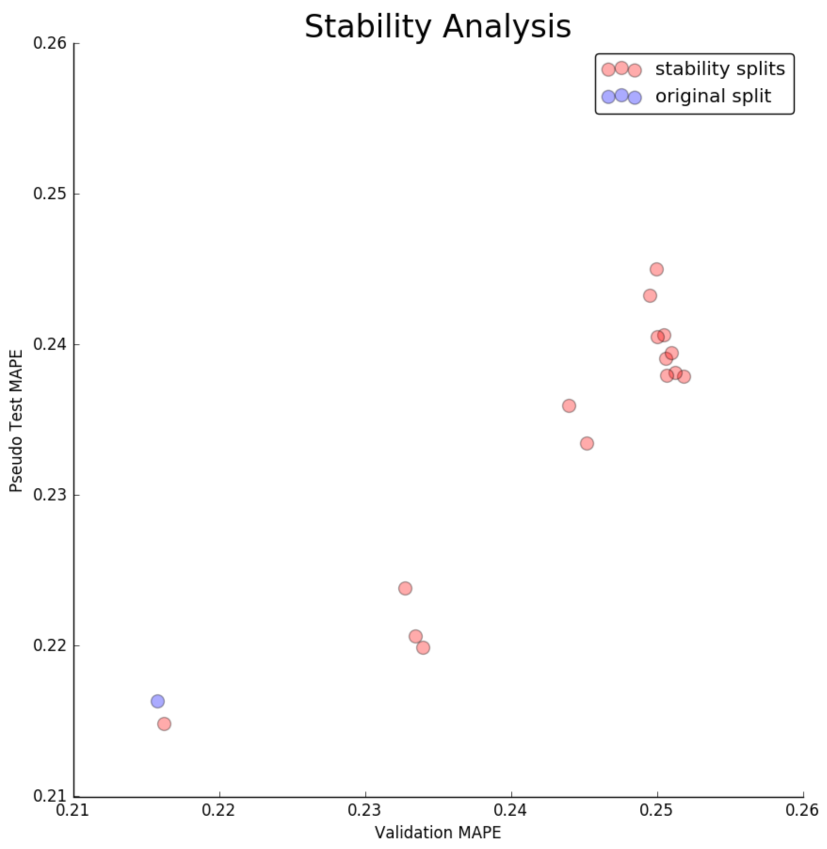

Figure 6: Pseudo test MAPE versus Validation MAPE obtained for various train test splits are shown in red. For comparison the original train-test split on which the model was constructed is shown in blue.

The sensitivity to train-test split is very high. This is evidence of a model that does not generalize well. As to why the model does not generalize, I propose no hypothesis. Its somewhat disappointing to me, simply because I put quite a bit of effort into this project. But as far as the DiTech challenge goes, one does not need the model to generalize well in order to win, only to predict well for the specific train and test sets provided. But having said that, the best MAPE achieved by this model is not competitive with world class MAPE values. I estimate that I would need a full test MAPE of 0.16 or so on these data in order to compete with the best of the best. This model seems to hit the proverbial wall around MAPE 0.21—much too high. One conclusion that could be drawn from this is that you're better off sticking to tools better sited to the task at hand (the Kaggle standard is boosted trees for regression) rather than experimenting with fascinating but complicated artificial neural networks if you want to win a challenge of the DiTech type.

Justification

The artificial neural network with optimized hyper-parameters significantly outperforms the guess-1 benchmark. This is not an insubstantial accomplishment, considering that the MAPE function is not overly friendly toward gradient descent. Thus the model has achieved the main goal I set for it. Having said that, it's not a world-class predictor of gaps in the DiTech challenge, and it is not stable with respect to new training data: if one were to implement this ANN in practice, one would have to optimize hyper-parameters all over again as new test data came in.

Conclusion

Free-Form Visualization

One thing I would like to know is how the model performs on a district level. Recall that DiTech threw us a curve ball when they declined to provide traffic data for District 54. My solution was to use as a proxy an average of the traffic data for the three districts most similar to District54 in terms of DiDi Chuxing supply, demand and gap. It would also be interesting to see how the model performs in terms of DiDi Chuxing commercial volume—does the model predict better or worse in districts with a lot of ride sharing?

Figure HHH shows the district level MAPE versus the district average gap for all 66 districts in the DiTech data. The model is clearly having difficulty predicting large gaps accurately. This makes sense, as the guess-1 approach also works best for small average gaps—guessing 1 really works well when the typical gap is 1 or 0. The cause of the poor performance at high gaps can only be speculated. Give that the model trims off most of the input features, albeit the ones deemed least important, it may simply be that the subtleties that would lead to better predictions for high gaps were eliminated from

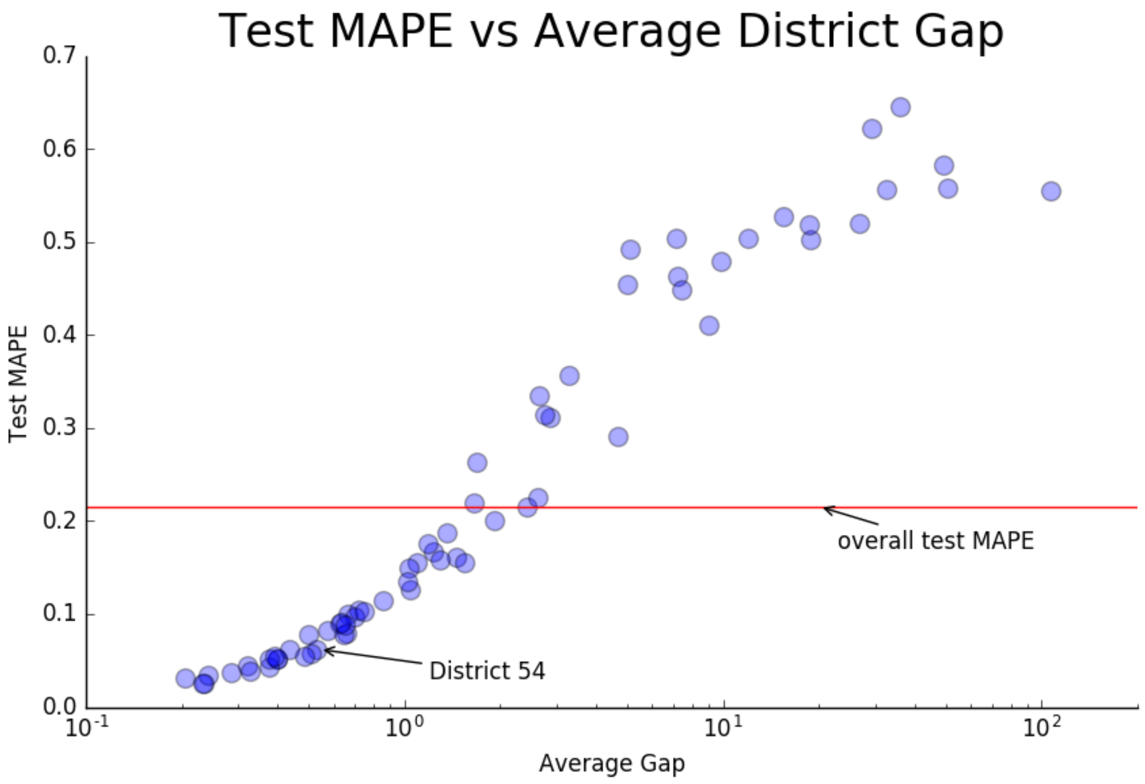

Figure 7: District-level test MAPE versus average district gap.

consideration a priori. Note that the prediction for District 54—the district for which traffic data was withheld—shows solid performance and does not stand out in the graph; thus the District 54 traffic approximation was not obviously detrimental.

Reflection

A neural network was constructed to predict the supply gap for the data in the DiTech challenge. The network was limited to three hidden layers. Optimal hyper-parameters were sought out using grid search, and further optimized using the described approach of perpetual search. The best known model substantially outperforms the benchmark guess-1 strategy. But the best model is not world-class caliber, and is very sensitive to training data. The model has difficulty predicting gaps when the gap is large. This is perhaps a sign that too much filtering of feature data was undertaken. At the present time the model is not robust enough for general purpose applicability, however in the realm of the DiTech challenge it must be deemed a moderate success.

This has been a very interesting project. I learned a great deal about neural networks, as well as manipulating and cleansing raw data in preparation for learning. Getting the neural network to learn was extremely difficult. It took a lot of playing around with hyper-parameters to find ones that worked. Although I didn't discuss this in the text, the choice of initial weights and biases is also very important.

When I started this project I was more interested in leaning about and building an ANN than winning the DiTech challenge. In that sense the project was a considerable success as I feel I learned a great deal about ANNs. It was also a good lesson for any machine learning engineer: use the right tool for the job. The next time I enter a Kaggle regression challenge I will stick to ensembling boosted trees.

Improvement

At the present time it is not clear to me why the model fails to predict better, nor why it is so sensitive to training data. Perhaps it would be better with substantially more input nodes and more hidden layers; alas the limits of my old laptop prevent me from investigating more sophisticated models. Or perhaps ANNs are not ideal for this sort of regression task—they're more often used for classification, such as the famous MNIST handwritten digit image dataset.

One obvious thing that I chose to overlook for purposes of limiting the scope of this work is the technique of dropout. This technique is known to strengthen connections between nodes and make the overall model more robust to changes in its input; it does this by dropping out a fraction of the nodes during each training iteration. Thus redundancy is worked into the model. This is one of the first things I would do if I were to continue this investigation.

It's clear that better solutions to the DiTech challenge exist. The best ones are most likely ensembles of different techniques, in which boosted trees play a key role. I am confident as well that better neural networks can be constructed. Any of the techniques discussed above are viable candidates for improvement. An investigation into sensitivity to training data and initial weights and biases would be worthwhile projects of their own.

References

- DiTech Challenge
- Deep Learning, Udacity
- Michael A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015
- Tilmann Gneiting, Making and Evaluating Point Forecasts, 2010