DeepSD: Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks

Dong Wang[1,2] Wei Cao[1,2] Jian Li[1] Jieping Ye[2]

[1] Institute for Interdisciplinary Information Sciences, Tsinghua University
[2] Bigdata Research Lab, Didi Chuxing
{cao-w13@mails, wang-dong12@mails, lijian83@mail}.tsinghua.edu.cn
yejieping@didichuxing.com

***Abstract*—The online car-hailing service has gained great popularity all over the world. As more passengers and more drivers use the service, it becomes increasingly more important for the the car-hailing service providers to effectively schedule the drivers to minimize the waiting time of passengers and maximize the driver utilization, thus to improve the overall user experience. In this paper, we study the problem of predicting the real-time car-hailing supply-demand, which is one of the most important component of an effective scheduling system. Our objective is to predict the gap between the car-hailing supply and demand in a certain area in the next few minutes. Based on the prediction, we can balance the supply-demands by scheduling the drivers in advance. We present an end-to-end framework called *Deep Supply-Demand (DeepSD)* using a novel deep neural network structure. Our approach can automatically discover complicated supply-demand patterns from the car-hailing service data while only requires a minimal amount hand-crafted features. Moreover, our framework is highly flexible and extendable. Based on our framework, it is very easy to utilize multiple data sources (e.g., car-hailing orders, weather and traffic data) to achieve a high accuracy. We conduct extensive experimental evaluations, which show that our framework provides more accurate prediction results than the existing methods.**

I. Introduction

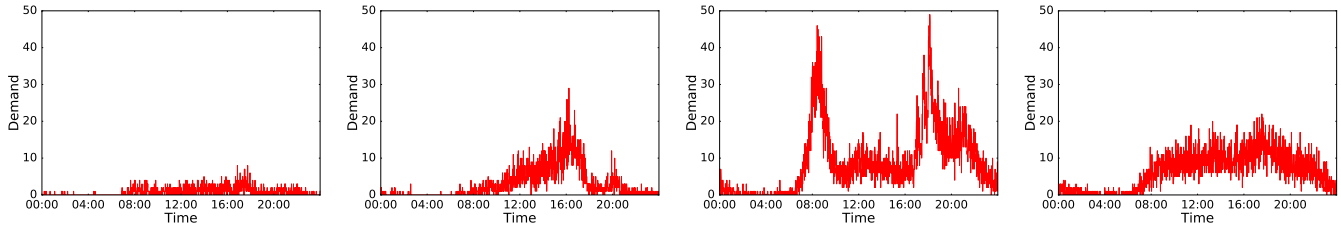
Online car-hailing apps/platforms have emerged as a novel and popular means to provide on-demand transportation service via mobile apps. To hire a vehicle, a passenger simply types in her/his desired pick up location and destination in the app and sends the request to the service provider, who either forwards the request to some drivers close to the pick up location, or directly schedule a close-by driver to take the order. Comparing with the traditional transportation such as the subways and buses, the online car-hailing service is much more convenient and flexible for the passengers. Furthermore, by incentivizing private cars owners to provide car-hailing services, it promotes the sharing economy and enlarges the transportation capacities of the cities. Several car-hailing mobile apps have gained great popularities all over the world, such as Uber, Didi, and Lyft. Large number of passengers are served and volume of car-hailing orders are generated routinely every day. For example, Didi, the largest online car-hailing service provider in China, handles around 11 million orders per day all over China. [1]

As a large number of drivers and passengers use the service, several issues arise: Sometimes, some drivers experience a hard time to get any request since few people nearby call the rides; At the same time, it is very difficult for some passengers to get the ride, in bad weather or rush hours, because the demand in the surrounding areas significantly exceeds the supply. Hence, it is an very important yet challenging task for the service providers to schedule the drivers in order to minimize the waiting time of passengers and maximize the driver utilization. One of the most important ingredient of an effective driver scheduler is the *supply-demand prediction*. If one could predict/estimate how many passengers need the ride service in a certain area in some future time slot and how many close-by drivers are available, it is possible to balance the supply-demands in advance by dispatching the cars, dynamically adjusting the price, or recommending popular pick-up locations to some drivers.

In this paper, we study the problem of predicting the car-hailing supply-demand. More concretely, our goal is to predict the gap between the car-hailing supply and demand (i.e., $\max(0; \text{demand} - \text{supply})$) for a certain area in the next few minutes. Our research is conducted based on the online car-hailing order data of Didi. To motivate our approach, we first present some challenges of the problem and discuss the drawback of the current standard practice for such problem.

- The car-hailing supply-demand varies dynamically due to different geographic locations and time intervals. For example, in the morning the demand tends to surge in the residential areas whereas in the evening the demand usually tends to surge in the business areas. Furthermore, the supply-demand patterns under different days of a week can be extremely different. Prior work usually distinguishes different geographic locations, time intervals or days of week and build several sub-models respectively [1]–[4]. Treating the order data separately and creating many sub-models are tedious, and may suffer from the lack of training data since each sub-model is trained over a small part of data.
- The order data contains multiple attributes such as the timestamp, passenger ID, start location, destination etc, as well as several "environment" factors, such as the traffic condition, weather condition etc. These attributes together provide a wealth of information for supply-demand prediction. However, it is nontrivial how to use all the attributes in a unified model. Currently, the most standard approach is to come up with many "hand-crafted" features (i.e., feature engineering), and fit them into an off-the-shelf learning algorithm such as logistic regression or random forest. However, feature engineering typically requires substantial human efforts (it is not unusual to see data science/ machine

[1] Homepage: http://www.xiaojukeji.com/en/index.html



(a) First area on March 9th (b) First area on March 13th (c) Second area on March 9th (d) Second area on March 13th

Fig. 1. Car-hailing demands under four different situations.

learning practitioners creating hundreds different features in order to achieve a competitive performance) and there is little general principle how this should be done. Some prior work only keeps a subset of attributes for training, such as the timestamp, start location and drops other attributes [2]–[6]. While this makes the training easier, discarding the attributes leads to the *information loss* and reduces the prediction accuracy.

To provide some intuitions for the readers and to illustrate the challenges, we provide an example in Fig.1.

Example 1: Fig. 1 shows the demand curves for two areas on March 9th (Wednesday) and March 13th (Sunday). From the figure, we can see very different patterns under different timeslots for the two areas. For the first area, few people require the car-hailing services on Wednesday. However, the demand increased sharply on Sunday. Such pattern usually occurs in the entertainment area. For the second area, we observe a heavy demand on Wednesday, especially during two peak hours around 8 o'clock and 19 o'clock (which are the commute times for most people during the weekdays). On Sunday, the demand of car-hailing services on this area reduced significantly. Moreover, the supply-demand patterns change from day to day. There are many other complicated factors that can affect the pattern, and it is impossible to list them exhaustively. Hence, simply using the average value of historic data or empirical supply-demand patterns can lead to quite inaccurate prediction results, which we show in our experiments (see Section VI). ■

To address the above challenges, we propose an end-to-end framework for supply-demand prediction, called *Deep Supply-Demand (DeepSD)*. Our framework is based on the deep learning technique, which has successfully demonstrated its power in a number of application domains such as vision, speech and natural language processing [7]–[9]. In particular, we develop a new neural network architecture, that is tailored to our supply-demand prediction task. Our model demonstrates a high prediction accuracy, requires little hand-crafted feature, and can be easily extended to incorporate new dataset and features. A preliminary version of our model achieved the 2nd place among 1648 teams in the Didi supply-demand prediction competition.[2] Our technical contributions are summarized below:

- We proposed an end-to-end framework based on a deep learning approach. Our approach can automatically learn the patterns across different spatio-temporal attributes (e.g. geographic locations, time intervals and days of week), which allows us to process all the data in a unified model, instead of separating it into the sub-models manually. Comparing with other off-the-shelf methods (e.g., gradient boosting, random forest [10]), our model requires a minimal amount feature-engineering (i.e., hand-crafted features), but produces more accurate prediction results.
- We devise a novel neural network architecture, which is inspired by the *deep residual network* (ResNet) proposed very recently by He et al. [11] for image classification. The new network structure allows one to incorporate the "environment factor" data such as the weather and traffic data very easily into our model. On the other hand, we can easily utilize the multiple attributes contained in the order data without much information loss.
- We utilize the *embedding* method [9], [12], a popular technique used in natural language processing, to map the high dimensional features into a smaller subspace. In the experiment, we show that the embedding method enhances the prediction accuracy significantly. Furthermore, with embedding, our model also automatically discovers the similarities among the supply-demand patterns of different areas and timeslots.
- We further study the *extendability* of our model. In real applications, it is very common to incorporate new extra attributes or data sources into the already trained model. Typically we have to re-train the model from the scratch. However, the residual learning component of our model can utilize these already trained parameters by a simple *fine tuning* strategy. In the experiment, we show that the fine-tuning can accelerate the convergence rate of the model significantly.
- Finally, we conduct extensive experiments on a large scale real dataset of car-hailing orders from Didi. The experimental results show that our algorithm outperforms the existing method significantly. The prediction error of our algorithm is $11{:}9\%$ lower than the best existing method.

II. Formulation and Overview

We present a formal definition of our problem. We divide a city into N non-overlapping square areas $\mathsf{a}_1; \mathsf{a}_2; \cdots; \mathsf{a}_\mathsf{N}$ and each day into 1440 timeslots (one minute for one timeslot). Then we define the car-hailing orders in Definition 1.

Definition 1 (Car-hailing Order): A car-hailing order o is defined as a tuple: the date when the car-hailing request was sent $\mathsf{o{:}d}$, the corresponding timeslot $\mathsf{o{:}ts} \in [1; 1440]$, the passenger

[2] http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016. The preliminary model we used for the competition was almost the same as the basic version of our model described in Section IV. Our final model, described in Section V, further refines the basic model by introducing a few new ideas, and is more stable and accurate. We are currently in an effort of deploying the model and incorporate it into the scheduling system in Didi.

ID $o.pid$, the area ID of start location $o.loc_s \in [N]$ and the area ID of destination $o.loc_d \in [N]$. If the a driver answered the request, we say it is a *valid order*. Otherwise, if no driver answered the request, we say it is an *invalid order*.

Definition 2 (Supply-demand Gap): For the d-th day, the supply-demand gap of the time interval $[t; t + C)$ in area a is defined as the total amount of invalid orders in this time interval. We fix the constant C to be 10 in this paper[3] and we denote the corresponding gap as $\mathrm{gap}_a^{d;t}$.

We further collected the weather condition data and traffic condition data of different areas which we refer to as the *environment data*.

Definition 3 (Weather Condition): For a specific area a at timeslot t in the d-th day, the weather condition (denoted as wc) is defined as a tuple: the weather type (e.g., sunny, rainy, cloudy etc.) $\mathrm{wc}.type$, the temperature $\mathrm{wc}.temp$ and the PM2.5 $\mathrm{wc}.pm$. All areas share the same weather condition at the same timeslot.

Definition 4 (Traffic Condition): The traffic condition describes the congestion level of road segments in each area: from Level 1 (most congested) to Level 4 (least congested). For a specific area a at timeslot t in the d-th day, the traffic condition is defined as a quadruple: the total amount of road segments in area a under four congestion levels.

Now, we can define our problem as below.

Problem Suppose the current date is the d-th day and the current time slot is t. Given the past order data and the past environment data, our goal is to predict the supply-demand gap $\mathrm{gap}_a^{d;t}$ for every area a, i.e., the supply-demand gap in the next 10 minutes.

Our paper is organized as follows. We first present several preliminaries in Section III, including the embedding method and the residual network which we mentioned in the introduction part. Then, we show a basic version of our model in Section IV. The basic version adopt a simple network structure and only uses the order data in the current day. In Section V, we present an advanced version which is an extension of the basic version. The advanced version utilize more attributes in the order data and it further incorporates the *historical order data* to enhance the prediction accuracy. In Section VI we conduct extensive experiment evaluations. Finally, we briefly review some related work in Section VII and conclude our paper in Section VIII.

III. Preliminary

A. Embedding Method

Embedding method is a feature learning technique which is widely used in Deep Learning, especially in natural language processing (NLP) tasks [9], [12], [13]. It is a parameterized function which maps the categorical values to the real numbers.

Specifically, neural networks treat every input as a real value. A simple way to transform a categorical value to real numbers is *one-hot representation*. For example, suppose the value of a categorical feature is 3 and the corresponding vocabulary size (highest possible value) is 5. Then, its one-hot representation is $(0; 0; 1; 0; 0)$. However, using such representation can be computationally expensive when the vocabulary size is huge. Moreover, such representation does not capture the similarity between different categories.

The embedding method overcomes such issues by mapping each categorical value into a low-dimensional space (relative to the vocabulary size). For example, the categorical value with one-hot representation equal to $(0; 0; 1; 0; 0)$ can be represented as the form of $(0{:}2; 1{:}4; 0{:}5)$. Formally, for each categorical feature, we build an embedding layer with parameter matrix $W \in R^{I \times O}$. Here I is the vocabulary size of input categorical value and O is the dimension of the output space (which we refer to as the embedding space). For a specific categorical value $i \in [I]$, we use $\mathrm{onehot}(i) \in R^{1 \times I}$ to denote its one-hot representation. Then, its embedded vector $\mathrm{embed}(i) \in R^{1 \times O}$ is equal to $\mathrm{onehot}(i)$ multiply the matrix W, i.e., the i-th row of matrix W. We usually have that $O \ll I$. Thus, even the vocabulary size is very large, we can still handle these categorical values efficiently. Furthermore, an important property of embedding method is that the categorical values with similar semantic meaning are usually very close in the embedding space. For example in our problem, we find that if two different areas share similar supply-demand patterns, then their area IDs are close in the embedding space. See Section VI for the details. We stress that the parameter matrix W in the embedding layer is optimized with other parameters in the network. We do not train the Embedding Layers separately.

Many non-trivial tasks have greatly benefited from very deep neural networks, which reveals that network depth is of crucial importance [14]–[16]. However, an obstacle to train a very deep model is the gradient *vanishing/exploding* problem, i.e., the gradient vanishes or explodes after passing through several layers during the backpropagation [17], [18]. To overcome such issue in deep neural networks, He et al. [11] proposed a new network architecture called the residual network (ResNet), which allows one to train very deep convolutional neural networks successfully.

The residual learning adds the *shortcut connections* (dashed line in Fig. **??**) and *direct connections* (solid line in Fig. **??**) between different layers. Thus, the input vector can be directly passed to the following layers though the shortcut connections. For example, in Fig. **??**, we use x to denote the input vector and $\mathcal{H}(x)$ to denote the desired mapping after two stacked layers. In the residual network, instead of learning the mapping function $\mathcal{H}(x)$ directly, we learn the residual mapping $\mathcal{F}(x) = \mathcal{H}(x) - x$ and broadcast $\mathcal{F}(x) + x$ to the following layers. It has been shown that optimizing the residual mapping is much easier than optimizing the original mapping [11], which is the key to the success of deep residual network.

IV. Basic Version

We first present the basic version of our model in this section. In Section V, we extend the basic version with a few new ideas, and present the advanced version of our model. The basic model consists of three parts. Each part consists of one or more blocks (recall that the block is the base unit of our model). In Section IV-A, we first process the "identity features" (area ID, timeslot, day of week) in the identity part. Next in Section IV-B, we describe the order part which processes the order data. The order part is the most important part of our model. In Section IV-C, we present the environment part. The environment part processes the weather data and traffic data.

[3]The constant 10 (minutes) is due to the business requirement. It can be replaced by any other constant.

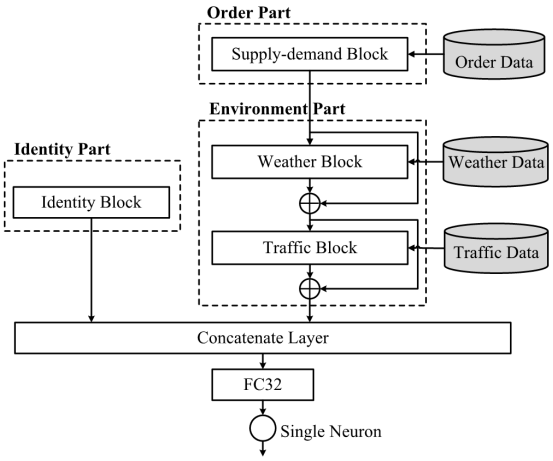


Fig. 2. Structure of basic DeepSD

Finally, in Section IV-D, we illustrate how we connect different blocks. The structure of our basic model is shown in Fig. 2.

A. Identity Part

The identity part consists of one block called the identity block. We call the features which identify the data item we want to predict as the "identity features". The identity features include the ID of area AreaID, the timeslot TimeID and the day of week (Monday, Tuesday, ..., Sunday) WeekID. For example, if we want to predict the supply-demand gap of area a in the time interval $[t; t + 10)$ in the d-th day and that day is Monday, then we have that $\text{AreaID} = a$, $\text{TimeID} = t$ and $\text{WeekID} = 0$.

Note that the features in the identity block are categorical. As we mentioned in Section III, we can either using the one-hot representation or embedding representation to transform the categorical values to real numbers. In our problem, since the vocabularies of AreaID and TimeID are very large, the one-hot representation leads to a high cost. Moreover, the one-hot representation treats the different areas or timeslots independently. However, we find that different areas at different time can share similar supply-demand patterns, especially when they are spatio-temporally close. For example, the demands of car-hailing are usually very heavy for all the areas around the business center at 19:00. Clustering these similar data items helps enhance the prediction accuracy. In our model, we use the embedding method to reduce the feature dimensions and discover the similarities among different areas and timeslots.

Formally, the structure of the identity part is shown in Fig. 3. We use three Embedding Layers to embed AreaID, TimeID and WeekID respectively. We then concatenate the outputs of three Embedding Layers by a *Concatenate Layer*. The Concatenate Layer takes a list of vectors as the input and simply outputs the concatenation of the vectors. We use the output of the Concatenate Layer as the output of the identity block, denoted as V_{id}. Furthermore, we stress that prior work [1], [3], [19] also clusters the similar data items to enhance the prediction accuracy. However, they treat the clustering stage as a separate sub-task and they need to manually design the distance measure, which is a non-trivial task. Our model is end-to-end and we can optimize the embedding parameters together with other parameters in the neural network. Hence

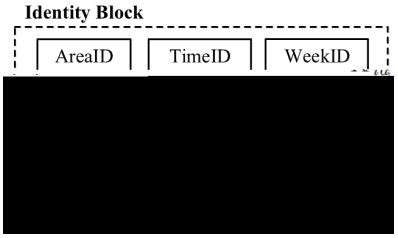


Fig. 3. Identity Block

we do not need to design any distance measure separately. The parameters are optimized through backpropagation towards minimizing the final prediction loss.

B. Order Part

The order part in the basic version consists only one block called the supply-demand block. The supply-demand block can be regarded as a three layer perception, which processes the order data. For a specific area a, to predict the supply-demand gap $\text{gap}_a^{d;t}$ of the time interval $[t; t + 10)$ in the d-th day, we consider the order set with timestamp in $[t - L; t)$ of the d-th day, which we denote as $S^{d;t}$. Here L is the window size which is specified as 20 minutes in the experiment section (Section VI). We then aggregate $S^{d;t}$ into a *real-time supply-demand vector*.

Definition 5 (Real-time supply-demand vector): For a specific area a, we define the real-time supply-demand vector in the d-th day at timeslot t as $V_{sd}^{d;t}$. $V_{sd}^{d;t}$ is a $2L$-dimensional vector, which consists of two parts. We denote the first L dimensions of $V_{sd}^{d;t}$ as $V_{A\,sd}^{d;t}$. The ℓ-th dimension of $V_{A\,sd}^{d;t}$ is defined as:

$$V_{A\,sd}^{d;t}(\ell) = |\{o \mid o \text{ is valid} \wedge o \in S^{d;t} \wedge o.ts = t - \ell\}|$$

In another word, $V_{A\,sd}^{d;t}(\ell)$ describes the amount of valid orders at $t - \ell$ in the current day. Similarly, we define the remaining part as $V_{B\,sd}^{d;t}$ which corresponds to the invalid orders in the previous L minutes. ■

We use $V_{sd}^{d;t}$ as the *Input Layer* of the supply-demand block. We then pass $V_{sd}^{d;t}$ through two *Fully-Connected* (abbr. FC) layers. A Fully-Connected Layer with input x is defined as

$$\text{FC}_{sz}(x) = f(x \cdot W + b)$$

where sz is the corresponding output size, $W; b$ are the parameters and f is the activation function which we specify in Section VI. We use FC_{64} as the first Fully-Connected Layer and the FC_{32} as the second Fully-Connected Layer. The output of the supply-demand block is the output of FC_{32}, denoted as V_{sd}. See Fig 4 for illustration.

C. Environment Part

In the environment part, we incorporate the information from the weather data through adding the weather block to the network and the traffic data through the traffic block.

For the weather condition, we first create a *weather condition vector* $V_{wc}^{d;t}$. We show the structure of the weather block in Fig. 5. The vector $V_{wc}^{d;t}$ consists of L parts. For a specific $\ell \in [L]$, we have the weather condition wc at timeslot

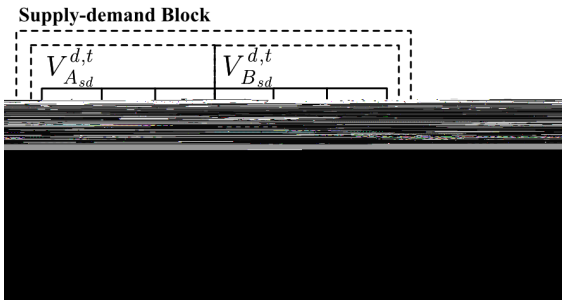


Fig. 4. Structure of Supply-demand Block

$t-‘$ in the d-th day and we embed the weather type wc:type into a low dimensional space. Then the ‘-th part of $V_{wc}^{d;t}$ is defined as the concatenation of the embedded weather type wc:type, the temperature wc:temp and the PM 2.5 wc:pm. Furthermore, note that the weather block also receives the output of the supply-demand block X_{sd} through a direct connection. We concatenate X_{sd} and $V_{wc}^{d;t}$ by a Concatenate Layer and pass the output of the Concatenate Layer through two Fully-Connected layers FC_{64} and FC_{32}. We denote the output of FC_{32} as R_{wc}. Then, the output of the weather block X_{wc} is defined as:

$$X_{wc} = X_{sd} \oplus R_{wc}$$

where $\oplus$ is the element-wise add operation and X_{sd} is obtained through the shortcut connection.

Note that the structure we used here is similar with ResNet as we mentioned in Section III. However, there are two main differences between our model and ResNet. First, instead of adding shortcut connections between layers, we add the shortcut connections between different blocks. Second, in ResNet, a layer only receives the input from previous layers through a direct connection whereas in our model a block receives the inputs from both the previous block and the dataset. Such structure on one hand is more suitable for handling the data from multiple sources. On the other hand, we show that in Section VI-H, such structure is highly extendable. We can easily incorporate new datasets or attributes into our model based on such structure.

For the traffic condition, recall that at each timeslot the traffic condition of a specific area can be represented as the total amount of road segments in four different congestion levels. We thus create a *traffic condition vector* $V_{tc}^{d;t}$ with L parts. Each part consists of four real values corresponding to the traffic condition at that time slot. We construct the traffic block in the same way as we construct the weather block. Then, we use $X_{tc} = X_{wc} \oplus R_{tc}$ as the output of the traffic block, as shown in Fig. 5.

D. Block Connections

We then connect all the blocks. Note that the supply-demand block, the weather block and the traffic block are already connected through the residual learning. The output vector of these stacked blocks is X_{tc}. We then concatenate the output of the identity block X_{id} and X_{tc} with a Concatenate Layer. We append a Fully-Connected Layer FC_{32} and a single neuron after the Concatenate Layer. The single neuron finally outputs the predicted supply-demand gap with the linear activation function, as shown in Fig. 2. We stress that our model is end-to-end, once we obtain the predicted value, we can calculate the loss based on the loss function and update each parameter with its gradient through backpropagation.

Fig. 5. Weather Block and Traffic Block

We further illustrate the intuition of our model. To predict the supply-demand gap, the most relevant and important data is the car-hailing order data. We use the supply-demand block to learn the useful feature vector from the order data. In our model, the learnt feature corresponds to the output of the supply-demand block, X_{sd}. The environment data can be regarded as the supplementary of the learnt features. Thus, we add the weather block to extract the residual R_{wc} and adjust the previous learnt features by adding R_{wc} to X_{sd}. The same argument holds for the traffic block.

V. Advanced Version

In this section, we present an advanced version of our model. Comparing with the basic model, the advanced model replaces the *order part* in Fig. 2 with an *extended order part* as shown in Fig. 6, which is composed of three blocks. The first block *extended supply-demand block* extends the original *supply-demand block* with a well-designed structure. Such structure enables our model to learn the dependence of the historical supply-demand over different days automatically, which we present in Section V-A. In Section V-B, we present the remaining two blocks, the *last call block* and the *waiting time block*, which have the same structure as the extended supply-demand block. Comparing with the basic version where we only use the number of orders, the new blocks contains passenger information as well.

A. Extended supply-demand block

Recall that in the basic version, we use the real-time supply-demand vector $V_{sd}^{d;t}$ to predict the supply-demand gap. In the extended order block, we further incorporate the historical order data to enhance the prediction accuracy, i.e., the car-hailing orders with date prior to the d-th day. We present the extended supply-demand block in two stages. In the first stage, we obtain an *empirical supply-demand vector* in time interval $[t-L;t)$

in the d-th day. Such empirical supply-demand vector is an estimation of $V_{sd}^{d;t}$ based on the historical order data. In the second stage, we use the real-time supply-demand vector and the empirical supply-demand vector to construct our extended supply-demand block.

1) First Stage: We first extract the empirical supply-demand vector in $[t-L;t)$ in the d-th day, denoted as $E_{sd}^{d;t}$. It has been shown that due to the regularity of human mobility, the patterns in the traffic system usually show a strong periodicity in time on a weekly basis [1]–[3], [5], [20], [21]. However, for different days of week, the supply-demand patterns can be very different. For example, in Huilongguan, a district in Beijing where most of IT employees live, the demand of car-hailing services in Monday morning is usually much more than that in Sunday morning. Motived by this, we first consider the historical supply-demands in different days of week. Formally, we use $\mathcal{M}$ to denote all the Mondays prior to the d-th day. For each day $m \in \mathcal{M}$, we calculate the corresponding real-time supply-demand vector in that day, denoted as $V_{sd}^{m;t}$ as we defined in Definition 5. We average the vectors $V_{sd}^{m;t}$ for all $m \in \mathcal{M}$. We call such average the *historical supply-demand vector on Monday*, denoted as $H_{sd}^{(Mon);d;t}$. Thus, we have that,

$$H_{sd}^{(Mon);d;t} = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} V_{sd}^{m;t} :$$

Similarly, we define the historical supply-demand vector on the other days of week: $H_{sd}^{(Tue);d;t}$, $H_{sd}^{(Wed);d;t}$, $:::$, $H_{sd}^{(Sun);d;t}$.

The empirical supply-demand vector $E_{sd}^{d;t}$ is defined as a weighted combination of $\{H_{sd}^{(Mon);d;t}; ::: ; H_{sd}^{(Sun);d;t}\}$. We refer to the weight vector as *combining weights of different weekdays*, denoted as p. In our model, such weight vector p is automatically learnt by the neural network according to the current AreaID and WeekID. The network structure is shown in Fig. 7. We first embed the current AreaID and WeekID into a low-dimensional space. We concatenate the embedded vectors and pass it into a Softmax Layer. A Softmax Layer takes the concatenation x as the input and outputs the weight vector p by

$$p^{(i)} = \frac{e^{x \cdot W_{:i}}}{\sum_j e^{x \cdot W_{:j}}}; \forall i = 1 ::: 7$$

where $W_{:j}$ is the j-th column of the parameter matrix W in the Softmax Layer. Then, we have that

$$E_{sd}^{d;t} = p^{(1)} \cdot H_{sd}^{(Mon);d;t} + ::: + p^{(7)} \cdot H_{sd}^{(Sun);d;t} : \quad (1)$$

We stress that most of prior work simply distinguish the historical data in weekdays and weekends separately [1]–[3], [6], [22]. However, on one hand, such method may suffer from the lack of training data. We only utilizes part of the data when we calculate the historical supply-demand vector. On the other hand, different areas can show different dependences over days of week. For example, in our experiment (Section VI), we find that for some areas, the supply-demands in Tuesdays are very different from the other days of week. Thus, to predict the supply-demand in Tuesday, we mainly consider the historical data in the past Tuesdays. For some other areas, the supply-demands in all the days of week are very similar. In this case, taking all the historical data into consideration leads to a more accurate result. Obviously, simply separating the historical data in weekdays and weekends can not such patterns.

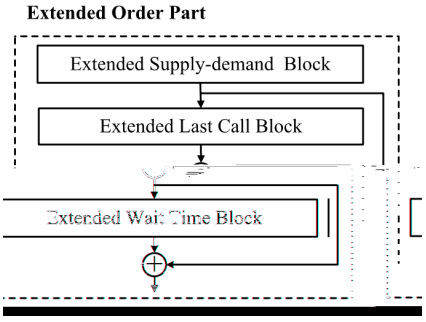


Fig. 6. Extended Order Part

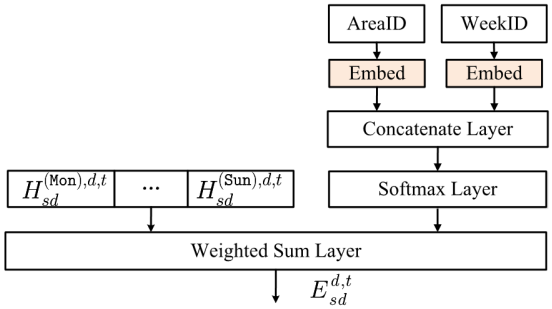


Fig. 7. Historical supply-demand vector H_{sd}

2) Second Stage: Next, we use the obtained empirical supply-demand vector and real-time supply-demand vector to construct our block. First, using the same method as we obtain $E_{sd}^{d;t}$, we calculate another empirical supply-demand vector in time interval $[t-L+10;t+10)$ in the current day, denoted as $E_{sd}^{d;t+10}$. Note that $E_{sd}^{d;t+10}$ is the empirical estimation of the real-time supply-demand vector $V_{sd}^{d;t+10}$. If we can estimate $V_{sd}^{d;t+10}$ accurately, we can easily predict the currently supply-demand gap.

In our model, we use the empirical estimations $E_{sd}^{d;t}$, $E_{sd}^{d;t+10}$ and the real-time supply-demand vector $V_{sd}^{d;t}$ to estimate $V_{sd}^{d;t+10}$. We first use the Fully-Connection Layers to project these three vectors onto the same low-dimensional space (in our experiment we fix the dimensionality to be 16). We denote the projected vectors as $Proj(V_{sd}^{d;t})$, $Proj(E_{sd}^{d;t})$ and $Proj(E_{sd}^{d;t+10})$. Instead of estimating $V_{sd}^{d;t+10}$ directly, we estimate the projection of $V_{sd}^{d;t+10}$. We denoted the estimated projection as $\hat{Proj}(V_{sd}^{d;t+10})$ and we have that,

$$\hat{Proj}(V_{sd}^{d;t+10}) = Proj(V_{sd}^{d;t}) - Proj(E_{sd}^{d;t}) + Proj(E_{sd}^{d;t+10}):$$

Finally, we concatenate $Proj(V_{sd}^{d;t})$, $Proj(E_{sd}^{d;t})$, $Proj(E_{sd}^{d;t+10})$, $\hat{Proj}(V_{sd}^{d;t+10})$, with a Concatenate Layer and pass it through two Fully-Connected layers FC_{64} and FC_{32}. We use the output of FC_{32} as the output of the extended supply-demand block. See Fig. 8 for an illustration.

We explain the reason that we estimate $V_{sd}^{d;t+10}$ in such way. The vector $Proj(V_{sd}^{d;t}) - Proj(E_{sd}^{d;t})$ indicates how the real-time supply-demand of $[t-L;t)$ deviates from its empirical estimation. We thus estimate $Proj(V_{sd}^{d;t+10})$ by adding such deviation to the projection of empirical estimation $Proj(E_{sd}^{d;t+10})$.

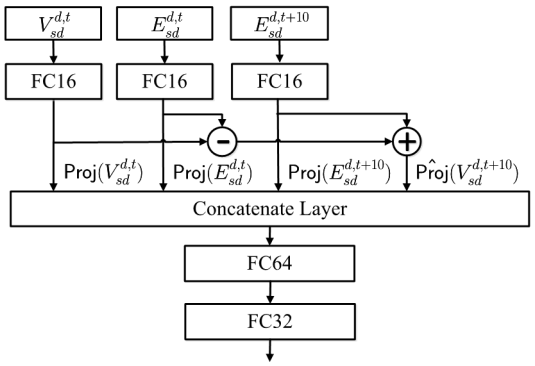


Fig. 8. Extended Supply-demand Block

Moreover, the projection operation on one hand reduce the dimension of each supply-demand vector from $2L$ to 16. On the other hand, we find that using the projection operation in our experiment makes our model more stable.

B. Last Call Block and Waiting Time Block

In this section, we present two additional blocks, called the last call block and the waiting time block. Note that the order data contains multiple attributes. However, when calculating the supply-demand vector, we did not consider the attribute o:pid. Thus, the supply-demand vector $V_{sd}^{d;t}$ does not contain any "passenger information". From $V_{sd}^{d;t}$, we can not answer the questions such as "how many unique passengers did not get the rides in the last 5 minutes" or "how many passengers waited for more than 3 minutes" etc. However, we find that the passenger information is also very important to supply-demand gap prediction. For example, if many passengers failed on calling the rides or waited for a long time, it reflects that the current demand exceeds the supply significantly which can lead to a large supply-demand gap in the next few minutes. We use the last call block and the waiting time block to provide the passenger information. Both of these two blocks have the same structure as the extended supply-demand block. In another word, we just replace the real-time supply-demand vector $V_{sd}^{d;t}$ in the extended supply-demand vector with the *real-time last call vector* and *real-time waiting time vector*.

For the last call block, we define the *last call vector* as follow.

Definition 6 (Real-time last call vector): For a specific area a at timeslot t in the d-th day, we first pick out the *last call orders* in $[t-L;t)$ for all passengers, (i.e. for a specific passenger pid, we only keep the last order sent by pid), and denote the order set as $S_L{}^{d;t}$. Then, the real-time last call vector $V_{lc}^{d;t}$ is defined as a $2L$-dimensional vector. We denote the first L dimension as $V_{A\,lc}^{d;t}$. For the $'$-th dimension of $V_{A\,lc}^{d;t}$, we have that

$$V_{A\,lc}^{d;t}(') = |\{\text{pid} \mid \exists o \in S_L{}^{d;t} \text{ s:t: o is valid} \wedge o{:}\text{pid} = \text{pid} \wedge o{:}\text{ts} = t - ' \}|$$

$V_{A\,lc}(')^{d;t}$ describes the amount of passengers whose last call is at $t-'$ and she/he successfully got the ride. Similarly, we define $V_{B\,lc}^{d;t}$ which corresponds to the passengers who did not get the rides.■

We explain the reason that we define the real-time last call vector. In our data, we find that if a passenger failed on calling a ride, she/he is likely to sent the car-hailing request again in the next few minutes. Especially, the last calls near timeslot t are highly relevant to the supply-demand gap in $[t; t+10)$.

Based on $V_{lc}^{d;t}$, we can further obtain the empirical last call vector $E_{lc}^{d;t}$ with the same way as we obtain $E_{sd}^{d;t}$. We thus construct the extended real-time last call block with the same structure as the extended supply-demand block.

For the waiting time block, we define the *real-time waiting time vector* $V_{wt}^{d;t} \in R^{2L}$ in the same way as we defining $V_{sd}^{d;t}$ and $V_{lc}^{d;t}$.

Definition 7 (Real-time waiting time vector): For a specific area a at timeslot t in the d-th day, we define the real-time waiting time vector as $V_{wt}^{d;t}$. The $'$-th dimension in the first part $V_{A\,wt}^{d;t}$ (first L dimensions) is the total amount of passengers who waited for $'$ minutes (from her/his first call in $[t-L;t)$ to the last call) and did get the rides at last. Similarly, we define the second part $V_{B\,wt}^{d;t}$ which corresponds to the wait time of passengers who did not get the ride.

We thus construct the extended waiting time block with the same structure of the extended supply-demand block.

Finally, we connect the supply-demand block, the last call block and the waiting time block through residual learning, as shown in Fig. 6. These three blocks together form the extended order part in the advanced model. We use the extended order part to replace the original order part and we thus obtain the advanced version of DeepSD.

C. Extendability

Finally, in this section we present the extendability of our model. In real applications, it is very common to incorporate new extra attributes or data sources into the previous model. For example, imagine that we have already trained a model based on the order data and the weather data. Now we obtained the traffic data and we want to incorporate such data to enhance the prediction accuracy. Typically, we have to discard the already trained parameters and re-train the model from beginning. However, our model makes a good use of the already trained parameters. In our model, such scenario corresponds to that we have trained a model with the order block and the weather block. To incorporate the traffic data, we construct the traffic block and connect the traffic block with previous blocks through residual learning, as we show in Fig. 2. Instead of re-training the model from the scratch, we use the already trained parameters as the initialized parameters and keep optimizing the parameters of the new model through backpropagation. We refer to such strategy *fine-tuning*. In the experiment (Section VI), we show that the fine-tuning accelerates the convergence rate significantly and makes our model highly extendable.

VI. Experiment

In this section, we report our experimental results on an real dataset from Didi. We first describe the details of our dataset in Section VI-A and the experimental setting in Section VI-B. Then, we compare our models with several other most popular machine learning algorithms in Section VI-C. In Section VI-D to Section VI-F, we show the effects of different components in our model. The advanced DeepSD can automatically extract the weights to combine the features of different days of a week. We present some interesting properties of the weights in Section

VI-G. Finally, we show some results about the extendability of our model in Section VI-H.

A. Data Description

In our experiment, we use the public dataset released by Didi in the Di-tech supply-demand prediction competition[4].

The order dataset contains the car-hailing orders from Didi over more than 7 weeks of 58 square areas in Hangzhou, China. Each area is about $3\text{km} \times 3\text{km}$ large. The order dataset consists of $11,467,117$ orders. The gaps in our dataset is approximately power-law distributed. The largest gap is as large as 1434. On the other hand, around 48% of test items are supply-demand balanced, i.e., $\text{gap} = 0$. Auxiliary information include weather conditions (weather type, temperature, PM 2.5) and traffic conditions (total amount of road segments under different congestion levels in each area).

The training data is from 23th Feb to 17th March (24 days in total). To construct the training set, for each area in each training day, we generate one training item every 5 minutes from 0:20 to 24:00. Thus, we have $58(\text{areas}) \times 24(\text{days}) \times 283(\text{items}) = 393,936$ training items in total. Due to the restriction of test data, we set the window size $L = 20$.

The test data is from 18th March to 14th April (28 days in total). During the test days, the first time slot is 7:30 and the last time slot is 23:30. We select one time slot t every 2 hours from the first time slot unit the last time slot, i.e., $t =$ 7:30, 9:30, 11:30, ..., 23:30. For each time slot t, we generate one test item. We use T to denote the set of test items.

1) Error Metrics: We evaluate the predicted results using the *mean absolute error* (MAE) and the *root mean squared error* (RMSE). Formally, we use $\text{pred}_a^{d;t}$ to denote the predicted value of $\text{gap}_a^{d;t}$. Then, the mean absolute error and the root mean squared error can be computed as follows:

$$\text{MAE} = \frac{1}{|T|} \sum_{(a;d;t)\in T} \text{gap}_a^{d;t} - \text{pred}_a^{d;t}$$

$$\text{RMSE} = \sqrt{\frac{1}{|T|} \sum_{(a;d;t)\in T} \text{gap}_a^{d;t} - \text{pred}_a^{d;t}}^2:$$

B. Model Details

We describe the model setting in this section.

1) Embedding: Recall that we map all the categorical values to a low-dimensional vectors via embedding (in Section IV-A and Section IV-C). The detailed settings of different embedding layers are shown in Table I.

2) Activation Function: For all Fully-Connected layers, we use *leaky rectified linear function (LReL)* [23] as the corresponding activation function. An LReL function is defined as:

$$\text{LReL}(x) = \max\{0.01 \cdot x; x\}:$$

For the final output neuron, we simply use the linear activation.

[4]http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016&&locale=en

TABLE I. EMBEDDING SETTING

Embedding Layers	Setting	Occurred Parts
Embedding of AreaID	$R^{58} \to R^8$	Identity Part, Extended Order Part
Embedding of TimeID	$R^{1440} \to R^6$	Identity Part
Embedding of WeekID	$R^7 \to R^3$	Identity Part, Extended Order Part
Embedding of wc:type	$R^{10} \to R^3$	Environment Part

3) Optimization Method: We apply the *Adaptive Moment Estimation (Adam)* method [24] to train our model. Adam is a robust mini-batch gradient descent algorithm. We fix the batch size to be 64. To prevent overfitting, we further apply the *dropout* method [25] with probability 0.5 after each block (except the identity block).

4) Platform: Our model is trained on a GPU server with one GeForce 1080 GPU (8GB DDR5) and 24 CPU cores (2.1GHz) in Centos 6.5 platform. We implement our model with Theano 0.8.2, a widely used Deep Learning Python library [26].

C. Performance Comparison

We train both the basic model and advanced model for 50 epochs. We evaluate the model after each epoch. To make our model more robust, our final model is the average of the models in the best 10 epochs.

To illustrate the effectiveness of our model, we further compare our model with several existing methods. The parameters of all the models are fine-tuned through the grid search.

- **Empirical Average**: For a specific t in area a, we simply use the *empirical average gap* $\frac{1}{|D_{\text{train}}|} \sum_{d\in D_{\text{train}}} \text{gap}_a^{d;t}$ as the prediction for the supply-demand gap in time interval $[t; t+10)$.
- **LASSO [10]**: The Lasso is a linear model that estimates sparse coefficients. It usually produces better prediction result than simple linear regression. Since LASSO can not handle the categorical variables, we transform each categorical variable to the one-hot representation. We use the LASSO implementation from the scikit-learn library [27].
- **Gradient Boosting Decision Tree**: Gradient Boosting Decision Tree (GBDT) is a powerful ensemble method which is widely used in data mining applications. In our experiment, we use a fine-tuned and efficient GBDT implementation XGBoost [28].
- **Random Forest**: Random Forest (RF) is another widely used ensemble method which offers comparable performance with GBDT. We use the RF implementation from the scikit-learn library [27].

For fair comparisons, we use the same input features for the above methods (except empirical average) as those used in DeepSD, including:

- AreaID, TimeID, WeekID
- Real-time supply-demand vector $V_{sd}^{d;t}$; Historical supply-demand vector of different days of week $H_{sd}^{(\text{Mon});d;t}; \ldots; H_{sd}^{(\text{Sun});d;t}$.
- Real-time last call vector $V_{lc}^{d;t}$; Historical last call vector of different days of week $H_{lc}^{(\text{Mon});d;t}; \ldots; H_{lc}^{(\text{Sun});d;t}$.