A Deep Learning Approach for Estimating Inventory Rebalancing Demand in Bicycle Sharing Systems

Petar Mrazovic[1], Josep L. Larriba-Pey[2], Mihhail Matskin[3]

[1, 3]Dept. of Software and Computer Systems, KTH Royal Institute of Technology, Stockholm, Sweden
[1, 2]Dept. of Computer Architecture, UPC Polytechnic University of Catalonia, Barcelona, Spain
Email: [1]mrazovic@kth.se, [2]larri@ac.upc.edu, [3]misha@kth.se

***Abstract*—Meeting user demand is one of the most challenging problems arising in public bicycle sharing systems. Various factors, such as daily commuting patterns or topographical conditions, can lead to an unbalanced state where the numbers of rented and returned bicycles differ significantly among the stations. This can cause spatial imbalance of the bicycle inventory which becomes critical when stations run completely empty or full, and thus prevent users from renting or returning bicycles. To prevent such service disruptions, we propose to forecast user demand in terms of expected number of bicycle rentals and returns and accordingly to estimate number of bicycles that need to be manually redistributed among the stations by maintenance vehicles. As opposed to traditional solutions to this problem, which rely on short-term demand forecasts, we aim to maximise the time within which the stations remain balanced by forecasting user demand multiple steps ahead of time. We propose a multi-input multi-output deep learning model based on Long Short-Term Memory networks to forecast user demand over long future horizons. Conducted experimental study over real-world dataset confirms the efficiency and accuracy of our approach.**

I. Introduction

Over the last decade, many cities around the world have introduced public Bicycle Sharing Systems (BSSs) as a more sustainable and less dangerous alternative to motorized forms of transport. BSSs offer a "green" solution to the first-and-last mile connection problem and provide a mobility service where other modes of transport are not available. However, despite the significant benefits from BSSs, their exploitation implies various problems among which the most prominent one is to ensure high bicycle availability in order to satisfy customers' needs and meet contractual service level agreements.

BSSs consist of a collection of bicycle rental stations strategically distributed over the service area. Each station is equipped with a self-service terminal and a number of bicycle stands. Using the self-service terminal, registered user can easily pick up a bicycle from one of the station's stands, and return it (usually within a certain time frame) to any other station with an empty stand. However, over time, user behaviour results in spatial imbalance of the bicycle inventory. In other words, various factors, such as daily commuting patterns or topographical conditions, can lead to an unbalanced state where the numbers of rented and returned bicycles differ significantly among the stations. Such state becomes critical when stations run completely empty or full, and thus prevent users from renting or returning bicycles, respectively.

In order to avoid the described service disruptions, operators need to dynamically rebalance BSSs by redistributing bicycles between stations using a fleet of maintenance vehicles. This is a very challenging task which consists of estimating the stations' inventory target levels (i.e., redistribution needs) based on predicted user demand. Even though this problem has been in focus of many research works over the past several years (e.g., [1–4]), more adaptive solutions which would rely on efficient demand forecast over long future horizons have not been sufficiently studied. Given that the optimal inventory levels aim at maximising the time within which the stations remain balanced (i.e., with enough bicycles and empty stands to meet the future user demand), we propose to forecast user demand multiple time steps into the future. Such approach allows for more accurate long-term inventory planning which can lower the number of redistribution runs and thus significantly reduce the operating cost.

However, accurate multi-step-ahead time series forecasting is a very challenging problem which faces growing amount of uncertainties such as the accumulation of errors, reduced accuracy, and lack of information. While traditional strategies to multi-step-ahead forecasting rely on recursive or direct use of one-step-ahead models [5] and thus suffer from the beforementioned drawbacks, we model our task as a machine learning problem and propose a deep learning approach to solve it efficiently. We introduce multi-input multi-output (MIMO) prediction model based on Long Short-Term Memory networks (LSTMs). The proposed model successfully forecasts the user demand (in terms of number of bicycle rentals and returns) for the upcoming time intervals, based on different conditions in BSS and the demand from the past several time intervals. Finally, we demonstrate a simple method to compute inventory target levels based on the forecast user demand.

The contributions of this work are threefold. We introduce a novel framework for estimating the inventory target levels in BSSs based on multi-step-ahead demand forecasting. We propose a deep learning model based on Long Short-Term Memory networks (LSTMs) to forecast user demand multiple steps into the future. We create an evaluation dataset using the real-world bicycle trip data from New York's BSS (CitiBike) and weather data from Weather Channel's API, and conduct experimental evaluation.

The rest of this paper is organised as follows. In the next section we formally describe the problem under study. In Section II we introduce a multi-step-ahead prediction model to forecast the stations' service demands. Section V reports on the experimental studies. In Section VI we give a brief overview of the related work. Finally, we conclude and discuss future work in Section VII.

II. Problem Formulation

As illustrated in Figure 1, the proposed rebalancing framework consists of demand forecasting model (1) and redistribution model (2). Based on historical and real-time bicycle trip data, and external data such as weather forecast, demand forecasting model predicts the future user demand in terms of expected number of bicycle rentals (i.e., pick-ups) and returns

(i.e., drop-offs). In order to satisfy the predicted demand, redistribution model estimates rebalancing targets for each of the stations in BSS. Before we discuss this workflow in more details, let us formally define some of the beforementioned concepts.

Definition II.1 (Bicycle trip data)**.** *The bicycle trip data R consist of a set of bicycle usage records $r = (s_o, \tau_o, s_d, \tau_d)$ where s_o and s_d represent pick-up and drop-off station, and τ_o and τ_d represent pick-up and drop-off time, respectively.*

Since traditional bicycle-sharing scheme allows users to rent a bicycle from any particular station and to return it back at any other station within some time frame, the user demand at each station in BSS can be naturally expressed in terms of the total number of pick-ups and drop-offs per time interval. Therefore, we define station pick-up and drop-off demands as follows.

Definition II.2 (Station pick-up and drop-off demand)**.** *Station pick-up demand $\mu_i(t)$ is the total number of bicycle rentals (pick-ups) at station s_i during time interval t. Similarly, station drop-off demand $\lambda_i(t)$ is the total number of bicycle returns (drop-offs) at station s_i during time interval t.*

Notice from the above definitions that the station demands at past time intervals can be easily computed from historical bicycle trip data. For example, station pick-up demand $\mu_i(t)$ can be computed as the total number of trip records $(s_o, \tau_o, s_d, \tau_d)$ where $s_o = s_i$ and $\tau_o \in t$. On the other hand, station pick-up demand $\mu_i(t)$ can be computed as the total number of trip records where $s_d = s_i$ and $\tau_d \in t$. We can formalise this computation as follows,

$$\mu_i(t) = |\{r \mid r \in R \wedge s_o = s_i \wedge \tau_o \in t\}| \quad (1)$$

$$\lambda_i(t) = |\{r \mid r \in R \wedge s_d = s_i \wedge \tau_d \in t\}| \quad (2)$$

Using the above equations (1) and (2), demand forecasting model is able to construct time series of station demands and exploit them to forecast the future demands. Additionally, the time series can be augmented with external data such as weather reports $w(t)$, as to introduce a context which can improve forecasting accuracy (we discuss the weather data in more details in the next section).

Based on forecast pick-up and drop-off demands and stations current inventory levels (i.e., number of available bikes), redistribution model computes the number of bicycles that need to be removed from or delivered to each station, in order to balance the stations and thus meet the predicted service demands. Similar to the definition proposed by Liu et al. [2], we define the optimal inventory level as the one which maximises the time duration within which the station remains balanced, i.e., with enough bicycles and empty stands to meet the future demand.

Definition II.3 (Station rebalancing target)**.** *Let c_i be the current inventory level, and l_i the capacity of station s_i. The station rebalancing target d_i, i.e., the total number of bikes that need to be removed from or delivered to s_i, is decided by the following calculation which aims to maximise the total number of consecutive time intervals within which s_i remains balanced,*

$$\underset{d_i}{\operatorname{argmax}}\{j \mid 0 \leqslant c_i + d_i + \sum_{t=t_0}^{t_{0+j}} (\lambda_i(t) - \mu_i(t)) \leqslant l_i\} \quad (3)$$

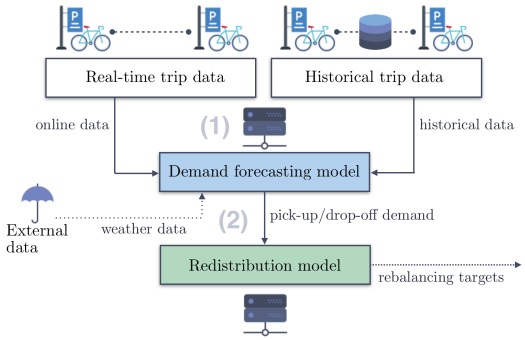


Fig. 1. System components of the proposed framework

Notice that rebalancing target d_i takes negative values when vehicles need to be removed from s_i, and positive values when they need to me delivered to s_i.

Notice again from Definition II.3 that in order to maximize the total time during which a station remains balanced, demand forecasting model needs to be able to forecast pick-up and drop-off demands multiple steps ahead of time.

Given the introduced preliminaries, we can split the problem under study into two parts: (1) forecasting future user demands, and (2) estimating station rebalancing targets.

Problem 1 (Multi-step-ahead demand forecasting)**.** *Given the pick-up demands $\mu_i(t-k+1), ..., \mu_i(t)$, drop-off demands $\lambda_i(t-k+1), ..., \lambda_i(t)$, and weather reports $w(t-k), w(t-k+1), ..., w(t)$ for the past k time intervals, the goal of the first problem is to forecast the future demands $\mu_i(t+1), ..., \mu_i(t+n)$ and $\lambda_i(t+1), ..., \lambda_i(t+n)$ for each station s_i in the next n time intervals.*

Problem 2 (Rebalancing target estimation)**.** *Given the forecast demands for n future time intervals, the goal of the follow-up problem is to compute rebalancing targets y_i for each station s_i using calculation (3) introduced by Definition II.3.*

III. Demand Forecasting

As opposed to traditional approaches to multi-step-ahead forecasting which relies on recursive or direct use of single-output models, in this section, we propose a multi-output deep learning approach which succeeds to outperform the traditional ones. We start by discussing the typical data available in BSSs and model multivariate input to our forecasting problem. Thereafter, we build a complete deep learning model based on Long Short-Term Memory (LSTM) networks.

A. Input modelling

Typical system data provided by BSS operators contains details about the trips between bicycle stations taken by individual users. In this work, we restrict ourselves to minimal feature set containing only trip's start and finish time, and IDs of origin and destination stations. This way, we can represent each trip with a 4-tuple (record) just like described in Definition II.1.

Given that each trip record represents one pick-up and one drop-off event at some points in continuous time domain, we need to discretize time into fixed-size intervals and count the total number of such events for each station. We choose 30-minute discretization interval, since in most BSSs users are allowed to rent bicycles in 30-minute increments. After applying equations (1) and (2) over the entire trip data and all

30-minute intervals within continuous data collection period, we obtain time series of the pick-up and drop-off demands, $\mu_i(t)$ and $\lambda_i(t)$, for each station s_i.

In order to apply deep learning to our forecasting problem, we transform the time series data into a set of overlapping time-lagged sequences using time-delay embedding. Assuming that the next values of the time series depend on at most k past values, and since we want to predict n steps into the future, we transform the time series into pairs of input and output vectors (sequences) $\boldsymbol{x}_t(\mu_i)$ and $\boldsymbol{y}_t(\mu_i)$ composed of k and n consecutive measurements of the original time series, i.e., $\boldsymbol{x}_t(\mu_i) = [\mu_i(t-k+1), ..., \mu_i(t-1), \mu_i(t)]$, $\boldsymbol{y}_t(\mu_i) = [\mu_i(t+1), \mu_i(t+2), ..., \mu_i(t+n)]$.

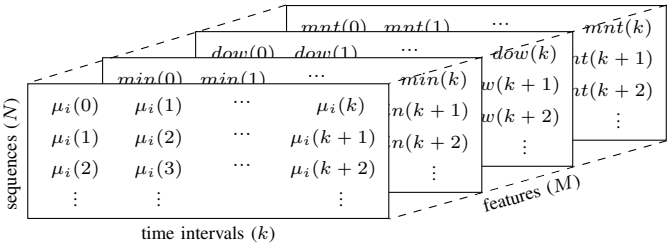
We can now construct input and output matrices $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{Y}(\mu_i)$ for each station s_i simply by stacking all of the sequences as rows, i.e.,

$$\boldsymbol{X}(\mu_i) = \begin{bmatrix} \mu_i(0) & \mu_i(1) & \dots & \mu_i(k) \\ \mu_i(1) & \mu_i(2) & \dots & \mu_i(k+1) \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$\boldsymbol{Y}(\mu_i) = \begin{bmatrix} \mu_i(k+1) & \mu_i(k+2) & \dots & \mu_i(k+n) \\ \mu_i(k+2) & \mu_i(k+3) & \dots & \mu_i(k+n+1) \\ \vdots & \vdots & & \vdots \end{bmatrix}.$$

In the same way we construct input and output matrices $\boldsymbol{X}(\lambda_i)$ and $\boldsymbol{Y}(\lambda_i)$ for each station s_i using the time series of the drop-off demand. Later in the paper we will see how different choices for embedding dimensions k and n affect the performance of the forecasting model.

Notice that for now our input data $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{X}(\lambda_i)$ is univariate, i.e., it depends on past values of a single input variable – number of pick-ups/drop-offs. However, given that the user demand varies strongly by time of day and day of week, by giving additional temporal context to the input data, we can help our model learn such dependencies and improve forecasting accuracy. Therefore, we augment the input matrices with the additional temporal variables (features). Variable $min(t)$ is introduced to represent time of day as a number of minutes from midnight to the beginning of time interval t. Days of week are represented as weekday numbers $dow(t) \in [1, 7]$, where 1 encodes Monday, and 7 encodes Sunday. Further, we also introduce variable $mnt(t) \in [1, 12]$ to represent months of year. In order to accommodate these new temporal features and still preserve sequential nature of input matrices $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{X}(\lambda_i)$, we reshape the matrices into third-order tensors and stack the temporal features along its third dimension. In other words, we transform $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{X}(\lambda_i)$ into $N \times k \times M$ tensors where N is the total number of input sequences, k is embedding dimension (length of input sequence), and M is the total number of input features. The following representation of input tensor $\boldsymbol{X}(\mu_i)$ further illustrates this transformation.



Above input representation allows us to conveniently introduce new features by expanding the third dimension (i.e., depth) of the tensor. In order to demonstrate this, let us consider weather information as an additional input to the forecast. We assume that a weather report $w(t)$ is available for each time interval t. The report consists of typical weather variables such as sky condition, temperature (°C), humidity (%), visibility (km), and wind speed (km/h), i.e., $w(t) = \big(sky(t), temp(t), hum(t), vis(t), wind(t)\big)$. Here it is important to emphasize that all of the above variables are numerical except the sky condition $sky(t)$ which is usually represented as a short phrase (e.g., *sunny*, *mostly cloudy*, *light rain*, *clear*, etc.). To make it numeric-valued, we can use one-hot encoding, i.e., introduce new variables for each possible condition phrase, setting them all to zero except for the one which corresponds to current sky condition, which is set to one. However, the number of possible sky conditions can be large, while most of them rarely take place. For example, the weather provider that we use in this work (Weather Channel) specifies 133 different condition phrase,while only 16 of them appear among the weather reports collected in this work. Therefore, in order to reduce the number of new condition variables, and thus prevent the forecasting model from overfitting, we analysed the correlation between number of pick-ups/drop-offs and sky conditions and then appropriately grouped possible condition phrases into three groups – *good*, *bad*, *moderate* (details of this analysis are omitted to conserve space). Using this grouping, we can now replace $sky(t)$ with only three new binary variables $sky\_good(t)$, $sky\_bad(t)$, and $sky\_moderate(t)$.

Finally, we incorporate weather information by expanding the third dimension of the input tensors $\boldsymbol{X}(\lambda_i)$ and $\boldsymbol{Y}(\lambda_i)$ for each weather variable in $w(t)$. This leaves us with tensors containing N sequences of k intervals and $M = 11$ features.

B. Deep learning model

Over the last several years, recurrent neural networks (RNNs) with hidden Long Short-Term Memory (LSTM) units [6] have become the model of choice for learning from data with sequential features. LSTM overcomes the problem of vanishing gradients [7], which makes very difficult for traditional RNNs to learn to correlate temporally distant events. This is achieved with carefully designed memory cells which protect their hidden activation using special gating mechanism to control the information flow through the cell. We refer the interested reader to [6] and [8] for a detailed description of LSTM units and memory cell implementation.

In this work, we harness the sequential nature of RNNs and propose an LSTM-based approach for user demand prediction in BSSs. In Figure 2 we illustrate an example of two-layered LSTM architecture unrolled through time, which assists us in introducing sequence-to-sequence (seq2seq) configuration [9] that allows us to forecast over multiple time steps. Notice here that input $\boldsymbol{x}$ and output $\boldsymbol{y}$ are sequences whose time steps are revealed in unrolled representation of the network. The illustrated network is composed of two LSTM layers named *encoder* and *decoder*. The purpose of the encoder is to convert input sequences of variable length and encode them in a fixed length representations, which is then used as the input state (i.e., context) for the decoder. In our example, the encoder processes all inputs in chronological order from $t-k+1$ to t and passes its hidden state to the decoder at the final sequence step of t, whereupon the decoder starts to generate an output sequence. The main advantage of this architecture

is that it allows us to use inputs of arbitrary length to predict for an arbitrary number of future time steps. The encoder-decoder LSTM was developed for natural language processing problems where it demonstrated state-of-the-art performance, specifically in the area of text translation. However, observe that this architecture is particularly appropriate in our case where user demands for n future time steps is predicted based on their values in k preceding time steps, where $n \neq k$. For more details about sequence-to-sequence prediction we refer the reader to [9] where the architecture was first introduced.

The encoder and decoder in the network in Figure 2 could easily have more layers to empower the capability to learn more complex dependencies. A naïve approach to our forecasting problem would be to train such multi-layered network separately over $\boldsymbol{X}(\mu_i), \boldsymbol{Y}(\mu_i)$ and $\boldsymbol{X}(\lambda_i), \boldsymbol{Y}(\lambda_i)$ for each station s_i. While this approach at first may seem as a good choice given that the networks are trained for specific stations, we will see in Section V that with the separate models we are essentially ignoring any correlation that might exist between the stations. In addition, maintaining multiple models requires additional computation and memory. Therefore, we propose to train a more advance network over data aggregated over all stations in BSS. We construct new input and output tensors by concatenating $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{Y}(\mu_i)$ for each station s_i, i.e.,

$$\boldsymbol{X}(\mu) = \begin{bmatrix} \boldsymbol{X}(\mu_0) \\ \boldsymbol{X}(\mu_1) \\ \vdots \\ \boldsymbol{X}(\mu_S) \end{bmatrix}, \boldsymbol{Y}(\mu) = \begin{bmatrix} \boldsymbol{Y}(\mu_0) \\ \boldsymbol{Y}(\mu_1) \\ \vdots \\ \boldsymbol{Y}(\mu_S) \end{bmatrix}, \forall s_i$$

where S is the total number of stations in BSS. In the same way we aggregate tensors $\boldsymbol{X}(\lambda_i)$ and matrices $\boldsymbol{Y}(\lambda_i)$, $\forall s_i$, into $\boldsymbol{X}(\lambda)$ and $\boldsymbol{Y}(\lambda)$. Now, in order to track which sequences belong to which station, we introduce an auxiliary input matrix $\boldsymbol{Q}$ which in each row t contains non-sequential features, such as station ID, of the corresponding sequence (i.e., row) t in $\boldsymbol{X}(\mu)$ and $\boldsymbol{Y}(\mu)$ (and $\boldsymbol{X}(\lambda)$ and $\boldsymbol{Y}(\lambda)$). In our case, in addition to station ID, matrix $\boldsymbol{Q}$ will also contain information about holidays. For example, row $\boldsymbol{q}_t = [123\ 1]$ of matrix $\boldsymbol{Q}$ suggests that sequence t is recorded at station with $ID = 123$ during a holiday (1).

Finally, in order to handle both sequential and non-sequential inputs we propose the architecture illustrated in Figure 3. The architecture contains two input layers, one for sequential inputs $\boldsymbol{X}(\mu)$ and $\boldsymbol{X}(\lambda)$, and one for non-sequential inputs $\boldsymbol{Q}$. Sequential inputs are fed directly to encoder-decoder LSTM network configured as shown in Figure 2. To obtain sequences of n future time steps, output of the decoder at each time step from t to $t+n$ is separately processed by fully-connected (inner product) layer. Given its temporal dimension, we refer to this layer as time-distributed fully connected layer (TD-FC). At this point, we introduce non-sequential data for each of the output sequences from the TD-FC layer. In other words, we row-wise concatenate $N \times P$ matrix $\boldsymbol{Q}$ (where P is the total number of non-sequential features) and $N \times n$ matrix $\boldsymbol{R}$ which is the output of TD-FC layer. Finally, the merged data is fed to two-layered fully connected network to produce final sequences of n future time steps.

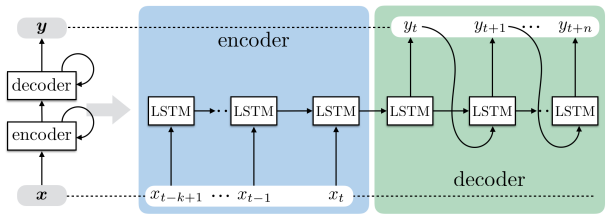


Fig. 2. Sequence-to-sequence architecture

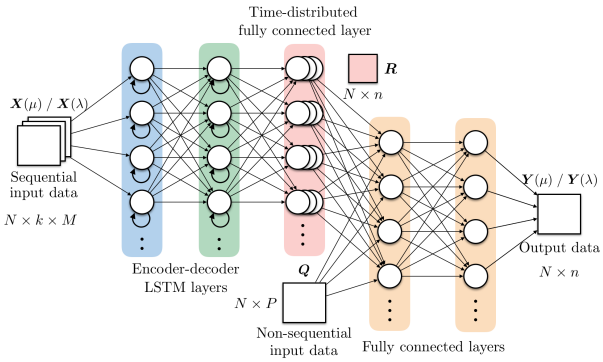


Fig. 3. The proposed network architecture

IV. Rebalancing target estimation

In the final step of the proposed framework, we use predicted user demands to compute rebalancing targets for each of the stations in BSS. Recall from Definition II.3 that this computation consists of finding a rebalancing target d_i which would maximise the total number of consecutive time intervals during which the station remains balanced. In order to find such rebalancing target, Algorithm 1 iterates over all future horizons, i.e., time intervals, and at each iteration it looks for a set of possible values for d_i ($\widehat{D}$) with respect to time intervals considered in previous iterations. $\widehat{D}$ is determined in line 6 based on the current inventory level c_i, the station's capacity l_i, and the total bicycle net flow (net) represented as the difference between expected number of returned and rented bicycles up until time interval of the current iteration (line 5). Thereafter, the algorithm restricts set $\widehat{D}$ by selecting subset of rebalancing targets d_i (D) which keep the station balanced over the current time interval, but also over all of the previous time intervals considered in past iterations (line 8). If such subset does not exist, i.e., D is empty, the algorithm stops the iteration (line 10). Finally, if there are several rebalancing target values left in D, the algorithm takes one with the minimal absolute value (line 13) in order to minimize the cost of rebalancing operation.

Notice that sets $\widehat{D}$ and D are connected, i.e., they are both intervals, which means that the intersection between them (lines 7-8) can be found in constant time. This means that the computation time of Algorithm 1 depends only on the number of future horizons n, i.e., its time complexity is linear ($O(n)$).

V. Experimental evaluation

Our experimental study is based on real-world bicycle trip data collected in New York's BSS known as CitiBike. We used a subset of this data containing 12,246,450 trips with bicycle pick-ups/drop-offs recorded at 750 stations over the period between January 1st, 2017 and October. 1st, 2017. Each pick-up and drop-off event is represented as a timestamp and ID of the station where the event was recorded. We augmented this data with weather information for each timestamp by obtaining historical weather data using Weather Channel's API. This data consists of typical weather variables that we introduced previously in Section III-A. We constructed input sequences composed of $k = \{4, 8, 16, 24\}$ past and $n = 8$ future 30-minute intervals, and then randomly split them into a training

ALGORITHM 1: Rebalancing target estimation

Input: Predicted pick-up and drop-off demands $\mu_i(t+j)$, $\lambda_i(t+j)$ $\forall j \in [1..n]$, current inventory status c_i, capacity of the station l_i
Output: Rebalancing target s_i

Procedure EstimateRebalancingTarget(s_i)
  $net \leftarrow 0$
  $D \leftarrow \varnothing$
  for $j \in [1..n]$ **do**
    $net \leftarrow net + \lambda_i(t+j) - \mu_i(t+j)$
    $D_j \leftarrow \{d_i \mid 0 \leqslant c_i + d_i + net \leqslant l_i\}$
    if $D \cap D_j \neq \varnothing$ **then**
      $D \leftarrow D \cap D_j$
    else
      break
    end
  end
  return $\min\{|d_i| | d_i \in D\}$

set (80%) and a test set (20%). Notice that 8 future time intervals corresponds to 4 hours, i.e., half of a common 8-hour working shift. In order to measure the performance of the proposed approach we use the mean absolute error (MAE) of user demands at each of the $n = 8$ predicted time intervals. In addition, we compare our approach with traditional neural network, i.e., multilayer perceptron, simple linear regression, and two ensemble methods that demonstrated competitive performance in previous related works [4, 10] – random forest and gradient boosting machines.

We developed the proposed deep learning model using Keras neural networks API, written in Python and running on top of TensorFlow machine learning library. The model's parameters were tuned manually by trial and error. The number of units contained in encoder and decoder LSTM layers and two fully connected layer were 128, 256, 256, 256, respectively. Given the very large number of training sequences (> 9 million) we trained our model in batches of 1024 sequences over 100 training epochs. In Figure 4a we plot MAE of pick-up and drop-off demand at each time horizon for different number of lagged time intervals k. As expected, the prediction error decreases as we increase the number of lagged observations, i.e. the model learns more complex dependencies. In the particular experiment, average MAE over 8 time horizons decreases from 0.799 pick-ups for $k = 4$ to 0.713 pick-ups for $k = 24$ (and from 0.754 to 0.681 drop-offs). Also notice that prediction error is lower at closer time horizons. For example, MAE of predicted pick-up demand in the next 30 minutes (one step ahead) is only 0.667 for $k = 24$, while MAE of the same prediction 4 hours (8 time steps) ahead of time is 0.741. This performance drop is more significant for smaller k values. For example, for $k = 4$, MAE increases 1.731% from first to last time horizon, and 1.001% for $k = 24$. Similar can be observed in case of drop-off prediction.

The guiding design principle behind our model was to separate sequential and non-sequential data in order to train a single model for all stations in a BSS. We have previously argued that this way we are not ignoring correlations that might exist between the stations, which can significantly improve the overall prediction accuracy. To justify our design choice, in Figure 4b we plot MAE of predicted pick-up and drop-off demands when each station is trained on a separate model. Notice that these models are identical to the one proposed in Figure 3 but without non-sequential input data and fully connected layers. The plot clearly demonstrates the advantage of our approach. MAE averaged over 8 time horizons is 0.882 pick-ups and 0.846 drop-offs in case of separate model training, and 0.713 pick-ups and 0.681 drop-offs in case of collective model training.

In Figure 5 we compare the performance of our model against the baselines. Given that the employed baseline models cannot handle sequential inputs, we represented each time step as a set of additional features (i.e., columns) in 2D input. As expected, simple linear regression model (LR) was not able to learn more complex nonlinear dependencies, hence the largest MAE of 1.218 pick-ups and 1.192 drop-offs across all 8 time horizons. Here we can also clearly observe the effect of error accumulation where MAE grows from 0.98 pick-ups and 0.95 drop-offs at the first time horizon to 1.32 pick-ups and 1.31 drop-offs at the last time horizon. This effect is least noticeable in case of multilayer perceptron (MLP). The employed MLP model was composed of 3 layers with 256 neurons each, and it was trained in the same way as our deep learning model (in batches of 1024 samples over 100 training epochs). MLP performs significantly better than LR, and comparably to ensemble models – random forest (RF) and gradient boosting machines (GBM). Our experiments confirm the efficiency of the ensemble models reported in previous works. They manage to successfully average out biases and reduce the variance. Finally, as can been seen in the plot, our method outperforms tested baselines. In average, it performs 41% better than LR, 11% better than MLP, 18% better than RT, and 13% better than GBM.

In order to demonstrate the applicability of our framework, in Figure 6 we visualize rebalancing targets for the stations in New York's BSS. Each rebalancing target is illustrated as circle whose color represents the amount and type of redistribution, i.e., shade of red for bicycle removals and shade of blue for bicycle restocking. In this particular example, we computed the rebalancing targets (Algorithm 1) based on inventory snapshot recorded on August 9th, 2017 at 5am (a) and 1pm (b), i.e., before morning and afternoon rush hours. In Figure 6a stations located in business areas in Lower (1) and Midtown (2) Manhattan are coloured red, meaning some number of bicycles need to be removed due to increased drop-off demand in morning hours. On the other hand, stations in residential areas, for example in East Village (3) and Upper East Side (4), are coloured blue, meaning increased pick-up demand is expected in morning hours and additional bicycles need to be brought to these stations. In Figure 6b, the color are reversed. Some stations in business dominant areas (e.g., 1, 2) does not have enough bicycles to meet high pick-up demand in afternoon hours (coloured blue), while some stations in residential areas (e.g., 3, 4) have to many bicycle and cannot meet upcoming drop-off demand.

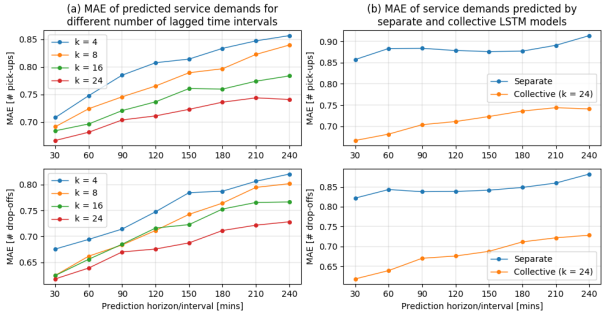


Fig. 4. Effects of different parameters on the prediction accuracy

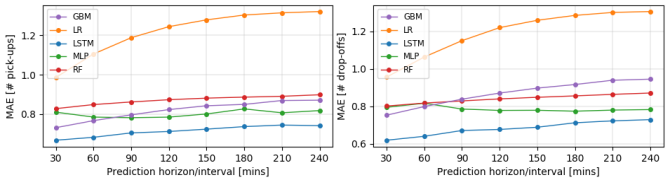


Fig. 5. MAE of (a) pick-up and (b) drop-off demand at each time horizon for different baselines.

VI. Related work

The problem of rebalancing BSSs by redistributing bicycles between stations using a fleet of maintenance vehicles is known in the literature as the *Bicycle sharing Rebalancing/Repositioning Problem* (BRP). BRP consist of (1) estimating the stations' inventory target levels (i.e., redistribution needs) and (2) computing optimal routes for a fleet of capacitated vehicles. In this paper we considered only the first stage of BRP. Even though BRP has been in focus of many research works over the past several years, more adaptive solutions to BRP which could efficiently adapt to unplanned events (e.g., weather changes, traffic jams, system failures, etc.) have not been well studied.

In their inspiring work, Liu et al. [2] proposed a k-nearest neighbour regressor based on a novel meteorological similarity measure to predict the station pick-up demand based on large-scale historic trip records. Based on further analysis on the station network they propose an inter-station bicycle transition model to predict the station drop-off demand. Despite the high accuracy, this approach is unable to adapt to current traffic events in BSS, given that it relies only on weather information. In addition, the demands are forecast directly for a specific time interval without considering recent observations over past several time intervals. In [4], Regue and Recker develop single-step-ahead demand forecasting model based on gradient boosting machines (GBMs). However, they also consider three different prediction time windows – 20, 40 and 60 minutes, in order to better optimise redistribution operations. Froehlich et al. [11] implemented four simple predictive models, including a Bayesian network to predict the availability of bicycles at each station in a BSS. In a similar work [12], ARMA estimator was employed to predict the station availability one step ahead of time. As opposed to beforementioned data-driven demand forecasting models, Nair et al. [13] provide a theoretical analysis of the user demand in BSSs using dual-bounded joint-chance constraints. They predict the near future demand for a short period and ensure that the system-wide demand is served with a certain probability. On the other hand, Raviv et al. [14] and Schuijbroek et al. [1] use Markov chain to model the station inventory over time. The pick-ups or drop-offs are represented as random variables with a known probability distribution. While these assumptions hold for one step or short term planning, it becomes intractable for multi-step or long-term planning due to the time-varying nature of the demand and the inter-dependency between the pickup and drop-off demands between stations at consecutive time steps [15]. Finally, in a very recent work [15], Ghosh et al. successfully employed Poisson distributions to represent user arrival process at each of the stations in a BSS.

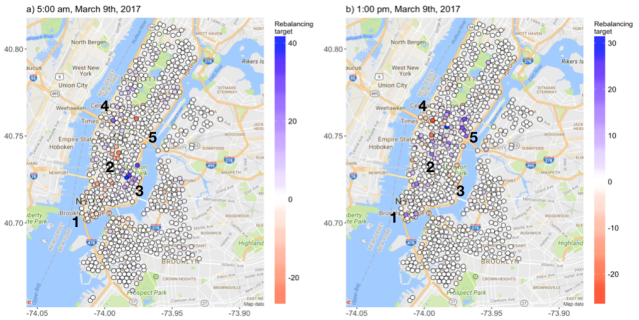


Fig. 6. Rebalancing targets in New York's BSS

VII. Conclusion and future work

Forecasting user demand multiple time steps into the future allows for more accurate long-term inventory planning which can lower the number of redistribution runs and thus significantly reduce the operating cost. We showed that our sequence-to-sequence deep learning network based on LSTMs can efficiently forecast user demand over multiple future horizons and outperform traditional machine learning methods commonly employed for demand prediction in shared transport. Further, we proved the applicability of the proposed model by developing a simple algorithm that is able to successfully estimate rebalancing requirements for each station in a BSS based on the predicted user demand. For the future work, we plan to develop an online (adaptive) framework for BRP which would take in consideration the real-time information from BSSs to re-estimate inventory target levels and accordingly to update (i.e., re-optimise) routes that were already assigned to rebalancing vehicles.

References

[1] J. Schuijbroek, R. C. Hampshire *et al.*, "Inventory rebalancing and vehicle routing in bike sharing systems," *European Journal of Operational Research*, vol. 257, no. 3, pp. 992–1004, 2017.

[2] J. Liu, L. Sun *et al.*, "Rebalancing bike sharing systems: A multi-source data smart optimization," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1005–1014.

[3] J. Pfrommer, J. Warrington *et al.*, "Dynamic vehicle redistribution and online price incentives in shared mobility systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1567–1578, 2014.

[4] R. Regue and W. Recker, "Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 72, pp. 192–209, 2014.

[5] S. B. Taieb, G. Bontempi *et al.*, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] R. Pascanu, T. Mikolov *et al.*, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[8] F. A. Gers, J. Schmidhuber *et al.*, "Learning to forget: Continual prediction with lstm," 1999.

[9] I. Sutskever, O. Vinyals *et al.*, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[10] J. Malani, N. Sinha *et al.*, "Forecasting bike sharing demand."

[11] J. Froehlich, J. Neumann *et al.*, "Sensing and predicting the pulse of the city through shared bicycling." in *IJCAI*, vol. 9, 2009, pp. 1420–1426.

[12] A. Kaltenbrunner, R. Meza *et al.*, "Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system," *Pervasive and Mobile Computing*, vol. 6, no. 4, pp. 455–466, 2010.

[13] R. Nair and E. Miller-Hooks, "Fleet management for vehicle sharing operations," *Transportation Science*, vol. 45, no. 4, pp. 524–540, 2011.

[14] T. Raviv and O. Kolka, "Optimal inventory management of a bike-sharing station," *IIE Transactions*, vol. 45, no. 10, pp. 1077–1093, 2013.

[15] S. Ghosh, P. Varakantham *et al.*, "Dynamic repositioning to reduce lost demand in bike sharing systems," *Journal of Artificial Intelligence Research*, vol. 58, pp. 387–430, 2017.