OD matrix estimation by deep learning using maps

（地図データとディープラーニングを用いたOD交通量推定）

2020/02/17

International Course Program of Civil Engineering
Undergraduate School of Global Engineering
Faculty of Engineering
Kyoto University

Danyel Koca

Abstract

Accurate travel demand modelling (TDM) is crucial for desirable transportation and urban planning. Origin–destination (OD) matrix estimation is an important component of TDM. Conventional OD matrix estimation methods require travel surveys which are costly and time-consuming. Moreover, conventional methods are too simplistic to represent real world situations while modern methods are too complicated to be scalable. Investigation of comprehensive and scalable OD matrix estimation methods that use low-cost inputs are an imperative for transportation researchers. One such method is deep learning.

People generate more and more data about their trips thanks to the new technologies such as ride hailing/sharing apps, GPS, and smart card. On the other hand, deep learning methods are being developed that can handle large amounts of data and can successfully perform difficult estimation tasks. Therefore, it is useful to see if deep learning can successfully predict an OD matrix. In this research, we develop a novel 2-step deep learning algorithm that uses nearby points of interest (POIs) of zones as input and estimates interzonal trips. We believe that this model will provide an inexpensive and scalable alternative to the conventional OD matrix estimation methods.

Acknowledgements

I would like to thank my supervisor Jan Dirk Schmöcker, and the head of the ITS laboratory Tadashi Yamada for facilitating a wonderful research experience. Moreover, I am grateful to Kouji Fukuda of Hitachi Laboratory for his valuable advices about the technical aspects of the research. I am indebted to Wenzhe Sun and all the other members of Intelligent Transport Systems Laboratory for insightful discussions and unyielding support.

Table of Contents

1. Introduction

Travel demand modelling is the estimation of frequency of trips between predetermined places, or zones, for various transportation modes and routes. Conventionally, 4-step modelling has been used to estimate demand: trip generation, trip distribution, mode choice and route assignment [1]. With trip generation, we estimate the number of trips originating and/or ending in zones. In trip distribution, trips originated and ended in the previous step are matched with destinations and origins, respectively. By the end of trip distribution, we have constructed an origin–destination (OD) matrix. This OD matrix could be obtained over various time spans such as an hour, a day or a week. After trip distribution, the trips are allocated to different travel modes and routes.

OD matrix estimation has traditionally involved OD surveys. Obtaining a representative survey sample size that can be used to infer about the whole OD matrix is expensive [2]. Recently, more data driven approaches have been emerging. For example, traffic counts or highway toll records are being used instead of travel surveys [3]. If we have marginal OD sums[1], gravity model can be used to allocate those sums into OD pairs [4]. According to the gravity model, the interzonal trip amount is proportional to the total trips produced at origin (or productivity), total trips ending at destination (or attraction), and reversely proportional to the friction between these zones. Conventionally, population, economic activity, and land use data have been used to describe production and attraction. Friction can be described as time, cost or inconvenience of an interzonal trip. On the other hand, when we have an old OD matrix or a current one with some empty entries, current and complete OD matrix can be constructed from that matrix using growth factor models [5].

No matter if we use the gravity model or growth factor models, we need a costly input data. In gravity model, production and attraction capacity of each zone should be quantified. On the other hand,

[1] Marginal OD sums are the sums over the rows and columns of the OD matrix. It is the product of the trip generation step.

growth factor models require at least a partial OD matrix. Therefore, it is imperative for us to develop an inexpensive model for travel demand estimation.

In this research, we propose a new method for estimating OD matrix with deep learning using maps. In line with the gravity model, it can be thought that productivity of the origin, attraction of the destination and the generalized cost of traveling from origin to destination are the 3 most determinate factors for total trip amount between an origin and destination. While cost can be easily calculated thanks to the modern map technologies, it is quite difficult to accurately describe the production and attraction capacity of zones. One common approach to determine attraction and production is to consider the land use. It is obvious that land use has an explanatory power in estimating trips; most people will go from residential areas to commercial and industrial areas in the morning and they will do the trip in the opposite direction in the evening. However, not all commercial areas have the same attraction capacity. For determining the capacity, we can further incorporate variables such as building or population density. More residential houses mean more people will do trips from those zones. Moreover, the contents of the zones are important in estimating the trips. A shopping area will attract people throughout the day while an entertainment area might attract people only in the evening. The effect of the contents of zones on travel behavior becomes even more pronounced when it comes to mode choice: an area with a small number of parking lots will attract relatively less cars. In the transportation research lexicon, places such as subway stations, parking lots, offices, and houses are called *points of interests* (POIs). Following the logic of this paragraph, we see that the attraction and production of each zone might be explained by what kind of POIs and how many of such POIs we have in that zone. For example, we can easily imagine the temporal distribution of trips between 2 zones, zone *i* and zone *j*, where *i* has 50 houses, 4 parks, 1 school, and *j* has 15 offices, 7 restaurants and 3 parking lots.

If we have the generalized cost, either travel time, monetary cost or distance in addition to the POI data, we hypothesize that it could be possible to estimate the OD matrix given these two datasets. In this research we test this hypothesis with a deep learning model.

2. Motivation

We have 2 reasons to conduct this research. Firstly, it is expensive and time-consuming to obtain OD matrix using traditional survey data. We want to develop a model that can create an accurate OD matrix with low-cost input. Our model is going to use POI and distance data as input. With today's technology, these datasets can be found easily by just looking at maps. For example, on Google Maps, one can find the nearby restaurants by just searching for "restaurants". If we were to do this for all the POI types, we can get an exhaustive list of POIs for our zones. Although one has to use Google Maps API to get such data and pay a fee, the fee is minimal.

Another option for acquiring input would be to tap into the data that is generated via open-source. One example is Open Street Map (OSM).[2] OSM provides free geographical data and maintained by volunteers. Although OSM data is free, the data quality differs from place to place due to the fact that OSM is open-source. For example, in places with high internet usage, high OSM familiarity and high number open-source contributors, the data of OSM matches the quality that of Google Maps. However, in some areas with little user activity, most of the POI data does not exist or it is obtained in bulk by OSM from government agencies, which naturally do not disclose the full data due to the privacy issues. In such cases, building footprints may be registered on OSM but the contents of buildings are unknown.

[2] openstreetmap.org

Another motivation for using map data is that, in most countries, local or central authorities have an exhaustive list of all the POIs found within a city because an establishment has to either register at the city hall or get a license. With a successful model that can predict the OD matrix with POI and distance data, the local or central governments can apply this in transportation planning using their own data for input and get rid of costly travel surveys.

All in all, digital maps have abundant and low-cost data, which makes it a perfect input candidate for our travel demand estimation study.

Secondly, we may know that a relationship exists between POI distribution, distance and travel demand, however, it is quite difficult to develop a model that is accurate and scalable. Therefore, we employ deep learning as our travel demand estimation model. Deep learning is a branch of machine learning, that is used for estimating a label (true value) given input. The name deep is used because the input data goes through different layers until the final output, changing shape and content along the way. In addition, nonlinear operations are implemented at each layer. Thanks to the complexity of the method, a deep learning model can learn intricate relationship between the input and the label. As of 2020, deep learning models have been successfully trained for tasks such as object detection, natural language processing and drug development. All these tasks have intricate relationships between the input and labels, similar our travel demand estimation problem.

Another reason we choose deep learning is the abundance of data. Google Maps claims to have more than 150 million places, or POIs, worldwide.[3] As mentioned earlier, this extensive map data can be

[3] https://cloud.google.com/maps-platform/

obtained with low-cost and fed into the model. A deep learning model is as good as quantity and quality of its training data; hence we are expecting that the model will be able to learn the relationship between POIs and trips.

In summary, we have decided to estimate the OD matrix by deep learning using maps because map data can be obtained with low cost and includes the POI and distance data, while deep learning can efficiently understand the intricate relationship between POI, distance and travel demand.

3. Literature review

OD matrix estimation models differ by the data they use or the method they employ. Here, we review literature based on these 2 factors.

3.1.Review by method

OD matrix estimation has been conventionally done by using 3 family of models which are, growth factor models, synthetic models, and opportunity models.

Growth factor models require an existing base matrix. For example, uniform growth factor method updates all the trips between OD pairs uniformly from an early OD matrix into the present one. The growth rate is calculated as the capacity increase in the whole network, either population increase or observed traffic increase. This model ignores the individual changes in links. Average growth factor method takes differences in zonal growth into account and updates the zones individually. Fratar method is another example of growth factor method which has been proposed in 1954 for estimating interzonal highway traffic [6]. Present (or past) OD matrix is updated into future (or current) OD matrix by fixing the total target trips originating and ending at each zone, making this a doubly constrained optimization problem. After successive iterations, present OD matrix elements are updated so that the sum of trips satisfy the target total trips. A disadvantage of growth factor models

is that they usually ignore the changes in the network characteristics such as link capacities, road quality, and infrastructure updates.

Synthetic models are a family of models that were imported to transportation research from another fields. A prominent example is gravity model which has been inspired by Newton's law of gravity. It is thought that trips between two zones are proportional to the masses (production and attraction) of the zones and inversely proportional to the distance (or generalized cost) between the two zones. Gravity model in its simplest form states that cumulative trip amount between zone *i* and *j*, T_{ij}, can be calculated as:

$$T_{ij} = K\frac{P_i A_j}{{d_{ij}}^n}$$

where,

P_i = Total trips originating from *i* (or productivity at *i*),

A_j = Total trips ending in *j* (or attraction at *j*),

d_{ij} = costs of travelling from *i* to *j*,

K, *n* = constants.

For P_i and A_j, proxy values instead of total trips can be used. These values should represent the productivity and attractiveness of zones, respectively. Values such as land use, population density, average floor to area ratio can be used to quantify productivity and attractiveness.

The first opportunity model was the intervening opportunities model which was proposed by Stoufer in 1940: "*the number of persons going a given distance is directly proportional to the number of opportunities at that distance and inversely proportional to the number of intervening opportunities*" [7]. The opportunity models fall under the general formula:

$$T_{ij} = T_i P(D_j)$$

where,

T_{ij} = trips between zones *i* and *j*,

T_i = total trips generated at zone *i*,

$P(D_j)$ = probability of trips ending in zone *j*.

Recently, more sophisticated explanatory and predictive models have been applied for OD matrix estimation such as, trip-chaining models [8], deep learning [9], Markov chain Monte Carlo method [10], and Bayesian inference [11].

3.2.Review by input data type

As mentioned earlier, OD surveys are costly, therefore researchers have been trying to estimate OD matrix with inexpensive proxy data. For example, one study used data from location based social network application *Foursquare* and trip planner *Rome2Rio* to estimate travel demand [12]. Some examples of input data among many are GPS [13], smart card [14, 15], mobile phone billing data [16], mobile network probe data [17], and Bluetooth [18]. Trips of the modern people are almost always recorded, either willingly by updating their status on social networks or unwillingly by usage of map services, WIFI or ride sharing apps. These large datasets are quite valuable because conscious input is small compared to the travel survey, hence reducing bias. Secondly, since these datasets are already being collected and stored by corporates, it can be anonymized and shared with researchers with minimal cost for developing better travel demand models.

The input data that we are planning to use comes from OSM. OSM data has been used as input for various prediction tasks. For example, the road network and POI data from OSM has been employed to estimate fine-grained population [19]. Another research uses OSM data to predict real time traffic conditions in China [20]. Moreover, OSM data has been used for predicting various parameters such as land use distribution [21], poverty rate [22], and transport fatalities [23].

4. Methodology

Here, we explain the 3 steps of our methodology, obtaining labeled dataset (OD matrix), acquiring and organizing inputs (POIs, distance), and model architecture (deep learning). Our work starts with finding a labeled dataset. When we find an OD matrix that we can use for training our model, we extract its drive network and POIs and use them as inputs to our model. Finally, we train the algorithm using the inputs and estimate the OD matrix. Please see Figure 1 for a flow chart of our methodology.

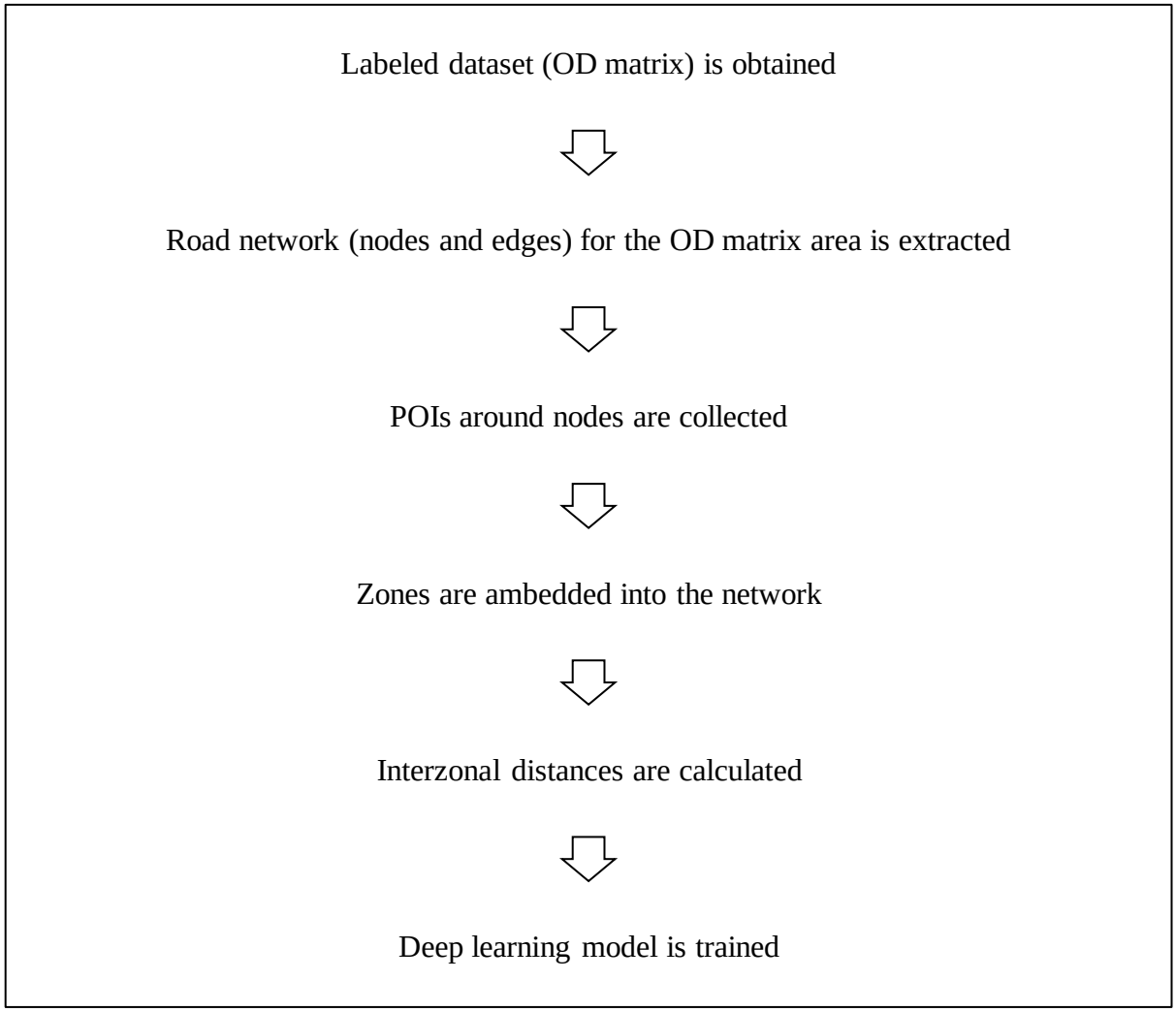

Figure 1: Flowchart of data collection and training

4.1. Obtaining labeled dataset

Although we have virtually ubiquitous input data (POIs, distance), our model is limited by travel demand data we have. Thanks to the advance of technology, most of our trips are leaving digital footprints which can be used as a labeled dataset for our model. Examples of such data are smart card data, ride-hail app data, mobile GPS data, automatic vehicle location (AVL) data. The trip data we use is the number of taxi trips in New York City (NYC) for a period of one month in January 2019. NYC Taxi and Limousine Commission has been publishing the taxi trip data since 2009. The trips from 3 different taxi types are included in the dataset: yellow taxi, green taxi and for-hire vehicles. Each row in the dataset is a trip that took place within January 2019 and includes pull-up (origin) and drop-off (destination) locations. These locations are represented by zone IDs. NYC is divided into 5 boroughs: Manhattan, Brooklyn, Queens, Bronx, and Staten Island where each borough is divided into zones. We have chosen Manhattan and Brooklyn as our target locations for cross-testing the model. Some of the zones in these 2 boroughs are islands which cannot be accessed by taxis. After deleting zones with no accessibility, Manhattan and Brooklyn had 62 and 61 zones respectively (3844 and 3721 OD pairs). Please see Figure 2 for Manhattan and Brooklyn zones and zone centroids. The number of trips that took place within Manhattan were 17,478,092 while Brooklyn had 4,258,389. If we add the cross-borough trips between Manhattan and Brooklyn, the total number of trips that took place in Manhattan and Brooklyn is 23,526,117 which amounts to around 5 taxi trips per residents of Manhattan and Brooklyn within a month [24]. Since Manhattan and Brooklyn boroughs had high number of taxi trips, we decided to use only the month of January 2019 thinking it would be enough for model.

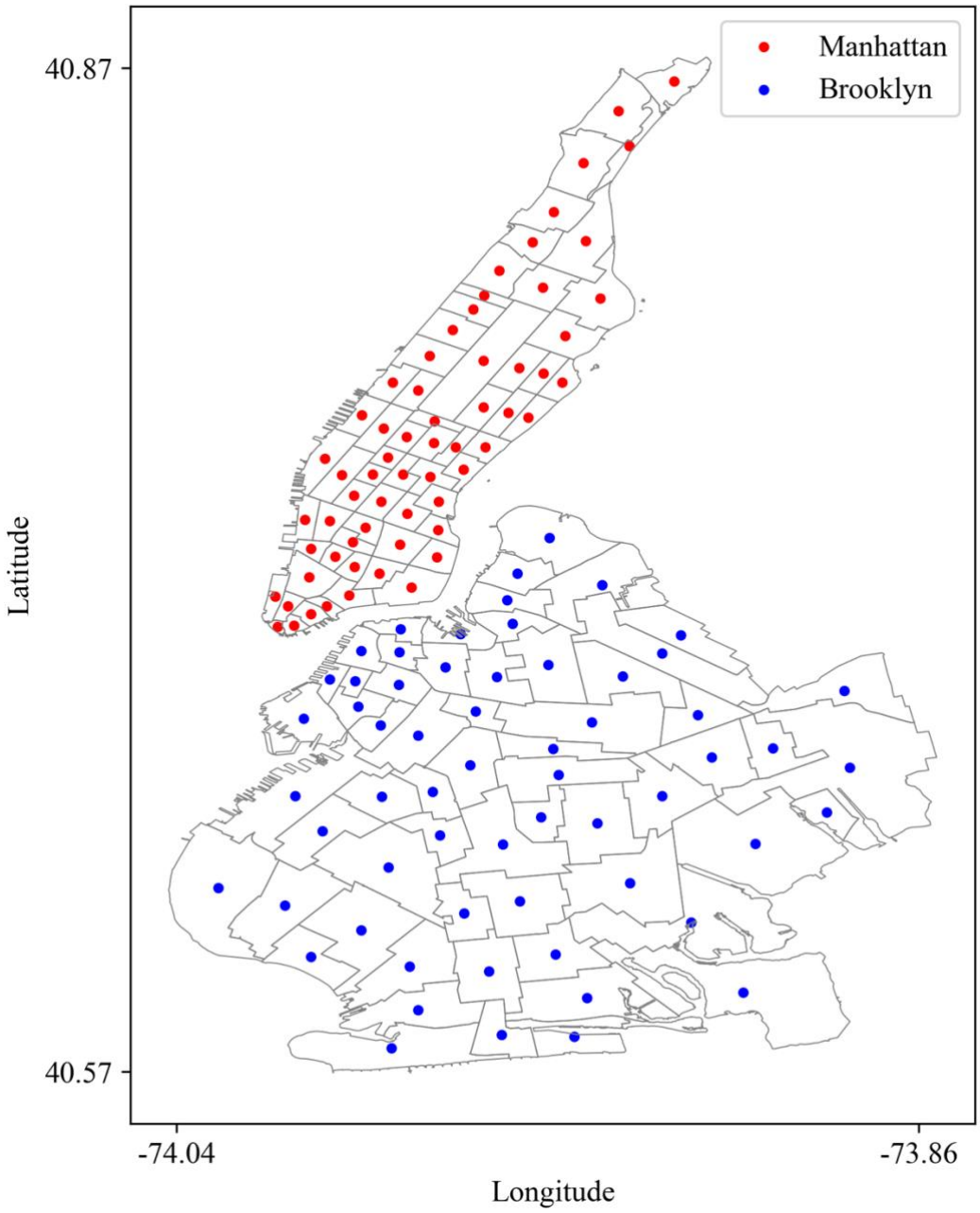

Figure 2: Zones of Manhattan and Brooklyn and their centroids

4.2. Acquiring and organizing inputs

We use OSM to get the POI and distance data. 3 main elements in OSM are nodes, ways and relations. Nodes are points on the map, a node can be an intersection or a corner of a building. Ways are collections of nodes. Ways represent the roads or the periphery of structures. For example, a rectangular building will be a "way" of 4 nodes which are located at the corners of the building while a highway will be a collection of many nodes lined up along the road. Relations are collections of nodes and ways. For example, a bus route may be represented as a collection of nodes (bus stops) and ways (bus route). Any element in OSM might have tags associated with it. A tag is a key-value pair. For example, a common key is amenity which shows places of entertainment, recreation, and etc. The key amenity will have a value associated with it, such as: shop, museum, park; creating amenity-shop, amenity-museum, amenity-park key-value pairs, or tags.

Our deep learning model requires a graph representation of the drive network of the city as input. A graph is a construct with nodes and edges. In our graph, nodes are road intersections while the edges are roads that link them. As mentioned, roads in real world are represented by ways in OSM. A road will have a highway key associated with it. The roads that we are interested in, the drivable roads, will have their highway key coupled with one of the below values[4]:

- Motorway
- Motorway link
- Trunk
- Trunk link
- Primary
- Primary link
- Secondary
- Secondary link
- Tertiary
- Tertiary link
- Unclassified
- Residential
- Road

[4] Please see https://wiki.openstreetmap.org/wiki/Key:highway for a full list of values that *highway* key can be coupled with.

If we want to create a graph of a road network of a certain area, we can get the OSM map data of that area and find all the ways that have the key highway and any value from above list. When 2 ways have a node in common, we can pinpoint that node as a road intersection and build our graph. The *OSMnx* library in Python helps us extract and visualize the road network from OSM. Each node in the extracted graph is an actual node in OSM and represents a road intersection. A node has attributes of longitude and latitude. Each edge has its length and road type as an attribute and represents a real road. Figure 3a shows a piece of map and Figure 3b is a network representation of that map, extracted using OSMnx.

Once we have our network, the next step is to extract the nearby POIs for each node in the network. Below are the steps that we follow for POI extraction:

1. Select a geographical area from OSM that encompasses the drive network that was extracted previously.
2. Download the OSM datafile from openstreetmap.org using overpass API.[5] At the end of this step we have all the information (nodes, ways, and relations with all the key-value attributes) for that area. Please see Table 1 that shows how nodes, ways and relations are stored in this data file.
3. We use *The Element Tree XML API* library of Python to extract the POIs of interest to us from the downloaded file. Deciding what is a POI and what is not can be ambiguous. For example, OSM has key value pairs such as "highway-junction", "natural-tree", "amenity-bench". Can junctions, trees or benches be counted as POIs? Since deep learning requires abundant input data, we keep the definition of POI as wide as possible. At the end of the day, the deep learning algorithm will determine the importance of

[5] https://www.openstreetmap.org/export

each of the POIs. Among dozens of keys an OSM element might have, we determined that the 24 of them can be counted as POIs. Once a node or a way has one of these 24 keys, we count it as a POI no matter what the value of the key is. (Relations are ignored because nodes and ways already contain the data that is stored in relations.) Those 24 keys are:

1. Leisure
2. Club
3. Sport
4. Parking
5. Natural
6. Bench
7. Amenity
8. Shop
9. Studio
10. ATM
11. Shelter
12. Healthcare
13. Memorial
14. Manmade
15. Historic
16. Subway
17. Bicycle parking
18. Toilets
19. Craft
20. Office
21. Train
22. Covered
23. Land use
24. Tourism

We store each element as a key-value pair. For example: amenity-graveyard, natural-beach, leisure-park. If the POI is extracted from an OSM node, the node already has its coordinate data, so we record it. If it comes from a way element, we get the centroid of the way and assume that the POI is located at the centroid.

4. At the end of step 3, we have a list of POIs. Each row in this list contains the type of POI and its coordinate. Using the nodes from the network we extracted earlier, we find the POIs within 100 meters radius (this number can be arbitrarily chosen) of each node. We use the haversine formula to calculate the distance between the 2 points on a circle:

$$d = 2r \sin^{-1} \sqrt{\sin\left(\frac{lat_2 - lat_1}{2}\right)^2 + \cos(lat_1)\cos(lat_2)\sin\left(\frac{lon_2 - lon_1}{2}\right)^2}$$

where,

d = distance between points *1* and *2*,

r = radius of the sphere (Earth),

lat_1 = latitude of point *1* in radians,

lon_1 = longitude of point *1* in radians,

lat_2 = latitude of point *2* in radians,

lon_2 = longitude of point *2* in radians.

Since Earth is not a sphere, the distance calculated with the haversine method will contain an error. In order to hedge for that error, we look for POIs that are within the circle centered at the node centroid with 110 meters radius.

5. At the end step 4, we finally have an input **POI** matrix of size *n* x *p*, where *n* is the number of nodes and *p* is the total number of types of POI. The POI_{ij} element shows how many *j* type of POI we have within 100 m radius of i^{th} node.

After network extraction and POI collection steps, each node in our graph will have an attribute vector containing the POIs around it. From here on, we take a look at our trip data (OD matrix). Our origins and destinations, or zones, will be contained within the network we just extracted. The number of zones is usually smaller than the number of nodes since it is limited by how much trip data we have. The geometries of the previously mentioned 123 (62 from Manhattan and 61 from Brooklyn) zones are published by NYC as part of NYC Open Data.[6] The centroids of these zones are found by using *shapely* library for Python and are associated with the nearest node in the network. Please see Figure 4 for the drive network that has the zone centroids

[6] https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc

(zone-nodes) embedded as nodes. From here, we proceed to calculate the interzonal distance. If our trip data includes the get on and get off times, we can calculate the travel time and use it as a cost. If we don't have this data, we can calculate the distance between each node and use it as a cost. However, the distance should not be direct distance between 2 points in space and should take the geometry of the route into account. The graph extracted by OSMnx preserves the road geometries and we can calculate the shortest paths between each zone-node using *NetworkX* library for Python. After this process we have cost matrix with the size of z x z where z is the number of zones, now each of them is a node in our network.

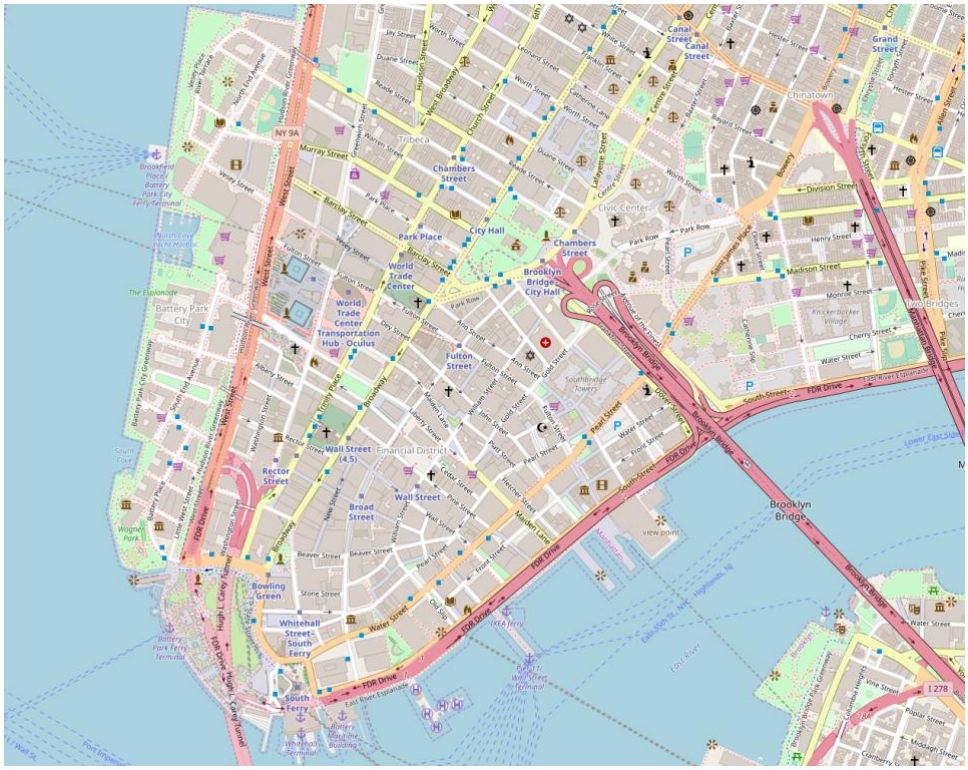

Figure 3a: A map tile on OSM

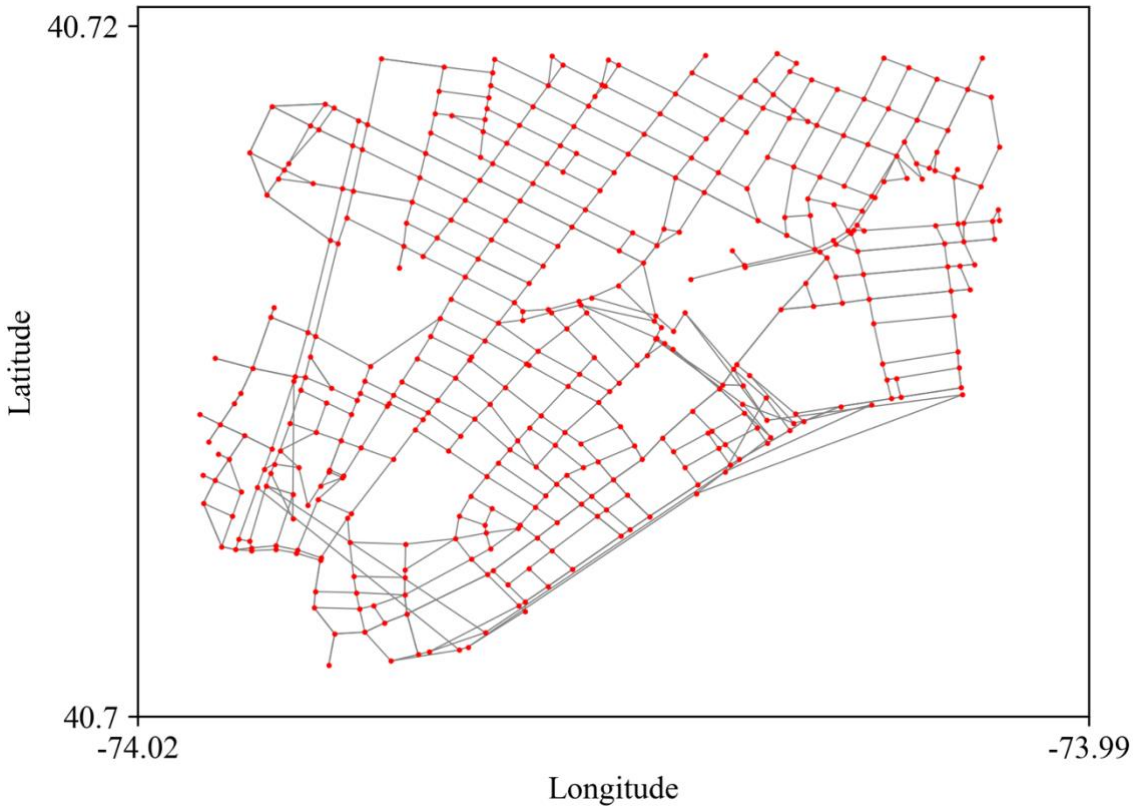

Figure 3b: Network representation of Figure 3a

Table 1: OSM data structure

Element	Data	Notes
Node	<node id="2568320815" lat="40.6949053" lon="-73.9809461" version="1" timestamp="2013-12-06T17:39:41Z" changeset="19309861" uid="1781294" user="Rub21_nycbuildings"> <tag k="addr:housenumber" v="31"/> <tag k="addr:postcode" v="11201"/> <tag k="addr:street" v="Fleet Walk"/> </node>	A node has an OSM id, latitude and longitude as attributes. Moreover, the node has tags and each tag is made up of k (key) and v (value). Here, we can observe that this node represents a house in the real world.
Way	<way id="690986719" version="1" timestamp="2019-05-19T05:59:04Z" changeset="70401645" uid="8369524" user="swappa"> <nd ref="42464621"/> <nd ref="42464614"/> <nd ref="42464610"/> <nd ref="5581955056"/> <nd ref="598043811"/>	Ways are a combination of nodes. Each <nd > found in the way element represents an OSM node. The reference attribute of the nd is the id of the node. Here, we can see that this way is a collection of 5 nodes, and it is a residential road.

	<tag k="highway" v="residential"/> <tag k="name" v="Prince Street"/> … <tag k="tiger:zip_right" v="11201"/> </way>	
Relation	<relation id="3971059" version="9" timestamp="2019-09-17T00:23:48Z" changeset="74553171" uid="429761" user="freebeer"> <member type="way" ref="297680846" role=""/> <member type="way" ref="25753498" role=""/> … <member type="way" ref="298198779" role=""/> <tag k="FIXME"	Relations are combinations of nodes and ways. While ways have <nd > elements, relations have <member > elements. Each member is either a node, a way, or in some cases, another relation. Each member will have its type, reference number (OSM id) and its role within this relation as attributes. In this case, the relation is a subway line. Here we see an interesting tag, which has the key value "FIXME". Although this relation was created by a user called "freebeer", another user has made a suggestion to change the value of a key-value pair. Since OSM is an open source project, the contributions from users are crucial to the accuracy of the data.

	v="This is a route=railway. If you wish to denote a passenger train (as this appears to do), use a route=train relation which uses this infrastructure"/> <tag k="from" v="Middle Village"/> <tag k="name" v="BMT Myrtle Avenue Line"/> <tag k="operator" v="MetropolitanTransportation Authority"/> <tag k="route" v="railway"/> … <tag k="wikipedia" v="en:BMT Myrtle Avenue Line"/> </relation>	

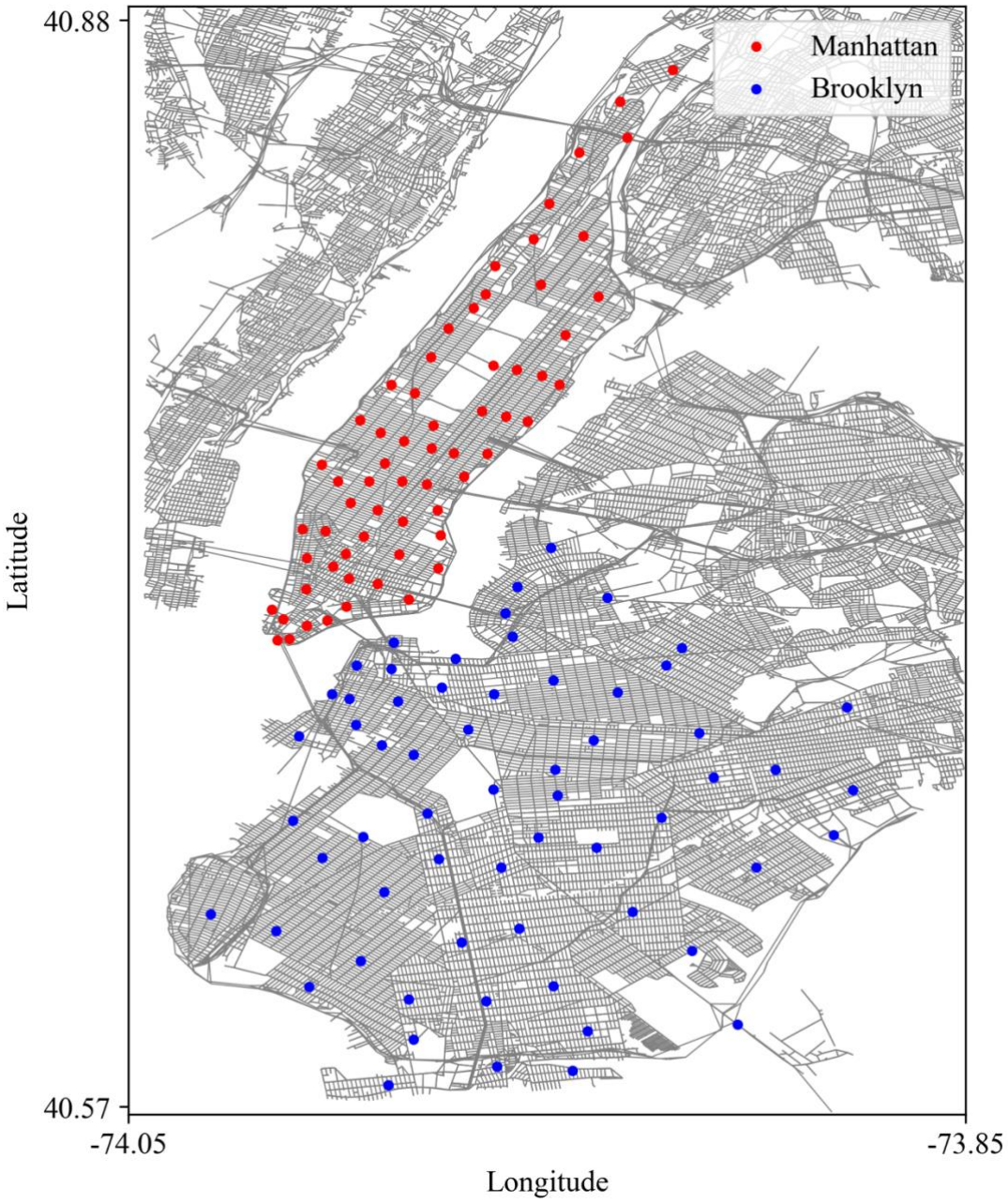

Figure 4: Zone centroids embedded as nodes in the drive network

4.3.Model architecture

The nodes in a city's transportation network cannot be thought independent of the other nodes, therefore, it is crucial to incorporate the nearby nodes' information to each node. This can be achieved by treating the network as graph and apply convolution. Graph convolution is a method used in machine learning for handling inputs where elements have interdependencies with each other. An example for this could be drug discovery, where machine learning is applied to identify molecules that are promising for drug development. Each molecule can be thought as a graph and the way each atom is connected to other atoms in the molecule defines the properties of the molecule. Hence, connections as well as atoms themselves should be used as an input for robust prediction. Our cities are extensive networks (graphs) where a zone's attraction or production is determined by how connected that zone is to the other zones in the city. Graph convolutional network (GCN) (or graph neural network) is a version of deep learning where graph convolution is applied at each layer. Let's assume that our city is represented as a graph of nodes (n) and edges (e). Each of our node has an attribute vector of POIs around them. The POIs for whole city are represented as a feature (or attribute) matrix $\mathbf{F}_{n \times p}$ where p is the number of total types of POIs. The graph of the city is represented by an adjacency matrix $\mathbf{A}_{n \times n}$ where A_{ij} is 1 if there is a link between nodes *i* and *j*, and 0 otherwise. Moreover, A_{ij} itself can have an attribute vector showing the length of the link or the time or cost it takes to go through that link. In our research, we simply use 1-0 representation of edges because we will later introduce link costs in our deep learning architecture. Then, at each graph convolution layer, our graph goes through transformation:

$$\mathbf{H}^{(i+1)} = \mathrm{f}(\mathbf{H}^{(i)}, \mathbf{A})$$

where,

$\mathbf{H}^{(0)} = \mathbf{F}_{n \times p}$ = input,

$\mathbf{H}^{(N)} = \mathbf{O}_{n \times q}$ = output of the graph convolution architecture.

As seen, the initial number of types of inputs (p) can be either increased or decreased down to q number attributes per each node. We can think of these final attributes as representative of the surroundings of the node (residential, commercial, industrial, and etc.). An effective function f that transforms the graph at each layer has been proposed by Kipf and Welling [25] as:

$$\mathbf{H}^{(i+1)} = \mathrm{f}\left(\mathbf{H}^{(i)}, \mathbf{A}\right) = \mathrm{ReLu}(\widehat{\mathbf{D}}^{-\frac{1}{2}}\widehat{\mathbf{A}}\widehat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\mathrm{i})}\mathbf{W}^{(i)} + \mathbf{b}^{(i)})$$

where,

ReLU = rectified linear unit activation function ($\mathrm{ReLu}(x) = \max(0, x)$),

$\widehat{\mathbf{A}}$ = **A** + **I** (Identity matrix),

$\widehat{\mathbf{D}}$ = node degree matrix of $\widehat{\mathbf{A}}$,

$\mathbf{W}^{(\mathrm{i})}$ = weight matrix at layer *i*,

$\mathbf{b}^{(\mathrm{i})}$ = bias matrix at layer *i*.

Here, identity matrix is added to **A** because **A** has a 0 diagonal (nodes don't have edges with themselves) which means we don't add the information of the node to itself when we are collecting information from neighboring nodes. By adding the identity matrix, the information of the neighboring nodes as well as that of node itself is passed on the next layer. Node degree matrix **D** is a diagonal matrix of size *n* x *n* showing how many links each node has. Here, $\widehat{\mathbf{D}}^{-\frac{1}{2}}$ is multiplied with $\widehat{\mathbf{A}}$ to normalize the attribute vector of each node. We use *deep graph learning (dgl)* library for Python for creation and convolution of our graph.

At the first layer of the GCN, a node is able to collect information of neighboring nodes of 1st degree. At the second layer, the neighboring nodes had already collected information about their neighboring layers at the first layer, hence the original node has access to the nodes of 2nd degree. Therefore, a node's information circle enlarges at each layer. When deciding how many layers that GCN will have, caution must be taken because if the number of layers is too small,

the nodes will lack critical neighboring information. On the other hand, if we have too many layers, each node's information will converge to an average value and we will lose critical local information. In our research, our graph convolution will have 4 layers (1 input, 2 hidden, 1 output), however further research is necessary for determining the optimal number of GCN layers for accurate prediction.

The result of the GCN is a matrix $\mathbf{O}_{n \text{ x } q}$, where each node has a vector of size q encapsulating the local POI data. In our research we have chosen q as 10. Although 10 may sound little, in order to run several tests in short time we have decided to keep the problem size small. From here on, **O** is re-organized to be fed to the upper deep learning layer, which is a multilayer perceptron (MLP). For each OD pair we have, we create a vector of size $2q$ + 1, where $1q$ elements are the POI output at origin, $1q$ elements are POI output at destination and 1 element is the cost (distance) between the origin and destination. The transformation function at each layer is

$$\mathbf{H}^{(i+1)} = \mathrm{f}(\mathbf{H}^{(i)}, \mathbf{A}) = \mathrm{ReLu}(\mathbf{H}^{(i)}\mathbf{W}^{(i)} + \mathbf{b}^{(i)})$$

where,

$\mathbf{H}^{(0)} = \mathbf{X}_{z^2 \text{ x } (2q+1)}$ = input,

z = number of zones (origins and destinations) (number of OD pairs = z^2),

$\mathbf{H}^{(N)} = \mathbf{Y}_{z^2 \text{ x } 1}$ = output of the MLP.

The output of the MLP is an OD matrix, where the elements are lined up in 1 column. The output is compared with the real values and loss is calculated. Thereafter, gradient of the loss with respect to weights and biases in GCN and MLP are calculated and those matrices are updated. We use *pytorch* library for python for training and testing the deep learning algorithm. Pytorch library records all the operations we do on inputs in our model and calculates gradients

automatically. We set the optimizer as Adam optimizer [26], loss criterion as mean square error loss[7] and learning rate as 0.01. Please see Figure 5 for flowchart of our deep learning model.

[7] https://pytorch.org/docs/stable/nn.html#torch.nn.MSELoss

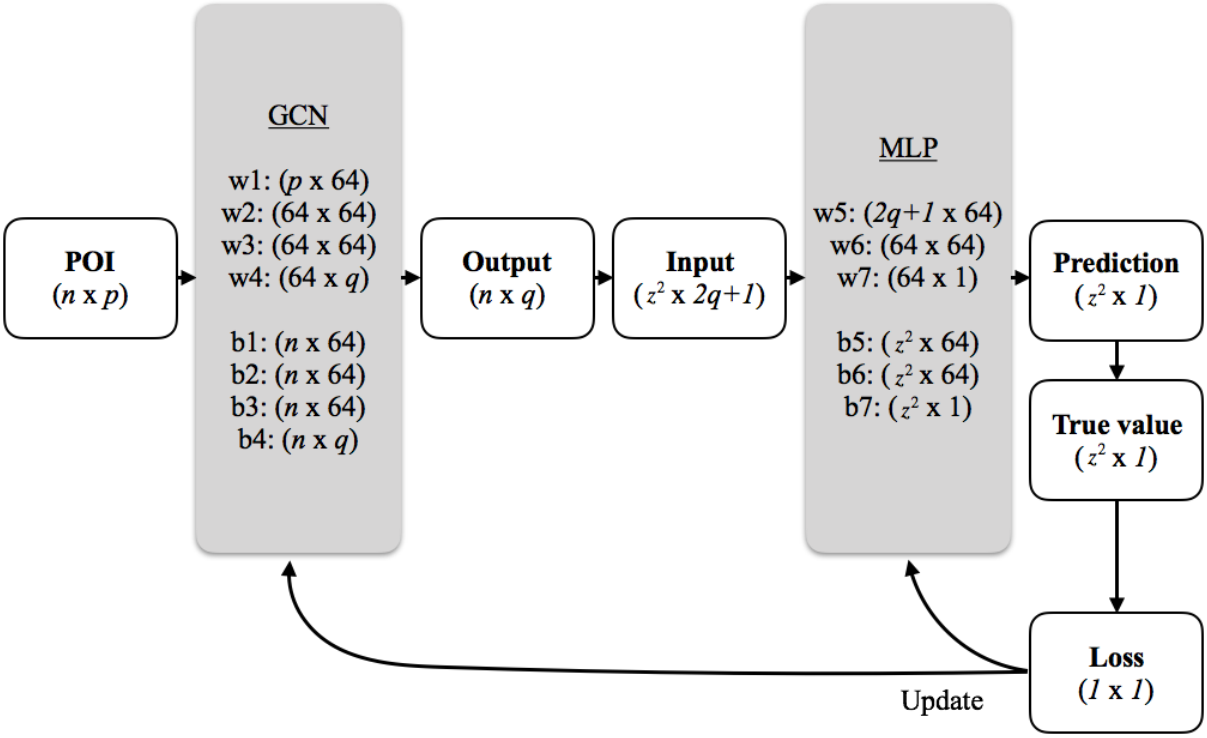

Figure 5: Deep learning model structure

5. Results

5.1. Training

We divide each of the datasets, Manhattan (M), Brooklyn (B), Manhattan and Brooklyn (MB), into training and testing subsets with a ratio of 80% and 20%. Before division we shuffle the dataset so that training and testing subsets will not have any idiosyncrasies. We train the algorithm for 500 epochs using the training subset. An epoch is an iteration during which the whole training subset goes through the model and weights are updated at the end of it. Please see Figures 6a, 6b and 6c for learning processes of these 3 areas. For M and MB, we see a normal learning behavior where the loss is converging to a value after some epochs. Test loss is closely following the training loss which shows that weights learned by using training dataset can explain the test dataset. We see a big jump in loss in B learning near the 400_{th} epoch, however it quickly goes back to converged value. Another peculiarity of Brooklyn learning is that, test loss seems to be less than the training loss almost throughout the learning. However, as the model reaches near 500_{th} epoch, training loss becomes less than the test loss, as expected.

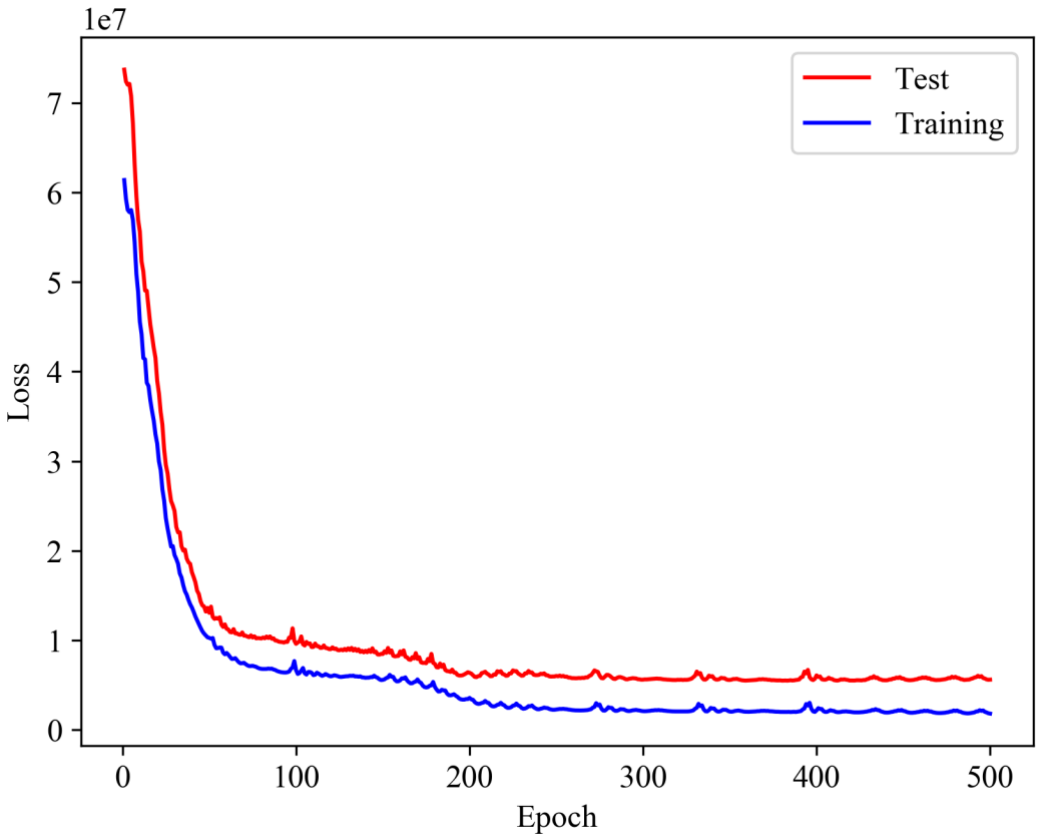

Figure 6a: Learning process of Manhattan travel demand estimation

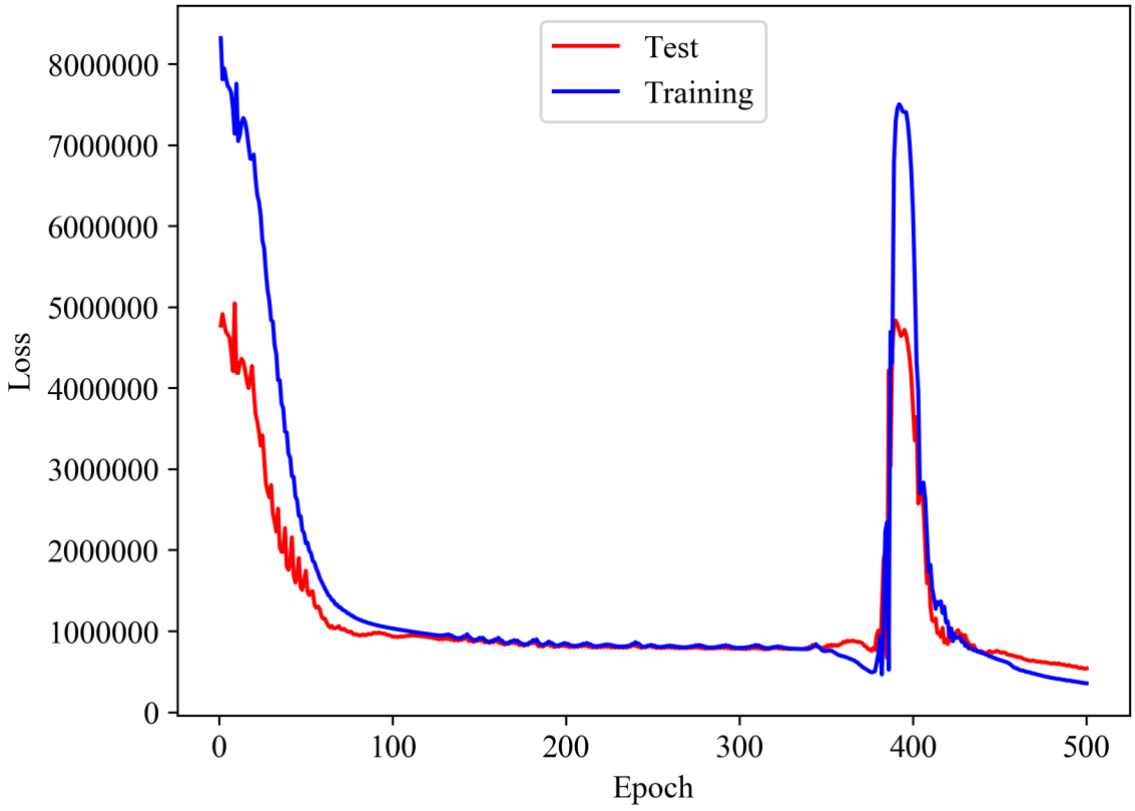

Figure 6b: Learning process of Brooklyn travel demand estimation

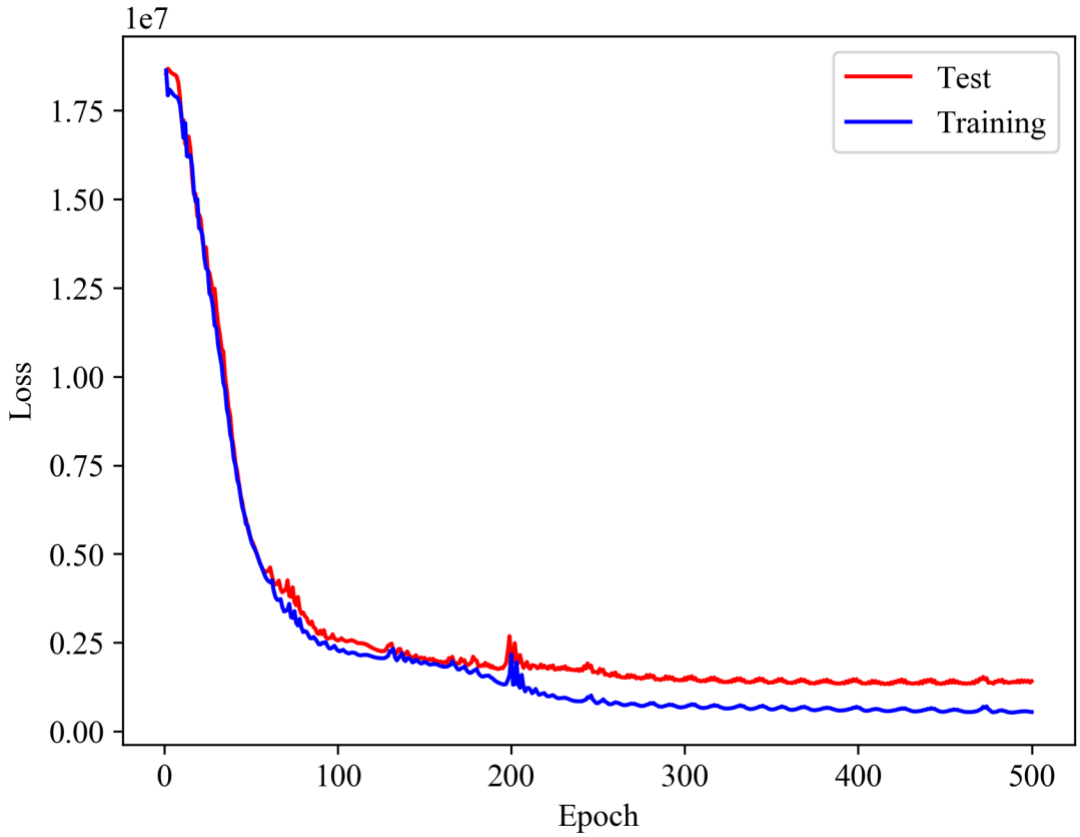

Figure 6c: Learning process of Manhattan and Brooklyn travel demand estimation

5.2. Validation

For validation, we calculate the R-squared of the test data at the end of the learning (500th epoch). M, B and MB reach R-squared of 89%, 86%, and 91%. Please see Figures 7a, 7b, and 7c for the distribution of labels (true value) and predictions for the testing subset (20% of the whole dataset).

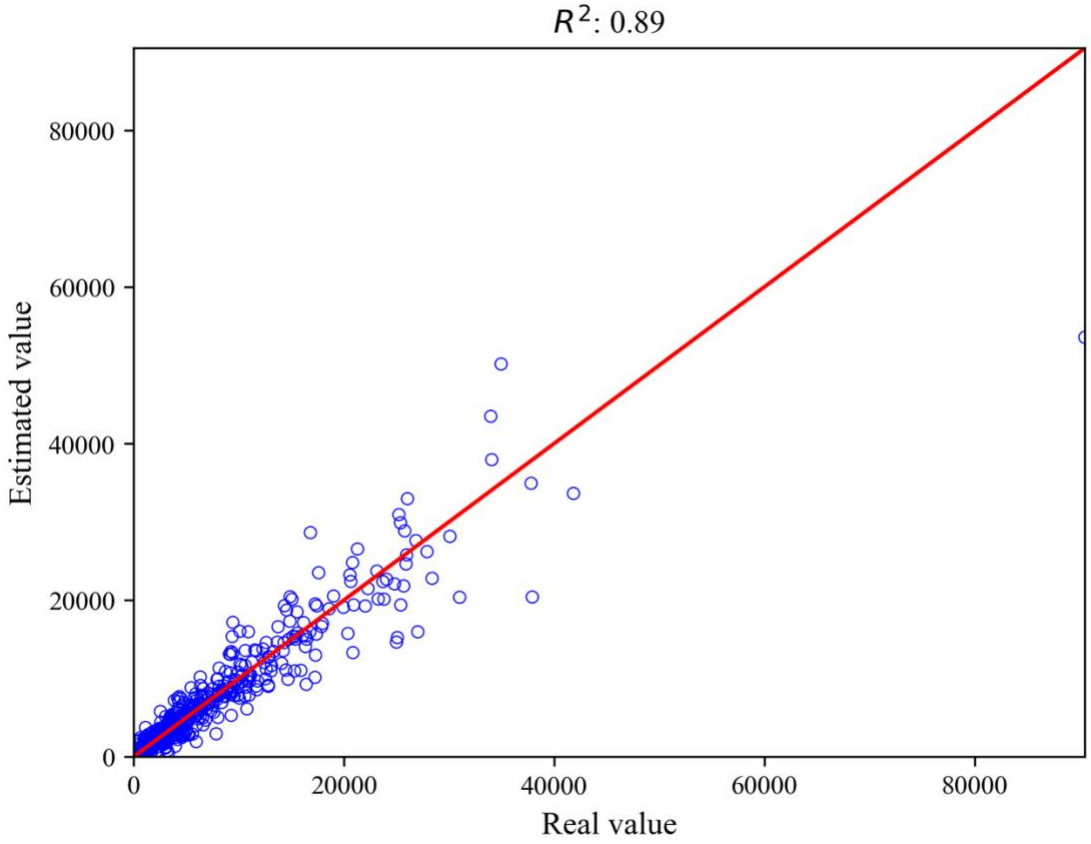

Figure 7a: Real vs. estimated value distribution for Manhattan

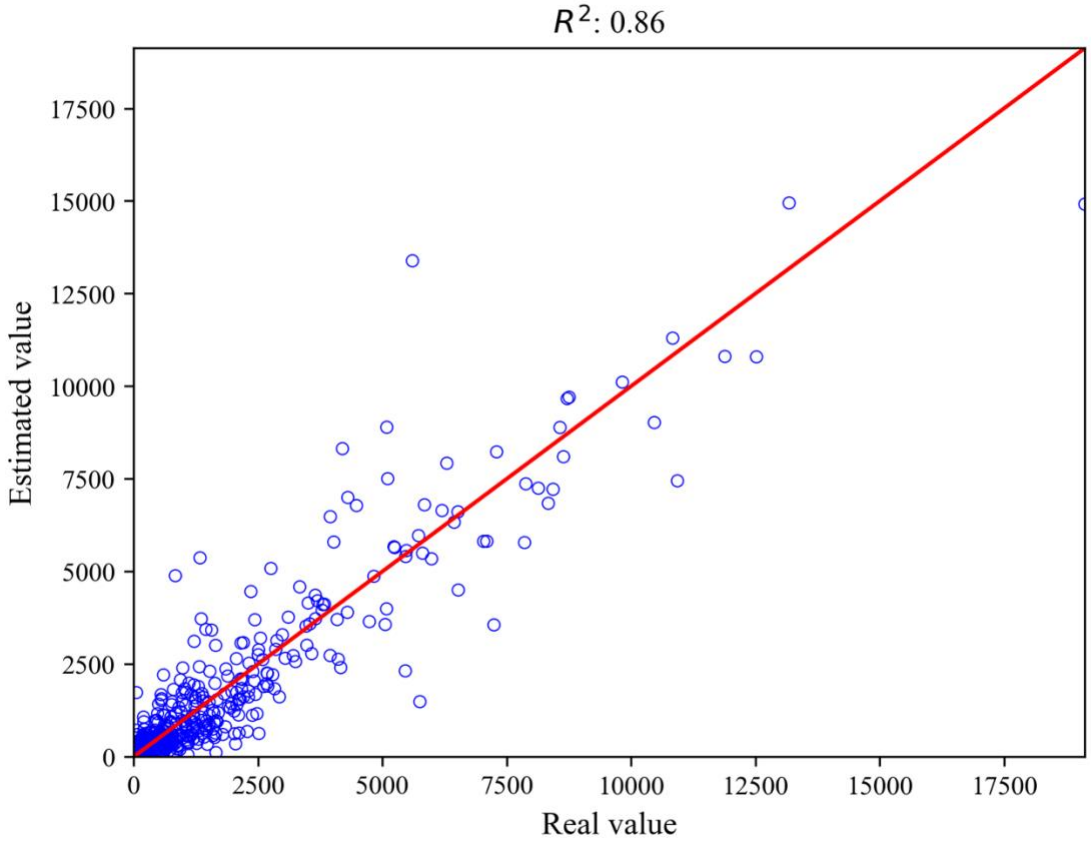

Figure 7b: Real vs. estimated value distribution for Brooklyn

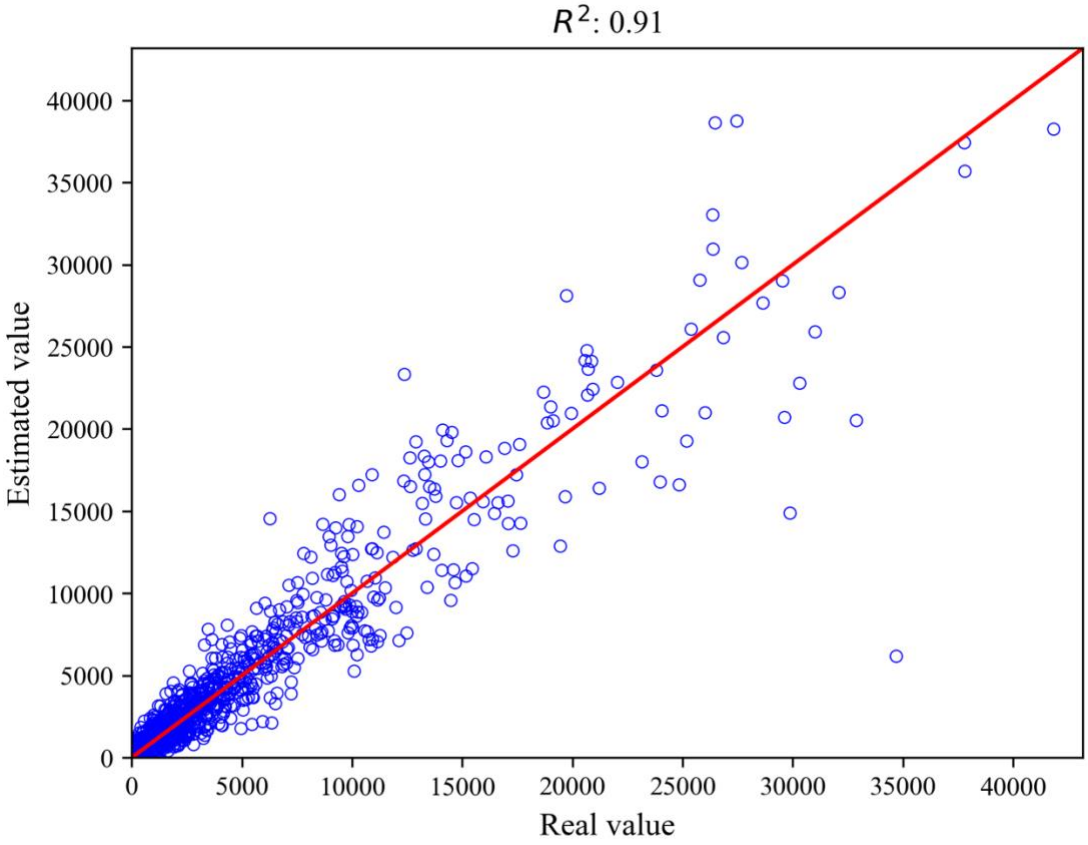

Figure 7c: Real vs. estimated value distribution for Manhattan and Brooklyn

5.3. Split ratio analysis

Split ratio is the ratio of data that we allocate for training from the whole dataset. Since this is a deep learning model, the more the training data, the better the prediction power, however we are aiming to understand the trip characteristics of our zones with this test. We postulate that, as the network gets smaller, the higher split ratio we are going to need. There are 2 reasons for this. Firstly, as network size decreases, we are missing some trips that either has an origin or destination outside the zone, which is decreasing the quality of our labeled data. Secondly, as the network gets smaller, the dataset gets smaller and some trip types cannot be found in the training subset, but only in the test subset. In other words, each trip gets more idiosyncratic and deep learning model may suffer low accuracy in low split ratios, simply because the model has not met some trip types while training. Following the same logic, larger datasets may reach high accuracy with low split ratios simply because the size of the dataset.

We implement the test by training the model with different split ratios from 10% up to 90% at 10% intervals. Please see Figure 8 for the results. We can confirm our hypothesis by observing that MB performs better than M and B for most of the split ratios. We can also see in Figure 8 that the model does not even need a large amount of data to reach satisfactory prediction. Even when the model uses 10% of the data for training, R-squared in all datasets is above 0.65.

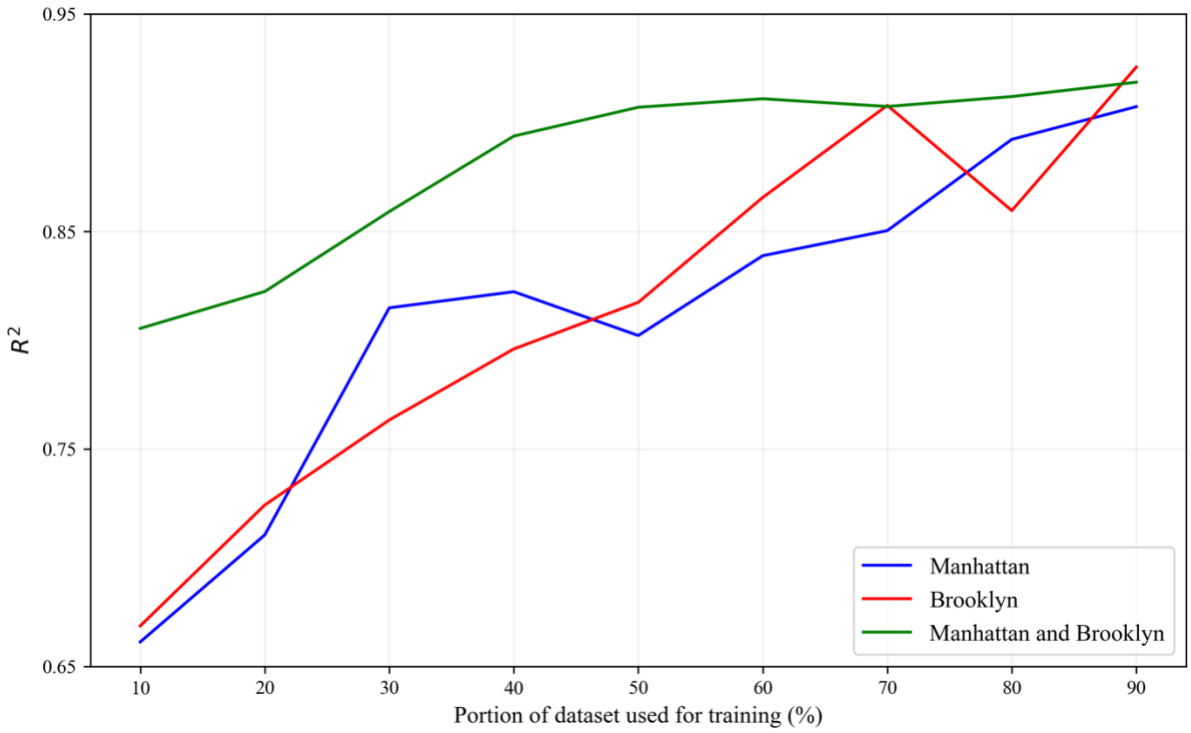

Figure 8: Model training with different train-test split ratios

5.4. Cost performance of the model

Until now, we have been shuffling the individual OD pairs and divide it into training and test datasets. However, in transportation planning, it is unlikely that we will have random OD observations in our cities that can be used as an input to our model. We will probably have data that has less diversity and some bias. In particular, we are likely to have data for all the trips that originate from some zones, and for others, we will have no information. Therefore, it might be realistic to shuffle the **zones**, split into the training and test subsets over the zones. For example, if we were to split the dataset with 80%-20%, the training dataset will have all the trips that are generated from 80% of the zones while the testing dataset will have the trips that are generated in other zones. If the model can satisfy this real-world condition, it can effectively replace the conventional OD matrix estimation method, hence we will be able to realize our goal of low-cost OD matrix estimation. We have trained our model with split ratios from 10% to 90% with 10% interval, 10 times at each step. Please see Figures 9a, 9b, and 9c for results of M, B and MB, respectively. We can see that as split ratio increases, deviation among results become smaller for M and MB. Please see Figure 9d for a comparison of M, B and MB. We can see that even when we use 10% of the zones, the model reaches a median R-squared above 0.4. Moreover, we can observe that R-squared reaches a saturation when 40% of the data is used. With this, we can conclude that, as long as we have 40% of the OD pairs observed, this model can successfully predict the unobserved OD pairs.

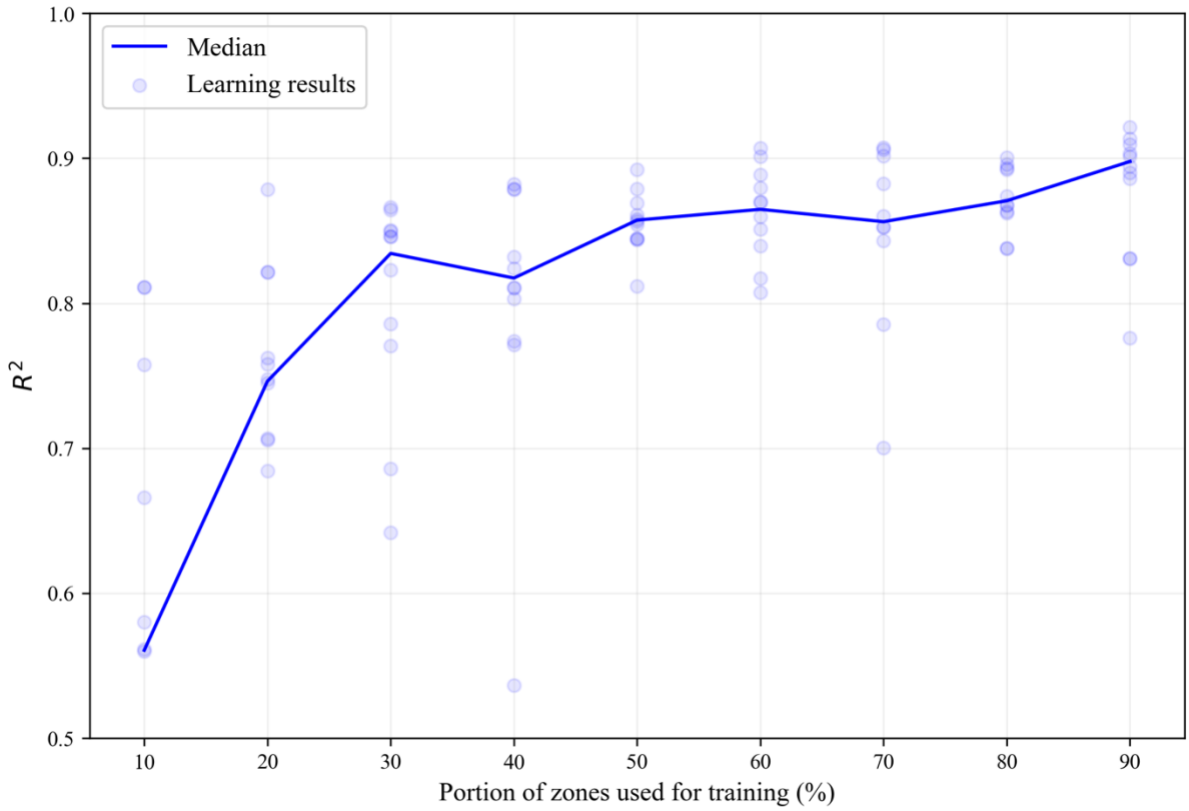

Figure 9a: Model training with different split ratios among zones for Manhattan (R^2 below 0.5 is not displayed)

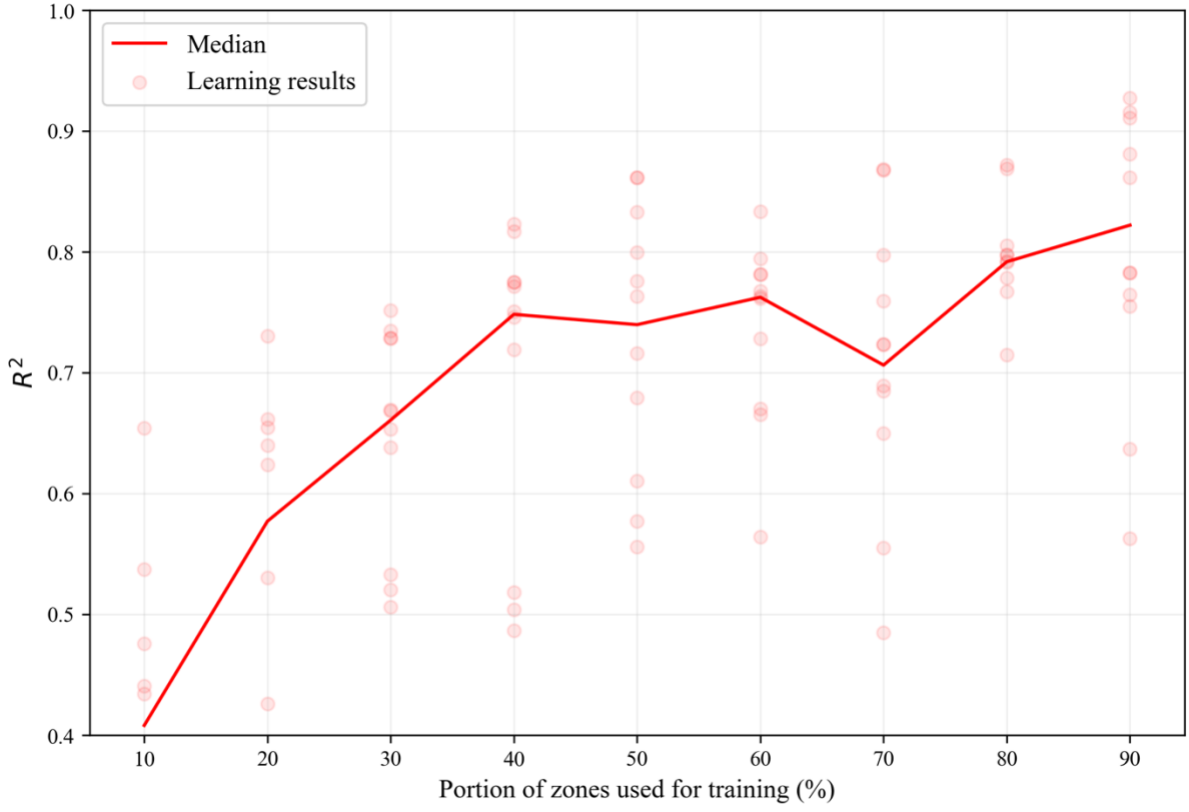

Figure 9b: Model training with different split ratios among zones for Brooklyn (R^2 below 0.4 is not displayed)

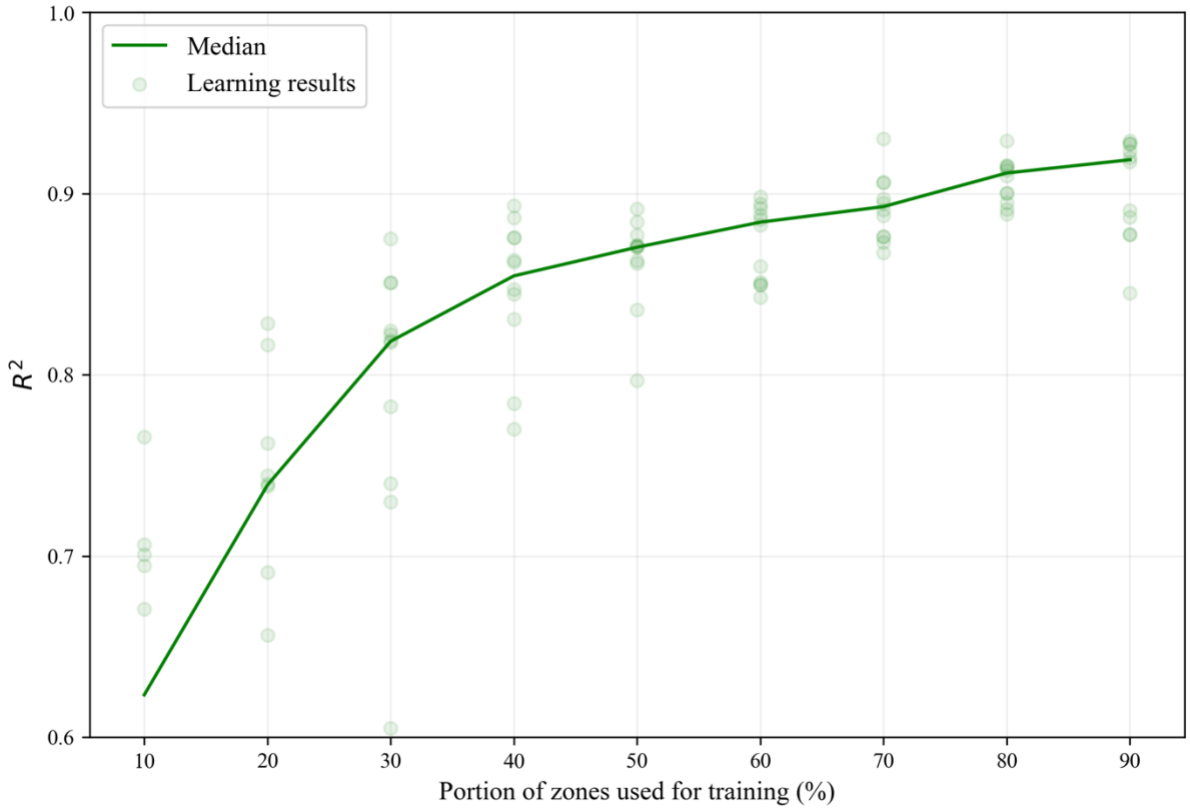

Figure 9c: Model training with different split ratios among zones for Manhattan and Brooklyn (R^2 below 0.6 is not displayed)

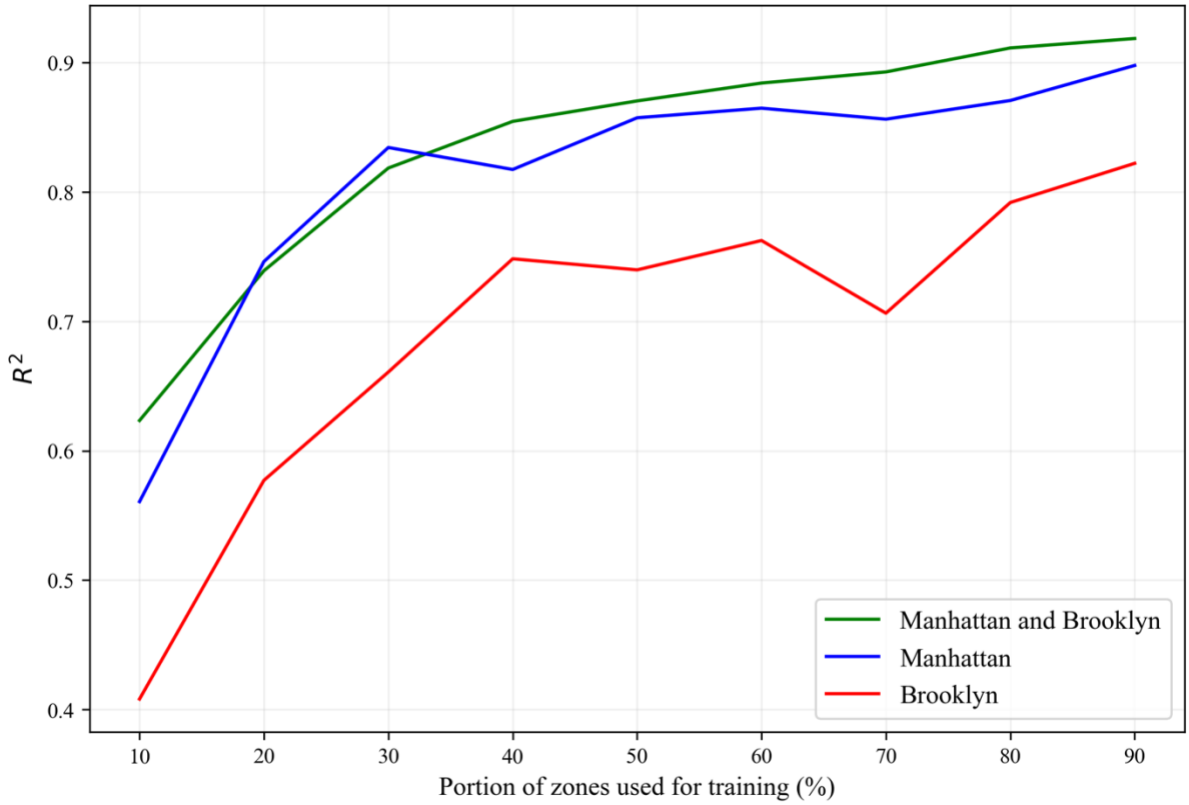

Figure 9d: Median R-squared results for various zone split ratios

5.5. Model performance with low-quality input data

Another real-world situation is that we will not have good-quality input to feed to our algorithm. As mentioned previously, POI data on OSM differs drastically from place to place. In order to test the scalability of our model, we conduct learning with different levels of input quality. In order to simulate POI data quality at different levels, we create an artificial deletion step just before the inference step in our model. At this step, each element of the POI matrix (n x p) may get deleted with a probability of d. With deletion, we don't mean to set that element to 0 but to an average value. Please see Figures 10a, 10b, and 10c for model results with different POI deletion probabilities. We see that POI deterioration has less effect on MB than M and B. For MB, we are even able to reach around 0.8 R-squared for 20% inferior POI data. We use the below R-squared formula.[8] While highest value it can get is +1, it can be arbitrarily large in the negative domain.

$$\mathrm{R}^2(\boldsymbol{y}, \hat{\boldsymbol{y}}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where,

$\boldsymbol{y}$ = vector containing predictions,

$\hat{\boldsymbol{y}}$ = vector containing true values,

y_i = predicted value,

$\hat{y}_i$ = true value,

$\bar{y}$ = average of predicted values,

n = label (or prediction) vector size.

[8] For calculation of R-squared, *scikit-learn* library for Python has been used and this formula is obtained from the official website of the library: https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score

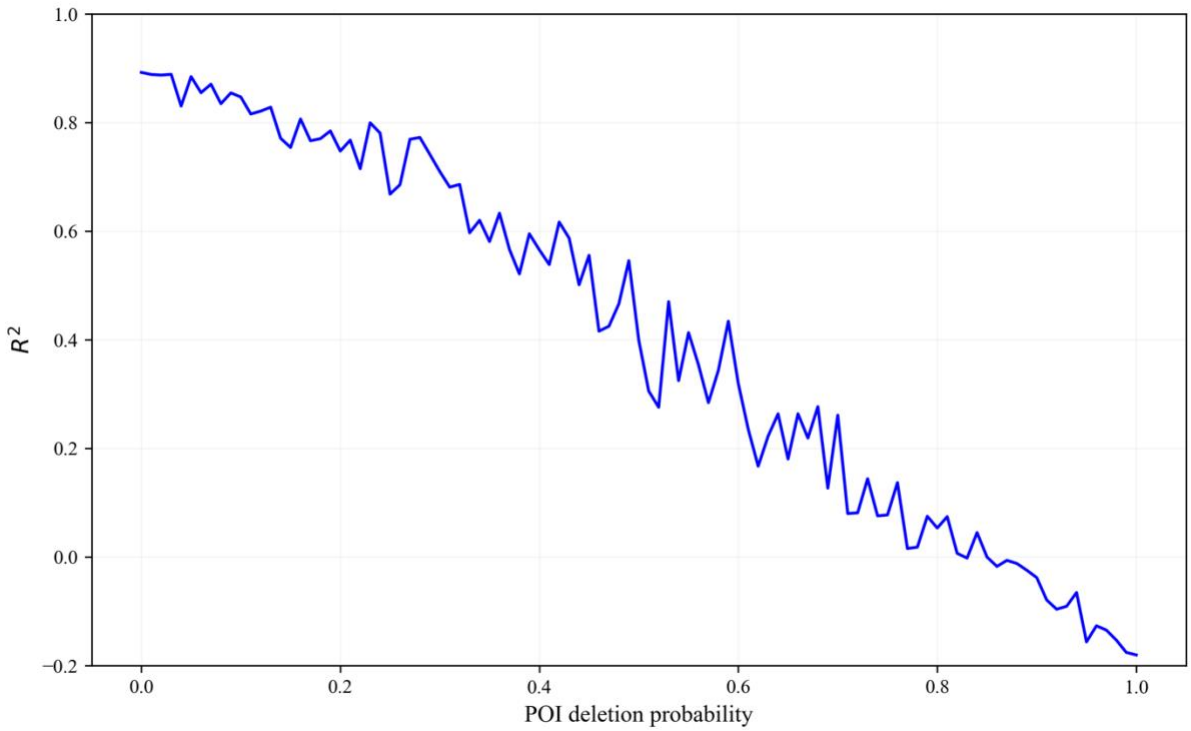

Figure 10a: POI quality analysis for Manhattan

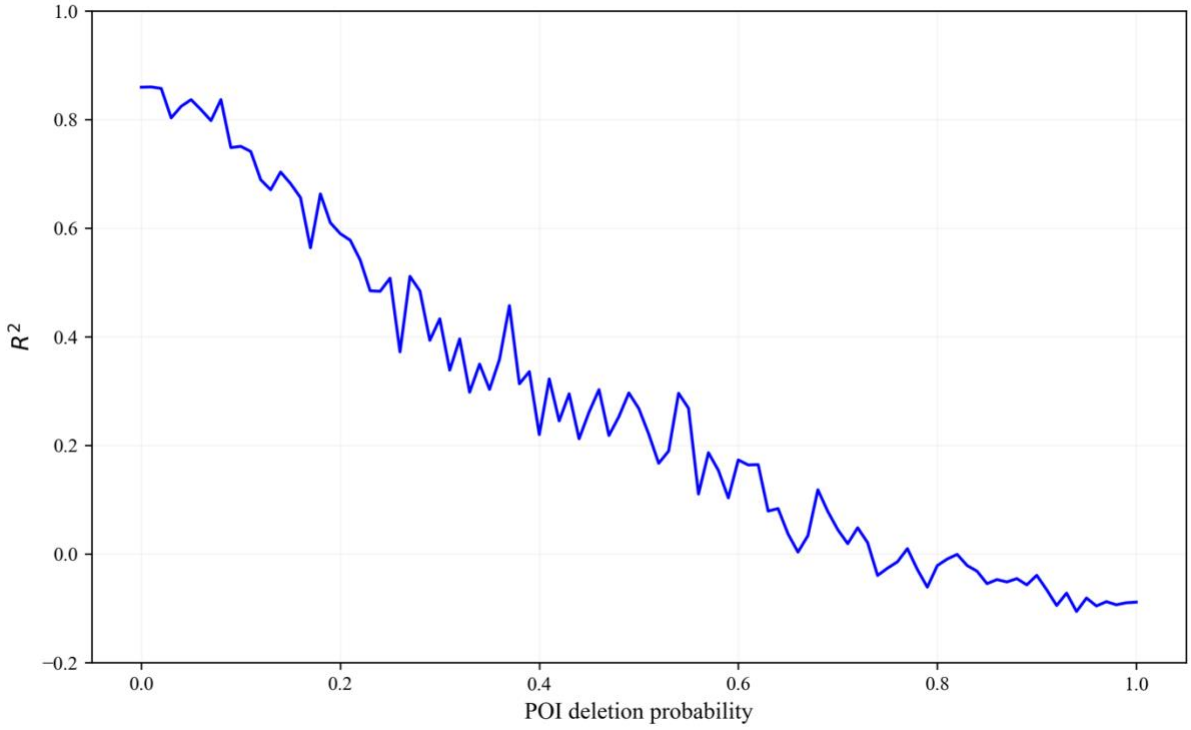

Figure 10b: POI quality analysis for Brooklyn

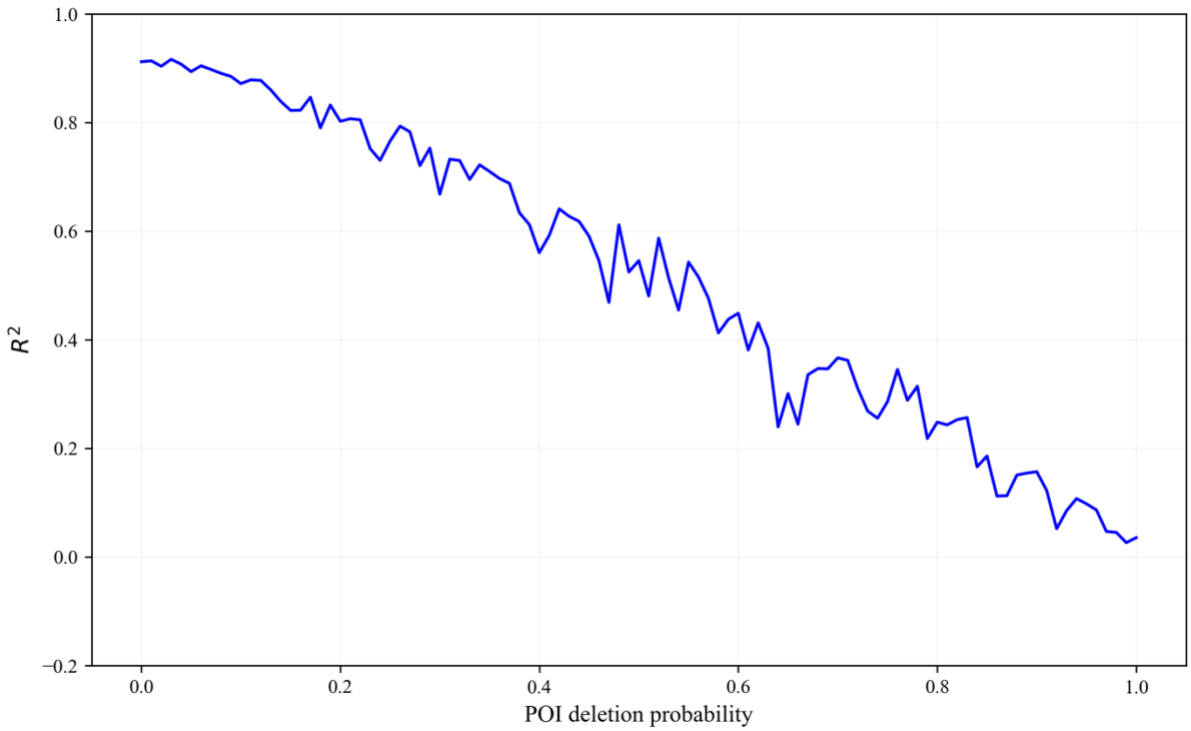

Figure 10c: POI quality analysis for Manhattan and Brooklyn

5.6.Cross-validation

It should be noted that our final objective with this research is to create a universal model that can work in cities with different characteristics. Until now, our training and testing datasets came from the same network. We want to see if the model trained on a city can successfully predict OD matrix of another city. We test this by doing 2 cross-validations between M-B and B-M. Unfortunately, the predictions made by the algorithm were not promising with R-squared below 30%.

5.6.1. Possible reasons for poor cross-prediction - 1: POI data quality difference

One reason for poor cross-prediction could be that 2 networks have similar characteristics, making prediction viable but POI data registered for Manhattan is far more than that of Brooklyn. On average, a node in Manhattan has 20.4 POIs around it, while a Brooklyn node has 6.2. Considering the relative popularity of Manhattan, it is likely that ratio of POIs that are recorded in Manhattan OSM is higher than Brooklyn. Moreover, Brooklyn, a less dense urban area might have less POIs than Manhattan. Since we don't have an exhaustive list of POIs for these places, it is impossible to confirm.

5.6.2. Possible reasons for poor cross-prediction - 2: different trip characteristics

Another reason for poor cross-prediction could be that Manhattan and Brooklyn have fundamentally different trip characteristic, making prediction inherently difficult. We conduct a test to see which POIs contribute most to the prediction quality in 3 places: M, B and MB. It might be the case that some POIs attract or produce many trips in Manhattan while it is insignificant in Brooklyn.

Similar to the POI quality test, we put a deletion step right before the inference, deleting (setting to average) each POI type one by one. The POI type is averaged for all the nodes in the network.

Then, we see which POI decrease the R-squared the most, using the case where none was deleted as a reference. Please see Figures 11a, 11b, and 11c for results that show the 9 POI types that decrease the R-squared the most when deleted, for M, B, and MB, respectively. By looking at the important POI types at M and B, we see that they don't have a single common POI type. These 2 places have indeed different characteristics. For example, a model trained in Manhattan puts too much importance *trees*, while trees are not that important in Brooklyn. It could also be the case that trees are not registered in OSM for Brooklyn, therefore the algorithm does not have good quality data to begin with. However, with the current dataset, we can conclude that an attempt to train a model in Manhattan and test in Brooklyn could be futile or any 2 places which have different determinants for trip distribution. However, as we have seen in MB case, the results are always better than those of M and B individually, which shows that if the model has an input from both areas, it can successfully predict any OD pair regardless of locations of O and D. Therefore, in order to create a deep learning model that can predict the OD matrix of a city, the algorithm should be exposed to a training dataset that includes the characteristics of the target city. Since it is difficult to quantify the characteristics of cities, or to find a city that has similar characteristics to the target city, the best way is to train the model with as diverse and many data as possible, which again shows that a deep learning model is as good as quantity and quality of its training data.

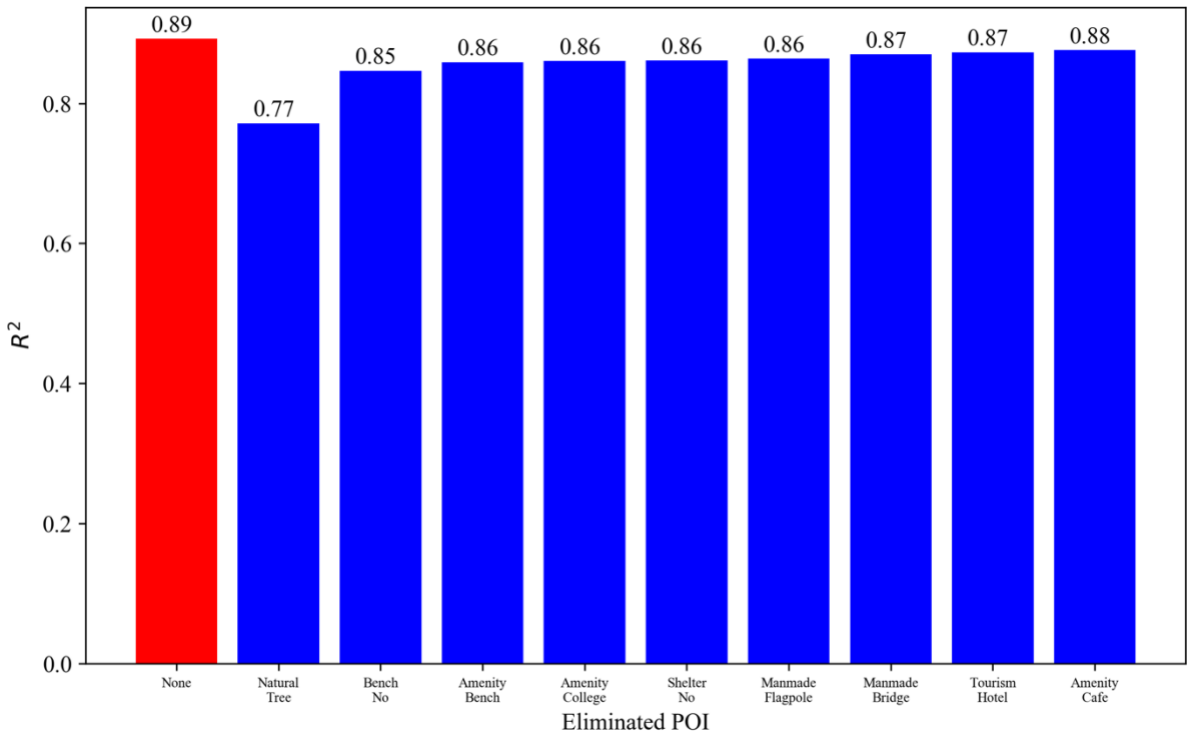

Figure 11a: Important POI detection for Manhattan

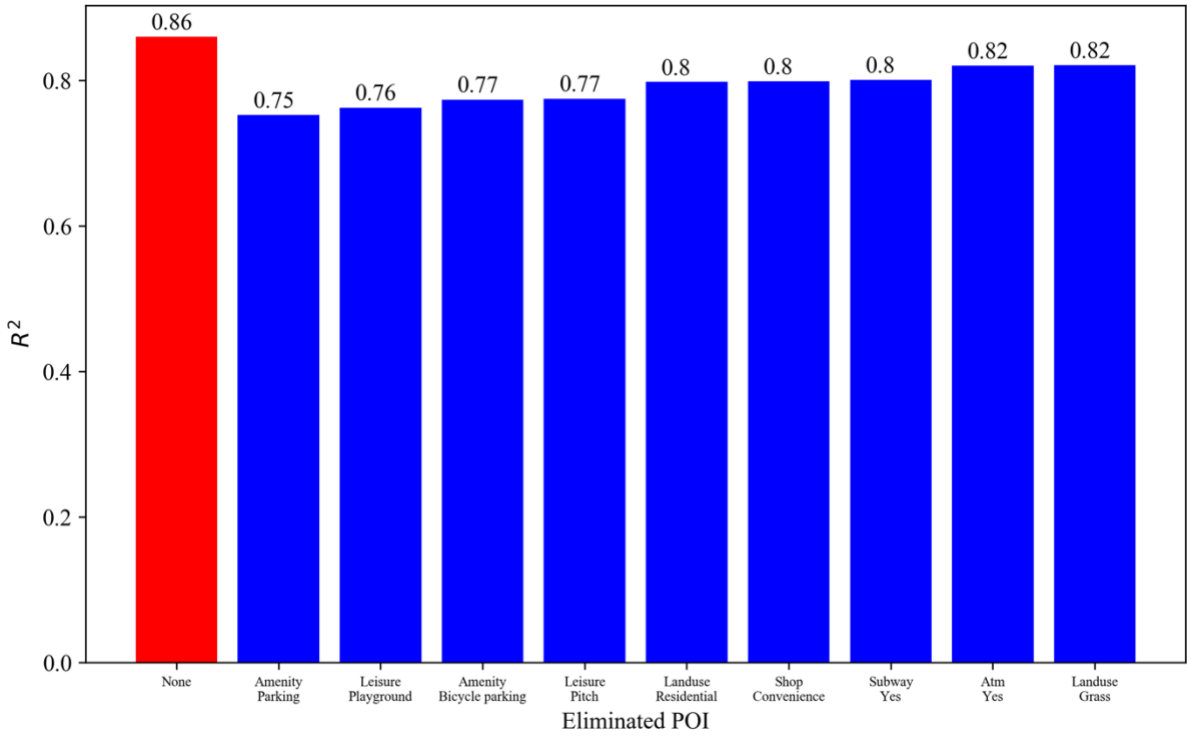

Figure 11b: Important POI detection for Brooklyn

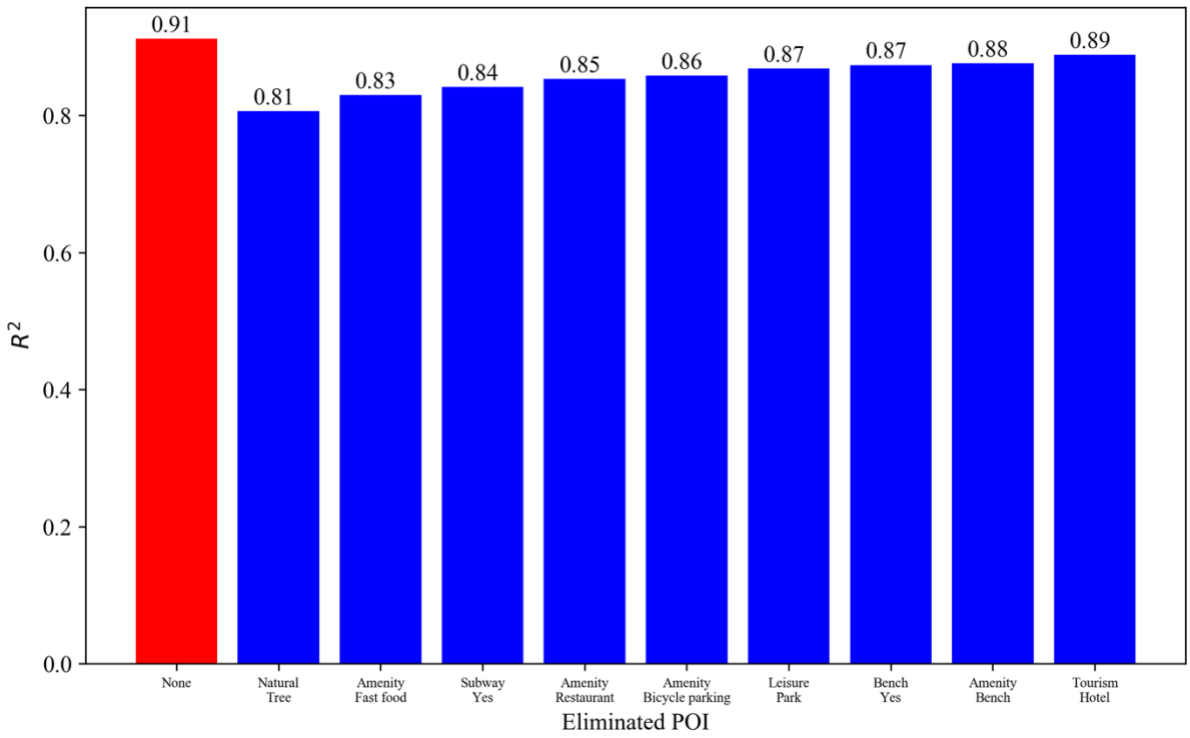

Figure 11c: Important POI detection for Manhattan and Brooklyn

6. Conclusion

Our motivation with this research was to develop a model for travel demand estimation that is cheap and scalable. We found that using different networks for training and testing could be fruitless for prediction accuracy. An effective way to achieve good prediction results is to have as large as possible training data which ideally includes similar trip patterns that is found in the target city. We have generated real world conditions by assuming that only some zones had trip data and model successfully filled in the missing OD pairs. This shows that our model can be used instead of conventional methods for OD matrix estimation. Considering that map data is virtually free, this model provides a cost-effective alternative.

6.1.Challenges

There are several challenges that need to be overcome in order for this model to be used in practice. Although we mentioned that this model does not require a costly input, it requires a lot of time to train the algorithm. For example, training the algorithm with MB data for different split ratios almost took a day on a relatively robust computer, even though we hosted our tensors on GPU. And the size of the problem was not very big compared with other large datasets. If we want to create a universal model, we should have a training dataset that is many times bigger than the MB case, which will require a large computing power. We know that city officials that want to train their own model may not have such resources or the knowledge to optimally use their resources (GPU memory allocation, parallel computing, and etc.).

Another challenge is the trip data. Thanks to some private and public authorities that make trip data publicly available, comprehensive models can be trained. As our society becomes more digitalized, we should expect to have more and more trip data that can be used for travel demand estimation.

The last challenge is the collection of good-quality input data. As mentioned before, city officials may have an exhaustive list of all the POIs found within their jurisdiction. For

individuals like us, we might use Google Maps if we want to have better data, although slightly expensive, for training travel demand prediction models.

6.2.Further work

Our methodology has so many parameters that can be fine-tuned for further improving the model. For example, we tested the model with different split ratios, however we weren't able to investigate the effect of learning rate on the model. Please see Table 2 for a list of parameters of our model. As further work, these parameters can be fine-tuned to create a more robust travel demand estimation model.

Table 2: Model parameters

Parameter name	**Value**
Deep learning	
Model architecture	Graph convolutional network (GCN) + Multilayer perceptron (MLP)
GCN hidden layers	2
GCN neurons at hidden layer	64
GCN activation function	ReLu
GCN output size	10
MLP hidden layers	1
MLP neurons at hidden layer	64
GCN activation function	ReLu
Epoch count	500
Split ratio	10%-90%, 10%interval
Optimizer	Adam optimizer
Loss criterion	Mean squared error loss
Learning rate	0.01
Advanced machine learning techniques	
Dropout rate	0
Max pooling	None
Mini batching	None
Data	
Locations	Manhattan and Brooklyn
Accepted POI types	24 keys (Total of 732 types)
Cost data	Distance
Road attributes	Not used
Trip	Taxi data
Trip data duration	1 month (January 2019)
Data collection	
Map data source	OSM
Nearby POI radius	100 m

References

[1] de Dios Ortúzar, J., & Willumsen, L. G. (2011). Modelling transport. John wiley & sons.

[2] Nguyen, S. (1984). Estimating origin destination matrices from observed flows. Publication of: Elsevier Science Publishers BV.

[3] Jung, W. S., Wang, F., & Stanley, H. E. (2008). Gravity model in the Korean highway. EPL (Europhysics Letters), 81(4), 48005.

[4] Oppenheim, N. (1995). Urban travel demand modeling: from individual choices to general equilibrium. John Wiley and Sons.

[5] Modi, K. B., Zala, L. B., Umrigar, F., & Desai, T. (2011, May). Transportation planning models: a review. In Proceedings of national conference on recent trends in engineering and technology.

[6] Fratar, T. J. (1954). Vehicular trip distribution by successive approximations. Traffic Quarterly, 8(1).

[7] Stouffer, S. A. (1940). Intervening opportunities: a theory relating mobility and distance. American sociological review, 5(6), 845-867.

[8] Barry, J. J., Newhouser, R., Rahbee, A., & Sayeda, S. (2002). Origin and destination estimation in New York City with automated fare system data. Transportation Research Record, 1817(1), 183-187.

[9] Jung, J., & Sohn, K. (2017). Deep-learning architecture to forecast destinations of bus passengers from entry-only smart-card data. IET Intelligent Transport Systems, 11(6), 334-339.

[10] Park, E. S., Rilett, L. R., & Spiegelman, C. H. (2008). A Markov chain Monte Carlo-based origin destination matrix estimator that is robust to imperfect intelligent transportation systems data. Journal of Intelligent Transportation Systems, 12(3), 139-155.

[11] Cheng, L., Zhu, S., Chu, Z., & Cheng, J. (2014). A Bayesian network model for origin-destination matrices estimation using prior and some observed link flows. Discrete Dynamics in Nature and Society, 2014.

[12] Llorca, C., Molloy, J., Ji, J., & Moeckel, R. (2018). Estimation of a long-distance travel demand model using trip surveys, location-based big data, and trip planning services. Transportation Research Record, 2672(47), 103-113.

[13] Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., & Damas, L. (2016). Time-evolving OD matrix estimation using high-speed GPS data streams. Expert systems with Applications, 44, 275-288.

[14] Alsger, A. A., Mesbah, M., Ferreira, L., & Safi, H. (2015). Use of smart card fare data to estimate public transport origin–destination matrix. Transportation Research Record, 2535(1), 88-96.

[15] Chan, J. (2007). Rail transit OD matrix estimation and journey time reliability metrics using automated fare data (Doctoral dissertation, Massachusetts Institute of Technology).

[16] Janzen, M., Vanhoof, M., Axhausen, K. W., & Smoreda, Z. (2016). Estimating long-distance travel demand with mobile phone billing data. In 16th Swiss Transport Research Conference (STRC 2016). Swiss Transport Research Conference (STRC).

[17] Bonnel, P., Fekih, M., & Smoreda, Z. (2018). Origin-Destination estimation using mobile network probe data. Transportation Research Procedia, 32, 69-81.

[18] Barceló Bugeda, J., Montero Mercadé, L., Bullejos, M., Serch, O., & Carmona Bautista, C. (2012). Dynamic OD matrix estimation exploiting bluetooth data in urban networkss. In Proceedings of the International Conference (pp. 116-121).

[19] Bast, H., Storandt, S., & Weidner, S. (2015, November). Fine-grained population estimation. In Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 1-10).

[20] Xu, F. F., Lin, B. Y., Lu, Q., Huang, Y., & Zhu, K. Q. (2016, October). Cross-region traffic prediction for china on openstreetmap. In Proceedings of the 9th ACM SIGSPATIAL International Workshop on Computational Transportation Science (pp. 37-42).

[21] Grippa, T., Georganos, S., Zarougui, S., Bognounou, P., Diboulo, E., Forget, Y., ... & Wolff, E. (2018). Mapping urban land use at street block level using openstreetmap, remote sensing data, and spatial metrics. ISPRS International Journal of Geo-Information, 7(7), 246.

[22] Zhao, L., & Kusumaputri, P. (2016). Openstreetmap road network analysis for poverty mapping.

[23] Moeinaddini, M., Asadi-Shekari, Z., & Shah, M. Z. (2014). The relationship between urban street networks and the number of transport fatalities at the city level. Safety science, 62, 114-120.

[24] https://www1.nyc.gov/site/planning/planning-level/nyc-population/current-future-populations.page [Accessed on January 30 2020]

[25] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[26] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.