Aaalborg University – Computer Science (IT)

Student: Ivan Iliev

Supervisor: Bin Yang

Summary of Master Thesis

This Master Thesis belongs to the specialization branch of Database Technologies (spDT). Moreover, it focuses on the research field of Stochastic Taxi Demands prediction. In details, this thesis proposes a multitask probabilistic machine learning model which can approximate distributions of demands between source regions and destination regions. Furthermore, the proposed model takes as an input a sequence of origin-destination matrices (OD-matrix) and it outputs distributions of approximated demand for each of the OD pairs in the OD-matrix.

The proposed model can approximate these distributions with the help of Mixture Density Networks (MDN). MDNs are artificial neural networks that can approximate conditional probability densities. In this project's case, MDNs are used to approximate taxi demands, and these demands are represented via Gaussian Mixture Models (GMM). Additionally, the proposed model relies on Long Short-Term Memory Networks (LSTMs) which are a state-of-the-art neural network that can learn long-term dependencies between the sequential inputs. Finally, as already mentioned, it is a multitask model, meaning, there is a task which approximates the demand for each of the OD-pairs of the OD-matrix, and all of these tasks are combined via a shared layer and can learn from each other so the overall accuracy of the model is improved.

This 10th semester Master Thesis is an extension of the work done during the 9th semester. In the 9th semester it was discovered that MDNs are sensitive type of neural networks and to train them properly, a dropout regularization technique needs to be used, also, they seem to work very well with big batch sizes and a lot of training epochs. For the Master Thesis it was decided to try to build a model which is a multitask model and it will include multiple MDNs as output layers, each of these MDN layers is approximating demand for a particular OD-pair. Interestingly, what was discovered during the experiment when the multitask model was built is that training an MDN can lead to a NaN loss. Especially, in the case when an MDN layer is directly connected with a ReLU activation layer. It turned out, there are papers which propose techniques for solving this NaN problem, these techniques were not used and implemented in the proposed model. Instead, LSTM was directly connected to the MDN layer(s), and the NaN loss issue never appeared. Eventually, it was discovered that a multitask model with multiple MDNs as output layers can be build and trained successfully. Interestingly, in the ablation study it was discovered that the multitask MDN model performs worse in terms of accuracy compared to a multitask single value prediction model. The exact reason for this is still not know, even though, it was examined that identical models, one of which outputs single value of demand, another with an MDN as an output layer and approximating distribution of demands, have the same RMSE and MAE when evaluated.

Predicting Stochastic Demand using a Multi-Task Recurrent Mixture Density Network

Ivan Iliev
Aalborg University
Aalborg, Denmark
iiliev18@student.aau.dk

Abstract

Recently, on-demand ride-sourcing mobile applications have increased their popularity. Being able to accurately predict taxi-demands can reduce the waiting times for the passengers, additionally, help the service providers with the organization of their taxi fleet. This paper proposes a moderate machine learning model, which is able to predict demands between sources and destinations for a small neighbourhood in Manhattan, New York. The components that have been used to build the probabilistic machine learning model are Long Short-Term Memory Neural Networks (LSTM), Mixture Density Networks (MDN) and Multi-Task Learning.

Introduction

Ride-haling services provided by Uber and DiDi have become a common and suitable way of transportation for the citizens of the big metropolises. These companies manage to provide services to their customers with the help of mobile applications. As a result, the orgin and the destination of the customer's order are known in advance (i.e. before the taxi has arrived to pick the customer).

The problems that these companies are facing are to reduce the waiting times for their customers, also, to reduce the number of the empty taxi-rides. As a consequence, by solving the above mentioned problems the ride-sourcing service providers will be able to increase their profits and customers' satisfaction, additionally, reduce the traffic congestion and the carbon emissions (i.e. benefiting the climate).

Unquestionably, there is a lot of incentive to have a forecasting model which can accurately predict future taxi demands. A lot of research has been done in this area of designing intelligent transportation systems. In the beginning, researchers were building predictive models [10] based on GPS trajectory data. However, a proper model cannot be built solely upon trajectory data, as the trajectory data cannot reveal the exact mobility patterns of the passengers [16]. In detail, in the GPS trajectory data it is hard to make a distinction between an actual customer order and an empty taxi-ride. Lately, the researchers have been given spatial-temporal data which contains taxi requests and orders. Interestingly, after doing some data pre-processing a big metropolis such as New York or Beijing can be partitioned into many non-overlapping regions. Then all of the taxi orders can be grouped by a source and/or destination region(s). Papers such as [17] managed to achieve great results in predicting the number of taxi requests that can arise from the different regions of a city. Nevertheless, being able to predict only demand from an origin region is not enough. Instead, a taxi company can better optimize their fleet if the demand between an origin region and a destination region can be predicted.

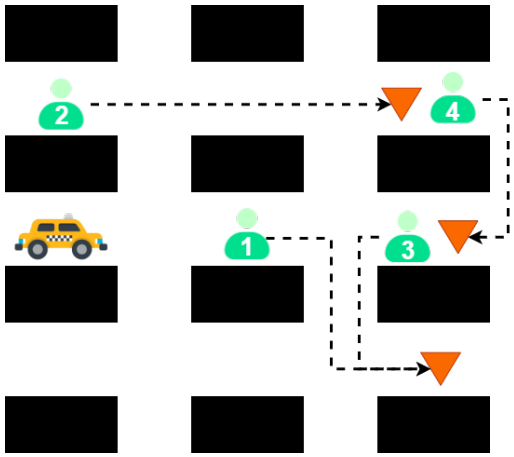

Figure 1: Ride sharing example

To demonstrate, the icons in green in fig. 1 are predicted by some machine learning model future customers. The orange triangles are their desired destinations (customers 1 and 3 have the same desired destination) and the taxi icon is the taxi that has to satisfy the requests of the customers. Each of the customers in the fig. 1 is to travel to a different region than their source region. There are a few ways that the taxi driver can approach these tasks (i.e a task meaning to pick-up a customer from their source region and drop-off him off at the desired destination region), depending on the information that the taxi driver has.

1. In the first case, let's imagine that the desired destinations of the customers are unknown for the taxi driver. For instance, a model which can only predict demand from a

source region is used. Logically, since the destinations (the orange triangles) are not known, what a taxi driver can do is to follow a greedy approach and always pick the customer who can be reached the fastest. Initially, the customer that the taxi can reach the fastest in fig. 1 is customer 1. Following, the sequence of picking-up and dropping-off the customers for the taxi driver will be: [1, 3, 4, 2]. As it can be seen in fig. 1 if the taxi driver follows the greedy approach, there will be an empty-ride between the different orders. In detail, as the taxi driver delivers customer 1 to the desired region, the taxi driver has to go to the upper regions to pick customer 3, following, as customer 3 is delivered, the taxi has to go for customer 4, finally, it will be customer 2's turn. It is obvious, that there will be an empty ride between the different customer orders and it is because of the wrong approach of the taxi driver, which is related to the fact that the destinations of the customers are unknown. Therefore, such models that predict demands only from origins and discard the destinations can hardly be used.

2. In the second case, a machine learning model which can predict demand between origins and destinations is used. Moreover, the destinations of the passengers are known and the taxi driver is aware of them. Therefore, as it can be seen in fig. 1 there are 3 orders which can be done one after another (i.e. in the order of [2, 4, 3]) and the taxi driver will not be doing any empty-rides between them. Interestingly, these are origin-destination pairs that are linked. In detail, the taxi driver will start with customer 2, then, as customer 2 is delivered, customer 4 can be picked without having to do an empty-ride to another region, eventually, there will be only one empty-ride in total and will be the one to pick customer 1. As it can be seen it is vital to know the destination regions of the customers. In the first case, there are 3 empty-rides for the taxi driver, in this case it is only 1 empty-ride.
3. In the third case, just as in the second case, the demands between sources and destinations are known. The difference is in this case the concept of ride-sharing is implemented. Meaning, the taxi driver can fulfill all of the customers' orders without having to do even a single empty-ride. As it can be seen in fig. 1, customers 1 and 3 have the same desired destination. Therefore, if they are willing to share a taxi, when the taxi driver picks up customer 3, the taxi may go to the region of customer 1 to pick him up, eventually, both of them will be delivered to the desired destination region. This final case presents the most optimal way that these orders can be fulfilled. According to [2], 80% of the rides in Manhattan can be shared by two people. The only downside of it is it requires a detour for one of the customers.

As it is already shown in this paper (i.e. with the examples above) and mentioned in others [16] and [9], only knowing the demand that can arise from a particular source region of a city is not enough. Demand between origin and destinations needs to be known. When the destinations of the customers are discarded (i.e. as in the first case example), the mobility patterns of the passengers cannot be revealed, and that makes it hard for the taxi companies to optimize their fleet.

Recently, papers such as [16] and [9] propose models which can predict demands between origins and destinations. They present the problem as **OD-matrix** (Origin-Destination matrix) prediction or Origin-Destination demand prediction. This paper, also focuses on the problem of OD-demand prediction. In comparison, with the other models which can only predict a single value of demand between an origin and a destination, the proposed model in this paper uses a **MDN** (Mixture Density Network) [3] and can approximate a distribution of demands between an origin and a destination. The motivation for using MDN is it has proven to be sufficient for creating probabilistic predictive models for various kinds of problems, such as speech synthesis [18], wind production [7], price volatility prediction [13] and others.

MDNs have the ability to approximate distributions and mixture models of any kind. In this project's case, when the demands data is fitted into a distribution, then, there is a great chance this distribution will be a multi-modal one. For instance, between an origin and a destination there can exist a two or three very different demand outcomes which have an equally high probability of happening. This is the kind of information which the MDN will be able to preserve. In comparison, a single value prediction model will average this multi-peak information, hence, it will be lost. Interestingly, when a model can approximate distributions of demand, then, that model can be used to design taxi-fleet optimization algorithms. Later in this introduction, simple statistical analysis have been done and will show multi-modal distributions and other kind of distributions exist in the data.

In the example in fig. 1 for simplicity reasons, the demand in each area was represented only of a single customer. In reality, the demands are varying, and in the central areas of a city during peak hours there can be even up to 80 persons who are willing to take a taxi ride between two particular locations. Most of the times, these varying demands are represented with multi-modal distributions, which are distributions with multiple peaks. Therefore, for the proposed in this paper model, it was decided that the demand distributions it will approximate will be mixture model distributions of type **GMM** (Gaussian Mixture Model) [12] with more than one Gaussian components. The motivations for choosing exactly GMM and not some other kind of mixture model are the following: firstly, there are many papers that have proven using MDN to approximate GMMs is optimal and produces accurate results [13, 7, 18]. Secondly, the distributions that can be observed in the historical data are of many types (Multimodal, Poisson, Normal, Exponential and others).

To demonstrate, let $S = \{v_1, v_2, v_3, ..., v_n\}$ be a sample series of hourly taxi demands between a particular origin and a particular destination in Manhattan, New York. (The series is sampled from the 2018 NYC Yellow taxis data set). Then, a simple statistical analysis can be done on the series S with the help of the following **PDF** (Probability Density Function) called $HPDF$ (abbreviation for Historical PDF):

$$HPDF(v_{observed}) = P(v_m | v_{m-1} = v_{observed}) \quad (1)$$

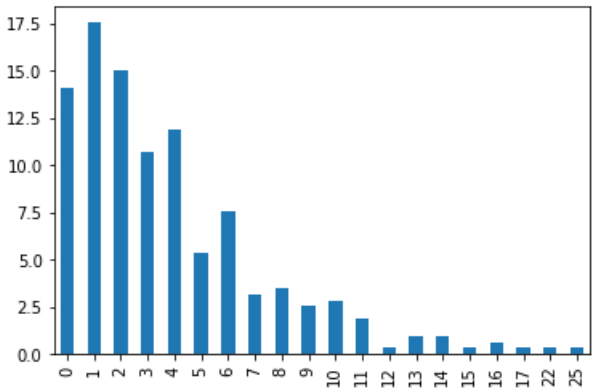

Figure 2: Historical taxi demands PDF when $HPDF(3)$

Fig. 2 represents a historical taxi demands PDF when $HPDF(3)$, the x-axis represents the different taxi demand outcomes and the y-axis represents the probability of each outcome in percentages. To clarify, what the function $HPDF$ does is showing taxi demands for the next hour based on the previous hour, where the previous hour is equal to particular value (in the case of fig. 2 that particular value is 3). Meaning, fig. 2 shows what kind of taxi demands can occur given that in the previous hour a demand of 3 taxi orders has been observed. Interestingly, fig. 2 looks a lot like a Poisson distribution. Next, it is examined what will be the historical taxi demands PDF when $HPDF(30)$.

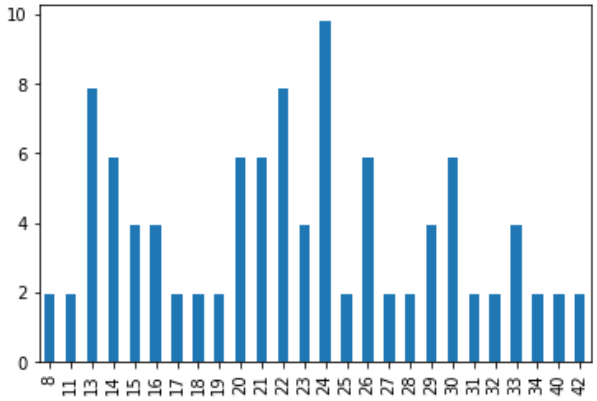

Figure 3: Historical taxi demands PDF when $HPDF(30)$

Fig. 3 represents the historical taxi demands when in the previous hour a demand of 30 taxi orders has been observed. As it can be seen, the distribution in fig. 3 is very different from the distribution in fig. 2. In contrast, the distribution in fig. 3 looks similar to a Binomial distribution or to a GMM with 2 components. Interestingly, there is an equal chance of 8% that the demand will be of 13 or 22 people. These are the very different outcomes that have an equal chance of happening and this is the kind of information that should not be neglected. Also, the motivation for using MDN and not a single value prediction model which will average all of the outcomes and use that as a prediction. The more the data is being explored, the more different kinds of distributions can be found. Eventually, after this analysis have been done and for simplicity reasons - GMM was selected as the kind of distribution to represent the demand between an origin and a destination.

The data set which was used to train the proposed machine leaning model is the 2018 NYC Yellow-taxi data set (using only the data for the Manhattan borough). In the data set - Manhattan has been split into 60 non-overlapping regions, which leads to a possible $60 \times 60 = 3600$ origin-destination pairs for which demand can be predicted. Nevertheless, it was decided that a simple predictive model will be built which will focus only on small subset of the regions. One of the reasons for the decision, is due to the time limitations of the project. The other reason is that there aren't many papers which have tried to build a Multitask-MDN machine learning model or any machine learning model which can approximate multiple distributions at the same time (especially in the field of designing intelligent transportation systems). Additionally, according to [9], in Beijing there are city management rules which bound the taxi drivers to a small number of regions. Moreover, if the taxi drivers are to receive an order which is out of their bounds, they have to reject it. Therefore, this model can easily be used by the taxi companies which operate in metropolises where driving restriction policies of this kind are in place.

The proposed in this paper model is a Multitask LSTM-MDN model. Multitask [5] meaning that there is a separate task approximating the demands for each of the sources and the destinations. Then, all of these tasks are trained together in a single machine learning model. **LSTM** [8] standing for Long Short-Term Memory network and is a type of **RNN** (Recurrent Neural Network) which is good at capturing the sequential dependencies between the inputs (i.e. the inputs coming from the different time steps). **MDN** [3] standing for Mixture Density Network, which was already mentioned and is a type of neural network which can conditionally approximate mixture models (in this paper's case **GMM**s).

Preliminaries

In this section, notations are defined to present in mathematical terms the problem that is solved. Additionally, a brief introduction and definitions have been provided for the different machine learning components that were used to build the model.

Regions: $L = \{l_1, l_2, .., l_n\}$ is the set of non-overlapping regions. For example, l_1 could be the Financial District in Manhattan, New York.

Time intervals: $T = \{t_1, t_2, .., t_m\}$ is the set of evenly partitioned time intervals. For example, t_1 corresponds to the time span between [2018/01/01 00:00] and [2018/01/01 01:00].

Taxi ride: o is a triple $(o.s, o.d, o.t)$ where $o.s$ and $o.d$ are the source and the destination regions of the customer order, and $\{o.s, o.d\} \subseteq L$. The last argument of the triple $o.t$ represents the time interval in which the customer order belongs to and $o.t \in T$.

Demand: the number of customers' orders taken between a source region s and a destination region d for a given time interval t. Mathematically expressed as:

$y^t_{s,d} = |\{\ o : o.t \in T_t \wedge o.s \in L_s \wedge o.d \in L_d\}|$

where $|\cdot|$ denotes the cardinality of the set and t, s, d are particular indexes of the corresponding sets for time intervals (i.e. t is index in T) and for locations (i.e. s and d are indexes in L).

OD-matrix: In each time interval t for any source region s and any destination region d, the total travel demand is denoted as $r_{s,d}$, then, all these demands are put in a matrix $M_t \in \mathbb{N}^{L \times L}$ (since any region can be reached from any region). Moreover, an OD-matrix is a matrix with the demands for all of the source and destination region pairs for a given time interval. For example, in fig. 4, on the left image it can be seen how the southern part of Manhattan has been partitioned into non-overlapping regions. On the right image in fig. 4, the formed OD-matrix of these regions is presented. Note, an OD-matrix represents all the demands between sources and destinations, and it is associated with a particular time interval.

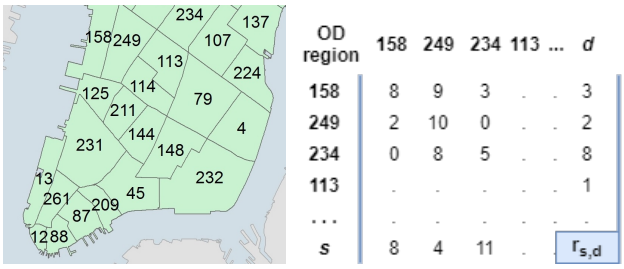

Figure 4: Origin-Destination matrix example

Demand prediction: Is to approximate conditional probability densities of demands for all of the OD-pairs for hour $t+1$, given a sequence of observed OD matrices $\{M_1, M_2, .., M_t\}$ until hour t.

Gaussian Mixture Model

A Mixture model is a probabilistic distribution which is composed of multiple distributions (referred to as components) and corresponding weights Φ which are per component. Therefore, a Gaussian Mixture model is a distribution which is formed from the weighted combination of more than one Gaussian distributions [12]. A Gaussian distribution by itself is a continuous distribution (i.e. a distribution with infinite number of outcomes) and it is defined with a mean μ and a variance σ^2. In fig. 5, the blue, the orange and the green curves represent Gaussian distributions with different means and variances. The red dashed line represents a Gaussian Mixture Model which is composed from the three Gaussian distributions and the corresponding weights are equally distributed.

Getting a probability from a Gaussian Mixture model with K number of components is defined in eq. 3. Since a Gaussian Mixture model is a linear combination of multiple Gaussian distributions (i.e. each component gets its impact),

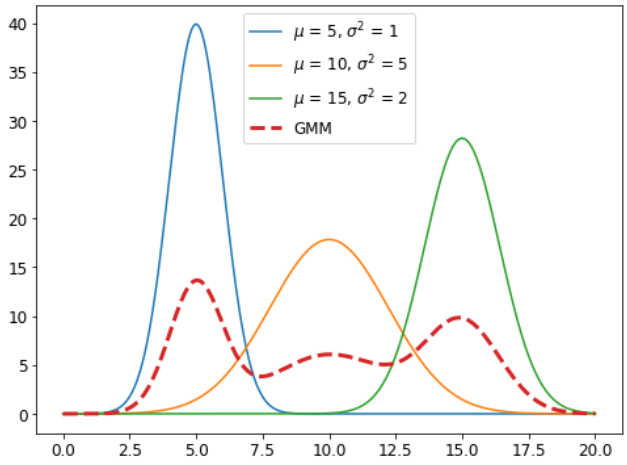

Figure 5: Gaussian Mixture Model example

then, there is a constraint that the coefficient weights Φ must sum to 1. In mathematical terms:

$$\sum_{i=1}^{K} \phi_i = 1. \tag{2}$$

Following, is eq. 3 which defines a GMM density.

$$P(x) = \sum_{i=1}^{K} \phi_i N\left(x|\mu_i, \sigma_i^2\right) \tag{3}$$

Where N is a Gaussian Density Function and it is expanded as the following equation:

$$N\left(x|\mu_i, \sigma_i^2\right) = \frac{1}{\sqrt{2\pi\sigma_i^2}} exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right) \tag{4}$$

Where exp is an exponential function and it is defined as the following:

$$exp(x) = e^x \tag{5}$$

Mixture Density Network

Mixture Density Network is proposed by C.M. Bishop [3] and is a type of Artificial Neural Network which can approximate conditional probability density functions (i.e. including all the different kinds of mixture models). In this project and as mentioned earlier, it is used to approximate GMMs of demands between origin regions and destination regions.

The MDN layer is a layer which is placed as an output layer of a machine learning model. Unlike the single value prediction output layers, which in most of the cases use **MSE** (Mean Squared Error) to propagate error, the MDN uses the **NLL** (Negative Log Likelihood) error function.

Machine learning data is split into input features X and output labels Y. In fig. 6 a simple MDN architecture is presented. The model in the figure takes X as an input and it

approximates Y via a 2 component GMM (the yellow circles), in mathematical terms: $P(Y|X)$.

When X is passed to the Neural Network in fig. 6, then, the Neural Network (the box in grey) outputs a vector Z (the circles in red). Afterwards, vector Z is transformed into a GMM with the help of softmax, linear and exponential activation functions. Note, that $|Z| = (c + 2) \times k$, where c is the number of predicted features and k is the number of GMM components.

In order for the neural network to set the correct values of the Z vector, a NLL of Y is taken and the NLL cost is propagated back to the corresponding Z neurons with corresponding derivative functions. To read more about how back-propagation works in a MDN, it can be found in C.M. Bishop's paper [3].

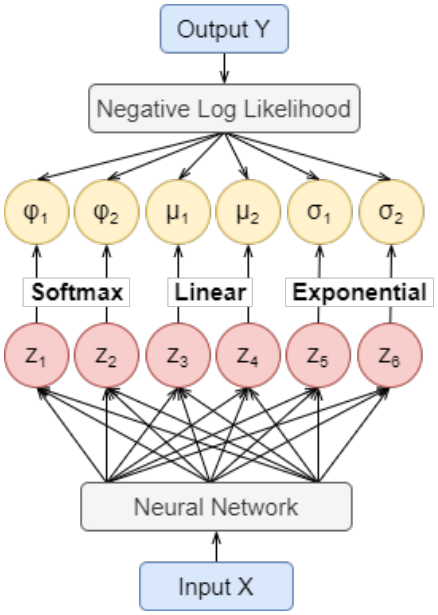

Figure 6: Mixture Density Network

There is a motivation for choosing the softmax, linear and exponential activation functions. There are two constraints that need to be met when producing GMMs. Firstly, as mentioned in the GMM section, the mixing coefficients Φ need to sum to 1 (a constraint described in eq. 2). C.M. Bishop proposes this to be solved with the help of the softmax activation function [3] and it is defined as:

$$\phi_i = \frac{exp(z_i^{\phi})}{\sum_{j=1}^{k} exp(z_j^{\phi})} \tag{6}$$

Where exp is the same as in eq. 5 and z_i^{ϕ} corresponds to a particular mixing coefficient neuron of vector Z. For instance, in fig. 6 the mixing coefficient ϕ_1 is mapped from z_1^{ϕ} which is the mixing coefficient neuron of vector Z for the first GMM component (in the figure it is the red z_1).

The second constraint which is needed in order to produce a GMM is that the variances σ^2 (in fig. 6 and in the following equations denoted just as σ) need to be positive. In order to achieve this, C.M. Bishop [3] proposes the use of exponential activation function:

$$\sigma_i = exp(z_i^{\sigma}) \tag{7}$$

Where exp function is the same as in eq. 5 and z_i^{σ} corresponds to a particular variance neuron of vector Z.

Finally, since the two constraints for producing GMMs have been met, then, the only missing part to transform vector Z into a GMM is the means μ of the components. Since, the means can take negative or positive scalar values, then, their values can be fetched directly from vector Z:

$$\mu_i = z_i^{\mu} \tag{8}$$

Where z_i^{μ} corresponds to a particular mean neuron of vector Z.

The error function which is used to propagate the error when training is a standard NLL error function and is defined as:

$$NLL = -\sum_{q} ln(\sum_{i=1}^{K} \phi_i(x^q) \times N(y^q|\mu_i(x^q), \sigma_i(x^q))) \tag{9}$$

Where q is a training sample composed of x and y, ln is a natural logarithmic function, and N is a Gaussian density function and it is the same as in eq. 4.

Multitask learning

Multitask learning is a machine learning technique proposed by R.Caruana [5] and suggests that different prediction tasks can be trained together to improve the overall accuracy. In fig. 7 it can be seen, there are two separate models which approximate GMMs of demands for particular OD-pairs (i.e. the pairs are 158-234 and 113-234). In contrast, in fig. 8 the tasks of approximating GMMs are combined via a shared layer (the neurons in yellow). Furthermore, because of the shared layer, information can be passed between the different tasks and what is learned from one task can help the others.

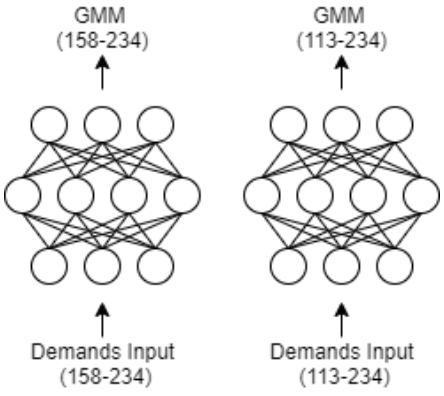

Figure 7: Separate task Learning

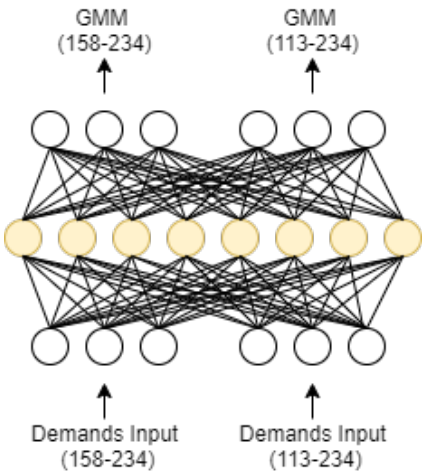

Figure 8: Multitask Learning

Dropout

Dropout [14] is a simple regularization technique which prevents a neural network from over-fitting. Since, it has been proposed, it has gained quite a lot of popularity and has been implemented in most of the machine learning frameworks. Briefly, when a machine learning model is learning, in many cases for predictions the model tends to rely too much on particular weights and neurons, which eventually leads to over-fitting. What dropout does to remove these dependencies on the particular neurons and weights is to randomly "drop them out" (meaning to ignore them or zero them out). According to [11], "if there is a hidden layer h, then, the activations in h when training will be zeroed out in the following way: $h = h \circ m$, where $\circ$ is element-wise multiplication, and m is a binary mask vector" and $|h| = |m|$. "Each value in the m vector is drawn independently from a $Bernoulli(p)$" where p is the chance of keeping a connection (opposite of dropping it out) and it is treated as a hyper-parameter in the context of a machine learning model. During testing or validation, all of the activations remain, but then h is scaled like: $h = p \times h$. Note, in the different machine learning frameworks dropout has been implemented in different ways, yet they are equivalent.

Long Short-Term Memory

LSTM (Long Short-Term Memory Networks) [8] presented in fig. 9 is a state-of-the-art **RNN** (Recurrent Neural Network) which does not suffer from the vanishing/exploding gradient problem. RNN is a type of neural network which works with sequential data, and it is widely used in the field of **NLP** (Natural Language Processing).

Unlike the simple RNNs which transfer only hidden state (Short-term memory) between the sequential inputs, the LSTMs transfer cell-state as well (Long-term memory). Interestingly, LSTMs have gates which control the inflow of information, moreover, these gates can learn what data from the input sequence is important and what data should be forgotten.

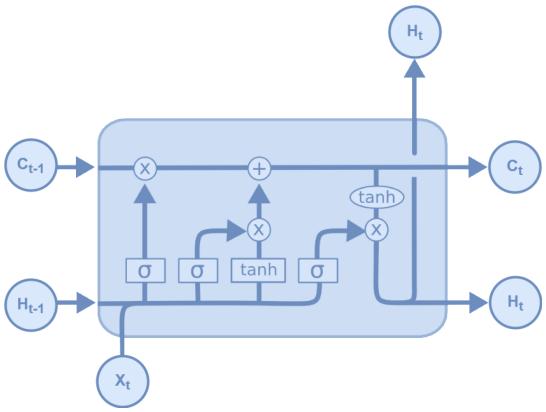

Figure 9: LSTM cell

The input gate i_t, decides whether to write to the cell-state c_t by squashing the signal of the previous hidden state h_{t-1} and the current input x_t through a Sigmoid function, later on, combining it with the output of the input modulation gate g_t via multiplication.

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{10}$$

The forget gate f_t uses the current input x_t and the previous hidden-state h_{t-1} to tell the cell-state c_t what information should be erased. The output of the forget gate is in the range of $[0, 1]$ (i.e. because of the Sigmoid activation), therefore, when it is multiplied with the cell-state c_t - information can be ignored (i.e. values will be zeroed out).

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{11}$$

The output gate o_t takes care of what information is forwarded as the next hidden state h_t. Furthermore, the cell-state c_t, the input x_t and the previous hidden state h_{t-1} have an impact on what the hidden state h_t will be. Just like in the input gate i_t and the forget gate f_t, the output gate relies on the multiplication operation to ignore values. Furthermore, this gate decides how much of the cell state c_t should be revealed.

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{12}$$

The input modulation gate g_t modulates the input x_t and the previous hidden state h_{t-1} via a Tanh activation function, which is in the range of [-1,1]. Interestingly, because of the function's range (i.e. it includes both positive and negative values), information can be added or removed before it is combined with the input gate i_t. Moreover, this gate decides how much it should be written to the cell state c_t

$$\mathbf{g}_t = tanh\left(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g\right) \tag{13}$$

The equation below shows how the cell-state c_t is created and modified through the sequential inputs with the help of the forget gate f_t, the previous cell-state c_{t-1}, the input gate i_t, and the input modulation gate g_t.

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{g}_t \tag{14}$$

The equation below shows how the new hidden state h_t is created via the output gate o_t and the current cell-state c_t.

$$\mathbf{h}_t = \mathbf{o}_t \circ tanh\left(\mathbf{c}_t\right) \tag{15}$$

Note, in the above described gates equations, the Ws are learnable parameters (i.e. matrices containing weights).

Proposed model

The proposed model in fig. 10 is a *many-to-one* sequence model. The model takes as an input a sequence of observed OD matrices $\{M_{t-n}, .., M_{t-1}, M_t\}$ until hour t, and it outputs a GMM for each OD-pair for hour $t+1$ (the next hour). The length of the OD matrices sequence is 14 (i.e. the model does a prediction for the next hour based on data from the previous 14 hours). In details, the forward pass to the model happens in the following order:

1. The OD matrices M are passed to an LSTM layer (in the picture it is marked as the big A), and the LSTM layer is composed of 512 LSTM cells.
2. A dropout is applied to the LSTM layer's output via a Dropout layer. The dropout frequency or rate is set to 0.3.
3. The dropout is directly connected with multiple MDN layers. Each of the MDN layers has the task to approximate the demand for a particular OD-pair. The number of the MDN layers is the same as the number of the OD-pairs. For example, MDN *S-D* is an MDN which approximates the demand between a source region S and a destination region D.
4. Finally, the MDN layer of each OD-pair outputs 5 GMM components, which can then be mapped to an actual GMM.

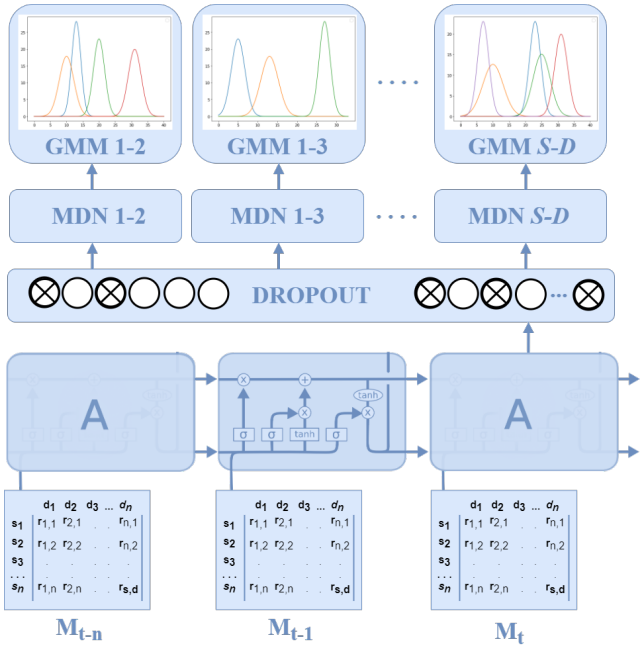

Figure 10: Proposed model

In the proposed model, the Multitask learning is achieved by sharing the LSTM and the Dropout layers. They are the layers that connect all of the different OD pair demand approximation tasks.

Evaluation

In this section, the proposed model is bench-marked against other machine learning models and evaluated according to relevant MDN evaluation metrics. Finally, an ablation study has been done which considers the proposed model without the MDN layers.

Evaluation Metrics

There are 3 evaluation metrics that have been used to evaluate the proposed model. The first one is average NLL and it was successfully used in [1]. The other two metrics **MAE** (Mean Absolute Error) and **RMSE** (Root Mean Squared Error) are standard well-known metrics for evaluating machine learning models. Nevertheless, if MAE and RMSE are to be calculated, a method for making a prediction from a GMM is required. The simplest method to make a prediction from a MDN is to calculate the expected value of the produced mixture model [4], which for a GMM is defined as:

$$E(\hat{y}) = \sum_{i=1}^{K} \phi_i \times \mu_i \tag{16}$$

Where $\hat{y}$ is a predicted GMM vector containing GMM components, K is the number of components, ϕ_i is the coefficient weight of a component, and μ_i is the mean of a component.

Average NLL Likelihood tells us how likely is for a true observed value y to have come from a distribution produced by an input x. Thus, NLL is the opposite of likelihood, and for computational simplicity it is logarithmic-scaled. The lower the average NLL is, the better the proposed model is. Average NLL is calculated in the following way:

$$Avg.NLL = \frac{NLL(q)}{n} \tag{17}$$

Where NLL is defined in eq. 9 and q is the training or the validation samples set, and $n = |q|$.

MAE MAE is a metric which is computed based on the average absolute difference between predicted values and true values. Since it is absolute, MAE is a metric which does not show the direction of the error (differences). Another, in MAE all the errors get equally weight and MAE is defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} abs(E(\hat{y_i}) - y_i) \tag{18}$$

Where abs is a function which returns the absolute value, and $E(\hat{y_i})$ is the expected value of the predicted GMM, and y_i is the true value.

RMSE RMSE is similar to MAE, nevertheless, in the way the RMSE is calculated (i.e. the errors are first squared then a root of the average sum is taken), it puts more weight on the outliers and the errors that are further away from the mean. Interestingly, MAE compared to RMSE is very optimistic, also, MAE can never be greater RMSE. The equation below shows how RMSE is calculated:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(E(\hat{y}_i) - y_i)^2} \tag{19}$$

Benchmark Models

The proposed model was bench-marked against another 2 machine learning models. One of which is a **MLP-MDN** (MLP standing for Multi Layer Perceptron, which is a simple feed-forward network), the other one is LSTM-MDN.

MLP-MDN MLP-MDN is a single task model which approximates the demands for a particular OD-pair. Furthermore, it takes as an input a demands vector of the previous 14 hours, and it approximates a GMM for the next hour. The model is composed of a single hidden feed-forward layer [15], dropout layer and a MDN layer. In order to approximate the demands for the whole OD-matrix, there is an MLP model for each of the OD-pairs.

LSTM-MDN The LSTM-MDN model is much similar to the MLP-MDN model, with the only difference that the single hidden layer is an LSTM layer instead of a simple feed-forward layer.

Evaluation results

In the table below, there are the evaluation results from the validation data-set. These results were taken from the last training epoch. The lower the Avg. NLL, MAE, RMSE, the better the model. The "MT-LSTM-MDN" is the proposed model in this paper, and it is the one that performs the best. One thing that stands out in the results, is that the NLL of the LSTM-MDN and the MT-LSTM-MDN are almost the same, nevertheless, their MAEs and RMSEs differ by 10%. Therefore, the Avg. NLL as a metric is not enough to judge on how accurate an MDN model is.

Model	**Avg. NLL**	**MAE**	**RMSE**
MLP-MDN	2,354	3,541	4,855
LSTM-MDN	2,068	3,454	4,766
MT-LSTM-MDN	2,067	3,106	4,317

Another thing these results prove is that the Multitask learning does its job and because all of the OD demands approximation tasks are trained together, the accuracy is improved by 10% compared to the LSTM-MDN model.

Ablation study

In the ablation study, the MDN part from the MT-LSTM-MDN, LSTM-MDN and MLP-MDN models has been removed. Therefore, now the proposed model and the benchmark models are no longer MDN models, but are a single value of demand prediction models. Likewise, it can be concluded whether the MDN models will have worse accuracy compared to the single value prediction models. Additionally, since there are no distributions to be approximated in the single value prediction models, the Average NLL evaluation metric is no longer relevant and it will not be used. Finally, the error function for the single value prediction models needs be changed as well. In this paper's study, the NLL error function was replaced with the **MSE** (Mean Squared Error) function - defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{20}$$

Where $\hat{y}_i$ is a single value of predicted demand.

The table below presents the results from the ablation study. As it can be seen, comparing the pairs of LSTM-MDN model vs the LSTM model and the MLP-MDN model vs the MLP model, their MAEs and RMSEs are almost equal, meaning, that the LSTM-MDN and MLP-MDN models are optimal (i.e. no accuracy is lost because of the added MDN layers). Nevertheless, when comparing the MT-LSTM model with the MT-LSTM-MDN model, there is a 10 % gap in the evaluation metrics. The MT-LSTM model is performing better than the MT-LSTM-MDN model. The reason for these accuracy differences between the two Multitask learning models is not known. One could argue that it is because of the introduced complexity of the multiple MDN layers in the MT-LSTM-MDN model. Another, could be that the MT-LSTM-MDN has the potential to be as accurate as MT-LSTM model when doing single value predictions, but, the MT-LSTM-MDN was not trained properly.

Model	**MAE**	**RMSE**
MLP	3,524	4,808
LSTM	3,454	4,751
MT-LSTM	2,952	4,104

The conclusion from the ablation study is when a MDN model is trained, always try to remove the MDN part of it and turn it into a single value prediction model. Likewise, when comparing the two it can be assured that accuracy is not lost because of complexity.

Experiment

In this section of the report, it is described what data was used for the machine learning experiment, what was done to cleanse the data, finally, what issues were faced when the proposed model was trained.

Data cleansing and data selection

The data-set that was used, and as mentioned in the Introduction, is the 2018 NYC Yellow Taxi data-set. Interestingly, in it there are 112 million taxi trips recorded for all of the New York boroughs. 100 thousands of these trips have a customer drop-off time happening before customer pick-up time (i.e. faulty data). There are also samples, with a pick-up time happening in year 2084 and trips which took more than 24 hours. All of that data was removed for the training and validation data-sets. Additionally, it was observed

that in month March there are twice more trips compared to the other months, thus, the data from March was removed as well. Finally, the cleansed taxi orders were grouped by the hour of the customer pick-up time and this was the very final data-set that was used for the development of the model.

As mentioned in the Introduction, the proposed model approximates demands for a small part of Manhattan, thus, it is not the case that all of the Manhattan regions have been used for the training/validation process. The total number of regions in Manhattan is 60, the proposed model uses only 6 of these. In detail, the region ids that were used are: $L = \{161, 162, 164, 170, 229, 233\}$, hence, the total number of OD pairs is $6 \times 6 = 36$. Eventually, the train/validation data-set split is 75% for the training data and both of the sets have not been shuffled.

Machine Learning setup

The machine learning framework that was used, to develop the proposed model and train it, is Tensorflow Keras version 2.0. Additionally, in order to be able to create MDN models, it is required that the package *tensorflow-probability* is installed as well.

For the proposed model and the benchmark models, the number of GMM components that they output per OD pair is 5. Also, the number of previous hours that is inputted into the models per OD pair is 14. The optimizer that was used to train the models is Adam [6] with its default settings. The batch size was set to 2000 and the number of training epochs for the benchmark models is 3000 and for the proposed model is 1500.

Findings during the experiment

In the beginning, when the Multitask learning model was developed, the idea for it was not be an LSTM model. It is well known that LSTMs can achieve tremendous results when predicting time-series, nevertheless, a lot of multiplication is involved which slows down the training process. Another, machine learning technique for predicting time series is to use 1D (1-Dimensional) Convolutional neural networks. In details, these **Conv1D** networks happen to work very well with the **ReLU** (Rectified Linear Unit) activation function. The ReLU function is an activation function which does not squash the signal that is being passed to it, furthermore, it is defined as:

$$Relu(x) = max(0, x) \tag{21}$$

Where max returns the bigger value of 0 or x. Hance, $ReLU$ can return either zero, either a positive value. Interestingly, what was discovered during the experiment was that ReLU can be used successfully to build single prediction models. Nevertheless, it cannot be used to build MDN models because on the very first training epochs the training loss would become **NaN** (Not a Number). This issue appears, when the calculated error underflows or overflows due to the limitations of the computer's hardware (i.e. it cannot represent too big or too small numbers). According to A.Brando's paper (*Mixture Density Networks for distribution and uncertainty estimation*) [4], there are 3 cases which can lead to a NaN loss when a MDN model is trained:

1. "$log(\epsilon), \epsilon \sim 0$ - There is a logarithm of a value that is very close to zero".
2. "$\frac{1}{\epsilon}, \epsilon \sim 0$ - There is a fraction with a denominator that is very close to zero."
3. "$e^x, x > 1e^4$ - There is an exponential of a 4 digit value."

Considering the situation where a hidden ReLU layer (placed before a MDN layer) returns a vector containing zeros, then, it can happen that some of the $\alpha(x)$ or $\sigma(x)$ are zeros, which will eventually lead to the 1st or the 2nd case of the above mentioned cases from A.Brando's paper [4]. On the other hand, as mentioned earlier ReLU is not the kind of function which squashes the signal, moreover, it is a function which returns the value itself as long as it is a positive one. Therefore, if a value returned from a hidden layer is bigger than 1000 (i.e. even 800), when passed through the exp function so the $\sigma(x)$ can be produced, then, the variances of the distribution will explode and the training loss will overflow, which is the situation described in case 3. There are tricks and workarounds described in A.Brando's paper [4] which can be used so the NaN loss does not appear. In this paper's case, it was decided that Conv1D with ReLU layer will be replaced with a LSTM layer and since it has been done, the NaN loss has not been observed.

When the LSTM-MDN model was trained, initially, there was no need for regularization or for an addition of a dropout layer. Nevertheless, in the image in fig. 11 the training and the validation loss is presented when training the model without a dropout layer. The y-axis of the chart is the NLL loss, and the x-axis is the number of training epochs. As it can be seen from the chart, the loss is fluctuating with a 0.5 in NLL and it is hardly converging towards a global minimum. Changing the hyper-parameters of the model such as the number of GMM components, the number of LSTM cells and the learning rate did not help much to make the training curve smoother. Eventually, after a lot of experimentation, a dropout layer was added right before the MDN layer and the curve became smoother and converging as it is in fig. 12.

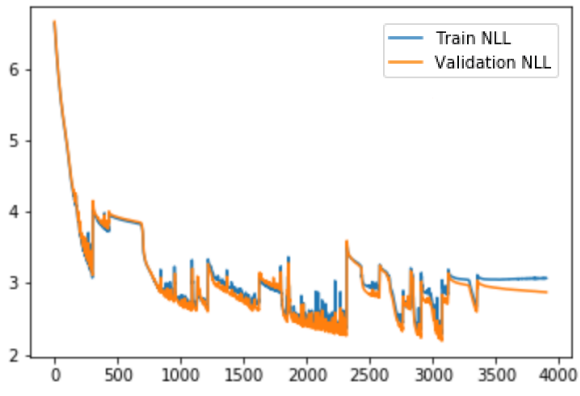

Figure 11: Training and validation loss without a dropout layer

Therefore, for the data that was used (i.e. the taxis data), it is essential to have dropout layer so that there is a smooth

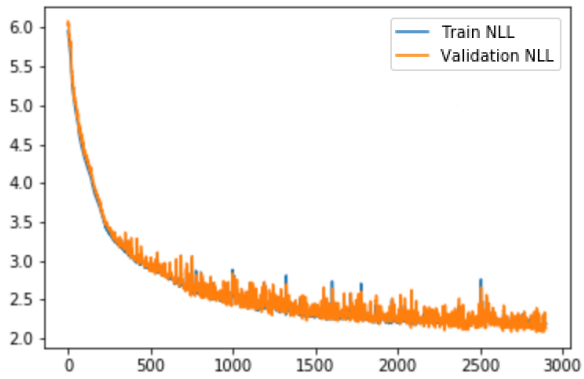

Figure 12: Training and validation loss with a dropout layer

training process which ensures that the global minimum will be reached and that the training process will not get stuck into a local minimum as it is in fig. 11.

Conclusion

It is really fascinating and intriguing how MDNs can be used to approximate distributions. Additionally, how instead of a single value prediction, there is a collection of predictions with a probability for each of them. Unfortunately, MDNs are quite sensitive and when used with real-world data can lead to a NaN loss training problems. Fortunately, it can be fixed with gradient clipping, log-sum-exp tricks and other techniques which can be found in papers like the A.Brando's paper [4]. Another, in this paper's case, it seems like MDNs take a lot of time to be trained and to converge, which is partially due to the use of a dropout layer is. In details, because a dropout layer is randomly ignoring part of the signal it is being passed, then, it is a requirement that the machine learning model is trained for longer.

The proposed model in this paper, works for a small neighbourhood of regions, in real-world scenarios it can be implemented for cities where driver restriction policies are in place. Even though, the proposed model losses some of its accuracy because of the addition of the MDN layers, it is still a model which can be used in the real-world. Also, it is a model which outputs multiple GMMs (i.e. there aren't many of these). The future work of this project would be to figure out why exactly the proposed model looses accuracy against a model without MDN layers. Another, since the Multitask learning models are not very scalable, it would be interesting to see how much this model can be scaled in terms of how many OD-pairs it can approximate. Moreover, will it be able to approximate all of the OD-pairs in Manhattan and how accurate will it be?

References

[1] R. Alligier. "Predictive Joint Distribution of the Mass and Speed Profile to Improve Aircraft Climb Prediction". In: (2020), pp. 1–10.

[2] Javier Alonso-Mora et al. "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment". In: *Proceedings of the National Academy of Sciences* 114.3 (2017), pp. 462–467.

[3] Christopher M Bishop. "Mixture density networks". In: (1994).

[4] Axel Brando Guillaumes. "Mixture density networks for distribution and uncertainty estimation". MA thesis. Universitat Politècnica de Catalunya, 2017.

[5] Rich Caruana. "Multitask learning". In: *Machine learning* 28.1 (1997), pp. 41–75.

[6] Trishul Chilimbi et al. "Project adam: Building an efficient and scalable deep learning training system". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 2014, pp. 571–582.

[7] Martin Felder, Anton Kaifel, and Alex Graves. "Wind power prediction using mixture density recurrent neural networks". In: (2010).

[8] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[9] Lingbo Liu et al. "Contextualized Spatial–Temporal Network for Taxi Origin-Destination Demand Prediction". In: *IEEE Transactions on Intelligent Transportation Systems* 20.10 (2019), pp. 3875–3887.

[10] Luis Moreira-Matias et al. "A predictive model for the passenger demand on a taxi network". In: (2012), pp. 1014–1019.

[11] Vu Pham et al. "Dropout improves recurrent neural networks for handwriting recognition". In: *2014 14th International Conference on Frontiers in Handwriting Recognition*. IEEE. 2014, pp. 285–290.

[12] Douglas A Reynolds. "Gaussian Mixture Models." In: *Encyclopedia of biometrics* 741 (2009).

[13] Christian Schittenkopf, Georg Dorffner, and Engelbert J Dockner. "Volatility prediction with mixture density networks". In: (1998), pp. 929–934.

[14] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[15] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. "Introduction to multi-layer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[16] Yuandong Wang et al. "Origin-destination matrix prediction via graph convolution: a new perspective of passenger demand modeling". In: (2019), pp. 1227–1235.

[17] Huaxiu Yao et al. "Deep multi-view spatial-temporal network for taxi demand prediction". In: (2018).

[18] Heiga Zen and Andrew Senior. "Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis". In: (2014), pp. 3844–3848.