Hyperparameter Optimization for Neural Network based Taxi Demand Prediction

Nicola Schwemmle[1]
Tai-Yu Ma[2]

Abstract: *Being able to accurately predict future taxi demand can beneficial not only for taxi companies but also for passengers and the environment as an intelligent taxi planning system can reduce waiting and idle driving times. This work proposes an extended Long Short-Term Memory (LSTM) neural network structure for predicting future taxi demand. Experiments are performed on taxi data from New York City. The model's hyperparameters are tuned using a very simple selection method based on predictions for only one location at a time. More complex algorithms, hyperopt and BOHB, are implemented to tune the model's hyperparameters in a more structured and comprehensive way which reduces the prediction error by 2.2% compared to the simple selection method. The results suggest that there are factors that limit the performance gains of popular hyperparameter optimization techniques but also that a relatively simple model can yield useful predictions and outperform several naive benchmark methods.*

Keywords*:* Taxi demand, Time series, Prediction, LSTM, Hyperparameter Optimization

1. Introduction

Before the rise of mobile internet and location systems like GPS, the process of finding a nearby taxi would most often involve waiting near busy streets or go to nearby, known taxi stands. Vice versa, taxi drivers could only wait at given locations or roam the streets looking for passengers. Nowadays, this search is often centralized (Safikhani et al., 2018) and intelligent systems try to match the given demand (i.e., passengers) with the given supply (i.e., taxis). Often, this task is done for the present, meaning currently empty taxis are sent to pick up currently waiting passengers. Being able to accurately predict future demand helps the allocation process because drivers can go to areas that are close and are expected to be in high demand soon instead of idly roaming areas where there are only a few potential passengers (Ma et al., 2019). Reliable forecasting methods can therefore reduce energy waste, waiting times and maybe even lead to fewer traffic jams in highly congested city centres (see for example Chu et al., 2018, Safikhani et al., 2018). These gains in efficiency also save the taxi deploying company and/or each driver money, which makes finding accurate predictive models an even more important field of research. Results from predicting taxi demand patterns can also yield insights into more general traffic patterns (e.g., public transport usage patterns, traffic jam forecasts). Any system used for this purpose should be small and simple enough to be easily employed in practice, which is why this work focusses on finding a relatively small network that can be trained and deployed cheaply but still yields good results.
For this work, various techniques to predict trip demand patterns, that have been employed in the past, are presented and explored. A basic architecture for a neural

[1] University of Luxembourg
[2] Luxembourg Institute of Socio-Economic Research

network based model to predict future demand is developed, and different methods of tuning this model are explored and evaluated using New York City taxi data[3]. The goal is not to come up with a revolutionary new network architecture but to find out how much the given basic architecture can be improved by tuning its hyperparameters. To this end, widely used deep learning architectures for time series prediction (i.e., "Long short-term memory" or LSTM networks, see Hochreiter and Schmidhuber, 1997) are combined with state-of-the-art methods for finding the best model setup. The approach optimizes the set of parameters that cannot be learned by a neural network. This so called hyperparameter tuning is often computationally expensive and the space that one searches over is extremely large (in theory when there are real valued hyperparameters, there are infinitely many possible parameter configurations), so developing efficient algorithms has been getting more and more attention over the last years to account for growing networks with ever more hyperparameters. To find out what impact a finely tuned model has over one that uses a default or random setup, Section 4.1 presents a model architecture that is then tuned following very simple method that only considers a few possible hyperparameters. In Section 4.2 this model is then tuned using more advanced techniques (mainly a bayesian algorithm called hyperopt, see Bergstra et al., 2013) to determine how much performance can be improved by a good set of hyperparameters.

2. Literature review

In this section, the state of the art of deep learning based methods for taxi demand prediction is outlined and an overview of existing approaches to hyperparameter optimization is provided.

2.1. Deep learning approaches to taxi demand prediction tasks

In recent years the focus of taxi demand prediction research has shifted from "classic" time series methods (e.g., Arima and VAR) to deep learning based methods, mostly using neural networks built of LSTM cells (Hochreiter and Schmidhuber, 1997). LSTM cells' recurrent parts consist of a memory cell that is updated in every time step and can theoretically use information on the first observation to adjust the output for the last input (see Yu et al., 2019, Section 2.2). Regular Neural Networks would have much worse performance for tasks with long-term dependencies in the data, because they only look at the current input. There exist many different types of LSTM cells, but they all share the basic feature of a memory cell that is updated in specific ways after each time step.

Zhao et al. (2017) use LSTM networks with an additional input dimension ("2DLSTM") to predict traffic volumes in Beijing. The two dimensions of the network correspond to the spatial and temporal dimensions of the data (Yu et al., 2019). They predict volumes for different points and forecast horizons. Although their model is better than the considered alternatives, it does not consistently outperform them (Zhao et al., 2017, Table 1). In the form that is proposed by the authors, the model does not offer a possibility to use additional metadata as inputs, but it could be adjusted for this purpose. The used data is not based on GPS locations of vehicles but on recordings from observation stations (data collection via cameras, induction coils and velocity radars, see Zhao et al., 2017, Section 4.1). As this work is focused on general traffic flow and not on the specific domain of taxis or MoD, the network architecture might not be applied to the New York City taxi data easily.

[3] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Ke et al. (2017) propose a fusion convolutional LSTM network (FCL-Net) based on the convolutional LSTM (ConvLSTM) network of Shi et al. (2015). The underlying ConvLSTM was not designed for traffic or trip demand forecasting, but for short-term forecasting of rain intensity for precipitation data collected in Hong Kong (Shi et al., 2015,). The general architecture however is very well suited for the trip demand prediction problem, as it is designed to combine the power of CNNs and LSTM cells for extracting spatial and temporal information, respectively. The "fusion" part of the FCL-Net is aimed at the inclusion of metadata that has no spatial dimension (the authors use weather data, dummies for the weekday and time of the day, but there is no restriction on the kind of metadata used). Clearly, the fusion is done by simply adding the weighted outputs of all component networks to obtain a final prediction. The authors additionally introduce a spatial aggregated random forest for feature selection (Ke et al., 2017) that selects the most informative input features. By leaving out features, that are found by the random forest to be not very informative, the authors can reduce training time, while still obtaining precise predictions. Although the model is used for data made available by a Chinese on-demand ride service provider, the data structure is very similar to the New York City taxi data and the model proves to be very powerful in comparison with other approaches (Ke et al., 2017).
Instead of using a network that directly predicts real valued demand volumes (e.g. in taxis/hour), Xu et al. (2018) combine LSTM cells with a Mixture Density Network (MDN) (Bishop, 1994) that outputs parameters of multivariate Gaussian Kernels for taxi demand prediction. To obtain real valued predictions, random samples are drawn from the Kernels, parametrized by the MDN. Spatial correlations in the most recent time step are exploited by conditioning the samples on the taxi volume for all other areas in the same time step. This approach is very different from combining CNNs and LSTM cells as it provides a very elegant way of extracting spatial dependencies without explicitly inputting locations. The data used includes information about "yellow cabs" (mostly employed in Manhattan) and "green cabs" (operate mostly in the suburbs of NYC).

2.2. Hyperparameter optimization methods

Hyperparameter optimization refers to procedures which try to find optimal values for model parameters that cannot be learned, trained or inferred from the data at hand. Tuning models based on intuition or experience can easily lead to ignoring possible performance improvements because some architecture is unusual or was just never tried. As more and more frameworks exist that offer at least some degree of deep learning without requiring expert or coding knowledge (see for example Google Cloud AutoML[4]), there is more demand for clearly structured methods that can handle a wide variety of data and return good models without human input. Another factor is that deep neural networks can be extremely expensive to train even once so any trial and error approach to tuning such a model's plethora of hyperparameters is infeasible. Feurer and Hutter (2019) give a good overview over the underlying problem definition and reasons why hyperparameter optimization is a difficult and popular topic.
Hyperparameter optimization techniques can be model-free (e.g., random or grid search (Feurer and Hutter, 2019)). Most popular methods, however, estimate the distribution of hyperparameters based on a density estimation model, draw promising suggestions from this distribution and then update it based on the resulting performance metric. In particular, hyperopt library (Bergstra et al., 2013) offers a popular algorithm that can, in theory, be applied to any optimization problem, but was

[4] https://cloud.google.com/automl

developed for hyperparameter optimization. Hyperopt mainly builds on a Tree Parzen Estimator (TPE) (Bergstra et al., 2011) that returns configurations with the highest expected improvement (EI) based on density-based estimates inferred from past evaluations. Komer et al. (2019) show how the hyperopt algorithm can be combined with other libraries to build integrated solutions that return optimized architectures more or less automatically. One strong reason for using hyperopt is the definition of search spaces that allows conditional structures, meaning that space can be defined in a way that only a subset of the parameters is considered at a time when having to choose between two or more options (e.g., layer type or optimizer) that themselves have parameters that need to be tuned. So only after an option is determined to be the most likely to improve the objective, the TPE considers the parameters conditional on this option and all the other parameters that are rendered without any effect for this specific setup are ignored. Consider for example a guided search that is set up to determine whether to use optimizer 1 or 2 and tune these at the same time. Both optimizers have an option a or b but, clearly, changing this option for optimizer 2 does not have any effect on the outcome when optimizer 1 is used. It could, however, happen that by chance the outcome does improve although only irrelevant parameters were changed and the algorithm then further explores the relationship between these unrelated options, therefore wasting resources and making good results less likely. This scenario cannot happen when implementing a well-designed conditional search space.
Another much simpler approach to ensure an efficient use of ressources is so called "bandit"-based approaches (Feurer and Hutter, 2019) that distribute a given budget (e.g., computation time, memory, iterations, available data etc.) between the candidates and then apply some early stopping rule to candidates that are not expected to perform well. The remaining candidates can then be explored in more depth using the additional resources freed by the terminated trials and the early stopping rule is applied again and so on. The most promising approach in this area is BOHB (combining **B**ayesian **O**ptimization and **H**yper**B**and) (Li et al., 2017; Falkner et al., 2018). This approach combines bandit methods with Bayesian optimization to achieve quick initial performance gains by replacing random draws with HyperBand and good results by switching to Bayesian Optimization later on (Feurer and Hutter, 2019, p.17).
Both BOHB and hyperopt have been shown to perform well on various tasks. For example, Komer et al. (2019) describe a procedure using hyperopt that outperforms the then best known both automated and manually tuned approaches for classifying Convex Shapes (Larochelle et al., 2007). Falkner et al. (2018) also report performance gains and drastic speedups over other common approaches. For this reason, these two algorithms will be used as the basis for the hyperparameter optimization in our study.

3. Data

The NYC taxi data[5] is openly available since 2009. It contains detailed records (pick up and drop off locations and times, fees, tolls, tax info, …) of every single taxi trip undertaken by any taxi company licensed in New York. For this study only the information related to where and when a ride started and ended is used. Since June 2016, the latitudes and longitudes of the pickup and drop off locations were no longer reported, but only the zones defined by the New York City Taxi and Limousine

[5] See https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page for an overview of all available data. The data used here was, however, retrieved via https://data.cityofnewyork.us/ because of the convenient API offered by Opendata. The data from both sources is identical.

Commission[6] (TLC) are reported. These zones are convenient to handle spatial data so a period from July 1, 2016 to June 30, 2019 is used. Instead of using data for all of New York City, the focus study area is Manhattan i.e. rides with a pick up location in one of the 69 zones shown in Figure 1. As it is not allowed to use cars on Liberty Island, this zone is dropped from the dataset. Only 68 zones are used in this study. The data set contains information on more than 200 million single rides, but this level of detail is not helpful for the task at hand, so the data is summarized to count the number of pick ups per zone per time period of 15-minutes[7]. The summarized dataset contains 105120 observations for 68 zones. The data is then scaled to have a mean of 0 and a standard deviation of 1. Note that to keep the characteristic differences between the different zones this is done per zone.

The model inputs are as follows:

- **Last 8 observations for each zone**: As is usual in time series prediction tasks, the main input is the past observations.
- **Features calculated based on the timestamp**: Two numerical values representing the day and the hour of the day (both evenly spaced between 0 and 1), a sine and cosine of this number for the day and time (to avoid hard cut-offs between 23:45 and 00:00 or Sunday and Monday) and a dummy indicating whether it is a weekend.
- **Weather data**[8]: The weather dataset contains the recordings of a weather station situated in Central Park. This data is only available at a daily scale, so each entry is recycled for the 96 15-minute blocks of every day. The features are maximum and minimum temperature, average windspeed, total precipitation, total snowfall and snow depth.

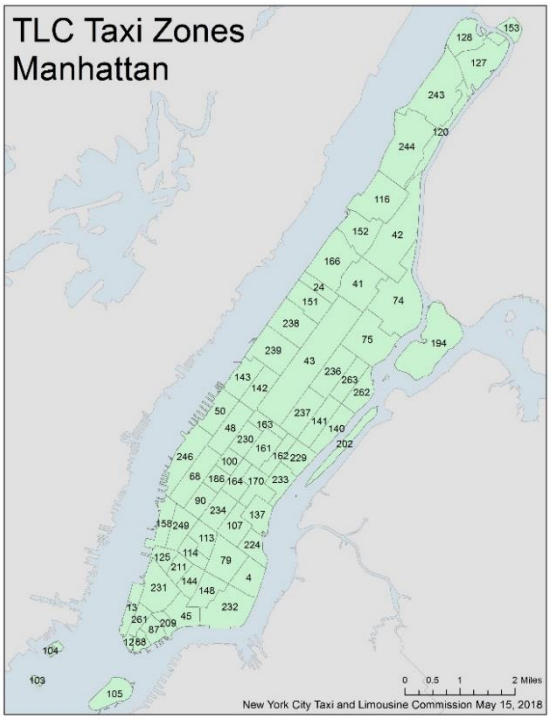
To speed up computation times, the hyperparameter optimization and all other model tuning activities are run on a dataset containing only July 2016, resulting in 2976 (out of the 105120 total) 15-minute blocks, the first 2400 are the training set, another 400 are set to be the validation set and the rest is the test set. Once models are tuned, they are trained on the full dataset excluding a test set to evaluate the performance on (80% training set, 15% validation set, 5% test set).

[6] http://www.nyc.gov/tlc/

[7] 15-minute blocks were chosen to provide a short enough time period to base decisions on where to go as a driver on, while still keeping the set of the data sized easy to handle.

[8] Provided by the National Oceanic and Atmospheric Administration (NOAA) available under https://www.ncdc.noaa.gov/data-access (station name: NY CITY CENTRAL PARK, NY US, station ID: USW00094728), see also: Menne et al. (2012)

Figure 1: TLC Taxi Zones Manhattan



4. Proposed LSTM model architecture and hyperparameter tuning

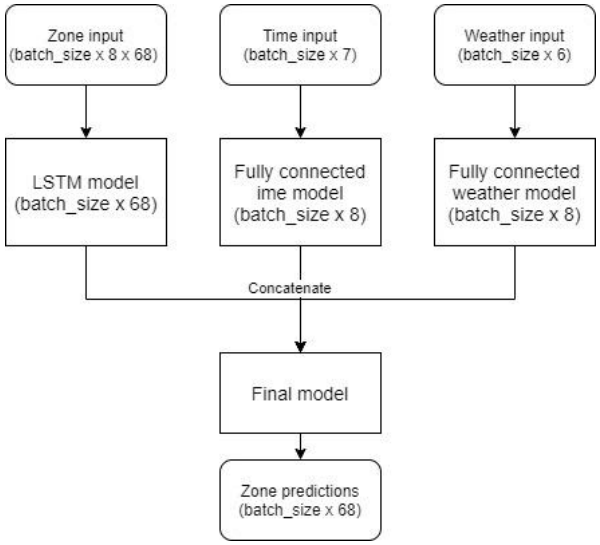
This section introduces an extended LSTM model architecture using a subset of the data to tune its parameters and then compares the rough tuning to an extensive hyperparameter search.

4.1. Proposed extended LSTM model architecture

The main feature of LSTM models is a memory component which allows the model to base outputs on historical data in addition to the current inputs, making it perform very well on time series data. This is achieved by carrying over a so-called cell state vector in every iteration (Yu et al., 2019). As the number of taxi pickups has a clear temporal dependency, an LSTM model is used handle the pick-up. The main idea is that sequences of past pick-up numbers for each zone are fed into one or more LSTM layer(s) and a final fully connected layer to end up with a prediction for the next value for each zone. Here the number of past observations is chosen to be 8 (i.e., 2 hours since every observation is a 15-minute block) so the LSTM portion of the model takes in 68 sequences of length 8 and returns one number per zone (i.e., the prediction for the next time step). As the additional data is structured differently, special attention is needed to efficiently use it. The time features do not need to be handled by a LSTM model because there is no added information of knowing the timestamp of the observation before or after the current observation. The weather data does only change once per day and there is also almost no added information in knowing the weather from earlier days, so this data is also not processed by an LSTM. Instead, the weather and time features are fed into a simple fully connected neural network with one or more hidden layers each. These two networks then return an array of length 8 each that contains “pre-processed” information on the current situation (the length of these arrays and the number of hidden layers in these models are part of the hyperparameter tuning in Section 4.2). The outputs of the three components are then

concatenated and fed into one or two more hidden layer(s), which combine the information of the three models and compute a final prediction for every zone. Figure 2 shows a high-level overview of the model architecture. Numbers in parentheses indicate the size of the output of each model where “batch size” can, in theory, be any number between 1 and the length of the dataset. For this section it is set to be 10 but the batch size is also a hyperparameter that must be tuned. When the model is used for predictions, after training, the batch size will be the same as the size of the input data[9].

Figure 2: Model Overview



The proposed model architecture will be used to determine the benefits of efficient hyperparameter optimization. This requires a reasonable baseline performance. There are no default parameters to set up the proposed architecture, so the approach to determine the rough model structure of Zhang et al. (2017) is used. The main idea of this approach is that instead of using the full dataset and predicting all 68 zones at the same time, a random subset of six zones is selected and several test runs are performed for the most important hyperparameters, i.e., the number of neurons in the LSTM layer, the number of LSTM layers, the number of hidden layers in the time model and the number of layers in the weather model. The values are determined in this order, meaning that the number of neurons for the LSTM layer found in the first experiment is then used as given in all the following experiments and equivalently for the other hyperparameters. Since the six zones used for this experiment are selected at random, one can expect to have some variation between the zones so that the parameters are tuned to fit not only one single type of zone (e.g., highly frequented zones). At the same time, the size of the dataset (1 zone per time step instead of 68) and the number of weights per model is reduced. This procedure therefore offers an advantage over setting an architecture beforehand (or any other design method that does not rely on trying out many configurations but rather guessing or knowing which design will be good) but does not require re-fitting too large models to the data many times. It is by far neither exhaustive nor structured enough to claim that this model setup is the best out of the infinite possible setups, but the method ensures a

[9] Here, this will almost always be the size of the test set. For any real-world application of the model, one would most likely input the current situation in order to get a prediction of the next 15 minutes so there this number would be 1.

reasonable performance at very low cost (i.e., computation time). Given the results of the trials, the model is set up to have one LSTM layer containing 128 neurons and three layers each for the time and weather features.
The tuned model is fit to the full dataset (excluding a test set of length 5247, or roughly 54 days) and the resulting test set MSE is **0.17578**.
The setup presented in this section clearly neglects a lot of other hyperparameters, such as the number of neurons in each of the hidden weather and time layers, choice of optimizers and its features and most importantly the interaction between the hyperparameters. It is possible that one would find different values when running the experiments in a different order. Furthermore, it is not guaranteed that the six zones that were selected at random are a good proxy for the full dataset so the hyperparameters might have been fit to characteristics that do not generalize well to the full dataset. In order to determine, whether the model has more potential, more thorough experiments will be carried out in the next section.

4.2. Hyperparameter optimization

This section introduces and implements two popular hyperparameter optimization techniques that were already mentioned in Section 2.2, namely the hyperopt (Bergstra et al., 2013) and BOHB (Falkner et al., 2018) and have been shown to perform well for various hyperparameter optimization tasks (Feurer and Hutter, 2019; Komer et al., 2019). Both algorithms have a similar Bayesian component but BOHB starts by training every possible configuration on a very small budget and increases it over time, whereas hyperopt uses 20 initial random evaluations to estimate the EI. A typical set of hyperparameters consists of discrete and continuous numerical parameters as well as discrete choices such as the optimizer, the type of layer or activation function and many more, that cannot be directly expressed as numbers. These characteristics and the fact that even relatively small (or shallow) neural networks are computationally expensive to train, make it impossible to just try out many candidate combinations of hyperparameters and choose the best one, because this method is often too restrictive in terms of memory or processing power and very likely to not find the best candidate without some form of guidance. For the task at hand, a search routine with multiple stages is employed, restricting the search space further and further in every stage in order to find a good trade-off between exploring and exploiting (Feurer and Hutter, 2019).
The target function to be minimized over the available hyperparameters is the loss of the fitted neural network on the validation set. In every iteration, a suggestion for a configuration is drawn from the underlying algorithm (hyperopt or BOHB). The model is fitted on the July 2016 dataset mentioned earlier.
There are **two hyperparameter searches each**, for both underlying algorithms and the experiment is carried out as follows (see Schwemmle (2021) for more details on the whole procedure):

1. Define one initial search space for both algorithms.
2. Run each experiment for 500 function evaluations (i.e., 500 training loops) and save the validation loss (MSE) for each evaluation.
3. Take the hyperparameter configuration that achieved the lowest validation MSE and use it as a basis to define a second search space (one search space per algorithm)
4. Run each search procedure again for 500 iterations to explore the new search spaces defined in step 3.

5. Take the two final configurations to build a model accordingly, train it on the full dataset and evaluate it on the test set (the last 5% or roughly 54 days of the data)

The idea is that the first run explores good values for the number of layers and compares the available optimizers. The number of neurons of the LSTM and prediction layers are still part of the first execution to find a rough estimate for them and to account for possible interaction effects of the number of neurons with the number of layers in the LSTM model. The number of iterations was set to 500 to prevent the trials from taking much more than 4 hours because connection instabilities sometimes caused problems for longer runtimes. The second run of the experiments is intended to fine tune the hyperparameters found in the first run by taking the number of layers as a given and exploring the number of neurons for each layer in more detail.
All experiments are run in Google Colab (Bisong, 2019) using Ray Tune (Liaw et al., 2018) to implement the search schedules and underlying algorithms. When using hyperopt, each model is trained for 5 epochs but for BOHB 5 epochs is only the maximum budget allotted in each iteration. BOHB schedules the search in such a way that only the most promising models are trained for 5 epochs, cutting short the training of every model configuration that is clearly inferior to others already seen. By doing this, BOHB can explore many possible configurations quickly at the start and then explore promising models in more detail later.
One caveat of minimizing the validation loss is that the hyperparameters could be implicitly fit to the validation data, meaning that even though the model fitting procedure can never access the validation dataset. Moreover, the hyperparameter optimization routine would just end up with hyperparameters that yield very good validation results by chance but as the resulting models are then evaluated on a totally unrelated test set of the full dataset and perform well this seems to be no issue here.

5. Results

The performance of the resulting models, the model from Section 4.1 and some benchmark methods are shown in Table 1. The considered benchmark methods are as follows:

- **Mean prediction**: Predict the mean, i.e., 0 for all zones[10]
- **Moving average**: The prediction for each zone is equal to the average of the 8 preceding observations in this zone.
- **One week before prediction**: The prediction for each zone is equal to the number of pickups that was observed for this zone exactly one week before, i.e., on the same day at the same time.
- **Current value prediction**: The prediction for each zone is equal to the current value observed for this zone. This means that the assumption for each prediction is that the next 15 minutes will be exactly like the last 15 minutes.
- **XGBoost** (Chen and Guestrin, 2016): A popular tree boosting system that has been shown to achieve very good results on various tasks. For this method, all inputs are put together in one long vector, so it utilizes the same data as the LSTM models

[10] This should give an MSE of exactly 1 because for standardized data the MSE is equivalent to the variance but the data scaling was based on the training set mean and standard deviation (in a real-world application these values are also not known a priori and are therefore often based on the already existing data). Since the overall number of taxi pickups has decreased between 2016 and 2019 the observed MSE for this method is lower than 1.

	Average MSE over 68 zones	Standard deviation of MSE
Mean Prediction	0.672	0.113
Moving average	0.261	0.226
One week before prediction	0.385	0.442
Current value prediction	0.307	0.430
XGBoost	0.266	0.344
LSTM 0 (rough tuning)	0.176	0.219
LSTM 1 (Hyperopt)	**0.172**	0.220
LSTM 2 (BOHB)	0.173	0.221

Table 1: Comparison of fine-tuned models with benchmark methods

Although the fine-tuned models outperform the benchmark model from Section 4.1, this happens only by a tiny margin that does not seem to justify the extensive search procedure presented here. Note that the standard deviation reported in this table is not the standard deviation of the errors over many fitting procedures but the result of calculating one single MSE for every single zone (the mean of these errors clearly is equal to the reported MSE) and then computing the standard deviation of this vector of 68 single errors. If this value would be zero, the model in question would perform equally well (or bad) on all the zones, meaning that either the zones are very similar, or the model captured the characteristics of each zone perfectly. This is clearly not the case and the fine-tuned models also perform almost identically to the benchmark model when looking at this measure.

The results of the experiments show that there is no guarantee that complicated hyperparameter tuning procedures yield a substantially better model for every problem. It seems that the very simple procedure presented in Section 4.1 already performs very well and additional performance are very hard to achieve (at least within the bounds defined for this study). One reason for this might be the discussed structure of the data or the resulting structure of the errors per zone. The existence of a few zones that have almost no pickups most of the time and then a few pickups at a few moments leads to very large errors for these zones even though the model performs very well on many other zones. The high errors can be explained by the fact that the data was standardized and therefore the predicted values express deviations from the mean, so when a low frequency zone has only few observations with many more pickups than usual, the value for this zone will be an extreme outlier that is hard to predict as it happens very rarely and can be therefore not inferred from the past observations or the meta features. These few missed outliers then dominate the error measure for the low frequency zones and inflate the overall error. One solution could be to break up the zone input sequences into two or three brackets depending on their average number of pickups and then using more than one LSTM model to handle the different zones differently. One could also argue that the low frequency zones are of no interest for possible users that want to know where many passengers are picked up and one could therefore restrict the underlying data even more to focus only on those high frequency zones that are of most interest. It is unclear whether such a model that only predicts the pickups for selected zones and has many blind spots would be very favourable.

The experiment runs took on average 169 minutes (201 minutes for the first and 137 minutes for the second run) when using hyperopt and 125 minutes (117 minutes for the first and 133 minutes for the second run) for BOHB. It is likely that these numbers could be improved by better parallelism and maybe more efficient training procedures

but regardless of possible improvements, BOHB has reached an almost identical performance than the hyperopt in significantly less time (the efficient use of available resources is also one main selling point for BOHB mentioned in Falkner et al., 2018).

6. Conclusion

The goal of this work was to find a simple LSTM based neural network that can predict the number of taxi pickups for the next 15 minutes in Manhattan over all relevant Taxi Zones and to explore to what extent this model can be improved by tuning its hyperparameters in different ways. A rough architecture was proposed that builds around a LSTM network but also incorporates weather data and time features using non-recurrent neural networks. Following Zhang et al. (2017), a small subset of the hyperparameters of the proposed model were tuned using only six of the total 68 zones and a small subset of the data. This very fast and simple approach found a model setup intended to be used as a benchmark when exploring different hyperparameter optimization techniques. The resulting model performs satisfactorily with very accurate results for most of the zones but fails to accurately predict the demand for zones with a very low average number of pickups. To improve the model performance, two state-of-the-art hyperparameter optimization techniques (hyperopt and BOHB) are implemented to find the best possible setup within the proposed architecture. The results indicate only a marginal performance gain (2.2% lower MSE) which is much less than what most of the results in the literature would suggest (Komer et al., 2019 report, for example, 5% better performance than the then best-known alternative approach, which is a much higher benchmark than the roughly tuned model here).
The conclusion of this work should by no means be that hyperopt and BOHB do not perform well or that there is no need for hyperparameter tuning but that there are certain limitations that can prevent them from performing as expected. One main limitation is the very heterogeneous data structure that even the best hyperparameter configuration cannot sufficiently account for. It is possible that a much more complicated model is needed (e.g., Ke et al., 2017 or Xu et al., 2018) for this task but even when this is the case it remains true that the rough tuning procedure shown here could not be significantly outperformed by an extensive hyperparameter tuning procedure.

Bibliography

Bergstra, J. S., Bardenet, R., Bengio, Y. and Kégl, B., *Algorithms for hyper-parameter optimization*, **Advances in neural information processing systems**, pp. 2546–2554, 2011

Bergstra, J., Yamins, D. and Cox, D. D., *Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms*, **Proceedings of the 12th Python in science conference**, 13, Citeseer, p. 20, 2013

Bisong, E., *Google Colaboratory*, **Building Machine Learning and Deep Learning Models on Google Cloud Platform**, Apress, Berkeley, CA, pp. 59–64, 2019

Chen, T., Guestrin, C., *Xgboost: A scalable tree boosting system*, **Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining**, ACM. pp. 785–794, 2016

Chu, K. F., Lam, A. Y. and Li, V. O., *Travel demand prediction using deep multi-scale convolutional lstm network*, **21st International Conference on Intelligent Transportation Systems (ITSC)**, IEEE, pp. 1402–1407, 2018

Falkner, S., Klein, A. and Hutter, F., *BOHB: Robust and efficient hyperparameter optimization at scale*, **Proceedings of the 35th International Conference on Machine Learning**, 80, PMLR, pp. 1437–1446, 2018

Feurer, M. and Hutter, F, *Hyperparameter optimization*, **Automated Machine Learning,** Springer International Publishing, pp. 3–33, 2019

Hochreiter, S. and Schmidhuber, J., *Long short-term memory*, **Neural computation**, 9(8), pp. 1735–1780, 1997

Ke, J., Zheng, H., Yang, H. and Chen, X. M., *Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach*, **Transportation Research Part C: Emerging Technologies,** 85, pp. 591–608, 2017

Komer, B., Bergstra, J. and Eliasmith, C., *Hyperopt-Sklearn*, **Automated Machine Learning**, Springer International Publishing, pp. 97–111, 2019

Larochelle, H., Erhan, D., Courville, A., Bergstra, J. and Bengio, Y., *An empirical evaluation of deep architectures on problems with many factors of variation*, **Proceedings of the 24th International Conference on Machine Learning**, ICML '07, Association for Computing Machinery, pp. 473–480, 2007

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. and Talwalkar, A., *Hyperband: A novel bandit-based approach to hyperparameter optimization*, **The Journal of Machine Learning Research**, 18(1), pp. 6765–6816, 2017

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E. and Stoica, I., *Tune: A research platform for distributed model selection and training,* 2018

Ma, T. Y., Rasulkhani, S., Chow, J. Y., and Klein, S., *A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers*, **Transportation Research Part E**, 128, pp. 417-442, 2019

Menne, M. J., Durre, I., Korzeniewski, B., McNeill, S., Thomas, K., Yin, X., Anthony, S., Ray, R., Vose, R. S., Gleason, B. E. and Houston, T., *Global historical climatology network - daily (ghcn-daily), version 3.26*, http://doi.org/10.7289/V5D21VHZ. (Accessed December 9th, 2020), 2012

Safikhani, A., Kamga, C., Mudigonda, S., Faghih, S. S. and Moghimi, B., *Spatiotemporal modeling of yellow taxi demands in new york city using generalized star models*, **International Journal of Forecasting,** 2018

Schwemmle, N., *Short-term spatio-temporal demand pattern predictions of trip demand*, Master Thesis, 10.5281/zenodo.4514434, 2021

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K. and Woo, W.-C., *Convolutional lstm network: A machine learning approach for precipitation nowcasting*, **Advances in neural information processing systems**, pp. 802–810, 2015

Xu, J., Rahmatizadeh, R., Bölöni, L. and Turgut, D., *Real-time prediction of taxi demand using recurrent neural networks*, **IEEE Transactions on Intelligent Transportation Systems**, 19(8), pp. 2572–2581, 2018

Yu, Y., Si, X., Hu, C. and Zhang, J., *A review of recurrent neural networks: Lstm cells and network architectures***, Neural computation,** 31(7), pp. 1235–1270, 2019

Zhang, Q., Wang, H., Dong, J., Zhong, G. and Sun, X., *Prediction of sea surface temperature using long short-term memory*, **IEEE Geoscience and Remote Sensing Letters**, 14(10), pp. 1745–1749, 2017

Zhao, Z., Chen, W., Wu, X., Chen, P. C. and Liu, J., *Lstm network: a deep learning approach for short-term traffic forecast*, **IET Intelligent Transport Systems**, 11(2), pp. 68–75, 2017