Discussion of architectures

The Seoul Bike Sharing dataset is a multivariate dataset with different datatypes (integers, floats and strings). It has 14 attributes: some of them use the same datatype to store values, but with different representation scales: this indicates that some sort of data normalisation is needed. All the variables provide information that is relevant to the problem. This dataset can be considered a labelled dataset, since the number of rented bikes at a given hour of a given day is the target values that the model uses for training. Thus, the dataset is optimal for the use of supervised learning algorithms.

It has 8760 instances: 24 hours for each of the 360 days in the time range that goes from December 1st 2017 to November 30th 2018. The dataset being time-based means that there is a strong relationship between datapoints that are in succession: the time of the year (i.e. "Seasons" and "Date") influences other parameters such as "Rainfall", "Snowfall", "Temperature". An exploratory data analysis shows (as one might expect) that the parameter with highest correlation value to "Rented Bikes" is the "Temperature" (53.8558), followed by "Hour" (41.0257). Unsurprisingly, the two parameters with highest correlation values to "Temperature" are "Dew Point" and "Seasons", which confirms the hypotheses of a strong correlation between seasonality and temperature, which is then reflected on the rented bikes count. This will be relevant in the next section.

The trend of the bike count in the dataset is subject to societal behaviour change: as highlighted in **this Statista report** [1], the average daily bike rental has witnessed a steady increase from 2015 to 2019. The presence of such trend implies that the model is not suitable for being used for predictions in timeframes that are too distant from the one which the model was trained on i.e. the year 2018 in Seoul. This means that unless the model "notices" and learns the linear upward trend of rented bikes and time of the year, it will not be able to generalise well on future or past dates, but it will only be able to produce the likely bike count on a potential day of the 2018 year with fictitious weather information.

The problem at hand is a time series forecasting regression problem: we want to create a model which ultimately produces a number, which is the number of bikes to be made available, and we want it to take advantage of the time-based nature of the dataset. The classes of architectures that best fit this problem are the Multi-Layer Perceptron (MLP) and Recurrent Neural Networks (RNN) architectures.

The MLP is constructed by sequentially initialising one or more layers of neurons, each of which containing an arbitrary number of neurons (excluding the input layer which reflects the dimensionality of the input space) as shown in Figure 1 (left). The information is sequentially propagated forward, starting from the input layer, through the hidden layers, all the way to the final layer. In the hidden layers, each neuron feeds its output to all the neurons of the next hidden layer. The final output is produced only on the final output layer, whose structure is determined by the type of task that is being tried to solve. The complexity of the model can be increased or decreased at will by modifying the number of neurons or number of layers, since the addition of each layer corresponds to the elevation of abstraction of the feature representation learnt by that layer. When dealing with linearly inseparable data, Machine Learning techniques like linear regression are of no use: MLP is next most viable choice in such situations.

The application of an MLP to the problem at hand would be advantageous due to the flexibility of the architecture: they can be used for classification problems in which a label is assigned to a given input, as well as regression problems where -like in our case- a real value is predicted as result. Being the MLP the building block architecture of virtually all other neural network architectures, it can potentially be applied to any problem as long as an appropriate (and powerful enough) representation of the problem is chosen. The downside is that in many non-trivial problems, choosing such representation is too hard, and the MLP is not able to achieve the maximum potential score. Furthermore, there have been developed new architectures that are able to deal specifically with multi-variate time-series predictions more efficiently. One of them is RNN, and, in particular, Long Short-Term Memory architectures (LSTM), which are a variant of RNNs.

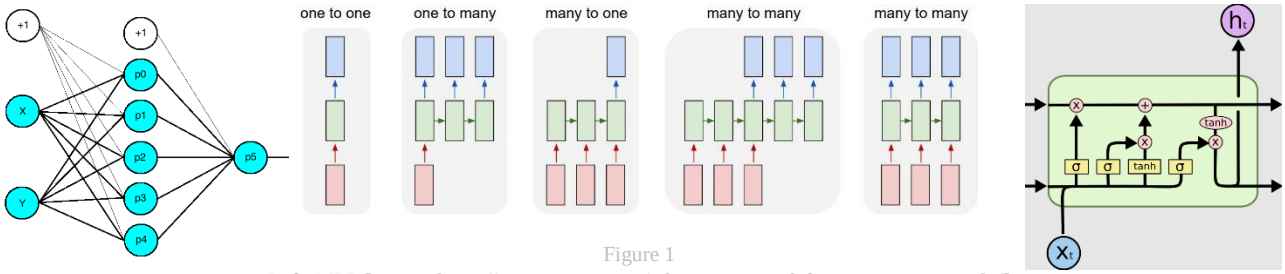


Figure 1
Left: MLP [source: https://iq.opengenus.org/when-to-use-multilayer-perceptrons-mlp/]
Mid: RNN [source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/]
Right LSTM [source: ttp://colah.github.io/posts/2015-08-Understanding-LSTMs/]

RNNs are one of the many advanced architectures that are constructed starting from the MLP. The recurrent nature of the architecture is given by the presence of feedback connections which can pass the output of one iteration to the next: after a number of iterations, the networks is able to take in both the "external" input, as well as the output produced by the previous iteration. What this effectively does is combining information from the past with the information in the present. Another main strength of this architecture is that it can be adapted to reflect the problem at hand and to use the best representation for handling the number of inputs given as well as the output wanted. As shown in Figure 1 (mid) ,The model could be structured so that the only input is received at the beginning of the network, and then information is propagated by being passed as the output of one iteration to the input of the next iteration; alternatively, each iteration could be fed an input which is combined with the output of the previous iteration. A similar concept is applied to how the network produces outputs: it can be produced either during each iteration or at the very end.

What makes RNNs problematic in many situations is their inability to deal with long-term dependencies of data: in our case, an RNN would most likely struggle to take advantage of weather information of several hours prior to the hour that it is being predicted. This makes the length of sequences of data that can be processed by an RNN rather limited. Another issue with the RNN's structure is the exploding/vanishing gradient: this issue, which occurs when the gradient is back-propagated through a very deep network thus being subject to many matrix multiplications, makes it so that if a gradient is considerably smaller than one, it will get lost along the propagation (vanishing gradient), whereas if it is considerably larger than one, it will become unreasonably big (exploding gradient). In both circumstances, the result is a failure in the network's training.

LSTM architectures (a more advanced version of RNNs) address both these issues, while maintaining all the advantages provided by a recurrent architecture. Fundamentally, they are composed of three subsections -the gates- each of which uses a sigmoid function to establish which information is useful to the model, as shown in Figure 1 (right). The forget gate "forgets" (discards) information that is not necessary for the model to make a decision for the task at hand; in the input gate, data is processed, and it is established whether it is relevant for the task or not, and it is decided how much data is allowed to pass through and being added to the cell state of the current time step; the output gate filters the data and decides what to be returned and what to be passed to the next iteration of the model.

One disadvantage of LSTMs though is the inability to obtain, process and retain spatial information about the data, caused by the fact that an LSTM structure can only handle one-dimensional data (in the case of multi-dimensional data, it is unfolded into one-dimension). This leads to a loss of spatial dependencies between data which might be a considerable drawback in some situations. However, such information is not relevant in our context and for the current task at hand, thus making LSTMs one of the best fits for our task, since it allows us to process a large amount of time-series data without having negative repercussions on the model's performance.

Creation and Application of Neural Network

Almost all the data contained in the dataset provides useful information for informing the model's decision. The only exception is the date variable: in order to be used, the "Date" variable would have to be subject to some form of encoding, in spite of the fact that the day's and year's number don't provide any substantial information. This is because 1) expect for the first 744 datapoints which belong to 2017, the whole dataset is

made up of values in the 2018 year (thus including the "year" part of the date does not add any relevant information to the model), and 2) the day of the month gets repeated from one to 28/30/31 in all months, so it is unlikely that the model is able to learn any significant abstraction from such a small dataset, making the inclusion of the "day" part of the date superfluous. One possible argument against it could be that with the inclusion of both the day and month, the model could learn to infer the general trend of the temperature and be able to predict that one of the first days of October is more likely to be hotter than one of the last days of the same month, just like one of the first days of June is more likely to be colder than the last. However, the dataset being limited to only 365 days prevents such inference to occur, thus making this argument invalid. From the date, the month is the only. The only part that adds valuable information to the model is the month since, as discussed earlier, there is a tight relationship between time of the year and rented bikes count. Therefore, a new "Month" variable was added and used in place of "Date", since the month is the only section which provides useful information for the task.

I also decided to include information about the bike counts for the seven days prior: while the day before will have all the 13 variables, the rest 6 days will instead only have the number of rented bikes. Without a doubt, additionally including the values of all the weather information about the past seven days would increase the model's accuracy; however, the computational demand would become unmanageable. Although advanced tools such as Google Colaboratory would be capable of supporting such computation, I opted to employ a wiser, more efficiency-oriented design choice. The relationship between seasonality and rented bikes count mentioned in the introduction implies that the use of the latter variable also carries some information about the former: as shown in Figure 2 (left), the amount of bikes rented on a monthly scale follows the same trend of increasing and decreasing of weather temperatures, with the exception of the month of October. This means that if for the past 7 days the weather temperature has been steadily increasing or decreasing, it is more likely that it will keep following that trend. In the event of an outlier, the weather information provided to the model ensures that the result is in line with the weather rather than being skewed towards following the trend.

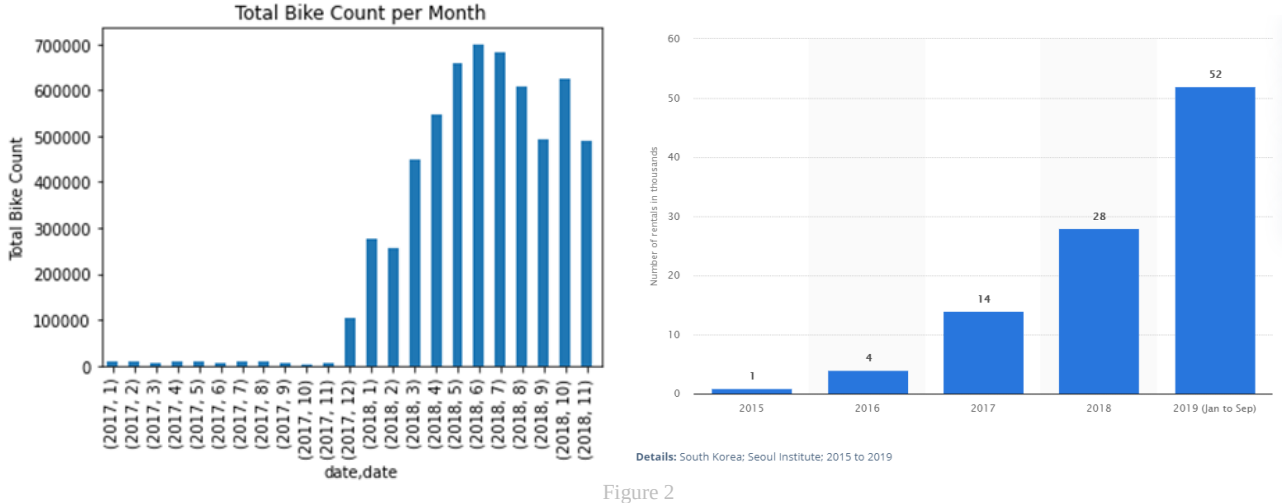


Figure 2
Left: Total Bike Count per Month
Right: Average daily number of Seoul Bike (public bike rental system) rentals in South Korea from 2015 to 2019
[source: https://www.statista.com/statistics/997380/south-korea-seoul-bike-daily-rental-number/]

Before being passed to the model for training, the data has been pre-processed. The categorical values contained in rows "Seasons", "Holiday" and "Functioning Day" have been encoded using integers. All the values have then been turned into floats and then scaled using `MinMaxScaler()` from sklearn, which preserves the shape of the original distribution, but scales all datapoints in the range zero to one. The data is then processed using a `series_to_supervised()` function.

This function, inspired by **a blogpost from MachineLearningMastery** [2], takes care of preparing the dataset into the supervised format which is ideal for a time-series problem. It does so by taking as input the number of past datapoints that are to be considered for making a single prediction, and inserting all the parameters of those datapoints into new columns: the end result for a single point contains not only the weather information about that hour, but also the number of past -in this case- (24*7=)168 hours. Next, since only the "Rented Bike

Count" variable is kept for the datapoints of the 6 days before, the columns containing information about all the other variables were dropped. Of course, the first 168 points in the dataset will not have enough previous points to make such adjustment: while the function as defined in [2] simply drops those rows, I decided to instead take a different approach, since losing 168 datapoints is not ideal and, as mentioned in the first section of the report, the yearly steady increase of number of daily bikes rentals makes it unfeasible to re-use the datapoints of the 2018 year's days that are missing from the 2017 year, since they would not be accurate. Instead, I used an equation using the data in [1] (average daily rental counts in 2017 and 2018 = 14 and 28 respectively, as shown in Figure 2, right):

$$Bike\ Count\ of\ day\ X\ in\ 2017 = \frac{Bike\ Count\ of\ day\ X\ in\ 2018 * 14}{28}$$

While I am aware of the potential fallacies of this method, I believe that it is more valuable to have the resulting datapoints produced in a somewhat reasonable way, than to either drop the initial 168 rows or reduce the amount of past information provided to the model. This is even more reasonable when seen in perspective: a 75/15/15 split was used, which means that the model will be trained using 6570 datapoints, only 168 of which will be potentially inaccurate results. On top of this, seen the upwards-trend of bike rentals over the years, the days towards the end of the year (which have a higher bike count) are arguably more impactful than the first ones, since they make the model tend towards a higher prediction value, which is in line with the real-world trend: thus, values that are incorrectly predicted higher than their true value might not necessarily hurt the model's final accuracy.

The decision of the train/validation/test split was chosen among the most common splits 70/15/15, 75/15/10, 80/10/10, and was heavily influenced by the dataset's composition. The graph in Figure 2 (left) illustrates how partitioning the dataset at different points can have an impact on the amount of data that the model is trained on. For example, using a 10% split for the test data means that the model will be tested on the last 876 hours (25th October to 30th of November), and the data has a downward trend which is likely to be correctly predicted. In contrast, by applying a 15% split on test data, the last 1314 hours (7th of October to 30th of November) will be used for testing which includes almost all of October: as mentioned in the earlier section, this month is the only "anomaly" where a month whose temperature is overall decreasing (which implies a similar behaviour in the number of rented bikes) records an overall higher number of bikes than the previous 2 months. This means that the model would be more likely to incorrectly predict the datapoints in the month of October: by using a 75/15/10 split, it is possible to instead include them in the validation portion, which will instead reinforce the model's prediction ability.

After the train/validation/test split was made, I employed Keras Tuner [3] to find the best configuration of hyperparameters for my model. Through trial-and-error, it was discovered that an architecture of three hidden layers was the most optimal one, which performed better than a similar architecture with one, two, four and five layers instead. The tuner was used for discovering the optimal number of units in the layers of the LSTM network: using batch size of 1 and a search space between 32 and 512 with a step of 32, it generated an optimal result of 192 neurons for the first layer, 352 neurons for the second layer and 448 neurons for the third layer. All layers have a bias neuron which is initialised at zero. The activation function used in the LSTM layer is the default tanh with sigmoid for the recurrent sections (as explained in "Discussion of Architectures"), while a linear activation function is used for the single (since we are trying to predict a single value) output layer for predicting the number of bikes. All other arguments of the LSTM layer have been kept as default. Another parameter that was tuned is the learning rate, searching through values between 0.1 and 0.000001 with step of 0.1: the optimal result was 0.0001. Being this value already small, I decided not to employ learning rate decay. This learning rate was passed as parameter to the Adam optimizer, while the other arguments like beta1, beta2 and epsilon were kept at the default values of 0.9, 0.999 and 0.0000001 respectively. The choice of the optimizer was made through trial-and-error: multiple tests have been performed across 1000 epochs and allowing the hyperparameters to be optimized by Keras Tuner, changing the optimizer between SGD [4], RMSprop [5], and Adam [6], and picking the one with the best result across 3 repeated tests. The discussion of results will follow in the "Result and Evaluation" section.

I applied a Batch Normalization layer to deal with internal covariate shift: momentum is kept at 0.99 since I will work with a batch-size of 1, which will allow to ignore the noise in learning; epsilon is also kept at 0.001 in order to minimize its impact on the normalization statistic; beta (learned offset factor) and moving variance are initialised to zero, while both moving mean and gamma (learned scaling factor) are initialised to one, and no regularizers nor constraints have been applied to gamma nor beta, in line with common practices. No dropout layer was inserted after the batch norm, since the random absence of nodes might make the normalization statistic noisy.

Using Keras' Hyperband [7] class, 50 epochs were tested for each combination of hyperparameters: following that, the most optimal configuration was trained for 1000 epochs with a factor of 10 and 5 hyperband iterations, using MSE as the objective to minimise. I also took advantage of Keras' callback feature by using early stopping with patience of 100, which ensures that a temporary drop in accuracy doesn't impede the descent towards lower loss values. By running 1000 epochs, I was able to record where the model reaches a plateau, when it starts to overfit, and when its accuracy starts to worsen. Finally, the model was trained for 683 epochs.

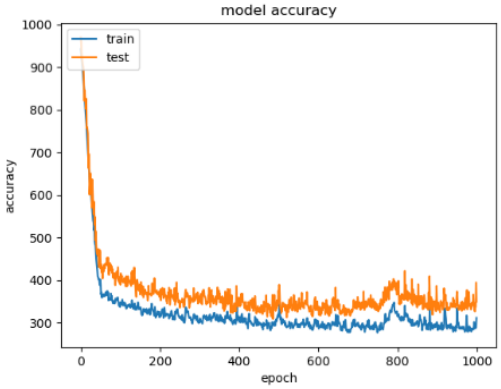
Results and Evaluation

Being this a regression task, the most common evaluation metrics for regression are: Mean Squared Error (MSE), Root MSE (RMSE) and Mean Absolute Error (MAE). I decided to use RMSE as the main metrics to evaluate the models, while also using the other metrics to inform the final decision of which model to keep. On one hand MSE suffers of a negative overestimation of the model's performance, since every error is squared (as shown in Table 1 below), on the other hand it is differentiable, meaning a better optimization can be achieved, which makes this metric widely preferred over the others.

When choosing the right optimizer to use, several runs were performed to make sure the best configuration was found, and although in three out of the five tests run on the validation data the Adam optimizer scored a better result in all the metrics, in the remaining two runs it scored a RMSE result lower not only than RMSprop but also than SGD; despite this, the remaining MAE for Adam being higher led me to the conclusion that the Adam was the optimal choice overall.

Using the best hyperparameter tuning possible for the network, the model was able to produce strong results on the validation data (as shown in Figure 1, where "accuracy" is the RMSE value) and the test data (as shown in Table 1). For deciding whether the model's accuracy is acceptable or not, a different metric was used. The highest value for bikes rented in a single day is 3556: this information will help put the model's results in perspective. In a scenario in which the maximum number of bikes rented was, for example, 350, an RMSE of 290 means that the model is predicting with an accuracy of 17.2%, which effectively means that the model is useless. However, as shown in Table 1, the RMSE error on unseen data is 303.2519, which, when observed in appropriate context, means that the model is predicting with an accuracy of 91.5%, which is extremely good.

Figure 3



	Validation	Test
RMSE	290.6013	303.2519
MSE	84449.0938	91961.6875
MAE	209.757	222.0884

Table 1

Despite the strong results, there are some reflections to make. Firstly, throughout the testing process it was evident that the incremental addition of samples to the training data produced increasingly better results. One likely explanation is that the small sample size on which the model was trained on makes the model unstable: since there are very few samples, each one of them has a heavier weight; thus outliers have a great destructive effect on the model's accuracy. On top of this, the overall upward trend of bike rentals discussed earlier, makes

the training process even less trivial and introduces a higher level of difficulty. It is also interesting to notice how the model's performance drastically worsens right after producing the best results. After this peak, the loss resumes its downward route, suggesting that perhaps the model could be able to achieve higher performance if more epochs were run. Future work might further investigate whether training the model for longer periods of time yields better results and explore if different combinations of less parameters passed to the model can achieve a lower RMSE than the one obtained by the proposed model. Additionally, in the event that a larger dataset is released with longer time-periods covered and perhaps more parameters included, future work may examine the performance of the proposed model trained on bigger samples of data.

References

[1] Statista, "South Korea: Seoul Bike daily rental number 2015-2019." https://www.statista.com/statistics/997380/south-korea-seoul-bike-daily-rental-number/ (accessed Mar. 30, 2021).

[2] MachineLearningMastery and Jason Brownlee, "Multivariate Time Series Forecasting with LSTMs in Keras." https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/ (accessed Mar. 30, 2021).

[3] TensorFlow Core, "Introduction to the Keras Tuner." https://www.tensorflow.org/tutorials/keras/keras_tuner (accessed Mar. 30, 2021).

[4] Keras, "SGD." https://keras.io/api/optimizers/sgd/ (accessed Mar. 30, 2021).

[5] Keras, "RMSprop." https://keras.io/api/optimizers/rmsprop/ (accessed Mar. 30, 2021).

[6] Keras, "Adam." https://keras.io/api/optimizers/adam/ (accessed Mar. 30, 2021).

[7] L. Li, K. Jamieson, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," 2018. Accessed: Mar. 29, 2021. [Online]. Available: http://jmlr.org/papers/v18/16-558.html.