*Forecasting bike-sharing demand using Attention-based LSTM model:*

*The case of New York City*

***Student:*** *Pezhman Lamei*

***Course:*** *Advance Machine Learning*

***Fall 2022***

**1| Summary**

I used the bike-sharing data set of New York City for two years, in 2017 and 2018. I compare the accuracy of the bidirectional LSTM model by adding an attention network. The results show that the accuracy improves by adding the attention network from 0.037 to 0.023 MSE for a train set. We can conclude that the attention mechanism is efficient for the bike-sharing demand of New York City. I noticed that previous projects focused mainly on the accuracy of models than checking how much their model could be useful for new data. By training the model initially, I found that model lacks overfitted data. We should note that some level of overfitting signals us that our model would be appropriate for future data. Therefore, I tried to make the model complicated by changing the number of hidden layers, features, and epochs. Finally, reducing the ADAM optimizer's learning rate (from 0.01 0 0.001) helped me reach relatively overfitted data. Reducing the learning rate increases the number of iterations and challenges the model with the local minimum problem. This contributes to previous findings that, along with emphasizing the model's accuracy, we should note how useful our model is in the future.

**2 | Problem statement**

Policymakers and private sectors focus on providing opportunities for people to behave sustainably in their daily life. Since the motor vehicles account as one of the main sources of greenhouse gas in Europe (40%) and in the U.S. (20%), ride-sharing has been considered one of the main strategies for green behavior (Eren & Uz, 2020). Reportedly, the global bike-sharing service market size was 1.5 billion U.S. dollars in 2018 (Choi & Choi, 2020). The need for optimizing demand management for bike-sharing as a crucial business activity is undeniable (Macioszek & Cieśla, 2022). An optimized bike-sharing system with optimal stations not only motivates individuals to use it with a high satisfaction level but also helps the whole city transportation system by linking bike-sharing users to it more efficiently (Yoon et al., 2012).

Thus, previous literature has applied various prediction models to forecast bike-sharing demand as a significant optimization element. There is sufficient literature forecasting the demand for bike usage by applying various time-series analyses and machine learning methods such as random forest, deep learning, and neural networks (Mehdizadeh Dastjerdi & Morency, 2022; Albuquerque et al., 2021). Some research has explored factors affecting demand as the dependent variable, such as weather conditions, taxi usage, and spatial variables (Ashqar et al., 2019; Singhvi et al., 2015). For example, the data in this project, which contains nine features, shows that factors including seasonality, the hour of the day, and the day of the week impact the trend (See Appendix 1). Moreover, demand is not stationary data, thus making forecasting bike-sharing time series data more challenging.

Chart

Description automatically generated

**Figure 1- Daily trend of demand**

Given multiple affecting variables on bike-sharing demand and non-stationary type of data, demand prediction is the potential for error and needs complicated models (Heydari et al., 2021). Accordingly, recent efforts have used several developments in the Recurrent neural networks (RNN) model, mainly the LSTM approach, to contribute to this literature stream by improving the prediction accuracy (Jiang, 2022; Pan et al., 2019). Nevertheless, a considerable challenge found in previous literature is the use range of the prediction models. Most of the existing studies evaluate the proposed method with only one or two datasets without the guarantee that the success of an effective method in one bike-share system could be replicated in another one (Jiang, 2022).

Therefore, our motivation is to focus on improving the LSTM model accuracy by adding transformer architecture (e.g., attention network) which is found rarely in previous studies. Furthermore, I focus on an issue that previous works haven’t paid attention adequately. Previous works emphesize on the mode’s accuracy and do not report how much their model would be useful for future datasets. I propose that overfitting is not unpleasant issue always and we need a level of overfitted data to ensure that our model will work in future. I give a solution for this problem by regulating ADAM learning rate. Given the detrimental role of vehicle motors, an efficient bike-sharing system is vital for densely populated cities to improve the transportation system (Eren & Uz, 2020). Therefore, there is no question that a bike-sharing system is necessary for New York City as the most populous and dense city in the U.S. This city has a population of 8,804,190 with 300.46 square miles (778.2 km2) (U.S. Census Bureau, 2020).

1. **| Techniques**

The focal technique used in this paper is the LSTM (long short-term memory) model, which is an evolution of RNN. When trained on long time series, RNNs typically suffer from the vanishing gradient or exploding gradient problem, which means that the parameters in the hidden layers either don’t change that much or they lead to numeric instability and chaotic behavior. This happens because the gradient of the cost function affects its memorizing capacity. Since one prominent feature of time series data is the long-term trend, we can conclude that RNN, which suffers from weak memory, is incapable of optimized forecasting. The alternative approach is the LSTM model that overcomes the vanishing gradient problem in the standard RNN by improving the gradient flow within the network (Mehdizadeh Dastjerdi & Morency, 2022). I use the Bidirectional LSTM model, which trains two instead of one LSTM on the input sequence. The first is on the input sequence as-is, and the second is on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.

In LSTM, each input corresponds to an output for the same time step. However, in many real cases, we want to predict an output sequence given an input sequence of different lengths without correspondence between each input and each output. To solve this challenge, the transformer architecture helps us to predict models with different input and output lengths. One of the frontiers of transformer architecture is the Attention network. Besides the advantage of predicting long input sequences ( relative to Seq2Seq models), the Attention network assigns different importance to the different elements of the input sequence and gives more attention to the more relevant inputs. I add this network to our Bidirectional LSTM model to test if it outperforms our first technique.

Last, I trained the model initially without drop out layer and ADAM optimizer. Previous findings indicate that this optimizer outperforms other methods in LSTM implementation. One point that I should note is that I changed the learning rate (from 0.01 to 0.001) in ADAM optimizer to reach better results. In the next section, I explain why I chose to change the learning rate regarding overfitting. I trained 90 % of the data as the training set. I trained three models: one without a drop-out layer, one with a drop-out layer, and one with an attention layer. Two hundred epochs are chosen to operate, and I change the batch size from 16 to 32 in order to control overfitting.

**4| Results & Conclusion**

Figure 2 shows the loss values for three models. All graphs imply that we have no overfitted data. Although we try to avoid overfitting problems in any ML and DL projects, observing a model with the ability to overfit is a good sign. If a model can overfit, it has enough entropic capacity to extract features (in a meaningful and non-meaningful way) from data. However, moving from model 1 to 3, we can observe that there is relatively overfitted data (difference between validation loss and training loss). We can infer that the model may not be complicated enough to generalize further similar datasets. I found that most bike-sharing demand forecasting projects provided by prior scientists have little attention to the generalization attribute (Jiang, 2020). For example, I reviewed several Kaggle projects in the related competition and figured out most of them get no overfitted results (See the example). Since a lower learning rate increases the number of iterations and causes the local minimum problem, this may help us to receive a reasonable level of overfitting. Therefore, I changed the learning rate from 0.01 to 0.001, which was helpful.

Graphical user interface, chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

**Figure 2- Validation and training loss**

To check the accuracy of models and the effectiveness of attention network, I compare the mean squared error of two models. Table 1 reports the results of error for two models incidating that adding attention layer improve the accuracy.

|  |  |  |
| --- | --- | --- |
| Models | Train set MSE | Test set MSE |
| LSTM | 0.03795494884252548 | 0.04994972422719002 |
| Attention-based LSTM | 0.02390124835073948 | 0.049110546708106995 |

**Table 1- Accuracy of models**

We conclude that attention-based LSTM model outperforms bidirectional LSTM in our bike-sharing demand forecasting project. The point that we should note is that transformer architecture does not necessarily perform better than regular RNN model. This depends on how much our data contains long sequences and has complicated structure.

The other conclusion from the results is that learning rate in ADAM optimizer is critical, such that lower learning rate helps us to have more iterations and probabely better generalized results. In this project, I encountered that the model ,with 0.01 learning rate of optimizer, suffers relatively more from the lack of overfitted data. As I explained, this implies that model may not perform well for new data. Potential reasons for this problem for this specific dataset are lack of important features and simplicity of model. For instance, literature has reported that we need public transportation spot locations to have better forecasting model for bike-sharing data. However, I tried to mitigate this negative point by regulating ADAM optimizer and batch size.

**5| Contribution**

This project has three contributions to the previous literature on time series forecasting. First, most of the previous projects focus on how much their model predicts well by reporting accuracy metrics, including MAPE, MSE, and RMSE. However, it is worth noting how much the model is overfitted. In fact, overfitting is not our foe, and in contrast, it could signal us about the usability of the model in the future. In this project, I showed that forecasting bike sharing data set by simple bidirectional LSTM lack of overfitting and lowering the learning rate of ADAM optimizer results in slightly overfitted data. This ensures that the model may perform efficiently with future similar data, specifically for bike-sharing demand, which depends on many unexpected factors, such as other transportation spots. Thus, training a generalized model is essential rather than replicating numerous projects. Therefore, I suggest further efforts to reduce the ADAM optimizer from 0.01 to 0.001 to have a better model.

Second, I illustrate that adding an attention network to bidirectional LSTM is efficient and provide better performance to our model. Testing the efficiency of the attention network is necessary because this transformer does not work optimally with short sequences and simple datasets. Additionally, the attention mechanism gives outstanding results in NLP models since it allows one to remember all the words in the input and recognize the most relevant words when formulating a response. This project contributes to the research using an attention network by noting that an attention-based model is efficient for time-series demand datasets.

Third, prior studies have predicted bike-sharing usage in various contexts and cities, such as New York, Washington, Chicago, London, and Montreal (Heydari et al., 2021). Findings show that users’ bike-sharing behavior differs by the structure of the city. For example, this behavior depends on the population and density of cities. Thus, showing the efficiency of the attention-based model for this specific dataset contributes to previous literature on bike-sharing demand forecasting. The bike-sharing companies in New York City can use an attention-based model for their prediction by following this project’s coding and results.

**References**

["2020 Population and Housing State Data"](https://www.census.gov/library/visualizations/interactive/2020-population-and-housing-state-data.html). [United States Census Bureau](https://en.wikipedia.org/wiki/United_States_Census_Bureau). August 22, 2021. Retrieved November 24, 2021.

Albuquerque, V., Sales Dias, M., & Bacao, F. (2021). Machine Learning Approaches to Bike-Sharing Systems: A Systematic Literature Review. In ISPRS International Journal of Geo-Information (Vol. 10, Issue 2, p. 62). MDPI AG. https://doi.org/10.3390/ijgi10020062

Ashqar, H. I., Elhenawy, M., Almannaa, M. H., Ghanem, A., Rakha, H. A., & House, L. (2017). Modeling bike availability in a bike-sharing system using machine learning. In 2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS). https://doi.org/10.1109/mtits.2017.8005700

Choi, Y., & Choi, E. J. (2020). Sustainable Governance of the Sharing Economy: The Chinese Bike-Sharing Industry. In Sustainability (Vol. 12, Issue 3, p. 1195). MDPI AG. https://doi.org/10.3390/su12031195

Chollet, F. (2021). Deep learning with Python. Simon and Schuster.

Eren, E., & Uz, V. E. (2020). A review on bike-sharing: The factors affecting bike-sharing demand. In Sustainable Cities and Society (Vol. 54, p. 101882). Elsevier BV. https://doi.org/10.1016/j.scs.2019.101882

Heydari S, Konstantinoudis G, Behsoodi AW. (2021) Effect of the COVID-19 pandemic on bike-sharing demand and hire time: Evidence from Santander Cycles in London. PLoS ONE 16(12): e0260969. https://doi.org/10.1371/journal.pone.0260969

Hossain, M. (2021). The effect of the Covid-19 on sharing economy activities. In Journal of Cleaner Production (Vol. 280, p. 124782). Elsevier BV. https://doi.org/10.1016/j.jclepro.2020.124782

Jiang, W. Bike sharing usage prediction with deep learning: a survey. Neural Comput & Applic 34, 15369–15385 (2022). https://doi.org/10.1007/s00521-022-07380-5

Macioszek, E., & Cieśla, M. (2022). External Environmental Analysis for Sustainable Bike-Sharing System Development. In Energies (Vol. 15, Issue 3, p. 791). MDPI AG. https://doi.org/10.3390/en15030791

Mehdizadeh Dastjerdi, A., & Morency, C. (2022). Bike-Sharing Demand Prediction at Community Level under COVID-19 Using Deep Learning. In Sensors (Vol. 22, Issue 3, p. 1060). MDPI AG. https://doi.org/10.3390/s22031060

Pan, Y., Zheng, R. C., Zhang, J., & Yao, X. (2019). Predicting bike sharing demand using recurrent neural networks. In Procedia Computer Science (Vol. 147, pp. 562–566). Elsevier BV. https://doi.org/10.1016/j.procs.2019.01.217

Singhvi, D., Singhvi, S., Frazier, P., Henderson, S.G., O'Mahony, E., Shmoys, D.B., & Woodard, D.B. (2015). Predicting Bike Usage for New York City's Bike Sharing System. AAAI Workshop: Computational Sustainability.

Yoon, J. W., Pinelli, F., & Calabrese, F. (2012). Cityride: A Predictive Bike Sharing Journey Advisor. In 2012 IEEE 13th International Conference on Mobile Data Management. 2012 13th IEEE International Conference on Mobile Data Management (MDM). IEEE. https://doi.org/10.1109/mdm.2012.16

**Appendix 1 (Demand and affecting variables)**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Appendix 2 (Python Codes)**

**Github repo :**

[**https://github.com/pezhmanlamei/Pezhman\_Lamei\_MachineLearning.git**](https://github.com/pezhmanlamei/Pezhman_Lamei_MachineLearning.git)

pip install attention

import numpy as np

import tensorflow as tf

from tensorflow import keras

import pandas as pd

import seaborn as sns

from pylab import rcParams

import matplotlib.pyplot as plt

from matplotlib import rc

from sklearn.model\_selection import train\_test\_split

from pandas.plotting import register\_matplotlib\_converters

from attention import Attention

from tensorflow.keras import Sequential

from tensorflow.keras.callbacks import Callback

from tensorflow.keras.layers import Dense, Dropout, LSTM,Bidirectional

from pandas import read\_csv

import numpy as np

from keras import Model

from keras.layers import Layer

import keras.backend as K

from keras.layers import Input, Dense, SimpleRNN

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.metrics import mean\_squared\_error

%matplotlib inline

%config InlineBackend.figure\_format='retina'

register\_matplotlib\_converters()

sns.set(style='whitegrid', palette='muted', font\_scale=1.5)

rcParams['figure.figsize'] = 22, 10

RANDOM\_SEED = 42

np.random.seed(RANDOM\_SEED)

tf.random.set\_seed(RANDOM\_SEED)

df = pd.read\_csv(

"New\_york.csv",

parse\_dates=['timestamp'],

index\_col="timestamp"

)

df.shape

df.head()

df['hour'] = df.index.hour

df['day\_of\_month'] = df.index.day

df['day\_of\_week'] = df.index.dayofweek

df['month'] = df.index.month

sns.lineplot(x=df.index, y="cnt", data=df);

df\_by\_month = df.resample('M').sum()

sns.lineplot(x=df\_by\_month.index, y="cnt", data=df\_by\_month);

fig,(ax1, ax2, ax3, ax4)= plt.subplots(nrows=4)

fig.set\_size\_inches(18, 28)

sns.pointplot(data=df, x='hour', y='cnt', ax=ax1)

sns.pointplot(data=df, x='hour', y='cnt', hue='is\_holiday', ax=ax2)

sns.pointplot(data=df, x='hour', y='cnt', hue='is\_weekend', ax=ax3)

sns.pointplot(data=df, x='hour', y='cnt', hue='season', ax=ax4);

fig,(ax1, ax2)= plt.subplots(nrows=2)

fig.set\_size\_inches(18, 14)

sns.pointplot(data=df, x='day\_of\_week', y='cnt', ax=ax1)

sns.pointplot(data=df, x='day\_of\_week', y='cnt', hue='season', ax=ax2);

train\_size = int(len(df) \* 0.9)

test\_size = len(df) - train\_size

train, test = df.iloc[0:train\_size], df.iloc[train\_size:len(df)]

print(len(train), len(test))

from sklearn.preprocessing import RobustScaler

f\_columns = ['t1', 't2', 'hum', 'wind\_speed']

f\_transformer = RobustScaler()

cnt\_transformer = RobustScaler()

f\_transformer = f\_transformer.fit(train[f\_columns].to\_numpy())

cnt\_transformer = cnt\_transformer.fit(train[['cnt']])

train.loc[:, f\_columns] = f\_transformer.transform(train[f\_columns].to\_numpy())

train['cnt'] = cnt\_transformer.transform(train[['cnt']])

test.loc[:, f\_columns] = f\_transformer.transform(test[f\_columns].to\_numpy())

test['cnt'] = cnt\_transformer.transform(test[['cnt']])

def create\_dataset(X, y, time\_steps=1):

Xs, ys = [], []

for i in range(len(X) - time\_steps):

v = X.iloc[i:(i + time\_steps)].values

Xs.append(v)

ys.append(y.iloc[i + time\_steps])

return np.array(Xs), np.array(ys)

time\_steps = 10

*# reshape to [samples, time\_steps, n\_features]*

X\_train, y\_train = create\_dataset(train, train.cnt, time\_steps)

X\_test, y\_test = create\_dataset(test, test.cnt, time\_steps)

print(X\_train.shape, y\_train.shape)

from keras import backend as K

model = keras.Sequential()

model.add(

keras.layers.Bidirectional(

keras.layers.LSTM(

units=128,

input\_shape=(X\_train.shape[1], X\_train.shape[2])

)

)

)

model.add(keras.layers.Dense(units=1))

optimizer = keras.optimizers.Adam(lr=0.01)

model.compile(loss='mean\_squared\_error', optimizer='adam')

K.set\_value(model.optimizer.learning\_rate, 0.001)

history = model.fit(

X\_train, y\_train,

epochs=200,

batch\_size=16,

validation\_split=0.1,

shuffle=False

)

import matplotlib.pyplot as plt

history\_dict = history.history

loss\_values = history\_dict["loss"]

val\_loss\_values = history\_dict["val\_loss"]

epochs = range(1, len(loss\_values) + 1)

plt.plot(epochs, loss\_values, "bo", label="Training loss")

plt.plot(epochs, val\_loss\_values, "b", label="Validation loss")

plt.title("Training and validation loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

from keras import backend as K

model = keras.Sequential()

model.add(

keras.layers.Bidirectional(

keras.layers.LSTM(

units=128,

input\_shape=(X\_train.shape[1], X\_train.shape[2])

)

)

)

model.add(keras.layers.Dropout(rate=0.2))

model.add(keras.layers.Dense(units=1))

model.compile(loss='mean\_squared\_error', optimizer='adam'

history = model.fit(

X\_train, y\_train,

epochs=200,

batch\_size=32,

validation\_split=0.1,

shuffle=False

)

import matplotlib.pyplot as plt

history\_dict = history.history

loss\_values = history\_dict["loss"]

val\_loss\_values = history\_dict["val\_loss"]

epochs = range(1, len(loss\_values) + 1)

plt.plot(epochs, loss\_values, "bo", label="Training loss")

plt.plot(epochs, val\_loss\_values, "b", label="Validation loss")

plt.title("Training and validation loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

y\_pred = model.predict(X\_test)

train\_mse = model.evaluate(X\_train, y\_train)

test\_mse = model.evaluate(X\_test, y\_test)

*# Print error*

print("Train set MSE = ", train\_mse)

print("Test set MSE = ", test\_mse)

y\_train\_inv = cnt\_transformer.inverse\_transform(y\_train.reshape(1, -1))

y\_test\_inv = cnt\_transformer.inverse\_transform(y\_test.reshape(1, -1))

y\_pred\_inv = cnt\_transformer.inverse\_transform(y\_pred)

plt.plot(np.arange(0, len(y\_train)), y\_train\_inv.flatten(), 'g', label="history")

plt.plot(np.arange(len(y\_train), len(y\_train) + len(y\_test)), y\_test\_inv.flatten(), marker='.', label="true")

plt.plot(np.arange(len(y\_train), len(y\_train) + len(y\_test)), y\_pred\_inv.flatten(), 'r', label="prediction")

plt.ylabel('Bike Count')

plt.xlabel('Time Step')

plt.legend()

plt.show();

plt.plot(y\_test\_inv.flatten(), marker='.', label="true")

plt.plot(y\_pred\_inv.flatten(), 'r', label="prediction")

plt.ylabel('Bike Count')

plt.xlabel('Time Step')

plt.legend()

plt.show();

model = Sequential([

*#Bidirectional(LSTM(units=100,input\_shape=(None, 2), return\_sequences=True)),*

LSTM(units=100,input\_shape=(X\_train.shape[1], X\_train.shape[2]), return\_sequences=True),

Dropout(0.2),

LSTM(units=100,return\_sequences=True),

LSTM(units=50,return\_sequences=True),

Dropout(0.2),

Attention(),

Dense(1, activation='linear')

])

model.compile(loss='mse', optimizer='adam')

history = model.fit(

X\_train, y\_train,

epochs=200,

batch\_size=32,

validation\_split=0.1,

shuffle=False

)

import matplotlib.pyplot as plt

history\_dict = history.history

loss\_values = history\_dict["loss"]

val\_loss\_values = history\_dict["val\_loss"]

epochs = range(1, len(loss\_values) + 1)

plt.plot(epochs, loss\_values, "bo", label="Training loss")

plt.plot(epochs, val\_loss\_values, "b", label="Validation loss")

plt.title("Training and validation loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

y\_pred = model.predict(X\_test)

train\_mse = model.evaluate(X\_train, y\_train)

test\_mse = model.evaluate(X\_test, y\_test)

*# Print error*

print("Train set MSE = ", train\_mse)

print("Test set MSE = ", test\_mse)

y\_train\_inv = cnt\_transformer.inverse\_transform(y\_train.reshape(1, -1))

y\_test\_inv = cnt\_transformer.inverse\_transform(y\_test.reshape(1, -1))

y\_pred\_inv = cnt\_transformer.inverse\_transform(y\_pred)

plt.plot(np.arange(0, len(y\_train)), y\_train\_inv.flatten(), 'g', label="history")

plt.plot(np.arange(len(y\_train), len(y\_train) + len(y\_test)), y\_test\_inv.flatten(), marker='.', label="true")

plt.plot(np.arange(len(y\_train), len(y\_train) + len(y\_test)), y\_pred\_inv.flatten(), 'r', label="prediction")

plt.ylabel('Bike Count')

plt.xlabel('Time Step')

plt.legend()

plt.show();

plt.plot(y\_test\_inv.flatten(), marker='.', label="true")

plt.plot(y\_pred\_inv.flatten(), 'r', label="prediction")

plt.ylabel('Bike Count')

plt.xlabel('Time Step')

plt.legend()

plt.show();