Forecasting bike rental demand for bikeshare systems

Yongming Ding, 116039910001
Jiawei Gu, 116039910070

1. Abstract

The idea of this project is from a Kaggle competition "Bike Sharing Demand" which provides dataset of Capital Bikeshare in Washington D.C. and asked to combine historical usage patterns with weather data in order to forecast bike rental demand.

This project generally answers a question of "How many bikes will meet users' demand in a future certain time".

2. Introduction

Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousand bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues.

Apart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. Opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position is explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city. Hence, it is expected that most of important events in the city could be detected via monitoring these data.

Predictions about the demand are vital when scheduling maintenance of the current bicycle fleet or when to acquiring additional vehicles. The goal of this project is to forecast the demand for bikes in dependency of weather conditions like outside temperature and calendric information e.g. holidays. This information and the demand structure is provided in a set with two years of daily historical data. The demand is given as the total daily demand and as a split for registered users and casual users. To make predictions machine learning is used to train regressors. Both a neural network and a deep neural network (DNN) regressor is trained for comparison.

3. Dataset

Bike-sharing rental process is highly correlated to the environmental and seasonal settings. For instance, weather conditions, precipitation, day of week, season, hour of the day, etc. can affect the rental behaviors. The core data set is related to the two-year historical log corresponding to years 2011 and 2012 from Capital Bikeshare system, Washington D.C., USA which is publicly available in http://capitalbikeshare.com/system-data. We aggregated the data on two hourly and daily basis and then extracted and added the corresponding weather and seasonal information. Weather information are extracted from http://www.freemeteo.com.

The Kaggle data spans the two years from January 1, 2011 to December 31, 2012. For the purposes of the competition, in addition to training data, there was a standardized, unlabeled test set provided to all contestants. The training data covers the first 11 months, and the test data covers of the last month. The test data consists 10,886 data points, one for each hour, with 12 features – given

below. The test set consists of 6,493 data points with the same features.

Table. 1. Description of data fields

Field	Description
season	1:spring, 2:summer, 3:fall, 4:winter
month	1 to 12
hour	0 to 23
Holiday	weather day is holiday or not
Weekday	day of the week
workingday	if day is neither weekend nor holiday is 1, otherwise is 0
weathersit	- 1: Clear, Few clouds, Partly cloudy, Partly cloudy - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	Normalized temperature in Celsius. The values are divided to 41 (max)
atemp	Normalized feeling temperature in Celsius. The values are divided to 50 (max)
hum	Normalized humidity. The values are divided to 100 (max)
windspeed	Normalized wind speed. The values are divided to 67 (max)
count	count of total rental bikes including both casual and registered

4. Models

a. Neural Network

Neural network is a computational model used in machine learning, computer science and other research disciplines, which is based on a large collection of connected simple units called artificial neurons, loosely analogous to axons in a biological brain. Connections between neurons carry a unidirectional signal with an activating strength that is proportional to the strength of the connection between those neurons. If the combined incoming signals are strong enough, the "postsynaptic" neuron becomes activated and a signal propagates to downstream neurons connected to it. Such systems can be trained from examples, rather than explicitly programmed, and excel in areas where the solution or feature detection is difficult to express in a traditional computer program.

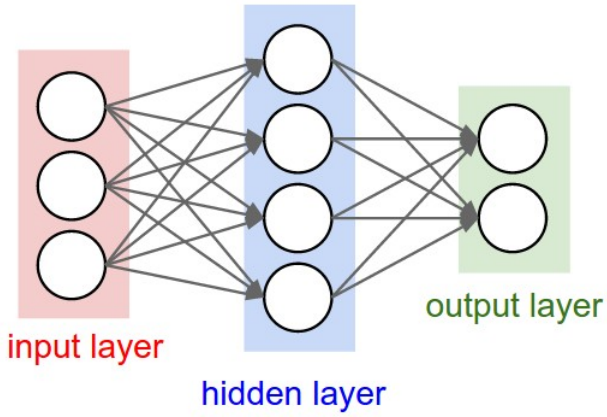

Fig. 1. Neural network model

One way to view the problem is throw at the problem a black-box machine learning model that is able to capture complexity, such as interacting variables. Neural networks are good at recognizing these hidden patterns, hence our choice to train a neural network. After trying a variety of parameters, the final neural network consisted of two hidden layers (50 nodes in the first hidden layer, 50 in the second hidden layer), trained using the parameter epochs of 5000 and learning rate or 0.075.

b. DNN

Deep learning is a class of machine learning algorithms that uses a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised). They are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation. The levels form a hierarchy of concepts.

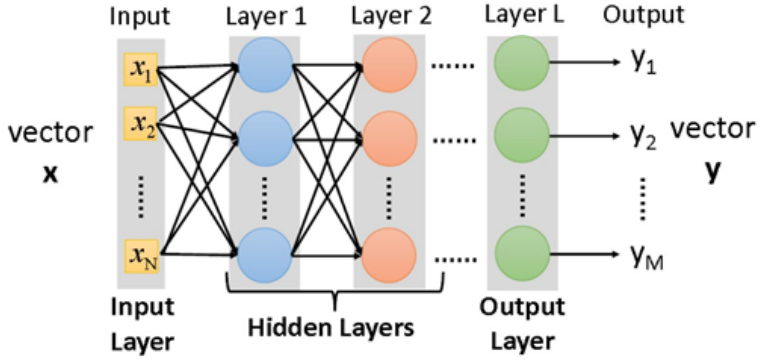

Fig. 2. Deep neural network model

We also use the tensor-flow DNN Regressor algorithm to train the data. We expect that DNN may be better than the neural network because it can solve very complex and non-linear problems if sufficiently tuned. Like all artificial neuronal networks, they come with some disadvantages. Neuronal networks are opaque algorithms, whom parameters and results are hard to interpret. Therefore it was difficult to tune the hyperparameters right.

5. Evaluation methods

To measure the performance of the regressions, two standard regression metrics are used: Root Mean Squared Error (RMSE) and Root Mean Squared Logarithmic Error (RMSLE). Both metrics are calculated for both regressor types.

a. RMSE

The RMSE is the root of the (more common) mean squared error (MSE). The MSE measures the difference between each true value and its prediction. The Differences are squared to eliminate the signs and to amplify lager differences/errors. Because this error would increase with the number of values, the mean error is calculated over all values. Sometimes the MSE is hard to interpret, because the measurement is also squared and does not relate directly to the measurement of the target values. To counter this problem the RMSE takes the root of the MSE to correct for the squared measurement. “The RMSE is directly interpretable in terms of measurement units, and so is a better

measure of goodness of fit than a correlation coefficient." The RMSE is calculated as follows:

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

Where n is the count of observations, y(i) are the true values and y^(i) are the predictions.

b. RMSLE

The RMSLE is calculated as follows:

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

where p_i is the value of prediction for the i-th datapoint, and a_i is the actual value. The metric is reasonable since the error value grows larger as the log value of the ratio of the predictions and the actual values grow larger.

RMSLE is usually used when huge differences in the predicted are not wanted to be penalized and the actual values when both predicted and true values are huge numbers.

6. Code Implementation

We use Python to implement **Neural Network** method **manually** and implement **DNN** method with **tensorflow**.

We push the project repository on Github.com at https://github.com/dreamtalen/bike_sharing. The project structure is shown in Table. 2.

Table. 2. Project structure

Function Name	Job of Function
nn _layer2	2-layer neural network munually implement, including training, prediction and drawing plot.
nn _layer3	3-layer neural network munually implement, including training and prediction.
dnn_local	Local version of DNN implement, including training and prediction and drawing plot.
dnn_server	Server version of DNN implement, including training and prediction.
dnn_drawplot	Get the result from dnn_server.py, draw prediction and loss plot to find the trend intuitively.

Several important code snippets (Please check the complete code in files):

nn _layer2.py

a. Define activation function (sigmoid function)

```
# Activation function is the sigmoid function
self.activation_function = lambda x: 1 / (1 + np.exp(-x))
```

b. Initialize weights between each layers and define learning rate

```
# Initialize weights
self.weights_input_to_hidden1 = np.random.normal(0.0, self.hidden1_nodes**-0.5,
                               (self.hidden1_nodes, self.input_nodes))

self.weights_hidden1_to_hidden2 = np.random.normal(0.0, self.hidden2_nodes**-0.5,
                               (self.hidden2_nodes, self.hidden1_nodes))

self.weights_hidden2_to_output = np.random.normal(0.0, self.output_nodes**-0.5,
                               (self.output_nodes, self.hidden2_nodes))
self.lr = learning_rate
```

c. Forward pass

```
### Forward pass ###
# Hidden layer 1
hidden1_inputs = np.dot(self.weights_input_to_hidden1, inputs)
hidden1_outputs = self.activation_function(hidden1_inputs)

# Hidden layer 2
hidden2_inputs = np.dot(self.weights_hidden1_to_hidden2, hidden1_outputs)
hidden2_outputs = self.activation_function(hidden2_inputs)

# Output layer
final_inputs = np.dot(self.weights_hidden2_to_output, hidden2_outputs)
final_outputs = final_inputs
```

d. Compute output error

```
# Output error
output_errors = targets - final_outputs
```

e. Back propagated error

```
# Backpropagated error
hidden2_errors = np.dot(self.weights_hidden2_to_output.T, output_errors)
hidden2_grad = hidden2_errors * hidden2_outputs * (1 - hidden2_outputs)

hidden1_errors = np.dot(self.weights_hidden1_to_hidden2.T, hidden2_errors)
hidden1_grad = hidden1_errors * hidden1_outputs * (1 - hidden1_outputs)
```

f. Update the weights

```
#  Update the weights
self.weights_hidden2_to_output += self.lr * output_errors * hidden2_outputs.T
self.weights_hidden1_to_hidden2 += self.lr * hidden2_grad * hidden1_outputs.T
self.weights_input_to_hidden1 += self.lr * hidden1_grad * inputs.T
```

g. Define loss function RMSLE

```
def RMSLE(predict_list, actual_list):
    return (sum((math.log(nonNeg(p) + 1) - math.log(nonNeg(a) + 1))**2 for p, a in zip(predict_list, actual_list))/len(predict_list))**0.5
```

h. Data preprocessing

```
# Dummy variables
dummy_fields = ['season', 'weathersit', 'mnth', 'hr', 'weekday']
for each in dummy_fields:
    dummies = pd.get_dummies(rides[each], prefix=each, drop_first=False)
    rides = pd.concat([rides, dummies], axis=1)
fields_to_drop = ['instant', 'dteday', 'season', 'weathersit',
                  'weekday', 'atemp', 'mnth', 'workingday', 'hr']
data = rides.drop(fields_to_drop, axis=1)
# print data.head()

# Scaling target variables
quant_features = ['casual', 'registered', 'cnt', 'temp', 'hum', 'windspeed']
# Store scalings in a dictionary so we can convert back later
scaled_features = {}
for each in quant_features:
    mean, std = data[each].mean(), data[each].std()
    scaled_features[each] = [mean, std]
    data.loc[:, each] = (data[each] - mean)/std

# Splitting the data into training, testing, and validation sets

# Save the last 21 days
test_data = data[-28*24:]
data = data[:-28*24]

# Separate the data into features and targets
target_fields = ['cnt', 'casual', 'registered']
features, targets = data.drop(target_fields, axis=1), data[target_fields]
test_features, test_targets = test_data.drop(target_fields, axis=1), test_data[target_fields]
```

i. Train Model

```
epochs = 5000
learning_rate = 0.075
hidden1_nodes = 50
hidden2_nodes = 50
output_nodes = 1

N_i = train_features.shape[1]
network = NeuralNetwork(N_i, hidden1_nodes, hidden2_nodes, output_nodes, learning_rate)

losses = {'train':[], 'test':[]}
mean, std = scaled_features['cnt']
for e in range(epochs):
    # Go through a random batch of 128 records from the training data set
    batch = np.random.choice(train_features.index, size=128)
    for record, target in zip(train_features.ix[batch].values,
                              train_targets.ix[batch]['cnt']):
        network.train(record, target)

    # Printing out the training progress
    train_loss = RMSLE(network.run(train_features)[0]*std+mean, train_targets['cnt'].values*std+mean)
    test_loss = RMSLE(network.run(test_features)[0]*std+mean, test_targets['cnt'].values*std+mean)
    if not e % 100:
        sys.stdout.write("\rProgress: " + str(100 * e/float(epochs))[:4] \
                         + "% ... Training loss: " + str(train_loss)[:5] \
                         + " ... Test loss: " + str(test_loss)[:5] + '\n')

    losses['train'].append(train_loss)
    losses['test'].append(test_loss)
```

j. Predict the result of test set

```
mean, std = scaled_features['cnt']
predictions = network.run(test_features)*std + mean
ax.plot(predictions[0], label='Prediction')
ax.plot((test_targets['cnt']*std + mean).values, label='Data')
ax.set_xlim(right=len(predictions))
ax.legend()

dates = pd.to_datetime(rides.ix[test_data.index]['dteday'])
dates = dates.apply(lambda d: d.strftime('%b %d'))
ax.set_xticks(np.arange(len(dates))[12::24])
_ = ax.set_xticklabels(dates[12::24], rotation=45)

plt.xlabel('Date')
plt.ylabel('Bike amount')
```

nn_layer3.py

3-layer implement is similar to the 2-layer version. We add one extra hidden layer so the code will not be shown in the report. Please check it in the nn_layer3.py

dnn_local.py

a. Define loss function RMSLE

```
def RMSLE(predict_list, actual_list):
    return (sum((math.log(nonNeg(p) + 1) - math.log(nonNeg(a) + 1))**2 for p, a in zip(predict_list, actual_list))/len(predict_list))**0.5
```

b. Data preprocessing

```
# Load and prepare the data
data_path = 'Bike-Sharing-Dataset/hour2.csv'
MODEL_DIR = 'Models2_505050/'
rides = pd.read_csv(data_path)

fields_to_drop = ['instant', 'dteday', 'yr', 'casual', 'registered']
data = rides.drop(fields_to_drop, axis=1)
# print data.head()

continuous_features = ['temp', 'atemp', 'hum', 'windspeed', 'hr']
categorical_features = ['season',  'holiday',  'workingday', 'weathersit', 'mnth', 'weekday']

# Splitting the data into training, testing, and validation sets

test_data = data[-28*24:]
train_data = data[:-28*24]
```

c. Converting Data into Tensors

```
# Converting Data into Tensors
LABEL_COLUMN = 'cnt'
def input_fn(df, training = True):
    # Creates a dictionary mapping from each continuous feature column nar
    continuous_cols = {k: tf.constant(df[k].values)
                       for k in continuous_features}

    # Creates a dictionary mapping from each categorical feature column na
    categorical_cols = {k: tf.SparseTensor(
        indices=[[i, 0] for i in range(df[k].size)],
        values=df[k].values,
        dense_shape=[df[k].size, 1])
        for k in categorical_features}

    # Merges the two dictionaries into one.
    feature_cols = dict(continuous_cols)
    feature_cols.update(categorical_cols)

    if training:
        # Converts the label column into a constant Tensor.
        label = tf.constant(df[LABEL_COLUMN].values)

        # Returns the feature columns and the label.
        return feature_cols, label

    # Returns the feature columns
    return feature_cols
```

d. Initialize DNN Regressor and train model

```
                                                                    combiner="sum"))
# DNN-Regressor
regressor = tf.contrib.learn.DNNRegressor(
    feature_columns=engineered_features, hidden_units=[200, 200, 200, 200, 200, 200, 200, 200],
    optimizer=tf.train.AdagradOptimizer(0.075)
)

# Training Our Model
step = 5000
losses = {'train':[], 'test':[]}
for train_time in range(1):
    start_time = time.time()

    wrap = regressor.fit(input_fn=train_input_fn, steps=step)

    end_time = time.time()

    predicted_output = regressor.predict(input_fn=lambda : input_fn(train_data, False))
    predicted_output = [i if i > 0 else 0 for i in predicted_output]
    train_loss = RMSLE(predicted_output, train_data[LABEL_COLUMN])

    predicted_output_test = regressor.predict(input_fn=test_input_fn)
    predicted_output_test = [i if i > 0 else 0 for i in predicted_output_test]
    test_loss = RMSLE(predicted_output_test, test_data[LABEL_COLUMN])

    losses['train'].append(train_loss)
    losses['test'].append(test_loss)
```

e. Predict the result of test set

```
predicted_output = regressor.predict(input_fn=test_input_fn)
predicted_output = [i if i > 0 else 0 for i in predicted_output]
print RMSLE(predicted_output, test_data[LABEL_COLUMN])

predictions = predicted_output
targets = list(test_data[LABEL_COLUMN])

fig, ax = plt.subplots(figsize=(64,8))

ax.plot(predictions, label='Prediction')
ax.plot(targets, label='Data')
ax.set_xlim(right=len(predictions))
ax.legend()

dates = pd.to_datetime(rides.ix[test_data.index]['dteday'])
dates = dates.apply(lambda d: d.strftime('%b %d'))
ax.set_xticks(np.arange(len(dates))[12::24])
_ = ax.set_xticklabels(dates[12::24], rotation=45)
plt.show()
```

dnn_server.py

DNN implement server version is similar to the local version. So the code will not be shown in the report. Please check it in the dnn_server.py

7. Results

We scan the parameter unit number and epochs to find the better error rate of the neural network because neuronal networks are opaque algorithms, whom parameters and results are hard to interpret. Two-layer neural network error rates with different unit numbers and epochs are shown in Table 3 while Three-layer neural network error rates with different unit numbers and epochs are shown in Table 4.

We can find that the final neural network consisted of two hidden layers (50 nodes in the first hidden layer, 50 in the second hidden layer), trained using the parameter epochs of 5000 and learning rate or 0.075 is the best one.

Table. 3. Two-layer neural network error rates with different unit numbers and epochs

Unit number /epochs	1000	2000	3000	4000	5000
10-10	0.719	0.6503	0.6582	0.6443	0.6701
20-20	0.7887	0.7999	0.7168	0.706	0.6847
30-30	0.6608	0.5885	0.5931	0.5954	0.5974
40-40	0.5948	0.5997	0.6042	0.6093	0.6128
50-50	0.57	0.5293	0.5463	0.5676	0.5371
60-60	0.5408	0.5169	0.5412	0.5607	0.5861
70-70	0.62	0.6049	0.5758	0.5498	0.6225
80-80	0.733	0.705	0.677	0.664	0.617
90-90	0.744	0.714	0.642	0.641	0.621
100-100	0.713	0.631	0.573	0.581	0.551

Table. 4. Two-layer neural network error rates with different unit numbers and epochs

Unit number /epochs	1000	2000	3000	4000	5000
10-10-10	0.5797	0.6027	0.6102	0.558	0.562
20-20-20	0.5447	0.5756	0.603	0.6023	0.5581
30-30-30	0.6587	0.6417	0.6181	0.6197	0.6309
40-40-40	0.7377	0.6714	0.6708	0.7525	0.7298
50-50-50	0.7163	0.8223	0.8305	0.7723	0.768
60-60-60	0.5243	0.5198	0.5151	0.5026	0.5359
70-70-70	0.5474	0.5676	0.5228	0.5208	0.517
80-80-80	0.6529	0.613	0.648	0.5978	0.5996
90-90-90	0.7064	0.7011	0.6754	0.6331	0.5901
100-100-100	0.7779	0.7626	0.7768	0.7313	0.7313

As shown in figure 3, it illustrates the training loss and test loss along with the training step. We can find that test loss is slightly larger than the training loss.

As shown in figure 4, it illustrates the prediction and data along with the date. We can find that the prediction is almost in line with the data.

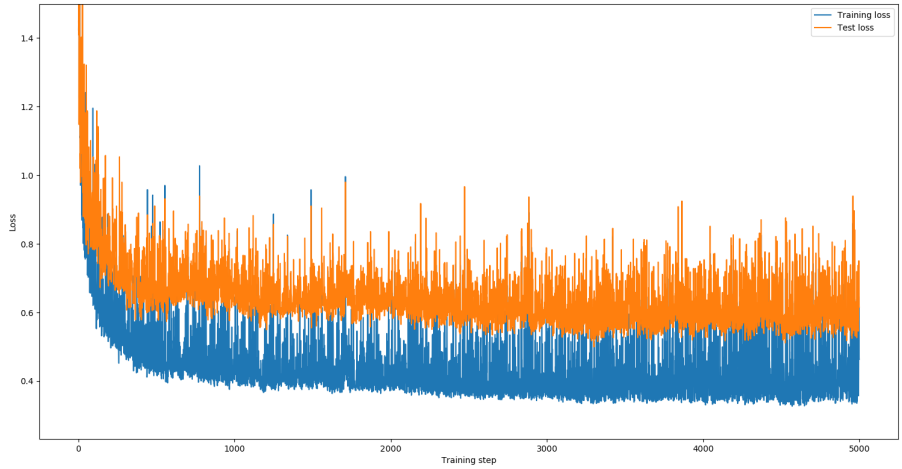

Fig. 3. Training loss and test loss of neural network with different training steps

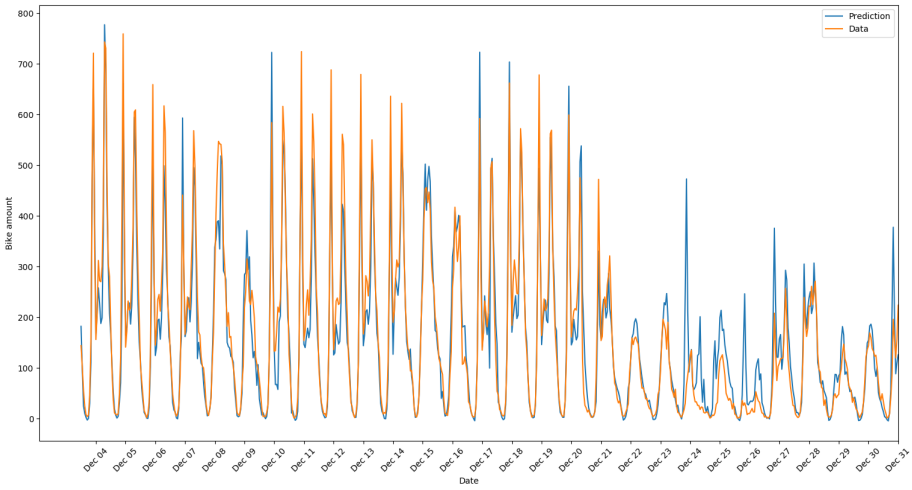

Fig. 4. Prediction and data of neural network with different dates

We scan the parameter unit number and epochs to find the better error rate of the deep neural network because deep neuronal networks are opaque algorithms, whom parameters and results are hard to interpret. Deep neural network error rates with different unit numbers and epochs are shown in Table 5.

We can find that the final deep neural network consisted of five hidden layers (200 nodes in the first hidden layer, 100 in the second hidden layer, 80 in the third hidden layer, 50 in the fourth hidden layer, 30 in the fifth hidden layer), trained using the parameter epochs of 5000 and learning rate or 0.075 is the best one.

Table. 5. Deep neural network error rates with different unit numbers and epochs

Unit number / epochs	20000	40000	60000	80000	100000
10-20-40-80	0.592	0.566	0.573	0.556	0.565
100-100-80-50	0.593	0.570	0.547	0.597	0.528
200-100-80-50-30	0.537	0.485	0.559	0.548	0.486
200-200-200-200-200-200-200-200	0.589	0.578	0.595	0.582	0.578

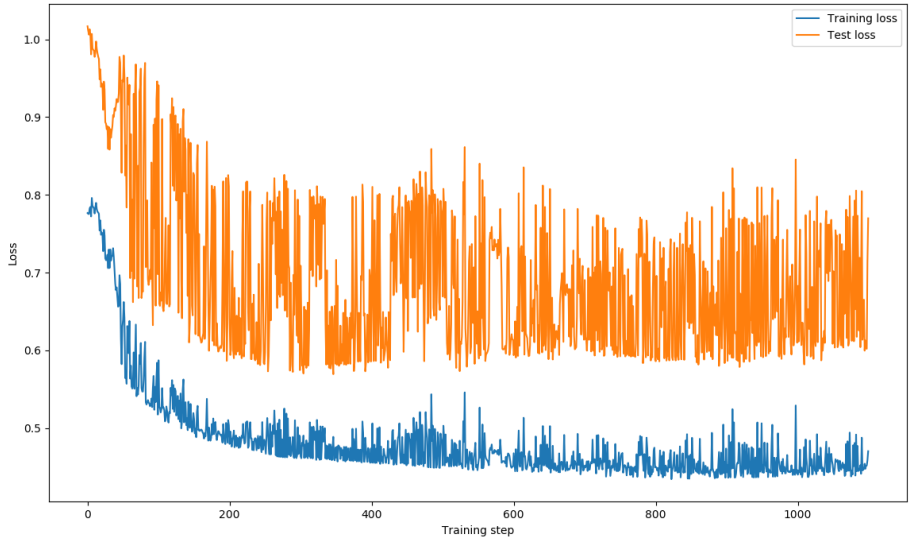

Fig. 5. Training loss and test loss of deep neural network with different training steps

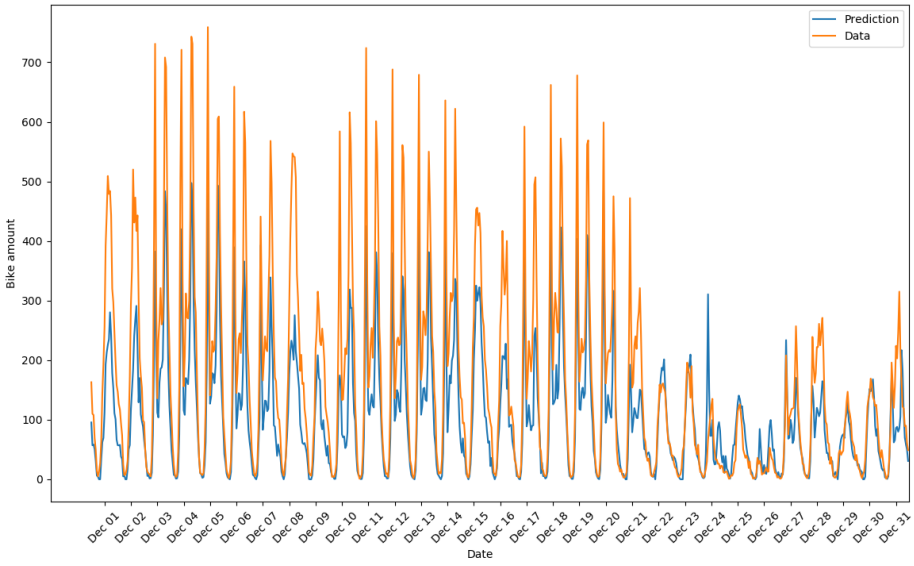

Fig. 6. Prediction and data of deep neural network with different dates

As shown in figure 5, it illustrates the training loss and test loss along with the training step. We can find that test loss is slightly larger than the training loss.

As shown in figure 6, it illustrates the prediction and data along with the date. We can find that the prediction is almost in line with the data.

8. Discussion

Through table 3, 4 and 5, we can find that various unit numbers can affect the precision of the prediction.

By comparison between the neural network and deep neural network, we can find that although the error rate of deep neural network is slightly better than the neural network one, in the figure 4 and figure 6, the prediction of deep neural network in the peak of the data is not as precise as the neural network one.

In practical, we consider that neural network may do the better job because the peak bike demand is much more important in this particular system.