Predict future sales

Yijia Chen

Introduction

The capstone is to use deep learning network to predict future sales. The data is from Kaggle: https://www.kaggle.com/c/competitive-data-science-predict-future-sales
In this project, I first explore the dataset and operate data preprocessing. Since this dataset is a time series so I choose LSTM as the main model of this project. In this project, we use Gradient Descent Algorithm to optimize the network. In the end we evaluate the result by Root Mean Square error.

Description of Dataset

The dataset is provided by 1C Company, one of the largest Russian software firms. The dataset includes train and test dataset, as well as information about each store, category and product. The training dataset includes 2,935,849 rows and testing dataset includes 214,200 rows.

You could download the data from this link:
https://storage.googleapis.com/predict_future_sales_data/data.zip

Long Short Term Memory

LSTM stands for Long Short Term Memory. It is a special model of Recurrent Neural Network (RNN). A typical neural network such as Convolution Neural Network (CNN) has no memory. It can not learn from the previous few steps. While RNN is able to learn the sequence happened in the past. However, if the previous useful information is very far from where it needed, RNN gets harder to learn. At this point, scientists developed Long short term memory.

LSTMs are explicitly designed to avoid the long-term dependency problem. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. These are to protect and control the cell state. First, the forget gate will decide which information to neglect. Second, LSTM need to think which information to store, it will need input gate, a sigmoid layer and a tanh layer. Here the new

information might replace the old one that stored in the cell. Last step is to decide what to output through the output gate. Then a simple basic LSTM finished.

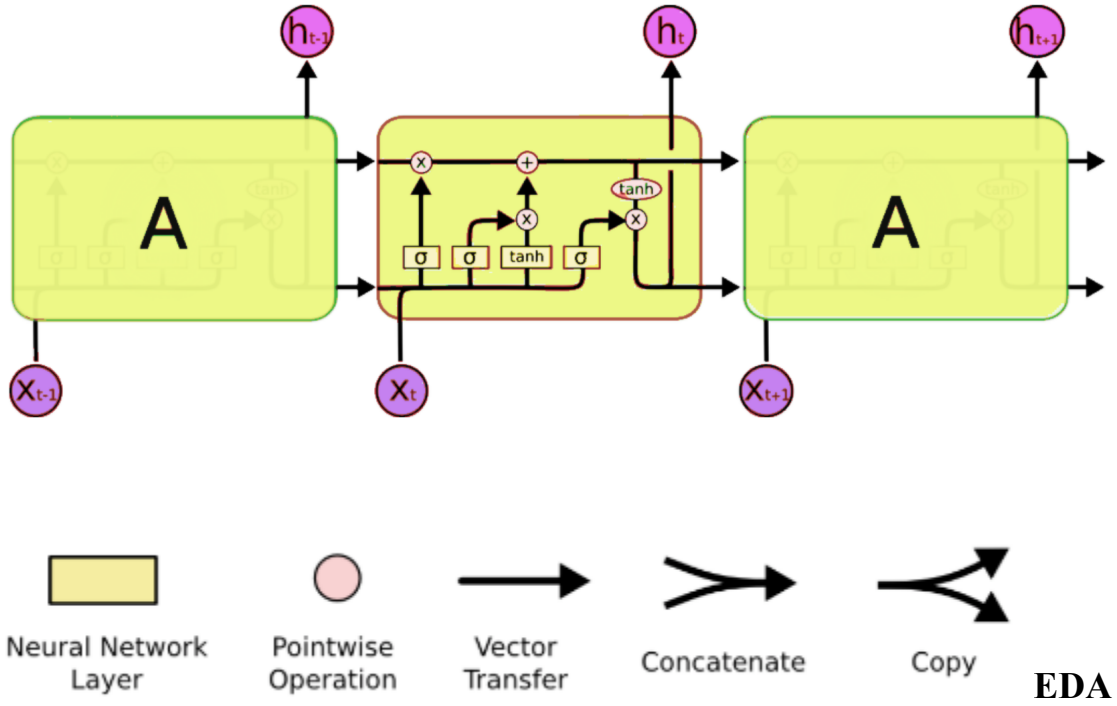


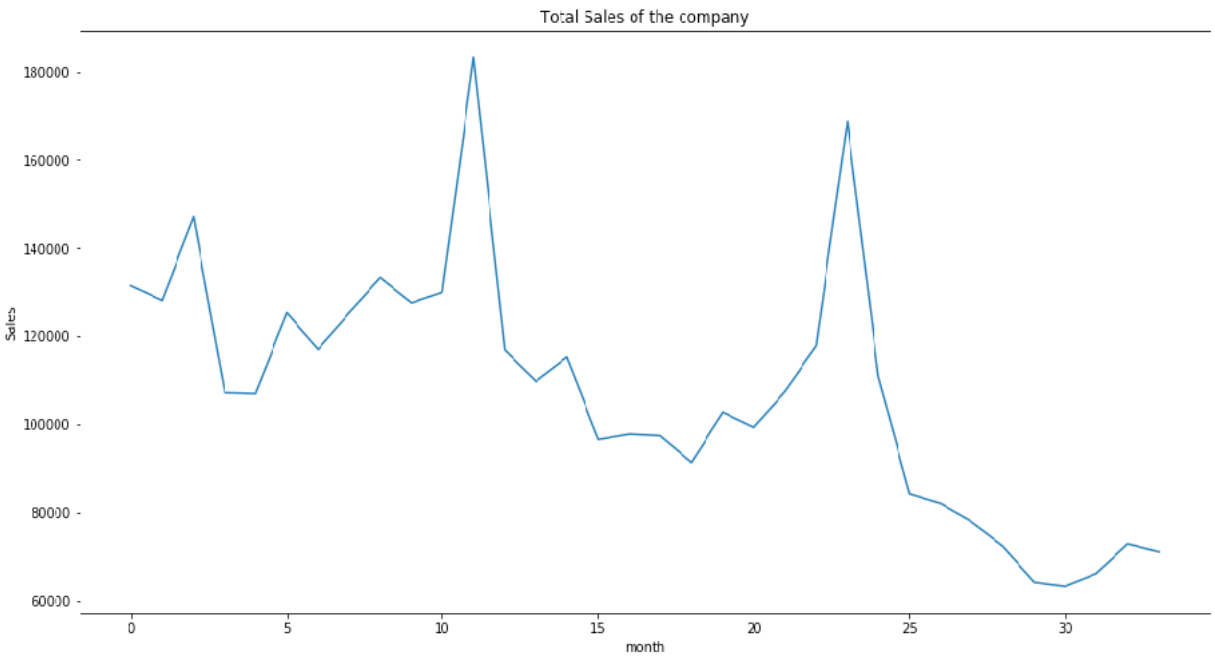
Figure from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Exploratory Data Analysis

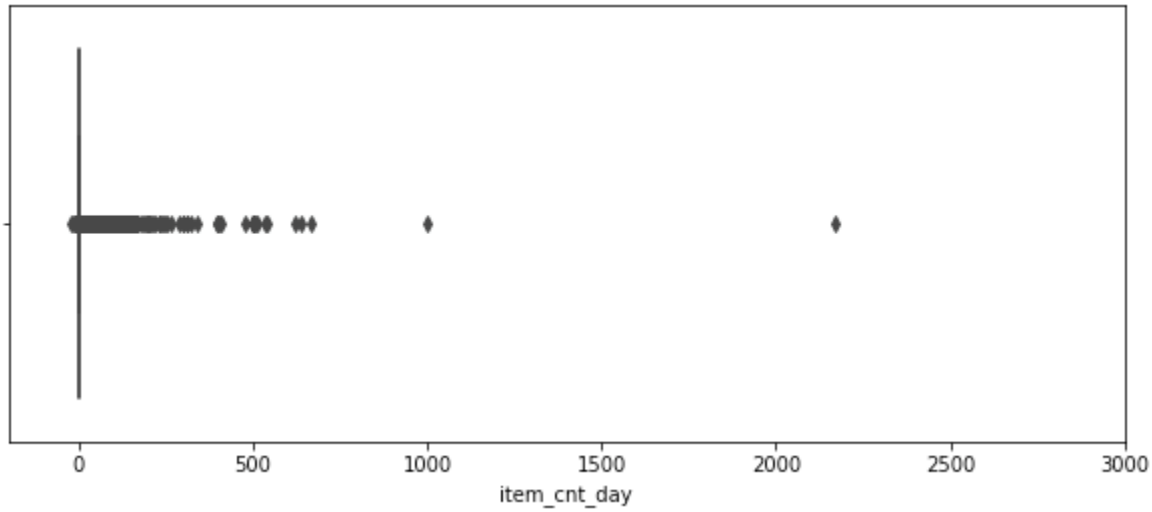
Here is how the data looks like. Item_cnt_day is the sales on that day for the particular shop and month. Each item is belonged to a category. By adding item_cnt_day would know the monthly sales of a product.

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_category_id	month
0	2013-01-02	0	59	22154	999.00	1.0	37	1
1	2013-01-03	0	25	2552	899.00	1.0	58	1
2	2013-01-05	0	25	2552	899.00	-1.0	58	1
3	2013-01-06	0	25	2554	1709.05	1.0	58	1
4	2013-01-15	0	25	2555	1099.00	1.0	56	1

The figure below shows the total sales of the company by month. We do see there is a seasonality and in general it is decreasing.



Check outliers. There are 1 item has more than 2000 sales on one day. Remove the outlier for a better modelling. Also there is no missing values.



Data Preprocessing

There are 2 csv file, the test.csv is the one we need to predict. I first check the shape of the sales_train.csv There are in total 2,935,849 rows. However, not all training set is used in testing csv, so I first merge them to get those products in the testing set. Thus there are in total 214,200 products.

As LSTM uses sigmoid and tanh function in the cell, a scalar to [0,1] is needed. LSTM is very sensitive to this. Since this is a Kaggle competition, the testing set does not have the target for us to compare. So firstly, I use the last month '2015-10' as the target to predict, thus we have data to evaluate.

Also using training and testing split with 70%. Training set has 149,940 products and testing set has 64,260 products.

LSTM only takes an 3D array as input, the input size should be [samples, time steps, features]. Time steps means how many step you want to look back to predict. As in this capstone, I tried LSTM on only one product, and on the whole dataset. There are different data preprocessing steps.

Using LSTM for one product, I need to recreate a dataset to have each row with one target and several look back columns. Column goes like t-2, t-1, t, t+1 (the one we predict).

t-3	t-2	t-1	t
2	3	1	2
3	1	2	5
1	2	5	2
2	5	2	4

For all the products, since we are looking back for 33 time steps, there is no extra recreate but only using pivot table to make each month a column.

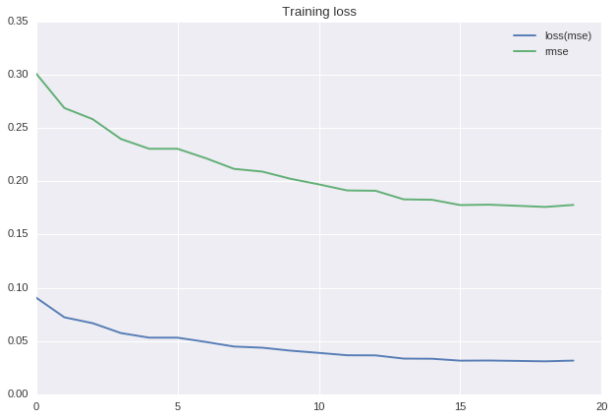
LSTM for one product

Firstly, I only try one product to use see how LSTM performs. The experiment result below is base on shop_id = 5 , item_id = 15238.

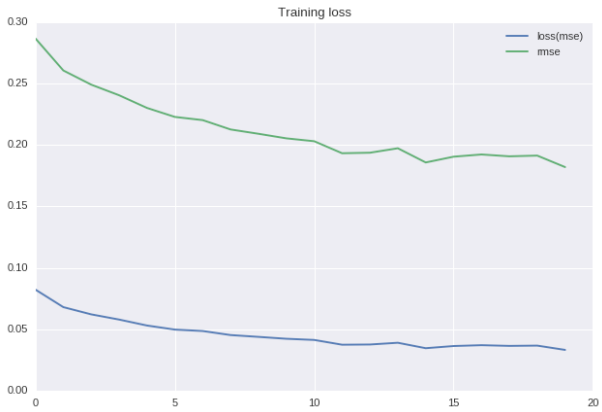
```
def model_1(): #add lookback
    model = Sequential()
    model.add(LSTM(128, input_shape=(1,look_back)))
    model.add(Dropout(0.2))

    model.add(Dense(1))
    adam = optimizers.Adam()
    model.compile(loss='mse', optimizer=adam, metrics=['mean_squared_error'])

    # model.summary()

    return model

model1 = model_1()

#record training time
start=time.time()
training1=model1.fit(X_train, Y_train, batch_size = 1, epochs = 20, shuffle=False)
end=time.time()
print("-----------------------------")
print("Total training time (seconds)", end-start)
```

This step is test the LSTM model. Also see how time steps would effect on this model. I compare 3 different window size from 1 to 5 and below is the training loss graphs for these three circumstances.

Look_back =1



Look_back = 3

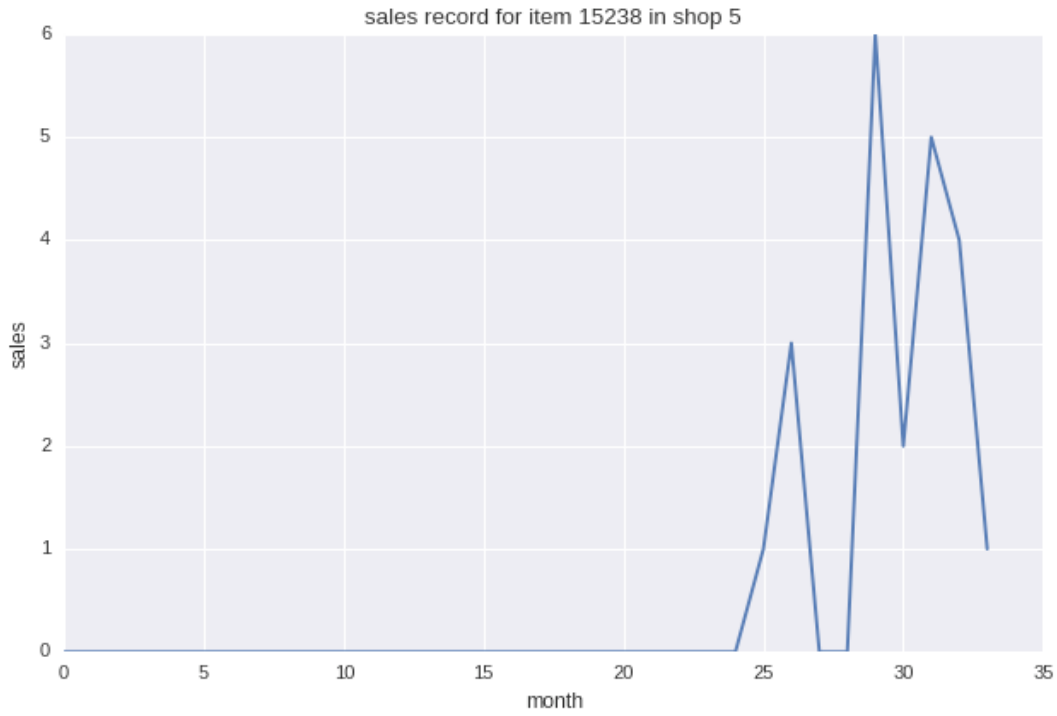


Look_back = 5

	Look back = 1	Look back = 3	Look back = 5
RMSE on training	0.0652	0.0362	0.0316
RMSE on testing	0.0463	0.2806	0.6994
Computation Time(s)	7.6529	7.1276	7.2821

From this simple example, we see that different window size could result in a quite different result. When look back 5 windows, the training result performs good but not on the testing data. In this experiment, all other parameters are the same.

Below is the sales record for this particular item. We see that for the first 24 months there is no sale for this item, it could probably because it is a new product. But it also shows a long look back for this product may not be a good choice.



Use LSTM to predict all products

To predict all products to reach our goal, use LSTM is a good choice.

Here we need to know what is the difference between stateful and stateless. So in stateless mode, LSTM updates parameters on batch 1. When go to batch 2, it will initial hidden state and cell state(usually zero) in the next state. Hidden state is the state of neurons, which is similar as hidden layers in neural network. While in stateful cases, it uses batch 1's last output hidden states and cell states as initial states for batch 2.

- **Stateless**

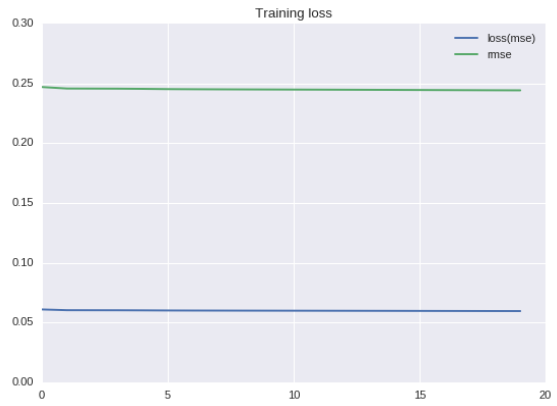
```
batch_size = 1260
look_back = 33
def model_1(): #change neurons
    model = Sequential()
    model.add(LSTM(64, batch_input_shape=(batch_size, x_train.shape[1], x_train.shape[2]),
                   stateful = False))
    model.add(Dense(1))
    adam = optimizers.Adam()
    model.compile(loss='mse', optimizer=adam, metrics=['mean_squared_error'])

    return model

model1 = model_1()

#record training time
start=time.time()
training1=model1.fit(x_train, y_train, batch_size = 1260, epochs = 20, shuffle=False)
end=time.time()
print("-------------------------")
print("Total training time (seconds)", end-start)

# make predictions
trainPredict = model1.predict(x_train, batch_size = 1260)
testPredict = model1.predict(x_test,batch_size = 1260)

#test error
test_rmse = model1.evaluate(x_test, y_test,batch_size = 1260)
print("-------------------------")
print('RMSE' , test_rmse[0])
```

- **Stateful**

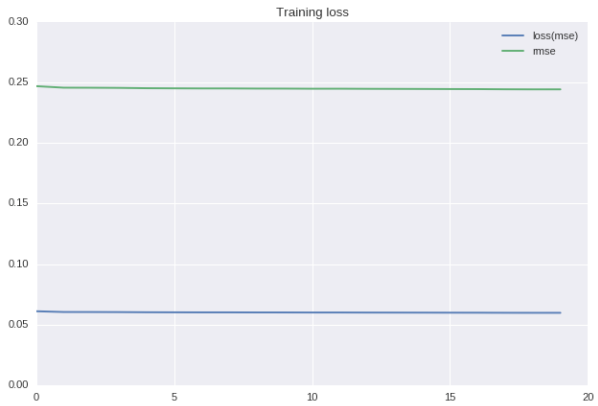
```
batch_size = 1260
look_back = 33
start=time.time()
loss=[]

model = Sequential()
model.add(LSTM(64, batch_input_shape=(batch_size, x_train.shape[1], x_train.shape[2]),
                   stateful = True))
model.add(Dense(1))
adam = optimizers.Adam()
model.compile(loss='mse', optimizer=adam, metrics=['mean_squared_error'])
    # model.summary()
for i in range(20): #num of epochs
    training = model.fit(x_train, y_train, epochs=1, batch_size=batch_size,
                         verbose=0, shuffle=False)
    loss.append(training.history['loss'])
    model.reset_states()

    # make predictions
trainPredict = model.predict(x_train, batch_size=1260)
testPredict = model.predict(x_test, batch_size=1260)
```

In this project, I try to compare stateful model and stateless model, as well as shuffling data.

	Hidden state	Batch size	stateful	shuffle	RMSE on test	Computation time
Model 1	32	1260	False	False	0.05941	88.8275
Model 2	64	1260	True	False	0.05945	93.122
Model 3	64	1260	True	True	0.05953	95.9197
Model 4	64	1260	False	False	0.05948	96.5655



Model 4 Stateless training loss



Model 2 Stateful training loss

Conclusion

From multiple experiments above, I learned how LSTM works in Keras, Different time steps would result in different computation time and RMSE. A choose of time steps is according to the original dataset. Also comparing stateful and stateless models shows that there is not a big difference in this data. It might because of the dataset here is very large and contains many zero.

Reference

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://www.kaggle.com/bapanes/sales-prediction-time-series-lstm-bapanes

https://www.kaggle.com/john850512/predict-future-sales-lstm

https://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

http://philipperemy.github.io/keras-stateful-lstm/

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://en.wikipedia.org/wiki/Long_short-term_memory