Competition: https://www.kaggle.com/c/competitive-data-science-predict-future-sales/overview (https://www.kaggle.com/c/competitive-data-science-predict-future-sales/overview)

Table of Contents

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from tensorflow.keras.regularizers import L1L2
from tensorflow.keras.callbacks import EarlyStopping
import csv
```

In [2]:

```
sales_train = pd.read_csv("sales_train.csv")
items = pd.read_csv("items.csv")
item_categories = pd.read_csv('item_categories.csv')
shops = pd.read_csv("shops.csv")

test = pd.read_csv('test.csv')
sample_submission = pd.read_csv('sample_submission.csv')
```

In [3]:

```
print('Train data')
display(sales_train.head(2))
display(sales_train.shape)
print('Test data')
display(test.head(2))
display(test.shape)
```

Train data

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	02.01.2013	0	59	22154	999.0	1.0
1	03.01.2013	0	25	2552	899.0	1.0

(2935849, 6)

Test data

	ID	shop_id	item_id
0	0	5	5037
1	1	5	5320

(214200, 3)

1. Data preprocessing

1.1 Анализ значений в наборе данных

In [4]:

```
sales_train.describe()
```

Out[4]:

	date_block_num	shop_id	item_id	item_price	item_cnt_day
count	2.935849e+06	2.935849e+06	2.935849e+06	2.935849e+06	2.935849e+06
mean	1.456991e+01	3.300173e+01	1.019723e+04	8.908532e+02	1.242641e+00
std	9.422988e+00	1.622697e+01	6.324297e+03	1.729800e+03	2.618834e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	-1.000000e+00	-2.200000e+01
25%	7.000000e+00	2.200000e+01	4.476000e+03	2.490000e+02	1.000000e+00
50%	1.400000e+01	3.100000e+01	9.343000e+03	3.990000e+02	1.000000e+00
75%	2.300000e+01	4.700000e+01	1.568400e+04	9.990000e+02	1.000000e+00
max	3.300000e+01	5.900000e+01	2.216900e+04	3.079800e+05	2.169000e+03

Заметим, что в наборе данных присутствуют отрицательные цены на товары и отрицательные объемы продаж.

Наблюдения с отрицательным "item_cnt_day"

In [5]:

```
sales_train[sales_train.item_cnt_day < 0].shape
```

Out[5]:

```
(7356, 6)
```

In [6]:

```
sales_train[sales_train.item_cnt_day < 0].item_cnt_day.value_counts()
```

Out[6]:

```
-1.0     7252
-2.0       78
-3.0       14
-5.0        4
-4.0        3
-6.0        2
-9.0        1
-16.0       1
-22.0       1
Name: item_cnt_day, dtype: int64
```

In [7]:

```
sales_train[sales_train.item_cnt_day == 1].item_cnt_day.value_counts()
```

Out[7]:

```
1.0    2629372
Name: item_cnt_day, dtype: int64
```

Отрицательные продажи - ошибка со знаком или нет?
Тест различных преобразований отрицательных значений показал, что среди:

- замены знака на положительный;
- зануления

лучший score получаем при занулении

In [8]:

```
sales_train.loc[sales_train[sales_train.item_cnt_day < 0].index, "item_cnt_day"] =
```

Наблюдения с отрицательным "item_price"

In [9]:

```
temp = sales_train[sales_train.item_price < 0]
temp
```

Out[9]:

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
484683	15.05.2013	4	32	2973	-1.0	1.0

In [10]:

```
test[(test.shop_id == temp.shop_id.values[0]) & (test.item_id == temp.item_id.value
```

Out[10]:

```
(0, 3)
```

В тестовом наборе нет наблюдения с shop_id = 32 & item_id = 2973. Удалим все наблюдения с такой парой индексов.

In [11]:

```
sales_train.drop([484683], inplace=True)
```

1.2 Data formating

train data (= sales_train) должны преобразовать так, чтобы индексами были shop_id & item_id. Поскольку стоит задача прогноза количества проданных экземпляров каждого продукта в каждом магазине за 34-ый месяц, посчитаем в каждом месяце общее число проданных экземпляров каждого продукта в каждом магазине. Для этого воспользуемся сводными таблицами `pd.dataframe.pivot_table()`

In [12]:

```
dataset = sales_train.pivot_table(index = ['shop_id','item_id'],values = ['item_cnt
display(dataset.head())
dataset.shape
```

		item_cnt_day																		
	date_block_num	0	1	2	3	4	5	6	7	8	9	...	24	25	26	27	28	29	30	31
shop_id	item_id																			
0	30	0	31	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
	31	0	11	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
	32	6	10	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
	33	3	3	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
	35	1	14	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

5 rows × 34 columns

Out[12]:

```
(424124, 34)
```

In [13]:

```
for i in range(len(dataset.columns)):
    dataset[dataset.columns[i]] = dataset[dataset.columns[i]].apply(lambda x: 0 if
```

Для обучения NN отберём только те наблюдения, пара индексов которых - shop_id & item_id, есть в тестовом наборе данных.

In [14]:

```
dataset = pd.merge(test,dataset,on = ['item_id','shop_id'],how = 'left')
dataset.fillna(0,inplace = True)
dataset_for_test = dataset.drop(['ID'], axis = 1)
dataset = dataset.drop(['shop_id','item_id','ID'], axis = 1)
```

```
/home/user/anaconda3/lib/python3.8/site-packages/pandas/core/reshape/m
erge.py:618: UserWarning: merging between different levels can give an
unintended result (1 levels on the left, 2 on the right)
  warnings.warn(msg, UserWarning)
```

с data scaling результаты хуже. Возможно при scaling data стоило подобрать другой learning_rate

In [15]:

```
def data_reshaping(x_train, y_train):
    x_train = x_train.astype("float32")# / dataset.values.max()
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1],  1))
    y_train = y_train.astype("float32")# / dataset.values.max()
    return(x_train, y_train)

def data_separation(x, target, with_validation=True):
    if(with_validation):
        x_train, x_val, y_train, y_val = train_test_split(x, target, test_size=0.2,
                                                          random_state=42, shuffle
        right_number_batches_train = (x_train.shape[0] // batch_size) * batch_size
        right_number_batches_val = (x_val.shape[0] // batch_size) * batch_size
        x_train, x_val, y_train, y_val = x_train[:right_number_batches_train], x_va
        return(x_train, x_val, y_train, y_val)
    else:
        x_train, y_train = x, target
        right_number_batches_train = (x_train.shape[0] // batch_size) * batch_size
        x_train, y_train = x_train[:right_number_batches_train], y_train[:right_num
        return(x_train, y_train)

def data_reshaping_test(x_test):
    x_test = x_test.astype("float32")# / dataset.values.max()
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],  1))
    return(x_test)
```

2. LSTM

2.1 Callbacks

In [16]:

```
'''
early_stopping_callback have a bug. To get rid of it, create own class, which inher
taken from [ https://github.com/tensorflow/tensorflow/issues/35634#issuecomment-665
 to get rid of bug when early_stopping_callback doesn't return best weights.
'''
class ReturnBestEarlyStopping(EarlyStopping):
    def __init__(self, **kwargs):
        super(ReturnBestEarlyStopping, self).__init__(**kwargs)

    def on_train_end(self, logs=None):
        if self.stopped_epoch > 0:
            if self.verbose > 0:
                print(f'\nEpoch {self.stopped_epoch + 1}: early stopping')
        elif self.restore_best_weights:
            if self.verbose > 0:
                print('Restoring model weights from the end of the best epoch.')
            self.model.set_weights(self.best_weights)

class LossAndErrorPrintingCallback(tf.keras.callbacks.Callback):

    def on_test_end(self, logs=None):
        print('Validation | Average losses: {:.3e}. MSE {:.1e}.'.format(logs['loss'

    def on_epoch_end(self, epoch, logs=None):#Вызывается в конце эпохи во время ОБУ
        print('Training | Average losses: {:.3e}. MSE {:.1e}.'.format(logs['loss'],

save_callback = tf.keras.callbacks.ModelCheckpoint(
    "checkpoint/checkpoint-{epoch:02d}", save_weights_only=False, monitor="loss", s
)

lr_scheduler = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="loss", factor=0.7, patience=7, mode="max", verbose=1, min_lr=0.0001, c
)
tensorboard_callback = keras.callbacks.TensorBoard(
    log_dir="tb_callback_dir", histogram_freq=1,
)
```

2.2 Model

In [17]:

```
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)

def LSTM(batch_size, regularization_set, dropout, learning_rate):
    model = keras.Sequential()
    model.add(keras.Input(batch_input_shape=(batch_size, time_steps, 1)))

    model.add(
        layers.LSTM(64, return_sequences=False, activation="tanh",stateful=False, d
    )
    model.add(layers.Dense(1))

    model.compile(
        loss='mse',
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        metrics=["mse"],
    )
    return(model)

def model_fitting(x_train, x_val, y_train, y_val, regularization_set,
                  learning_rate, batch_size, dropout, n_epoch=10, with_validation=T

    model = LSTM(batch_size, regularization_set, dropout, learning_rate)

    if(with_validation):
        model.fit(x_train, y_train, batch_size=batch_size , epochs=n_epoch, verbose
                  validation_data=(x_val, y_val),
                  callbacks=[LossAndErrorPrintingCallback(),
                             ReturnBestEarlyStopping(monitor='mse', min_delta=0, pa
    else:
        model.fit(x_train, y_train, batch_size=batch_size , epochs=30, verbose=1, s
                 callbacks=[tf.keras.callbacks.ModelCheckpoint(monitor='loss', verb

    return(model)
```

2.3 Custom Grid Search with cross-validation

In [18]:

```
def cross_validation_separation(k_fold):
    x_val = x[k_fold*right_number_batches_val : (k_fold+1)*right_number_batches_val
    y_val = target[k_fold*right_number_batches_val : (k_fold+1)*right_number_batche

    x_train = np.delete(x, range(k_fold*right_number_batches_val, (k_fold+1)*right_
    x_train = x_train[:right_number_batches_train]
    y_train = np.delete(target, range(k_fold*right_number_batches_val, (k_fold+1)*r
    y_train = y_train[:right_number_batches_train]
    return(x_train, x_val, y_train, y_val)

def get_mse_score(model, batch_size, x, y):
    output = model.predict(x, batch_size=batch_size)
    mse_score = mean_squared_error(y, output)
    return(mse_score)

def cross_validation(k_folds, reg, lr, batch_size, dropout, n_epoch=10):
    validation_mse_list, train_mse_list = list(), list()
    for k_fold in range(k_folds):
        print(f"k_fold: {k_fold+1}/{k_folds}")
        x_train, x_val, y_train, y_val = cross_validation_separation(k_fold)
        model = model_fitting(x_train, x_val, y_train, y_val, regularization_set=re
                              learning_rate=lr, batch_size=batch_size, dropout=drop

        validation_mse = get_mse_score(model, batch_size, x_val, y_val)
        validation_mse_list.append(validation_mse)
        train_mse = get_mse_score(model, batch_size, x_train, y_train)
        train_mse_list.append(train_mse)
    return(validation_mse_list, train_mse_list)
```

In [ ]:

```
regularization_weights = [L1L2(l1=0.0, l2=0.0), L1L2(l1=0.01, l2=0.0), L1L2(l1=0.00
batch_size_list = [1000, 2000]
learning_rates = [0.0005, 0.001]
dropout_list = [0.0, 0.2, 0.4]

time_steps = 33
grid_search_dict = {}
model_list = list()

k_folds = 5
target = dataset.values[:,-1]
x = dataset.values[:,:-1]
x, target = data_reshaping(x, target)

for batch_size in batch_size_list:
    print(f"\n\n\n ----------- batch_size: {batch_size} -----------")
    right_number_batches_train = int(( (dataset.shape[0]*(k_folds-1)/k_folds) // ba
    right_number_batches_val = int( ( (dataset.shape[0]*(1)/k_folds) // batch_size)
    for lr in learning_rates:
        for reg in regularization_weights:
            for dropout in dropout_list:
                reg_key = (f'batch_size {batch_size}, learning_rate {lr}, dropout {

                validation_mse_list, train_mse_list = cross_validation(k_folds, reg
                validation_mse_mean = np.mean(validation_mse_list)
                train_mse_mean = np.mean(train_mse_list)
                print(f"validation_mse_mean: {round(validation_mse_mean,3)}. train_
                grid_search_dict[reg_key] = f"val_mse_mean:{validation_mse_mean} &
```

In [ ]:

```
def save_dict_to_file(dic, name):
    f = open(f'{name}.txt','w+')
    f.write(str(dic))
    f.close()

def load_dict_from_file(name):
    f = open(f'grid_search_result/{name}.txt','r')
    data=f.read()
    f.close()
    return eval(data)

def save_dict_in_csv(name):
    dict_1 = load_dict_from_file(name)
    with open(f'{name}.csv', 'w+') as f:
        sorted_dict = dict(sorted(dict_1.items(), key = lambda x: x[1])[:])
        for key in sorted_dict.keys():
            f.write("%s,%s\n"%(key,sorted_dict[key]))

def update_dict(name_1, name_2):
    dict_1 = load_dict_from_file(name_1)
    dict_2 = load_dict_from_file(name_2)
    dict_1.update(dict_2)
    save_dict_to_file(dict_1, f'{name_1}_{name_2}')

def top_5_optimal_params(file_names):
    print('Top 5 sets model parameters with MSE score on validation data')
    for f_name in file_names:
        gs_dict = load_dict_from_file(f_name)
        display(dict(sorted(gs_dict.items(), key = lambda x: x[1])[:5]))

save_dict_to_file(grid_search_dict, 'GS_Cross_validation_not_full_WithOUT_zero_dele
```

2.4 Использование оптимальных параметров для обучения конечной модели

Cross-validation of model with optimal params

In [19]:

```
reg_optimal = L1L2(l1=0.0, l2=0.0)
batch_size_optimal = 2000
batch_size = batch_size_optimal
lr_optimal = 0.0005
dropout_optimal = 0.2

n_epoch = 7
k_folds = 5
time_steps = 33
target = dataset.values[:,-1]
x = dataset.values[:,:-1]
x, target = data_reshaping(x, target)

right_number_batches_train = int(( (dataset.shape[0]*(k_folds-1)/k_folds) // batch_
right_number_batches_val = int( ( (dataset.shape[0]*(1)/k_folds) // batch_size) * b


validation_mse_list, train_mse_list = cross_validation(k_folds, reg_optimal, lr_opt
print(f"validation_mse_list: {validation_mse_list}, \ntrain_mse_list: {train_mse_li
print(f"validation_mse_mean: {np.mean(validation_mse_list)}, \ntrain_mse_mean: {np.
```

```
k_fold: 1/5
Validation | Average losses: 1.293e+02. MSE 1.3e+02.
Training | Average losses: 6.385e+00. MSE 6.4e+00.
Validation | Average losses: 1.289e+02. MSE 1.3e+02.
Training | Average losses: 6.165e+00. MSE 6.2e+00.
Validation | Average losses: 1.286e+02. MSE 1.3e+02.
Training | Average losses: 6.055e+00. MSE 6.1e+00.
Validation | Average losses: 1.284e+02. MSE 1.3e+02.
Training | Average losses: 5.979e+00. MSE 6.0e+00.
Validation | Average losses: 1.282e+02. MSE 1.3e+02.
Training | Average losses: 5.918e+00. MSE 5.9e+00.
Validation | Average losses: 1.281e+02. MSE 1.3e+02.
Training | Average losses: 5.874e+00. MSE 5.9e+00.
Validation | Average losses: 1.280e+02. MSE 1.3e+02.
Training | Average losses: 5.817e+00. MSE 5.8e+00.
k_fold: 2/5
Validation | Average losses: 1.435e+01. MSE 1.4e+01.
Training | Average losses: 3.490e+01. MSE 3.5e+01.
Validation | Average losses: 1.405e+01. MSE 1.4e+01.
Training | Average losses: 3.471e+01. MSE 3.5e+01.
Validation | Average losses: 1.383e+01. MSE 1.4e+01.
Training | Average losses: 3.453e+01. MSE 3.5e+01.
Validation | Average losses: 1.369e+01. MSE 1.4e+01.
Training | Average losses: 3.437e+01. MSE 3.4e+01.
Validation | Average losses: 1.356e+01. MSE 1.4e+01.
Training | Average losses: 3.427e+01. MSE 3.4e+01.
Validation | Average losses: 1.352e+01. MSE 1.4e+01.
Training | Average losses: 3.420e+01. MSE 3.4e+01.
Validation | Average losses: 1.345e+01. MSE 1.3e+01.
Training | Average losses: 3.415e+01. MSE 3.4e+01.
k_fold: 3/5
Validation | Average losses: 7.307e+00. MSE 7.3e+00.
Training | Average losses: 3.680e+01. MSE 3.7e+01.
Validation | Average losses: 7.191e+00. MSE 7.2e+00.
Training | Average losses: 3.652e+01. MSE 3.7e+01.
Validation | Average losses: 7.099e+00. MSE 7.1e+00.
```

```
Training | Average losses: 3.634e+01. MSE 3.6e+01.
Validation | Average losses: 7.047e+00. MSE 7.0e+00.
Training | Average losses: 3.622e+01. MSE 3.6e+01.
Validation | Average losses: 7.025e+00. MSE 7.0e+00.
Training | Average losses: 3.616e+01. MSE 3.6e+01.
Validation | Average losses: 6.990e+00. MSE 7.0e+00.
Training | Average losses: 3.609e+01. MSE 3.6e+01.
Validation | Average losses: 6.948e+00. MSE 6.9e+00.
Training | Average losses: 3.602e+01. MSE 3.6e+01.
k_fold: 4/5
Validation | Average losses: 2.666e+00. MSE 2.7e+00.
Training | Average losses: 3.800e+01. MSE 3.8e+01.
Validation | Average losses: 2.479e+00. MSE 2.5e+00.
Training | Average losses: 3.767e+01. MSE 3.8e+01.
Validation | Average losses: 2.362e+00. MSE 2.4e+00.
Training | Average losses: 3.748e+01. MSE 3.7e+01.
Validation | Average losses: 2.267e+00. MSE 2.3e+00.
Training | Average losses: 3.727e+01. MSE 3.7e+01.
Validation | Average losses: 2.208e+00. MSE 2.2e+00.
Training | Average losses: 3.718e+01. MSE 3.7e+01.
Validation | Average losses: 2.188e+00. MSE 2.2e+00.
Training | Average losses: 3.713e+01. MSE 3.7e+01.
Validation | Average losses: 2.145e+00. MSE 2.1e+00.
Training | Average losses: 3.704e+01. MSE 3.7e+01.
k_fold: 5/5
Validation | Average losses: 1.431e+00. MSE 1.4e+00.
Training | Average losses: 3.824e+01. MSE 3.8e+01.
Validation | Average losses: 1.328e+00. MSE 1.3e+00.
Training | Average losses: 3.789e+01. MSE 3.8e+01.
Validation | Average losses: 1.281e+00. MSE 1.3e+00.
Training | Average losses: 3.762e+01. MSE 3.8e+01.
Validation | Average losses: 1.265e+00. MSE 1.3e+00.
Training | Average losses: 3.746e+01. MSE 3.7e+01.
Validation | Average losses: 1.252e+00. MSE 1.3e+00.
Training | Average losses: 3.736e+01. MSE 3.7e+01.
Validation | Average losses: 1.258e+00. MSE 1.3e+00.
Training | Average losses: 3.729e+01. MSE 3.7e+01.
Validation | Average losses: 1.227e+00. MSE 1.2e+00.
Training | Average losses: 3.725e+01. MSE 3.7e+01.
validation_mse_list: [128.01704, 13.449088, 6.9475803, 2.1449168, 1.22
71178],
train_mse_list: [5.7845006, 34.1463, 35.960766, 36.99511, 37.1905]
validation_mse_mean: 30.357147216796875,
train_mse_mean: 30.01543617248535
```

Заметим, что ошибка что на training data, что и на validation data почти монотонно уменьшается.
Создадим модель, которая будет предсказывать значения для тестового набора данных.
P.S.: в данном случае создаётся модель без валидационной выборки, поскольку на kaggle есть тестовый набор данных, на котором модель будет апробирована.

In [20]:

```
reg_optimal = L1L2(l1=0.0, l2=0.0)
batch_size_optimal = 2000
batch_size = batch_size_optimal
lr_optimal = 0.0005
dropout_optimal = 0.2
time_steps = 33


target = dataset.values[:,-1]
x = dataset.values[:,:-1]
x, target = data_reshaping(x, target)

x_train, y_train = data_separation(x, target, with_validation=False)

model = LSTM(batch_size, reg_optimal, dropout_optimal, lr_optimal)

model.fit(x_train, y_train, batch_size=batch_size , epochs=10, verbose=0, shuffle=T
          callbacks=[LossAndErrorPrintingCallback(), save_callback, tensorboard_cal

train_mse = get_mse_score(model, batch_size, x_train, y_train)
print(f"train MSE:{train_mse}")
#display(model.history.history)
```

```
WARNING:tensorflow:From /home/user/.local/lib/python3.8/site-packages/
tensorflow/python/ops/summary_ops_v2.py:1277: stop (from tensorflow.py
thon.eager.profiler) is deprecated and will be removed after 2020-07-0
1.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compa
red to the batch time (batch time: 0.0102s vs `on_train_batch_end` tim
e: 0.0274s). Check your callbacks.
Training | Average losses: 3.052e+01. MSE 3.1e+01.
WARNING:tensorflow:From /home/user/.local/lib/python3.8/site-packages/
tensorflow/python/training/tracking/tracking.py:111: Model.state_updat
es (from tensorflow.python.keras.engine.training) is deprecated and wi
ll be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are app
lied automatically.
WARNING:tensorflow:From /home/user/.local/lib/python3.8/site-packages/
tensorflow/python/training/tracking/tracking.py:111: Layer.updates (fr
om tensorflow.python.keras.engine.base_layer) is deprecated and will b
e removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are app
lied automatically.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-01/assets
Training | Average losses: 3.024e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-02/assets
Training | Average losses: 3.006e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-03/assets
Training | Average losses: 2.995e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-04/assets
Training | Average losses: 2.989e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-05/assets
Training | Average losses: 2.984e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-06/assets
Training | Average losses: 2.979e+01. MSE 3.0e+01.
```

```
INFO:tensorflow:Assets written to: checkpoint/checkpoint-07/assets
Training | Average losses: 2.971e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-08/assets
Training | Average losses: 2.965e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-09/assets
Training | Average losses: 2.960e+01. MSE 3.0e+01.
INFO:tensorflow:Assets written to: checkpoint/checkpoint-10/assets
train MSE:29.563417434692383
```

2.5 Prediction

In [21]:

```
n_batch = 700

reg_optimal = L1L2(l1=0.0, l2=0.0)
batch_size_optimal = 2000
batch_size = batch_size_optimal
lr_optimal = 0.0005
dropout_optimal = 0.2
time_steps = 33


target = dataset.values[:,-1]
x = dataset.values[:,:-1]
x, target = data_reshaping(x, target)

new_model = LSTM(n_batch, reg_optimal, dropout_optimal, lr_optimal)
new_model.load_weights(f'checkpoint/checkpoint-1.02548/variables/variables')

x_train, y_train = data_separation(x, target, with_validation=False)
predicted_values = new_model.predict(x_train, batch_size=batch_size).flatten()
train_mse = mean_squared_error(y_train, predicted_values)
print(f"MSE on training: {train_mse}")

def get_model_for_test(model, n_batch = 1):#if wanna use model to predict value, fo
    # re-define model
    new_model = LSTM(n_batch, reg_optimal, dropout_optimal, lr_optimal)
    # copy weights
    old_weights = model.get_weights()
    new_model.set_weights(old_weights)
    return(new_model)
```

```
WARNING:tensorflow:Model was constructed with shape (700, 33, 1) for i
nput Tensor("input_7:0", shape=(700, 33, 1), dtype=float32), but it wa
s called on an input with incompatible shape (2000, 33, 1).
MSE on training: 29.528867721557617
```

In [22]:

```
x_test = pd.merge(test,dataset_for_test,on = ['item_id','shop_id'],how = 'left')
print(x_test.shape)
x_test.fillna(0,inplace = True)
display(x_test.head())
test_id = x_test.pop('ID')
x_test.drop(['shop_id','item_id'], axis=1, inplace=True)
x_test.drop(x_test.columns[0], axis=1, inplace=True)
x_test.head()
```

(214200, 37)

	ID	shop_id	item_id	(item_cnt_day, 0)	(item_cnt_day, 1)	(item_cnt_day, 2)	(item_cnt_day, 3)	(item_cn
0	0	5	5037	0.0	0.0	0.0	0.0	
1	1	5	5320	0.0	0.0	0.0	0.0	
2	2	5	5233	0.0	0.0	0.0	0.0	
3	3	5	5232	0.0	0.0	0.0	0.0	
4	4	5	5268	0.0	0.0	0.0	0.0	

5 rows × 37 columns

Out[22]:

	(item_cnt_day, 1)	(item_cnt_day, 2)	(item_cnt_day, 3)	(item_cnt_day, 4)	(item_cnt_day, 5)	(item_cnt_day, 6)
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 33 columns

In [23]:

```
x_test = data_reshaping_test(x_test.values)
```

In [24]:

```
#new_model = get_model_for_test(model, n_batch)
predicted_values = new_model.predict(x_test, batch_size=n_batch).flatten()
submission = pd.DataFrame({'ID':test_id.values,'item_cnt_month':predicted_values})
submission.to_csv('LSTM_full.csv',index = False)
```

Kaggle result: Score: 1.02548 (50% в leaderboard [4818/9684])
[https://www.kaggle.com/konstantinlp/competitions (https://www.kaggle.com/konstantinlp/competitions)](https://www.kaggle.com/konstantinlp/competitions)