Фреймворк PyTorch для разработки искусственных нейронных сетей

Урок 5. Рекурентные сети для обработки последовательностей

Практическое задание

- Обучить GRU, LSTM для предсказания временного ряда на примере https://www.kaggle.com/c/favorita-grocery-sales-forecasting (https://www.kaggle.com/c/favorita-grocery-sales-forecasting) (для каждого типа продуктов)

Выполнил ***Соковнин ИЛ***

Набор данных и определение проблемы

B [ ]:
```
import torch
import torch.nn as nn

import seaborn as sns
import numpy as np  # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
%matplotlib inline
```

B [1]:
```
import os, sys

PATH_DATA = '.\\data\\'
print(PATH_DATA)
```

```
.\data\
```

B [ ]:
```
%%time

train = pd.read_csv(PATH_DATA + "train.csv", sep=",")
print(train.shape)
# train = train.set_index('id')
train.head(3)
```

B [ ]:
```
train.info()
```

B [3]:
```
# функция для оптимизации использования памяти
def reduce_memory(df):
    """Снижает размерности строк"""

    float_cols = df.select_dtypes(include=['float']).columns
    int_cols = df.select_dtypes(include=['int64']).columns

    df[float_cols] = df[float_cols].astype('float32')
    df[int_cols] = df[int_cols].astype('int32')

    return df
```

B [4]:
```
reduce_memory(train)
train.info()
```

B [ ]:
```
train_dt = train[['date', 'store_nbr', 'unit_sales']]
train_dt.info()
# train_dt.info(memory_usage='deep')
train_dt.head(3)
```

B [ ]:
```
%%time

train_dt.to_csv(PATH_DATA + 'train_dt.csv', index=False)
```

B [ ]:
```
# del (train_dt)
# del(train)

# Освобождаем память
import gc
gc.collect()
gc.collect() # два раза подряд, для надёжности
```

B [2]:
```
%%time

train_dt = pd.read_csv(PATH_DATA + "train_dt.csv", sep=",")
print(train_dt.shape)
train_dt.info()
train_dt.head(3)
```

```
(125497040, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125497040 entries, 0 to 125497039
Data columns (total 3 columns):
 #   Column      Dtype
---  ------      -----
 0   date        object
 1   store_nbr   int64
 2   unit_sales  float64
dtypes: float64(1), int64(1), object(1)
memory usage: 2.8+ GB
Wall time: 33.2 s
```

Out[2]:

	date	store_nbr	unit_sales
0	2013-01-01	25	7.0
1	2013-01-01	25	1.0
2	2013-01-01	25	2.0

Data processing

Subseting, join and aggregating data

B [6]:
```
stores = pd.read_csv(PATH_DATA + "stores.csv")
```

B [9]:
```
%%time

t=train_dt.groupby(['store_nbr','date'], as_index=False).agg({"unit_sales": "sum"})
train = pd.merge(t, stores, how='left', on=['store_nbr'])
mask=train['state']=='Pichincha'
train=train.loc[mask]
train=train.groupby(['date'], as_index=False).agg({"unit_sales": "sum"})
```

```
Wall time: 125 ms
```

Sales for Pichincha state (train data)

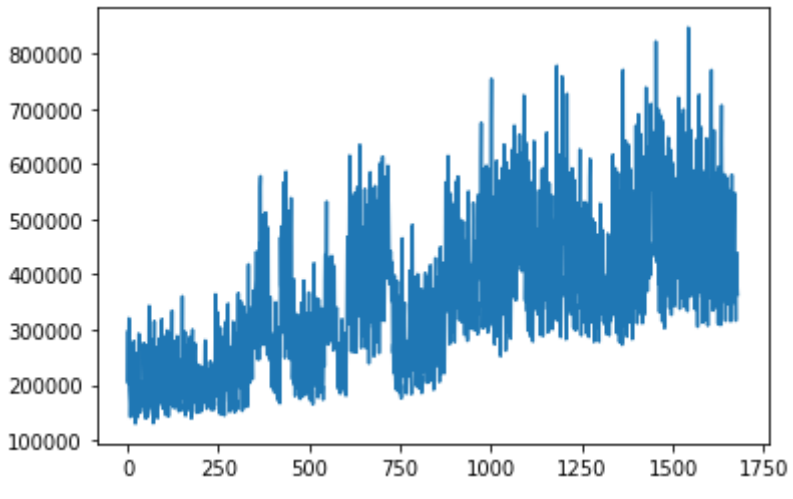
B [23]:
```
print(train.shape)
train.info()
train.tail(12)
```

```
(1679, 2)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1679 entries, 0 to 1678
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   date        1679 non-null   object
 1   unit_sales  1679 non-null   float32
dtypes: float32(1), object(1)
memory usage: 32.8+ KB
```

Out[23]:

	date	unit_sales
1667	2017-08-04	425713.15625
1668	2017-08-05	500614.56250
1669	2017-08-06	547592.12500
1670	2017-08-07	396073.43750
1671	2017-08-08	347288.25000
1672	2017-08-09	367813.87500
1673	2017-08-10	315940.31250
1674	2017-08-11	406504.40625
1675	2017-08-12	389945.59375
1676	2017-08-13	438714.53125
1677	2017-08-14	377126.43750
1678	2017-08-15	363382.59375

B [24]:
```
import matplotlib.pyplot as plt

plt.plot(train['unit_sales'])
plt.show()
```



There is a trend over time especially for the 2015 year, by 2015 and 2016 the slope gets lower and loses trend. It's easy to recognize how the sales increasing over the last quarters.

B [25]:
```
# Сохраняем результат в файл
train.to_csv(PATH_DATA + 'train_data.csv', index=False)
```

Задача состоит в том, чтобы предсказать продажи, k дней, исходя из первых n дней.

Решение - 1

При выполниении Д/З использовалась статья:
Прогнозирование временных рядов с LSTM в Python - https://pythobyte.com/time-series-prediction-using-lstm-with-pytorch-in-python-521ce3ed/ (https://pythobyte.com/time-series-prediction-using-lstm-with-pytorch-in-python-521ce3ed/)

Загрузка подготовленных данных

B [36]:
```
data_df = pd.read_csv(PATH_DATA + "train_data.csv", sep=",")
print(data_df.shape)
data_df.info()
data_df.head(3)
```

```
(1679, 2)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1679 entries, 0 to 1678
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   date        1679 non-null   object
 1   unit_sales  1679 non-null   float64
dtypes: float64(1), object(1)
memory usage: 26.4+ KB
```

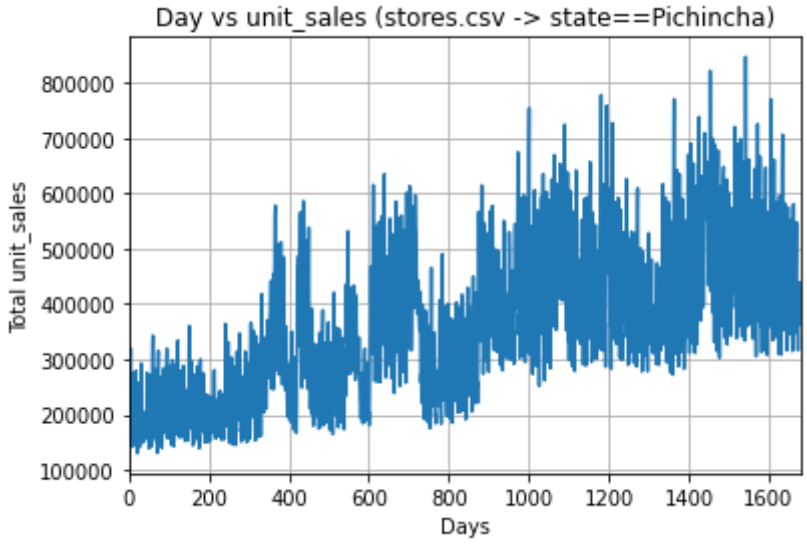
Out[36]:

	date	unit_sales
0	2013-01-02	295729.00
1	2013-01-03	203589.56
2	2013-01-04	203090.77

Задача состоит в том, чтобы предсказать продажи, k дней, исходя из первых n дней.

B [56]:
```
plt.title('Day vs unit_sales (stores.csv -> state==''Pichincha'')')
plt.ylabel('Total unit_sales')
plt.xlabel('Days')
plt.grid(True)
plt.autoscale(axis='x',tight=True)
plt.plot(data_df['unit_sales'])

# Период: [2013-01-02, 2017-08-15]
# stores.csv-> 'state'=='Pichincha'
```

Out[56]:
```
[<matplotlib.lines.Line2D at 0xd3d88faac0>]
```



Вывод показывает, что со временем среднее количество продаж, увеличилось. Количество продаж в течение года, колеблется.

B [38]:
```
data_df['date'].head(1), data_df['date'].tail(1)
```

Out[38]:
```
(0    2013-01-02
 Name: date, dtype: object,
 1678    2017-08-15
 Name: date, dtype: object)
```

B [39]:
```
data_df['unit_sales'].head(1),
```

Out[39]:
```
(0    295729.0
 Name: unit_sales, dtype: float64,)
```

B [40]:
```
data_df.shape
```

Out[40]:
```
(1679, 2)
```

B [41]:
```
data_df.columns
```

Out[41]:
```
Index(['date', 'unit_sales'], dtype='object')
```

B [45]:
```
all_data = data_df['unit_sales'].values.astype(float)
print(len(all_data))
print(all_data)
```

```
1679
[295729.   203589.56 203090.77 ... 438714.53 377126.44 363382.6 ]
```

Далее мы разделим наш набор данных на обучающие и тестовые наборы. Алгоритм LSTM будет обучаться на обучающем наборе. Затем модель будет использоваться для составления прогнозов на тестовом наборе. Прогнозы будут сравниваться с фактическими значениями в тестовом наборе для оценки производительности обученной модели.

Первые 1179 записи будут использоваться для обучения модели, а последние 500 записей будут использоваться в качестве тестового набора. Следующий сценарий делит данные на обучающие и тестовые наборы.

Рзделил на train и test

B [46]:
```
test_data_size = 500

train_data = all_data[:-test_data_size]
test_data = all_data[-test_data_size:]
print(len(train_data))
print(len(test_data))
```

```
1179
500
```

B [47]:
```
print(test_data)
```

```
[463464.44 662771.7  777635.6  432228.84 379148.5  408562.4  315344.3
 379869.22 525681.7  615886.7  361546.75 346025.97 365172.22 286234.47
 360473.7  485494.7  758259.1  733211.44 598701.94 574309.44 607436.5
 455092.47 566246.06 607025.3  360556.84 395416.12 367973.34 282274.84
 382455.6  588470.2  726919.8  576447.8  410174.12 436490.8  336640.72
 412194.03 583969.56 430678.62 407555.06 341508.56 396695.16 327221.53
 357535.9  505352.3  591819.7  392385.44 334703.4  376803.2  302710.72
 383055.9  492384.97 570427.56 371079.28 298319.9  343448.9  300307.38
 467630.97 434441.28 529404.56 405086.6  505375.12 438254.8  366658.94
 405301.44 548177.75 626164.5  398302.34 339516.12 379247.56 289516.16
 352132.06 456281.03 516675.97 373675.47 312857.   375318.47 323141.56
 418650.4  530918.4  448298.   384105.88 339030.47 362372.9  293393.9
 350778.4  470979.47 527770.75 374150.75 304862.3  340076.1  286655.62
 461741.25 572743.44 608798.56 466160.44 377913.47 410742.7  313421.22
 383011.3  465985.66 500112.28 359535.84 320359.03 347832.47 277695.56
 361476.   468647.   493292.88 362288.12 319602.28 365259.8  308444.06
 354514.62 432444.7  471421.72 354865.16 304890.97 349297.38 277917.6
 356431.6  473099.4  527954.8  466867.88 394504.8  419023.25 328737.88
 385499.25 453629.66 480067.   372139.97 314227.84 350280.4  310392.84
 392851.78 365451.3  396912.6  364645.88 369110.06 383001.78 291589.3
 360458.12 441714.9  473626.2  374471.38 334228.   357303.8  289464.56
 354701.28 419565.34 465837.   348012.9  315536.5  387690.44 358075.
 467590.12 586909.56 616427.25 441514.03 398499.22 417547.3  325860.97
 368710.3  522037.94 592226.5  359018.38 305653.3  341224.72 293501.22
 375220.56 505287.72 583639.44 362657.12 336675.56 368436.12 277358.06
 367296.   484823.44 549001.3  353791.1  304356.25 348458.47 272319.28
 387961.9  673548.5  770145.06 460402.16 381965.38 423668.34 309891.94
 390995.12 534073.5  641770.7  371162.5  310525.88 361760.53 287210.3
 363509.8  483937.22 634696.9  378508.62 319655.25 349989.8  313285.1
 380458.12 478035.62 590154.2  345732.6  319530.34 350665.06 282527.5
 352199.94 463279.6  550229.75 337697.75 404710.97 520685.6  423444.5
 484246.88 501860.66 669286.56 445955.06 402756.62 436370.12 311151.03
 389851.25 518677.66 689669.4  392302.12 349324.94 456319.97 329310.7
 392253.62 524196.44 653760.2  411857.78 344049.22 412258.53 311151.4
 397537.9  494591.25 612632.3  405711.72 326923.53 416376.1  443592.12
 451991.84 607862.5  738212.56 435581.78 558682.06 484160.   369419.75
 434574.6  513729.78 631380.1  432274.8  393365.56 456990.9  398098.25
 491535.38 588138.3  708790.56 553564.25 565971.5  630885.44 593527.75
 657560.44 615579.3  573048.1  434090.47 491901.9  450327.34 588584.75
 511673.2  821605.   593083.06 543055.9  421969.25 475042.25 635830.94
 699178.06 439697.3  383640.38 424036.25 331650.94 425414.9  576171.25
 687957.44 440458.38 374599.56 422154.56 314912.16 389002.44 614338.3
 677396.06 423388.75 367988.   396213.1  302443.12 396694.22 534258.94
 612420.75 360612.38 392637.56 518516.12 391475.6  461979.97 627573.94
 647873.3  424168.56 386743.12 437854.97 338544.06 420857.53 519171.56
 625173.   429363.56 326573.   395921.94 346731.53 414260.78 601157.7
 481698.84 392746.38 371873.8  389214.97 342609.3  412215.84 506520.84
 415810.72 427032.12 574030.56 528591.44 437234.75 486198.8  634886.4
 719913.7  428920.88 392718.1  414182.44 337546.1  427418.56 560114.9
 688801.4  405771.2  358424.9  435844.38 360524.06 444982.66 580663.2
 698361.94 440732.56 387534.6  414912.06 341132.47 421799.47 577746.25
 655279.2  414395.4  343433.16 435010.4  333400.88 479072.4  846761.25
 667628.3  425840.8  426987.44 465979.1  359403.88 410164.1  557645.9
 661129.25 383562.53 363347.75 390707.84 381376.3  391271.53 489505.72
 588035.75 403925.78 365425.62 413136.34 336920.56 418117.84 520669.4
 643719.56 376779.84 357462.38 389982.44 305131.8  410782.12 549925.25
 553516.56 724922.56 466016.6  480529.78 369266.84 419505.38 559246.44
 666372.7  412736.88 359063.25 398554.88 312685.2  396189.72 585819.94
 457783.03 404543.4  410219.22 430186.28 317481.28 407295.38 544834.3
 646224.56 395874.84 355708.9  384084.3  307008.34 508710.8  470003.12
 566144.1  383562.25 357776.4  419738.88 402582.66 499819.66 641629.75
 769796.9  479630.72 403969.7  450756.72 334530.47 420854.84 537741.4
 660609.25 387059.38 369275.25 401436.8  340075.72 450500.03 584338.9
 502295.06 409387.53 402469.4  401057.4  310034.2  402967.3  511104.44
 595126.9  373753.88 331420.5  377250.38 308123.25 423757.34 641649.8
 705943.   484996.62 421630.56 450327.84 352576.66 421154.5  521784.4
 581352.06 400255.25 351584.75 380746.44 315660.2  390896.06 495107.34
 574725.94 412239.56 358715.53 396156.7  346297.7  408199.94 482838.97
 552268.1  398815.53 343215.47 371557.16 314677.34 421051.9  516314.22
 580407.25 440220.3  478229.34 482223.4  365336.56 425713.16 500614.56
 547592.1  396073.44 347288.25 367813.88 315940.3  406504.4  389945.6
 438714.53 377126.44 363382.6 ]
```

Нормализуем данные с помощью скалера min/max с минимальными и максимальными значениями -1 и 1 соответственно.

B [48]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(-1, 1))
train_data_normalized = scaler.fit_transform(train_data .reshape(-1, 1))
```

B [49]:
```
print(train_data_normalized[:5])
print(train_data_normalized[-5:])
```

```
[[-0.46759538]
 [-0.76280631]
 [-0.76440442]
 [-0.51727788]
 [-0.38837831]]
[[ 0.17825205]
 [-0.29365632]
 [-0.39480955]
 [-0.28146193]
 [-0.34653452]]
```

Преобразуем наборы данных в тензоры для использования в модели PyTorch

B [50]:
```
train_data_normalized = torch.FloatTensor(train_data_normalized).view(-1)
```

Преобразование обучающих данных в последовательности и соответствующие метки.

B [51]:
```
train_window = 12
```

B [52]:
```
# Далее мы определим функцию с именем create_input_sequences .
# Функция примет необработанные входные данные и вернет список кортежей.
# В каждом кортеже первый элемент будет содержать список из 12 пунктов, соответствующих продажам, за 12 дней,
# второй элемент кортежа будет содержать один пункт, т. е. количество продаж за 12+1-й день.

def create_inout_sequences(input_data, tw):
    """Функция принимает необработанные входные данные и возвращает список кортежей."""

    inout_seq = []
    L = len(input_data)
    for i in range(L-tw):
        train_seq = input_data[i:i+tw]
        train_label = input_data[i+tw:i+tw+1]
        inout_seq.append((train_seq ,train_label))

    return inout_seq
```

B [57]:
```
#Создаём последовательности
train_inout_seq = create_inout_sequences(train_data_normalized, train_window)
```

B [58]:
```
train_inout_seq[:5]
```

Out[58]:
```
[(tensor([-0.4676, -0.7628, -0.7644, -0.5173, -0.3884, -0.8063, -0.8378, -0.8758,
          -0.9611, -0.8950, -0.6928, -0.5331]),
  tensor([-0.9013])),
 (tensor([-0.7628, -0.7644, -0.5173, -0.3884, -0.8063, -0.8378, -0.8758, -0.9611,
          -0.8950, -0.6928, -0.5331, -0.9013]),
  tensor([-0.9053])),
 (tensor([-0.7644, -0.5173, -0.3884, -0.8063, -0.8378, -0.8758, -0.9611, -0.8950,
          -0.6928, -0.5331, -0.9013, -0.9053]),
  tensor([-0.8320])),
 (tensor([-0.5173, -0.3884, -0.8063, -0.8378, -0.8758, -0.9611, -0.8950, -0.6928,
          -0.5331, -0.9013, -0.9053, -0.8320]),
  tensor([-0.9536])),
 (tensor([-0.3884, -0.8063, -0.8378, -0.8758, -0.9611, -0.8950, -0.6928, -0.5331,
          -0.9013, -0.9053, -0.8320, -0.9536]),
  tensor([-0.8745]))]
```

LSTM

- **input_size** : Соответствует количеству объектов во входных данных. Хотя длина нашей последовательности равна 12, для каждого дня у нас есть только 1 значение, то есть общее количество выручки, поэтому входной размер будет равен 1.
- **hidden_layer_size** : Определяет количество скрытых слоев вместе с количеством нейронов в каждом слое. У нас будет один слой из 100 нейронов.
- **output_size** : Количество элементов в выводе, так как мы хотим предсказать количество выручки на 1 день в будущем, размер вывода будет равен 1.

B [60]:
```
class LSTM(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=100, output_size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size

        self.lstm = nn.LSTM(input_size, hidden_layer_size)

        self.linear = nn.Linear(hidden_layer_size, output_size)

        self.hidden_cell = (torch.zeros(1,1,self.hidden_layer_size),
                            torch.zeros(1,1,self.hidden_layer_size))

    def forward(self, input_seq):
        lstm_out, self.hidden_cell = self.lstm(input_seq.view(len(input_seq) ,1, -1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]
```

B [61]:
```
model = LSTM()
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

B [62]:
```
print(model)
```

```
LSTM(
  (lstm): LSTM(1, 100)
  (linear): Linear(in_features=100, out_features=1, bias=True)
)
```

Обучение модели

Мы будем тренировать нашу модель в течение 300 эпох. Потери будут печататься через каждые 25 эпох.

B [99]:
```
epochs = 300

for i in range(epochs):
    for seq, labels in train_inout_seq:
        optimizer.zero_grad()
        model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                             torch.zeros(1, 1, model.hidden_layer_size))

        y_pred = model(seq)

        single_loss = loss_function(y_pred, labels)
        single_loss.backward()
        optimizer.step()

    if i%25 == 1:
        print(f'epoch: {i:3} loss: {single_loss.item():10.8f}')

print(f'epoch: {i:3} loss: {single_loss.item():10.10f}')
```

```
epoch:   1 loss: 0.01400300
epoch:  26 loss: 0.02697833
epoch:  51 loss: 0.01458593
epoch:  76 loss: 0.01115733
epoch: 101 loss: 0.00395531
epoch: 126 loss: 0.00441030
epoch: 151 loss: 0.00000258
epoch: 176 loss: 0.00243663
epoch: 201 loss: 0.00056983
epoch: 226 loss: 0.00014471
epoch: 251 loss: 0.00329403
epoch: 276 loss: 0.00000148
epoch: 299 loss: 0.0005294156
```

Прогнозы

Отфильтруем последние 10 значений из обучающего набора:

B [100]:
```
fut_pred = 300

test_inputs = train_data_normalized[-train_window:].tolist()
print(test_inputs)
```

```
[0.3236887753009796, -0.25877830386161804, -0.4524434804916382, -0.20583787560462952, -0.34753096103668213, -0.3208041191101074, 0.007369729224592447, 0.17825205624103546, -0.2936563193798065, -0.3948095440864563, -0.2814619243144989, -0.34653452038764954]
```

Составление прогнозов

B [101]:
```
model.eval()

for i in range(fut_pred):
    seq = torch.FloatTensor(test_inputs[-train_window:])
    with torch.no_grad():
        model.hidden = (torch.zeros(1, 1, model.hidden_layer_size),
                        torch.zeros(1, 1, model.hidden_layer_size))
        test_inputs.append(model(seq).item())
```

B [102]:
```
test_inputs[fut_pred:]
```

Out[102]:
```
[0.08698885142803192,
 0.3161541819572449,
 0.2977752685546875,
 0.15500690042972565,
 0.15066291391849518,
 0.07353302836418152,
 0.030370503664016724,
 0.06411997973918915,
 0.2871330976486206,
 0.33945566415786743,
 0.16767989099025726,
 0.1467825025320053]
```

Преобразовать нормализованные прогнозные значения в фактические прогнозные значения.

B [103]:
```
actual_predictions = scaler.inverse_transform(np.array(test_inputs[train_window:] ).reshape(-1, 1))
print(actual_predictions)
```

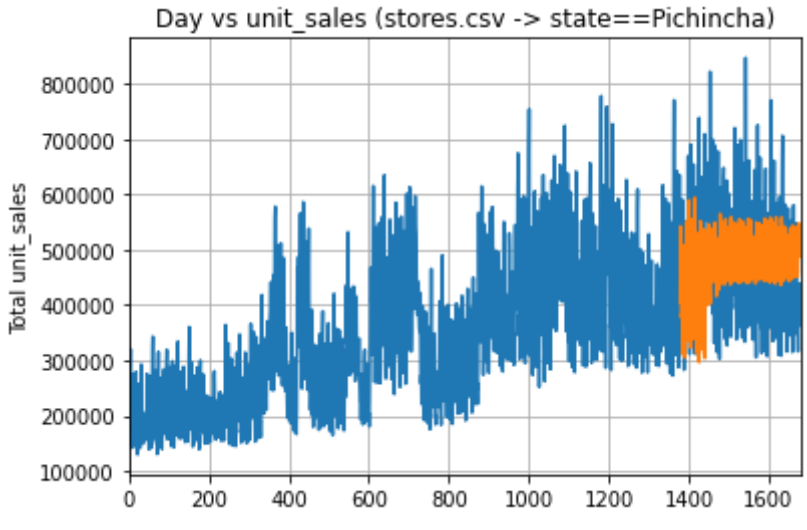
```
[447265.48845804]
[441644.34773358]
[481430.3693803 ]
[550819.68544409]
[514598.74778904]
[485124.44496737]
[474973.40404524]
[454630.69229014]
[444485.1765894 ]
[470823.97208096]
[541300.39870808]
[514585.22308886]
[471532.05617744]
[467920.36592013]
[447378.15088483]
[438963.95493598]
[479106.26034535]
[556026.84383975]
[512813.21761581]
[474572.99292522]
```

B [ ]:

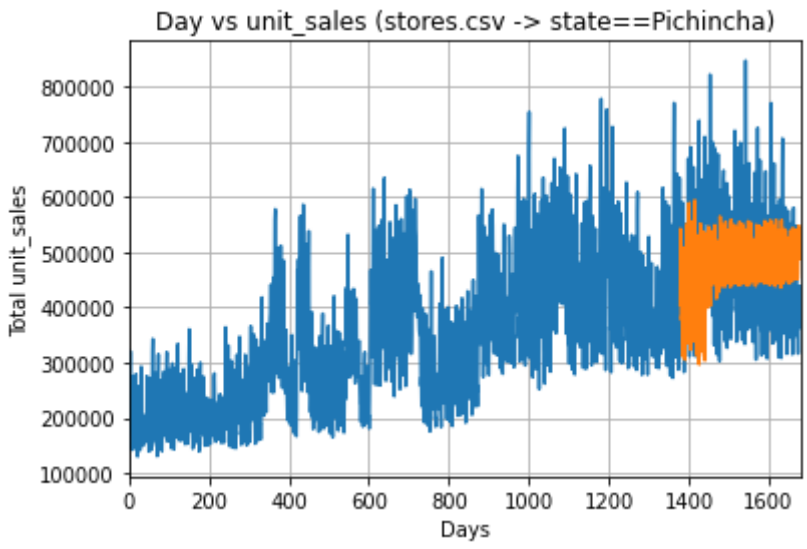
B [104]:
```
x = np.arange(1378, 1678, 1)
print(x)
```

```
[1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391
 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405
 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419
 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433
 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447
 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461
 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475
 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489
 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503
 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517
 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531
 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545
 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559
 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573
 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587
 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601
 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615
 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629
 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643
 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657
 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671
 1672 1673 1674 1675 1676 1677]
```

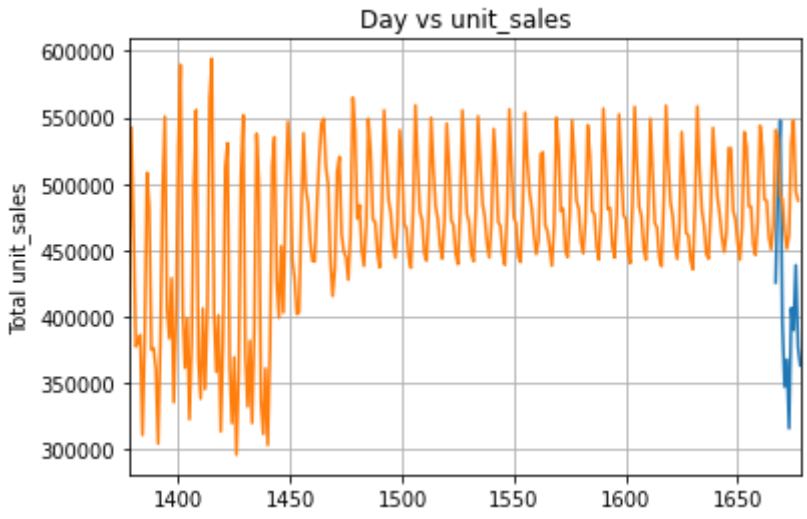
B [105]:
```
plt.title('Day vs unit_sales (stores.csv -> state==''Pichincha'')')
plt.ylabel('Total unit_sales')
plt.grid(True)
plt.autoscale(axis='x', tight=True)
plt.plot(data_df['unit_sales'])
plt.plot(x, actual_predictions)
plt.show()
```



B [106]:
```
plt.title('Day vs unit_sales (stores.csv -> state==''Pichincha'')')
plt.ylabel('Total unit_sales')
plt.xlabel('Days')
plt.grid(True)
plt.autoscale(axis='x',tight=True)
plt.plot(data_df['unit_sales'])
plt.plot(x,actual_predictions)
plt.show()
```



B [107]:
```
plt.title('Day vs unit_sales')
plt.ylabel('Total unit_sales')
plt.grid(True)
plt.autoscale(axis='x', tight=True)

plt.plot(data_df['unit_sales'][-train_window:])
plt.plot(x,actual_predictions)
plt.show()
```



Прогнозы не очень точны, но алгоритм смог уловить тенденцию

GRU (меняем LSTM на GRU как на лекции)

B [108]:
```
class GRU(nn.Module):
    def __init__(self, input_size=1, hidden_layer_size=100, output_size=1):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size
        self.gru = nn.GRU(input_size, hidden_layer_size)
        self.linear = nn.Linear(hidden_layer_size, output_size)
        self.hidden_cell = (torch.zeros(1,1,self.hidden_layer_size),
                            torch.zeros(1,1,self.hidden_layer_size))

    def forward(self, input_seq):
        gru_out, self.hidden_cell = self.gru(input_seq.view(len(input_seq) ,1, -1), self.hidden_cell)
        predictions = self.linear(gru_out.view(len(input_seq), -1))
        return predictions[-1]
```

B [109]:
```
model = GRU()
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

B [110]:
```
print(model)
```

```
GRU(
  (gru): GRU(1, 100)
  (linear): Linear(in_features=100, out_features=1, bias=True)
)
```

B [ ]:
```
epochs = 10

for i in range(epochs):
    for seq, labels in train_inout_seq:
        optimizer.zero_grad()
        model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                             torch.zeros(1, 1, model.hidden_layer_size))

        y_pred = model(seq)

        single_loss = loss_function(y_pred, labels)
        single_loss.backward()
        optimizer.step()

    if i%25 == 1:
        print(f'epoch: {i:3} loss: {single_loss.item():10.8f}')

print(f'epoch: {i:3} loss: {single_loss.item():10.10f}')
```

B [ ]:

B [ ]:

B [ ]:

B [ ]:

B [ ]:

B [ ]:

B [ ]:

B [ ]:

Решение - 2

B [ ]:
```
import numpy
import matplotlib.pyplot as plt
import pandas as pd
import math
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

Загрузка подготовленных данных

B [5]:
```
train = pd.read_csv(PATH_DATA + "train_data.csv", sep=",")
print(train.shape)
train.info()
train.head(3)
```

```
(1679, 2)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1679 entries, 0 to 1678
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   date        1679 non-null   object 
 1   unit_sales  1679 non-null   float64
dtypes: float64(1), object(1)
memory usage: 26.4+ KB
```

Out[5]:

	date	unit_sales
0	2013-01-02	295729.00
1	2013-01-03	203589.56
2	2013-01-04	203090.77

Разделение выборки для обучающих и тестовых данных

B [6]:
```
train_size = int(len(train) * 0.75)
test_size = len(train) - train_size

print(train_size,test_size, len(train))
```

```
1259 420 1679
```

B [7]:
```
train1= train[0:train_size]
test =  train[train_size:len(train)]
print(len(train1), len(test))
```

```
1259 420
```

B [8]:
```
train1=train1.set_index("date")
test=test.set_index("date")
train=train.set_index("date")
train1=train1.values
test=test.values
train=train.values
```

```
Determine the number of previous time steps to use as input variables to predict the next time period. In this case
(look_back) determinated to 1
```

B [9]:
```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

B [11]:
```
# import numpy

look_back = 1
trainX, trainY = create_dataset(train1, look_back)
testX, testY = create_dataset(test, look_back)
```

B [19]:
```
trainX.shape, trainY.shape
```

Out[19]:
```
((1257, 1), (1257,))
```

```
Multilayer Perceptron model
A simple network with 1 input (look_back) , 1 hidden layer with 8 neurons and one (1) output layer.
```

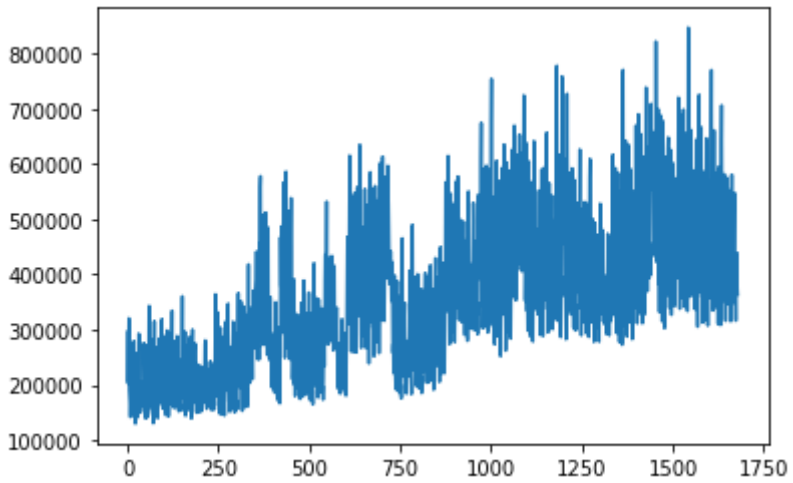
B [14]:
```
model = Sequential()
model.add(Dense(8, input_dim=look_back, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=200, batch_size=2, verbose=2)
```

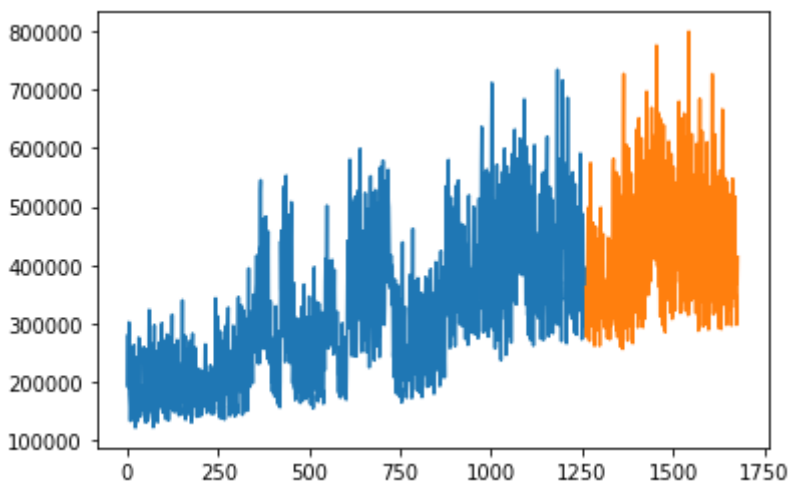
```
Epoch 1/200
629/629 - 5s - loss: 14533445632.0000 - 5s/epoch - 7ms/step
Epoch 2/200
629/629 - 2s - loss: 7524797440.0000 - 2s/epoch - 2ms/step
Epoch 3/200
629/629 - 1s - loss: 7560857088.0000 - 1s/epoch - 2ms/step
Epoch 4/200
629/629 - 1s - loss: 7531424256.0000 - 1s/epoch - 2ms/step
Epoch 5/200
629/629 - 2s - loss: 7566036480.0000 - 2s/epoch - 2ms/step
Epoch 6/200
629/629 - 2s - loss: 7550424576.0000 - 2s/epoch - 2ms/step
Epoch 7/200
629/629 - 2s - loss: 7579824128.0000 - 2s/epoch - 2ms/step
Epoch 8/200
629/629 - 2s - loss: 7572335616.0000 - 2s/epoch - 2ms/step
Epoch 9/200
629/629 - 1s - loss: 7573846016.0000 - 1s/epoch - 2ms/step
Epoch 10/200
```

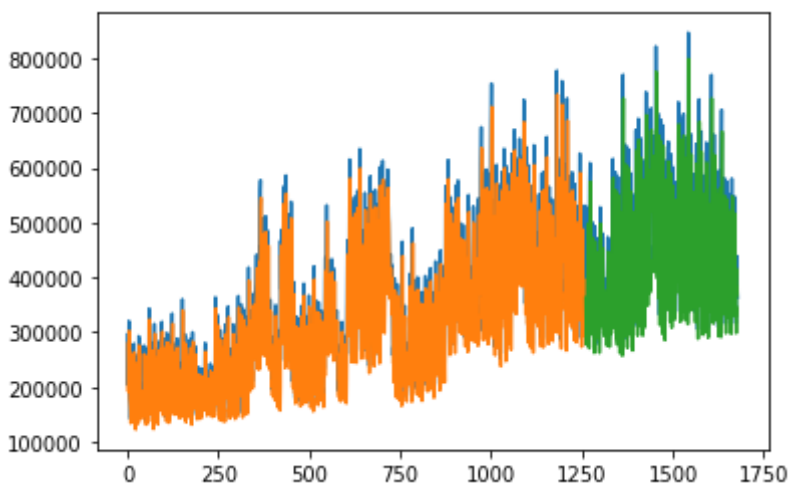
B [16]:
```
trainScore = model.evaluate(trainX, trainY, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore, math.sqrt(trainScore)))
testScore = model.evaluate(testX, testY, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore, math.sqrt(testScore)))
```

```
Train Score: 7544227328.00 MSE (86857.51 RMSE)
Test Score: 13193974784.00 MSE (114865.03 RMSE)
```

B [17]:
```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

trainPredictPlot = numpy.empty_like(train)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

testPredictPlot = numpy.empty_like(train)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(train)-1, :] = testPredict

plt.plot(train)
plt.show()
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
plt.plot(train)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```







The average error in the training data is 86.857 units and the averrage in the test data is 114.865 units sold per day

сеть: LSTM слои+Conv1D +Dense слой.

B [ ]:
```
callback = EarlyStopping(monitor='val_mae', patience=2)

model = Sequential()

model.add(LSTM(64, input_shape = (inputs.shape[1], inputs.shape[2]), return_sequences="True"))
model.add(LSTM(64, return_sequences="True")) # <None, 3,32>
model.add(Conv1D(64, 3, activation="linear")) #(None,3,64)
#model.add(Conv1D(64, 1, activation="linear"))
model.add(Flatten())                          # (None, 3*64)
model.add(Dense(3, activation="linear"))      # (None,3)

model.add(Dense(1, activation="linear"))

model.compile(loss="mse", optimizer="adam", metrics=['mae'])

history = model.fit(
    dataset_train,
    epochs=epochs,
    validation_data=dataset_val,
    callbacks=[callback, tensorboard_callback])


plt.plot(history.history['mae'][1:],
         label='Средняя абсолютная ошибка на обучающем наборе')
plt.plot(history.history['val_mae'][1:],
         label='Средняя абсолютная ошибка на проверочном наборе')
plt.ylabel('Средняя ошибка')
plt.legend()
plt.show()
```

B [ ]:

B [ ]:

Все параметры класса RNN
Приведём список всех параметров класса RNN в фреймворке PyTorch:
nn.RNN
… (input_size, hidden_size, num_layers=1, nonlinearity='tanh', bias=True, … batch_first=False, dropout=0, bidirectional=False) [doc]

Отметим не упомянутый ранее параметр dropout. По умолчанию он равен нулю. При ненулевом значении, после каждого слоя (num_layers > 1), кроме последнего, вставляется слой dropout, который с вероятностью dropout случайно "отключает" (делает нулевыми) часть элементов тензоров на выходах каждой ячейке.

Установка параметра bias в значение False ликвидирует вектор смещения после перемножения матриц.

https://qudata.com/ml/ru/NN_RNN_Torch.html (https://qudata.com/ml/ru/NN_RNN_Torch.html) - ML: Рекуррентные сети на PyTorch

https://www.kaggle.com/code/byronvinu/lstm-network-for-regression/notebook (https://www.kaggle.com/code/byronvinu/lstm-network-for-regression/notebook)

B [ ]: