In [12]:

```
import os, warnings, random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import tensorflow as tf
import tensorflow.keras.layers as L
from tensorflow.keras import optimizers, Sequential, Model

# Set seeds to make the experiment more reproducible.
def seed_everything(seed=0):
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'

seed = 0
seed_everything(seed)
warnings.filterwarnings('ignore')
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

In [13]:

```
test = pd.read_csv('test.csv', dtype={'ID': 'int32', 'shop_id': 'int32',
                                                    'item_id': 'int32'})
item_categories = pd.read_csv('item_categories.csv',
                              dtype={'item_category_name': 'str', 'item_catego
ry_id': 'int32'})
items = pd.read_csv('items.csv', dtype={'item_name': 'str', 'item_id': 'int32'
,
                                                 'item_category_id': 'int32'})
shops = pd.read_csv('shops.csv', dtype={'shop_name': 'str', 'shop_id': 'int32'
})
sales = pd.read_csv('sales_train.csv', parse_dates=['date'],
                    dtype={'date': 'str', 'date_block_num': 'int32', 'shop_id'
: 'int32',
                      'item_id': 'int32', 'item_price': 'float32', 'item_cnt_d
ay': 'int32'})
```

In [14]:

```
train = sales.join(items, on='item_id', rsuffix='_').join(shops, on='shop_id',
rsuffix='_').join(item_categories, on='item_category_id', rsuffix='_').drop([
'item_id_', 'shop_id_', 'item_category_id_'], axis=1)
```

In [15]:

```
test_shop_ids = test['shop_id'].unique()
test_item_ids = test['item_id'].unique()
# Only shops that exist in test set.
train = train[train['shop_id'].isin(test_shop_ids)]
# Only items that exist in test set.
train = train[train['item_id'].isin(test_item_ids)]
```

In [16]:

```
train_monthly = train[['date', 'date_block_num', 'shop_id', 'item_id', 'item_c
nt_day']]
train_monthly = train_monthly.sort_values('date').groupby(['date_block_num',
'shop_id', 'item_id'], as_index=False)
train_monthly = train_monthly.agg({'item_cnt_day':['sum']})
train_monthly.columns = ['date_block_num', 'shop_id', 'item_id', 'item_cnt']
train_monthly = train_monthly.query('item_cnt >= 0 and item_cnt <= 20')
# Label
train_monthly['item_cnt_month'] = train_monthly.sort_values('date_block_num').
groupby(['shop_id', 'item_id'])['item_cnt'].shift(-1)

display(train_monthly.head(10).T)
display(train_monthly.describe().T)
```

	0	1	2	3	4	5	6	7	8	
date_block_num	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
shop_id	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	
item_id	33.00	482.00	491.00	839.00	1007.00	1010.00	1023.00	1204.00	1224.00	124
item_cnt	1.00	1.00	1.00	1.00	3.00	1.00	2.00	1.00	1.00	
item_cnt_month	2.00	1.00	1.00	1.00	1.00	1.00	1.00	nan	nan	

	count	mean	std	min	25%	50%	75%	max
date_block_num	593829.00	20.18	9.14	0.00	13.00	22.00	28.00	33.00
shop_id	593829.00	32.07	16.90	2.00	19.00	31.00	47.00	59.00
item_id	593829.00	10015.02	6181.82	30.00	4418.00	9171.00	15334.00	22167.00
item_cnt	593829.00	2.10	2.31	0.00	1.00	1.00	2.00	20.00
item_cnt_month	482536.00	2.07	2.17	0.00	1.00	1.00	2.00	20.00

In [17]:

```
monthly_series = train_monthly.pivot_table(index=['shop_id', 'item_id'], colum
ns='date_block_num',values='item_cnt', fill_value=0).reset_index()
monthly_series.head()
```

Out[17]:

date_block_num	shop_id	item_id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1
0	2	30	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	
1	2	31	0	4	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	
2	2	32	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	
3	2	33	1	0	0	0	0	0	0	0	0	0	2	1	1	0	0	0	0	
4	2	53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

In [18]:

```
first_month = 20
last_month = 33
serie_size = 12
data_series = []

for index, row in monthly_series.iterrows():
    for month1 in range((last_month - (first_month + serie_size)) + 1):
        serie = [row['shop_id'], row['item_id']]
        for month2 in range(serie_size + 1):
            serie.append(row[month1 + first_month + month2])
        data_series.append(serie)

columns = ['shop_id', 'item_id']
[columns.append(i) for i in range(serie_size)]
columns.append('label')

data_series = pd.DataFrame(data_series, columns=columns)
data_series.head()
```

Out[18]:

	shop_id	item_id	0	1	2	3	4	5	6	7	8	9	10	11	label
0	2	30	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	30	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	31	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	31	0	0	0	0	0	0	0	0	0	0	0	0	1
4	2	32	2	2	0	2	0	0	1	0	0	0	0	1	0

In [19]:

```
data_series = data_series.drop(['item_id', 'shop_id'], axis=1)
```

In [20]:

```
labels = data_series['label']
data_series.drop('label', axis=1, inplace=True)
train, valid, Y_train, Y_valid = train_test_split(data_series, labels.values,
test_size=0.10, random_state=0)
```

In [21]:

```
X_train = train.values.reshape((train.shape[0], train.shape[1], 1))
X_valid = valid.values.reshape((valid.shape[0], valid.shape[1], 1))

print("Train set reshaped", X_train.shape)
print("Validation set reshaped", X_valid.shape)
```

```
Train set reshaped (200327, 12, 1)
Validation set reshaped (22259, 12, 1)
```

In [22]:

```
serie_size =  X_train.shape[1] # 12
n_features =  X_train.shape[2] # 1

epochs = 20
batch = 128
lr = 0.0001

lstm_model = Sequential()
lstm_model.add(L.LSTM(10, input_shape=(serie_size, n_features), return_sequenc
es=True))
lstm_model.add(L.LSTM(6, activation='relu', return_sequences=True))
lstm_model.add(L.LSTM(1, activation='relu'))
lstm_model.add(L.Dense(10, kernel_initializer='glorot_normal', activation='rel
u'))
lstm_model.add(L.Dense(10, kernel_initializer='glorot_normal', activation='rel
u'))
lstm_model.add(L.Dense(1))
lstm_model.summary()

adam = optimizers.Adam(lr)
lstm_model.compile(loss='mse', optimizer=adam)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 12, 10)            480
_________________________________________________________________
lstm_1 (LSTM)                (None, 12, 6)             408
_________________________________________________________________
lstm_2 (LSTM)                (None, 1)                 32
_________________________________________________________________
dense (Dense)                (None, 10)                20
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 1,061
Trainable params: 1,061
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
encoder_decoder = Sequential()
encoder_decoder.add(L.LSTM(serie_size, activation='relu', input_shape=(serie_size, n_features), return_sequences=True))
encoder_decoder.add(L.LSTM(6, activation='relu', return_sequences=True))
encoder_decoder.add(L.LSTM(1, activation='relu'))
encoder_decoder.add(L.RepeatVector(serie_size))
encoder_decoder.add(L.LSTM(serie_size, activation='relu', return_sequences=True))
encoder_decoder.add(L.LSTM(6, activation='relu', return_sequences=True))
encoder_decoder.add(L.TimeDistributed(L.Dense(1)))
encoder_decoder.summary()

adam = optimizers.Adam(lr)
encoder_decoder.compile(loss='mse', optimizer=adam)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_3 (LSTM)                (None, 12, 12)            672       
_________________________________________________________________
lstm_4 (LSTM)                (None, 12, 6)             456       
_________________________________________________________________
lstm_5 (LSTM)                (None, 1)                 32        
_________________________________________________________________
repeat_vector (RepeatVector) (None, 12, 1)             0         
_________________________________________________________________
lstm_6 (LSTM)                (None, 12, 12)            672       
_________________________________________________________________
lstm_7 (LSTM)                (None, 12, 6)             456       
_________________________________________________________________
time_distributed (TimeDistri (None, 12, 1)             7         
=================================================================
Total params: 2,295
Trainable params: 2,295
Non-trainable params: 0
_________________________________________________________________
```

In [24]:

```
encoder_decoder_history = encoder_decoder.fit(X_train, X_train,
                                              batch_size=batch,
                                              epochs=epochs,
                                              verbose=2)
```

```
Epoch 1/20
1566/1566 - 40s - loss: 1.5515
Epoch 2/20
1566/1566 - 35s - loss: 1.1078
Epoch 3/20
1566/1566 - 36s - loss: 1.0432
Epoch 4/20
1566/1566 - 36s - loss: 1.0177
Epoch 5/20
1566/1566 - 35s - loss: 1.0002
Epoch 6/20
1566/1566 - 35s - loss: 0.9844
Epoch 7/20
1566/1566 - 36s - loss: 0.9721
Epoch 8/20
1566/1566 - 36s - loss: 0.9616
Epoch 9/20
1566/1566 - 36s - loss: 0.9567
Epoch 10/20
1566/1566 - 35s - loss: 0.9539
Epoch 11/20
1566/1566 - 35s - loss: 0.9451
Epoch 12/20
1566/1566 - 35s - loss: 0.9431
Epoch 13/20
1566/1566 - 35s - loss: 0.9391
Epoch 14/20
1566/1566 - 35s - loss: 0.9397
Epoch 15/20
1566/1566 - 35s - loss: 0.9401
Epoch 16/20
1566/1566 - 35s - loss: 0.9406
Epoch 17/20
1566/1566 - 35s - loss: 0.9350
Epoch 18/20
1566/1566 - 35s - loss: 0.9315
Epoch 19/20
1566/1566 - 35s - loss: 0.9269
Epoch 20/20
1566/1566 - 35s - loss: 0.9262
```

In [25]:

```
rpt_vector_layer = Model(inputs=encoder_decoder.inputs, outputs=encoder_decode
r.layers[3].output)
time_dist_layer = Model(inputs=encoder_decoder.inputs, outputs=encoder_decoder
.layers[5].output)
encoder_decoder.layers
```

Out[25]:

```
[<tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x7fe920cea590>,
 <tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x7fe92a928990>,
 <tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x7fe92a4f5950>,
 <tensorflow.python.keras.layers.core.RepeatVector at 0x7fe92a70b310>,
 <tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x7fe92a70b610>,
 <tensorflow.python.keras.layers.recurrent_v2.LSTM at 0x7fe92a7e0690>,
 <tensorflow.python.keras.layers.wrappers.TimeDistributed at 0x7fe92a7c36d0>]
```

In [26]:

```
encoder = Model(inputs=encoder_decoder.inputs, outputs=encoder_decoder.layers[
2].output)
```

In [27]:

```
train_encoded = encoder.predict(X_train)
validation_encoded = encoder.predict(X_valid)
print('Encoded time-series shape', train_encoded.shape)
print('Encoded time-series sample', train_encoded[0])
```

```
Encoded time-series shape (200327, 1)
Encoded time-series sample [2.6450368e-36]
```

In [28]:

```
train['encoded'] = train_encoded
train['label'] = Y_train

valid['encoded'] = validation_encoded
valid['label'] = Y_valid

train.head(10)
```

Out[28]:

	0	1	2	3	4	5	6	7	8	9	10	11	encoded	label
207604	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0
45150	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0
143433	0	0	4	2	1	2	2	1	0	0	0	1	3.89	1
202144	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0
136088	0	0	0	0	0	0	0	1	0	0	1	0	0.32	1
121675	0	0	0	0	0	1	0	0	0	0	0	0	0.28	0
185281	0	0	0	0	0	0	0	0	0	0	0	0	0.00	1
70087	0	0	0	0	0	0	0	0	3	0	1	3	2.00	0
105249	0	0	0	0	0	0	0	0	0	0	0	0	0.00	0
183257	0	0	0	0	0	3	0	1	0	0	0	1	1.33	0

In [29]:

```
last_month = serie_size - 1
Y_train_encoded = train['label']
train.drop('label', axis=1, inplace=True)
X_train_encoded = train[[last_month, 'encoded']]

Y_valid_encoded = valid['label']
valid.drop('label', axis=1, inplace=True)
X_valid_encoded = valid[[last_month, 'encoded']]

print("Train set", X_train_encoded.shape)
print("Validation set", X_valid_encoded.shape)
```

```
Train set (200327, 2)
Validation set (22259, 2)
```

In [30]:

```
mlp_model = Sequential()
mlp_model.add(L.Dense(10, kernel_initializer='glorot_normal', activation='rel
u', input_dim=X_train_encoded.shape[1]))
mlp_model.add(L.Dense(10, kernel_initializer='glorot_normal', activation='rel
u'))
mlp_model.add(L.Dense(1))
mlp_model.summary()

adam = optimizers.Adam(lr)
mlp_model.compile(loss='mse', optimizer=adam)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 10)                30        
_________________________________________________________________
dense_5 (Dense)              (None, 10)                110       
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 11        
=================================================================
Total params: 151
Trainable params: 151
Non-trainable params: 0
_________________________________________________________________
```

In [31]:

```
mlp_history = mlp_model.fit(X_train_encoded.values, Y_train_encoded.values, epochs=epochs, batch_size=batch, validation_data=(X_valid_encoded, Y_valid_encoded), verbose=2)
```

```
Epoch 1/20
1566/1566 - 2s - loss: 11.9316 - val_loss: 1.6461
Epoch 2/20
1566/1566 - 2s - loss: 1.3293 - val_loss: 1.2130
Epoch 3/20
1566/1566 - 2s - loss: 1.2345 - val_loss: 1.1845
Epoch 4/20
1566/1566 - 2s - loss: 1.2152 - val_loss: 1.1798
Epoch 5/20
1566/1566 - 2s - loss: 1.2090 - val_loss: 1.1803
Epoch 6/20
1566/1566 - 2s - loss: 1.2056 - val_loss: 1.1715
Epoch 7/20
1566/1566 - 2s - loss: 1.2037 - val_loss: 1.1704
Epoch 8/20
1566/1566 - 2s - loss: 1.2019 - val_loss: 1.1818
Epoch 9/20
1566/1566 - 2s - loss: 1.2013 - val_loss: 1.1685
Epoch 10/20
1566/1566 - 2s - loss: 1.1991 - val_loss: 1.1680
Epoch 11/20
1566/1566 - 2s - loss: 1.1979 - val_loss: 1.1701
Epoch 12/20
1566/1566 - 2s - loss: 1.1977 - val_loss: 1.1683
Epoch 13/20
1566/1566 - 2s - loss: 1.1966 - val_loss: 1.1668
Epoch 14/20
1566/1566 - 2s - loss: 1.1959 - val_loss: 1.1786
Epoch 15/20
1566/1566 - 2s - loss: 1.1948 - val_loss: 1.1688
Epoch 16/20
1566/1566 - 2s - loss: 1.1943 - val_loss: 1.1672
Epoch 17/20
1566/1566 - 2s - loss: 1.1944 - val_loss: 1.1703
Epoch 18/20
1566/1566 - 2s - loss: 1.1940 - val_loss: 1.1749
Epoch 19/20
1566/1566 - 2s - loss: 1.1935 - val_loss: 1.1795
Epoch 20/20
1566/1566 - 2s - loss: 1.1933 - val_loss: 1.1668
```

In [32]:

```
latest_records = monthly_series.drop_duplicates(subset=['shop_id', 'item_id'])
X_test = pd.merge(test, latest_records, on=['shop_id', 'item_id'], how='left',
suffixes=['', '_'])
X_test.fillna(0, inplace=True)
X_test.drop(['ID', 'item_id', 'shop_id'], axis=1, inplace=True)
X_test.head()
```

Out[32]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	(
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	(
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	(
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	(
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	(

In [34]:

```
X_test_reshaped = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))
print(X_test_reshaped.shape)
```

```
(214200, 34, 1)
```

In [35]:

```
lstm_test_pred = lstm_model.predict(X_test_reshaped)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 12, 1) for input K
erasTensor(type_spec=TensorSpec(shape=(None, 12, 1), dtype=tf.float32, name
='lstm_input'), name='lstm_input', description="created by layer 'lstm_inpu
t'"), but it was called on an input with incompatible shape (None, 34, 1).
```

In [ ]:

```
test_encoded = encoder.predict(X_test_reshaped)
```

In [ ]:

```
X_test['encoded'] = test_encoded
X_test.head()
```

In [ ]:

```
X_test_encoded = X_test[[33, 'encoded']]
print("Train set", X_test_encoded.shape)
X_test_encoded.head()
```

In [ ]:

```
mlp_test_pred = mlp_model.predict(X_test_encoded)
```