In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
import torch
from torch.utils.data import Dataset, Sampler, DataLoader
import torch.nn as nn
import torch.nn.functional as F
from tqdm import tqdm
import csv
```

Data Features

In [ ]:
```
calendar = pd.read_csv('calendar.csv')
sales = pd.read_csv('sales_train_validation.csv')
sales_2 = pd.read_csv('sales_train_evaluation.csv')
prices = pd.read_csv('sell_prices.csv')
```

In [ ]:
```
def plots ():

    mean = sales.mean(axis = 0)

    plt.plot(range(21),mean[0:21])
    plt.show()

    prices.groupby('wm_yr_wk').mean()

    plt.plot(range(len(prices.groupby('wm_yr_wk').mean())),prices.grou
pby('wm_yr_wk').mean())
    plt.show()

#plots()
```

In [ ]:
```
def construct_dates ():

    # weeks
    weeks = pd.get_dummies(calendar.wday)
    weeks.columns = ['sat','sun','mon','tue', 'wen','thu','fri']

    # months
    month = pd.get_dummies(calendar.month)

    # events
    # calendar['event_name_1'].unique().size
    # calendar['event_name_1'].unique()

    event = pd.get_dummies(calendar.event_type_1)
```

```
    event2 = pd.get_dummies(calendar.event_type_2)

    for i in event2.columns:

        event[i][event2[i]==1]=1

    #event.iloc[1968]
    #event.info()

    global date_features

    # Construct dates dataset
    date_features = pd.concat([calendar['wm_yr_wk'], weeks, month, eve
nt,
        calendar['snap_CA'],calendar['snap_TX'],calendar['snap_WI']],
axis = 1)
    date_features.head()

    date_features.to_csv('date_features.csv' )

#construct_dates ()

def construct_items():

    # items
    dept = pd.get_dummies(sales.dept_id)
    pd.get_dummies(sales.cat_id).head()
    store = pd.get_dummies(sales.store_id)

    # Construct items dataset

    global item_features
    item_features = pd.DataFrame()
    item_features['store_item'] = sales['store_id']+sales['item_id']
    item_features = pd.concat([item_features , dept, store], axis = 1)
    item_features.to_csv('item_features.csv' )

#construct_items()

# sales

def construct_sales():
    sales_data = sales_2.drop(['id', 'dept_id', 'cat_id', 'state_id','
item_id','store_id'], axis = 1)
    sales_data.to_csv('sales_data.csv' )

#construct_sales()

#prices

def construct_prices ():

    global prices_data
```

```
    prices['store_item'] = prices['store_id']+prices['item_id']
    prices.drop(['store_id', 'item_id'],axis = 1, inplace = True)

    #mean = prices.sell_price.mean()
    #std = prices.sell_price.std()
    #prices.sell_price = (prices.sell_price - mean)/ std

    #min_max_scaler = preprocessing.MinMaxScaler()
    #prices.sell_price = min_max_scaler.fit_transform(prices.sell_pric
e.values.reshape(-1, 1))

    prices_data = pd.pivot_table(prices, values='sell_price',
            index=['store_item'], columns=['wm_yr_wk'])

    index = item_features.store_item
    prices_data = prices_data.reindex(index)

    prices_data.loc['WI_3HOUSEHOLD_2_406'][prices_data.loc['WI_3HOUSEH
OLD_2_406']>30] = 30
    prices_data.loc['WI_1HOUSEHOLD_2_406'][prices_data.loc['WI_1HOUSEH
OLD_2_406']>30] = 30
    prices_data.loc['WI_2HOUSEHOLD_2_406'][prices_data.loc['WI_2HOUSEH
OLD_2_406']>30] = 30
    prices_data.loc['TX_1HOUSEHOLD_2_466'][prices_data.loc['TX_1HOUSEH
OLD_2_466']>30] = 30
    prices_data.loc['TX_1HOUSEHOLD_2_178'][prices_data.loc['TX_1HOUSEH
OLD_2_178']>30] = 30
    prices_data.loc['WI_2HOUSEHOLD_2_250'][prices_data.loc['WI_2HOUSEH
OLD_2_250']>30] = 30

    prices_data.to_csv('prices_data.csv' )

#construct_prices ()
```

Dataset

In [ ]:
```
calendar_path = 'date_features.csv'
items_path = 'item_features.csv'
sales_path = 'sales_data.csv'
prices_path = 'prices_data.csv'
```

In [ ]:
```
def preprocess():

    calendar = pd.read_csv(calendar_path, index_col = 0)
    items = pd.read_csv(items_path, index_col = 0)
    sales = pd.read_csv(sales_path, index_col = 0)
    prices = pd.read_csv(prices_path,index_col = 0)

    price_std = prices.stack().std()
    price_mean = prices.mean().mean()

    sales_std = sales.stack().std()
    sales_mean = prices.mean().mean()

    prices_standar = (prices - price_mean) / price_std
    sales_standar = (sales - sales_mean) / sales_std

    return calendar,items, prices_standar, sales_standar, sales
```

In [ ]:
```
class Data (Dataset):

    def __init__(self):

        self.calendar, self.items, self.prices, self.sales, self.sales
_target = preprocess()

    def construct_index_train (self):


        date = []
        item = []

        for i in tqdm(range (67)):

            if i < 47:

                date_idx = 9 + i * 28 + 27
                week = (self.calendar.iloc[date_idx].wm_yr_wk)

                for j in range( len (self.prices)):

                    if not np.isnan(self.prices.iloc[j].loc[str(week)]
) :

                        date.append(i)
                        item.append(j)

            else:
                for j in range( len (self.prices)):

                    date.append(i)
                    item.append(j)
```

```
        idxzip = pd.DataFrame(data={"date": date, "item": item})
        idxzip.to_csv("./idxzip.csv", sep=',',index=False)

        return zip(date, item)


    def __getitem__(self, index):

        index1, item_idx = index

        date_idx = 9 + index1 * 28
        dates = self.calendar.iloc[date_idx : date_idx + 28].copy()

        item = self.items.iloc[item_idx].copy()
        store_item = item['store_item']

        weeks = dates['wm_yr_wk'].unique()
        dates['price'] = ""
        dates['not_sold'] = pd.Series(np.zeros((28)), index=dates.inde
x)

        for week in weeks:

            weekprice = self.prices.loc[store_item,str(week)]

            if np.isnan(weekprice):

                dates.loc[dates.wm_yr_wk == week,
                    'price'] = 8
                dates.loc[dates.wm_yr_wk == week,
                    'not_sold'] = 1
            else:
                dates.loc[dates.wm_yr_wk == week,
                    'price'] = weekprice

        past_sales = self.sales.iloc[item_idx , date_idx - 9 : date_id
x].mean()

        X_dates = dates.drop(['wm_yr_wk'], axis=1).values.astype('floa
t32')

        X_item = item[1:].values
        X_item = np.insert(X_item,0,past_sales).astype('float32')

        Y = self.sales_target.iloc[item_idx , date_idx : date_idx + 28
].values.astype('float32')


        return X_dates, X_item, Y

    def __len__ (self):
```

```
        return ((len(self.calendar)-9)/28) * len(self.items)

dataset = Data ()
```

In [ ]:
```
train_index = dataset.construct_index_train()

#idxdf = pd.read_csv('idxzip.csv')
#train_index = zip(idxdf.date.tolist() , idxdf.item.tolist())
```

Sampler / Dataloader

In [ ]:
```
class Sampler(Sampler):

    def __init__(self, train, test = False, val = False, train_index =
None):

        self.items = 30490
        self.train = train
        self.test = test
        self.val = val

        if self.train == True:
            self.dates = range(67)
            self.lenght = self.items * 67
            self.index = train_index
        elif self.test == True:
            self.dates = range(67,68)
            self.lenght = self.items

        elif self.val == True:
            self.dates = range(68, 69)
            self.lenght = self.items
        else:
            self.dates = range(69, 70)
            self.lenght = self.items

    def __iter__(self):

        if self.train == False:

            date = []
            item = []
            for i in self.dates:
                for j in range(self.items):

                        date.append(i)
                        item.append(j)

            return iter(zip(date, item))

        else :

            return iter(self.index)

    def __len__(self):

        return self.lenght
```

In [ ]:
```
train_sampler = Sampler(train = True, train_index = train_index)
test_sampler = Sampler(train = False, test = True)
val_sampler = Sampler (train = False, val = True)
pred_sampler = Sampler (train = False)
```

In [ ]:
```
train_loader  = DataLoader(dataset,
    batch_size=4, sampler=train_sampler, shuffle=False, num_workers =
3, drop_last=True)

test_loader  = DataLoader(dataset,
    batch_size=1, sampler=test_sampler, shuffle=False, num_workers = 3
)

val_loader  = DataLoader(dataset,
    batch_size=1, sampler=val_sampler, shuffle=False, num_workers = 3)

pred_loader  = DataLoader(dataset,
    batch_size=1, sampler=pred_sampler, shuffle=False, num_workers = 3
)
```

Model

In [ ]:
```
class Network(nn.Module):

    def __init__(self, dates_dim, item_dim, lstm_hidden, hidden1,
            seq_len = 28, lstm_num_layers = 1, LSTM_dropout = 0):

        super(Network, self).__init__()

        self.lstm = nn.LSTM(dates_dim, lstm_hidden, lstm_num_layers,
                        batch_first = True, dropout = LSTM_dropout )
        self.linear_out = nn.Linear(lstm_hidden, 1)

        self.linear_1 = nn.Linear(item_dim, hidden1)
        self.linear_h0 = nn.Linear(hidden1, lstm_hidden)
        self.linear_c0 = nn.Linear(hidden1, lstm_hidden)

        self.sigm = nn.Sigmoid()
        self.relu = nn.ReLU()


    def forward (self, x_dates, x_item):

        x_item = self.sigm(self.linear_1(x_item)) # (batch, hidden1)
        h0 = self.sigm(self.linear_h0(x_item))    # (batch, lstm _hidd
en)
        c0 = self.sigm(self.linear_c0(x_item))    # (batch, lstm _hidd
en)

        h0 = torch.unsqueeze(h0, 0)               # (1, batch, lstm _h
idden)
        c0 = torch.unsqueeze(c0, 0)               # (1, batch, lstm _h
idden)

        output, (hn, cn) = self.lstm(x_dates, (h0, c0)) # (batch, 28,
lstm_hidden)
        output = self.relu(self.linear_out(output))    # (batch, 28, 1
)
        output = torch.squeeze(output, 2)            # (batch, 28)

        return output
```

In [ ]:
```
model = Network (28, 18, 60, 25)
```

In [ ]:
```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Train

In [ ]:
```
epochs = 4
prin = 3200
test = 320000

avg_test_loss = []
min_test_loss = 8.81

model.train()
```

In [ ]:
```
def train ():
    for epoch in range(epochs):

        for i, data in tqdm(enumerate(train_loader)):

            x_date, x_item, target = data
            model.zero_grad()
            pred =  model(x_date, x_item)
            loss = criterion (pred, target)
            loss.backward()
            optimizer.step()

            if i % prin == 0:

                print ('\ni:', i , '  Epoch:', epoch)
                print ('\nLoss: ', loss.item())
                print (data[0].shape, data[1].shape, data[2].shape)

                if (i % test == 0 and i != 0):

                    model.eval()
                    test_loss = []
                    for testdata in test_loader:

                        x_date_test, x_item_test, target_test = testda
ta
                        pred_test =  model(x_date_test, x_item_test)
                        loss_test = criterion (pred_test, target_test)
                        test_loss.append(loss_test.item())

                    avg_test_loss.append(np.mean(test_loss))
                    model.train()
                    print ('\n\n\nAvg test loss:  ', np.mean(test_loss
))
                    print ('\n', avg_test_loss, '\n\n\n')
                    if np.mean(test_loss) <= min_test_loss:

                        torch.save(model.state_dict(), './state_dict8.
pt')
                        valid_loss_min = np.mean(test_loss)

    model.eval()
    test_loss = []
```

```
    for testdata in test_loader:

        x_date_test, x_item_test, target_test = testdata
        pred_test =  model(x_date_test, x_item_test)
        loss_test = criterion (pred_test, target_test)
        test_loss.append(loss_test.item())

    avg_test_loss.append(np.mean(test_loss))
    print ('\n\n\nAvg test loss:  ', np.mean(test_loss))
    print ('\n', avg_test_loss, '\n\n\n')
    if np.mean(test_loss) <= min_test_loss:

        torch.save(model.state_dict(), './state_dict8.pt')
        valid_loss_min = np.mean(test_loss)

    print ('\n --------------------------\n test loss :', avg_test_lo
ss)

train()
```

In [ ]:
```
model.load_state_dict(torch.load('./state_dict8.pt'))

model.eval()
val_loss = []

index = pd.Index(pd.read_csv('subm_idx.csv', header=None, index_col =
0).iloc[:,0])
submission = pd.DataFrame(index = index, columns = ['F1','F2','F3','F4
','F5','F6','F7','F8','F9','F10',
                                                    'F11','F12','F13','F14','F15',
'F16','F17','F18','F19',
                                                    'F20','F21','22','F23','F24','
F25','F26','F27','F28'])
```

In [ ]:
```
for i, valdata in tqdm(enumerate(val_loader)):

    x_date_val, x_item_val, target_val = valdata
    pred_val =  model(x_date_val, x_item_val)
    loss_val = criterion (pred_val, target_val)
    val_loss.append(loss_val.item())
    submission.iloc[i] = pred_val.detach().numpy()

    if i % 3000 == 0:

        print (np.mean(val_loss))

print ('val loss : ', np.mean(val_loss))
```

In [ ]:
```
for i, preddata in tqdm(enumerate(pred_loader)):

    x_date_pred, x_item_pred, target_pred = preddata
    pred_pred =  model(x_date_pred, x_item_pred)
    submission.iloc[i+30490] = pred_pred.detach().numpy()

submission.to_csv('submission8.csv' )
```

In [ ]:
```
print ('Test MSEloss :', avg_test_loss)
print ('Validation MSEloss : ', np.mean(val_loss))
print ('---------------------------')
print ('Test RMSEloss :', np.sqrt(avg_test_loss))
print ('Validation RMSEloss : ', np.sqrt(np.mean(val_loss)))
```