Лабораторная работа №6:

Щипицина К.В.

ИУ5-22М

"Разработка системы предсказания поведения на основании графовых моделей"

Цель: обучение работе с графовым типом данных и графовыми нейронными сетями.

Задача: подготовить графовый датасет из базы данных о покупках и построить модель предсказания совершения покупки.

Графовые нейронные сети

Графовые нейронные сети - тип нейронной сети, которая напрямую работает со структурой графа. Типичным применениями GNN являются:

- Классификация узлов;
- Предсказание связей;
- Графовая классификация;
- Распознавание движений;
- Рекомендательные системы.

В данной лабораторной работе будет происходить работа над **графовыми сверточными сетями**. Отличаются они от сверточных нейронных сетей нефиксированной структурой, функция свертки не является .

Подробнее можно прочитать тут: https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b

Тут можно почитать современные подходы к использованию графовых сверточных сетей https://paperswithcode.com/method/gcn

---

Датасет

В качестве базы данных предлагаем использовать датасет о покупках пользователей в одном магазине товаров RecSys Challenge 2015 (https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015).

Скачать датасет можно отсюда: https://drive.google.com/drive/folders/1gtAeXPTj-c0RwVOKreMrZ3bfSmCwl2y-?usp=sharing (lite-версия является облегченной версией исходного датасета, рекомендуем использовать её)

Также рекомендуем загружать данные в виде архива и распаковывать через пакет zipfile или/и скачивать датасет в собственный Google Drive и примонтировать его в колаб.

Установка библиотек, выгрузка исходных датасетов

```
# Slow method of installing pytorch geometric
# !pip install torch_geometric
# !pip install torch_sparse
# !pip install torch_scatter

# Install pytorch geometric
!pip install torch-sparse -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-cluster -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-spline-conv -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-geometric -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
!pip install torch-scatter==2.0.8 -f https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from torch-sparse) (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scipy->torch-sparse) (1.21.6)
Installing collected packages: torch-sparse
Successfully installed torch-sparse-0.6.13
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Collecting torch-cluster
  Downloading https://data.pyg.org/whl/torch-1.11.0%2Bcu113/torch_cluster-1.6.0-cp37-cp37m-linux_x86_64.whl (2.5 MB)
     |████████████████████████████████| 2.5 MB 26.1 MB/s
Installing collected packages: torch-cluster
Successfully installed torch-cluster-1.6.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Collecting torch-spline-conv
  Downloading https://data.pyg.org/whl/torch-1.11.0%2Bcu113/torch_spline_conv-1.2.1-cp37-cp37m-linux_x86_64.whl (750 kB)
     |████████████████████████████████| 750 kB 36.2 MB/s
Installing collected packages: torch-spline-conv
Successfully installed torch-spline-conv-1.2.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Collecting torch-geometric
  Downloading torch_geometric-2.0.4.tar.gz (407 kB)
     |████████████████████████████████| 407 kB 27.6 MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (4.64.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (1.21.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (1.4.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (1.3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (2.11.3)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (2.23.0)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (3.0.9)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from torch-geometric) (1.0.2)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->torch-geometric) (
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->torch-geomet
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->torch-geometric) (2022
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->to
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from req
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->torch-geometric) (2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->torch-geometri
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->torch-geometric
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->torch-ge
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->torch-geometric)
Building wheels for collected packages: torch-geometric
  Building wheel for torch-geometric (setup.py) ... done
  Created wheel for torch-geometric: filename=torch_geometric-2.0.4-py3-none-any.whl size=616603 sha256=555075e4801fe09ccf
  Stored in directory: /root/.cache/pip/wheels/18/a6/a4/ca18c3051fcead866fe7b85700ee2240d883562a1bc70ce421
Successfully built torch-geometric
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.0.4
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
Collecting torch-scatter==2.0.8
  Downloading torch_scatter-2.0.8.tar.gz (21 kB)
Building wheels for collected packages: torch-scatter
  Building wheel for torch-scatter (setup.py) ... done
  Created wheel for torch-scatter: filename=torch_scatter-2.0.8-cp37-cp37m-linux_x86_64.whl size=3221895 sha256=0c2caf4454
  Stored in directory: /root/.cache/pip/wheels/96/e4/4e/2bcc6de6a801960aedbca43f7106d268f766c3f9f8ab49b3a5
Successfully built torch-scatter
Installing collected packages: torch-scatter
```

```python
import numpy as np
import pandas as pd
import pickle
import csv
import os
from sklearn.preprocessing import LabelEncoder
import torch

# PyG - PyTorch Geometric
from torch_geometric.data import Data, DataLoader, InMemoryDataset
from tqdm import tqdm
RANDOM_SEED = 42 #@param { type: "integer" }
BASE_DIR = '/content/' #@param { type: "string" }
np.random.seed(RANDOM_SEED)
```

RANDOM_SEED: 42

BASE_DIR: " /content/ "

```python
# Check if CUDA is available for colab
torch.cuda.is_available
```

```
<function torch.cuda.is_available>
```

```
!gdown --id 1ebYsQlb6PRQMOE7Cnx1CwWVFDNJkYuzR
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and wil
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1ebYsQlb6PRQMOE7Cnx1CwWVFDNJkYuzR
To: /content/yoochoose-data-lite.zip
100% 49.8M/49.8M [00:00<00:00, 289MB/s]
```

```python
# Unpack files from zip-file
import zipfile
with zipfile.ZipFile(BASE_DIR + 'yoochoose-data-lite.zip', 'r') as zip_ref:
    zip_ref.extractall(BASE_DIR)
```

Анализ исходных данных

```
# Read dataset of items in store
df = pd.read_csv(BASE_DIR + 'yoochoose-clicks-lite.dat')
# df.columns = ['session_id', 'timestamp', 'item_id', 'category']
df.head()
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (3) have mixed types.Spec
  exec(code_obj, self.user_global_ns, self.user_ns)
```

	session_id	timestamp	item_id	category
0	9	2014-04-06T11:26:24.127Z	214576500	0
1	9	2014-04-06T11:28:54.654Z	214576500	0
2	9	2014-04-06T11:29:13.479Z	214576500	0
3	19	2014-04-01T20:52:12.357Z	214561790	0
4	19	2014-04-01T20:52:13.758Z	214561790	0

```
# Read dataset of purchases
buy_df = pd.read_csv(BASE_DIR + 'yoochoose-buys-lite.dat')
# buy_df.columns = ['session_id', 'timestamp', 'item_id', 'price', 'quantity']
buy_df.head()
```

	session_id	timestamp	item_id	price	quantity
0	420374	2014-04-06T18:44:58.314Z	214537888	12462	1
1	420374	2014-04-06T18:44:58.325Z	214537850	10471	1
2	489758	2014-04-06T09:59:52.422Z	214826955	1360	2
3	489758	2014-04-06T09:59:52.476Z	214826715	732	2
4	489758	2014-04-06T09:59:52.578Z	214827026	1046	1

```
# Filter out item session with length < 2
df['valid_session'] = df.session_id.map(df.groupby('session_id')['item_id'].size() > 2)
df = df.loc[df.valid_session].drop('valid_session',axis=1)
df.nunique()
```

```
session_id    1000000
timestamp     5557758
item_id         37644
category          275
dtype: int64
```

```
# Randomly sample a couple of them
NUM_SESSIONS = 60000 #@param { type: "integer" }
sampled_session_id = np.random.choice(df.session_id.unique(), NUM_SESSIONS, replace=False)
df = df.loc[df.session_id.isin(sampled_session_id)]
df.nunique()
```

NUM_SESSIONS: 60000

```
session_id     60000
timestamp     334117
item_id        19486
category         118
dtype: int64
```

```
# Average length of session
df.groupby('session_id')['item_id'].size().mean()
```

```
5.568833333333333
```

```
# Encode item and category id in item dataset so that ids will be in range (0,len(df.item.unique()))
item_encoder = LabelEncoder()
category_encoder = LabelEncoder()
df['item_id'] = item_encoder.fit_transform(df.item_id)
df['category']= category_encoder.fit_transform(df.category.apply(str))
df.head()
```

	session_id	timestamp	item_id	category
0	9	2014-04-06T11:26:24.127Z	3695	0
1	9	2014-04-06T11:28:54.654Z	3695	0
2	9	2014-04-06T11:29:13.479Z	3695	0
102	171	2014-04-03T17:45:25.575Z	10635	0

```
# Encode item and category id in purchase dataset
buy_df = buy_df.loc[buy_df.session_id.isin(df.session_id)]
buy_df['item_id'] = item_encoder.transform(buy_df.item_id)
buy_df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
  This is separate from the ipykernel package so we can avoid doing imports until
```

	session_id	timestamp	item_id	price	quantity
33	189	2014-04-04T07:23:10.719Z	5576	4711	1
46	489491	2014-04-06T12:41:34.047Z	13388	1046	4
47	489491	2014-04-06T12:41:34.091Z	13389	627	2
57	396	2014-04-06T17:53:45.147Z	13579	523	1
61	70353	2014-04-06T10:55:06.086Z	15174	41783	1

```
# Get item dictionary with grouping by session
buy_item_dict = dict(buy_df.groupby('session_id')['item_id'].apply(list))
buy_item_dict
```

```
 1439748: [10968, 11081],
 1440946: [8930, 1213],
 1442774: [14448],
 1445273: [13390],
 1449601: [15865],
 1450532: [4036, 1165],
 1452693: [4359, 13408, 14451, 13523],
 1454342: [6082],
 1456618: [4374, 11776],
 1459927: [13438, 15622],
 1460198: [14450, 14451, 14451, 14450],
 1460323: [5544,
  12326,
  13216,
  13592,
  13633,
  7163,
  8761,
  11650,
  6974,
  13144,
  6973,
  11649,
  11648,
  181],
 1463706: [13766, 13766, 13766, 13766],
 1464933: [9795],
 1468993: [14450, 13535, 14451],
 1473907: [13848, 13439, 9275, 13848, 9275, 13439],
 1475487: [15793, 15791],
 1477591: [13840, 14425, 13529, 14451, 13840, 14425, 13529, 14451],
 1479551: [13394],
 1481656: [655],
 1482626: [13846, 14450, 13438],
 1482757: [2094],
 1489531: [13535, 14451],
 1489998: [13037],
 1492547: [12507, 11213],
 1495173: [14451, 13401, 13035],
 1498052: [14451, 11081],
 1506466: [13764],
 1506563: [13655, 8943, 3598],
 1510023: [14278],
 1519191: [14448],
 1519934: [14450, 14425, 14451, 13438],
 1521264: [14450, 14451],
 1521999: [12926,
          15760,
          2800,
          11139,
          1747,
          10619,
          14554,
          13109,
          3067,
          11608,
          2215],
         1526834: [5404, 5146, 5404, 5146],
```

Сборка выборки для обучения

```
# Transform df into tensor data
def transform_dataset(df, buy_item_dict):
    data_list = []

    # Group by session
    grouped = df.groupby('session_id')
    for session_id, group in tqdm(grouped):
        le = LabelEncoder()
        sess_item_id = le.fit_transform(group.item_id)
        group = group.reset_index(drop=True)
        group['sess_item_id'] = sess_item_id

        #get input features
        node_features = group.loc[group.session_id==session_id,
                            ['sess_item_id','item_id','category']].sort_values('sess_item_id')[['item_id','category']].dro
        node_features = torch.LongTensor(node_features).unsqueeze(1)
        target_nodes = group.sess_item_id.values[1:]
        source_nodes = group.sess_item_id.values[:-1]

        edge_index = torch.tensor([source_nodes,
                                target_nodes], dtype=torch.long)
        x = node_features

        #get result
        if session_id in buy_item_dict:
            positive_indices = le.transform(buy_item_dict[session_id])
            label = np.zeros(len(node_features))
            label[positive_indices] = 1
        else:
            label = [0] * len(node_features)

        y = torch.FloatTensor(label)

        data = Data(x=x, edge_index=edge_index, y=y)

        data_list.append(data)

    return data_list

# Pytorch class for creating datasets
class YooChooseDataset(InMemoryDataset):
    def __init__(self, root, transform=None, pre_transform=None):
        super(YooChooseDataset, self).__init__(root, transform, pre_transform)
        self.data, self.slices = torch.load(self.processed_paths[0])

    @property
    def raw_file_names(self):
        return []

    @property
    def processed_file_names(self):
        return [BASE_DIR+'yoochoose_click_binary_100000_sess.dataset']

    def download(self):
        pass

    def process(self):
        data_list = transform_dataset(df, buy_item_dict)

        data, slices = self.collate(data_list)
        torch.save((data, slices), self.processed_paths[0])
```

```
# Prepare dataset
dataset = YooChooseDataset('./')
```

```
Processing...
  0%|          | 0/60000 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: UserWarning: Creati
100%|██████████| 60000/60000 [03:22<00:00, 296.26it/s]
Done!
```

Разделение выборки

```
# train_test_split
dataset = dataset.shuffle()
one_tenth_length = int(len(dataset) * 0.1)
train_dataset = dataset[:one_tenth_length * 8]
val_dataset = dataset[one_tenth_length*8:one_tenth_length * 9]
test_dataset = dataset[one_tenth_length*9:]
len(train_dataset), len(val_dataset), len(test_dataset)
```

```
(48000, 6000, 6000)
```

```
# Load dataset into PyG loaders
batch_size= 512
train_loader = DataLoader(train_dataset, batch_size=batch_size)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use
  warnings.warn(out)
```

```
# Load dataset into PyG loaders
num_items = df.item_id.max() +1
num_categories = df.category.max()+1
num_items , num_categories
```

```
(19486, 117)
```

Настройка модели для обучения

```
embed_dim = 128
from torch_geometric.nn import GraphConv, TopKPooling, GatedGraphConv, SAGEConv, SGConv
from torch_geometric.nn import global_mean_pool as gap, global_max_pool as gmp
import torch.nn.functional as F

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # Model Structure
        self.conv1 = GraphConv(embed_dim * 2, 128)
        self.pool1 = TopKPooling(128, ratio=0.9)
        self.conv2 = GraphConv(128, 128)
        self.pool2 = TopKPooling(128, ratio=0.9)
        self.conv3 = GraphConv(128, 128)
        self.pool3 = TopKPooling(128, ratio=0.9)
        self.item_embedding = torch.nn.Embedding(num_embeddings=num_items, embedding_dim=embed_dim)
        self.category_embedding = torch.nn.Embedding(num_embeddings=num_categories, embedding_dim=embed_dim)
        self.lin1 = torch.nn.Linear(256, 256)
        self.lin2 = torch.nn.Linear(256, 128)
        self.bn1 = torch.nn.BatchNorm1d(128)
        self.bn2 = torch.nn.BatchNorm1d(64)
        self.act1 = torch.nn.ReLU()
        self.act2 = torch.nn.ReLU()

    # Forward step of a model
    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch

        item_id = x[:,:,0]
        category = x[:,:,1]


        emb_item = self.item_embedding(item_id).squeeze(1)
        emb_category = self.category_embedding(category).squeeze(1)

        x = torch.cat([emb_item, emb_category], dim=1)
        # print(x.shape)
        x = F.relu(self.conv1(x, edge_index))
        # print(x.shape)
        r = self.pool1(x, edge_index, None, batch)
        # print(r)
        x, edge_index, _, batch, _, _ = self.pool1(x, edge_index, None, batch)
        x1 = torch.cat([gmp(x, batch), gap(x, batch)], dim=1)

        x = F.relu(self.conv2(x, edge_index))

        x, edge_index, _, batch, _, _ = self.pool2(x, edge_index, None, batch)
        x2 = torch.cat([gmp(x, batch), gap(x, batch)], dim=1)

        x = F.relu(self.conv3(x, edge_index))

        x, edge_index, _, batch, _, _ = self.pool3(x, edge_index, None, batch)
        x3 = torch.cat([gmp(x, batch), gap(x, batch)], dim=1)

        x = x1 + x2 + x3

        x = self.lin1(x)
        x = self.act1(x)
        x = self.lin2(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.act2(x)

        outputs = []
        for i in range(x.size(0)):
            output = torch.matmul(emb_item[data.batch == i], x[i,:])

            outputs.append(output)

        x = torch.cat(outputs, dim=0)
        x = torch.sigmoid(x)

        return x
```

Обучение нейронной сверточной сети

```
# Enable CUDA computing
device = torch.device('cuda')
model = Net().to(device)
# Choose optimizer and criterion for learning
optimizer = torch.optim.Adam(model.parameters(), lr=0.0015)
crit = torch.nn.BCELoss()
```

```
# Train function
def train():
    model.train()

    loss_all = 0
    for data in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        output = model(data)

        label = data.y.to(device)
        loss = crit(output, label)
        loss.backward()
        loss_all += data.num_graphs * loss.item()
        optimizer.step()
    return loss_all / len(train_dataset)
```

```
# Evaluate result of a model
from sklearn.metrics import roc_auc_score
def evaluate(loader):
    model.eval()

    predictions = []
    labels = []

    with torch.no_grad():
        for data in loader:

            data = data.to(device)
            pred = model(data).detach().cpu().numpy()

            label = data.y.detach().cpu().numpy()
            predictions.append(pred)
            labels.append(label)

    predictions = np.hstack(predictions)
    labels = np.hstack(labels)

    return roc_auc_score(labels, predictions)
```

```
# Train a model
NUM_EPOCHS =  12 #@param { type: "integer" }
for epoch in tqdm(range(NUM_EPOCHS)):
    loss = train()
    train_acc = evaluate(train_loader)
    val_acc = evaluate(val_loader)
    test_acc = evaluate(test_loader)
    print('Epoch: {:03d}, Loss: {:.5f}, Train Auc: {:.5f}, Val Auc: {:.5f}, Test Auc: {:.5f}'.
          format(epoch, loss, train_acc, val_acc, test_acc))
```

NUM_EPOCHS: 12

```
  8%|▊         | 1/12 [00:47<08:45, 47.75s/it]Epoch: 000, Loss: 0.69025, Train Auc: 0.52159, Val Auc: 0.52578, Test Auc: 0.5
 17%|█▋        | 2/12 [01:31<07:33, 45.38s/it]Epoch: 001, Loss: 0.49150, Train Auc: 0.57037, Val Auc: 0.56462, Test Auc: 0.5
 25%|██▌       | 3/12 [02:14<06:40, 44.53s/it]Epoch: 002, Loss: 0.41170, Train Auc: 0.60795, Val Auc: 0.57826, Test Auc: 0.57
 33%|███▎      | 4/12 [02:58<05:51, 43.98s/it]Epoch: 003, Loss: 0.37954, Train Auc: 0.63892, Val Auc: 0.59589, Test Auc: 0.5
 42%|████▏     | 5/12 [03:41<05:05, 43.61s/it]Epoch: 004, Loss: 0.35453, Train Auc: 0.66910, Val Auc: 0.60876, Test Auc: 0.5
 50%|█████     | 6/12 [04:23<04:19, 43.28s/it]Epoch: 005, Loss: 0.33685, Train Auc: 0.70074, Val Auc: 0.62465, Test Auc: 0.6
 58%|█████▊    | 7/12 [05:06<03:35, 43.17s/it]Epoch: 006, Loss: 0.31992, Train Auc: 0.73408, Val Auc: 0.63180, Test Auc: 0.6
 67%|██████▋   | 8/12 [05:49<02:52, 43.05s/it]Epoch: 007, Loss: 0.30017, Train Auc: 0.76687, Val Auc: 0.64112, Test Auc: 0.6
 75%|███████▌  | 9/12 [06:32<02:08, 42.96s/it]Epoch: 008, Loss: 0.28806, Train Auc: 0.79602, Val Auc: 0.64866, Test Auc: 0.62
 83%|████████▎ | 10/12 [07:15<01:25, 42.98s/it]Epoch: 009, Loss: 0.27429, Train Auc: 0.82846, Val Auc: 0.65896, Test Auc: 0.
 92%|█████████▏| 11/12 [07:58<00:43, 43.14s/it]Epoch: 010, Loss: 0.25916, Train Auc: 0.85200, Val Auc: 0.66123, Test Auc: 0.
100%|██████████| 12/12 [08:41<00:00, 43.50s/it]Epoch: 011, Loss: 0.25032, Train Auc: 0.86827, Val Auc: 0.66505, Test Auc: 0.6
```

Проверка результата с помощью примеров

```
# Подход №1 - из датасета
evaluate(DataLoader(test_dataset[40:60], batch_size=10))
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use
  warnings.warn(out)
0.8325581395348838
```

```
# Подход №2 - через создание сессии покупок
test_df = pd.DataFrame([
      [-1, 15219, 0],
      [-1, 15431, 0],
      [-1, 14371, 0],
      [-1, 15745, 0],
      [-2, 14594, 0],
      [-2, 16972, 11],
      [-2, 16943, 0],
      [-3, 17284, 0]
], columns=['session_id', 'item_id', 'category'])

test_data = transform_dataset(test_df, buy_item_dict)
test_data = DataLoader(test_data, batch_size=1)

with torch.no_grad():
    model.eval()
    for data in test_data:
        data = data.to(device)
        pred = model(data).detach().cpu().numpy()

        print(data, pred)
```

```
100%|██████████| 3/3 [00:00<00:00, 218.42it/s]DataBatch(x=[1, 1, 2], edge_index=[2, 0], y=[1], batch=[1], ptr=[2]) [0.0008603
DataBatch(x=[3, 1, 2], edge_index=[2, 2], y=[3], batch=[3], ptr=[2]) [0.04850192 0.02720772 0.0045843 ]
DataBatch(x=[4, 1, 2], edge_index=[2, 3], y=[4], batch=[4], ptr=[2]) [0.01518223 0.00499504 0.37376937 0.02905548]

/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning: 'data.DataLoader' is deprecated, use
  warnings.warn(out)
```

Результаты

Значение метрики AUC = 0.83

В ходе работы были изменены следующие гиперпараметры: количество сессий (50000->60000), количество эпох (5->12), скорость обучения (0.001->0.0015)

✓ 0 сек. выполнено в 13:05 ● ✕