Лабораторная работа №6:

"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.h
!pip install torch-geometric -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.htm
!pip install torch-scatter==2.0.8 -f https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
```

```
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-pack
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-wheel
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-cp
     |████████████████████████████████| 2.5 MB 41.7 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-wheel
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.
     |████████████████████████████████| 750 kB 44.0 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-wheel
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 31.0 MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loc
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/di
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.w
    Stored in directory: /root/.cache/pip/wheels/18/a6/a4/ca18c3051fcead866fe7b85700
  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-wheel
  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_x
    Stored in directory: /root/.cache/pip/wheels/96/e4/4e/2bcc6de6a801960aedbca43f71
  Successfully built torch-scatter
  Installing collected packages: torch-scatter
```

```
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/ "

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

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

```
# 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: DtypeW
  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 = 65000 #@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: 65000

```
session_id     65000
timestamp     362213
item_id        20034
category         122
dtype: int64
```

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

```
5.572723076923077
```

```
# Encode item and category id in item dataset so that ids will be in range (0,len(df.item.
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	3787	0
1	9	2014-04-06T11:28:54.654Z	3787	0
2	9	2014-04-06T11:29:13.479Z	3787	0
94	154	2014-04-03T08:59:07.398Z	14015	0
95	154	2014-04-03T09:00:18.944Z	16271	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: SettingWithCopyWarni
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/u
  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	5707	4711	1
46	489491	2014-04-06T12:41:34.047Z	13816	1046	4
47	489491	2014-04-06T12:41:34.091Z	13817	627	2
57	396	2014-04-06T17:53:45.147Z	14011	523	1
61	70353	2014-04-06T10:55:06.086Z	15642	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
```

```
 1648427: [14908, 13966, 14278, 14909, 12508, 10866],
 1649499: [5180],
 1650542: [13747],
 1650811: [14338],
 1651739: [13836, 14963],
 1653303: [8608, 8608],
```

```
 1655121: [13813],
 1655197: [14079,
  14053,
  12117,
  14304,
  13858,
  13966,
  14902,
  15660,
  14081,
  8354,
  14053,
  14304,
  14079,
  14081,
  13966,
  14902,
  12117,
  8354,
  15660,
  13858],
 1657604: [14375, 14370, 14377, 14744, 14320],
 1663779: [12412],
 1664362: [13338, 14223, 14274],
 1665372: [13966, 14909],
 1666988: [11230, 13639, 3414],
 1670197: [14338],
 1670831: [14278, 4275, 4275, 14278, 4275, 14278],
 1671752: [14909, 13966, 14278, 14192],
 1673383: [16391],
 1675762: [11995, 14200],
 1676516: [14269],
 1677874: [14693, 14278, 13803],
 1678726: [16295, 15929],
 1680706: [13966, 10866, 14284, 14908],
 1683503: [14288, 13866, 13953, 14288],
 1683519: [13966, 14909, 13960],
 1684999: [14908, 13966, 14882, 14908, 13966, 14882],
 1686589: [16500],
 1690711: [10929, 10929],
 1691882: [14231, 14220],
 1692087: [13796, 14235],
 1694197: [14909, 13966],
 1694476: [13836, 14017],
 1698484: [14909, 13966, 6683, 14283],
 1699464: [11385, 6809, 10058],
 1700783: [14760],
 1700801: [14908, 14909, 13954],
 1702029: [14909, 13966],
 1706378: [8956, 4486],
 1706896: [13688, 13679],
 1710366: [2613, 3056, 14083],
 1711881: [13839, 13973],
```

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('se
        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%|          | 0/65000 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipyk
100%|██████████| 65000/65000 [03:34<00:00, 303.17it/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)
```

```
(52000, 6500, 6500)
```

```
# 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: UserWarnin
  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
```

```
(20034, 121)
```

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=e
        self.category_embedding = torch.nn.Embedding(num_embeddings=num_categories, embedd
        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=0.001)
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 =   40#@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: {:.5
          format(epoch, loss, train_acc, val_acc, test_acc))
```

NUM_EPOCHS: 40

```
  2%|           | 1/40 [00:51<33:31, 51.57s/it]Epoch: 000, Loss: 0.68526, Train Auc:
  5%|           | 2/40 [01:39<31:08, 49.17s/it]Epoch: 001, Loss: 0.49604, Train Auc:
  8%|           | 3/40 [02:26<29:51, 48.42s/it]Epoch: 002, Loss: 0.39674, Train Auc:
 10%|           | 4/40 [03:14<28:57, 48.26s/it]Epoch: 003, Loss: 0.36389, Train Auc:
 12%|           | 5/40 [04:01<27:50, 47.73s/it]Epoch: 004, Loss: 0.33559, Train Auc:
 15%|           | 6/40 [04:48<26:51, 47.38s/it]Epoch: 005, Loss: 0.31933, Train Auc:
 18%|           | 7/40 [05:34<25:58, 47.22s/it]Epoch: 006, Loss: 0.30288, Train Auc:
 20%|           | 8/40 [06:21<25:07, 47.10s/it]Epoch: 007, Loss: 0.29189, Train Auc:
 22%|           | 9/40 [07:08<24:16, 46.97s/it]Epoch: 008, Loss: 0.28172, Train Auc:
 25%|           | 10/40 [07:55<23:26, 46.87s/it]Epoch: 009, Loss: 0.27004, Train Auc:
 28%|           | 11/40 [08:41<22:36, 46.78s/it]Epoch: 010, Loss: 0.25793, Train Auc
 30%|           | 12/40 [09:28<21:50, 46.81s/it]Epoch: 011, Loss: 0.25349, Train Auc:
 32%|           | 13/40 [10:14<21:00, 46.68s/it]Epoch: 012, Loss: 0.23775, Train Auc
 35%|           | 14/40 [11:01<20:14, 46.72s/it]Epoch: 013, Loss: 0.22150, Train Auc:
 38%|           | 15/40 [11:48<19:27, 46.68s/it]Epoch: 014, Loss: 0.21303, Train Auc
 40%|           | 16/40 [12:35<18:42, 46.77s/it]Epoch: 015, Loss: 0.20253, Train Auc:
 42%|           | 17/40 [13:22<17:56, 46.80s/it]Epoch: 016, Loss: 0.19527, Train Auc
 45%|           | 18/40 [14:08<17:07, 46.70s/it]Epoch: 017, Loss: 0.18689, Train Auc:
 48%|           | 19/40 [14:55<16:22, 46.81s/it]Epoch: 018, Loss: 0.18504, Train Auc
 50%|           | 20/40 [15:42<15:35, 46.79s/it]Epoch: 019, Loss: 0.18138, Train Auc:
 52%|           | 21/40 [16:29<14:49, 46.81s/it]Epoch: 020, Loss: 0.16871, Train Auc
 55%|           | 22/40 [17:16<14:01, 46.77s/it]Epoch: 021, Loss: 0.16117, Train Auc:
 57%|           | 23/40 [18:02<13:14, 46.71s/it]Epoch: 022, Loss: 0.15905, Train Auc
 60%|           | 24/40 [18:49<12:27, 46.70s/it]Epoch: 023, Loss: 0.15087, Train Auc:
 62%|           | 25/40 [19:36<11:40, 46.73s/it]Epoch: 024, Loss: 0.14731, Train Auc
 65%|           | 26/40 [20:22<10:53, 46.67s/it]Epoch: 025, Loss: 0.14913, Train Auc:
 68%|           | 27/40 [21:09<10:07, 46.72s/it]Epoch: 026, Loss: 0.15386, Train Auc
 70%|           | 28/40 [21:56<09:20, 46.74s/it]Epoch: 027, Loss: 0.15167, Train Auc:
 72%|           | 29/40 [22:43<08:35, 46.87s/it]Epoch: 028, Loss: 0.13762, Train Auc
 75%|           | 30/40 [23:30<07:47, 46.80s/it]Epoch: 029, Loss: 0.13520, Train Auc:
 78%|           | 31/40 [24:16<07:00, 46.70s/it]Epoch: 030, Loss: 0.12630, Train Auc
 80%|           | 32/40 [25:02<06:12, 46.59s/it]Epoch: 031, Loss: 0.12172, Train Auc:
 82%|           | 33/40 [25:49<05:26, 46.66s/it]Epoch: 032, Loss: 0.12220, Train Auc
 85%|           | 34/40 [26:36<04:39, 46.63s/it]Epoch: 033, Loss: 0.11961, Train Auc:
 88%|           | 35/40 [27:22<03:53, 46.63s/it]Epoch: 034, Loss: 0.12095, Train Auc
```

```
 90%|█████████ | 36/40 [28:09<03:06, 46.66s/it]Epoch: 035, Loss: 0.11968, Train Auc:
 92%|█████████▎| 37/40 [28:56<02:20, 46.77s/it]Epoch: 036, Loss: 0.11896, Train Auc
 95%|█████████▌| 38/40 [29:43<01:33, 46.77s/it]Epoch: 037, Loss: 0.11892, Train Auc:
 98%|█████████▊| 39/40 [30:29<00:46, 46.71s/it]Epoch: 038, Loss: 0.12062, Train Auc
100%|██████████| 40/40 [31:16<00:00, 46.91s/it]Epoch: 039, Loss: 0.11694, Train Auc:
```

Проверка результата с помощью примеров

```python
# 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: UserWarnin
  warnings.warn(out)
0.8620689655172413
```

```python
# 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, 245.73it/s]DataBatch(x=[1, 1, 2], edge_index=[2,
DataBatch(x=[3, 1, 2], edge_index=[2, 2], y=[3], batch=[3], ptr=[2]) [1.6608219e-08
DataBatch(x=[4, 1, 2], edge_index=[2, 3], y=[4], batch=[4], ptr=[2]) [3.0303802e-08

/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarnin
  warnings.warn(out)
```

✓ 0 сек. выполнено в 20:37