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

---

Установка библиотек, выгрузка исходных датасетов

```
import  torch
print(torch.__version__)
```

```
1.11.0+cu113
```

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

```
# Install pytorch geometric
!pip install gdown
!pip install torch

# !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.htm
# !pip install torch-geometric -f https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
# !pip install torch-scatter==2.0.9 -f https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html

!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
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (1.11.0+cu1
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Collecting torch-sparse
  Downloading https://data.pyg.org/whl/torch-1.11.0%2Bcu113/torch_sparse-0.6.13-cp37-cp37m-
     |████████████████████████████████████████| 3.5 MB 30.4 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from torch-
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (fr
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/
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-
     |████████████████████████████████████████| 2.5 MB 37.9 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/
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-cp
     |████████████████████████████████████████| 750 kB 28.0 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-wheels/public/
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 28.3 MB/s

Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from torch-g
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torch-
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from torch-
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from torch
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from torch
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from tor
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages (from to
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from py
```

```
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/py
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (fro
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=(
  Stored in directory: /root/.cache/pip/wheels/18/a6/a4/ca18c3051fcead866fe7b85700ee2240d8
Successfully built torch-geometric
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.0.4
```

```
!pip install torch-scatter -f https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/si
Looking in links: https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
Collecting torch-scatter
  Downloading https://data.pyg.org/whl/torch-1.11.0%2Bcu113/torch_scatter-2.0.9-cp37-cp37m-li
     |████████████████████████████████████████| 7.9 MB 17.5 MB/s
Installing collected packages: torch-scatter
Successfully installed torch-scatter-2.0.9
```

```
import numpy as np
import pandas as pd
import pickle
import csv
import os

from sklearn.preprocessing import LabelEncoder
# CUDA_LAUNCH_BLOCKING=1
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/drive/MyDrive/MMO/' #@param { type: "string" }
BASE_DIR = '/content/' #@param { type: "string" }
np.random.seed(RANDOM_SEED)


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

RANDOM_SEED: 42

BASE_DIR: "/content/"

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

```
# # Подключение к gogle диску
# from google.colab import drive
```

```
#  drive.mount('/content/drive')

#  #  Вывод  содержимого  папки  на  диске
#  import  os
#  data_root  =  '/content/drive/MyDrive/MMO'
#  print(os.listdir(data_root))


!gdown  --id  1JMt9TtWFw6Hosy1aLAtNtDoEdCiAlR87
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was dep
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1JMt9TtWFw6Hosy1aLAtNtDoEdCiAlR87
To: /content/yoochoose-data-lite.zip
100% 49.8M/49.8M [00:00<00:00, 104MB/s]
```

```
#  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()
```

```
#  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()
```

```
# 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(d
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()
```

```
# 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()
```

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

```
1759453: [13523],
1762533: [9003],
1763361: [13786, 2492],
1764317: [10619, 8670, 3851, 3790, 3376, 10279, 12453, 2569],
1764946: [13755, 11213, 14448],
1767864: [13936, 13756, 9205, 13935],
1775386: [13537, 13372],
1776397: [14451, 13390, 13585],
1780132: [14290],
1782183: [13848, 13523, 14450, 13934],
1783711: [3960],
1787382: [13529, 13438, 14450, 14450, 14451, 13535, 13840],
1787867: [14292, 13644, 13522, 13933, 13934],
1789289: [13397, 13811, 3429],
1789521: [3082, 13408, 2573],
1792339: [13529, 13529],
1794516: [12933],
1798914: [14241, 7063],
1799584: [8665],
1802618: [13132],
1803716: [13883, 13883],
1807291: [8648, 13754],
1807804: [14726, 14726],
1808092: [13832],
1809431: [4387, 13939, 13932, 14291],
1815429: [7698, 14457],
1817108: [13144],
1819732: [14496],
1822064: [259, 9088],
1824159: [13408, 13897],
1825758: [13934, 12521],
1826954: [13250]
```

```
1828084: [13649, 13647, 13650],
1829906: [13401],
1830611: [13261],
1831466: [12569, 12509, 14073, 12509, 9218],
1832139: [2423, 188],
1832963: [13752, 14496, 14292],
1835453: [13932],
1838421: [6106, 6107],
1841093: [55],
1845774: [13940, 13845],
1847923: [13944, 14538],
1848376: [13942, 13940],
1851472: [3688, 1625],
1853764: [2227],
1856377: [2078, 11787, 2078, 11787],
1857674: [13933, 13934],
1858184: [3750, 11319, 6420, 442],
1859963: [11288, 11288, 11291, 3913, 11288],
1861698: [13935, 9289, 9231, 9235],
1861751: [13385],
1862619: [13751, 11916, 14448],
1866224: [14291, 13864, 13558, 13841, 13842],
1869789: [13307, 13393, 14451],
1869871: [6959, 741, 2172, 10448],
1872286: [13580, 13882],
...}
```

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','
        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_lau
100%|██████████| 60000/60000 [03:17<00:00, 303.21it/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.
  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
        self.category_embedding = torch.nn.Embedding(num_embeddings=num_categories, embe
        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 = 10 #@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)
```

NUM_EPOCHS: 10

```
    print('Epoch: {:03d}, Loss: {:.5f}, Train Auc: {:.5f}, Val Auc: {:.5f}, Test Auc
                format(epoch, loss, train_acc, val_acc, test_acc))
```

```
 10%|█         | 1/10 [00:47<07:07, 47.46s/it]Epoch: 000, Loss: 0.70469, Train Auc: 0.52358,
 20%|██        | 2/10 [01:31<06:01, 45.20s/it]Epoch: 001, Loss: 0.64089, Train Auc: 0.54819
 30%|███       | 3/10 [02:14<05:11, 44.53s/it]Epoch: 002, Loss: 0.58818, Train Auc: 0.5527
 40%|████      | 4/10 [02:57<04:23, 43.91s/it]Epoch: 003, Loss: 0.55480, Train Auc: 0.577
 50%|█████     | 5/10 [03:40<03:38, 43.67s/it]Epoch: 004, Loss: 0.51879, Train Auc: 0.59
 60%|██████    | 6/10 [04:24<02:54, 43.54s/it]Epoch: 005, Loss: 0.47351, Train Auc: 0.6
 70%|███████   | 7/10 [05:08<02:11, 43.71s/it]Epoch: 006, Loss: 0.43985, Train Auc: 0.
 80%|████████  | 8/10 [05:51<01:27, 43.65s/it]Epoch: 007, Loss: 0.41514, Train Auc: 0
 90%|█████████ | 9/10 [06:35<00:43, 43.57s/it]Epoch: 008, Loss: 0.38756, Train Auc:
100%|██████████| 10/10 [07:18<00:00, 43.85s/it]Epoch: 009, Loss: 0.36523, Train Auc
```

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.
  warnings.warn(out)
0.34782608695652173
```

```
# 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, 232.70it/s]DataBatch(x=[1, 1, 2], edge_index=[2,
DataBatch(x=[3, 1, 2], edge_index=[2, 2], y=[3], batch=[3], ptr=[2]) [0.4110777  0.35444528 0
DataBatch(x=[4, 1, 2], edge_index=[2, 3], y=[4], batch=[4], ptr=[2]) [0.09336587 0.04997347 0
```

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

Выводы

Как видно из результатов, значение метрики AUC = 74.75%.

В ходе работы были изменены следующие гиперпараметры: количество эпох (5->10), скорость обучение (0.001->0.0015), количество сессий (50000->60000).

✓ 0 秒 完成时间：16:58