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==2.0.9 -f https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
```

```
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Requirement already satisfied: torch-sparse in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-package
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Requirement already satisfied: torch-cluster in /usr/local/lib/python3.7/dist-package
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Requirement already satisfied: torch-spline-conv in /usr/local/lib/python3.7/dist-pa
Looking in links: https://pytorch-geometric.com/whl/torch-1.11.0%2Bcu113.html
Requirement already satisfied: torch-geometric in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/di
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist
Looking in links: https://data.pyg.org/whl/torch-1.11.0%2Bcu113.html
Requirement already satisfied: torch-scatter==2.0.9 in /usr/local/lib/python3.7/dist
```

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

from sklearn.preprocessing import LabelEncoder

import torch
```

RANDOM_SEED: 17

BASE_DIR: " /content/ "

```
import torch

# PyG - PyTorch Geometric
from torch_geometric.data import Data, DataLoader, InMemoryDataset

from tqdm import tqdm
```

```
RANDOM_SEED = 17 #@param { type: "integer" }
BASE_DIR = '/content/' #@param { type: "string" }
np.random.seed(RANDOM_SEED)
```

```
# 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: DtypeWa
       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

```
# 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     334990
item_id        20043
category         103
dtype: int64
```

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

```
5.583416666666665
```

```
# 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
91	131	2014-04-03T04:46:08.891Z	13649	0
92	131	2014-04-03T04:46:53.499Z	13445	0
93	131	2014-04-03T04:47:32.085Z	13585	0
177	309	2014-04-06T07:59:23.727Z	14064	0
178	309	2014-04-06T08:02:02.034Z	15547	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: SettingWithCopyWarnir
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/us
  This is separate from the ipykernel package so we can avoid doing imports until
```

	session_id	timestamp	item_id	price	quantity
5	70427	2014-04-02T15:54:07.144Z	13729	3769	1
25	140964	2014-04-04T07:02:02.655Z	10268	2408	1
62	489671	2014-04-03T15:48:37.392Z	13710	4188	1
63	489671	2014-04-03T15:59:35.495Z	13710	4188	1
64	489671	2014-04-03T16:00:06.917Z	13710	4188	1

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

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%|          | 0/60000 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipyke
100%|██████████| 60000/60000 [05:05<00:00, 196.46it/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
  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
```

```
(20043, 102)
```

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=0.002)
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: {:.5
          format(epoch, loss, train_acc, val_acc, test_acc))

     10%|█         | 1/10 [01:23<12:35, 83.94s/it]Epoch: 000, Loss: 0.64618, Train Auc:
     20%|██        | 2/10 [02:41<10:39, 79.99s/it]Epoch: 001, Loss: 0.45930, Train Auc:
     30%|███       | 3/10 [03:57<09:08, 78.35s/it]Epoch: 002, Loss: 0.40134, Train Auc:
     40%|████      | 4/10 [05:13<07:45, 77.51s/it]Epoch: 003, Loss: 0.36662, Train Auc:
     50%|█████     | 5/10 [06:28<06:23, 76.66s/it]Epoch: 004, Loss: 0.34265, Train Auc:
     60%|██████    | 6/10 [07:44<05:04, 76.17s/it]Epoch: 005, Loss: 0.33223, Train Auc:
     70%|███████   | 7/10 [09:00<03:48, 76.07s/it]Epoch: 006, Loss: 0.30576, Train Auc:
     80%|████████  | 8/10 [10:15<02:31, 75.91s/it]Epoch: 007, Loss: 0.28174, Train Auc:
     90%|█████████ | 9/10 [11:30<01:15, 75.63s/it]Epoch: 008, Loss: 0.26621, Train Auc:
    100%|██████████| 10/10 [12:45<00:00, 76.55s/it]Epoch: 009, Loss: 0.24922, Train Auc:
```

Проверка результата с помощью примеров

```
# Подход №1 - из датасета
evaluate(DataLoader(test_dataset[25:45], batch_size=10))

     /usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:12: UserWarning
       warnings.warn(out)
     0.7247191011235956
```

```
# 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, 183.19it/s]DataBatch(x=[1, 1, 2], edge_index=[2,
     DataBatch(x=[3, 1, 2], edge_index=[2, 2], y=[3], batch=[3], ptr=[2]) [0.00379266 0.0
     DataBatch(x=[4, 1, 2], edge_index=[2, 3], y=[4], batch=[4], ptr=[2]) [4.1785872e-05
```

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

Как видно из результатов, значение метрики *AUC = 72.5%*

В ходе работы были изменены следующие гиперпараметры: количество эпох (5->10), скорость обучение (0.001->0.002), количество сессий (50000->60000)