Higher School of Economics

Faculty of Computer Science

Financial Technologies and Data Analysis

Relation Prediction in Knowledge Graphs

Выполнили:

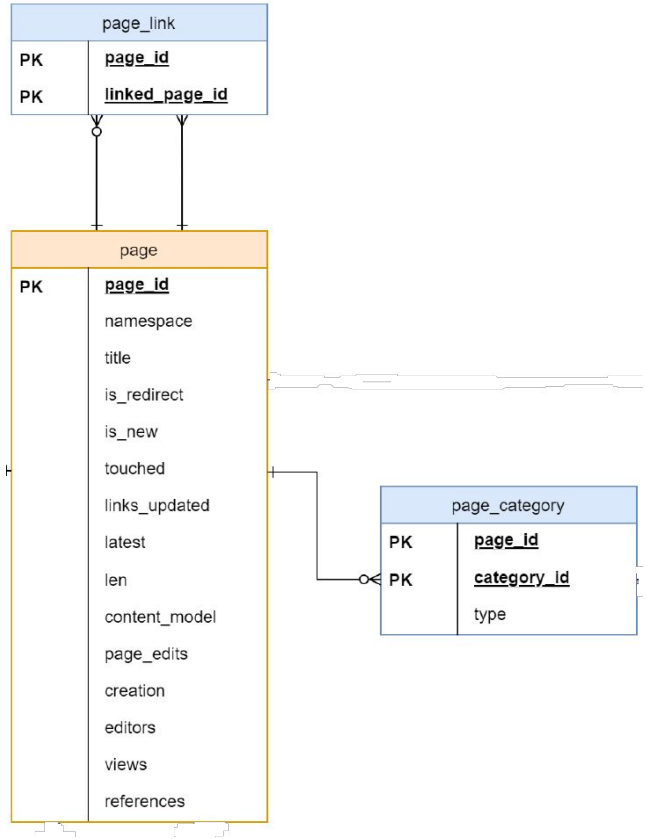
Азевич Марк
Озерова Дарья
Никита Лобачев

Moscow, 2024

Цель работы

Наша задача состояла в прогнозировании связующих ребер в гетерогенных графах. Цель состоит в том, чтобы предсказать недостающие связи между узлами в графе знаний.

В пайплайне ML мы пройдем следующие шаги:
1. Загрузка и предварительная обработка данных. Мы загрузим набор данных графа знаний в PyG и предварительно обработаем его для создания узловых и реберных представлений.
2. Определение модели. Мы определим архитектуру модели GCN или GAT, используя встроенные классы PyG для графовых нейронных сетей.
3. Обучение и оценка. Мы обучим модель на данных графа знаний и оценим ее производительность с использованием таких показателей, как ROC AUC и Accuracy.
4. Настройка гиперпараметров. Мы рассмотрим различные гиперпараметры и методы оптимизации, чтобы улучшить производительность модели.
5. Вывод. Наконец, мы будем использовать обученную модель для прогнозирования невидимых данных и анализа результатов.



Предобработка данных

Wikipedia Knowledge Graph dataset

Table	Dimension	Size
page	53,710,529 x 15	5.47 GB
page_link	566,536,991 x 2	9.36 GB
page_category	165,501,704 x 3	3.52 GB

https://zenodo.org/records/6346900

Предобработка данных

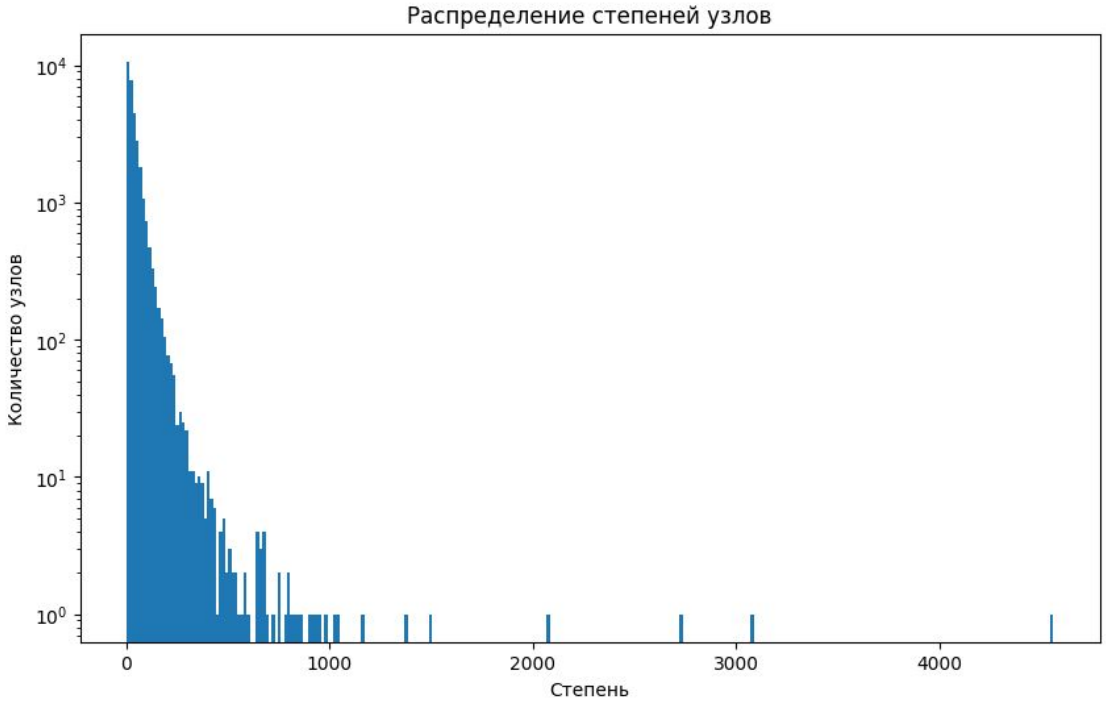
К датасету были применены следующие фильтры:

1. *Больше 200 символов в статье;*
2. *Больше 100 000 просмотров у статьи;*
3. *Из категорий отобрали только самые популярные, к которым относятся более 100 000 страниц, таких получилось около 70.*

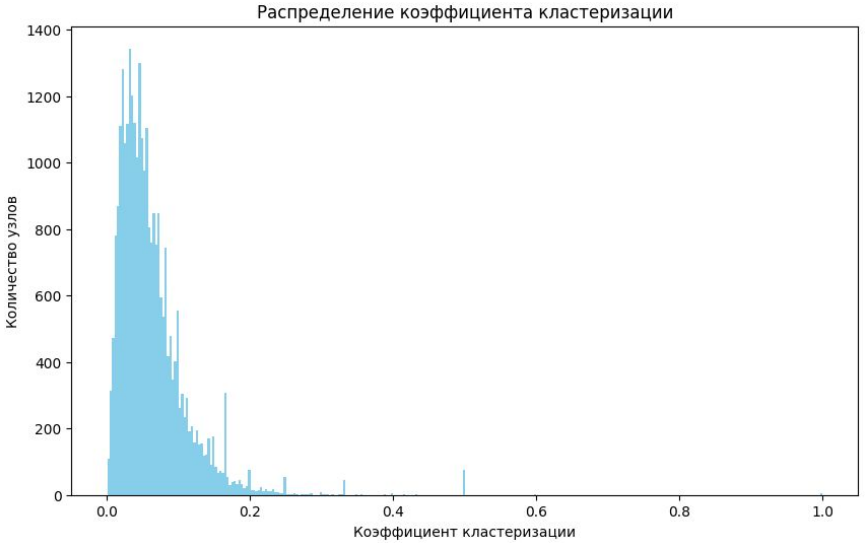
Принадлежность страницы к категориям внесли отдельными бинарными столбцами, так как одна страница может относиться сразу к нескольким категориям

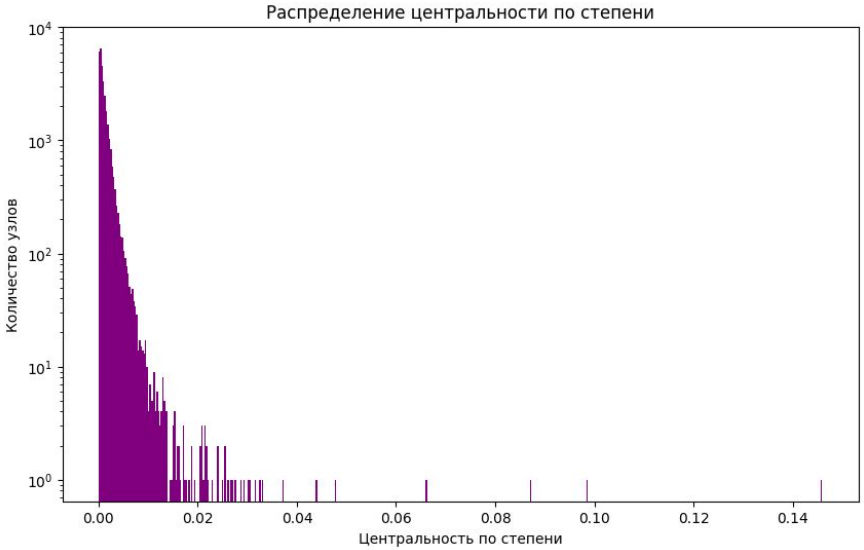
Характеристики получившегося графа

- Количество узлов: 31251
- Количество рёбер: 608092
- Плотность графа: 0.00062
- Средняя степень узлов: 38.2



Характеристики получившегося графа

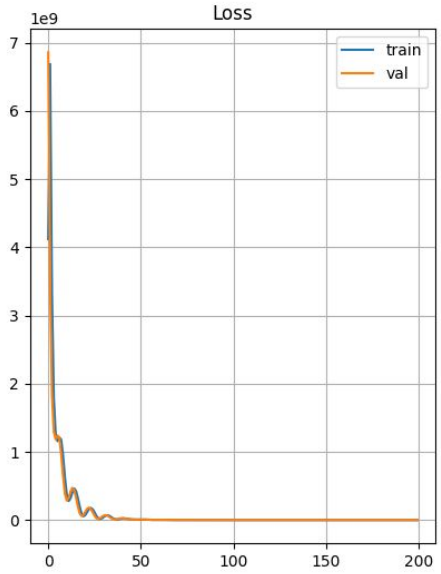


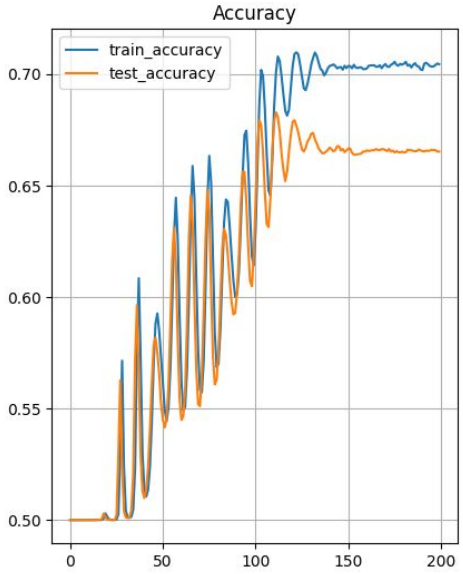


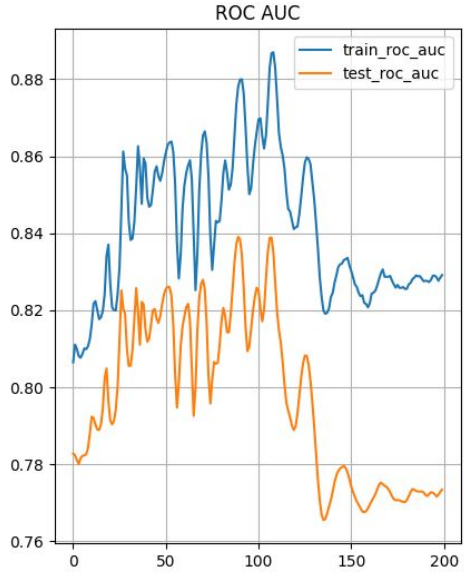
Модель - GCN

```
class GCN_v2(torch.nn.Module):
    def __init__(self):
        super(GCN_v2, self).__init__()
        self.conv1 = GCNConv(data.num_node_features, 512)
        self.conv2 = GCNConv(512, 1024)
        self.prelu = PReLU(1024)
        self.conv3 = GCNConv(1024, 64)

    def encode(self):
        x = self.conv1(data.x, data.train_pos_edge_index)
        x = self.conv2(x, data.train_pos_edge_index)
        x = self.prelu(x)
        x = self.conv3(x, data.train_pos_edge_index)
        return x

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)
        return logits
```

Модель - GCN


Loss
1e9
train
val

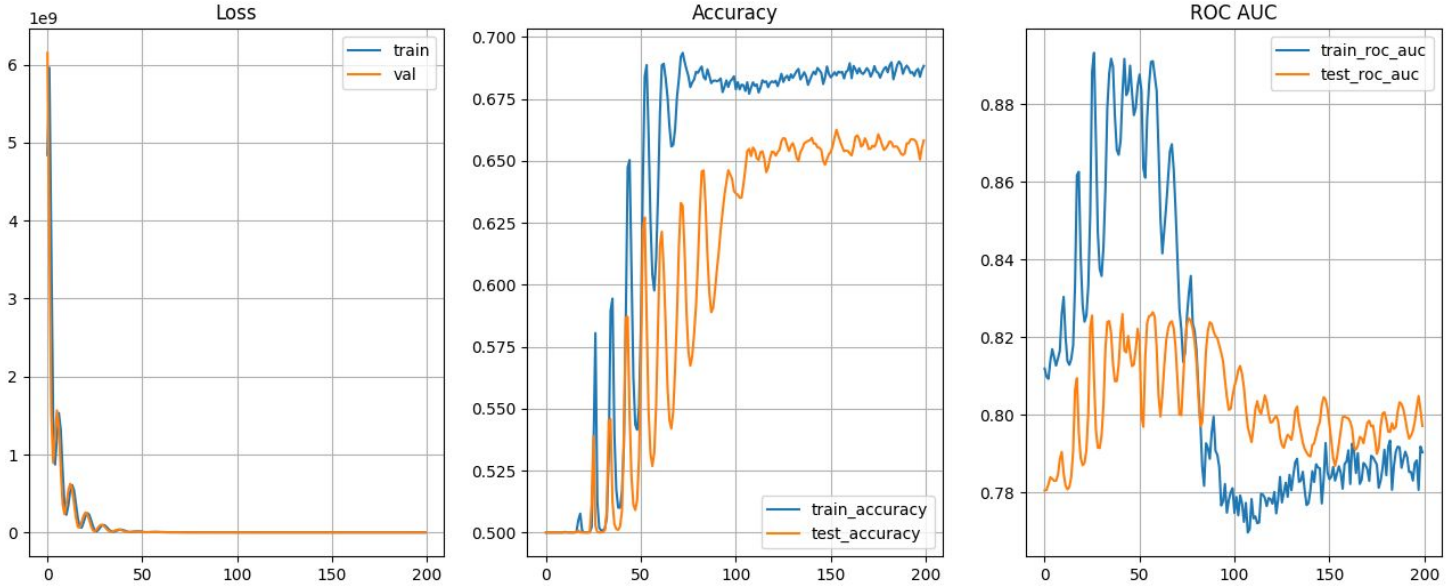

Accuracy
train_accuracy
test_accuracy


ROC AUC
train_roc_auc
test_roc_auc

GCN + dropout

```
class GCN_v3(torch.nn.Module):
    def __init__(self):
        super(GCN_v3, self).__init__()
        self.conv1 = GCNConv(data.num_node_features, 512)
        self.conv2 = GCNConv(512, 1024)
        self.prelu = PReLU(1024)
        self.dropout = torch.nn.Dropout(p=0.2)
        self.conv3 = GCNConv(1024, 64)


    def encode(self):
        x = self.conv1(data.x, data.train_pos_edge_index)
        x = self.conv2(x, data.train_pos_edge_index)
        x = self.prelu(x)
        x = self.dropout(x)
        x = self.conv3(x, data.train_pos_edge_index)
        return x

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)
        return logits
```

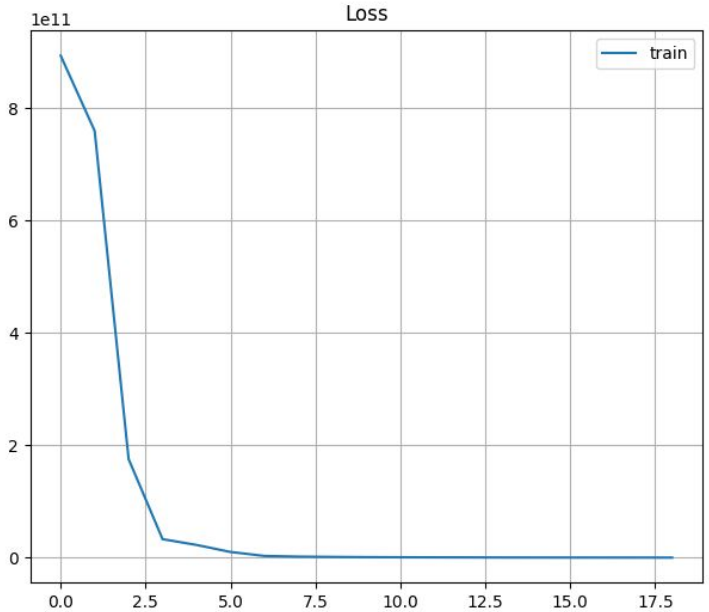
GCN + dropout

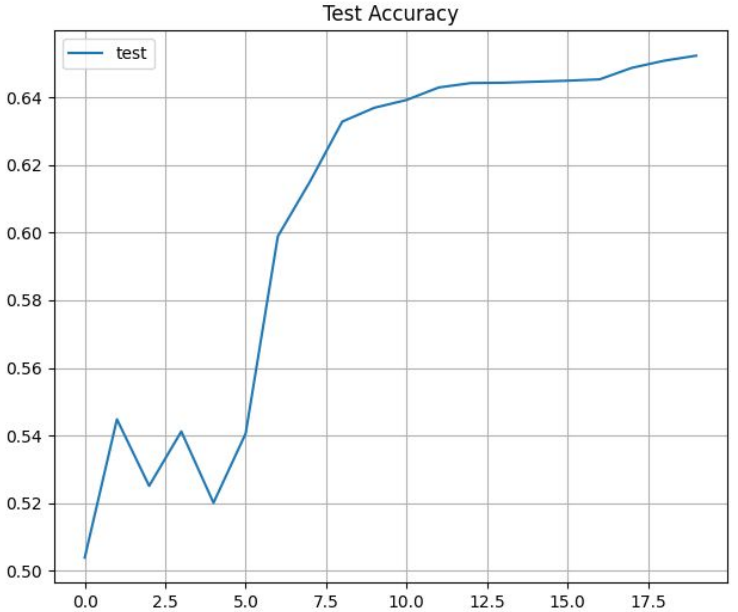

Loss
1e9
train
val
Accuracy
train_accuracy
test_accuracy
ROC AUC
train_roc_auc
test_roc_auc

Cheb

```
class Cheb(torch.nn.Module):
    def __init__(self):
        super(Cheb, self).__init__()
        self.conv0 = ChebConv(data.num_node_features, 512, K=2)
        self.conv1 = ChebConv(512, 1024, K=2)
        self.conv2 = ChebConv(1024, 64, K=2)
        self.prelu = PReLU(1024)

    def encode(self):
        x = self.conv0(data.x, data.train_pos_edge_index)
        x = self.conv1(x, data.train_pos_edge_index)
        x = self.prelu(x)
        x = self.conv2(x, data.train_pos_edge_index)
        return x

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)
        return logits
```

Cheb

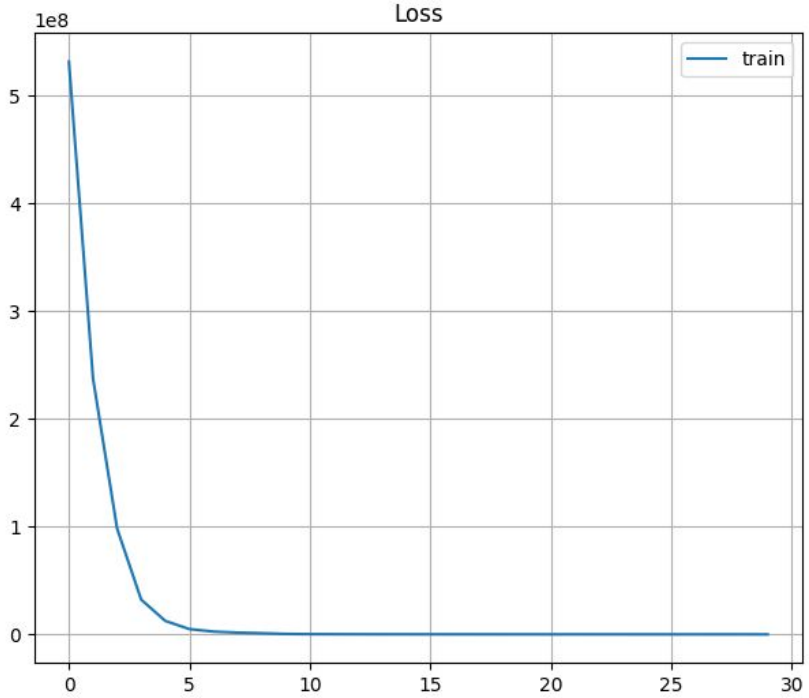

Loss
1e11
train
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

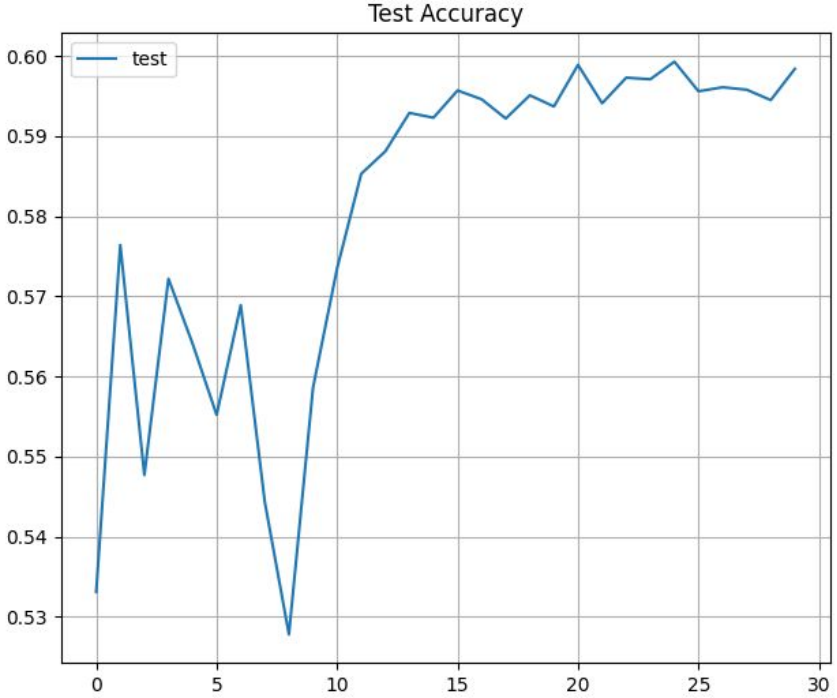

Test Accuracy
test
0.64
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Gat

```
class GAT(torch.nn.Module):
    def __init__(self):
        super(GAT, self).__init__()
        self.conv1 = GATConv(data.num_node_features, 128)
        self.conv2 = GATConv(128, 64)
        self.prelu = PReLU(128)

    def encode(self):
        x = self.conv1(data.x, data.train_pos_edge_index)
        x = self.prelu(x)
        x = self.conv2(x, data.train_pos_edge_index)
        return x

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)
        return logits
```

Gat


Loss
1e8
train
5
4
3
2
1
0
0
5
10
15
20
25
30


Test Accuracy
test
0.60
0.59
0.58
0.57
0.56
0.55
0.54
0.53
0
5
10
15
20
25
30

Transformer

```
class Transformer(torch.nn.Module):
    def __init__(self):
        super(Transformer, self).__init__()
        self.conv0 = TransformerConv(data.num_node_features, 512)
        self.conv1 = TransformerConv(512, 1024)
        self.conv2 = ChebConv(1024, 512, K=3)
        self.prelu = PReLU(1024)
        self.lin = nn.Linear(512, 64)

    def encode(self):
        x = self.conv0(data.x, data.train_pos_edge_index)
        x = self.conv1(x, data.train_pos_edge_index)
        x = self.prelu(x)
        x = self.conv2(x, data.train_pos_edge_index)
        x = self.lin(x)
        return x

    def decode(self, z, pos_edge_index, neg_edge_index):
        edge_index = torch.cat([pos_edge_index, neg_edge_index], dim=-1)
        logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)
        return logits
```

Transformer