基于 Transformer 的学生行为预测实验报告

学号：2024103442　　姓名：杨海杰　　导师：赵晓薇　研究方向：教育数据分析

1. 实验部分

1.1 模型描述

本文提出基于 Transformer 的模型。通过学习节点嵌入表示来预测下游连接符号预测任务。核心结构包含：

1. 使用可学习的正负度向量优化节点嵌入：量化节点的正 / 负度数对其重要性的影响，弥补传统 GNN 忽略度数信息的缺陷。为每个节点定义两个可学习的嵌入向量：正度数嵌入 c^+和负度数嵌入 c^-。将度数嵌入与节点初始特征相加，生成初始节点表示：$h_i^{(0)} = x_i + c^+_{\deg^+(v_i)} + c^-_{\deg^-(v_i)}$

2. 邻接矩阵编码：将图的一阶邻接结构（直接正 / 负邻居）融入自注意力机制，增强局部结构建模能力。对邻接矩阵 A 进行归一化，得到 $\hat{A}$，作为自注意力计算中的偏置项：$\overline{A}_{ij} = \frac{(h_iW_Q)(h_jW_K)^T}{\sqrt{d}} + \widehat{A_{ij}}$

3. Transformer 层结构：采用标准 Transformer 的多头自注意力（MHA）和前馈神经网络（FFN），并引入层归一化（LN）和残差连接。

核心公式：$Q = H^WQ$, $K = H^WK$, $V = H^WV$,

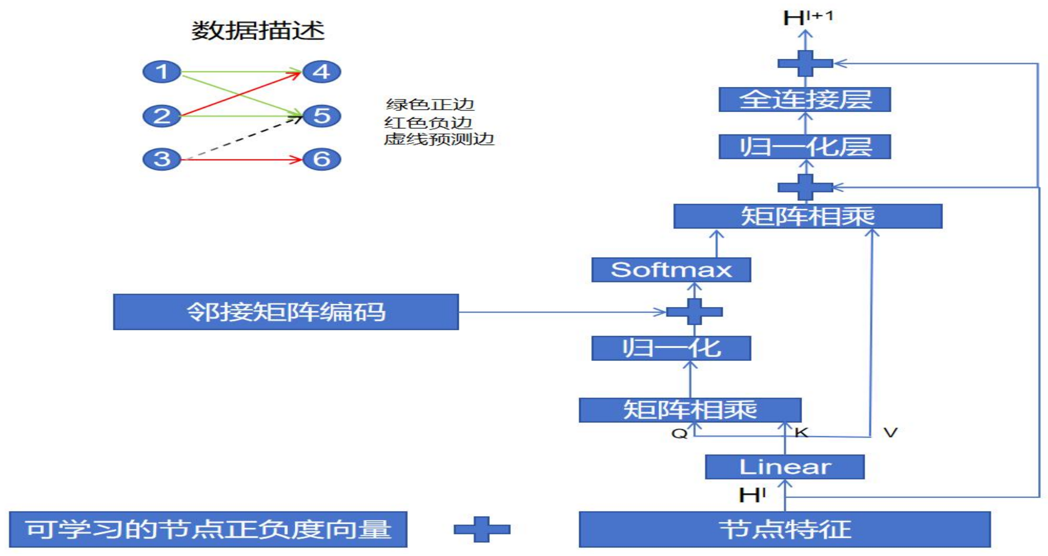
$h'(l)=MHA(Attn(h(l-1)))+h(l-1)$

$h(l)=FFN(LN(h'(l)))+h'(l)$

4. 边符号预测：预测节点间链接的正负号。

1.2 模型图与参数

模型结构图：



模型图左上为数据实例，表示两个不同的实体之间的关联。右侧为经典的 Transformer 框架，为了融入图数据的结构性，嵌入了归一化后的邻接矩阵。
经运行代码打印的真实模型参数详细统计:

参数名称	形状	参数数量	是否可训练
node_in_lin.weight	[128, 128]	16,384	True
node_in_lin.bias	[128]	128	True
centrality_encoding.z_pos	[12, 128]	1,536	True
centrality_encoding.z_neg	[12, 128]	1,536	True
layers.0.attention.heads.n.q.weight n~(0,3)	[128, 128]	16,384*4	True
layers.0.attention.heads.n.q.bias n~(0,3)	[128]	128 *4	True
layers.0.attention.heads.n.k.weight n~(0,3)	[128, 128]	16,384*4	True
layers.0.attention.heads.n.k.bias n~(0,3)	[128]	128 *4	True
layers.0.attention.heads.n.v.weight n~(0,3)	[128, 128]	16,384*4	True
layers.0.attention.heads.n.v.bias n~(0,3)	[128]	128 *4	True
layers.0.ln_n.weight n~(1,2)	[128]	128*2	True
layers.0.ln_n.bias n~(1,2)	[128]	128*2	True
layers.0.ff.weight	[128, 128]	16,384	True
layers.0.ff.bias	[128]	128	True
lin.weight	[3, 256]	768	True
lin.bias	[3]	3	True
node_out_lin.weight	[128, 128]	16,384	True
node_out_lin.bias	[128]	128	True

按模块划分的参数数量:

layers	280,832
node_in_lin	16,512
node_out_lin	16,512
centrality_encoding	3,072
lin	771

总参数量: 317,699
可训练参数量: 317,699

1.3 测试数据集与评价标准

1.数据集介绍：

如下图所示，第一列为学生 id,第二列表示题目 id,第三列为学生在相应题目上的得分，满分 5 分，实验设置以 4 为域值，小于 4 代表学生未掌握此题，则为

负边；大于等于 4 代表学生已经掌握此题，则为正边；
其中详细数据如下：
训练集：39,901 条
测试集：9,974 条

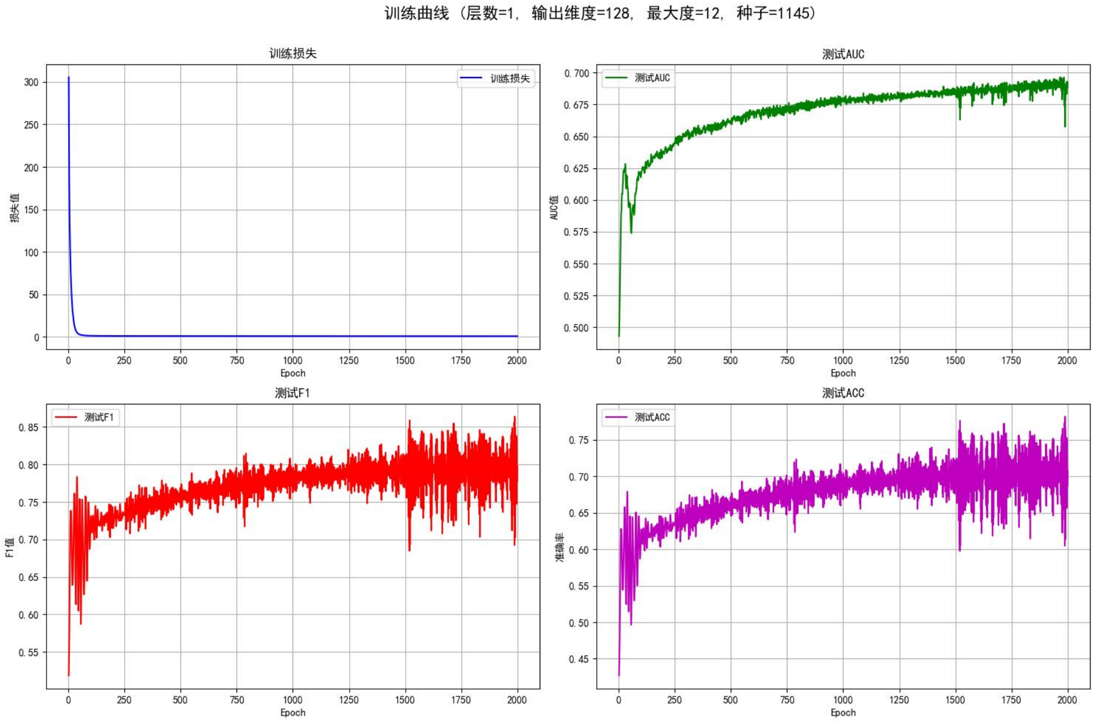
```
15 72 4.00
1073 1895 4.00
1987 1580 4.00
2076 435 4.00
360 226 4.00
621 1216 4.00
659 287 4.00
701 352 5.00
227 638 5.00
905 1279 5.00
237 634 5.00
39 1882 5.00
949 1190 5.00
439 905 5.00
1675 1605 5.00
```

2. 评价标准
经过训练后，在模型最优参数下测试的指标如下：
准确率（Accuracy）：0.6896
AUC：0.7422

1.4 实验环境与结果

1.环境：
Python=3.9.12
torch = 2.3.1+cuda11.8
numpy=1.26.4
scikit-learn=1.5.1
pandas=2.2.2
tqdm=4.64.0
scipy=1.10.1
torch-cluster=1.6.3
torch-scatter=2.1.2
torch-sparse=0.6.18
torch-spline-conv=1.2.2
torch_geometric=2.5.3
torchvision=0.18.1
torchaudio=2.3.1
3. 实验结果
(1) 性能数据
{"dataset": "data", "num_layers": 1, "output_dim": 128, "max_degree": 12, "mean_auc": 0.6896101101340166, "mean_acc": 0.7422297974734309, "total_time": 218.4356291294098, "auc_values": [0.6896101101340166], "acc_values": [0.7422297974734309]}

(2) 训练曲线



2. 核心代码实现

2.1 使用可学习的正负度向量优化节点嵌入：

```
class CentralityEncoding(nn.Module):
    def __init__(self, max_degree: int,   node_dim: int):
                super().__init__()
        self.max_degree = max_degree
        self.node_dim = node_dim
        self.z_pos = nn.Parameter(torch.randn((max_degree, node_dim)))
        self.z_neg = nn.Parameter(torch.randn((max_degree, node_dim)))

    def forward(self, x: torch.Tensor, pos_edge_index: torch.Tensor, neg_edge_index:
torch.tensor) -> torch.Tensor:
        num_nodes = x.shape[0]
        positive_degrees=torch.bincount(pos_edge_index[0],
minlength=num_nodes)
        negative_degrees = torch.bincount(neg_edge_index[0],
minlength=num_nodes)
        positive_degrees = self.decrease_to_max_value(positive_degrees,
self.max_degree - 1)
        negative_degrees = self.decrease_to_max_value(negative_degrees,
self.max_degree - 1)
        x += self.z_pos[positive_degrees] + self.z_neg[negative_degrees]
        return x
```

```
    def decrease_to_max_value(self, x, max_value):
        x[x > max_value] = max_value
        return x
```

2.2 邻接矩阵编码：

```
class EdgeEncoding(nn.Module):
    def __init__(self):
                super().__init__()
    def forward(self, pos_edge_index: Tensor, neg_edge_index: Tensor, num_nodes: torch.tensor) -> torch.Tensor:adj_matrix = torch.zeros((num_nodes, num_nodes), dtype=torch.float)
        adj_matrix[pos_edge_index[0], pos_edge_index[1]] = 1
        adj_matrix[pos_edge_index[1], pos_edge_index[0]] = 1

        adj_matrix[neg_edge_index[0], neg_edge_index[1]] = -1
        adj_matrix[neg_edge_index[1], neg_edge_index[0]] = -1

        row_sum = adj_matrix.sum(dim=1, keepdim=True)
        epsilon = 1e-10
        normalized_adj_matrix = adj_matrix / (row_sum + epsilon)
        return normalized_adj_matrix
```

3. Transformer 层结构（从下向上调用）：

```
class GraphormerAttentionHead(nn.Module):
    def __init__(self, dim_in: int, dim_q: int, dim_k: int):
                super().__init__()
        self.q = nn.Linear(dim_in, dim_q)
        self.k = nn.Linear(dim_in, dim_k)
        self.v = nn.Linear(dim_in, dim_k)
    def forward(self,x: torch.Tensor,adj_matrix: torch.Tensor) -> torch.Tensor:
        query = self.q(x)
        key = self.k(x)
        value = self.v(x)
        a = self.compute_a(key, query)
        a = a + adj_matrix
        softmax = torch.softmax(a, dim=-1)
        x = softmax.mm(value)
        return x

    def compute_a(self, key, query):
        a = query.mm(key.transpose(0, 1)) / query.size(-1) ** 0.5
        return a

class GraphormerMultiHeadAttention(nn.Module):
```

```
    def __init__(self, num_heads: int, dim_in: int, dim_q: int, dim_k: int):
            super().__init__()
        self.heads = nn.ModuleList(
            [GraphormerAttentionHead(dim_in, dim_q, dim_k) for _ in range(num_heads)]
        )
        self.linear = nn.Linear(num_heads * dim_k, dim_in)

    def forward(self, x: torch.Tensor,adj_matrix: torch.Tensor) -> torch.Tensor:
            return self.linear(torch.cat([attention_head(x, adj_matrix) for attention_head in self.heads], dim=-1))


class GraphormerEncoderLayer(nn.Module):
    def __init__(self, node_dim, num_heads):
            super().__init__()

        self.node_dim = node_dim
        self.num_heads = num_heads
        self.attention = GraphormerMultiHeadAttention(
            dim_in=node_dim,
            dim_k=node_dim,
            dim_q=node_dim,
            num_heads=num_heads)
        self.ln_1 = nn.LayerNorm(node_dim)
        self.ln_2 = nn.LayerNorm(node_dim)
        self.ff = nn.Linear(node_dim, node_dim)

    def forward(self,x: torch.Tensor,adj_matrix: torch.Tensor,) -> Tuple[torch.Tensor, torch.Tensor]:
        x_prime = self.attention(self.ln_1(x), adj_matrix) + x
        x_new = self.ff(self.ln_2(x_prime)) + x_prime
        return x_new
```

3．训练和测试：

```
def train(model, optimizer, x, train_pos_edge_index, train_neg_edge_index, dataset_name, i):
    model.train()
    optimizer.zero_grad()
    z = model(x, train_pos_edge_index, train_neg_edge_index, dataset_name, i)
    loss = model.loss(z, train_pos_edge_index, train_neg_edge_index)
    loss.backward()
    optimizer.step()
    return loss.item()
```

```
def test(model, x, train_pos_edge_index, train_neg_edge_index,
test_pos_edge_index, test_neg_edge_index, dataset_name, i):
    model.eval()
    with torch.no_grad():
        z = model(x, train_pos_edge_index, train_neg_edge_index, dataset_name,
i)
    return model.test(z, test_pos_edge_index, test_neg_edge_index)
```

4. 训练曲线

```
def plot_training_curve(history, args, dataset_name, seed):

    epochs = range(1, len(history['loss']) + 1)

    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    fig.suptitle(
        f'{dataset_name} 训练曲线 (层数={args.num_layers}, 输出维度
={args.output_dim}, 最大度={args.max_degree}, 种子={seed})',
        fontsize=16)

    axes[0, 0].plot(epochs, history['loss'], 'b-', label='训练损失')
    axes[0, 0].set_title('训练损失')
    axes[0, 0].set_xlabel('Epoch')
    axes[0, 0].set_ylabel('损失值')
    axes[0, 0].grid(True)
    axes[0, 0].legend()

    axes[0, 1].plot(epochs, history['auc'], 'g-', label='测试 AUC')
    axes[0, 1].set_title('测试 AUC')
    axes[0, 1].set_xlabel('Epoch')
    axes[0, 1].set_ylabel('AUC 值')
    axes[0, 1].grid(True)
    axes[0, 1].legend()

    axes[1, 0].plot(epochs, history['f1'], 'r-', label='测试 F1')
    axes[1, 0].set_title('测试 F1')
    axes[1, 0].set_xlabel('Epoch')
    axes[1, 0].set_ylabel('F1 值')
    axes[1, 0].grid(True)
    axes[1, 0].legend()

    axes[1, 1].plot(epochs, history['acc'], 'm-', label='测试 ACC')
    axes[1, 1].set_title('测试 ACC')
    axes[1, 1].set_xlabel('Epoch')
    axes[1, 1].set_ylabel('准确率')
    axes[1, 1].grid(True)
```

```
        axes[1, 1].legend()

        plt.tight_layout()
        plt.subplots_adjust(top=0.9)

        os.makedirs('./training_curves', exist_ok=True)
        filename = 
f"./training_curves/{dataset_name}_{args.num_layers}_{args.output_dim}_{args.max
_degree}_{seed}.png"
        plt.savefig(filename)
        print(f"训练曲线已保存至: {filename}")
        plt.close()
```