实验报告-实验三

摘要

摘要：简要介绍要解决的问题，所使用的方法步骤，取得的结果或结论。

关键字： 1　2　3

一、 导言

车辆路径规划是现代交通的一项重要任务。优化车辆路线不仅可以大大降低运营成本，同时也能够提高客户满意度。然而，车辆路径问题（VRP 问题，即 Vehicle Routing Problem）为 NP 难问题，在有效解决大规模问题上仍具有十足的挑战性。在论文 [1] 中，作者提出了一种新颖的深度学习模型，其通过将点和边的信息进行嵌入，然后通过结合强化学习和分类任务的方法来学习生成车辆路线安排的启发式策略，本实验的目的即为复现其论文中使用的方法和结果。

二、 实验过程

2.1 总体结构

结合论文 [1] 中的思想，本次实验我们小组所实现的模型结构示意图如下：

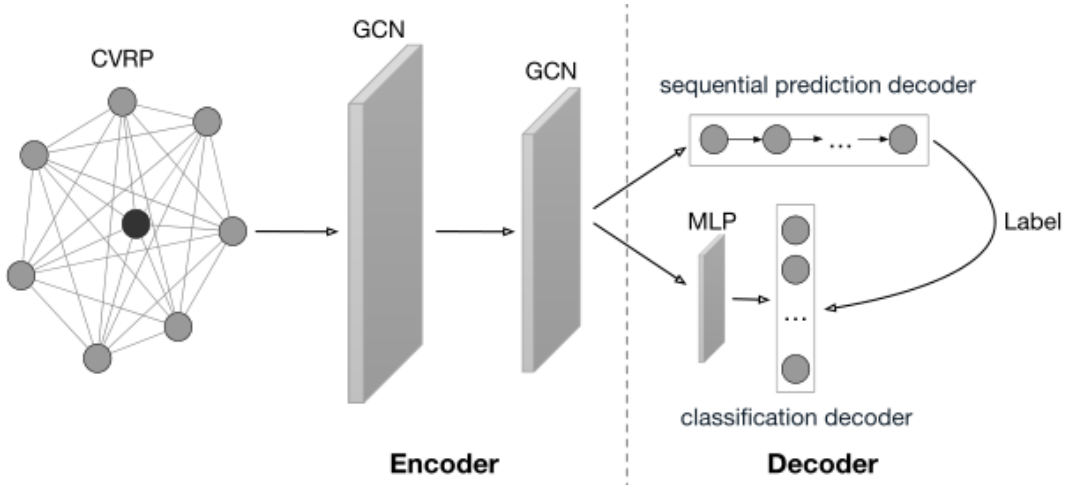

图 **1**　模型结构

本实验中实现的深度模型采用编码-解码（encoder-decoder）架构，其中编码部件将问题中的边和点编码为具有边。点信息的实数嵌入，然后将这些嵌入输入两个联合训练的解码器——序列预测解码器（squential prediction decoder）和分类解码器（classification decoder）来进行问题的求解。

2.2 编码器

编码器主要通过仿射变换将点信息（坐标、需求等信息）、边信息（长度、邻接矩阵等信息）映射为实数向量，然后将其通过图神经网络进行信息的聚合得到相应嵌入。

2.2.1 点特征的提取

对于位于 0 号点的仓库，其特征可以通过对于其坐标进行仿射变换得到；对于其余点，其特征可以通过拼接对于坐标、点需求进行仿射变换得到，即 i 号点的特征 x_i 计算如下：

$$x_i = \begin{cases} \text{Relu}\left(W_1 x_{c_0} + b_1\right), & \text{if } i = 0, \\ \text{Relu}\left(\left[W_2 x_{c_i} + b_2; W_3 x_{d_i} + b_3\right]\right), & \text{if } i \geq 0 \end{cases}$$

其中 x_c 代表坐标，W 代表仿射变换的矩阵权值，b 代表偏置，x_d 代表需求。

根据上述公式和定义，可以编写点特征提取的关键代码如下：

```
# 仓库默认下标0
depot = node[:, 0, :]
demand = demand[:, 1:].unsqueeze(2)
customer = node[:, 1:, ]

# Node and edge embedding
depot_embedding = self.Relu(self.node_W1(depot))
customer_embedding = self.Relu(torch.cat([self.node_W2(customer), self.node_W3(demand)],
    dim=2))
x = torch.cat([depot_embedding.unsqueeze(1), customer_embedding], dim=1)
```

2.2.2 边特征的提取

将邻接矩阵定义如下：

$$a_{ij} = \begin{cases} 1, \text{ if } i \text{ and } j \text{ are } k \text{ -nearest neighbors}, \\ -1, \text{ if } i = j, \\ 0, \text{ others.} \end{cases}$$

因此 i 号点连接 j 号点边特征 y_{ij} 可以计算如下：

$$y_{ij} = \text{Relu}\left(\left[W_4 m_{ij} + b_4; W_5 a_{ij} + b_5\right]\right)$$

其中 m_{ij} 代表 i 号点到 j 号点的距离，k 为超参数。

利用 pytorch 提供的 scatter 函数，可以方便地根据下标来对数据进行填充，因此可以编写边特征提取的相关代码如下：

```
edge = dis.unsqueeze(3)
self_edge = (torch.arange(0, node_num).unsqueeze(0)).T.unsqueeze(0).repeat(batch_size, 1, 1).to(device)
order = dis.sort(2)[1] # aij = -1
neighbor_index = order[:, :, 1:self.k+1] # aij = 1
a = torch.zeros_like(dis)
a = torch.scatter(a, 2, neighbor_index, 1)
a = torch.scatter(a, 2, self_edge, -1).to(device)
e = self.Relu(torch.cat([self.edge_W4(edge), self.edge_W5(a.unsqueeze(3))], dim=3))
```

2.2.3 图卷积神经网络

在得到点和边的特征后，利用邻接矩阵，将其输入图神经网络进行信息聚合即可得到点和边的嵌入。首先对点和边的特征进行仿射变换：

$$h_i^0 = W_{E1}x_i + b_{E1}$$
$$h_{e_{ij}}^0 = W_{E2}y_{ij} + b_{E2}.$$

然后将此富含信息的实向量通过 L 层图卷积得到最终的编码。

（1）点信息的编码：

$$h_{N(i)}^{\ell} = \sigma\left(W^{\ell}AGG\left(\left\{h_{i'}^{\ell-1}, \forall i' \in N(i)\right\}\right)\right)$$
$$h_i^{\ell} = COMBINE\left(h_i^{\ell-1}, h_{N(i)}^{\ell}\right)$$

$N(i)$ 代表点 i 的邻域，σ 代表激活函数，AGG 代表注意力机制。注意力机制实质上也是一种信息聚合的过程，不过其主要使用点积和相关操作，获取注意力权重来进行信息的线性加和来进行实现，利用 pytorch 提供的多头注意力机制接口即可进行定义，对于本实验，将头的数目设置为 1 即可：

```
self.attn = nn.MultiheadAttention(hidden_dim, num_heads=1)
```

每层图卷积网络的关键代码可编写如下：

```
h_nb_node = self.ln1_node(x + self.Relu(self.W_node(self.attn(x.permute(1, 0, 2), neighbor.permute(1, 0, 2, 3), neighbor.permute(1, 0, 2, 3))[0].permute(1, 0, 2))))
h_node = self.ln2_node(h_nb_node + self.Relu(self.V_node(torch.cat([self.V_node_in(x), h_nb_node], dim=-1))))
```

（2）边信息的编码：

$$h_{N(i)}^{\ell} = \sigma\left(W_I^{\ell}AGG_I^{\ell}\left(h_i^{\ell-1}, \left\{h_v^{\ell-1}, \forall v \in N(i)\right\}\right)\right.$$

$$h_{N(e_{ij})}^{\ell} = \sigma\left(W_E^{\ell} AGG_E^{\ell}\left(\left\{h_{e_{ij}}^{\ell-1}, h_i^{\ell-1}, h_j^{\ell-1}\right\}\right)\right.$$
$$AGG_E^{\ell}\left(\left\{h_{e_{ij}}^{\ell-1}, h_i^{\ell-1}, h_j^{\ell-1}\right\}\right) = W_{e1}^{\ell} h_{e_{ij}}^{\ell} + W_{e2}^{\ell} h_i^{\ell} + W_{e3}^{\ell} h_j^{\ell}$$

本质上与点信息编码的思想相似，都是通过注意力机制和仿射变换来进行信息的聚合，不过边的信息需要联系其直接连接的点来获取信息，而点是通过其邻域获取信息。每层图卷积网络的关键代码可编写如下：

```
# edge
x_from = x.unsqueeze(2).repeat(1,1,node_num,1)
x_to = x.unsqueeze(1).repeat(1,node_num,1,1)
h_nb_edge = self.ln1_edge(e + self.Relu(self.W_edge(self.W1_edge(e) + self.W2_edge(x_from)
    + self.W3_edge(x_to))))
h_edge = self.ln2_edge(h_nb_edge + self.Relu(self.V_edge(torch.cat([self.V_edge_in(e),
    h_nb_edge], dim=-1))))
```

（3）通过 L 层图卷积层：

综上，通过 L 层图卷积的关键代码可编写如下：

```
x = self.nodes_embedding(x)
e = self.edges_embedding(e)

for layer in self.gcn_layers:
    # x: BatchSize * V * Hid
    # e: B * V * V * H
    x, e = layer(x, e, neighbor_index)
return x, e
```

2.3 序列预测解码器

使用编码器得到的实向量编码输入 GRU（Gated recurrent unit，门控循环单元），以及基于上下文的注意力机制来将点嵌入映射为车辆路线序列。论文 [1] 中提及，不使用自注意力机制而是使用 GRU 是因为车辆路线序列解与前面的步骤紧密相关。

2.3.1 门控循环单元

对于车辆路线序列选择的随机策略，可以分解如下

$$P(\boldsymbol{\pi} \mid s; \theta) = \prod_{t=0}^{T} p\left(\pi_{t+1} \mid S, \boldsymbol{\pi}_t; \theta\right)$$
$$= \prod_{t=0}^{T} p\left(\pi_{t+1} \mid f\left(S, \theta_e\right), \boldsymbol{\pi}_t; \theta_d\right)$$

其中 $f(S,\theta_e)$ 为编码器，θ_d 为可学习的参数。在本实验中使用 GRU 来对上式中的项进行估计，即使用如下公式

$$p(\pi_t \mid f(S,\theta_e), \pi_{t-1}; \theta_d) = p(\pi_t \mid f(S,\theta_e), z_t; \theta_d)$$

通过使用 GRU 的应状态向量 z_t 来对在 $t-1$ 时刻的策略 π_{t-1} 进行嵌入，相关代码如下：

```
batch_size = x.size(0)
batch_idx = torch.arange(0, batch_size).unsqueeze(1).to(device)
last_x = x[batch_idx, last_node]
last_x = last_x.permute(1, 0, 2)
_, hidden = self.gru(last_x, hidden)
z = hidden[-1]
```

2.3.2 基于上下文的注意力机制

在解码时，使用注意力机制来对 $p(\pi_t \mid z_t, f(S,u;\theta_e);\theta_d)$ 进行计算。注意力机制将结合所有图中点的编码来对于下一个时间步的点进行选取，称选取的概率为上下文权重（context weight）[1]，其等价于约束后的归一化注意力权重：

$$u_{ti} = \begin{cases} -\infty, & \forall j \in N_{mt} \\ h_a^T \tanh\left(W^G [v_i; z_t]\right), & \text{otherwise} \end{cases}$$

其中 N_{mt} 为掩码掉的不符合约束的节点，相关代码如下：

```
_, u = self.pointer(z.unsqueeze(0), x.permute(1,0,2), x.permute(1,0,2))
u = u.permute(1,0,2)
u = u.masked_fill_(mask, -np.inf)
probs = self.sm(u)
```

2.3.3 带有约束的解码

2.4 分类解码器

利用边的编码输入分类解码器来预测对应边是否会在最终解中出现：

$$p_{e_{ij}}^{\text{VRP}} = \text{softmax}\left(MLP\left(h_{e_{ij}}^L\right)\right) \in [0,1]^2$$

分类解码器由简单的多层感知器进行是（1）或否（0）的分类来进行联合训练。多层感知器由三层神经元组成，隐藏层维度都为 256，关键代码如下：

```
class ClassificationDecoder(nn.Module):
    def __init__(self, input_dim):
        super(ClassificationDecoder, self).__init__()
```

```
        self.MLP = nn.Sequential(
            nn.Linear(input_dim, 256), nn.ReLU(),
            nn.Linear(256, 256), nn.ReLU(),
            nn.Linear(256, 2)
        )
        self.sm = nn.Softmax(-1)

    def forward(self, e):
        a = self.MLP(e).squeeze(-1)
        out = self.sm(a)
        return out
```

2.5 实验设计

三、 结果分析

交代实验环境，算法设计设计的参数说明；结果（图或表格），比如在若干次运行后所得的最好解，最差解，平均值，标准差。分析算法的性能，包括解的精度，算法的速度，或者与其他算法的对比分析。算法的优缺点；本实验的不足之处，进一步改进的设想。

四、 结论

在本次实验中

编码器

解码器

encoder-decoder 思想

遇到的问题：batch 化、约束添加

简要结论或者体会。

参考文献

[1] Duan L, Zhan Y, Hu H, et al. Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach[C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020: 3054-3063.

[2] Bello I, Pham H, Le Q V, et al. Neural combinatorial optimization with reinforcement learning[J]. arXiv preprint arXiv:1611.09940, 2016.

[3] Kool W, van Hoof H, Welling M. Attention, Learn to Solve Routing Problems![C]//International Conference on Learning Representations. 2018.

[4] Joshi C K, Laurent T, Bresson X. An efficient graph convolutional network technique for the travelling salesman problem[J]. arXiv preprint arXiv:1906.01227, 2019.