【源码阅读系列】一：GraphSAGE代码阅读（2）

原创 TwT 小韬学算法 2020-04-02

2.2 encoders.py

首先调用aggregator的forward函数得到所有节点的特征矩阵：

```
neigh_feats = self.aggregator.forward(nodes, [self.adj_lists[int(node)] for node in nodes], self.nu
```

判断是否为GCN模式，如果不是GCN模式，则需要进行拼接过程，反之，不需要进行拼接过程：

```
ifnot self.gcn:
    if self.cuda:
        self_feats = self.features(torch.LongTensor(nodes).cuda())
    else:
        self_feats = self.features(torch.LongTensor(nodes))
    combined = torch.cat([self_feats, neigh_feats], dim=1)
else:
    combined = neigh_feats
```

最后进行一个加权和非线性激活：

```
combined = F.relu(self.weight.mm(combined.t()))
```

2.3 model.py

首先封装了一个总的模型类：

```
class SupervisedGraphSage(nn.Module):
```

其中定义了推理流、损失函数以及相关初始化操作，forward函数调用网络层最后一层，然后递归式搜索到第一层函数，再从前到后执行（后面会详细说明）：

```
def forward(self, nodes):
    # k=2 的聚合过程
```

```
        embeds = self.enc(nodes)
        # 全连接层
        scores = self.weight.mm(embeds)
        return scores.t()
```

下面的函数用于加载cora数据集，并返回三个返回值，分别为特征矩阵（也就是每一个节点的原始特征向量组成的一个矩阵，一个维度为节点个数，一个维度为节点的特征维度）、标签向量、邻接表，其中前两个为 `ndarray` 类型，第三个为 `defaultdict` 类型：

```
def load_cora():
```

下面的函数为实现训练和测试的主体函数：

```
def run_cora():
```

定义了一个嵌入字典（随机初始化的），本质上是一个函数，features其实是一个函数句柄，是Embedding的 `forward` 函数，接受一个 `torch.LongTensor` ，实现部分嵌入向量的查询功能（详细内容参考PyTorch官方文档）：

```
features = nn.Embedding(2708, 1433)
```

将cora数据集的特征矩阵 `feat_data` 转换为 `Parameter` 类型作为嵌入字典的权重值。下面开始定义K=2时2层迭代过程的推理流程（也就是网络层结构）：

```
agg1 = MeanAggregator(features, cuda=False)
enc1 = Encoder(features, 1433, 128, adj_lists, agg1, gcn=True, cuda=False)
agg2 = MeanAggregator(lambda nodes : enc1(nodes).t(), cuda=False)
enc2 = Encoder(lambda nodes : enc1(nodes).t(), enc1.embed_dim, 128, adj_lists, agg2,base_model=enc1
```

第一行实例化一个 `MeanAggregator` 对象，将 `Embedding` 对象作为参数，调用 `MeanAggregator` 的 `__init__` 函数；第二行实例化一个 `Encoder` 对象，并将对象agg1作为参数传递，1433为输入时的嵌入维度，128为第一层处理后的嵌入维度；第三行继续实例化一个 `MeanAggregator` 对象，但是此时参数为匿名函数表达式，值得注意的是（此处困扰了我半个多小时），这里只是把 `lambda` 这个函数对象作为参数传递给 `__init__` 进行构造，而不会实际计算出lambda表达式的结果，也就是说此处不会调用 `enc1` 的 `forward` 函数；第四行同理。到此为止，不会有任何 `forward` 函数的调用过程，完全是不断调用各个类的

`__init__`。再往下的代码就是切分数据集得到训练集、测试集和验证集，然后通过循环的方式进行训练参数。构造优化器：

```
optimizer = torch.optim.SGD(filter(lambda p : p.requires_grad, graphsage.parameters()), lr=0.7)
```

经典步骤，（1）初始化梯度为0（2）前向推理（3）反向传播计算梯度（4）优化参数

```
optimizer.zero_grad()
loss = graphsage.loss(batch_nodes, Variable(torch.LongTensor(labels[np.array(batch_nodes)])))
loss.backward()
optimizer.step()
```

此处还有一个重点，就是关于整个结构的forward过程是如何进行的，继续拿出下面的这个代码：

```
agg1 = MeanAggregator(features, cuda=False)
enc1 = Encoder(features, 1433, 128, adj_lists, agg1, gcn=True, cuda=False)
agg2 = MeanAggregator(lambda nodes : enc1(nodes).t(), cuda=False)
enc2 = Encoder(lambda nodes : enc1(nodes).t(), enc1.embed_dim, 128, adj_lists, agg2,base_model=enc1
```

首先会调用：

```
enc2.forward()
```

此时enc2作为Encoder对象，enc2的forward函数会先调用agg2的forward函数（详情见Encoder代码），然后在agg2函数中，当计算到下面这里的时候：

```
if self.cuda:
    embed_matrix = self.features(torch.LongTensor(unique_nodes_list).cuda())
else:
    embed_matrix = self.features(torch.LongTensor(unique_nodes_list))
```

此时 `agg2` 的 `feature` 函数其实是一开始给定的那个 `lambda匿名函数`，所以此时会调用 `enc1` 的 `forward` 函数，而 `enc1` 的 `forward` 函数又会调用 `agg1` 的 `forward` 函数，那么这里就呈现出一种递归的调用逻辑。此处之所以这么设置，是因为代码中把 `agg` 层和 `enc` 层看作了一个整体，所以调用 `enc` 的 `forward` 并且让 `enc` 去调用 `agg`，实现层内部调用；而每一层的 `agg` 的参数是上一层 `enc` 的结果，实现层之间调用。为了更加直白，我抽象式地描述一下，假设网络层为A->B->C，通过调用C的方式运行整个网络，C在计算时会加载参数，

而参数是B计算出来的结果，同理B再去找A计算出的结果，这个过程也就是入递归的过程，然后A计算后退出进入B最后进入C，也就是出递归的过程。

[注]代码部分已经结束了，后面有时间的话会按照pytorch版本对其他的聚合方式以及无监督方式继续复现。如果有理解不到位的地方，欢迎给出指导意见！