使用强化学习进行自动电压控制

描述

本工程是一个展示如何使用multi-agent强化学习算法进行电力系统自动电压控制的示例。电力系统被划分为特定数量的区域，每个区域被分配给一个agent进行控制。每个区域的agent观察区域内各节点的有功功率、无功功率、电压幅值、电压相角，并调节发电机机端电压的幅值，使得每个节点的电压都保持在0.95~1.05范围内.
电力系统运行环境使用`pypower`和`gym`库实现. 强化学习算法和深度神经网络模型使用基于`Pytorch`的`machin`.

结构

`./powerenv.py`: 电力系统运行环境 。
`./voltage_controller.py`: AVC模型。

依赖

以下引用自Machin的Github仓库

> Machin is hosted on PyPI. Python >= 3.5 and PyTorch >= 1.5.0 is required. You may install the Machin library by simply typing
>
> ```
> pip install machin
> ```
>
> You are suggested to create a virtual environment first if you are using conda to manage your environments, to prevent PIP changes your packages without letting conda know.
>
> ```
> conda create -n some_env pip
> conda activate some_env
> pip install machin
> ```

主程序流程

模型训练

首先, 自定义`Actor`和`Critic`类, 其中类初始化的参数为状态和动作的维数.

```
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(Actor, self).__init__()
        ...

    def forward(self, state):
        ...

class Critic(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(Actor, self).__init__()
        ...
```

```
    def forward(self, state):
        ...
```

然后，根据算例所使用的电网模型，创建`pypower`格式的电网标准模型，以及各智能体控制区域的节点编号和发电机编号.

```
    # pypower标准case对象
    ppc = case...()
    # 各智能体控制区域的节点编号，二维列表，节点下标从0开始编号
    agentBuses = [
        [...],
        [...],
        ...
    ]

    # 各智能体控制区域的发电机编号，二维列表，节点下标从0开始编号
    agentGens = [
        [...],
        [...],
        ...
    ]
```

最后，将`pypower`标准模型和`agentBuses`、`agentGens`传入`VoltageController`类，然后使用`train`方法训练模型。

```
vc = VoltageController(ppc, agentBuses, agentGens, Actor, Critic)
vc.train()
```

代码示例见`voltage_controller.py`的`main`部分。

模型使用

模型训练完成后，在线控制时通过调用`VoltageController`类的`act`方法并传入电网运行状态对应的`pypower`标准case对象`ppc`，获得机端电压的控制量。

```
vc.act(ppc)
```

子程序接口

`powerenv.py`

`CLASS PowereNet(ppc, areaBuses, areaGens, vref = 1.0, alpha=0.1, beta=0.5))`

- 初始化参数
  - **ppc**: `Dict`, `pypower`标准case
  - **areaBuses**: `List[List[]]`, 各智能体控制区域的节点编号的列表
  - **areaGens**: `List[List[]]`, 各智能体控制区域的发电机编号的列表
  - **vref**: 系统的参考电压标幺值 p.u.
- `reset()`
  重置系统状态
- `get_state(ppc)`
  将`pypower`标准case对象转换为`Machin`状态格式

    - 参数
        - **ppc**: `pypower`标准case
    - 返回
        - **state**: 满足`Machin`中`MADDPG`类所需的输入格式的电网运行状态
- `step(action)`
  将`action`作用于环境，环境转换到下一状态
    - 参数
        - **action**: `Machin`输出格式的动作
    - 返回
        - **state**: 环境的下一运行状态
        - **reward**: 进入环境下一运行状态获得的奖励
        - **terminal**: 环境运行是否终止

`voltage_controller.py`

`CLASS VoltageController(ppc, agentBuses, agentGens, Actor, Critic)`

- 初始化参数
    - **ppc**: `Dict`, `pypower`标准case
    - **areaBuses**: `List[List[]]`, 各智能体控制区域的节点编号的列表
    - **areaGens**: `List[List[]]`, 各智能体控制区域的发电机编号的列表
    - **Actor**: `nn.Module`，Actor网络对应的类
    - **Critic**: `nn.Module`，Critic网络对应的类
- `train(max_episodes=100, max_steps=20, solved_reward = 0.1, solved_repeat = 5)`
    - 参数
        - **max_episodes**: 最多进行的episode个数
        - **max_step**: 每个episode最多进行的步数
        - **solved_reward**: 训练完成时回报的阈值
        - **solved_repeat**: 训练完成时至少应完成的训练次数
    - 返回
        - **action_losses**：`List`，Actor损失
        - **value_losses**：`List`，Critic损失
        - **time_step**: `List`，时间序列
- `act(ppc)`
    - 参数
        - **ppc**: `Dict`, `pypower`标准case