**场景：**

**略**

**建模：**

* **状态(state)定义：状态定义为二维数组（60，30）**

1. **第一个维度为60，指环境中包含总共60辆EV**
2. **第二个维度为30，指每辆EV有30个属性**

* **动作(action)定义：动作定义为二维数组（60，201）**

1. **60表示在每次决策中有60辆EV**
2. **201表示将动作决策离散化为201个类别，每次决策每一辆EV可选择的动作为201个分类，网络输出的是201个分类概率**

* **奖励(reward)定义：总充放电收益+消纳新能源收益+总车辆损耗**

1. **总充放电收益：根据每次动作时的电价，以及每辆EV每次动作充放电功率，可求出每辆车的收益，对每辆EV的收益求和，即为总充放电收益**
2. **消纳新能源收益：根据场景描述，为固定值**
3. **总车辆损耗：根据公式计算每辆EV在每次动作中的损耗，求和即为总车辆损耗**

* **优化目标total\_reward：最大化总收益total\_reward**

**根据场景描述，决策时间为两天，每次决策时长取5mn，每次决策收益为reward。**

**将每次决策的收益累加即为total\_reward。**

**DQN算法流程（伪代码）：**

**初始化环境Env；**

**初始化经验回放R;**

**初始化action\_value网络Q(s,a; θ)；**

**初始化target action\_value网络 (s,a; )；**

**决策次数(policy\_times) = 时长(48小时)/每次决策执行时间(1/12小时)**

**迭代次数(epoch) = 1**

**批次大小batch\_size = N**

**初始化变量**

**同步target action\_value网络步数C**

**for epoch = 1,agent do:**

**获取环境初始状态s0,初始total\_reward =0**

**For step in range(policy\_times):**

**if random.uniform > :**

**at =argmax Q(st,a;θ)**

**将at映射到动作空间，转为充放电功率值**

**else:**

**at = Env.random\_action()**

**执行动作at，得到奖励rt以及环境状态st+1，将数据(st, at,rt ,st+1,)存入R**

**if len(R) > N:**

**从R中随机采样N组(batch\_size)数据(si, ai,ri ,si+1,)。**

**执行一次梯度下降算法来更新action\_value网络：**

**if step % C==0:**

**更新target action\_value网络：**

**end for**

**enf for**