Environment:
环境：跟智能体交互，不断变化。

函数 1： reset 重启动，重新返回一批航班计划。 Batch_size（天数） * seq_len(每天航班数) * dim(航班属性)

```
def reset(self, training=True):
```

state: 维护两个数据结构，已分配，**未分配(可行区域)。《------调用规则引擎>**

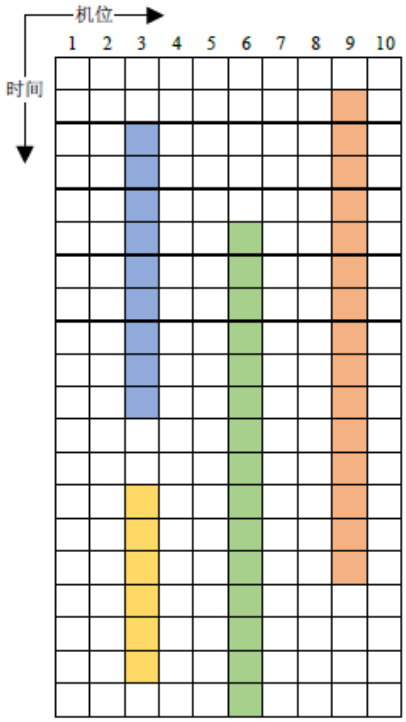


图 2-9 机位资源图

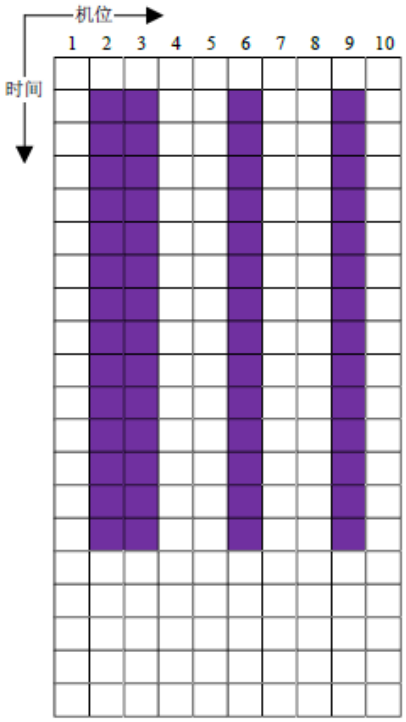


图 2-10 航班任务图

函数 2： 获取智能体的动作，或者（状态，动作）pair。（上一步状态 st，上一步动作（机位分配）action），更改环境状态，也即是输出 （下一步状态 st+1，奖励 rt）

初始状态：某一天航班计划 f1, f2,...,fT， 都是未分配
中间状态：f1,f2,..fk 已分配，现在 fk+1 分配
中止状态：所有航班 f1, f2,...,fT,都已分配 (Terminal state)

```python
def step(self, state, action):
    # current state
    obs, masked = state
    # next obs: shuffle the current observation according to the action
    next_obs = obs.scatter(1, action.long().unsqueeze(-1).expand_as(obs), obs)

    # return the reward: r(s(t+1)) - r(s(t))
    reward = self.reward(state) - self.reward((next_obs, masked))

    return (next_obs, masked), reward
```

怎么计算立即奖励 immediate reward？—— 当前状态-动作（st,at）瞬间奖励。
累计记录： expected reward (accumulative reward) : 现在和未来奖励之和。

奖励怎么算:

评价指标	10	机位利用率： (1) 靠桥率： 全天靠桥率=使用近机位的客运航班数量/客运航班总数； 过站靠桥率=过站客运航班使用近机位的航班数量/过站客运航班总数； 始发靠桥率=始发客运航班使用近机位的航班数量/始发客运航班总数； (2) 机位周转频次： 所有机位周转频次=机位使用次数/机位数 近机位周转频次=近机位使用次数/近机位数 (3) 机位调整情况： T 分钟内更改机位的比率=T 分钟内更改机位的航班数量/航班总数； (4) 滑行冲突率： 滑行冲突率=滑行冲突航班数/航班总数；
	11	服务质量： (1) 旅客步行时长： 旅客离港步行时长=安检口到机位的步行时间； 旅客进港步行时长=机位到到达口的时间； 转机步行时长=旅客出港到转机的登机口的步行时间； (2) 航司商业价值： 航班停桥价值系数：航司考虑航线差异、旅客构成等市场因素，定义系数 航司商业价值 = 所有航班的累计停桥价值系数
	12	能耗及资源利用率： (1) 滑行油耗：

__reward(配置，接口)

1. Abstract class
2. 输入：(st, at, st+1)， 输出数值 reward。 （整个过程 batch 操作，batch（天数））

直观实现思路： 每个机场，重写 reward 类。

1. 读配置文件方式它。细化：每个 reward 写一个继承类，CompositeReward =(（reward，系数），（r2，系数 2），…)

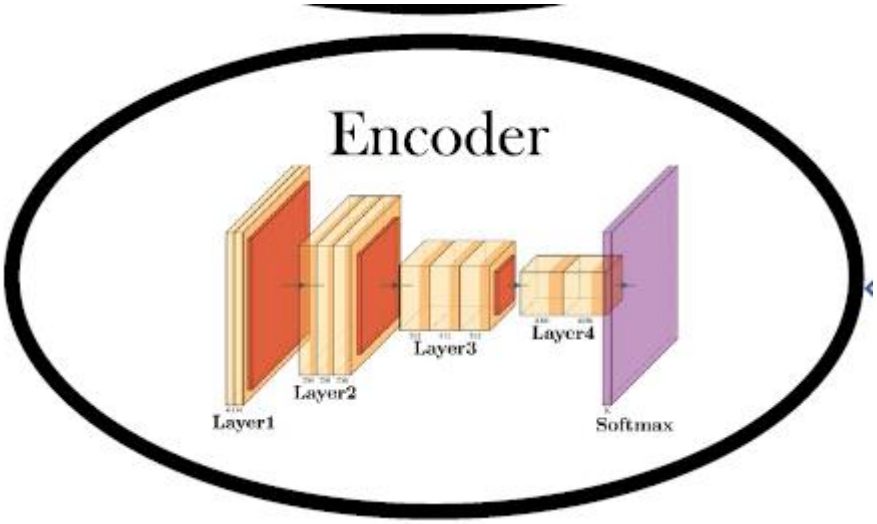
___将来留下自适应模块：根据历史（人工分配）经验，学习系数（或者目标），再指导智能体学习(inverse reinforcement learning),

Actor:

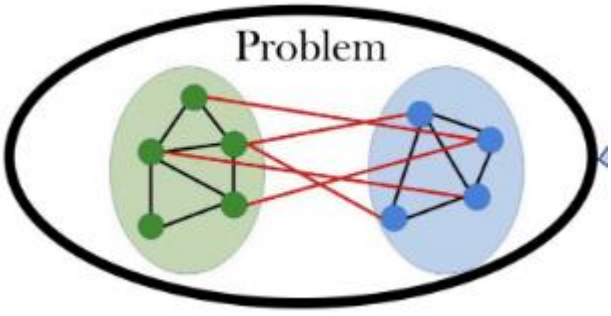
-- 输入一日的航班计划，给每个航班分配一个停机位
-- 迭代思想，逐一航班分配

--换一种思想：（最开始，初始解->迭代优化； 前面 K 个航班， 后面[k+1,T]航班逐一分配）
假设某一天航班计划 f1, f2,...,fT， 假设 f1,f2,..fk 已分配，现在 fk+1 分配

--算法角度： Actor 输入是 environment 输出的状态 state， Actor 输出（或智能体动作）选择某一个机位 pk
机位 pk = Actor(状态 state)。 Actor 本身是一个神经网络，或者运筹优化算法。



--结合上下文， state = f1,f2,..fk 已分配 + fk+1,...fT 未分配
--已分配怎么表示： f1,f2,..fk 已分配通过二部图表示。 左侧是航班，右侧是机位



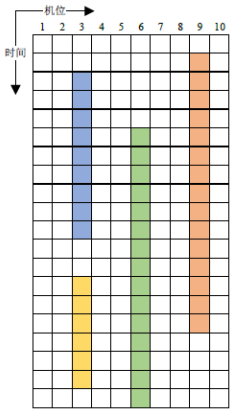


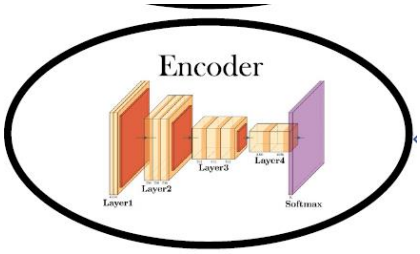
图 2-9 机位资源图

二部图通过矩阵表示： 机位与时间（00：00-23：59：59，5 分钟离散化， 24 小时*60 分钟/5 分钟= 288 个空格）

规则引擎：RuleEngine

输入：

1. 智能体——（环境 st，动作 at），返回 True/False，动作可不可行。



-> RuleEngine （mask 不可能动作）-》可行解

2. 用到环境里，(环境 st，动作 at，，上一步可行解区域)，返回 下一步状态 st+1（已分配+未分配可行解)

   本质返回：后续所有航班的可行解，最终体现 F1: (a1,a2,a3)… F2(a2,a5…)

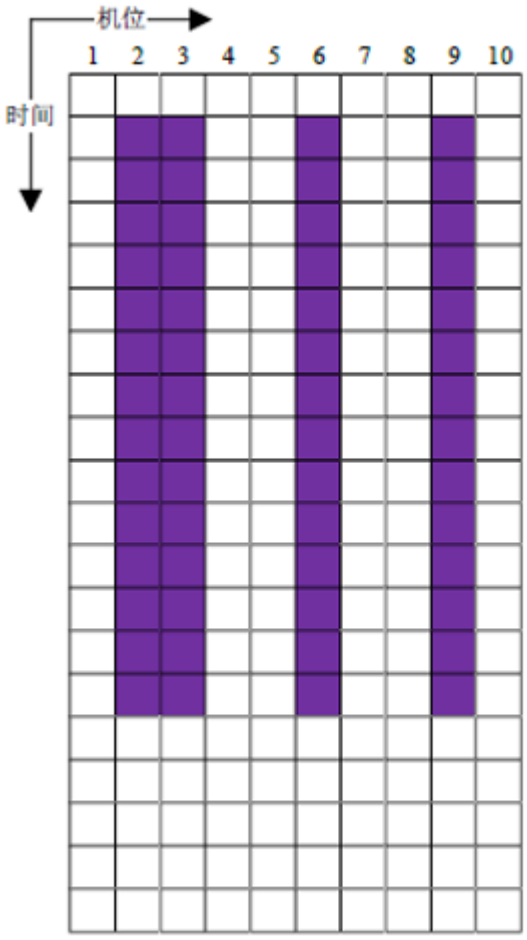


图 2-10 航班任务图

train(Actor, Enviroment)

1. For epoch=1…T:
2. State = Enviroment.reset()

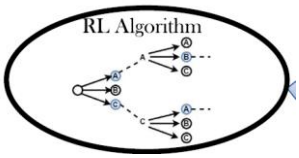
   While(state is not terminal):

Action = Actor(state)
Action = RuleEngine.Mask_action(state, action)
Action = pickup(Action)// MCST, random
(next_state, reward) = Enviroment.step(state, action)

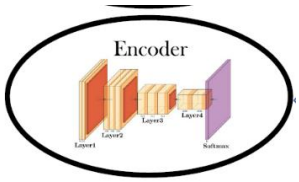
ReplayerBuffer.store([state, action, next_state, reward])

State = next_state

3. 训练网络——选择 TD3 或者其它强化学习算法，训练 Actor,本质是训练神经网络 Encoder

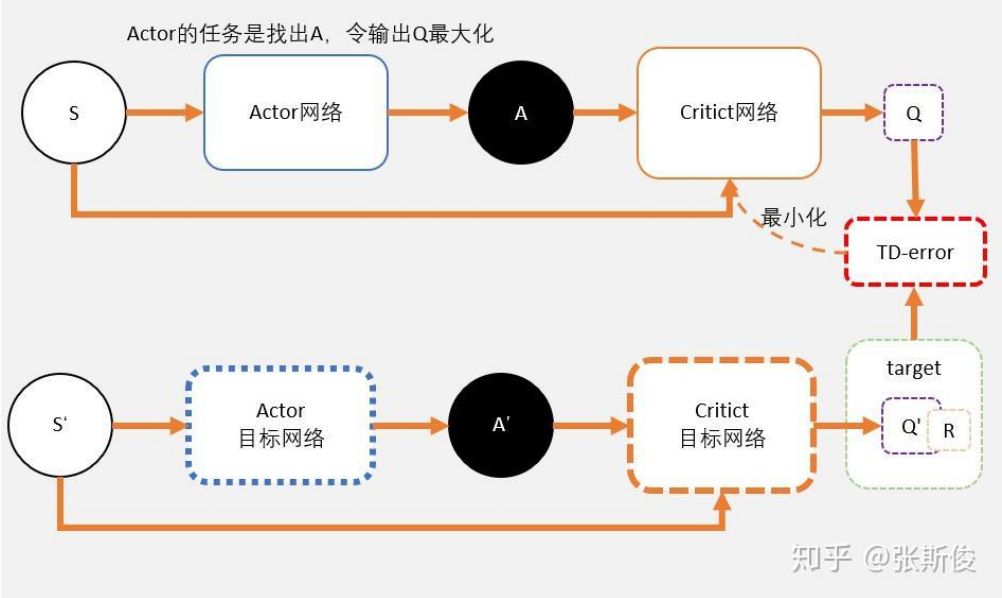


（训练算法）



（智能体）

采取哪个算法，训练 Encoder 网络，采取 TD3



目标是：累计记录 Q(st, at)而不是立即记录 r

At = Actor(st)
At+1 = Target_Actor(st+1)
TD3: Q(st, at) = r + Q’（st+1, at+1）[Target Actor, Target critic]

predictor(Actor, Enviroment)

```
State = Enviroment.reset()
While( state is not terminal):
        Action = Actor( state)
        Action = RuleEngine.Mask_action(state, action)
        Action = pickup(Action)// MCST, random
       (next_state, reward) = Enviroment.step(state, action)
       State = next_state
```