"Resource Management with Deep Reinforcement Learning" 의 재현에 관해

이현성

On-Policy Monte-Carlo Policy Gradient with Baseline

On-policy

Each episode is generated by somehow exploiting policy π.

Monte-Carlo를 이용해 episode를 simulate한 뒤, 마지막 결과의 retur을 이용해 모델을 update하는 방법을 REINFORCE라 한다.

Baseline
Mainly used to reduce variance of the training and thus make the learning fast.

Paramatized Baseline is called “Critic”.

$b_t = \frac{1}{n}\sum_i^n v_t^i$, where v_t^i is return of timestamp t of i^{th} episode.

Is this good discipline to follow?...

REINFORCE with Baseline

Policy Gradient With Baseline

$$\nabla J(\theta) = c\mathrm{E}_{s|}\left[\sum_{a}\left(q_{\pi}(s,a) - b(s)\right)\nabla_{\pi}(a|s,\theta)\right]$$

There is only Actor(paramatized policy) but not Critic.

Update is only made for θ.

REINFORCE with Baseline

- $J(\theta) = \mathrm{E}\left[\sum_a \left(q_\pi(S,a) - b(S)\right)\nabla_\theta \pi(a|S,\theta)\right] =$

$$\mathrm{E}_\mathrm{S}\left[\sum_a \pi(a|s,\theta)\left(q_\pi(S,a) - b(S)\right)\frac{\nabla_\theta \pi(a|S,\theta)}{\pi(a|S,\theta)}\right] =$$

$$\mathrm{E}_{\mathrm{s,a}}\left[\left(G - b(S)\right)\left(\frac{\nabla_\theta \pi(A|s,\theta)}{\pi(A|s,\theta)}\right)\right]$$

- $\theta_{t+1} = \theta_t + \lambda\left(G_t - b(S_t)\right)\nabla \log\left(\pi(A_t|S_t)\right)$

On-Policy Monte-Carlo Policy Gradient with Baseline

Monte-Carlo episode generation

s :=initial state for given environment.

UNTIL s is not terminate state

observe reward of current state r

sample action a from $\pi_\theta(s)$, $a \sim \pi_\theta(s)$ // on-policy monte-carlo policy-gradient

observe next action s' after action a with state s

update $s = s'$

Return visited states and corresponding action and rewards.

Environment Formulation

Only one job arrives at one timestep.
Scheduler chooses one or more job at each timestep.

$r_j := (r_1, r_2, \ldots r_d)$, r_i denotes the size of required resource of type i.
T_j := the time needed to complete job j. Also called duration of job j.

No preemption and No malleability
개별 작업의 리소스는 작업 도중동안 바뀌지 않고, 끝나지 않은 작업을 다른 작업으로 변경하지 않음.

Resources are fluidic(No fragmentations).

Action formulation

Differntiate timestamp and timestep

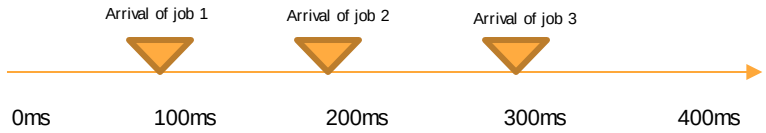
Timestamp: discrete index for (states, actions, rewards)

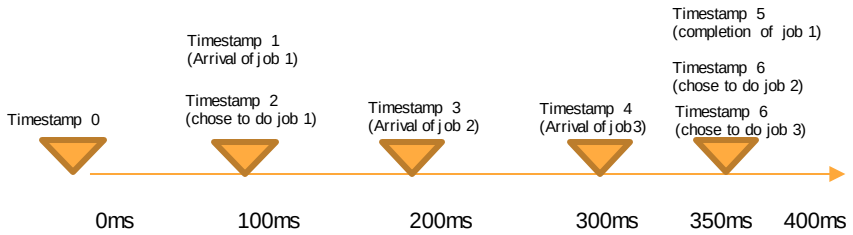
Timestep: actual flow of time.

Decay constant γ should be 1…?

This is mainly for handle scheduling multiple jobs in same timestep.

Thus action space $A = \{0, 1, \ldots, N\}$ (which jobs to choose, 0 means no job is chosen, N is the number of jobs arrived but not done.)





Each timesteps are 50ms.

Reward formulation

To minimize average completion time

$$r_s := -\#\ of\ remaining\ jobs\ at\ state\ s$$

To minimize average job slowdown

$$r_s := \sum_{j \in J} -\frac{1}{T_j}$$

($\frac{1}{T_j}$ means little slowdown for time-consuming job makes small effect on J).

I think r_s to be 0 for timestamps that make no actual progress in timestep.

Simulation of job arriving and scheduling

2 resources. Dominant and secondary with capacity 1 and 1 respectively.

For each time-step, job arrives at prob p_a

80% of job is short. 20% of job is large.

Duration : [1 – 3] timestep for short jobs, [10 – 15] timestep for large jobs.

Demand of resource: [0.25 – 0.5] for dominant jobs and [0.05 – 0.1] for secondary jobs.

Limitations on environment-model formulation

1. Offline update(using MC to make episodes)

   Update is delayed until enough episodes are observed.

2. Episodic formulation

   Job scheduling is continuous but it models it as episodic tasks.

3. Static baseline

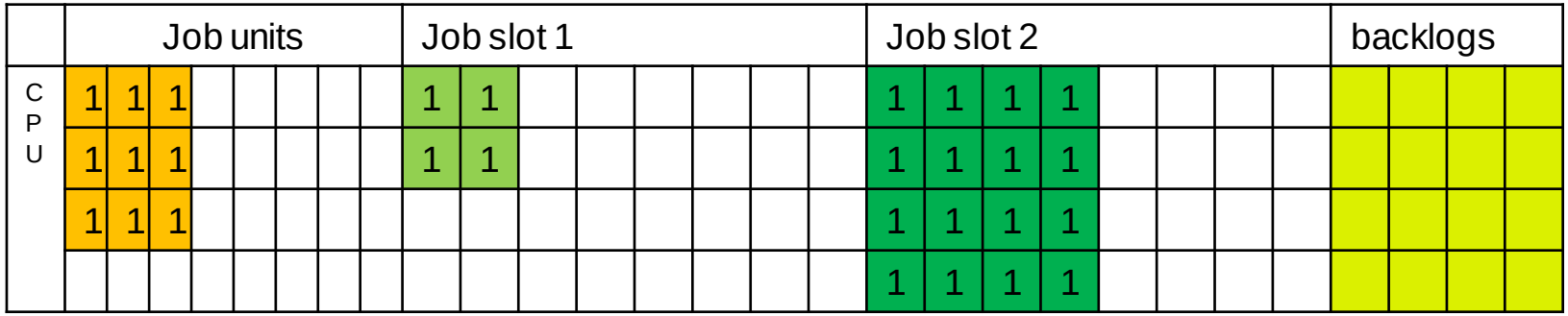
   The agent of environment creates baseline b by taking average of observed returns.

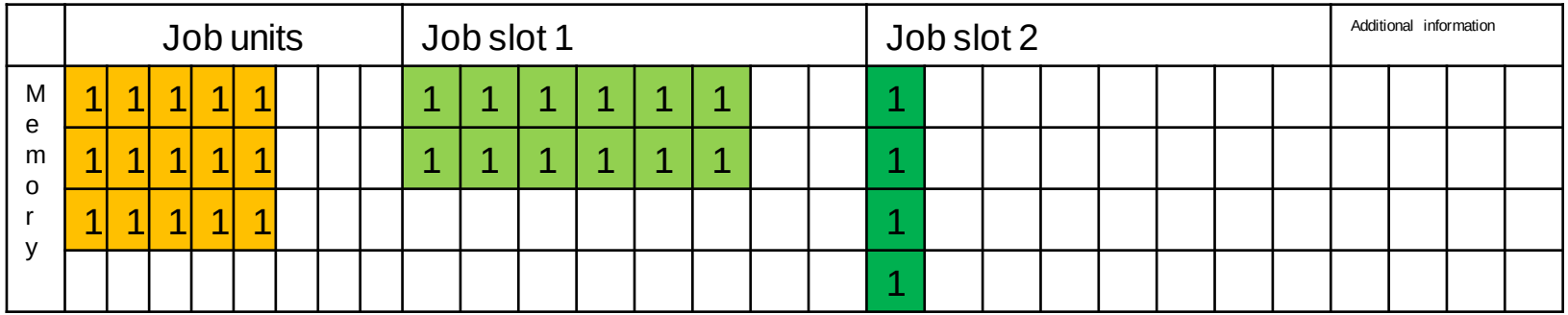
Transform State S into ANN input x

Example

- Two Resources: CPU and Memory.
  - CPU and memory have 8 units.
- Two jobs working: i_1
  - i_1 needs two unit CPU and four unit Memory and denote requirement to be(3, 5) for convenience.
  - i_1 is going to finish in two timesteps.
- Two jobs waiting to be fetched: j_1, j_2.
  - Required resources are (2,6), (4, 1), respectively.

Transform State S into ANN input x



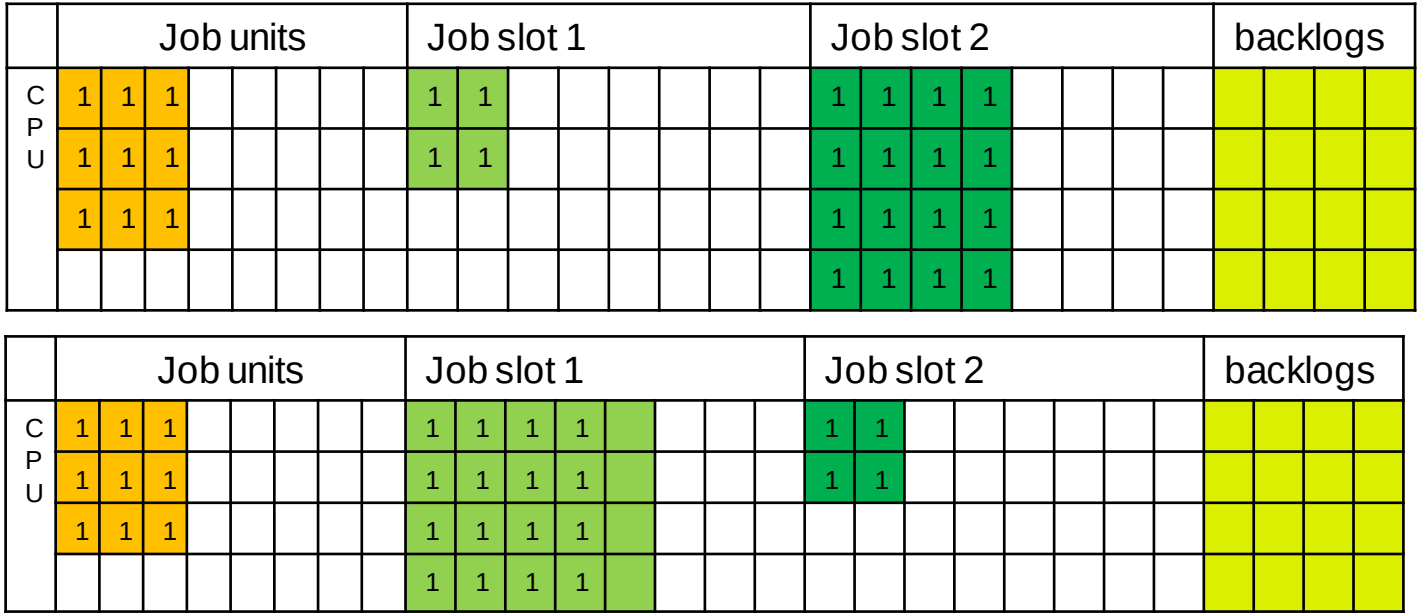


Limitation of input formation

1. MLP는 1차원 vector를 input으로 받는다. 2차원으로 표현해도, 어차피 모델이 flatten한다.

Limitation of input formation

- Two inputs bring different action distribution but they are actually same states.



Limitations

- Neural network that handle x[width:width*2] and x[width*2:width*3]… are different. They have same structure. Why split them?
- I think it's better to use shared model for each slots.