신호 최적화 코드 설명
(multiagent-TF2 중심으로)

2022.07.11.

내용

- 환경 설정
- 코드 구성
- 신호 최적화 실행
- 정책 : Sappo
- 행동 : gr, offset, gro, kc
- 다중 에이전트(학습/추론) 제어
- 분산 처리 지원

환경 설정

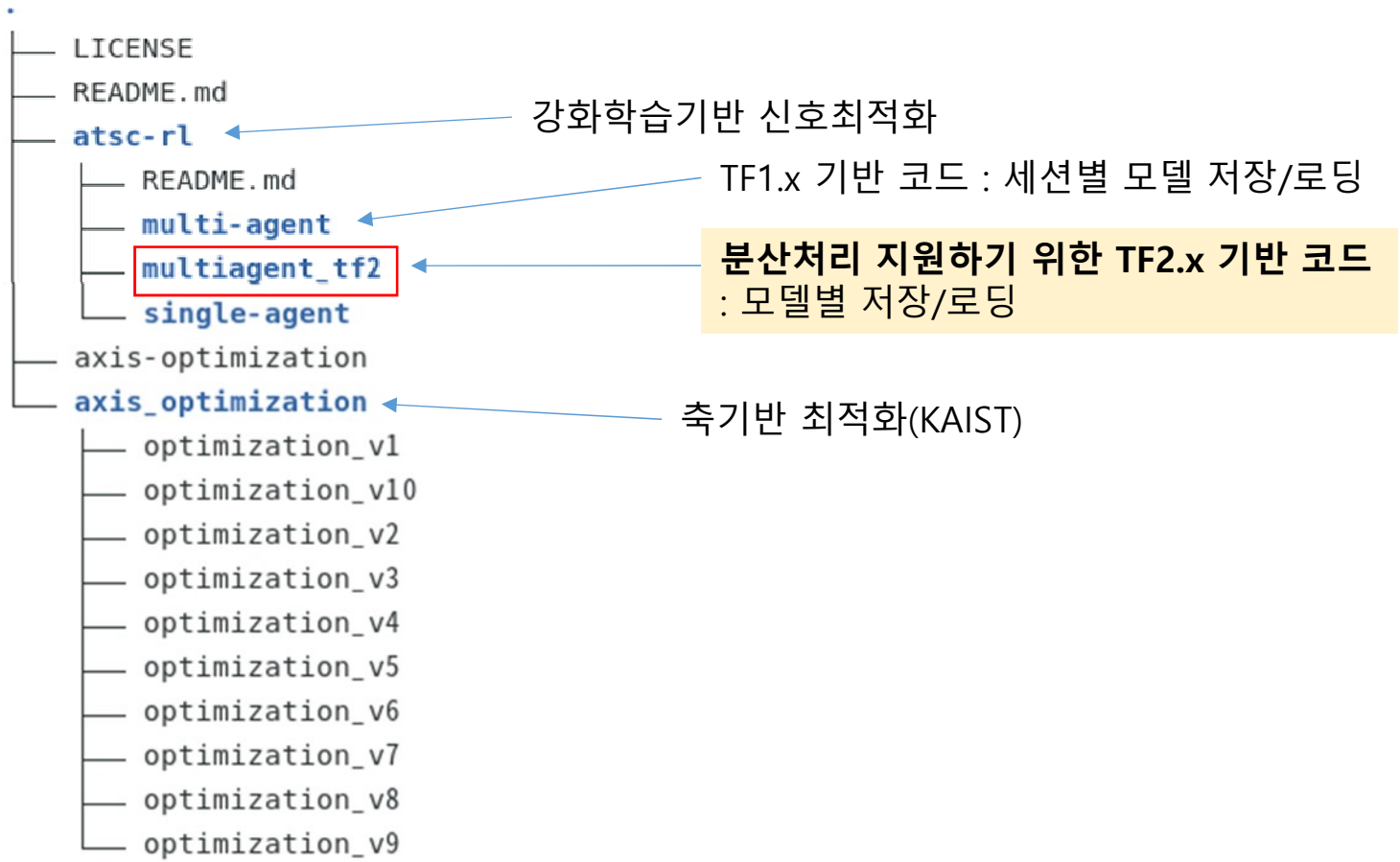
- 단일
  - 시뮬레이터 설치
  - SALT_HOME 설정
  - Miniforge3 다운로드 & 설치
    - 콘다 환경 생성 : ex, uniq.opt
      - ✓ Python 3.8
      - ✓ Tensorflow 2.3.0, keras 2.4.3, pandas, gym, matplotlib, deprecated, …
  - 교통 시뮬레이션 데이터 준비

- 분산
  - 패스워드없이 명령어 수행 가능 환경 구축
  - 참여 노드에 단일 노드 환경 구축
    - 동일한 경로에 최적화 프로그램 설치(복사) : (todo)스크립트화(?)

Ref. https://etri.gov-dooray.com/project/posts/3312471706655197412

코드 구성 : 디렉토리

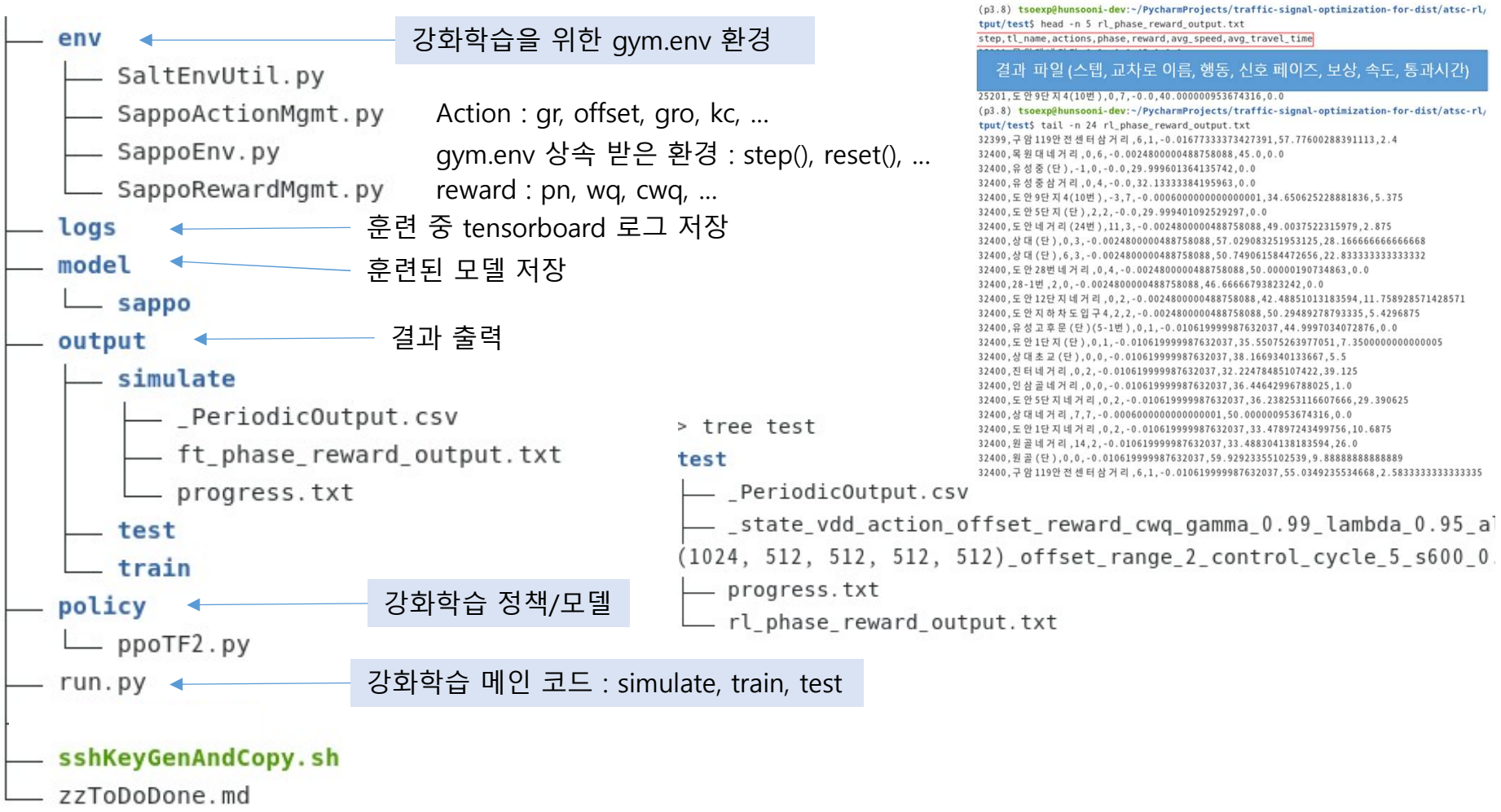
- https://github.com/etri-city-traffic-brain/traffic-signal-optimization



atsc-rl(**A**daptive **T**raffic **S**ignal **C**ontrol with **R**einforcement **L**earning)

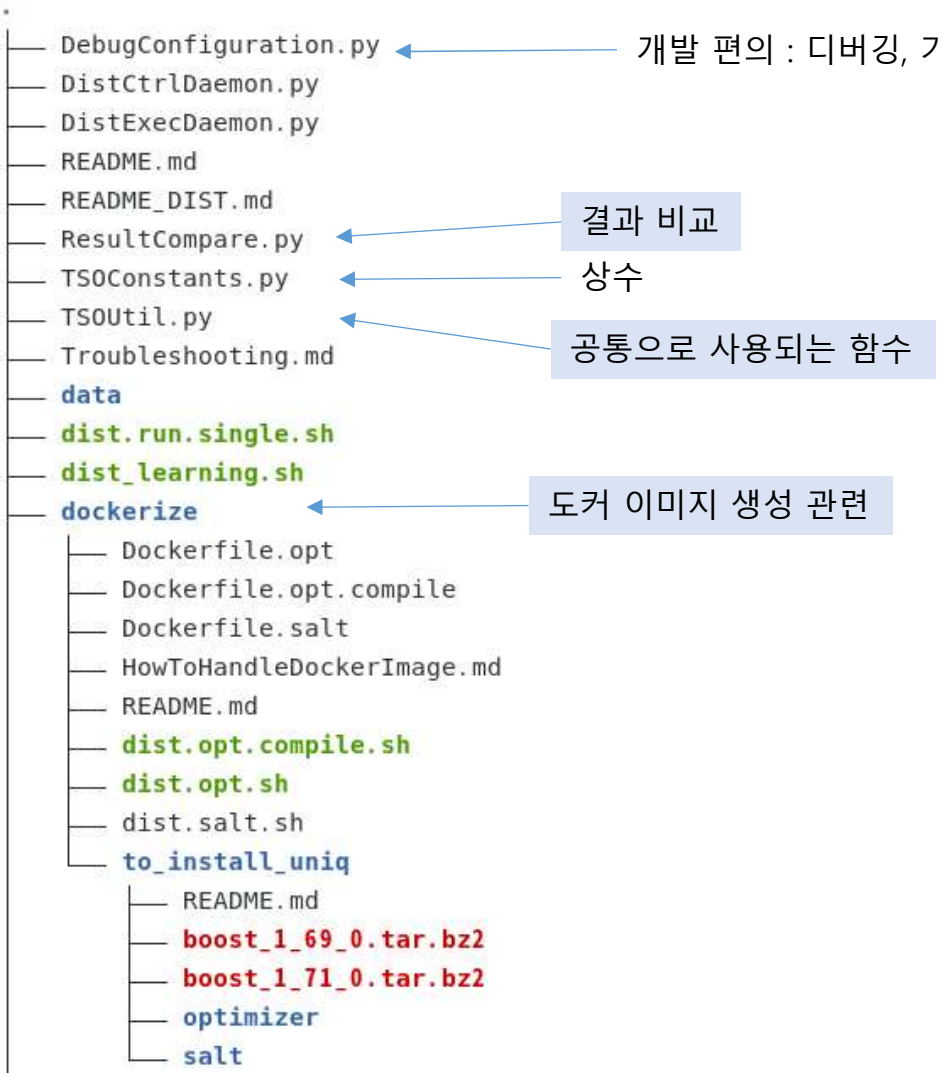
코드 구성 : 다중 에이전트 강화학습

- https://github.com/etri-city-traffic-brain/traffic-signal-optimization/atsc-rl/multiagent_tf2



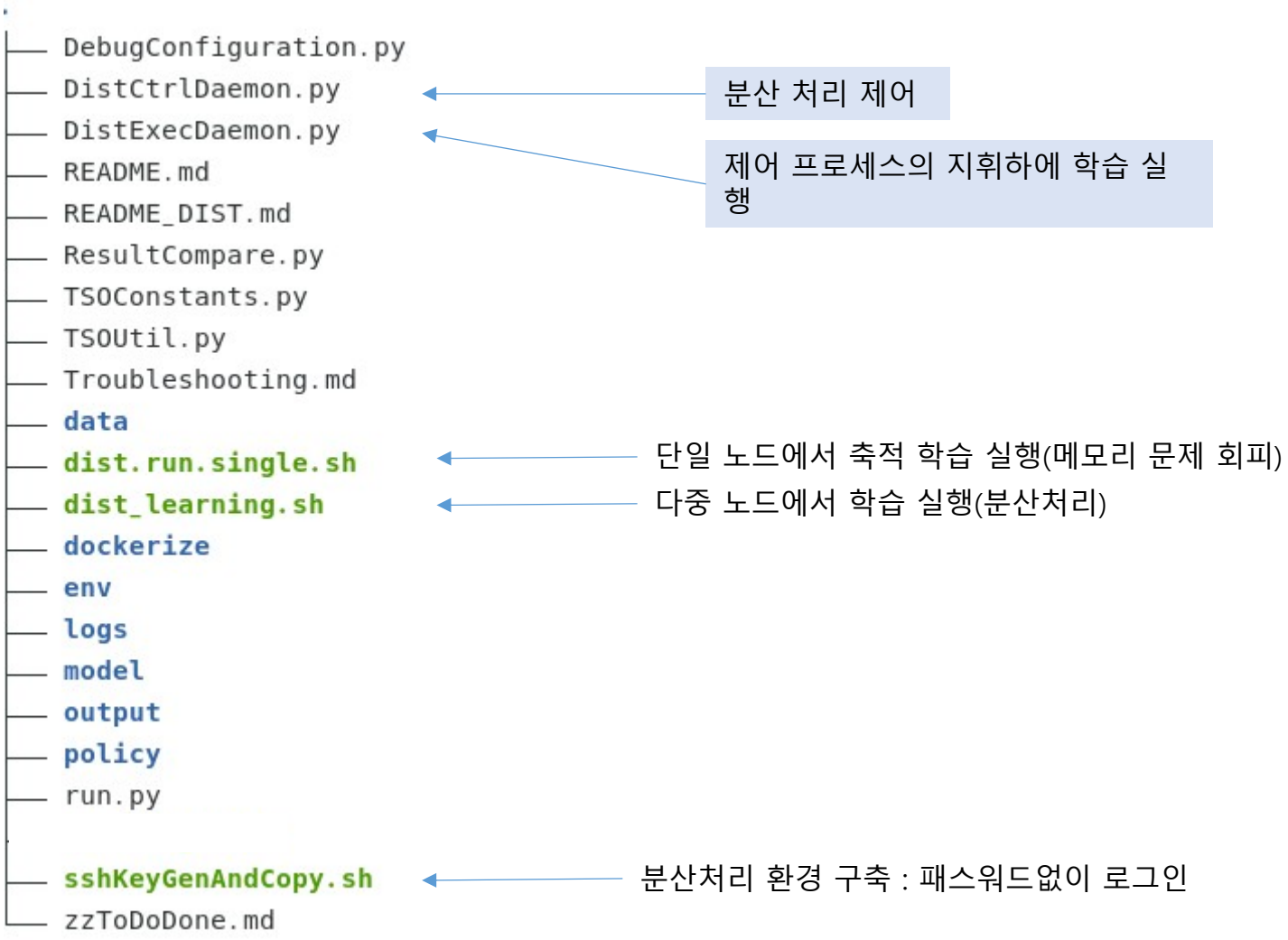
코드 구성 : 다중 에이전트 강화학습, 도커화

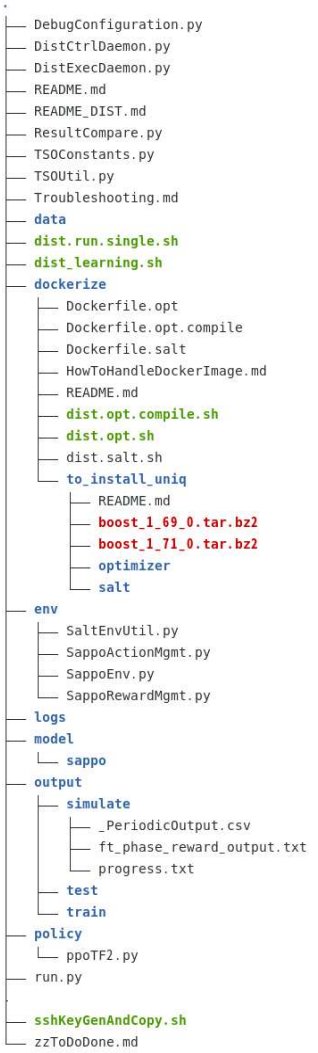
- https://github.com/etri-city-traffic-brain/traffic-signal-optimization/atsc-rl/multiagent_tf2



코드 구성 : 분산 처리

- https://github.com/etri-city-traffic-brain/traffic-signal-optimization/atsc-rl/multiagent_tf2





신호 최적화 실행 : simulate, train, test

- python run.py --mode simulate --map 'doan' --target-TL 'SA 101, SA 104'
  - ./output/simulate/ 경로에 결과 파일 생성
    - _PeriodicOutput.csv ft_phase_reward_output.txt progress.txt

- python run.py --mode train --map 'doan' --target-TL 'SA 101, SA 104' --method sappo
  - ./output/train/ 경로에 결과 파일 생성
    - _PeriodicOutput.csv progress.txt train_epoch_tl_reward.txt train_epoch_total_reward.txt
  - ./model/{method}/ 훈련된 모델 저장

- python run.py --mode test --map 'doan' --target-TL 'SA 101, SA 104' --model-num 123 --result-comp true
  - ./output/test/ 경로에 결과 파일 생성
    - _PeriodicOutput.csv rl_phase_reward_output.txt progress.txt {fn_result-comp}.csv

신호 최적화 실행 : 명령행 인자

```
usage: run.py [-h] [--mode {train,test,simulate}] [--scenario-file-path SCENARIO_FILE_PATH]
              [--map {dj_all,doan,sa_1_6_17}] [--target-TL TARGET_TL] [--start-time START_TIME]
              [--end-time END_TIME] [--method {sappo}] [--action {kc,offset,gr,gro}] [--state {v,d,vd,vdd}]
              [--reward-func {pn,wt,wt_max,wq,wq_median,wq_min,wq_max,tt,cwq}]
              [--cumulative-training CUMULATIVE_TRAINING] [--model-num MODEL_NUM]
              [--infer-model-num INFER_MODEL_NUM] [--result-comp RESULT_COMP] [--io-home IO_HOME]
              [--epoch EPOCH] [--warmup-time WARMUP_TIME] [--model-save-period MODEL_SAVE_PERIOD]
              [--print-out PRINT_OUT] [--action-t ACTION_T]
              [--reward-info-collection-cycle REWARD_INFO_COLLECTION_CYCLE]
              [--reward-gather-unit {sa,tl,env}] [--gamma GAMMA] [--epsilon EPSILON]
              [--epsilon-min EPSILON_MIN] [--epsilon-decay EPSILON_DECAY]
              [--epoch-exploration-decay EPOCH_EXPLORATION_DECAY] [--ppo-epoch PPO_EPOCH]
              [--ppo-eps PPO_EPS] [--_lambda _LAMBDA] [--a-lr A_LR] [--c-lr C_LR]
              [--network-size NETWORK_SIZE] [--optimizer OPTIMIZER] [--actionp ACTIONP] [--mem-len MEM_LEN]
              [--mem-fr MEM_FR] [--offset-range OFFSET_RANGE] [--control-cycle CONTROL_CYCLE]
              [--add-time ADD_TIME] [--infer-TL INFER_TL] [--infer-model-path INFER_MODEL_PATH]
              [--num-of-optimal-model-candidate NUM_OF_OPTIMAL_MODEL_CANDIDATE]
```

Ref. run.py, TSOUtil.py

신호 최적화 실행 : 명령행 인자 상세(1/2)

```
optional arguments:
  -h, --help            show this help message and exit
  --mode {train,test,simulate}
                        train - RL model training, test - trained model testing, simulate - fixed-time simulation before test
  --scenario-file-path SCENARIO_FILE_PATH
                        home directory of scenario; relative path
  --map {dj_all,doan,sa_1_6_17}
                        name of map
  --target-TL TARGET_TL
                        target signal groups; multiple groups can be separated by comma(ex. --target-TL 'SA 101,SA 104')
  --start-time START_TIME
                        start time of traffic simulation; seconds
  --end-time END_TIME   end time of traffic simulation; seconds
  --method {sappo}      optimizing method
  --action {kc,offset,gr,gro}
                        kc - keep or change(limit phase sequence), offset - offset, gr - green ratio, gro - green ratio+offset
  --state {v,d,vd,vdd}  v - volume, d - density, vd - volume + density, vdd - volume / density
  --reward-func {pn,wt,wt_max,wq,wq_median,wq_min,wq_max,tt,cwq}
                        pn - passed num, wt - wating time, wq - waiting q length, tt - travel time, cwq - cumulative waiting q
                        length
  --cumulative-training CUMULATIVE_TRAINING
                        whether do cumulative training based on a previously trained model parameter or not
  --model-num MODEL_NUM
                        trained model number
  --infer-model-num INFER_MODEL_NUM
                        trained model number for inference; this value is valid only when infer-TL is exist
  --result-comp RESULT_COMP
                        whether compare simulation result or not
  --io-home IO_HOME     home directory of io; relative path
  --epoch EPOCH         training epoch
  --warmup-time WARMUP_TIME
                        warming-up time of simulation
  --model-save-period MODEL_SAVE_PERIOD
                        how often to save the trained model
  --print-out PRINT_OUT
                        print result each step
```

신호 최적화 실행 : 명령행 인자 상세(2/2)

```
--action-t ACTION_T   the unit time of green phase allowance
--reward-info-collection-cycle REWARD_INFO_COLLECTION_CYCLE
                      Information collection cycle for reward calculation
--reward-gather-unit {sa,tl,env}
                      sa: sub-area, tl : traffic light, env : traffic environment
--gamma GAMMA         gamma
--epsilon EPSILON     epsilon for exploration
--epsilon-min EPSILON_MIN
                      minimum of epsilon for exploration
--epsilon-decay EPSILON_DECAY
                      epsilon decay for exploration
--epoch-exploration-decay EPOCH_EXPLORATION_DECAY
                      epsilon decay for an epoch; has meaning when we do cumulative training
--ppo-epoch PPO_EPOCH
                      model fit epoch
--ppo-eps PPO_EPS
--_lambda _LAMBDA
--a-lr A_LR           learning rate of actor
--c-lr C_LR           learning rate of critic
--network-size NETWORK_SIZE
                      size of network in ML model; string of comma separated integer values are expected
--optimizer OPTIMIZER
                      optimizer for ML model
--actionp ACTIONP     [in KC] action 0 or 1 prob.(-1~1): Higher value_collection select more zeros
--mem-len MEM_LEN     memory length
--mem-fr MEM_FR       memory forget ratio
--offset-range OFFSET_RANGE
                      offset side range
--control-cycle CONTROL_CYCLE
                      how open change the traffic signal table by ML agent
--add-time ADD_TIME   unit of duration change when we do green-ratio adjustment
--infer-TL INFER_TL   signal groups to do inference with pre-trained model; multiple groups can be
                      separated by comma(ex. --infer_TL 'SA 101,SA 104')
--infer-model-path INFER_MODEL_PATH
                      directory path which will be use to find the inference model
--num-of-optimal-model-candidate NUM_OF_OPTIMAL_MODEL_CANDIDATE
                      number of candidate to compare reward to find optimal model
```

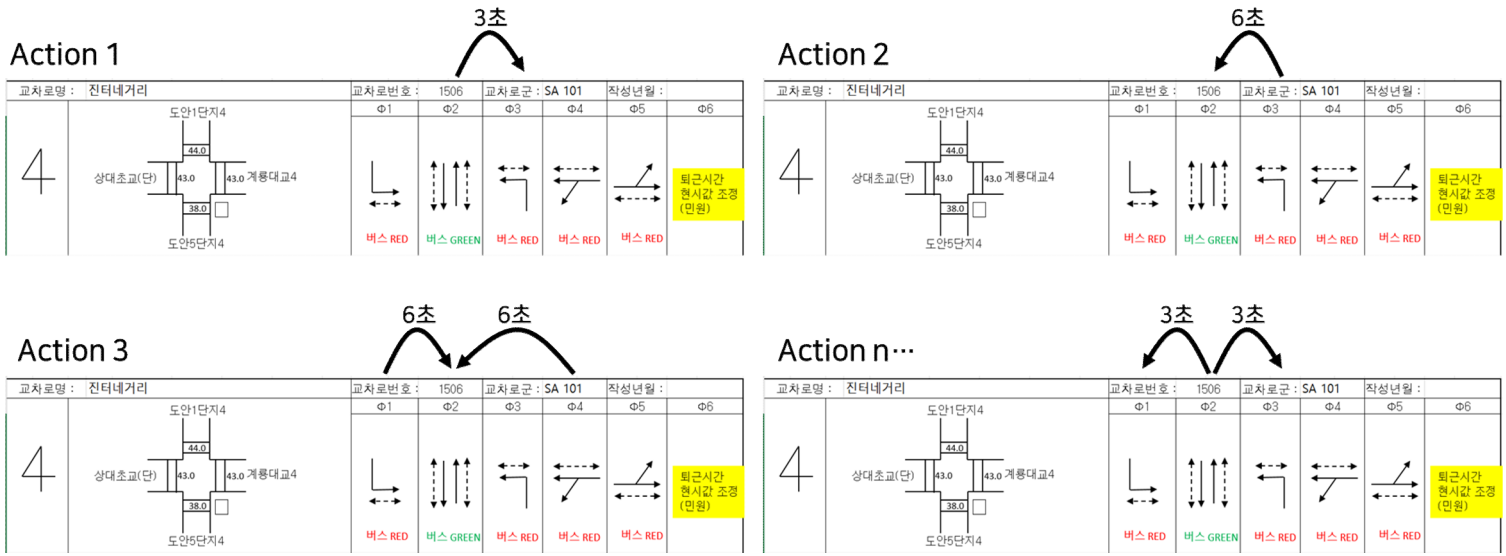
Policy : SAPPO

- SAPPO
  - 교차로 그룹(SA)에 속하는 교차로들을 하나의 모델의 출력으로 제어
  - 모델의 출력 수가 교차로 그룹에 속하는 교차로 수에 비례
    - 예, SA 101의 경우 속하는 교차로가 10개
      - ✓ action=[0.0843686 -0.36210205 -0.5634587 0.10146301 -0.17175334 -0.18904327 0.03833684 0.57406149 0.10136297 -0.51851053]
  - PPO 모델 이용
    - mgpi 자료 참고

Ref. https://pylessons.com/BipedalWalker-v3-PPO
https://github.com/pythonlessons/Reinforcement_Learning/blob/master/BipedalWalker-v3_PPO/BipedalWalker-v3_PPO.py

Action : gr, offset, gro, kc

- gr : green ratio, 녹색 시간 조정
  - 모델 출력을 미리 정의된 녹색시간 조정 테이블에 매핑
    - 녹색시간 조정 테이블
      - ✓ 최소/최대 녹색 제약 조건을 만족시키는 범위에서 조정
      - ✓ 주신호를 중심으로 일정 시간 가감 : 예, 현시 1과 현시 4에 각각 6초 축소 후 현시 2에 12초 증가

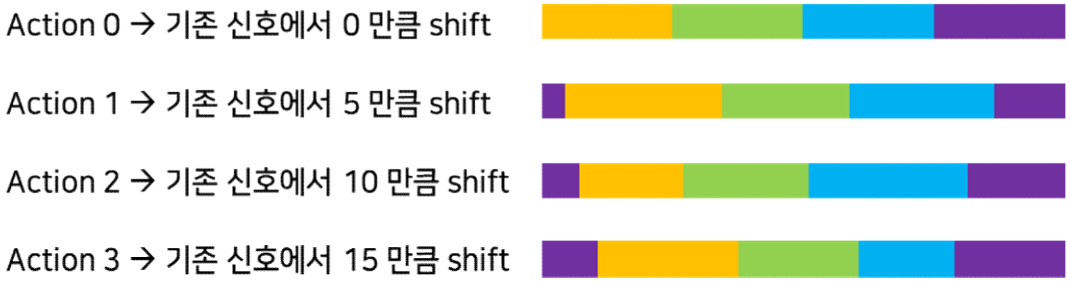


예, 미리 정의된 녹색시간 조정 테이블

Ref. SappoActionMgmt.py, SappoEnv.py

Action : gr, offset, gro, kc

- gr : green ratio, 녹색 시간 조정
- Offset, 오프셋 조정
  - 모델 출력을 오프셋 값으로 변환
    - -(cycle_length/2) ~ (cycle_length/2) 사이 값
  - 고정 신호에서 해당 값 만큼 이동



- gro : gr + offset, 녹색 신호 조정과 오프셋 조정을 동시에 진행
  - 출력을 분할하여 활용
    - 예, 출력 { (offset, gr), (offset, gr),..., (offset, gr) }
- kc : keep or change
  - 신호 변경 여부 결정

Ref. SappoActionMgmt.py, SappoEnv.py

학습 흐름

PPO example – cartpole

https://github.com/InSpaceAI/RL-Zoo/blob/master/PPO.py

```
if __name__ == "__main__":
    env = gym.make(env_name)
    agent = PPOAgent()
    episode = 0
    score = 0
    state = env.reset()
    while episode < episode_num:
        for t in range(n_step):
            #env.render()
            action, v_t, logp_t = agent.get_action([state])
            action, v_t, logp_t = action[0], v_t[0], logp_t[0]
            discrete_action = 0 if action < 0 else 1
            next_state, reward, done, info = env.step(discrete_action)

            states = np.r_[states, [state]] if t else [state]
            actions = np.r_[actions, [action]] if t else [action]
            values = np.r_[values, v_t] if t else [v_t]
            logp_ts = np.r_[logp_ts, logp_t] if t else [logp_t]
            dones = np.r_[dones, done] if t else [done]
            rewards = np.r_[rewards, reward] if t else [reward]

            state = next_state
            score += 1
            if done:
                print(f"{episode + 1} Episode / Score:{score}")
                score = 0
                state = env.reset()
                episode += 1

        v_t = agent.get_action([state])[1][0]
        values = np.r_[values, v_t]
        next_values = np.copy(values[1:])
        values = values[:-1]
        adv, target = get_gaes(rewards, dones, values, next_values, gamma, _lambda, True)
        agent.update(states, actions, target, adv, logp_ts)
    env.close()
```

// gym 환경 및 agent 준비

// 현재 state를 agent에 입력으로 넣어서 action을 가져옴
// continuous value를 discrete action으로 변경(cartpole은 action이 0, 1)
// discrete action을 env.step에 전달하여 다음 상태, 보상 등 정보를 가져옴

// 학습을 위해 각 변수들을 memory처럼 모아둠
// if t else [xxx] 부분은, 에피소드가 새로 시작할 때 마다 memory를 초기화하기 위함임
// ppo같은 on-policy 알고리즘은 dqn과 같은 off-policy 알고리즘과 다르게 경험을 재활용하지 않고 해당 에피소드에서만

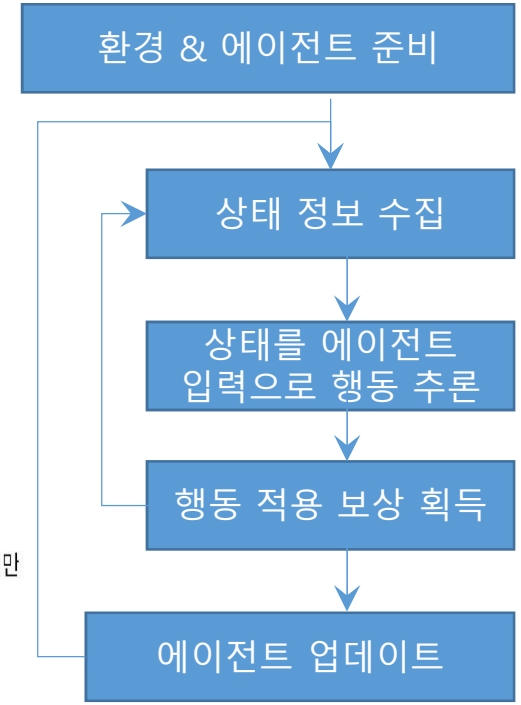
// next state가 현재 state가 됨

// done이면(pole이 넘어지면) 환경 리셋

// 한 에피소드가 끝나면 그동안 모은 경험으로 agent 업데이트

Generalized Advantage Estimation (GAE)

```
def get_gaes(rewards, dones, values, next_values, gamma, _lambda, normalize):
    deltas = [r + gamma * (1 - d) * nv - v for r, d, nv, v in zip(rewards, dones, next_values, values)]
    deltas = np.stack(deltas)
    gaes = copy.deepcopy(deltas)
    for t in reversed(range(len(deltas) - 1)):
        gaes[t] = gaes[t] + (1 - dones[t]) * gamma * _lambda * gaes[t + 1]
    target = gaes + values
    if normalize:
        gaes = (gaes - gaes.mean()) / (gaes.std() + 1e-8)
    return gaes, target
```

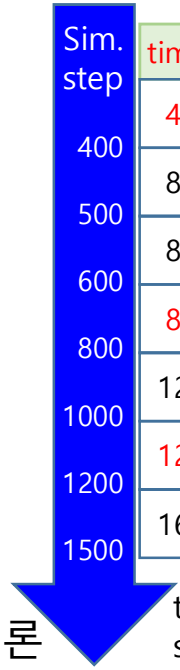


From mgpi's slide : 신호최적화_3차 코드리뷰_0405.pptx

다중 에이전트 (학습/추론) 제어

- Time_To_Act 테이블 이용하여 다중 에이전트 학습/추론 제어
  - agent 별로 다음 추론해야 하는 시간을 기억
  - 가장 가까운 추론 시간까지 step 증가 후 해당 에이전트 추론
  - 추론한 에이전트의 다음 추론 시간 증가

- 예
  - 가정
    - ✓ A는 400초, B는 500초, C는 600초 주기로 추론을 한다
    - ✓ 현재 시간 0초
  - 다음 추론 시간
    - ✓ { A:400, B:500, C:600}
  - 가장 가까운 추론 시간인 400초까지 시뮬레이션 스텝 증가 후 A 추론
  - A의 다음 추론 시간 증가
    - ✓ 다음 추론 시간 : { A:800, B:500, C:600 }
  - 가장 가까운 추론 시간인 500초까지 스텝 증가.. B 추론 …… 반복

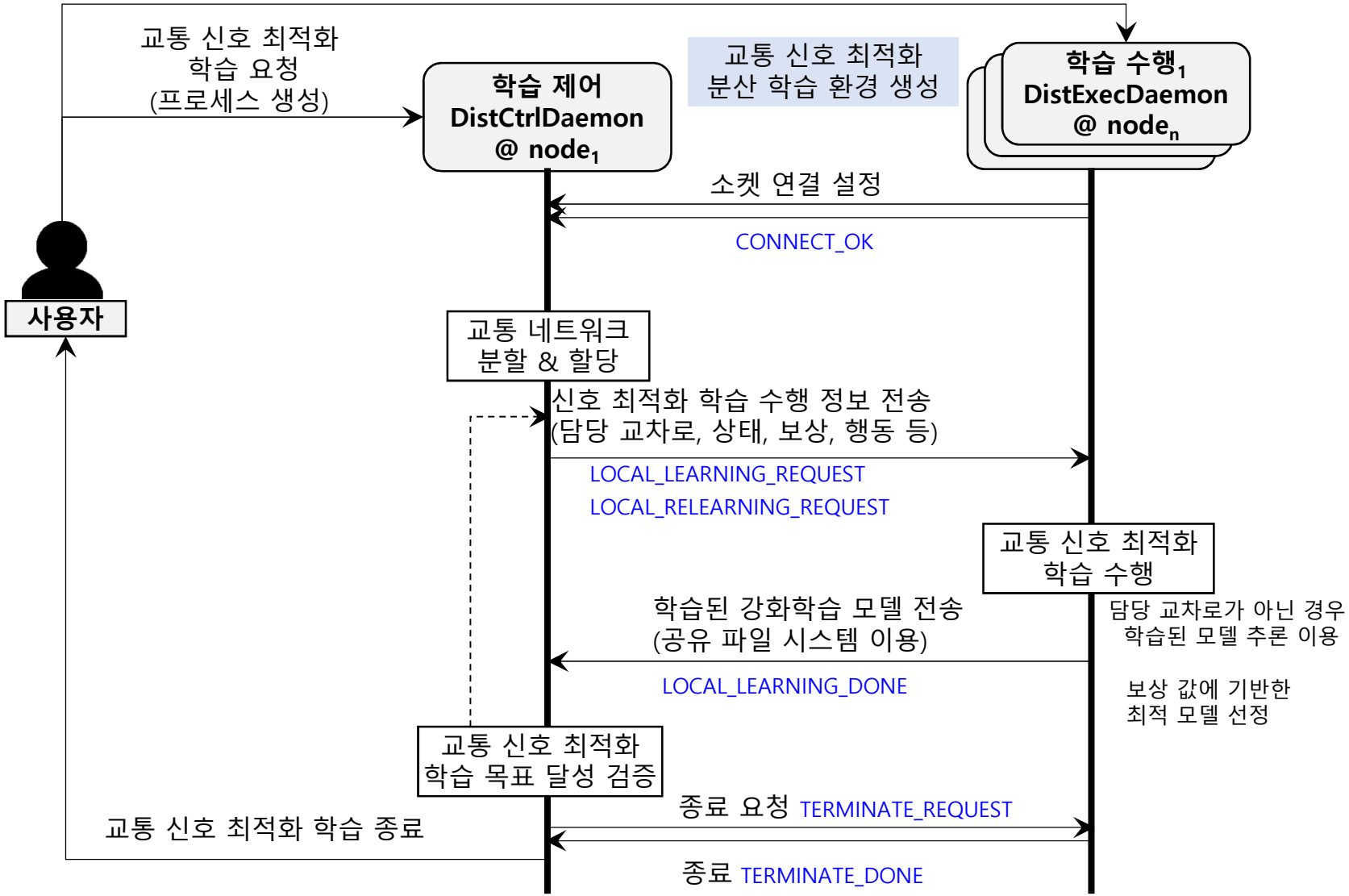


time_to_act TBL 변화		
400	500	600
800	500	600
800	1000	600
800	1000	1200
1200	1000	1200
1200	1500	1200
1600	1500	1800

time_to_act 인 step에서 추론/학습

Ref. SappoEnv.py

분산 처리 지원 : 구조, 흐름



Ref. DistCtrlDaemon.py DistExecDaemon.py TSOConstants.py run.py

분산처리 지원 : 흐름 예

가정 : 4개의 SA(SA_{101}, SA_{104}, SA_{107}, SA_{111})를
2개의 노드($Compute_{05}$, $Compute_{06}$)를 이용하여 분산 학습

Round	Ctrl Daemon($Compute_{05}$)	Exec Daemon($Compute_{05}$)	Exec Daemon($Compute_{06}$)
0	그룹 분할 할당 & 학습 요청 SA_{101}, SA_{107} : $Comp_{05}$ SA_{104}, SA_{111} : $Comp_{06}$		
		SA_{101}, SA_{107} : 훈련 & 최적 모델 저장 SA_{104}, SA_{111} : 고정 신호	SA_{104}, SA_{111} : 훈련 & 최적 모델 저장 SA_{101}, SA_{107} : 고정 신호
	성능 목표 달성 검증 : 미달 SA_{101}, SA_{104}, SA_{107}, SA_{111} 추론		
1	재학습 요청		
		SA_{101}, SA_{107} : 훈련 & 최적 모델 저장 SA_{104}, SA_{111} : 모델 추론	SA_{104}, SA_{111} : 훈련 & 최적 모델 저장 SA_{101}, SA_{107} : 모델 추론
	성능 목표 달성 검증 : 미달 SA_{101}, SA_{104}, SA_{107}, SA_{111} 추론		
2	재학습 요청		
		SA_{101}, SA_{107} : 훈련 & 최적 모델 저장 SA_{104}, SA_{111} : 모델 추론	SA_{104}, SA_{111} : 훈련 & 최적 모델 저장 SA_{101}, SA_{107} : 모델 추론
	성능 목표 달성 검증 : 달성 SA_{101}, SA_{104}, SA_{107}, SA_{111} 추론		

분산처리 지원 : 제어 데몬 명령행 실행 인자

```
usage: DistCtrlDaemon.py [-h] [--port PORT] [--validation-criteria VALIDATION_CRITERIA]
                         [--num-of-learning-daemon NUM_OF_LEARNING_DAEMON]
                         [--model-store-root-path MODEL_STORE_ROOT_PATH]
                         [--copy-simulation-output COPY_SIMULATION_OUTPUT]
                         [--mode {train,test,simulate}] [--scenario-file-path SCENARIO_FILE_PATH]
                         [--map {dj_all,doan,sa_1_6_17}] [--target-TL TARGET_TL]
                         [--start-time START_TIME] [--end-time END_TIME] [--method {sappo}]
                         [--action {kc,offset,gr,gro}] [--state {v,d,vd,vdd}]
                         [--reward-func {pn,wt,wt_max,wq,wq_median,wq_min,wq_max,tt,cwq}]
                         [--cumulative-training CUMULATIVE_TRAINING] [--model-num MODEL_NUM]
                         [--infer-model-num INFER_MODEL_NUM] [--result-comp RESULT_COMP]
                         [--io-home IO_HOME] [--epoch EPOCH] [--warmup-time WARMUP_TIME]
                         [--model-save-period MODEL_SAVE_PERIOD] [--print-out PRINT_OUT]
                         [--action-t ACTION_T]
                         [--reward-info-collection-cycle REWARD_INFO_COLLECTION_CYCLE]
                         [--reward-gather-unit {sa,tl,env}] [--gamma GAMMA] [--epsilon EPSILON]
                         [--epsilon-min EPSILON_MIN] [--epsilon-decay EPSILON_DECAY]
                         [--epoch-exploration-decay EPOCH_EXPLORATION_DECAY] [--ppo-epoch PPO_EPOCH]
                         [--ppo-eps PPO_EPS] [--_lambda _LAMBDA] [--a-lr A_LR] [--c-lr C_LR]
                         [--network-size NETWORK_SIZE] [--optimizer OPTIMIZER] [--actionp ACTIONP]
                         [--mem-len MEM_LEN] [--mem-fr MEM_FR] [--offset-range OFFSET_RANGE]
                         [--control-cycle CONTROL_CYCLE] [--add-time ADD_TIME] [--infer-TL INFER_TL]
                         [--infer-model-path INFER_MODEL_PATH]
                         [--num-of-optimal-model-candidate NUM_OF_OPTIMAL_MODEL_CANDIDATE]
```

run.py 명령행 인자

```
optional arguments:
  -h, --help            show this help message and exit
  --port PORT
  --validation-criteria VALIDATION_CRITERIA
  --num-of-learning-daemon NUM_OF_LEARNING_DAEMON
  --model-store-root-path MODEL_STORE_ROOT_PATH
  --copy-simulation-output COPY_SIMULATION_OUTPUT
                        whether do copy simulation output(PeriodicOutput.csv) to keep test history
                        or not
```

← target_improvement_rate

분산처리 지원 : 학습 수행 데몬 명령행 실행 인자

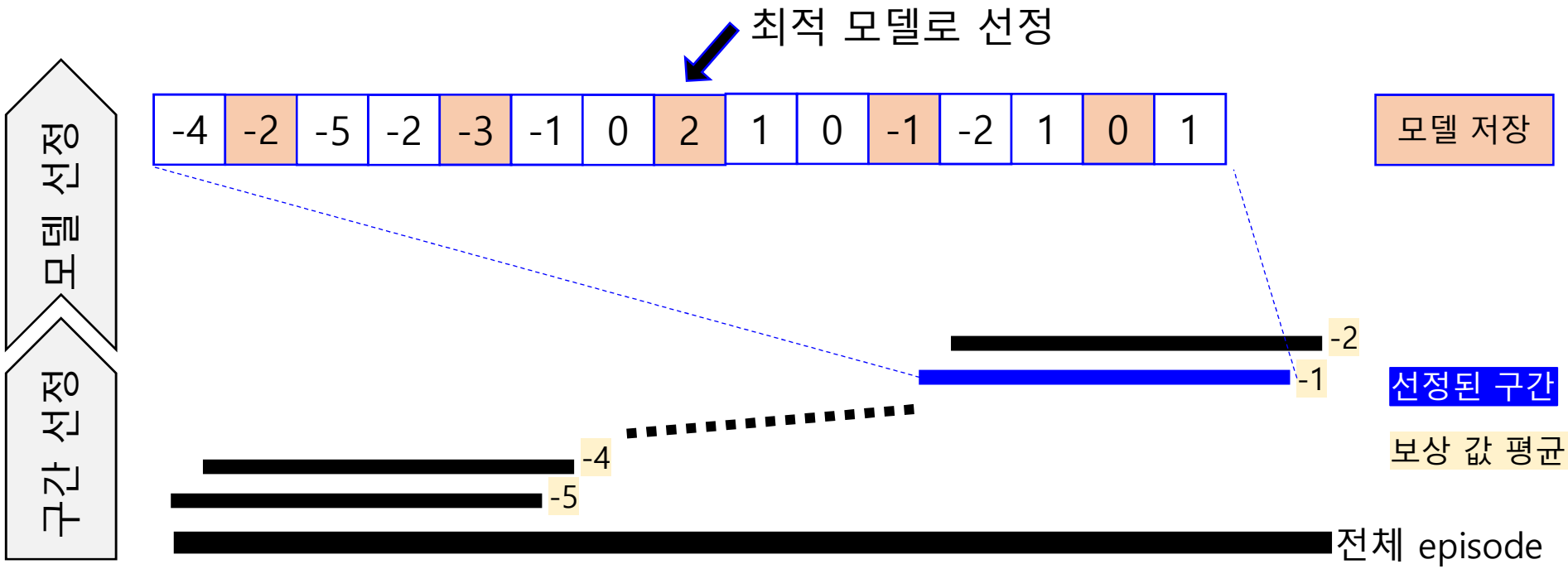
```
usage: DistExecDaemon.py [-h] [--port PORT] [--ip-addr IP_ADDR]

optional arguments:
  -h, --help         show this help message and exit
  --port PORT
  --ip-addr IP_ADDR
```

<-- 제어 데몬에 접속

분산 처리 지원 : 최적 모델 선정

- 보상 정보에 기반하여 최적 모델 선정
  - 전체 episode에 대해 보상 값 평균이 높은 구간 선정
    - 구간 : num-of-optimal-model-candidate * model-save-period
  - 선정된 구간 내에서 저장된 모델 중 보상 값이 가장 높은 모델 선정



가정 : num-of-optimal-model-candidate=5, model-save-period=3

분산처리 지원 : 최적 모델 전달, 결과 저장

● 공유 FS 이용

예,
가정 : 4개의 SA(SA_{101}, SA_{104}, SA_{107}, SA_{111})를
2개의 노드($Compute_{05}$, $Compute_{06}$)를 이용하여 분산 학습

Compute06

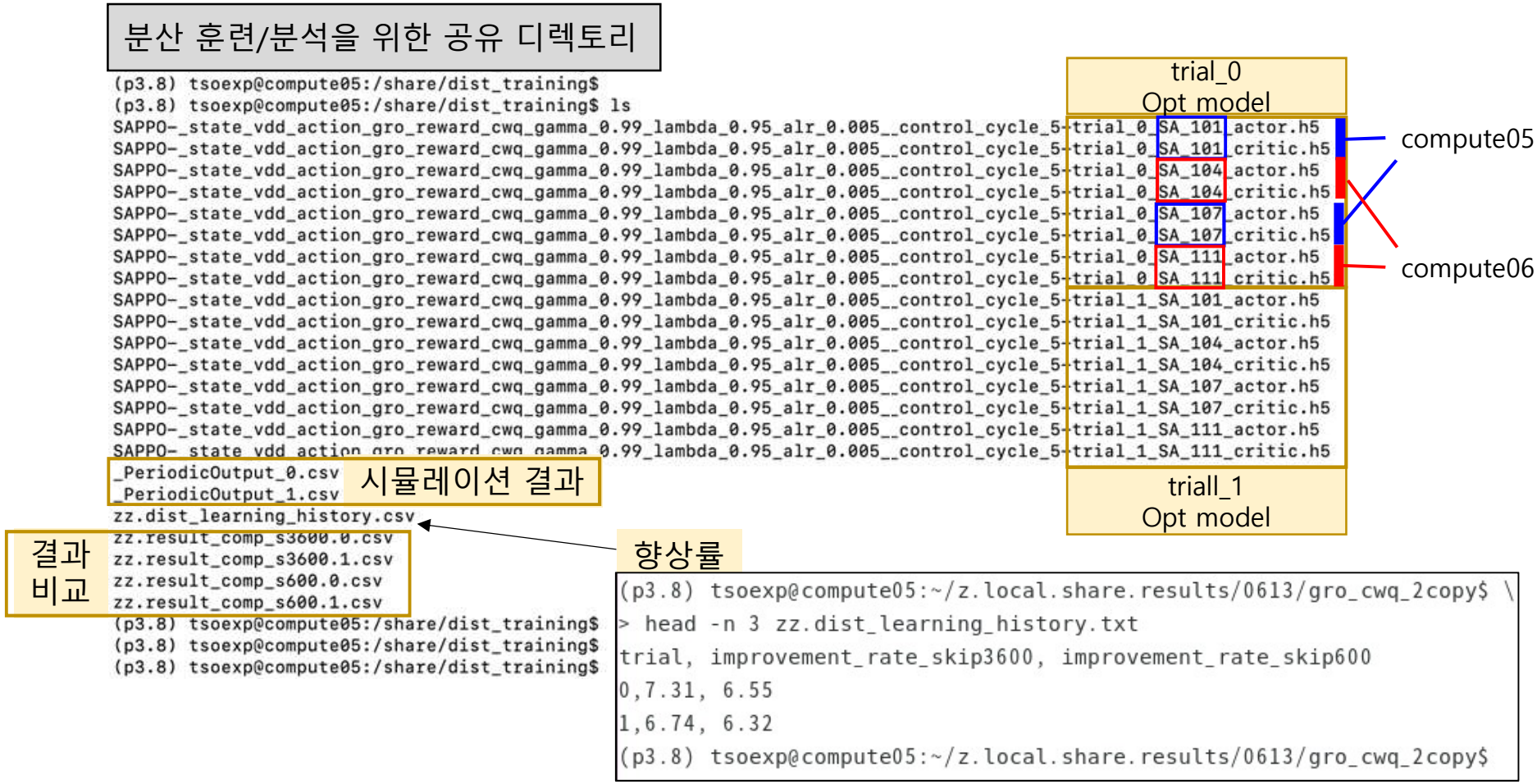
```
(p3.8) tsoexp@compute06:~/z.uniq/0613/multiagent_tf2.cwq$ ls
DebugConfiguration.py  TSOUtil.py          dockerize  output
DistCtrlDaemon.py      Troubleshooting.md  env        policy
DistExe
README.
README_
ResultC
TSOCons
(p3.8)
> head
./model
.005_mL
-185
./model
.005_mL
-155
(p3.8)
```

Compute05

```
(p3.8) tsoexp@compute05:~/z.uniq/0613/multiagent_tf2.cwq$ ls
DebugConfiguration.py  __pycache__          out.ctrl
DistCtrlDaemon.py      config.py            out.exec
DistExecDaemon.py      data                 out.tb
README.md              dist.run.single.sh   output
README_DIST.md         dist_learning.sh     policy
ResultCompare.py       dockerize            run.py
TSOConstants.py        env                  sshKeyGenAndCopy.sh
TSOUtil.py             logs                 zz.optimal_model_info.SA_101
Troubleshooting.md     model                zzToDoDone.md
(p3.8) tsoexp@compute05:~/z.uniq/0613/multiagent_tf2.cwq$ \
> head -n 2 zz.optimal_model_info.SA_101
./model/sappo/SAPPO-_state_vdd_action_gro_reward_cwq_gamma_0.99_lambda_0.95_alr_0.005_clr_0
.005_mLen_1000_mFR_0.9_netSz_(1024, 512, 256, 128, 64)_offset_range_2_control_cycle_5-trial
-175
./model/sappo/SAPPO-_state_vdd_action_gro_reward_cwq_gamma_0.99_lambda_0.95_alr_0.005_clr_0
.005_mLen_1000_mFR_0.9_netSz_(1024, 512, 256, 128, 64)_offset_range_2_control_cycle_5-trial
-25
(p3.8) tsoexp@compute05:~/z.uniq/0613/multiagent_tf2.cwq$
```

분산처리 지원 : 최적 모델 전달, 결과 저장

● 공유 FS 이용



분산처리 지원 : 분산 훈련 관련 파일

```
class _FN_PREFIX_ :
    '''
    file name prefix
    '''
    # (a prefix of) file name  to save optimal model info
    OPT_MODEL_INFO = "zz.optimal_model_info"

    # (a prefix of) file name to save a comparison of result
    RESULT_COMP = "zz.result_comp"

    # file name to save the history of distributed learning
    DIST_LEARNING_HISTORY = 'zz.dist_learning_history.csv'

    # (a prefix of) file name to save the contents of replay memory
    REPLAY_MEMORY = 'zz.replay_memory'
```

"TSOConstants.py" 119 줄 --19%-- 23,0-1 22%

다음 번에는....

- 코드 살펴보고 질문
  - 주석 확인
  - todo 로 언급된 것들이 확인
    - 반영 여부
    - 해결/개선 방안

시각적 분석 도구 http://101.79.1.124:8080/