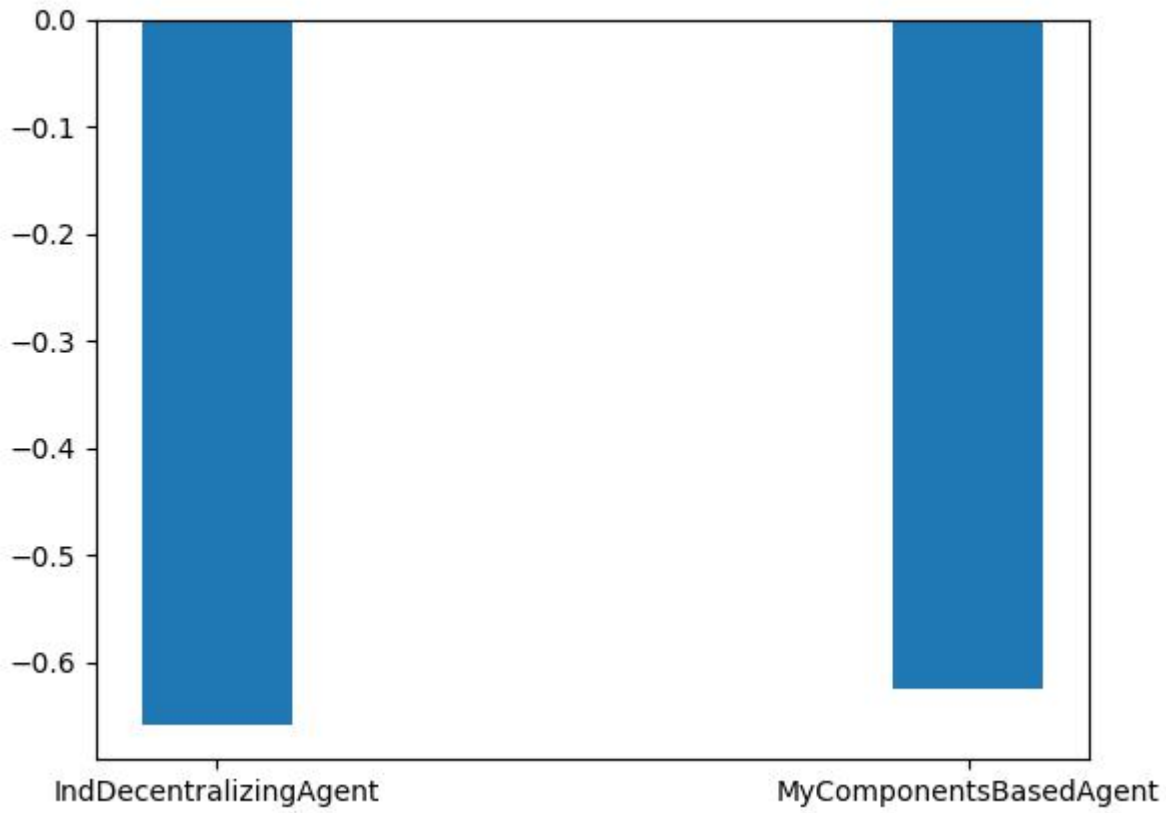
Policy Agents with MADDPG in SCML2020World - Evaluation

-- Dynamic range allowed negotiation issues

YUE NING, 10.01.2021,
Karlsruhe, Germany

Train and evaluate agents with IndDecentralizingAgent

Good result



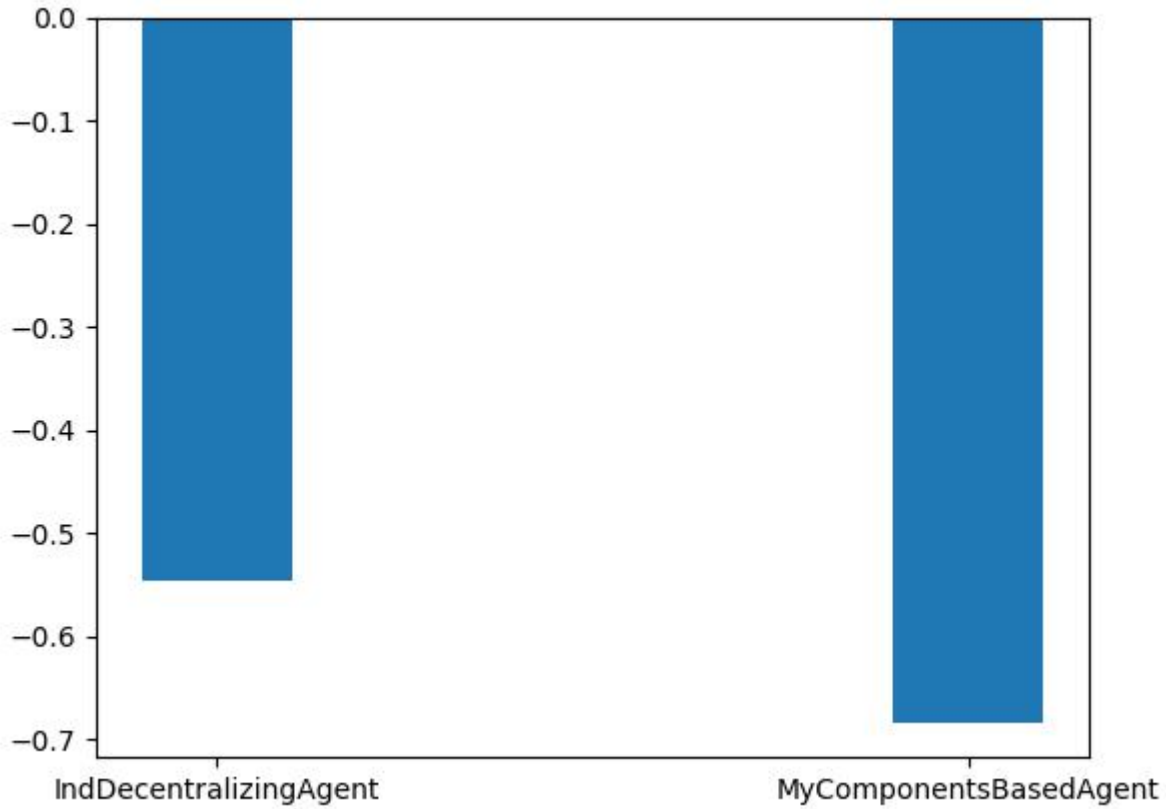
Baseline: score of IndDecentralizingAgent

The different between agents:
range of unit price of product
for Negotiation

Run in the following World
configuration,
n_steps = 100
agents_type =
[IndDecentralizingAgent,
MyComponentsBasedAgent]

Train and evaluate agents with IndDecentralizingAgent

Bad result



Under configuration,
n_steps = 100
agents_type =
[IndDecentralizingAgent,
MyComponentsBasedAgent]

Code, Training

action.m: decided action for sell negotiation

action.b: decided action for buy negotiation

```
# training period, action has been set up in env
if sell:
    if self.action.m is not None:
        # set up observation
        # self.state.o_role = sell
        self.state.o_negotiation_step = self.awi.current_step
        # for debug
        self.state.o_step = step
        self.state.o_is_sell = sell

        self.state.o_q_values = qvalues
        self.state.o_u_values = uvalues
        self.state.o_t_values = tvalues
        uvalues = tuple(np.array(uvalues) + (np.array(self.action.m)*self.action.m_vel).astype("int32"))

else:
    # for buyer
    if self.action.b is not None:
        uvalues = tuple(np.array(uvalues) + (np.array(self.action.b) * self.action.b_vel).astype("int32"))
```

Code, Select Model

Load all learned policies of this Agent(e.g. 00MyC@0), and random choice one from these.

```
dirs = POLICIES
policies = self._search_policies(dirs)

scope_prefix = self.name.replace("@", '-')
scopes = [scope_prefix + "_seller", scope_prefix + "_buyer"]
_tmp_index = []
for index, policy in enumerate(policies):
    if dirs[index]+scopes[0] in policy and dirs[index] + scopes[1] in policy:
        _tmp_index.append(index)
if not _tmp_index:
    logging.info(f"Do not load trained model for {self}, use default logic!")
    print(f"Do not load trained model for {self}, use default logic!")
    return
else:
    self.model_path = dirs[random.choice(_tmp_index)]
```

Summary

Dynamic range allowed negotiation issues has little effect on the agent.

Solved Problems

1. dynamically decide sell and buy range allowed negotiation issues.
2. Learn different policies in different worlds

Still existing problems

1. The score is not good. No substantial improvement
2. Possible reasons
    1. The training steps are not enough. Now there are 100 episodes, and a simulation episode can have up to 100 steps.
    2. Designs of observation, reward of agent are not good enough.
    3. **Trained policies are just suitable in a fixed world, although agents learn different policies in different worlds, but just randomly select the policies from learned policies.**

Summary

Questions:
1. **What factors affect the ranges allowed negotiation issues,** These factors will be regarded as important conditions for designing agents' observation.
2. **Also what factors affect concurrent negotiation issues**

Possible next step
1. Implements a predictor to predict the type of world instead random, when running the agent in tournaments.
2. Focus on the Concurrent negotiation control.

reward

```
rew = 0

# means in this world step, the agent starts a negotiation except initial state
if agent.state.o_negotiation_step == agent.awi.current_step:
    rew = (agent.state.f[2] - agent.state.f[1]) / (agent.state.f[0]) * REW_FACTOR

gap = []
for entity in world.entities:
    if entity is agent: continue
    if entity.action_callback == "system": continue
    if entity.action_callback is None: continue
    initial_balance = [_.initial_balance for _ in world.factories if _.agent_id == entity.id][0]
    current_balance = [_.current_balance for _ in world.factories if _.agent_id == entity.id][0]
    gap.append((current_balance - initial_balance) / initial_balance)

rew -= np.mean(np.array(gap))
return rew
```

observation

```
o_m = self.awi.profile.costs
o_m = o_m[:, self.awi.profile.processes]

# agent information, agent's
o_a = np.array([self._horizon])

# catalog prices of products
o_u_c = self.awi.catalog_prices
# TODO: excepted value after predict
o_u_e = np.array([self.expected_inputs, self.expected_outputs, self.input_cost, self.output_price])
# TODO: trading strategy, needed and secured
o_u_t = np.array([self.outputs_needed, self.outputs_secured, self.inputs_needed, self.inputs_secured])

# running negotiation and negotiation request of agent
o_q_n = np.array([
    self.running_negotiations,
    self.negotiation_requests,
])

o_t_c = np.array([self.awi.current_step / self.awi.n_steps])

# 2. Economic gap
economic_gaps = []
economic_gaps.append(self.state.f[2] - self.state.f[1])
economic_gaps = np.array(economic_gaps)

# return np.concatenate(economic_gaps + o_m.flatten() + o_a + o_u_c + o_u_e + o_u_t + o_q_n.flatten() +

return np.concatenate((economic_gaps.flatten(), o_m.flatten(), o_a, o_u_c, o_q_n.flatten(), o_t_c))
```