QLAgent : An Agent For Competing The SCML OneShot Track

Authors: Ran Sandhaus: ransandhaus@gmail.com, Ophir Haroche: ophirbh@gmail.com, Nadav Spitzer: nspitzern@gmail.com, Laor Spitz: laorsp7@gmail.com - Bar Ilan University

In this work we are aiming to compete on the OneShot track of the SCML competition, using a reinforcement-learning based approach.

1. Introduction

1.1. Abstract - Our Approach

We decided to use the **Q-learning** algorithm - a well-known and widely used reinforcement learning algorithm, as our base. We decided to use it since it is a model-free algorithm and given a finite MDP, in case the problem is modeled correctly in the algorithm, it is guaranteed to converge to optimal policy to produce highest reward - here it is interpreted as the highest profit (here we try to learn the best policy, per opposing agent, for actions during negotiation). **We further built on top of this algorithm**, to fine-tune it to the negotiation problem and specifically the SCML setting, to make it achieve higher scores faster. **For example, added heuristics and constraints on action selection**, to avoid obvious pitfalls or exploit obvious "easy profit" cases. **We also presented a method to encode states and actions in a generic way**, which is not trivial since a particular tournaments' properties (such as price ranges) may change between simulations. This enabled us a way for an efficient pre-training.

2. Approach In Detail

2.1. Negotiation-Adapted Q-Learning

2.1.1. General Settings

Important general settings:

1. Q-learning is formalized for discrete states and actions. Since in this problem we have continuous values for some items (such as price), we apply a **discretization technique** in order to make Q-Learning usable for the problem and also in order to optimize memory and runtime performance (trading off with learned policy's quality).
2. Since opposite agents are expected to be different, we apply a Q-Learning process per opposite agent.
3. Q-Learning algorithm is implemented with a ε-greedy strategy where the exploration probability drops as simulation progresses; this is in order to enable more exploratory nature on early stages to be able to escape local maxima and pursue a higher profit in the longer term and eventually reduce the exploration to maximize the profit in the obtained settings at the later parts of the simulation.
4. A Q-Learning process per opposite agent is doing the learning loop throughout the entire simulation run. In Q-Learning terminology, an episode of the algorithm is a single negotiation process and a step of the algorithm is a pass between states of the negotiation process, within the negotiation process.

In order to apply Q-Learning, we need to define the problem in terms of states, action and rewards. These will be defined next.

2.1.2. States

In the OneShot game, our agent tries to maximize its profit by having negotiations where the sell price is higher than the buy price. A negotiation happens in a single step of the "world" (simulation). Above we mentioned that there will be a learning process per opposite agent, thus the Q-Learning model will be applied to a bilateral negotiation process of our agent against another single agent.

During a bilateral negotiation, our agent's "environment" has the following properties:

- Current needs - requirement to sell/buy items as specified by the exogenous contracts;
- The current offer we see from the opposing agent we're negotiating with - this is only a part of the state if we're in the middle of the process (in the beginning, when we're proposing an offer, we still don't have a counter offer to consider)

From these environmental properties we derive the states possible in the system:

Initial state options:

- n - Initial quantity requirement specified by the exogenous contract on the step. It can be a value within 0 to the number of production lines (this is defined by the game rules). This is the initial state in case propose() is called first, i.e. in the beginning of the negotiation the system called our agent to give an initial proposal.

- **($p_{0,}\ q_0,\ n$)** - in addition to the initial quantity requirement, we have an initial counter offer (meaning the opponent was called first and the first time the system called us, it is with the respond() method). p_0 is the unit price proposed and q_0 is the quantity proposed. p_0, q_0 value ranges are dictated by some minimal & maximal value per of these issues, defined by the system per agent.

Intermediate state options:

- **($p_m, q_m,\ p_0,\ q_0,\ n$)** - a state where we previously offered ($p_m,\ q_m$) (unit price of p_m and quantity of q_m) and got a counter offer of ($p_0,\ q_0$) and our current external needs are n. Taking ($p_m,\ q_m$) in the state is not mandatory, but since agents in the general case can be stateful players, taking this into account may improve the learning of their behavior. As specified before, the $p_m,\ q_m,\ p_0,\ q_0$ value ranges are dictated by some minimal & maximal value per one of these issues, defined by the system per agent.

Terminal (final) state options:

- **END** - negotiation ended without acceptance, i.e. there's no deal (either our agent ended it or the opposite one did).
- **ACCEPT** - negotiation ended with acceptance, i.e. there's a deal (either our agent accepted a counter offer or the opposite one accepted our offer - one agent is going to sell to the other).

Note: we eventually represented the prices (p), quantities (q) and needs (n) using mapped indexes and not absolute value; see 2.4.c for more details.

2.1.3. Actions

The actions executed by the model are:

- **(p, q)** - an offer of q items in price p per item; (note - we eventually represented them using mapped indexes and not absolute value; see 2.4.c for more details).
- **"end"** - the agent uses this action in response to a counter offer - in order to end the negotiation without accepting.
- **"accept"** - the agent uses this action in response to a counter offer - in order to accept the counter offer, hence a negotiation ends successfully - a deal is signed.

Action selection (using Q-learning epsilon greedy framework, with additional heuristics and constraints) is the way the agent selects negotiation issues' assignments and responses, per opponent.

2.1.4. Rewards

The utility definition for our agent is as defined by the Q-learning framework : accumulated reward over time. We will define the rewards here, in the context of negotiation. The "rewards" are naturally related to what the agent gains from the negotiation eventually. This means that the profit achieved during a successful negotiation will affect the rewards considered by the algorithm. This makes sense, since we want the agent to learn how to maximize its profit and Q-Learning maximizes the reward, so it is reasonable to directly define the reward using the profit. We get the following immediate reward assignments when associated with moving to states:

State transitions	Reward	Reason
To intermediate state	0	No profit - we didn't do any profit during this transition, nor do we know how to measure it's contribution to making final profit
To ACCEPT state	A function of the profit earned*	Profit is a direct way to estimate reward

*The profit being made is calculated using the following method:

- In case you're in L0 level ("seller" - you profit from selling with a price bigger than the exogenous price):

$$Profit = (agreed\ price_{unit} - exogenous\ price_{unit}) * quantity - production\ cost$$

- In case you're in L1 level ("buyer" - you profit from buying with a price smaller than the exogenous price):

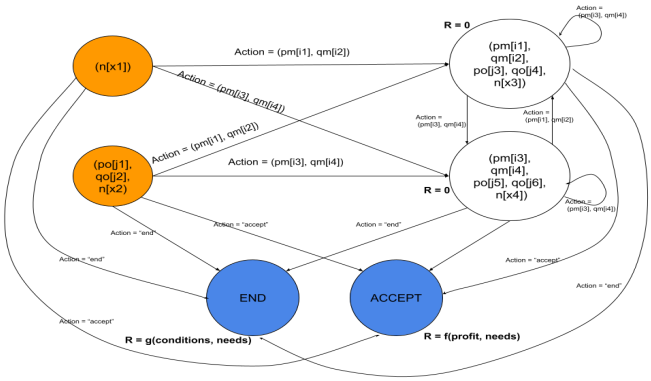
$$Profit = (exogenous\ price_{unit} - agreed\ price_{unit}) * quantity - production\ cost$$

Further reward design is applied in order to make the agent learn to better (and hopefully, faster) assign correct significance and meaning (either positive or negative) during the simulation. The following considerations are taken in account:

1. When the agent is not fulfilling its whole exogenous quantity requirement, or in case it exceeds this amount, it is penalized. We can take this penalty into account in order to make the reward more realistic. Since the deviation is a result of all negotiations in a simulation step, the penalty for the entire deviation is scaled by the amount of negotiations executed successfully so far in the simulation step.
2. In the case that negotiation ends with no acceptance, we tried optimizing the rewarding function, to give different significance to different situations related to transitions into the END state. The situation differs by the following statuses:
   a. Needs: how much we're left with to trade (or how much we exceeded) - being far away from needs=0 in case negotiation fails, is usually penalized
   b. What was the difference between our last offer and the opponent's last offer, in relation to our needs.

2.1.5. Markov Decision Process

As a result from modeling the states, actions and rewards as discussed - we get the following state diagram:



In orange - initial states; In white - intermediate states; In blue - terminal states.

Note that the initial states and intermediate states in this diagram are just examples; They are replicated across all possible values of p,q,n (as discussed before when we presented the states in detail).

Note that we eventually represented the prices, quantities and needs using mapped indexes and not absolute value; see 2.4.c for more details.

2.2. Further Executed Improvements

In order to better improve performance and provide faster learning, we applied additional techniques:

a. **Q table initialization** that "prefers" (gives higher value) to more profitable actions.

b. **Pre-training**: we added the ability to store the Q table learned by the agent and load an existing table from a file. We use a separated table for seller and buyer types of agents.

c. **"Universally"-encoded states and actions:** in order to be able to use the pre-trained Q tables, keeping in mind every simulation can have different properties (different price ranges between simulations for example), we encoded the states and actions into indexes rather than absolute values. I.e. instead of the actual values of prices and quantities appearing in the state/action in the table, on table creation we mapped them to an index (according to their place in the range: for example if price range is 5 to 25, and quantized range is [5,10,15,20,25], we represented them as [0,1,2,3,4]). On other simulations using the pre-loaded Q table, we map the "absolute" states and actions (where absolute values of prices, quantities and needs appear) to the "indexes space" to access/update the corresponding Q-table value. The assumption here is that the relative connections between the prices and quantities will have more weight in decision - which is exploited here since the indexes are keeping the original values inter-relations (high index == high original price, for example). This kind of mechanism enabled us to infer from long pre-training to other, short simulations - which helped us be more well fitted for the actual tournament (where you have to have a good performance on a very early stage). **So an important note here - the states and actions described before are actually not represented by the simulation specific values - each value is represented by a corresponding index in the Q table.**

d. **Heuristics and constraining**: In order to make the algorithm avoid obvious bad situations or try “hunt” for easy profit without learning - which can be very essential in a short tournament where every action we do may count to overall score - we added more heuristics and constraints on action selection, on top of the epsilon-greedy mechanism (which natively the Q-learning algorithm won’t be able to detect, unless trained for every situation and even after that - there’s the chance to “ignore” these cases. Thus the heuristics we added can help address that). For example, avoiding buying too much and becoming bankrupt, or with some probability try to sell at a very high price (trying to exploit an opponent's weakness directly without learning).

2.3. Reference To Agent’s Skills

2.3.1. Risk Management

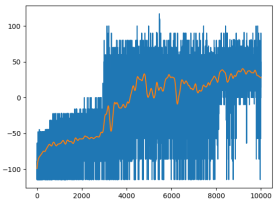
We expect that given enough negotiations and given well-tuned rewards and well-initialized Q-table, the agent will eventually learn a policy that from the one hand tries to maximize profit (as directed by using estimated profit-based rewards and more techniques to avoid over-conceding like in 2.3.4(2.b)) and from the other hand tries to avoid extra risk (as directed by penalizing estimated deviation from buying/selling exactly everything - see 2.3.4(2.a)). We do this through the learning process since there may be delicate situations unpredictable by us that it is best the agent will learn on its own given the environment.
However, together with the learning that may handle situations we can’t expect, the mechanism described in 2.4.d directly tries to address, amongst other purposes, the subject of risk management directly (constraining selection of actions that will bring us to bankruptcy or lose money).

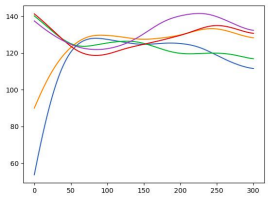
2.3.2. Parallel Negotiations

A separated Q-learning process is done against each opponent. However, all these processes “live” in the same environment in parallel. Thus the different Q-learning processes may influence each other - in terms of left items to trade (“needs”). This means that this dimension of the state can change unexpectedly - transitions in this dimension become more stochastic in nature. However Q-learning is model-free - i.e. it doesn’t need the transition probabilities ahead - thus we can say it can be used in this type of environment. The separated learning processes are affecting each other, but we expect that the learning processes will eventually “learn” to adapt to the environment and to generate optimal reward and not on the account of other negotiations - since the penalties embedded into the rewards are environment-wide and not private to the process (they take the current “needs” in account which is common to all processes) - thus there is attempt to make the different processes be aware of the global agent state and not to “play against each other” but for the greater good of our agent.

3. Evaluation

The agent was put to compete against GreedyOneShotAgent and in most cases performed better in one-on-one controlled scenarios and tournament runs. It was also superior to LearningAgent in most one-on-one controlled scenarios (note: this is true with the original version we submitted for the original date of the tournament - beginning of 7.2021 - but the final pre-trained Q table we submitted may not be optimal for some of these - since we tried hard to win against the agents in the actual online tournaments since then and this was our objective. Also for the actual tournament we added some probabilistic opportunistic behavior in addition to the Q learning - which may harm its performance against these example agents in tournaments but was supposed to help make “easy profit” in the real tournament, if possible). The following figures depict the agent’s learning curves (reward over time). On the left: learning behavior during self-play. On the right: multiple negotiations’ reward over time (one curve per opponent).





4. Further Possible Improvements

The following more sophisticated improvements are possible:
- Hyper-parameter search using an upper reinforcement learning layer or bayesian optimization.
- Examining other continuous space and action state RL algorithms.
- A better way to combine the different parallel negotiation learning processes into one best policy.