LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTER THESIS

Exploring the impact of markets on the credit assignment problem in a multiagent environment

Zarah Zahreddin

Entwurf vom October 3, 2021

LMU
LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTER THESIS

Exploring the impact of markets on the credit assignment problem in a multiagent environment

Zarah Zahreddin

Professor:	Prof. Dr. Claudia Linnhoff-Popien
Supervisor:	Kyrill Schmid Robert Müller
Submission Date:	13. December 2021

I hereby affirm that I wrote this Master Thesis on my own and I did not use any other sources and aids than those stated.

Munich, 13. December 2021

. . . .
Signature

Abstract

Hier kommt der Abstract hin. Hier kommt der Abstract hin.

Contents

1. Introduction

- Motivation
- Goal
- Research Question
- Structure

2. Background

- Describe the technical basis of your work
- Do not tell a historical story - make it short

2.1. Reinforcement Learning

rl components

Sutton and Barto wrote in “Reinforcement learning: An introduction”[SB18] that Reinforcement learning (RL) is based on two components that interact with each other: an environment and an agent, see Figure 2.1. Those interactions take part during a time period with discrete timesteps $t \in \mathbb{N}_0$ until a goal is reached or the ending condition applies. Formally the journey of the agent finding the goal state is described as the Markov Decision Process (MDP) and every method that leads the agent there is a reinforcement learning method. When multiple agents act in the same environment the Markov decision process is called a stochastic game [BBDS10].

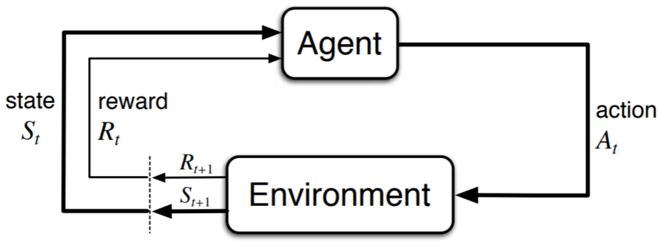


Figure 2.1.: The cycle of agent-environment interaction as shown in “Reinforcement learning: An introduction”[SB18]

sets and values

The state S_t is part of a set S containing all possible environment states. Since its likely that not all actions are valid in each environment state the agents action selection is based on a restricted set $A_t \in A(S_t)$. In a multiagent environment however, every agent chooses its action and adds it into a joint action set, which is executed collectively on the environment [BBDS10]. The reward R_t is element of a set of possible rewards R, which is a subset of real numbers $R \subset \mathbb{R}$. Therefore, the reward can potentially be negative or very low to emphasize a bad action. The general concept of RL, as defined by Sutton and Barto, is to maximize rewards. Thus, unlike machine learning approaches the agent starts with no knowledge about good or bad actions and enhances the decision-making by aiming to improve the reward.

policy

Sutton and Barto continue by defining the agents action selection with respect to the current state as a policy π. They explain further that a policy could be as simple as a lookup table, mapping states to actions or it could contain a complicated search process for the best decision. In most cases however, policies return actions with assigned percentages for the current state. During environment interactions agents gain rewards, which then can be used to update the policy accordingly. For example, should the reward be low or negative it could be interpreted as a penalty. In return the policy $\pi(a \mid s)$ could then be adapted to change to a very low probability for that action in combination with that certain state. So next time the agent finds itself in that state the bad action is not very likely to be chosen again.

value function

While rewards only rate the immediate situation, a value function, i.e. the state-value function $v_\pi(s)$ for a policy π, can be used to estimate the long-term value of a state s. The result is the total accumulated reward an agent could get following that state and choosing actions based on the current policy. States that offer immediate high reward could end in low reward streaks. In the opposite case, a low reward state could subsequently yield high rewards. Therefore, value functions are of great use to achieve the maximum reward.

exploration vs exploitation

The last part to note about RL is that it entails the problem of balancing exploration and exploitation. In order to learn, an agent has to explore the options given. However, since maximizing rewards is the goal an agent could become greedy and exploit its knowledge by choosing actions of which it knows to result in small but positive rewards. If an agent doesn't explore enough the best action sequence will stay hidden. Whereas when an agent always explores without exploiting its knowledge, chances are that the reward will not be optimal.

2.2. Proximal Policy Optimization

intro

TODO: introduce PPO as learning algorithm!

In 2017 Schulman et al. introduced the concept of Proximal Policy Optimization (PPO) in the article “Proximal Policy Optimization Algorithms”[SWD+17]. This section is solely based on that article in order to explain the Algorithm. Policy optimization is the improvement of the action selection strategy π based on the current state s_t. This is achieved by rotating two steps: 1. sampling data from the policy and 2. optimizing that data through several epochs.

TRPO, Advantage func

The origin of PPO lies in a similar approach called Trust Region Policy Optimization (TRPO). TRPO strives to maximize the following function:

$$\underset{\theta}{maximize}\, \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t - \beta\, KL[\pi_{\theta_{old}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \tag{2.1}$$

with $\hat{A}_t$ as an estimator of the advantage function. The advantage function often calculated with the state-value function $V(s)$, a reward r and a discount coefficient λ over a period of Time t

$$\hat{A}_t = -V(s_t) + r_t + \lambda r_{t+1} + \ldots + \lambda^{T-t+1} r_{T-1} + \lambda^{T-t} V(s_T) \tag{2.2}$$

The fraction in the Minuend of (2.1) can be replaced by $r(\theta)$ and represents the probability ratio of an action in the current policy in comparison to the old policy, with θ being a policy parameter. The result of $r(\theta)$ is greater than one, if an action is very probable in the current policy. Otherwise the outcome lies between zero and one. Schulman et al. further describe that TRPO maximizes the "surrogate" objective

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}\hat{A}_t] = \hat{\mathbb{E}}_t[r(\theta)\hat{A}_t] \tag{2.3}$$

However, maximized on its own without a penalty this results in a large outcome and leads to drastic policy updates. problem

In order to stay in a trust region, as the name suggests, a penalty is subtracted TRPO from the surrogate function (2.3). The penalty is the Subtrahend of equation (2.1) and contains the fixed coefficient β. Regardless of the function details and outcome of KL, the coefficient β is hard to choose, since different problems require different penalty degrees. Even in a single problem it could be necessary to adapt the coefficient, due to changes within the setting. PPO

Therefore Schulman et al. introduced

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r(\theta)\hat{A}_t, clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.4}$$

which is very similar to (2.1) but does not require coefficients. The first part of min contains L^{CPI} (2.3). The second part contains a *clip* function which narrows the space of policy mutation with the small hyperparameter ϵ. After applying the clip function $r(\theta)$ lies between $[1-\epsilon, 1+\epsilon]$. Calculating the minimum of the clipped and unclipped probability ratio produces the lower bound of the unclipped $r(\theta)$, preventing the policy to change drastically. PPO Algo

PPO is defined by the following equation

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{2.5}$$

with c_1 and c_2 as coefficients. The authors point out that the loss function $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$ combines the policy surrogate and the value function error term and is necessary once a neural network shares parameters between policy and value function. Finally an entropy bonus S is added to ensure exploration. Schulman et al. continues to show an example of an Algorithm using PPO, see Fig. 2.2. N detonates (parallel) actors collecting data in T timesteps in each Iteration. Afterwards the policy is optimized in K epochs by computing the Loss function (2.5) on the corresponding NT timesteps of data, using a minibatch.

2.3. Deep Q-Learning

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2.2.: Exemplary use of PPO, as shown in “Proximal Policy Optimization Algorithms”[SWD+17]

3. Related Work

- Definition of field of research
- Scientific Scope
- Which comparable work in research exists?
- Separation from other works

3.1. Credit Assignment Problem

intro and comp. problems

Realistic RL scenarios often involve multiple agents solving problems together, for example robots working in warehouses and factories. Such multiagent environments come with many difficulties. On one hand in a scenario where agents work independently it is very probable that they get in each other's way in order to score highest or finish a task, preventing the overall goal to be achieved.

coop problems

In cooperative environments on the other hand, agents share the reward and therefore can not tell who contributed useful actions. Hence, all agents receive the same reward regardless of their contribution, which aggravates learning. The independence problem is discussed in chapter 3.2 whereas the cooperation challenge is the focus point of this chapter.

coop and problem

Sutton and Barto [SB18] define a RL environment as cooperative, when agents execute their actions collectively each timestep but receive one overall reward in return. In this case individual learning is difficult or even impossible. Collective actions may contain bad choices that could be rewarded or, in case of a penalty, good actions that would be punished. Deciding which agent deserves more or less of the common reward is referred to as the credit assignment problem (CAP) [Min61].

CAP definition and kinds

The CAP originated in a one-agent environment that only returned reward once the goal is reached or the terminating condition applied. A popular example of this is a chess game. In 1961, Minsky [Min61] elaborated on this by explaining that a player wins or loses the game, but cannot retrace which decision got him there. Sutton later on decomposed the CAP into subproblems, namely the structural and temporal CAP [Sut84]. He suggests, that the temporal CAP is assigning credit to each chess move by determining when the position improves or worsens, rewarding or penalizing that certain action. On the contrary, the structural CAP is assigning credit to the internal decision that leads to each particular action.

CAP multi

Transferring the single-agent CAP into a multiagent environment Agogino and Tumer [AT04] imply that the problem shifts from being of temporal to structural type. They explain that while a single agent faces the temporal CAP due to many steps taken within an extended time period, in the multiagent case it becomes a structural CAP because of multiple actions in a single-time-step. Since the actions are executed all at once, the problem is now evaluating the decision that lies underneath.

cap solution dr

Over the years many solutions and theories emerged in order to solve various CAP scenarios. An example for a simple approach is the difference reward (DR) [AT04],[NKL18]. The idea is to calculate the reward with the joint multiagent actions as always. In every step however, each agent decomposes that reward by calculating the difference between a new reward and the old one. The new reward is generated with the same actions, only modifying the action of the current agent, setting it to a default or waiting value. With this method each agent has the opportunity to learn how they contributed to the resulting state and reward, enabling individual learning. High DR values indicate lucrative actions of the analyzing agent. The opposite case applies for low valued DRs.

3.2. Markets

intro mixed motive

As described earlier, agents that share an environment and act independently can often hinder each other from reaching the common or individual goal. Sutton and Barto defined a game to be competitive, when agents receive varying reward signals [SB18]. In most cases agents follow a mixed-motive, meaning that their individual rewards could sometimes align and sometimes be in conflict. An environment is purely competitive, when the increase in reward of one agent leads to reward decrease of the others.

SMG details

Schmid et al. introduced in “Stochastic Market Games” [SBM$^+$21] concepts adding incentives when agents act cooperatively in mixed-motive settings to improve the overall rewards for all participants. The idea of a Stochastic Market Game (SMG) is to enable dominant cooperative strategies through a global and impartial trading market. A stochastic game becomes a SMG if two conditions are met. First, the environment actions of agents are extended with market actions. Second, the reward function adjusts the calculated rewards based on agreements met in the market executions. Furthermore Schmid et al. defined two types of markets: unconditional and conditional markets.

sm

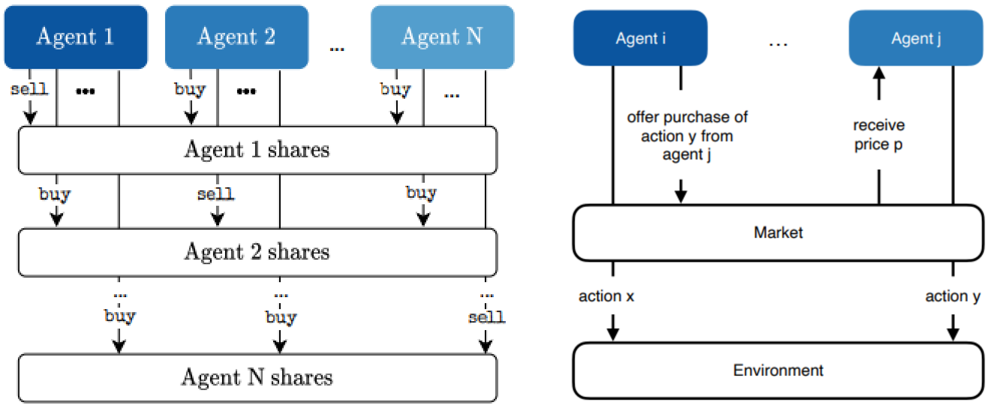
They compare the concept of unconditional markets to companies and shareholders, since shareholders do not need to fulfill any conditions to receive the dividends. In unconditional SMGs both companies and shareholders are agents that buy and sell shares as market actions. Figure 3.1a shows such a shareholder market (SM). During each timestep, every agent has the possibility to put their share on the market or to announce a buying offer directed to another agent.

sm transaction

If the buying offer coincide with a share that is up for sale in the same step, a market transaction is registered. From there on out the shareholder participates in the reward of the transaction agent by a fixed dividend d. Schmid et al. mention that an optional price p can be defined as a price a seller receives from the buyer upon each share purchase. They claim however, that agents with high rewards are very likely to gift their shares in order to align the goals of the other agents with their own. Shareholders profit from the success of the selling party through the dividends.

am

On the contrary, the authors define conditional markets similar to purchase contracts, where buyers pay a fixed price p to sellers when they in turn meet the buyers demand. A proposed conditional SMG is the so called action market (AM). In this case actions are extended with a buying offer, containing one expected action from one specific agent, see figure 3.1b.



(a) Shareholder market (taken from "Stochastic Market Games"[SBM+21])

(b) Action market

Figure 3.1.: Illustrated Markets as defined in "Stochastic Market Games"[SBM+21]

am transaction

A purchase is established if the specified agent happens to execute the environment action the buyer expected. It is important to emphasize that the matching happens during one timestep, leaving it to chance, whether purchases take place. Hence, agents do not know in advance if and what action another agent could be buying from them. Despite this uncertainty, the researchers showed, that both market implementations yielded promising results. An increase of the overall rewards of participating agents in mixed-motive games was seen.

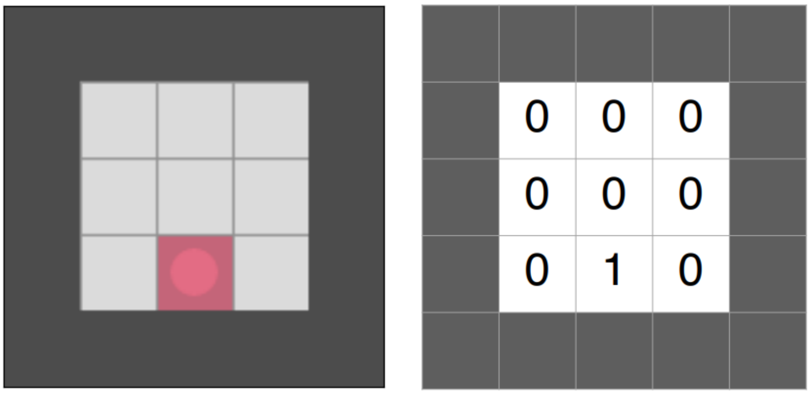
4. Approach

- What is your plan?
- How do you proof that it worked? -> Metric and Experiments

4.1. Coloring Environment

origin and intro

A RL environment is a versatile and unbiased instance, that can can be used to visualize agent behavior and environmental changes. In figure 4.1a, the environment used in this work is presented. It originated from an openAI project called “Minimalistic Gridworld Environment” [CBWP18], which is designed for one agent whose main goal is to solve labyrinth puzzles. For the purpose of this research however, the environment is changed heavily, becoming the “Coloring Environment”. Multiple agents can act in the new instance to try and achieve a new goal - to color all walkable cells.



(a) Human visualization of the coloring environment. A dot represents one agent. Cells change their color when agents moves on them.

(b) Simplified agent observation of the current environment state. The number 1 represents a colored cell.

Figure 4.1.: Representations of the coloring environment

cell objects

Figure 4.1b shows a simplified environment observation an agent processes each timestep. Every environment cell holds information about the object it represents, being either Walls, Floors or Agents. Furthermore, each object contains information about its current color, whether or not it is accessible for an agent and, in case of a floor tile, if it is colored.

cell objects

Floor cells keep the coloration state in binary form, as displayed in 4.1b, with a 1 signalizing that the cell is colored. The environment reacts to agents movements by coloring the cells they visit. Agents successfully solve the environment once all fields are colored. Otherwise agents loose by using up a limited amount of steps. If a cell is already in coloration state 1 and an agent walks over it again the bit is switched and the cell is reset to 0, removing its coloration. Besides moving up, down, left and right an agent can also execute the action wait, to stay in place.

cooperative multiagents

When multiple agents act in the coloring environment, each one has a different random color. In the human representation (figure 4.1a) cells adopt the color of the capturing agents. Yet, The primary focus in cooperative agent compositions is only the binary state. All agents receive the same maximum reward when the grid is fully colored, making it irrelevant what colors the cells have.

mixed-motive multiagents

The opposite is the case in competitive scenarios. In mixed-motive settings for example, agents only gain high rewards once the grid is fully colored, with the twists that it depends on their contribution. The reward is generated by looking up the percentage a color is present and assigns that value as reward to the corresponding agent. In a fully competitive scenario the reward calculations stay the same, only disabling the bit switching. Therefore, agents can directly capture already colored cells when they walk over them.

4.2. Reward Calculations

activation line

The allocation of rewards is closely related to the composition of the agents, which can be specified by the user in training or visualization runs. In addition to the composition, the environment shape can be set, a number of agents selected and more. A basic example command for a training run is shown in listing 4.1.

Code Listing 4.1: Exemplary command to execute training with three agents in a coloring environment with PPO as algorithm

```
$ python -m Coloring.scripts.train
    --algo ppo
    --model ppo-training
    --env Empty-Grid-v0
    --grid-size 9
    --agents 3
    --max-steps 350
    --setting mixed-motive
```

command algo and model

The "algo" parameter can be either "ppo" or "dqn" to choose a learning algorithm. This argument is the only required setting for training. All other configurations, including those not listed in 4.1, have default values and are discussed in the sections 4.3 and 4.4. An overview of all training parameters and their default values is listed in Appendix

A. The model defines the name for a destination folder, in which all logs, recordings and status updates are stored. command env

Line 4 and 5 configures the environment. Alternatively to the empty grid option of "env", as shown in figure 4.1a, four homogeneous rooms can be generated with "FourRooms-Grid-v0" to increase the difficulty. The rooms are of the same size and each room is accessible to all adjoining neighbors by one opening, which is random and changes in each episode. The overall size of the grid is set in Line 5, however all grids in every layout options have outer walls that narrow the area in which agents can move. settings

The amount of agents that act in the environment is set through the argument "agents" and the maximum quantity of steps they can execute is defined with "max-steps". To gain the maximum reward, the agents need to color the whole field before they run out of steps. Lastly, the argument "setting" specifies the composition of the agents. If no setting is set the agents work cooperatively. In the example of 4.1 the setting "mixed-motive" is chosen. The last option here is "mixed-motive-competitive". environment reward

In each step agents get separate environment rewards based on their coloration. Agents that color a field receive a reward of 0.1, whereas agents that reset a field get a penalty of -0.1. In the competitive mode agents can not reset fields and therefore receive no penalty. In this case the contrary happens, capturing cells of other agents, yield a positive reward of 0.1. Agents that just wait get a reward of 0. All rewards are written into a list and returned by the environment. The position in the list indicates the receiving agent. In algorithm 1 the process of adapting the initial environment rewards with the specified training arguments is summarized.

Algorithm 1: Reward calculation each step

observation, rewards, done, info = environment.step(actions)
if *cooperative setting* **then**
  rewards = calculate one cooperative reward
end
if *market specified* **then**
  rewards = execute market actions and return transaction rewards coop reward
end
if *done* **then**
  rewards = calculate final rewards
end
return observation, rewards, done, info

First, for a cooperative setting a new homogeneous reward needs to be calculated out of the environment rewards. The calculation for that is summing up all list values and checking if they exceed an upper or lower bound, for instance $[-0.1, 0.1]$. If that is the case then the new reward is set to the corresponding limit, otherwise the sum is taken as is. This step is necessary, due to more participating agents possibly leading to a really big or really small sum. This in turn could decrease the importance of the final reward for reaching the environment goal. This calculation is skipped for a mixed agent composition. market actions

Second, the market transactions are executed, if a shareholder or action market is specified. The market details are discussed in Section 4.4. One thing to note here is that agents can execute transactions in each step, spending their current reward on items for sale or receive the purchase price from buyers. Therefore the rewards change in this step too.

done calculations

Lastly, the final reward is calculated when done is set to true. That is the case when the environment goal is reached or all steps are used up. Algorithm 2 shows the executed calculations of that case. Again the first thing to check is whether the agents work together. If not, each agents' grid coloration percentage, based on their color presence, is added to their reward. Otherwise the environment goal condition is checked. If the grid is fully colored the value one is added to each agent reward, since everyone gets the same feedback in cooperation. Finally, the final market calculations are included into the rewards, see chapter 4.4 for details.

Algorithm 2: Final reward calculation

```
if mixed setting then
    for each agent do
        rewards[agent] += agent color percentage on the field
    end
else
    // cooperative setting
    if grid fully colored then
        for each agent do
            // add the maximum value of 1 to each agent reward
            rewards[agent] += 1
        end
    end
end
if market specified then
    rewards = final market adjustments executed on rewards
end
return rewards
```

4.3. Learning Process

Independent learning

In order to compare different settings and agent compositions easily, each agent manages its own learning improvement, observation and action selection. Therefore all calculations and estimations are executed independently, for instance policy updates and value estimations. They also set up their own neural networks and optimizers and update them only with their own values. However, observations still connect the agent experiences, by including the positions of all agents on the grid and reacting to their joint actions in each step.

process structure

Depending on the learning algorithm the corresponding class is instantiated by the training script, as shown in Figure 4.2. PPO and DQN both extend a base class that provides some abstract methods and a multiprocessing operation to execute actions on several environments at once. If specified, the base class returns data, allowing the training script to create recordings and log files to enable evaluation.

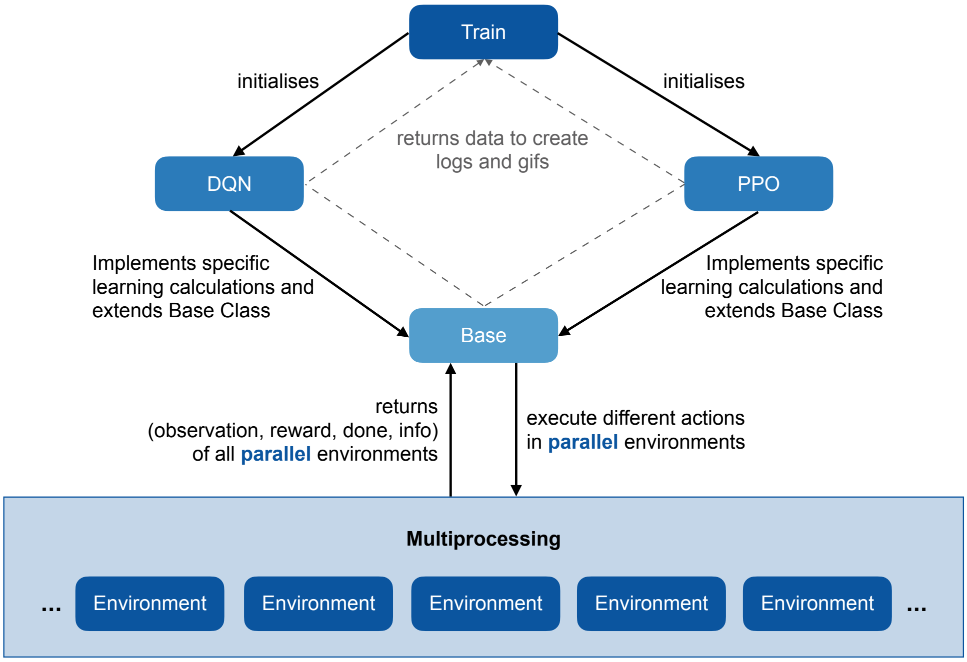


Figure 4.2.: The training structure

training setup

First, the training begins by creating n environments based on the `--procs` setting of the training command, see Appendix A for the parameter list. Each environment has the same configurations, set through the arguments, for example `--grid-size`, `--agents` and `--agent-view-size`. Second, the amount of `--frames` is taken from the parameters, defining a loop until that number is reached. In this case frames are equivalent to agent steps. During the loop, the defined training algorithm `--algo` is executed.

base class

Since both, DQN and PPO have similar procedures, they share a function in the base class. In it experiences are gathered through a certain amount of actions which are executed on parallel environments. The action amount is set with `--frames-per-procs`. During that period details like gained rewards, environment observations, actions and more are stored in base class variables that are accessible by the learning algorithms. When all experience actions are executed, the base variables are reset and log values are returned, as shown in Figure 4.2. Afterwards the frame counter in the training script is updated. The training loop ends when the frames counter is greater than the argument frames. Otherwise more experience batches are gathered.

action selection

Both learning algorithms implement their own action selection method. The PPO implementation relies on an actor-critic neural network, letting the actor part calculate a probability distribution of the action space. In case of DQN a linear neural network assigns Q values to actions, choosing the maximum value with an epsilon greedy practice. In both variants the action selection results in one action for each agent and for each environment. The environments finally receives a joint action array that merged the agents action.

dqn learning

After the action selection, in a DQN training, the target network is updated with a transition, saving observation, action, reward and the observation that followed. With `--initial-target-update` a value is set, stating how many transitions are saved before the model is optimized the first time. Afterwards an update occurs every frame. In the update a batch of size `--batch-size` is randomly selected out of the saved transitions to calculate the Huber Loss. After `--target-update` steps the target network is update with a new policy.

5. berechnung ppo + parameter 6. berechnung dqn + parameter Independent learning! each agent has own actor/critic or complete network agent observation explanation PARAMETERS! 16 parallel envs with frames, nn takes observation how the values and percentages are calculated

4.4. Market Settings

MARL Challenges

TODO: RL Skizze 2.1 erweitern wie agenten pro schritt am markt handeln und was in done passiert! Each of the three compositions presented in chapter 4.1 lead to learning problems or game losses. Cooperation may reward misbehavior, namely field resetting, leading to the CAP of chapter 3.1. In mixed-motive or fully competitive settings the overall goal may be never reached due to greediness or disorder. This research further compares the effects of markets not only on competitive settings as suggested by Schmid et al. [SBM+21], but rather on all three configurations.

action space

As mentioned earlier, agents have five possible action choices each timestep: moving up, down, left, right or simply to wait. Adding a market to the stochastic game expands this one dimensional action space into a three dimensional action space. - am

- am-goal
- am-goal-no-reset
- sm
- sm-goal
- sm-goal-no-reset

5. Results

- Result presentation
- Description of images and charts
  - One Agent Environment vs MARL
  - Best cases of reward, trades, grid coloration, field resets
  - Worst cases of above
  - influence of markets

6. Discussion

- Are the findings as expected?
- Why are the things as they were observed?
- New experiments that provide further insights
- Make your results more comprehensible
  - challenges of markets (i.e. agents didn't need to sell shares/buy actions)
  - final reward is not necessary easy to interpret (did agent do good actions or did he just pick good market actions that sold)?
  - maybe markets need to be more specific (let agents know what others want to buy before choosing action!) and less based on chance - and/or more dynamic, i.e. instead of fixed prices agents can decide what to pay for actions/shares, so that they can for them self decide how important the trade is, and in case of shareholder market, maybe enable multi share purchase?

7. Conclusion

(Briefly summarize your work, its implications and outline future work)
- What have you done?
- How did you do it?
- What were the results?
- What does that imply?
- Future work

Appendix A.

Training Parameters

```
required arguments:
--algo ALGO           Algorithm to use for training. Choose between 'ppo' and 'dqn'.

optional arguments:
-h, --help            show this help message and exit
--seed SEED           random seed (default: 1)
--agents AGENTS       amount of agents
--model MODEL         Name of the (trained) model, if none is given then a name is
                      generated. (default: None)
--capture CAPTURE     Boolean to enable capturing of environment and save as gif
                      (default: True)
--env ENV             name of the environment to train on (default: empty grid)
--agent-view-size AGENT_VIEW_SIZE
                      grid size the agent can see, while standing in the middle (default:
                      5, so agent sees the 5x5 grid around him)
--grid-size GRID_SIZE
                      size of the playing area (default: 9)
--max-steps MAX_STEPS
                      max steps in environment to reach a goal
--setting SETTING     If set to mixed-motive the reward is not shared which enables a
                      competitive environment (one vs. all). Another setting is
                      percentage-reward, where the reward is shared (coop) and is based
                      on the percanted of the grid coloration. The last option is
                      mixed-motive-competitive which extends the normal mixed-motive
                      setting by removing the field reset
                      option. When agents run over already colored fields the field
                      immidiatly change the color the one of the agent instead of
                      resetting the color. (default: empty string - coop reward of one if
                      the whole grid is colored)
--market MARKET       There are three options 'sm', 'am' and '' for none. SM =
                      Shareholder Market where agents can auction actions similar to
                      stocks. AM = Action Market where agents can buy specific actions
                      from others. (Default = '')
--trading-fee TRADING_FEE
                      If a trade is executed, this value determens the price (market type
                      am) / share (market type sm) the agents exchange (Default: 0.05)
--frames FRAMES       number of frames of training (default: 1.000.000)
--frames-per-proc FRAMES_PER_PROC
```

```
                      number of frames per process before update (default: 1024)
--procs PROCS         Number of processes/environments running parallel (default: 16)
--recurrence RECURRENCE
                      number of time-steps gradient is backpropagated (default: 1). If >
                      1, a LSTM is added to the model to have memory.
--batch-size BATCH_SIZE
                      batch size for dqn (default: ppo 256, dqn 128)
--gamma GAMMA         discount factor (default: 0.99)
--log-interval LOG_INTERVAL
                      number of frames between two logs (default: 1)
--save-interval SAVE_INTERVAL
                      number of updates between two saves (default: 10, 0 means no saving)
--capture-interval CAPTURE_INTERVAL
                      number of gif caputures of episodes (default: 10, 0 means no
                      capturing)
--capture-frames CAPTURE_FRAMES
                      number of frames in caputure (default: 50, 0 means no capturing)
--lr LR               learning rate (default: 0.001)
--optim-eps OPTIM_EPS
                      Adam and RMSprop optimizer epsilon (default: 1e-8)
--epochs EPOCHS       number of epochs for PPO (default: 4)
--gae-lambda GAE_LAMBDA
                      lambda coefficient in GAE formula (default: 0.95, 1 means no gae)
--entropy-coef ENTROPY_COEF
                      entropy term coefficient (default: 0.01)
--value-loss-coef VALUE_LOSS_COEF
                      value loss term coefficient (default: 0.5)
--max-grad-norm MAX_GRAD_NORM
                      maximum norm of gradient (default: 0.5)
--clip-eps CLIP_EPS   clipping epsilon for PPO (default: 0.2)
--epsilon-start EPSILON_START
                      starting value of epsilon, used for action selection (default: 0.9
                      -> high exploration)
--epsilon-end EPSILON_END
                      ending value of epsilon, used for action selection (default: 0.05
                      -> high exploitation)
--epsilon-decay EPSILON_DECAY
                      Controls the rate of the epsilon decay in order to shift from
                      exploration to exploitation. The higher the value the slower
                      epsilon decays. (default: 1000)
--replay-size REPLAY_SIZE
                      Size of the replay memory (default: 100000)
--initial-target-update INITIAL_TARGET_UPDATE
                      Frames until the target network is updated, Needs to be smaller
                      than target update! (default: 10000)
--target-update TARGET_UPDATE
                      Frames between updating the target network, Needs to be smaller or
                      equal to frames-per-proc and bigger than initial target update!
                      (default: 100000 - 10 times the initial memory!)
```

List of Figures

List of Tables

Code Listings

Bibliography

[AT04] Adrian K Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *AAMAS*, volume 4, pages 980–987, 2004.

[BBDS10] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[CBWP18] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018. Accessed on September 2021.

[Min61] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[NKL18] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. 2018.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[SBM+21] Kyrill Schmid, Lenz Belzner, Robert Müller, Johannes Tochtermann, and Claudia Linnhoff-Popien. Stochastic market games. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 384–390. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[Sut84] Richard S Sutton. *Temporal credit assignment in reinforcement learning.* PhD thesis, University of Massachusetts Amherst, 1984.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.