Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Nome corso di Laurea

A STUDY ON TAXATION POLICY ON HETEROGENEOUS NON-ANALITICAL BEHAVING AGENTS

Relatore:
Chiar.mo Prof.
NOME RELATORE

Presentata da:
NOME LAUREANDO

Sessione
Anno Accademico

Introduction

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

What is Foundation

In this chapter, I am going to introduce the package developed by Alex Trott and Stephen Zheng and provided by Salesforce called Foundation[5]. This package offers the possibility to create simulations with multiple agents that interact in a 2D world. At first, there will be a presentation of the gather-and-build simulation with the relevant entities and dynamics. Furthermore, the basic agents will be dismantled in their main components to better understand their action space and scope, and finally, there will be a small discussion on the behavior of the policymaker.

1.1 Gather and trade

The simulation that will be used through the dissertation is called gather and trade. This simulation takes place in a 2D map that represents the world where the agents live and interact. The shape of the world is a 25x25 grid where are disposed various kinds of entities. Within this world, 4 agents are free to move around, gather resources, trade them and use them to build houses. These agents are different for their skill level, allowing them to have higher/lower rewards for their actions. A fifth agent, called policymaker, is asked to tax and redistribute the income of the four previous agents, based on information

about their endowments, but not their skill.

1.1.1 World and entities

As aforementioned the map is a 25x25 grid where are present some entities. Some of these are visible in the world as water or stone, others are just present in the agent's endowment as coins or labour. The complete set of entities is:

- Water
- House
- Source Block (Wood and Stone)
- Wood and Stone
- Coins
- Labor

Water is a simple world block that has no use other than avoiding agents passing through. We can see from Figure 1.1 that the water is used to divide the world into 4 macro-areas, each one with different resources. A House is a block that is not present at the beginning of the simulation, but it can be built by agents in an empty block, agents can walk through houses.

Source blocks are two entities that spawn stochastically resources, namely wood, and stone, as we can see from Figure 1.1 in the four areas divided by the water we have a zone with both wood and stone source block, two other areas with just one kind of source block and one that has no resources at all.

Coins are the measurement of the value produced in the world. Coins are generated when a house is built, the agent that builds the house is rewarded with a certain amount of coins that varies with the skill level.

Labor is a measurement of the total effort exerted by an agent, it is generated every time an agent takes an action and is percieved as a dis-utility from the agent.

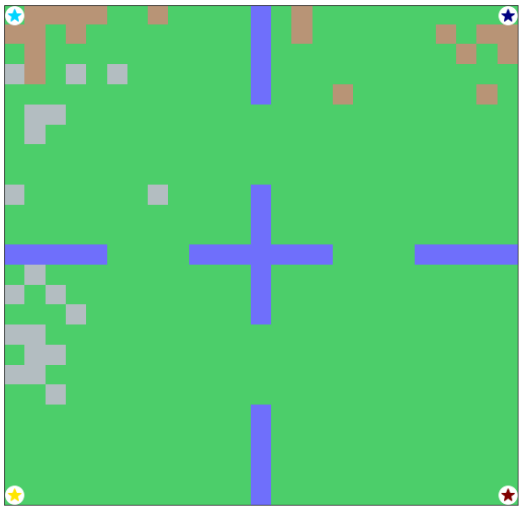

Figure 1.1: Rendering of the world at inital conditions: *this is a rendering of the world map at the first timestep of the simulation, blue blocks are water, brown blocks are wood sources, grey blocks are Stone sources. The four corners are the starting positions for the 4 agents.*

1.1.2 Game dynamics

A problem that has a continuous flow of agent-environment interactions can be formalized by a Finite Markov Decision problem [4]. In particular, this simulation is a partial-observable multi-agent Markov Games (MGs). The problem is defined by the tuple $(S, A, R, \gamma, o, \mathcal{T})$ where S is the state space, A is the action space, R the reward space and gamma a dicount factor.

The simulation is composed of a series of episodes, each of length H time-steps. At every point in time $t \in [0, H]$ the episode is characterized by a state s_t, representing the current world environment, every agent performs an action $a_{i,t}$ given the current partial observation $o_{i,t}$ of the world state, and receives a reward r_{it}. Afterwards the environment transits to a new state s_{t+1} according to the transition distribution $\mathcal{T}(s_{t+1}|s_t, \boldsymbol{a}_t)$. This chain of interactions state-action-reward/state carries on until the end of an episode.

$$s_0 \rightarrow_{\boldsymbol{o}_0} \boldsymbol{a}_0 \rightarrow r_1, s_1 \rightarrow_{\boldsymbol{o}_1} \boldsymbol{a}_1 \rightarrow \ldots \rightarrow_{\boldsymbol{o}_{H-1}} \boldsymbol{a}_{H-1} \rightarrow r_H, s_H$$

Here $\boldsymbol{a}$ and $\boldsymbol{o}$ are the vectorized observations and actions for all the five agents. Given the particular structure of the simulation every single agent will receive an observation at every time-step (different for everyone, more on that later), but only at the 4 basic agents will be asked to act, the policymaker will act only upon a certain condition met. In this case, the episode lasts for 1000 time-step and the policymaker is asked to act (tax the other agents) every other 100 steps. The existence of multiple episodes is necessary for the 4 agents and the policymaker to define their own optimal policy $\pi_i(o_{i,t}, h_{i,t-1}; \theta_i^*)$, this optimization process will be discussed in the RL chapter.

1.1.3 Agents

Fromthe information above we know that the four basic agents are endowed with labor, coins, wood and stone. They live in the world map, can act within it and their objective is to maximize their γ-discounted utility function. Now I will describe in more detail the agents starting from the information that they receive at each time-step, then talking about the actions that they are allowed to take and finally about their objective.

Observation space: Given that this simulation is a partial-observable multi-agent Markov Game, the observation that agent i receive at time t is not complete but partial, this information can be summarized in the following way:

- $o_{i,t}^{\text{world state}}$: world map situation surrounding the agent, this is limited to the 11x11 grid around the agent i.
- $o_{i,t}^{\text{market state}}$: full information about the market state for wood, stone and available trades.
- $o_{i,t}^{\text{agent}}$: public resources and coin endowments (this information is also available to the policy maker), private labor performed and skill level.
- $o_{i,t}^{\text{tax}}$: tax structure

- $o_{i,t}^{\text{other}}$: other information (ex. action mask)

the full observation space can be seen in Table 5

Action space: The agent can take one action per time-step and can choose this action from the 50 listed below:

- **Movement**: 4 actions for the basic movements N, S, E, W
- **Gather**: 1 action for gathering resources
- **Trade**: 44 actions for trading resources
- **Idle**: 1 action that does nothing

The movement's actions along with gather do not need much of an explanation, these are simple actions that increases labor by 0.21 units each time pursued. The building action requires the agent to consume (destroy) one unit of wood and one unit of Stone, as a consequence he gains 2.1 units of labor and an amount of coin that depends on his skill level. The most complicated set of actions are the one that rules trading. Each one of them is a combination of the 11 price levels [0,1,...,10] that the agent is willing to (pay/request) for each side (bid/ask) and for each resource (wood/stone). A trading open action remains open for 50 turns, if in this time span it is matched by the corresponding counteraction at the same price (a bid for an ask and vice versa) then the trade takes place and each agent gets 0.05 unit of labor.

Furthermore, notice that during the episode agents might incur into a series of inconclusive actions, such as moving north while at the north border of the map or building a house without the required wood and stone. The so called action mask, present in the observation space, is used in the learning process to avoid wasting time exploring these possibilities. It is an array of binary values of length 50 that "masks" all meaningless actions.

Agent objective Agents in the simulation earn coins when building houses or trading goods, the utility for the four agents is an isoelastic utility:

$$u_i(x_{i,t}, l_{i,t}) = crra(x^c_{i,t}) - \vartheta_k l_{i,t}\,, \quad crra(z) = \frac{z^{1-\eta} - 1}{1-\eta}\,,\ \eta > 0 \tag{1.1.1}$$

Where $l_{i,t}$ is the cumulative labor associated with the actions taken up to time t, $x^c_{i,t}$ is the coin endowment and ϑ is a function utilized for labor annihilation with the following specification $\vartheta_k = 1 - exp\left(-\frac{episode\ completitions}{energy\ warm\text{-}up\ constant\ (k)}\right)$. This variable will play an important role during the two-step optimization process purposed in the original paper. In particular, during phase 1 of training, the labor cost is annihilated to help agents avoid sub-optimal behaviors. And η determines the degree of non-linearity of the utility. This utility function is assumed to be the same for all the agents.

The maximization problem is solved for a rational behaving agent by optimizing the total discounted utility over time,

$$\forall i\ :\ \max_{\pi_i} \mathbb{E}_{a_i \sim \pi_i, \boldsymbol{a}_{-i} \sim \boldsymbol{\pi}_{-i}, s' \sim \mathcal{T}} \left[\sum_{t=1}^{H} \gamma^t r_{i,t} + u_i(x_{i,0} l_{i,0}) \right] \tag{1.1.2}$$

with $r_{i,t} = u_i(x_{i,t}, l_{i,t}) - u_i(x_{i,t-1}, l_{i,t-1})$ being the istantaneous reward of agent i at time t. Equation 1.1.2 illustrates a multi-agent optimization problem in which actors optimize their behavior at the same time since the utility of each agent is dependent on the behavior of other agents. Another agent, for example, may deny an agent access to resources, limiting how many houses the agent can create in the future and hence its utility. While computing equilibrium for complicated environments like this is still out of reach, we will see later how RL may be utilized to produce meaningful, emergent behaviors.

1.1.4 Policymaker

The policymaker, or social planner, differs deeply from the previous agents. Being the focus of the research question its structure and behavior change a lot in every single simulation. Due to the complexity of creating a multi-agent neural network with different objectives I was not able to introduce

a policymaker that behaves according to an RL algorithm. This makes the policymaker less interesting from a structural point of view but its behavior is still meaningful for the results it produces.

The only observation needed is the number of coins that each agent possesses. With this value is possible to place an agent within a tax bracket and tax it accordingly to some predefined values.

There will be five simulations in this thesis:

- US taxation
- Italian taxation
- Free market
- Communism
- Flat tax

and the policymaker will act accordingly to these taxation systems. The US taxation is based on the 2018 US federal tax and brackets, and the Italian is based on the 2020 INPS tax and brackets. The brackets are created in such a way that 1 coin in the simulation is equal to 1000 $ or €863.93 according to the tax system.

Chapter 2

Reinforcement Learning

Reinforcement learning is the process of learning what to do to maximize a numerical reward. A mechanism where the learner is not told what action to take, but instead must discover which action yields the highest reward by trying them. In this case, actions might affect not only the immediate reward but also future situations and all the subsequent rewards. These two characteristics – Trial-and-error search and delayed reward – are the two most important distinguishing features of reinforcement learning.

Reinforcement learning differs from supervised learning since training is not based on an external dataset of labeled examples, where each situation (observation) is labeled with the correct action to perform (often identify a category). RL, although one might erroneously think the opposite, is also different from unsupervised learning. The main objective for unsupervised learning is to find hidden structures in an unlabeled dataset, whereas RL's main objective is to maximize a reward signal.

In this chapter, we will see the definition of a Markov decision process and those of the state and action value function. Afterward, we will observe how optimization is carried out in an approximate solution method, in particular in the case of the proximal policy optimization since it is the technique utilized

for training the neural network.

2.1 Finite Markov Decision process (MDPs)

Finite Markov decision processes are a class of problems that formalize subsequent decision making, where not only the influence of the action is exerted on immediate reward but also on those in the future. MDP's are suited for RL since they frame the process of learning through repeated interaction between an agent (decision maker), and an environment (ruled by fixed state transition function).

More specifically an agent is asked to take an action $a_t \in \mathcal{A}$ at every time step $t = 0, 1, \dots$. To do so the agent is provided with an observation of the current environment's state $s_t \in \mathcal{S}$ and a reward $r_t \in R$ from the previously performed action. Afterwards the environment update it s state following a transition distribution $\mathcal{T}(s_{t+1}|a_t, s_t)$ and a numerical reward $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$. This process is reproduced every subsequent timestep, this concatenation of interaction is a MDP.

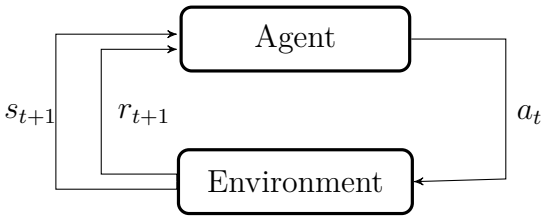

Figure 2.1: Agent enviroment interaction dynamic: *this is the basic dynamic of the interaction between the agent and the enviroment in a MDP, at time t the agent provides an action a and the enviroment respond with a state s and rewad r that are used from the agent to decide the action at t+1 and so on.*

Objective and Rewads: The main objective of RL is to maximize the total number of rewars it receive. This reward r_t is passed from the environment to

the agent at every timestep as a consequence of his actions. In the case of the Gather and Trade the reward is $r_{i,t} = u_i(x_{i,t}, l_{i,t}) - u_i(x_{i,t-1}, l_{i,t-1})$.

Since the agents want to maximize the total upcoming reward we can write this value simply as the sum of all the future rewards.

$$U_t \doteq r_{t+1} + r_{t+2} + ... + r_H,$$

Where H is the total length of the episode, and at the time step $t = H$ the episode ends. This state is called the terminal state and is a particular state because regardless of the final condition of the agent it reset the environment to the initial condition and restart completely the episode. Another specification for the total reward can implement the concept of discounting, which is more appropriate in the case of economical simulations, thus within the experiments the agent has to maximize his future discounted utility:

$$U_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{H-t-1} r_H, \tag{2.1.1}$$

the discount rate $\gamma \in [0, 1]$ determines the value that the agent assigns to the future reward, a reward received at k timesteps in the future is only valued γ^{k-1} times what it would be valued today. When the value of γ approaches 0 the agent is more "myopic" and puts most of his interest in immediate rewards, while if it approaches 1 the interest is more projected in the future due to the stronger impact of future rewards.

Value functions: One of the most important elements of RL is the value function. The estimation of this function is one of the crucial points of RL, in fact it tries to quantify the expected return of the rewards it expects to receive. Furthermore, the expected reward depends on the action that the agent decides to take, thus the value function are defined in terms of policies, which are acting behaviors.

If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that the agent takes the action $a_t = a$ given the state $s_t = s$. The aim of RL is to change the policy based on experience across episodes to find an optimal behavior.

We can write a value funciton for a state s under the policy π. This function is the expected return when the inital state is s and the policy followed is π.

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[U_t | s_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{H} \gamma^k r_{t+k+1} \middle| s_t = s \right], \quad \text{for all } s \in \mathcal{S} \tag{2.1.2}$$

$v_\pi(s)$ is called the state-value function for policy π. Following from this equation, it is possible to define the value of taking an action a in the state s following the policy π:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[U_t | s_t = s, a_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{H} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right], \tag{2.1.3}$$

and $q_\pi(s,a)$ is called the action-value function for policy π. The important concept here is that the value functions in 2.1.2 and 2.1.3 can be estimated from experience.

There are multiple strategies to determine an optimal behavior starting from the evaluation of these functions, we can divide these ways in two main groups, tabular solution methods and approximate solution methods[4]. For the former we have Montecarlo methods, Dynamic programming, Temporal-difference learing, n-step bootsrap and others. While for the latter we have On/Off-policy methods with approximation and policy gradient methods.

For the purpose of this thesis we are going to foucs only on policy gradient methods and a particular set of optimization policy called proximal policy optimization.

2.2 Approximate solution Methods

The approximate solution methods are sets of strategies thought for those problems, such as ours, where the set of possible states is enormous. It is very likely that every state encountered in a simulation will never have been encountered before. Thus, to make meaningful decisions there is the need to be able to generalize from previous state that are, to some extent, similar. This is accomplished by borrowing the exising generalization theory, usually by using the process of function aproximation. However, we are going to use policy gradient methods that do not necessarily need to estimate a value function through function aproximation.

2.2.1 Policy gradient Methods

Policy gradient methods are a set of parameterized policies that can select actions without the use of a value function. The method consists in calculating the estimator of the policy gradient and feeding it into a stochastic gradient ascent algorithm. We denote with $\pi_\theta(a|s)$ the stochastic policy such that

$$\pi_\theta(a|s) = \pi(a|s,\theta) = Pr\left\{a_t = a|s_t = s\,,\,\theta_t = \theta\right\} \tag{2.2.1}$$

for the probability of choosing action a at time t given the state s_t and the parameters θ. The aim of the policy gradient methods is to maximize a perfomance measure $J_t(\theta)$. This is done by updating the parameters θ with a gradient of the performance itself:

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{2.2.2}$$

2.2.2 Proximal Policy Optimization Algorithms

Proximal policy optimization algorithms (PPOs) are a family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent.

PPO utilizes as a performance measure the advantage $A(s,a) = q(s,a) - v(s)$, which represents how good an action is compared to the average action in a specific state.

It also relies on the ratio of the policy that we want to refine to the older policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{2.2.3}$$

The surrogate objective function that we optimize with PPO is

$$\mathcal{L}^{CLIP}(\theta) = \hat{E}_t \left[min(r_t(\theta)\hat{A}_t \, , \, clip(r_t(\theta) \, , \, 1-\epsilon \, , \, 1+\epsilon)\hat{A}_t) \right] \tag{2.2.4}$$

The maximization of this surrogate funcition takes the advantages of trust region policy optimization (TRPO) [2], adjusting the parameters in the dierction of the ratio $r_t(\theta)$, and maintaining this change bounded in a "clipped" region to avoid drastic refinements of the policy. This is shown well in the Figure 2.2

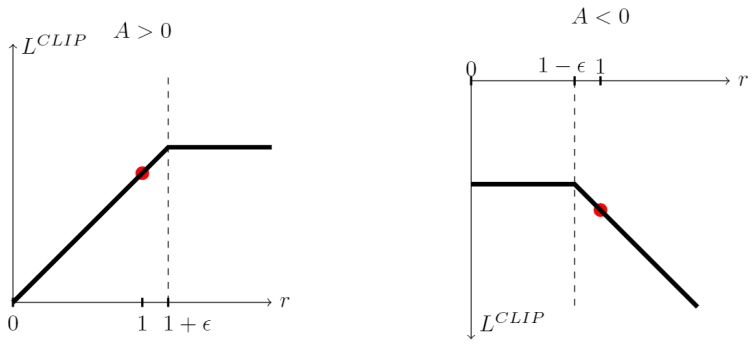

Figure 2.2: One step L^{CLIP} as a function of $r_t(\theta)$: *this is the function L^{CLIP} as a function of the probability ratio $r_t(\theta)$ when the advantage A is positive or negative. The red dot is the starting point of the optimization. (Figure from [3])*

The surrogate function can be further augmented by adding an entropy bonus to ensure sufficient exploration. By adding these terms, we obtain the following objective function:

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{2.2.5}$$

where S in an entropy bonus, L_t^{VF} is a squared-error loss and c_1, c_2 are coefficients.

The pseudocode below better explains all the procedure done by the PPO in refining the parameters.

Algorithm 1 PPO with clipped objective

Input: inital policy θ_0, clipping threshold ϵ

for $k = 0, 1, 2...$ **do**

Collect a set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$

Estimate the adventages $\hat{A}_t^{\pi_k}$, using any advantage est. alg.

▷ we will use GAE[2]

Compute policy update

$$\theta_{k+1} = arg \max_\theta \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam [1]), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = E_t \left[\sum_{t=0}^{T} \left[min(r_t(\theta)\hat{A}_t^{\pi_k} \, , \, clip(r_t(\theta) \, , \, 1-\epsilon \, , \, 1+\epsilon)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

Chapter 3

Experiments

In this chapter I will present the two phases learning process utilized by the fully connected neural network (FCNet) to maximize the agents utility. Afterwards there will be a brief description of the 5 environments I am going to study (one per taxation analyzed) and my conjecture on how the agents will respond to these taxation.

3.1 Experiment Setup

After understanding the Foundation framework and the process used by RL to optimize a policy we can focus on the model used. In the paper used as a reference the authors used a combination of a convolutional neural network and a LSTM [5]. This could grant them good spatial information from the CNN and a memory buffer from the LSTM. These feature are sensate because there is the need of process a map and because the agents share the same network.

In the experiments, however, it was used a fully connected neural network that do not share the same feature as the one above. The complete stucture of the model is shown in Table 3.1.

This setup might encur in a loss of efficiency for plenty of reasons. First of all,

Model: "FCNet"			
Layer (type)	Output Shape	Param #	Connected to
observations (InputLayer)	[(None, 1260)]	0	
encoder (Encoder)	(None, 256)	389632	observations[0][0]
fc_out (Dense)	(None, 50)	12850	encoder[0][0]
value_out (Dense)	(None, 1)	257	encoder[0][0]
Total params: 402,739			
Trainable params: 402,739			
Non-trainable params: 0			

Table 3.1: Fully connected neural network utilized.

as said before, the feature of the LSTM/CNN are missing. Then due to lack of resources it was not possible to do a proper hyper parameter tuning, probably leading to inefficiencies. However, as we will see, the FCNet can provide us with meaningful results, in particular the agents will show emerging behaviors and specialization. These are feature that were also finded in the paper by Zhang and Trott, the difference is present in the efficiency of the agents that is lower for the model used here.

There are many parameters that have to be set at this stage (for a semi-complete list check the appendix appendice). A first part of these values are related to the training, while a second part to the environment. I would like to stress some of the latter parameters, to better undersand the simulation.

First the skill levels, these are set to be different for each agent, the skill is the amount of coins received when building an house. The set of 4 skills is fixed and these are taken from a pareto distribution. Namely the values are (11.3; 13.3; 16.5; 22.2). These value are always assinged to the same starting location in the map, thus the agent starting in the bottom right will always have the highest skill and so on. Other important parameters are the episode lenght H which is set to 1000 time steps, the resource respawn probability which is set to 1% per timestep and the inital coin endowment that is set to 0.

Once defined the environment is defined it is possible to start the training, this will happen in two phases. The first phase to introduce labour, and the second to introduce the taxation.

3.2 Phase 1 training

The first training phase is necessary in order to get the FCNet 3.1 accustomed to the world dynamics and avoid falling into unoptimal behavior once disincentives factors are introduced. These two factors, as already discussed, are labour costs and taxation. In this first phase the interest is in getting the agent accustomed to the labour costs, while the taxes are completely removed and will be investigated in the second phase of training.

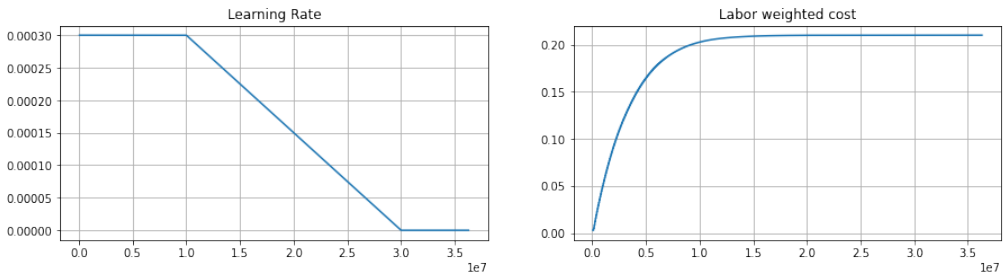

Figure 3.1: Learning rate and Labor weighed cost for Phase 1 training: *this was the schedule for respectively the learning rate and the labor cost in the phase 1 of the training, here the learning rate goes to 0 because we don't seek convergence now.*

The training agenda is built to run for a total of 30M time-steps, as we can see from Figure 3.1 the learning rate is set to be at 1e-4 for 10M steps then it linearly reduces to 0 in 20M time-steps. While the labour weight increases following the function:

$$\vartheta_k = 1 - exp\left(-\frac{episode\ completitions}{energy\ warm\text{-}up\ constant\ (k)}\right) \tag{3.2.1}$$

where the warm-up constant is set to 10000. This setup gives the FCNet time to learn how to respond properly to the dis-utility generated by the labor.

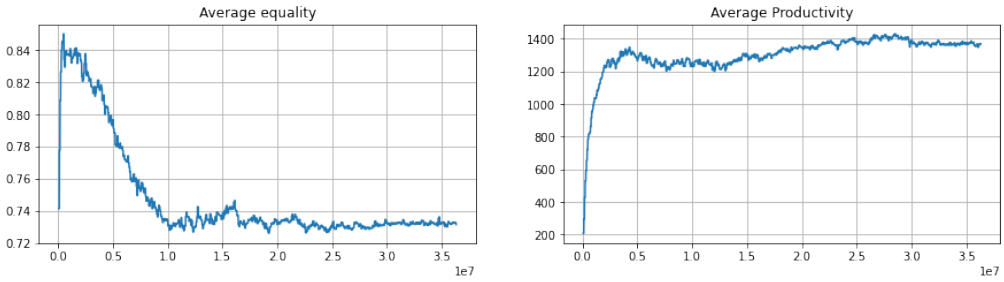

Figure 3.2: Equality and Productivity during Phase 1: *here we have the evolution of equality between agents and total production in the training of phase 1*

In Figure 3.2 we can see two of the most important variables that we are considering. On the left we have the average equality, this value is calculated by the following formula:

$$equality(\mathbf{x_t^c}) = 1 - \frac{N}{N-1} \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} |x_{i,t}^c - x_{j,t}^c|}{2N \sum_{i=1}^{N} x_{i,t}^c} \tag{3.2.2}$$

with $0 < equality(\mathbf{x_t^c}) < 1$, this function returns 1 if the endowments are equally split between the N (4) agents, and 0 if only one agent owns all of the coin in the economy. What we see is that after 36M time-steps the equality settled around 0.73, this is justified by the difference in coin endowment between the agent with the highest skill (Agent 0 in Figure 3.3) and the other agents. And we see that the average productivity of he economy settles around 1400.

The results obtained so far are not important for the final analysis since there were no criterion used to stop the training other than a time constraint. This means that the FCNet might have not reached convergence and an optimal policy. However we can keep these results in mind as a benchmark for the phase 2 trainig as we will use the parameters obtained to carry on the training.

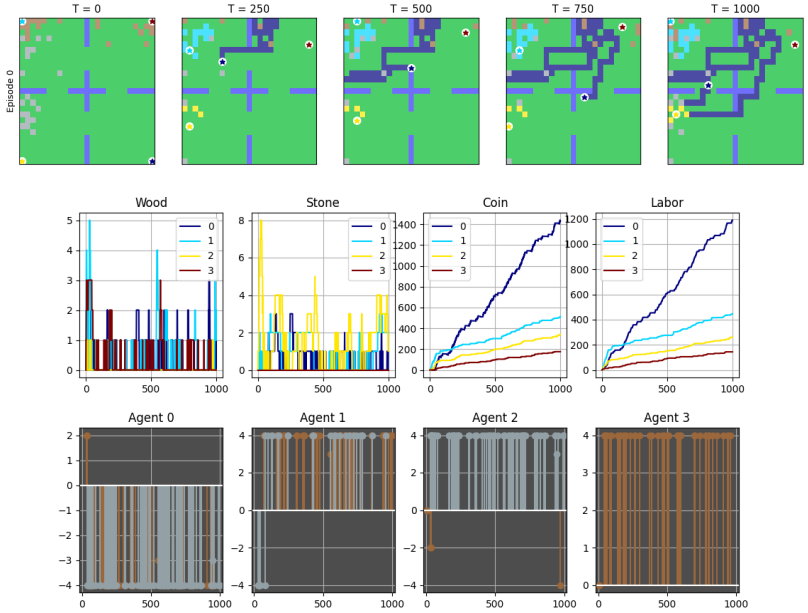

Figure 3.3: Phase 1 full brake down: *this is a more complete overview of one random simulation crated using the weights at the end of phase 1 training*

3.3 Phase 2 Trainig

In the second phase of the training the learning rate schedule is changed to be 3e-4 for the first 35M time-steps and then will linearly decrease to 1e-6 in a span of 15M time-steps. This will grant a faster learning rate in the inital part of the trainig when the FCNet will suffer a huge decrease in efficiency due to the introduction of the taxation.

The starting point for this second phase are the weights obtained from phase 1, and the training will be formed of five different simulations diverging from this inital state. These simulation are: US taxation, Italian Taxation, free market, Communism and Flat Tax.

3.3.1 Free Trade

Free trade is the first and simplest kind of simulation that we can carry on starting from the result in phase 1. This training consist in keep training the FCNet without imposing any kind of taxation.

It is reasonable to belive that the result of this first training should produce a more productive economy since the agent's utility function is positive in marginal coin endowment. However the equality between agents might suffer from a higher production.

3.3.2 US taxation

The second kind of taxation that will is used is the US taxation. This is based on the 2018 Federal brachets and percentages. In particular we can see from Table 3.2 the detailed stucture.

Bracket in $	0-9700	9700-39475	39475-84200	84200-160725	160725-204100	204100-510300	510300+
Bracket in Coin	0-9.7	9.7-39.5	39.5-84.2	84.2-160.7	160.7-204.1	204.1-510.3	510.3+
Tax*	10%	12%	22%	24%	32%	35%	37%

Table 3.2: 2018 US federal tax system: *this is the proportional system that was in place in the US in 2018, notice that the tax is marginal (i.e. if you are in the second bracket you will pay 970$ + 12% of the amount above 9700$)*

With the introduction of these lump sum tax plus redistribution is easy to conclude that there might me an increase in equality, however this migh produce a negative effect on total production since the high-skilled agent are disincentivised to produce.

3.3.3 Italian Taxation

As of the US's the Italian is a marginal system with brackets, however in this case the fiscal pressure is higer and the brachets smaller. As a reference it was used the 2020 INPS brachets and percentages (Table 3.3). However, the

brackets were calculated as if in dollars, so all the values were multiplied by 1.1575, the exchange rate at October 1st 2021.

Bracket in €	0-15000	15000-28000	28000-55000	55000-75000	75000+
Bracket in Coin	0-17	17-32	32-63	63-86	86+
Tax*	23%	27%	38%	41%	43%

Table 3.3: 2020 INPS tax system: *this is the proportional system that was in place in the Italy in 2020, notice that the tax is marginal (i.e. if you are in the second bracket you will pay 3450€+ 27% of the amount above 15000€)*

In this case there is an even higher pressure, wich might push toward higher equality and lower total production.

3.3.4 Communism

Communism is the opposite to Free market, in this simulation the fiscal pressure is 1, meaning that every 100 timestep all the agents are taxed for the entire amount of coins they own and then redistibuted. This system will probably deliver the loewst production and the highest equality.

3.3.5 Flat tax

3.4 Training Results

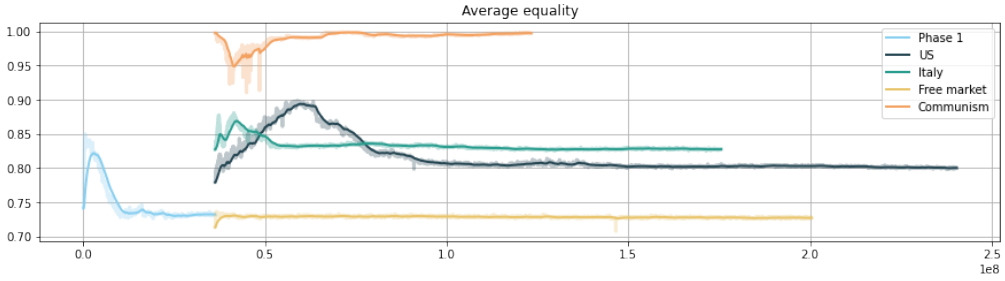

Figure 3.4: Equality after training:

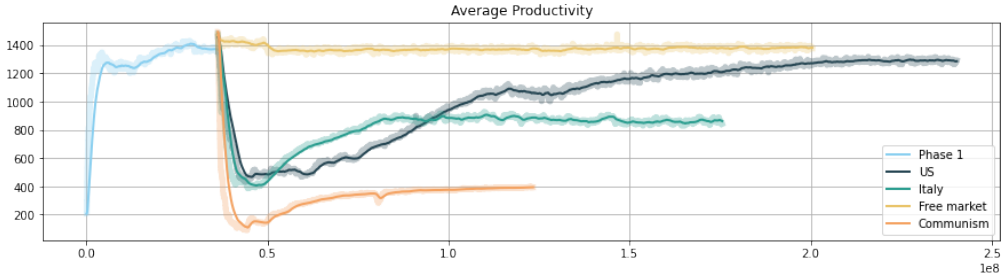

Figure 3.5: Productivity after training:

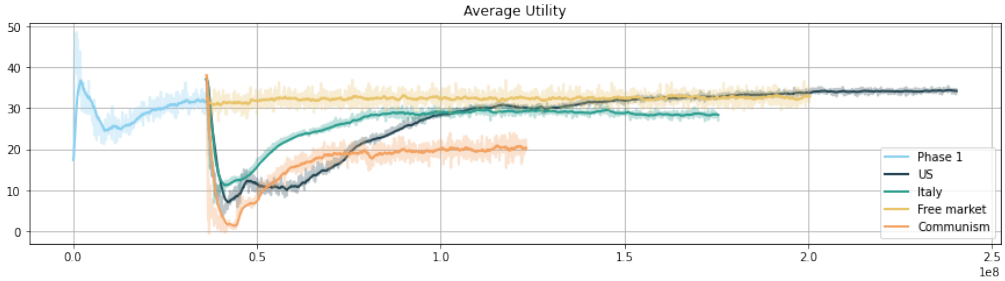

Figure 3.6: Utility after training:

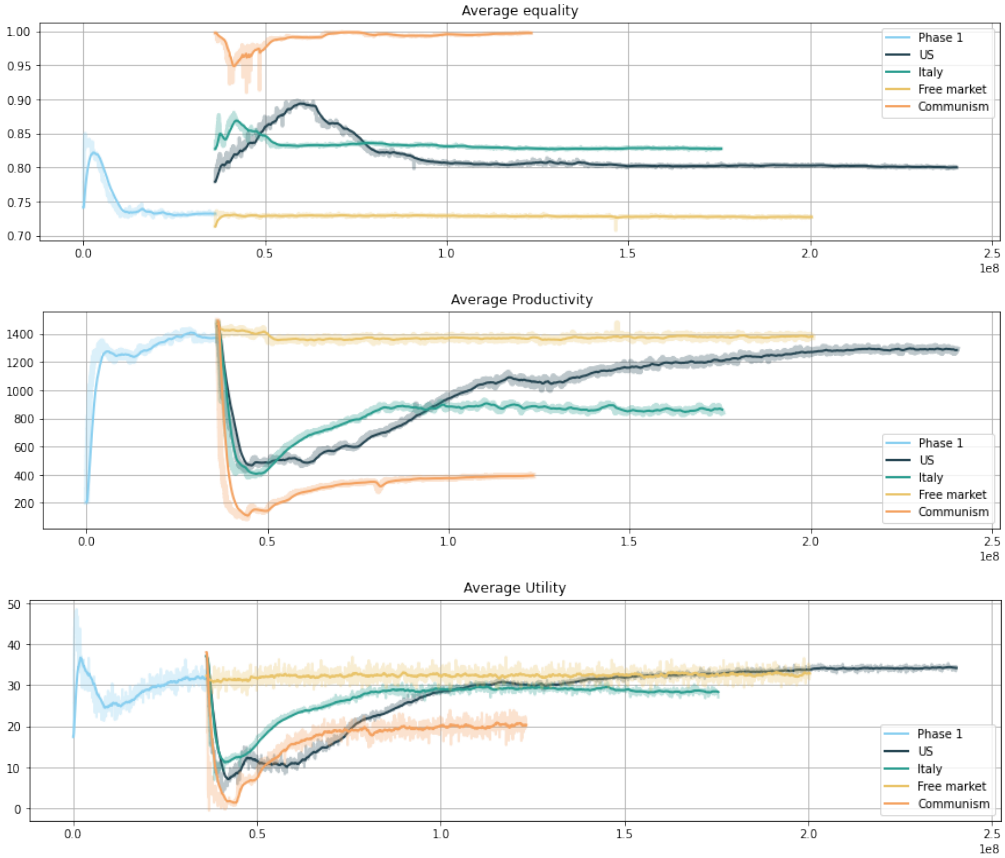

Figure 3.7: Phase 2 full brake down: *this is a more complete overview of one random simulation crated using the weights at the end of phase 1 training*

Appendix

Parameter		Value
Episode Lenght	H	1000
World height		25
World width		25
Resources respawn prob.		0.01
Inital Coin endowment		0
Iso-elastic utility exponent		0.23
Move labor		0.21
Gather labor		0.21
Trade labor		0.05
Build labor		2.1
Base build payout		10
Max skill multiplier		3
Max bid/ask price		10
Max bid/ask order duration		50
Max simultaneous open orders		5
Tax period duration		100
Min bracket rate		0%
Max bracket rate		100%

Table 4: Your caption.

Variable Name	Dimension	Bounds
world Map	(7, 11, 11)	{0;1}
world-idx_map	(2, 11, 11)	{0,1,...,5}
world-loc-row	(1,)	[0,1]
world-loc-col	(1,)	[0,1]
world-inventory-Coin	(1,)	[0,inf)
world-inventory-Stone	(1,)	
world-inventory-Wood	(1,)	
time	(1, 1)	
Build-build_payment	(1,)	
Build-build_skill	(1,)	
ContinuousDoubleAuction-market_rate-Stone	(1,)	
ContinuousDoubleAuction-price_history-Stone	(11,)	
ContinuousDoubleAuction-available_asks-Stone	(11,)	
ContinuousDoubleAuction-available_bids-Stone	(11,)	
ContinuousDoubleAuction-my_asks-Stone	(11,)	
ContinuousDoubleAuction-my_bids-Stone	(11,)	
ContinuousDoubleAuction-market_rate-Wood	(1,)	
ContinuousDoubleAuction-price_history-Wood	(11,)	
ContinuousDoubleAuction-available_asks-Wood	(11,)	
ContinuousDoubleAuction-available_bids-Wood	(11,)	
ContinuousDoubleAuction-my_asks-Wood	(11,)	
ContinuousDoubleAuction-my_bids-Wood	(11,)	
Gather-bonus_gather_prob	(1,)	
action_mask	(50,)	

Table 5: Full observation space.

Bibliography

[1] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[2] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[3] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

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

[5] Stephan Zheng et al. "The ai economist: Improving equality and productivity with ai-driven tax policies". In: *arXiv preprint arXiv:2004.13332* (2020).