Simulating the Effect of Universal Basic Income on Social Welfare in a Gather-and-Build-Game

Kolja Sielmann

Hamburg University of Applied Sciences, 20099 Hamburg, Germany
`kolja.sielmann@haw-hamburg.de`

Abstract. The effects of basic income, a certain number of coins paid to every individual without conditions, have received growing interest in research in recent years. Real-world studies are difficult to perfom due to their high costs. Simulating them in a multi-agent environment could therefore help to gain information on its influence on economic individuals and social welfare. We extend the AI-Economist's recently published foundation framework by a simple universal basic income model and train RL agents using Reinforce and PPO to act as an individual in the environment and a social planner to set taxes and the universal basic income to measure its influence on social welfare.

Keywords: economics, universal basic income, reinforcement learning

1 Introduction

In recent years, numerous applications of reinforcement learning in the field of economics and finance have been researched [4]. Above all, reinforcement learning has been applied to learn policies and bots for stock markets [1,17]. Recently, Zheng et al. have published a paper in which they proposed a framework that simulates an environment as a Gather-and-Build-game and used reinforcement learning methods to learn a tax policy that optimizes the environments social welfare [18].

Our main contribution is adding universal basic income to the environment and measuring its influence on the environment's social welfare. The effect of universal basic income on the social welfare has received growing interest in research in recent years [3,7,10,11]. Van Parijs describes the basic income as giving "all citizens a modest, yet unconditional income, and let them top it up at will with income from other sources" [10]. The introduction of universal basic income has a positive effect on the people's well-being, which has shown especially during the COVID-19 pandemic [2]. And, it could affect social welfare positively as well. In our implementation, the amount of coins paid as universal basic income is controlled by the social planner, which is trained along with the individuals, called mobile agents. Alternatively, the basic income can be fixed to a constant value. Then, it is interesting to see how this is influencing the planner's tax policy. For training, we implement both the Reinforce [16] algorithm and Proximal Policy Optimization [14].

This paper will be structured as follows: Section 2 contains a short overview on social welfare and tax policies. In section 3, the environment's details will be explained. Section 4 will address the Reinforcement Learning methods that we implemented for this work and how they are applied to learn tax policies. In section 5, we will explain some logic that we added to the simulation, including the universal basic income, followed by a description of our experiments and results follows in section 6 and the discussion in section 7.

2 Basics on Taxation

This section addresses some basics of economics and taxation. First, section 2.1 explains how an environment's social welfare can be measured. In section 2.2, we will give some information on the traditional taxation methods used in [18]. All of them are already implemented in the framework.

2.1 Social Welfare

To find optimal taxation schedules, it must be determined what exactly is to be optimized. The planner tries to maximize the social welfare. Zheng et al. use a social welfare function (SWF) that is a product of the environments' productivity and equality among the agent's wealth. To measure the equality $\text{EQ}(x^c) \in [0,1]$ they use

$$\text{EQ}(x^c) = 1 - \text{GINI}(x^c) \cdot \frac{N}{N-1}, \tag{1}$$

where $x^c = (x_1^c, x_N^c)$ is the vector of coin endowments for the number of agents N and the GINI index is a well-known measure of inequality with

$$\text{GINI}(x^c) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \left\| x_i^c - x_j^c \right\|}{2N \sum_{i=1}^{N} x_i^C}. \tag{2}$$

Zheng et al. define the productivity of an environment as the sum of the agents' coin endowments, which is given by:

$$\text{PROD}(x^C) = \sum_{i=1}^{N} x_i^c. \tag{3}$$

As stated before, they now use

$$\text{SWF}(x^c) = \text{EQ}(x^c) \cdot \text{PROD}(x^c) \tag{4}$$

to measure the social welfare of the environment [18].

2.2 Tax Policies

Following Zheng et al. [18], we use the Baseline Taxation methods to compare the RL agents' results with: US federal single-filer 2018 tax schedule and the Saez tax formula [12].

For the US federal tax schedule, there are fixed marginal tax rates

$$\tau = [0.1, 0.12, 0.22, 0.24, 0.32, 0.35, 0.37] \tag{5}$$

given [18].

The Saez formula is a framework for computing optimal income tax rates proposed by Emmanuel Saez in 2001 [12]. They compute the effect of a change in tax rates on the social welfare, called elasticity. The effect can be divided into a mechanical effect and a behavioral response. The mechanical effect just expressed the change of tax for a constant agent's pre-tax income for a change in the tax rate. The behavioral response expresses the change on the pre-tax income in dependence of a change in the tax rate. This elasticity is then used to compute the optimal tax rate. See [12] for details on computation.

3 Environment

We use the AI Economist's economic simulation framework called Foundation, introduced by Zheng et al. in 2020 [18]. The gym-like framework simulates a Gather- and Build Game with multiple mobile agents which can collect and trade material such as stone and wood, and build houses in a grid-world to earn coins. The simulation is running for a certain number of timesteps T. The 25×25 grid-world that is used is shown in fig. 1. The simulation also includes a planning agent which sets taxes for the individuals. For comparison of the agent's learned tax policy with other state-of-the-art methods, the framework also includes the possibility to use different tax policies. In this section, some of the environment's included dynamics will be explained. For more detail, see the original paper [18]. Later, in section 5 we will explain some extensions we made to the environment.

3.1 Dynamics

First, we will describe some of the environment's general dynamics.

Coin As mentioned before, coins are earned by trading resources (stone and wood) and building houses using resources. The framework is build in an expandable and customizable way, meaning that further resources and activities can be added and e.g. the number of resources needed to build a house can be adjusted to obtain a more realistic simulation. However, for our work we use the default values and components. Thus, an agent can build a house with one unit of both stone and wood.

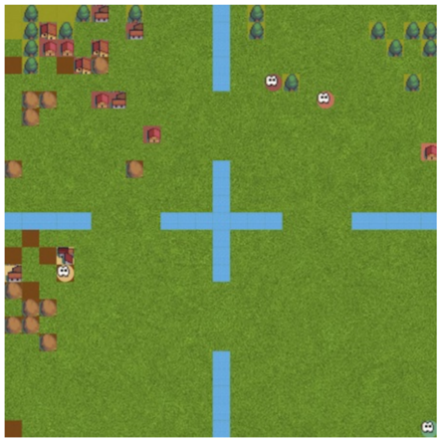


Fig. 1. Example of a Grid-World from [18], containing wood, stone, built houses and water.

Skill The coin earned by building a house depends on the agent's building skill, which is generally determining how much coin an agent is able to earn by one unit of labor. Every agent also has a collecting skill which determines how many resources it collects when deciding to collect either stone or wood.

Labor An agent's labor is defined as the sum of labor related to the agent's activities. For example, building a house, gathering material and trading is associated with a fixed effort or labor that is needed for this activity.

Income Taxation The simulation uses periodic income taxes with bracketed schedules: Bracketed schedules define B income brackets $[m_b, b_{b+1}]$, with $m_0 = 0$ and $m_B = \infty$. Let T_p with $T \mod T_p = 0$ be the tax period length. Then, tax period p ranges from timestep $p \cdot T_p$ to $(p+1) \cdot T_p$ for every $p \in \{0, \dots, \frac{T}{T_p}\}$. For tax period p, the planner will choose a marginal tax rate τ_b for every income bracket at timestep $p \cdot T_p$ and collect taxes in the following T_p timesteps which will be redistributed evenly between the mobile agents at the end of the episode. The tax payment for an agent's income z in one tax period is then defined by

$$\begin{aligned} T(z) &= \sum_{b=0}^{B} \tau_b \cdot ((m_{b+1} - m_b)\mathbb{1}(z > m_{b+1}) + (z - m_b)\mathbb{1}(m_b < z \leq m_{b+1}))) \\ &= \sum_{b=0}^{B} \tau_b * \max(\min(m_{b+1}, z) - m_b, 0), \end{aligned} \tag{6}$$

where

$$\mathbb{1}(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{else} \end{cases} \tag{7}$$

is an indicator function. The marginal tax rate τ_b determines the gradient of the total tax in the bracket as shown in fig. 2.

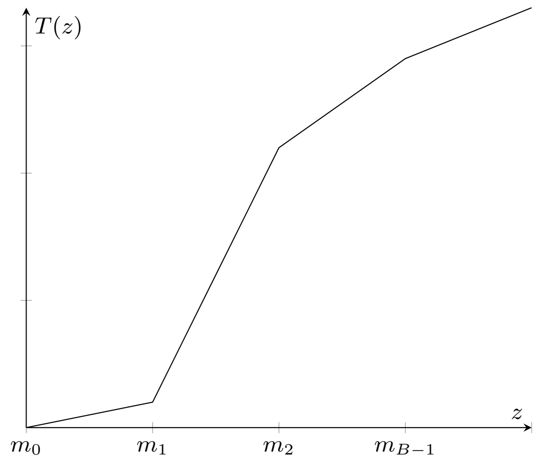


Fig. 2. Bracketed Schedules

Reward An agent will then try to maximize its coin outcome while minimizing the labor done to achieve that outcome. This is achieved by the way the environment computes the agent's reward: Labor has a negative impact on the reward while the coin earned has a positive impact. Mathematically the reward is expressed as the difference between the agent's utility (see eq. (8)) between timesteps. The agent's utility at timestep t is defined as

$$u_i(x^c_{i,t}, l_{i,t}) = \text{CRRA}(x^c_{i,t}) - l_{i,t},, \tag{8}$$

where $l_{i,t}$ denotes the total labor of the i-th agent until timestep t and

$$\text{CRRA}(z) = \frac{z^{1-\eta} - 1}{1 - \eta}. \tag{9}$$

Constant Relative Risk Aversion (CRRA) is a model for the diminishing marginal utility of money [5,18]. η is a hyperparameter that determines the extent of diminution. Figure 3 shows the CRRA function for different values of η. A high η means that an agent which has already earned a high number of coins benefits less from earning another coin compared to a small η.

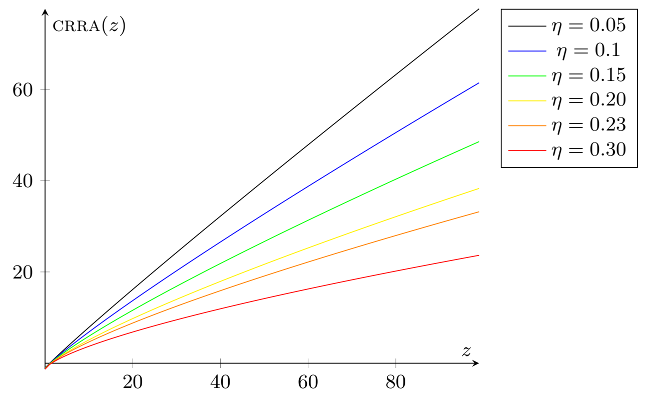


Fig. 3. CRRA for different η-values.

Zheng et al. use $\eta = 0.23$, which we follow. Equation (8) then defines the utility as a difference of the earned coins' utility and the labor done.

The reward is then computed as follows:

$$u_i(x^c_{i,t}, l_{i,t}) - u_i(x^c_{i,t-1}, l_{i,t-1}). \tag{10}$$

The planner's reward is computed as the change in social welfare at a certain timestep. The social welfare function used has already been described in section 2.1. Now, in order to compute rewards at the current timestep t the difference

$$\text{SWF}_t - \text{SWF}_{t-1} \tag{11}$$

is used, where SWF_t denotes the social welfare at timestep t, computed with eq. (4) and coin endowments x^c_t at timestep t.

3.2 Observations

The observations of the environment are returned as a dictionary in every step, containing one entry for each agent. The entries contain a map in format $H \times W \times C$. C is the number of objects (e.g. stone) which can be located in a certain point of the map. The planner receives the full map, such that $H = W = 25$, while the mobile agents only receive a local view of the map with $H = W = 11$. Additionally, the observations contain some of the agent's and the environment's information such as its skill and the default for coins earned by building a house, which is then combined with the agent's skill to compute the coin received by the specific agent. The time is also part of the observation, as a fraction of the whole episode's length, which is fixed. Each agent also receives an action mask within its observation, with information on which action is allowed in the current state. The agents also receive information on the current tax rates. The planning agent also gets information on each mobile agent's income.

3.3 Actions

The planner can select one out of 22 rates ($\{0, 0.05, 0.1, ..., 1.0\}$) for each of seven tax brackets. Actions are taken only every 10th timesteps, otherwise all actions but the NO-OP action are masked via the action mask mentioned above.

The mobile agents can decide to move, gather, trade or build a house. The action mask is a result of conditions registered for the actions, e.g. owning one unit of both stone and wood to build a house.

4 Reinforcement Learning

This section will address the Reinforcement Learning methods we use theoretically in addition to some basics of Reinforcement Learning. We use two different training methods: PROXIMAL POLICY OPTIMIZATION (PPO) [14] and REINFORCE [16].

The general problem from section 3 can be described as a multi-agent system. In this paper we use an environment with six mobile agents and one planner. Following [18], all six mobile agents use the same underlying model to learn a policy for efficiency. However, all agents receive a different observation containing its skill, inventory, masks etc. such that they can still choose different actions. The planning agent uses a separate underlying model. Details on the models' structure are summarized in fig. 11 for the mobile agents' model.

For multi-agent systems, the interaction can be separated into competitive and collaborative elements. In competitive environments, the agents will compete with each other trying to maximize their own rewards. In collaborative environments, the agents need to work together to reach a target and maximize their reward [9].

For the economic simulation described in section 3, both elements can be found. On the one hand, agents compete for resources which can be gathered and then used to build houses etc. On the other hand, they can collaborate by trading their resources and maximizing the rewards of both, e.g. by enabling both to build a house by that trade.

The same applies to the interaction between the planning agent and the mobile agents: The mobile agents are dependent on the planning agent to set a tax that is not too high, such that they receive enough income for their labor. However, mobile agents will try to maximize their own reward, which could lead to less equality and less reward for the planning agent as a consequence.

We will now start to mathematically define the multi-agent system. First, we define the set of mobile agents

$$\mathscr{M} = \{m_i | i \in \{1, \dots, n_{agents}\}\}, \tag{12}$$

where n_{agents} is the number of mobile agents. The set of agents is then defined by

$$\mathscr{A} = \mathscr{M} \cup \{p\}, \tag{13}$$

with planning agent p. Zheng et al. follow Sutton & and Barto [15] by defining the "PARTIAL-OBSERVABLE MULTI-AGENT MARKOV GAME" [18] as a Tuple

$$(S, A, r, \mathscr{A}, \gamma, o, \mathcal{I}), \tag{14}$$

where S is the set of states, with s^t the state at timestep t. A defines the Action spaces for every agent. Every agent $i \in \mathscr{A}$ receives a partial observation o_i^t of the world at timestep t, as described earlier in section 3.2. At every timestep t, every agent i also receives a reward r_i^t and transitions to the next step by using the *env.step* function which Zheng et al. denote mathematically as transition distribution $\mathcal{I}(s_{t+1}|s_t, a_t)$. For each agent $i \in \mathscr{A}$, a policy

$$\pi_i(a_i^t | o_i^t, h_i^t; \theta_i) \tag{15}$$

needs to be learned to maximize the agent's discounted rewards

$$d_i^t = \sum_{j=0}^{T-t} \gamma^j \cdot r_i^{t+j}, \tag{16}$$

where $\gamma \in (0, 1)$ is the discount factor, T is the overall length of an episode and θ_i denotes the model's weights. h_i^t is a hidden state which is implemented using a Long Short-Term Memory Network [6] in both Zheng et al.'s and this work.

Zheng et al. describe the optimization problem using inner-outer-loop RL, where the mobile agents learn in the inner loop within the tax periods and the social planner learns to adapt its tax policy at the start of every tax period. However, the training structure is not a nested loop, but the social planner and agents learn simultaneously with the social planner using the agents' actions within the tax period for its training [18].

For clarification, we will now define this training algorithm in a very general way without using a concrete learning algorithm, so we basically rewrite the inner-outer-loop algorithm from [18], that is sampling from multiple environments simultaneously. [1] [2]

4.1 Reinforce

REINFORCE [16] is a policy gradient method first proposed by Williams et al. REINFORCE is a Monte-Carlo method, so the full episode has to be sampled before training using the trajectories. The algorithm is shown in the following algorithm 2. Note that we write the algorithm in a way that allows integration in the general training algorithm 1 above: More specifically, we assume that the episode's data is already generated as $D_i \forall i \in \mathscr{A}$, such that only the training itself is mentioned.[3]

[1] Note that for some algorithms, e.g. REINFORCE, $\mathfrak{h} = T$ is required.

[2] The states used as model input are actually a series of timesteps of state, which are used by the LSTM. Collection the timesteps is done by a memory class in our implementation, not by the environment. For simplicity, here it is assumed that the *e.step*() function already returns a series of timesteps.

[3] θ_i is the same as θ_j for all $i, j \in \mathscr{M}$

Algorithm 1 Inner-Outer-Loop RL-algorithm based on [18].

```
    Input: number of episodes $n_{episodes}$
    Input: episode length $T$, tax period length $T_p$, Sampling horizon $\mathfrak{h}$,
    Input: number of environments $n_{envs}$
    Input: set of agents $\mathscr{A} = \mathscr{M} \cup \{p\}$
    Output: trained policies $\pi_i$ for every agent $i \in \mathscr{A}$
 1: $\mathscr{E}$ = list of $n_{envs}$ environments with episode length $T$ and tax period length $T_p$
 2: for episode $\in \{1, \ldots, n_{episodes}\}$ do
 3:     s = [$e.reset()$ for $e \in \mathscr{E}$]
 4:     initialize lists $D_i, i \in \mathscr{A}$        ▷ transitions for mobile agents $\mathscr{M}$ and planner $p$
 5:     for $t \in \{1, ..., T\}$ do
 6:         Sample action $a_{i,e}$ for $e \in \mathscr{E}$ and $i \in \mathscr{A}$ using $\pi_i$
 7:                                         ▷ $a^e_i$ == NO-OP unless $t \mod T_p == 1$
 8:         for $e \in \mathscr{E}$ do
 9:             for $i \in \mathscr{A}$ do
10:                 $o^t_{e,i}, r^t_{e,i} = e.step(s, a^t_{e,i})$
11:         for $i \in \mathscr{A}$ do
12:             $D_i = D_i \cup \{(a^t_{e,i}, o^t_{e,i}, r^t_{e,i}) | e \in \mathscr{E}\}$
13:         $s = s'$
14:         if $t \mod \mathfrak{h} == 0$ then
15:             for $i \in \mathscr{A}$ do
16:                 train policy $\pi_i$ on $D_i$ using a training algorithm
17:                                                         ▷ $\pi_i = \pi_j \forall i, j \in \mathscr{M}$
18:             reset set $D_i$ for every $i \in \mathscr{A}$
19: return policies $\pi_i$ with $i \in \mathscr{A}$
```

Algorithm 2 Multi Agent REINFORCE Algorithm, adapted from [15].

```
    Input: episode trajectories $D_i$ for every agent $i \in \mathscr{A}$
    Input: episode length $T$
    Input: learning rate $\alpha > 0$
    Input: policy parameters $\theta_i$ for every agent
    Output: trained policies $\pi_i$ for every agent $i \in \mathscr{A}$

 1: for $i \in \mathscr{A}$ do
 2:     for $e \in \mathscr{E}$ do
 3:         for $t \in \{1, ..., T\}$ do
 4:             sample $(a^t_{e,i}, o^t_{e,i}, r^t_{e,i})$ from $D_i$
 5:             Compute the discounted rewards $d^t_i$ using eq. (16).
 6:             Apply the baseline:
                    $$d^t_i = \frac{d^t_i - \bar{d}^t_i}{\sqrt{\mathrm{VAR}(d^t_i)}}. \quad (17)$$
 7:             Update the policy parameters by
                    $$\theta_i + \alpha \cdot d^t_i \cdot \nabla \ln \pi_i(a^t_{e,i} | o^t_{e,i}, h^t_{e,i}; \theta_i). \quad (18)$$
```

In practice, instead of eq. (18) we compute

$$y_{e,i}^t = (1 - \alpha \cdot d_i^t) \cdot \pi_i(a_{e,i}^t | o_{e,i}^t, h_{e,i}^t; \theta_i) + \alpha \cdot d_i^t \cdot a_{e,i}^t, \tag{19}$$

which is equivalent[4]. The sampled observation $o_{e,i}^t$ and $y_{e,i}^t$ are the used to train the policy using the Adam Optimizer [8].

Reinforce is an algorithm that is easy to implement but it also has some issues. First, Reinforce is a Monte Carlo Method meaning that a complete episode has to be sampled before the policy can be trained, as can be seen in algorithm 2. This results in a low bias because the value function is known as the whole episode is simulated. However, the variance is high since real instead of estimated values are used. This can lead to slow training and a high number of episodes needed. Increasing α could lead to faster, but unstable training [15]. Another issue of Reinforce is that often the policy will be trained to a local optimum and stop exploring. Using an entropy bonus can solve this problem [9]. The entropy bonus will be explained and applied in the following section 4.2.

4.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) was introduced by Schulman et al. in 2017 [14]. They propose different loss function to train the actor network. One of these uses a KL divergence which is measuring the distance between distributions, in this case the old and the new policy. However, we use a clipping function that is combining an actor and a critic and is defined by

$$L_t^{CLIP+VF+S}(\theta^a, \theta^c) = \hat{\mathbb{E}}[L_t^{CLIP}(\theta^a) - c_1 \cdot L_t^{VF}(\theta^c) + c_2 S[\pi_{\theta^a}(s_t)]], \tag{20}$$

where θ^a, θ^c are the actor's and the critic's losses.[5] The critic learns a state value function, generalized advantage estimation [13]. Generalized advantage estimation uses an estimator of the advantage of taking action a^t in state s^t to reduce the variance of the policy gradients.[6] It is defined by

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{21}$$

V denotes the value function that is in practice estimated by the critic. The generalized advantage estimator (GAE) the action at timestep t is then given by

$$\hat{A}^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{T_{\max}-t} (\gamma\lambda)^l \cdot \delta_{t+l}^V, \tag{22}$$

where $T_{\max}$ denotes the maximum timestep sampled. We abbreviate $\hat{A}^{\mathrm{GAE}(\gamma,\lambda)}$ by $\hat{A}$ in the following. λ has a similar meaning as in the TD(λ) algorithm [13].

[4] which has been shown in the lecture

[5] In our implementation, actor and critic share parameters.

[6] We exclude the indices e, i for the current environment and agent for simplicity.

The critic's loss L_t^{VF} is then the squared error between the critic's output and the generalized advantage estimator. S denotes an entropy bonus computed as

$$\sum_a \pi(a|o^t, h^t; \theta^c) \cdot \ln \pi(a|o^t, h^t; \theta^c). \tag{23}$$

The entropy bonus is high when the distribution is close to uniform such that all actions have a similar probability of being taken and low when one action has a probability close to 1. Maximizing the entropy or minimizing the negative entropy can therefore make sure that all actions keep a realistic chance of being selected, such that the policy still explores the environment [9].

The clipping loss is defined by

$$L^{\text{CLIP}} = \mathbb{E}[\min(R_t(\theta^a)\hat{A}, \text{clip}(R_t(\theta^a), 1-\epsilon, 1+\epsilon)\hat{A})]. \tag{24}$$

R_t denotes the ratio between the old an the updated policy,

$$R_t(\theta^c) = \frac{\pi(a^t|o^t, h^t; \theta^c)}{\pi_{\text{old}}(a^t|o^t, h^t; \theta^c)}. \tag{25}$$

The second term inside the min of eq. (24) clips the ratio into an ϵ-range such that the updates to the model weights do not exceed ϵ and the loss is not getting too big, and therefore prevents unstable training [14]. The full algorithm is then given by algorithm 3.

Algorithm 3 Multi Agent PPO Algorithm.

```
    Input: episode trajectories $D_i$ for every agent $i \in \mathscr{A}$
    Input: Sampling horizon start and end $\mathfrak{h}_{\text{start}}, \mathfrak{h}_{\text{end}}$    Input: $1 \geq \gamma, \lambda > 0$
    Input: learning rate $\alpha$
    Input: actor and critic parameters $\theta_i^a, \theta_i^c$ for every agent
    Output: trained policies $\pi_i$ for every agent $i \in \mathscr{A}$

1: for $i \in \mathscr{A}$ do
2:     for $e \in \mathscr{E}$ do
3:         for $t \in \{1, ..., T\}$ do
4:             sample $(a_{e,i}^t, o_{e,i}^t, r_{e,i}^t)$ from $D_i$
5:             Compute the generalized advantage estimation $\hat{A}$ using eq. (22)
6:             Compute the critic's mse $L_t^{VF}$
7:             Compute the actor's loss $L_t^{CLIP+VF+S}$ using eq. (20)
8:             Update the actor's parameters by descending the gradient
                   $\theta_i^a = \theta_i^a - \nabla_{\theta_i^a} L_t^{CLIP+VF+S}$    (26)
9:             Update the critic's parameters by descending the gradient
                   $\theta_i^c = \theta_i^c - \nabla_{\theta_i^c} L_t^{VF}$    (27)
```

5 Extensions to the Environment

While training the policies we sometimes observed a lack of activity by the mobile agents when using the original environment. As stated earlier, the mobile agents try to minimize labor and maximize the received coins. However, no activity leads to zero labor, zero coins gained and thus, to zero rewards. In practice, there always are costs like salaries or costs for location that are not represented in the environment. Therefore, the agents sometimes learned to do nothing to not get negative rewards. To overcome this problem and analyzing the influence of some components, we extend the environment by adding and adapting components which are encouraging the agent to act.

5.1 Building Houses with Experience

We adopt the original component for building houses by adding experience. For every agent, the component saves an experience value representing the number of houses built by the agent already. The labor needed to build a house is then reduced by a discount factor γ_H for every house already built. Let l_H be the labor needed to build the first house. Then, building the k-th house consumes a labor of

$$l_H \cdot (\gamma_H)^k. \tag{28}$$

Especially in shortened periods compared to the original paper by Zheng et al. we often have seen little activity in terms of building houses. This component tries to solve this problem while reduced work for repeated tasks is also a realistic assumption. In our experiments, we used $\gamma_H = 0.8$.

5.2 Universal Basic Income

The main component for our analysis is the universal basic income which is integrated into the taxation and redistribution component. At the end of every tax period the taxes are enacted and redistributed. However, we now use a *universalbasicincome* that is always distributed to every agent. In general, the component uses the collected taxes to pay the universal basic income. The coin that is left is then also redistributed to every agent. When the collected taxes are not sufficient to pay the universal basic income, the planner will take a debt such that its coin endowment x_p^c is reduced and might be negative. As soon as the collected taxes exceed the coins needed to pay the universal basic income the debt will be repaid by the surplus. The universal basic income is set by the planner for every tax period. The agents therefore will receive an income independent of the environment's productivity. In our experiments, we will analyze the universal basic income's influence on the social welfare. Remember that eq. (3) computed the environment's productivity of the sum of the agent's coin endowments. We adapt the equation such that

$$\text{PROD}(x^c) = \sum_{i=1}^{n_{agents}} x_i^c + x_p^c \tag{29}$$

to include the planner's endowment or debt in the productivity.

5.3 License

In real environments, universal basic income is mainly designed to cover basic expenditure for food and rent. To model such regular expenses we add a license component to the environment. The license (e.g. for gathering resources in the environment) is automatically paid to the planner at the start of each tax period. The amount will be set by the planner. The collected coins are then redistributed along with the universal basic income and the tax redistribution. However, since this license coins are either set by the social planner or fixed for all agents, it can not accurately model the basic expenditure named earlier. In possible future work, further extensions should be made to the environment for that reason.

5.4 Debts

In the original framework, negative coin endowments are forbidden. We add the possibility of negative coins for both mobile agents and the planner. Currently, there are no interest rates for negative coins. To be able to handle negative endowments, we must rewrite the utility function eq. (8) such that

$$u_i^-(x_{i,t}^c, l_{i,t}) = \begin{cases} u_i(x_{i,t}^c, l_{i,t}) & \text{if } x_{i,t}^c > 0 \\ x_{i,t}^c - l_{i,t} \end{cases}, \tag{30}$$

since the CRRA function cannot be used for negative coin endowments.

6 Experiments and Results

The details on our experiments and our results will be described in section 6.3. First, we will give some information about the environment parameters used and the model's training in sections 6.1 and 6.2, respectively.

6.1 Environment Parameters

Some parameters of the environment can be set manually. Figure 4 shows an overview on the parameters we use in this work. The main difference compared to Zheng et al.'s paper is the number of timesteps per episode, which we have to decrease for computational efficiency. The resource regeneration probability determines the probability of a resource respawning after being gathered by a mobile agent. Since we use shorter episodes, we increase the probability of respawning in one timestep.

Variable		Original Paper [18]	This Paper
Number of agents	n_{agents}	4	3
episode length (timesteps)	T	1000	100
tax period (timesteps)	T_P	100	10
resource regeneration probability		0.01	0.1
Move Labor		0.21	0.021
Gather Labor		0.21	0.21
Trade Labor		0.05	0.025
Build Labor	l_H	2.1	2.1
Default build payout		10	10
Build payment max skill multiplier		3	10
Build experience discount	γ_H		0.8
Max bid/ask price		10	5
Max bid/ask order duration		50	5
Max universal basic income			3.
Max license coin			3.

Fig. 4. Comparison of environment parameters in [18] and this work.

6.2 Model Training

For both the planner and the actor, we train models that are using Long Short-Term Memory [6] Layers. The map is processed by a Convolutional LSTM followed by a Global Average Pooling Layer. The flattened observations, including the tax rates etc., also passed through a LSTM layer. Then the observations are concatenated and processed by some Dense layers. To put out the distribution of actions to be taken, the Dense layers' output is then multiplied by the action mask.

For PPO, parameters between actor and critic are shared. For the critic's output, different Dense layers are used at the model's end. Figure 11 shows a summary of a mobile agent's model with shared parameters between actor and critic. [7]

Figure 6 shows that PPO trains much faster than Reinforce as expected. We therefore use the PPO models in the following experiments. For both algorithms, the training could be further optimized by optimizing hyperparameters and training for more episodes. However, our trained models are sufficient for our purpose of analyzing the universal basic income's effect, for which we do not need to train an optimal model.

6.3 Experiments

We now use the PPO policies to run episodes and measure the influence of the universal basic income. Figure 7 shows the grid-map at different timesteps. The

[7] For the planner, the model's structure is more complicated since it can take multiple actions at one timestep (setting a tax rate for every bracket etc.).

Algorithm	Variable		Value
REINFORCE	Number of episodes	$n_{episodes}$	200
	Sampling horizon	$\mathfrak{h}$	T
	Number of environments	n_{envs}	40
	Reward discount factor	γ	0.99
	Step size	α	0.01
	Planner learning rate		0.01
	Agent learning rate		0.01
PPO	Number of episodes	$n_{episodes}$	120
	Sampling horizon	$\mathfrak{h}$	20
	Number of environments	n_{envs}	30
	Reward discount factor	γ	0.95
	Value function loss weight	c_1	0.2
	Entropy loss weight	c_2	0.02
	GAE lambda	λ	0.99
	Planner learning rate		0.001
	Agent learning rate		0.001
Shared	Planner LSTM timesteps		10
	Agent LSTM timesteps		5

Fig. 5. Training parameters

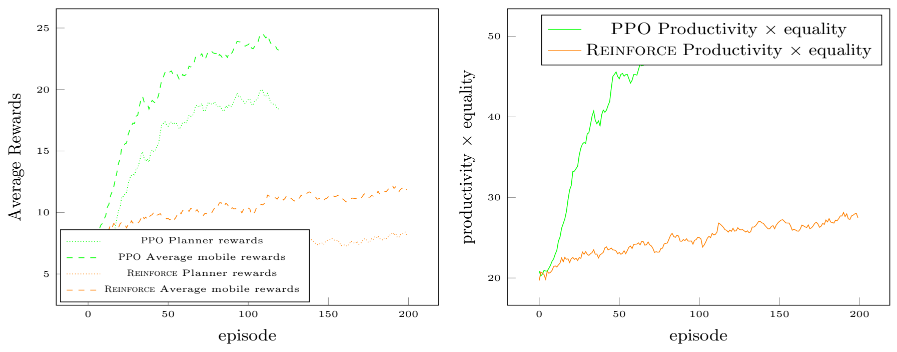


Fig. 6. Comparison of PPO and REINFORCE training in terms of rewards and the social welfare of the environment. For both 10-episode averages were used for plotting.

beige and olive-green visualize wood and stone, respectively. The circled stars correspond to the agents. The squares in the agents' colors represent their built houses. In general, little activity can be seen since the environment is built for longer episodes.

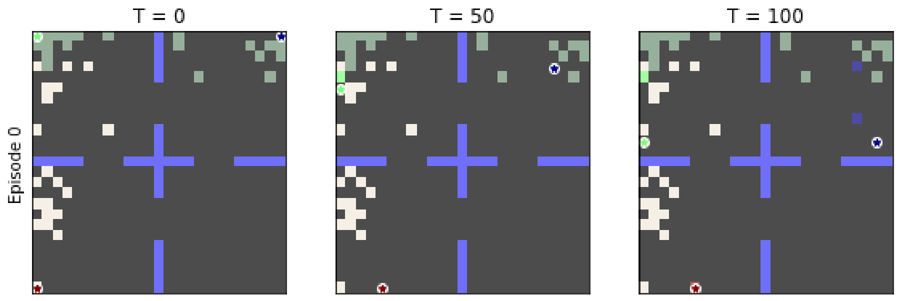


Fig. 7. Map resulting of an episode using the PPO policies.

Then, we compared the trained PPO policy to the tax policies described in section 2.2. For all experiments we used the average taxes, basic incomes etc. from multiple environments. The marginal tax rates set by our trained policy are very similar to the ones computed using the Saez framework. Lower incomes are taxed a little less though as can be seen in fig. 8.

Additionally, the social welfare achieved with the RL agents and Saez is almost the same. However, for Saez it seems like there was one high income in the last timestep and sampling more environments might be necessary for more accurate results. In terms of equality, the RL agents achieve better results.

Now, we will measure the influence that the universal basic income has on the social welfare. For that, we will run the simulation for 5 environments with different settings:

– The universal basic income and the license coins are controlled by the planning agent, abbreviated as (model_ubi_lc),
– the universal basic income is controlled by the planner, while the license coins is controlled by the planner, (model_ubi_x_lc),
– both universal basic income and license coins are fixed, (x_ubi_y_lc).

Figure 9 compares the average social welfare of 15 environments for different environment settings in terms of basic income and license coin. First, it shows that our extensions to the environment lead to a higher overall social welfare compared to the original environment, which is basically equivalent to 0_ubi_0_lc. However, since the agents are trained in environments with basic income and license coins enabled (and set by the social planner), we can not take reliable conclusions from that fact. Indeed, comparing the environments in which the

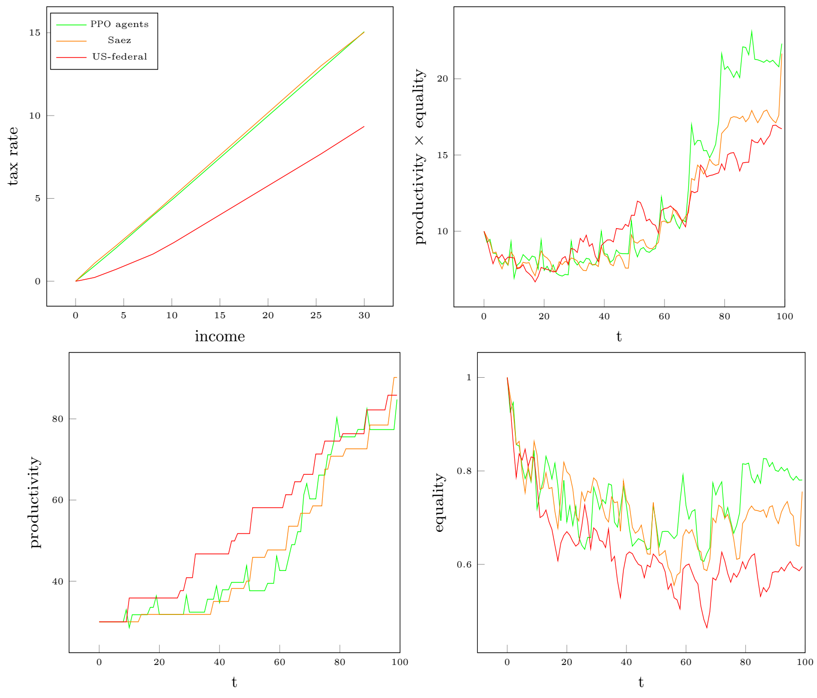


Fig. 8. Comparison of Tax frameworks with average values from 10 environments.

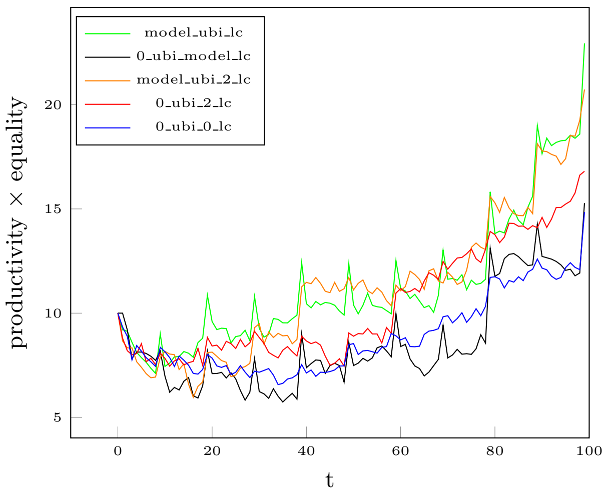


Fig. 9. Comparison of social welfare with and without universal basic income.

license coins were fixed to 2 shows that a basic income that is controlled by the social planner leads to higher social welfare indeed compared to disabled basic income. The same applies when the amount of license coins to be paid is controlled by the social planner. In fact, both equality and productivity are higher when the social planner controls the basic income compared to when basic income is disabled as shown in fig. 10.

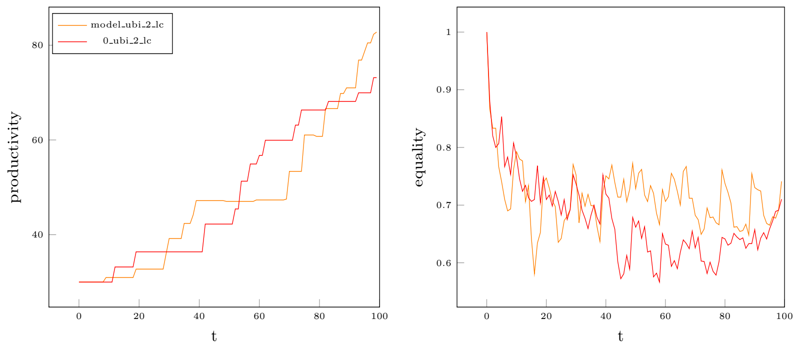


Fig. 10. Comparison of productivity and coin equality with and without universal basic income.

Despite the restrictions on the realism of the framework, which will be discussed in more detail in the following section, our experiments suggest that a basic income can increase an environment's social welfare.

7 Discussion

We have added a universal basic income component to the AI-Economist's foundation framework and implemented PPO and the Reinforce algorithm to train policies for economic individuals and a social planner. We observed that in the AI-Economist's economic simulation paying individuals a basic income can indeed increase equality and productivity and thus social welfare by enabling all individuals to participate by trading resources with each other and paying basic expenditures, modeled by the license coin.

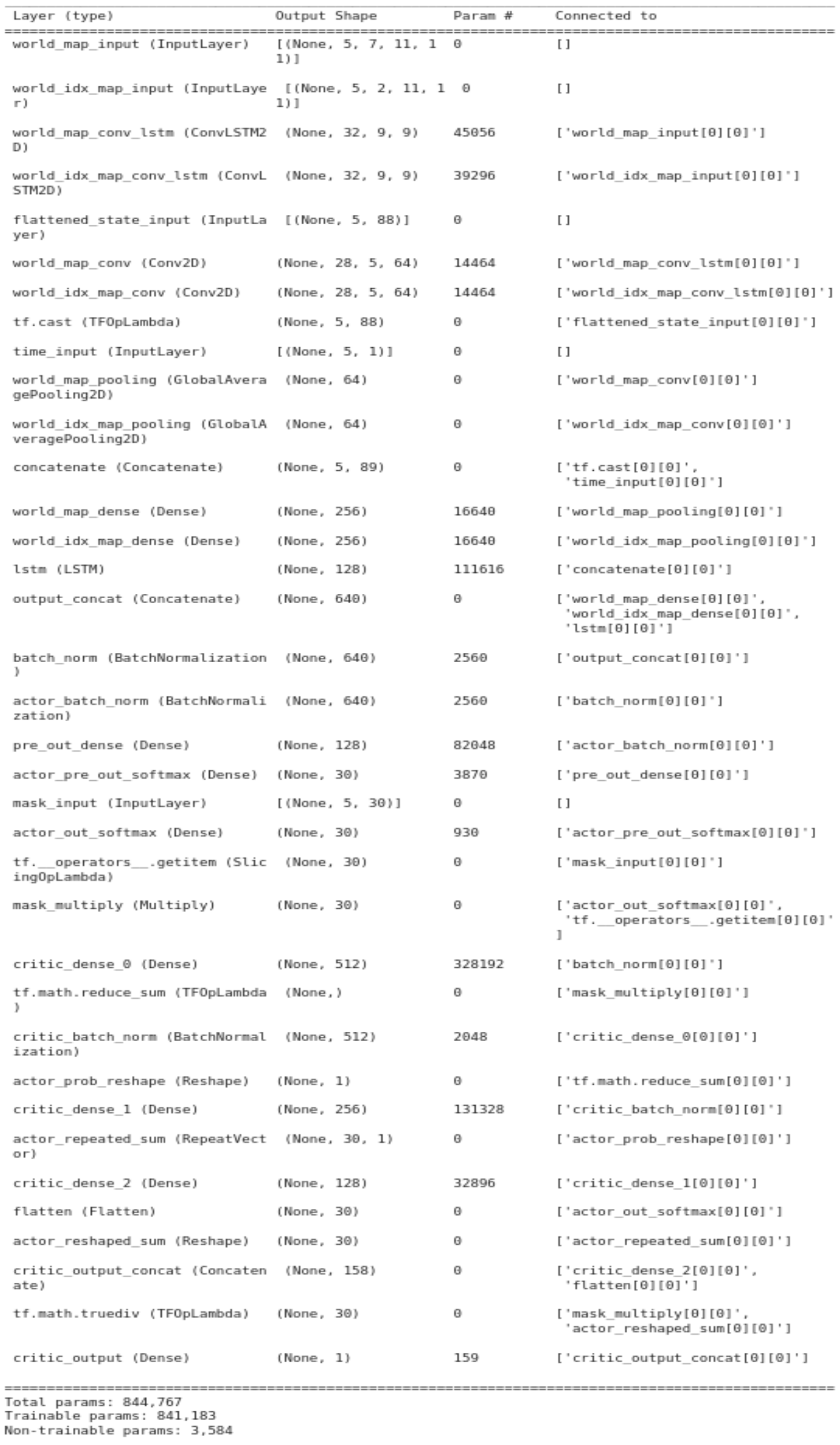
A basic income is designed to cover the individual's most necessary costs such as food or rent. As already mentioned, we tried to model this by introducing a license coin that has to be paid at every tax period's start. This is a very simple and inaccurate model since individuals could reduce their costs for rent etc. up to a certain point. To receive more meaningful results those necessary costs should be modeled in a more sophisticated way. That could for example include modeling individual's health, influenced by food costs or rent, which could then be chosen by the individual agents. The health could then influence the social planner's expenditures for medical treatment such that the planner benefits from individuals being healthy. To receive a more realistic environment, introducing more sources for an individual's income would be helpful in possible future work as well.

References

1. Azhikodan, A.R., Bhat, A.G.K., Jadhav, M.V.: Stock trading bot using deep reinforcement learning. In: Saini, H.S., Sayal, R., Govardhan, A., Buyya, R. (eds.) Innovations in Computer Science and Engineering. pp. 41–49. Springer Singapore, Singapore (2019)
2. Banerjee, A., Faye, M., Krueger, A., Niehaus, P., Suri, T.: Effects of a universal basic income during the pandemic. UC San Diego technical report (2020)
3. Banerjee, A., Niehaus, P., Suri, T.: Universal basic income in the developing world. Annual Review of Economics **11**(1), 959–983 (2019). https://doi.org/10.1146/annurev-economics-080218-030229, https://doi.org/10.1146/annurev-economics-080218-030229
4. Charpentier, A., Elie, R., Remlinger, C.: Reinforcement learning in economics and finance (2020)
5. Debreu, G.: Representation of a preference ordering by a numerical function. Decision processes **3**, 159–165 (1954)
6. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (Nov 1997). https://doi.org/10.1162/neco.1997.9.8.1735
7. Hoynes, H., Rothstein, J.: Universal basic income in the united states and advanced countries. Annual Review of Economics **11**(1), 929–958 (2019). https://doi.org/10.1146/annurev-economics-080218-030237, https://doi.org/10.1146/annurev-economics-080218-030237
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017)
9. Lapan, M.: Deep Reinforcement Learning Hands-On - Second Edition. Packt Publishing (Jan 2020)
10. Parijs, P.V.: Basic income: A simple and powerful idea for the twenty-first century. Politics & Society **32**(1), 7–39 (2004). https://doi.org/10.1177/0032329203261095, https://doi.org/10.1177/0032329203261095
11. Raventós, D.: Basic Income: The Material Conditions of Freedom. Pluto Press, London Ann Arbor, Mich (09 2007)
12. Saez, E.: Using Elasticities to Derive Optimal Income Tax Rates. The Review of Economic Studies **68**(1), 205–229 (01 2001). https://doi.org/10.1111/1467-937X.00166, https://doi.org/10.1111/1467-937X.00166
13. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation (2018)
14. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)
15. Sutton, R., Barto, A.G.: Reinforcement learning : An Introduction. MIT Press, Cambridge, Mass, 2 edn. (2018)
16. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning **8**(3), 229–256 (1992)
17. Xiong, Z., Liu, X., Zhong, S., Yang, H., Walid, A.: Practical deep reinforcement learning approach for stock trading. CoRR **abs/1811.07522** (2018), http://arxiv.org/abs/1811.07522

18. Zheng, S., Trott, A., Srinivasa, S., Naik, N., Gruesbeck, M., Parkes, D.C., Socher, R.: The ai economist: Improving equality and productivity with ai-driven tax policies (2020)

A Appendix



```
______________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==============================================================================================
 world_map_input (InputLayer)   [(None, 5, 7, 11, 1  0           []
                                1)]

 world_idx_map_input (InputLaye  [(None, 5, 2, 11, 1  0          []
 r)                             1)]

 world_map_conv_lstm (ConvLSTM2  (None, 32, 9, 9)    45056       ['world_map_input[0][0]']
 D)

 world_idx_map_conv_lstm (ConvL  (None, 32, 9, 9)    39296       ['world_idx_map_input[0][0]']
 STM2D)

 flattened_state_input (InputLa  [(None, 5, 88)]     0           []
 yer)

 world_map_conv (Conv2D)        (None, 28, 5, 64)    14464       ['world_map_conv_lstm[0][0]']

 world_idx_map_conv (Conv2D)    (None, 28, 5, 64)    14464       ['world_idx_map_conv_lstm[0][0]']

 tf.cast (TFOpLambda)           (None, 5, 88)        0           ['flattened_state_input[0][0]']

 time_input (InputLayer)        [(None, 5, 1)]       0           []

 world_map_pooling (GlobalAvera  (None, 64)          0           ['world_map_conv[0][0]']
 gePooling2D)

 world_idx_map_pooling (GlobalA  (None, 64)          0           ['world_idx_map_conv[0][0]']
 veragePooling2D)

 concatenate (Concatenate)      (None, 5, 89)        0           ['tf.cast[0][0]',
                                                                  'time_input[0][0]']

 world_map_dense (Dense)        (None, 256)          16640       ['world_map_pooling[0][0]']

 world_idx_map_dense (Dense)    (None, 256)          16640       ['world_idx_map_pooling[0][0]']

 lstm (LSTM)                    (None, 128)          111616      ['concatenate[0][0]']

 output_concat (Concatenate)    (None, 640)          0           ['world_map_dense[0][0]',
                                                                  'world_idx_map_dense[0][0]',
                                                                  'lstm[0][0]']

 batch_norm (BatchNormalization  (None, 640)         2560        ['output_concat[0][0]']
 )

 actor_batch_norm (BatchNormali  (None, 640)         2560        ['batch_norm[0][0]']
 zation)

 pre_out_dense (Dense)          (None, 128)          82048       ['actor_batch_norm[0][0]']

 actor_pre_out_softmax (Dense)  (None, 30)           3870        ['pre_out_dense[0][0]']

 mask_input (InputLayer)        [(None, 5, 30)]      0           []

 actor_out_softmax (Dense)      (None, 30)           930         ['actor_pre_out_softmax[0][0]']

 tf.__operators__.getitem (Slic  (None, 30)          0           ['mask_input[0][0]']
 ingOpLambda)

 mask_multiply (Multiply)       (None, 30)           0           ['actor_out_softmax[0][0]',
                                                                  'tf.__operators__.getitem[0][0]'
                                                                 ]

 critic_dense_0 (Dense)         (None, 512)          328192      ['batch_norm[0][0]']

 tf.math.reduce_sum (TFOpLambda  (None,)             0           ['mask_multiply[0][0]']
 )

 critic_batch_norm (BatchNormal  (None, 512)         2048        ['critic_dense_0[0][0]']
 ization)

 actor_prob_reshape (Reshape)   (None, 1)            0           ['tf.math.reduce_sum[0][0]']

 critic_dense_1 (Dense)         (None, 256)          131328      ['critic_batch_norm[0][0]']

 actor_repeated_sum (RepeatVect  (None, 30, 1)       0           ['actor_prob_reshape[0][0]']
 or)

 critic_dense_2 (Dense)         (None, 128)          32896       ['critic_dense_1[0][0]']

 flatten (Flatten)              (None, 30)           0           ['actor_out_softmax[0][0]']

 actor_reshaped_sum (Reshape)   (None, 30)           0           ['actor_repeated_sum[0][0]']

 critic_output_concat (Concaten  (None, 158)         0           ['critic_dense_2[0][0]',
 ate)                                                             'flatten[0][0]']

 tf.math.truediv (TFOpLambda)   (None, 30)           0           ['mask_multiply[0][0]',
                                                                  'actor_reshaped_sum[0][0]']

 critic_output (Dense)          (None, 1)            159         ['critic_output_concat[0][0]']

==============================================================================================
Total params: 844,767
Trainable params: 841,183
Non-trainable params: 3,584
```

Fig. 11. Summary of the mobile agent's model.