# Using Reinforcement Learning to Create Optimal Tax Policies

Presented to the faculty of Lycoming College in partial fulfillment of the requirements for Departmental Honors in Physics and Astronomy

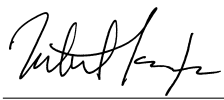
by

Amy Kushlan

Lycoming College

April 27 2022

Approved by:



# Using Reinforcement Learning to Create Optimal Tax Policies

Amy Kushlan

**Abstract**

The purpose of this research was to create a reinforcement learner that could generate optimal tax policies for a model economy. The economic model was loosely based on the United States tax system, using a 7-bracket progressive tax. The reinforcement learner relies solely on the values of various social welfare functions from the wealth distribution to make decisions and learn how to best maximize these functions.

## 1 Introduction

Though the term Machine Learning was first used in 1959 [8], the field is more important now than ever. Reinforcement learning is a particularly burgeoning field, with algorithms being created that can outperform a human benchmark at any Atari 2600 game [3] and beat the world champion at the game Go [9]. A team at deepsense.ai even trained a reinforcement algorithm to run like a human [6]. The versatility of reinforcement learning allows it to be applied to a variety of fields, including economics.

Machine learning has been used for numerous economic and tax studies, from measuring the effect of taxes [2], to identifying cases of tax evasion [10], to predicting inaccurate tax returns before they are filed [4]. None of these models, however, use reinforcement learning. Applying reinforcement learning to an economic model can be difficult because of differing definitions of economic "success," but it can be done. In this model, reinforcement learning has been used to attempt to find optimal tax policies in a simulated economy.

## 2 Reinforcement Learning

Reinforcement learning is a type of unsupervised machine learning in which a model is trained through rewards and punishments. The machine learning model is provided with a reward function with the goal of maximizing it. The only basis on which the model learns is the effect of its actions on the value of the reward function. The model learns by trial and error, making choices and observing the effect, until it learns the best way to solve the problem it is given and maximize its reward.

A reinforcement learning model learns from scratch - it is given no input other than the reward function and its current status. It essentially learns by trial and error. The model relies entirely upon the effect of its changing state on the reward function to make decisions. The model in this project uses an optimization method called gradient descent. The model examines the gradients of different possible decisions relative to the reward function to determine the best possible action. This allows it to travel to states of higher and higher reward with each training iteration.

# 3 The Reinforcement Learning Model

The model used for this learner is a single-layer neural network created using the Keras package from the Tensorflow python library. The first layer of the neural net has the same number of neurons as there are agents in the population and uses the ReLU activation function. The output layer has the same number of neurons as there are number of brackets in the tax system and uses the Sigmoid activation function. The Sigmoid function was chosen to limit the outputs to values between 0 and 1, which would create tax rates between 0% and 100% when used in the tax code.

## 3.1 The Reward Function

The reward functions used in this model are economic functions known as Social Welfare Functions (SFWs). SWFs are used to measure the desirability of a particular economic status using a variety of factors. The SWFs used here are primarily dependent on the wealth of individuals and the wealth distribution of the economy. The primary SWFs used in this model are the Sen and the Rawlsian, as well as the Gini Coefficient, as outlined below. The Gini Coefficient is a value representing the inequality present in an economy. It is a value between 0 and 1, with 0 representing total equality (all agents have equal wealth) and 1 representing complete inequality (one agent holds all of the wealth). The Gini coefficient in earlier versions of the model was calculated using the following code:

```
def gini(x):
    # requires all values in x to be zero or positive numbers
    n = len(x)
    s = x.sum()
    r = np.argsort(np.argsort(-x)) # calculates zero-based ranks
    return 1 - (2.0 * (r*x).sum() + s)/(n*s)
```

In later models, the gini_coefficient function from the QuantEcon python library was used. [1]

The Sen social welfare function is inversely proportional to the Gini coefficient. It is represented

by the following function:

$$r = w_{avg} * (1 - g) \tag{1}$$

where $w_{avg}$ is the average wealth of the population and $g$ is the Gini coefficient. In a state of perfect equality (i.e. $g = 0$), the Sen will equal the average wealth. This is the reward function used most often when running the model because equality is maximized when the Sen is maximized.

The Rawlsian SWF is simply the wealth of the poorest individual in an economy. When the Gini coefficient is 0, the Sen and Rawlsian should be equal.

### 3.2 The Environment

The environment is where the majority of the code is executed. This section of the model is where agents collect wealth, the model is trained and used to determine tax policies, and rewards are calculated and normalized.

The first portion of the environment code defines how agents receive income and accumulate wealth. Agents are assigned a Pareto-distributed skill score that determines their income. For every round in a set number of rounds, defined as $t\_max$, agents collect an income based on their skill score plus a normally distributed random value. This income is added to an agent's total wealth, which is then taxed.

The next section is where the model determines the wealth distribution is processed by the RL learner. There is a 75% chance of the learner's tax policy being used, and a 25% chance of a randomly determined tax policy being used. This randomness is key to allowing the RL learner to truly find the optimal policies. The learner uses a process known as gradient descent to choose its policies. It calculates the gradient of possible options, and chooses the options most likely to maximize its reward. However, the reward function may have more than one maximum. If the learner becomes focused on one maximum, it may miss another, even higher maximum somewhere else. The random element allows the learner the opportunity to discover these other maximums. If the random policy is worse, the learner can return to where it was previously.The following cell allows the code in the prior cell to run multiple times and records the rewards and gradients as values in a list.

The next portion is a key part of the reinforcement learning process: discounting rewards. The discount process allows the model to better understand what actions lead to worsened results. If the model ends in a less-than-ideal state, it is likely that the last action taken is not solely responsible. The discount factor $\gamma$ applies exponentially to every future action. The reward of the second action is multiplied by $\gamma$, the third is multiplied by $\gamma^2$, and so on. As the learner is trained, it learns to prioritize early decisions that will have longer-lasting positive impacts, because these earlier actions

will have the greatest reward. This model uses a gamma value of $\gamma = 0.95$.

The following section trains the learner using functions defined in all of the previous cells. The next cell uses the trained model to create tax policies for a new wealth distribution. The model then outputs the final wealth distribution and tax policies, the Gini coeffiecient, Sen SWF, and Rawlsian SWF.

# 4 Initial Economic Model and Tax Code

The first version of this code was used to run many trials with variation of different variables within the model. These trials provided insight into the inner workings of the model and revealed errors that would later need to be fixed.

## 4.1 The Accumulation Model

The primary economic model used in this research is an accumulation model. The economy consists of an assigned number of agents, $num\_agents$, who are each assigned a Pareto-distributed skill value. This skill determines an agent's income. Agents begin with an initial amount of wealth, $init_coin$, which was set to 0 in most circumstances. The model runs for a certain number of time steps, $t\_max$, between rounds of taxation. Each time step represents a pay period in which an agent collects an income determined by their skill. This wealth accumulates until the tax round, when agents are taxed and the collected tax is evenly redistributed amongst the agents. This cycle repeats a set number of times, $n\_iter$, until the model finishes and reports final results.

## 4.2 The Tax Code

The tax system used in this model is intended to be loosely based of the progressive tax system used in the United States. The system has seven tax brackets, each represented by a percentile. Agents are sorted into brackets based on the percentile of their wealth relative to other agents in the economy. Each tax bracket has its own tax rate determined by the reinforcement learner, which is applied only to wealth that falls into that bracket. For example, if the highest bracket has a minimum wealth of 2000 and a tax rate of 10%, an agent will be taxed at that 10% rate only on wealth that falls above the minimum value for that bracket.

The tax code determines which bracket every agent falls into based on their wealth and taxes them accordingly. The total taxed wealth is summed and then evenly redistributed amongst all agents. This process is completed after a set number of rounds of income collection by the agents.

### 4.3 Results

When the model completes its run, it outputs the final tax policy, a 1 by 7 array filled with the wealth values of each agent, and the Gini, Sen, and Rawlsian SWFs for the final wealth distribution.

After running hundreds of trials, modifying numerous different factors in the model such as the number of training iterations, the learning rate of the model, the activation function, and the number of layers in the neural net, there was one very clear trend in the tax policies generated: every final policy consisted of entirely 0s and 1s. This means every single tax rate was either 0% or 100%. Examples of these policies and their corresponding SWFs for various numbers of training iterations can be seen in Table 1.

| **n_iterations** | **Tax Policy** | **Gini** | **Sen** | **Rawlsian** |
|---|---|---|---|---|
| 150 | [0., 0., 0., 0., 0., 1., 1.] | 2.3493312245603803e-06 | 4199.249957462055 | 4196.789977591393 |
| 500 | [1., 1., 1., 1., 1., 0., 1.] | 4.704895950613519e-08 | 7118.552408253258 | 7118.519251123787 |
| 500 | [0., 1., 0., 0., 0., 0., 0.] | 0.2135801420686243 | 1630.6183513363264 | 1019.7974441832396 |
| 1000 | [0., 0., 1., 1., 1., 0., 1.] | 4.6880398496540465e-06 | 3385.832858783289 | 3380.5703273472645 |
| 1000 | [0., 0., 1., 1., 1., 1., 0.] | 0.2770259757601541 | 2002.7600338521747 | 1218.288506185662 |
| 10000 | [1., 1., 0., 1., 1., 0., 0.] | 0.3302953433381619 | 1653.2953436180965 | 866.693637105453 |

Table 1: A selection of tax policies and their corresponding SWFs for varying numbers of training iterations. Every tax policy generated by the RL learner consists of entirely 1s and 0s. Policies with a 1 in the highest bracket had Gini coefficients of effectively 0.

Another trend emerged after all of these trials that can be seen in Table 1: every tax policy with a 1 in the highest bracket has a Gini coefficient of essentially 0. This is understandable, because taxing the wealthiest individuals at 100% for the highest portion of their wealth should eventually lead to equality. This then begs the question, why did the learner not simply put a 1 in the highest bracket for every run? Further analysis of the model itself uncovered several issues that could have been responsible, as outlined in Section 4.4.

### 4.4 Issues with Initial Model

The initial version of the model had several issues that were progressively fixed as the research went on. The first issue was with the percentiles used to define the tax brackets. The original code had defined the percentiles as a numpy array with values [0.124,0.493,0.812,0.95,0.971,0.995], which would later be used with the numpy.perentile function to separate the agents into their brackets based on their wealth percentile. However, the numpy.percentile function takes whole number inputs as the

percentiles, meaning the original brackets had all but the least wealthy 0.995% of agents in the top tax bracket. This is one explanation for why the learner was able to consistently create tax functions that minimized the Gini coefficient, especially with a 1 in the top bracket: the wealthiest 99.005% of agents were being taxed at a rate of 100% and the wealth of that 99.005% of the population was then redistributed. This process will always result in a Gini coefficient that approaches 0. This was remedied by changing the brackets to a numpy array with values [12.4,49.3,81.2,95,97.1,99.5].

The second issue lies within the tax code. The taxation process separated agents into separate arrays based on what tax bracket they fell into and calculated their owed tax. However, the calculated tax payment was in no way tied to the index of the payee in the overall population, so there was no guarantee that every agent was paying the correct amount. The learner cannot create optimal policies if agents are paying random amounts instead of the rates it determined. The tax code had to be completely rewritten to remedy this. The new version of the tax code iterates through every agent in the population, determines what tax bracket they fall into, and taxes them accordingly, all in one step. This prevents the agents and their respective tax payments from being jumbled, but also causes the code to run much slower, as the for-loop is much more computationally intense. With 7 tax brackets and 1,000 agents, the kernel would run out of memory before the model had finished running. To avoid this problem, the number of brackets was reduced to 3 with percentile array [33.3,66.7], and the number of agents was reduced to varying values under 100.

Any results shown from here forward that have a 7-value tax policy used the original tax code and 1000 agents. Results with a 3-value tax policy used the newest version of the tax code and the number of agents specified in that section.

# 5 Further Modifications to the Model

The purpose of creating this model and reinforcement learner was to generate tax policies that maximized certain SWFs. Though the learner successfully minimized the Gini coefficient and maximized equality, the binary tax policies that the model output would be unusable in a real-world economy. Numerous modifications were made to the model in an attempt to generate more realistic policies without 0% and 100% tax rates.

## 5.1 The BDY Game

One attempt at avoiding binary tax policies was to replace the Accumulation Model with a version of the BDY Game. The BDY Game is an agent-based model that represents a rudimentary economy. The model uses a population of agents who each start with a pre-determined wealth. Two agents

are randomly chosen from the population, one as a "winner," the other a "loser." The "loser" gives a predetermined amount of wealth to the "winner." These exchanges continue until the model is stopped. [7] [5]

The following code was used to replace the accumulation model with the BDY game:

```
for n in range(n_rounds):
        payer_index = np.random.choice(np.argwhere(money > 0).flatten(),1)[0]
        payee_index = np.random.choice(len(money),1)[0]
        while payee_index == payer_index:
            payee_index = np.random.choice(len(money),1)[0]
        money[payer_index] = money[payer_index] - del_wealth #exchange
        money[payee_index] = money[payee_index] + del_wealth
```

The loser is chosen from all agents with a wealth greater than 0, so agents are not allowed to go into debt. The winner is chosen from all agents. If the same agent is chosen as both the winner and loser, a new winner is chosen. The winner gains a set amount of wealth *del_wealth*, and the loser loses *del_wealth*. This process repeats for a set number of times, $n_rounds$.

The BDY game was unsuccessful at avoiding binary tax policies, as well as maximizing equality in the economy. Results of the final version of the BDY model using the Sen reward function can be seen in Table 2.

| **Tax Policy** | **Gini** | **Sen** | **Rawlsian** |
|---|---|---|---|
| [1., 0., 1., 0., 0., 1., 0.] | 0.4697727343724323 | 53.3060516287214 | -0.4452258788988519 |
| [0., 0., 0., 0., 0., 0., 0.] | 0.4703864314075147 | 53.10505470560863 | -0.756070332009898 |
| [1., 0., 1., 1., 1., 0., 0.] | 0.44729242400080016 | 74.43425675607406 | 5.4539887666935645 |
| [1., 0., 0., 1., 1., 1., 1.] | 8.08427621399139e-05 | 83.40317292068195 | 82.73560720868196 |
| [1., 0., 0., 1., 0., 1., 0.] | 0.5041682675786782 | 49.27673383990851 | -0.11916607378627955 |

Table 2: Results of the BDY game version of the model using the Sen reward function. The learner performed slightly worse with this model than the accumulation model. Gini coefficients were higher overall, and some trials had agents who finished in debt.

These results are similar to the accumulation model, in that the one tax policy with a 1 in the highest bracket resulted in a near-zero Gini coefficient. The other tax policies resulted in worse Gini coefficients than the accumulation models, with some agents even going into debt as seen through the Rawlsian SWF value.

### 5.2 Non-Zero Gini Coefficient Reward Function

After consulting with Dr. Mica Kurtz about the binary tax policies, he proposed that the code may be outputting boundary values because it was attempting to achieve a boundary state; since the code was targeting the minimum possible Gini coefficient, perhaps the model was using minimum and maximum possible tax rates to try to achieve its goal. With this concept in mind, two new reward functions were designed to target non-zero Gini coefficients. The first reward function is as follows:

$$r = -g^2 + 2gg^* - (g^*)^2 \tag{2}$$

where $g$ is the Gini coefficient of the current wealth distribution and $g^*$ is the target Gini value. This equation represents an inverted parabola with an absolute maximum at $g = g^*$. A graphical representation of this reward function can be seen in Figure 1.

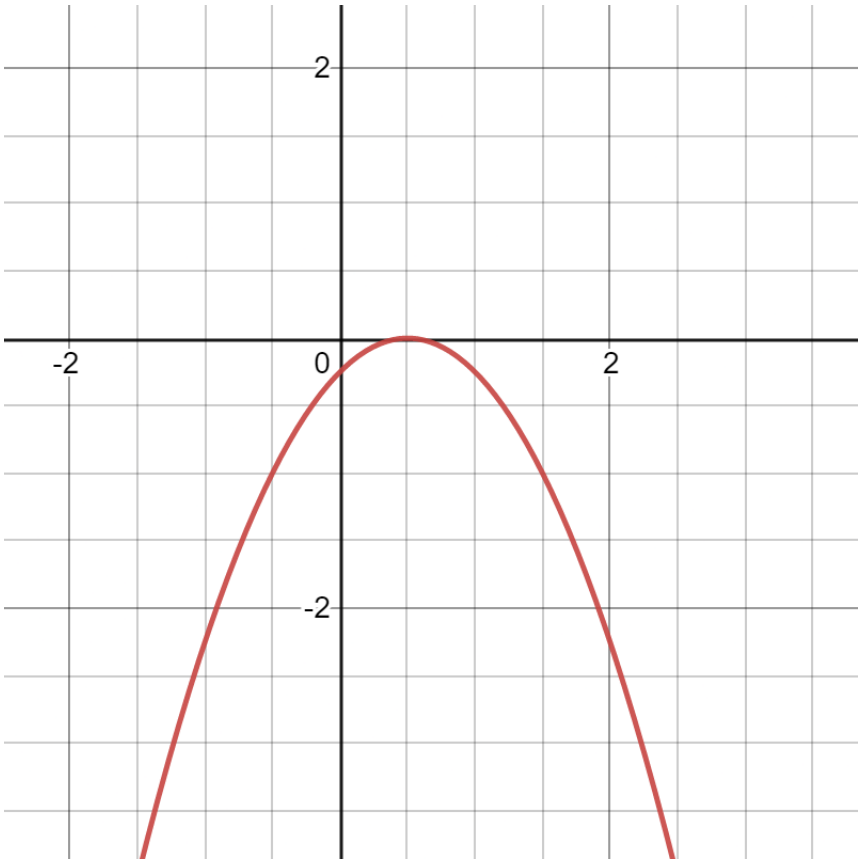



Figure 1: A graphical representation of the parabolic non-zero Gini reward function with a target Gini value of $g^* = 0.5$. The function has one absolute maximum located at the target Gini value and no other local extrema.

The second reward function is as follows:

$$r = \begin{cases} g & \text{if } g \leq g^* \\ -g + 2g^* & \text{if } g > g^* \end{cases} \tag{3}$$

where $g$ and $g^*$ are defined the same as in [2]. This function is a linear piecewise function with an absolute maximum at $g = g^*$. A graphical representation of this reward function can be seen in Figure 2.

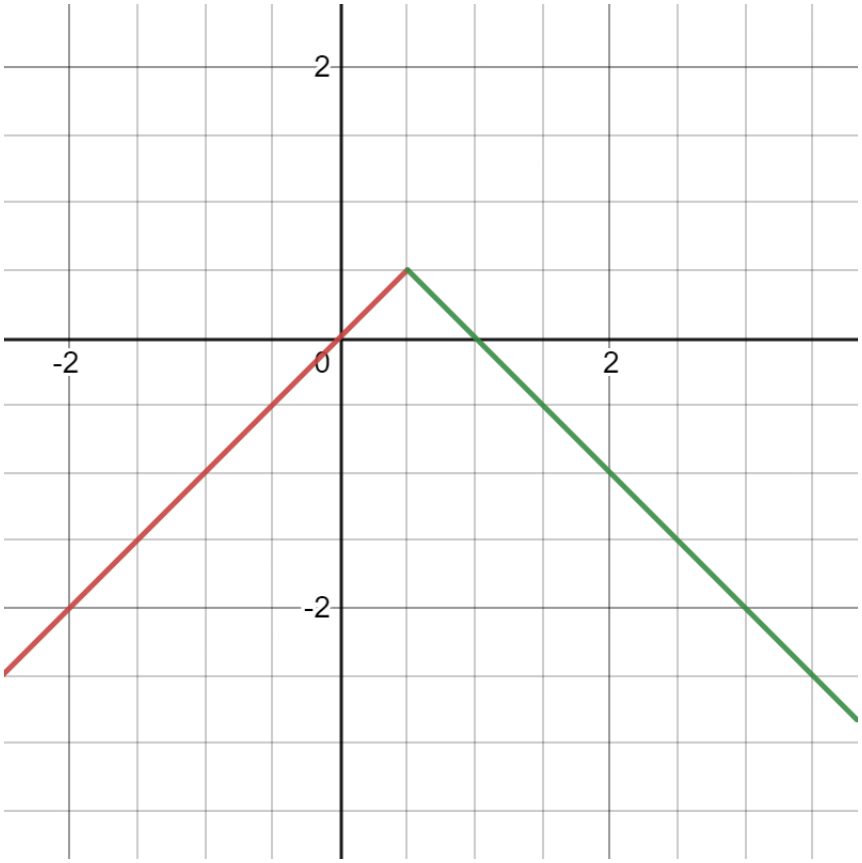



Figure 2: A graphical representation of the linear piecewise non-zero Gini reward function with a target Gini $g^* = 0.5$. The function has one absolute maximum at the target Gini value and no other local extrema.

Since both [2] and [3] have maximums only at $g = g^*$, the learner should generate tax rates that target a wealth distribution with a Gini coefficient of $g^*$ in order to maximize its reward.

Both of these reward functions produced extremely inconsistent results. The learner did a poor job of creating policies to target a specific Gini coefficient and predominantly still targeted 0. It also continued to create binary tax policies, regardless of how many training iterations it was given. Representative results from trials using the parabolic non-zero Gini function (Eq 2) with 1500 training

iterations can be seen in Table 3.

| Target Gini | Tax policy | Gini | Sen | Rawlsian |
|---|---|---|---|---|
| 0.8 | [0., 1., 1., 0., 1., 1., 1.] | 1.4361438264352535e-08 | 1671.5057406472918 | 1671.5033641298335 |
| 0.4 | [1., 1., 1., 1., 1., 1., 1.] | 0.0 | 1457.5395221270053 | 1457.5395221270053 |
| 0.3 | [0., 0., 1., 1., 0., 0., 0.] | 0.7536714724313082 | 435.4999895477045 | 27.049799888495482 |
| 0.3 | [1., 0., 1., 1., 1., 1., 1.] | 2.1556939230003885e-08 | 4517.188257581 | 4517.178617282269 |
| 0.1 | [0., 1., 0., 0., 0., 0., 0.] | 0.2933374359981551 | 872.0816938922709 | 572.0418140932265 |
| 0.1 | [1., 0., 0., 1., 1., 1., 1.] | 6.590294895295639e-07 | 2327.7545331306924 | 2327.6026612013147 |

Table 3: Select results using the parabolic non-zero Gini reward function with 1500 training iterations. Despite the reward functions having only one maximum located at the target Gini value, the learner was unable to create tax policies that would result in a wealth distribution with the targeted Gini coefficient.

### 5.3 Income Tax Model

This model was meant to loosely parallel to United States tax system, which is, in the most basic sense, a 7-bracket progressive income tax. The tax bracket percentiles in the model, however, were based off of the wealth of the agents, and their entire accumulated wealth was taxed every round. The U.S. tax system is based off of income, not wealth. A wealth tax will always approach a 0 Gini coefficient, because none of the agents' wealth remains truly untouched by the taxation process. To remedy this, the taxation process was rewritten to use the agents' income instead of wealth. Their income, based on their initial skill score, is used to determine tax brackets in the current version of the model. The only wealth that is taxed is wealth that has been accumulated since the last taxation round, not an agent's total wealth.

Trials were run with the income tax model using the non-zero Gini and Sen reward functions. Results using the non-zero Gini reward function with a target Gini of 0.3, 10 agents, and varying numbers of training iterations can be seen in Table 4. Results using the Sen reward function with 50 agents and varying numbers of training iterations can be seen in Table 5.

| N_iter | Tax policy | Gini | Sen | Rawlsian |
|---|---|---|---|---|
| 50 | [0., 0., 1.] | 0.17423875916861897 | 840.3744169414564 | 549.0514829865062 |
| 50 | [0., 1., 1.] | 0.031405451362707136 | 2321.579785654157 | 2294.202964097496 |
| 30 | [1., 0., 1.] | 0.07617230437270807 | 1259.392594917953 | 1142.0420879142264 |
| 30 | [1., 1., 1.] | 0.0008636999070268186 | 1142.295137314656 | 1140.9271578244015 |

Table 4: Results of the income tax model using the non-zero Gini reward function with a target Gini value of 0.3 and varied training iterations. The Gini coefficients were no longer strictly driven to 0 as they were with the wealth tax, but was unable to achieve the target value.

| N_iter | Tax Policy | Gini | Sen | Rawlsian |
|---|---|---|---|---|
| 30 | [0., 1., 1.] | 0.015519749665351091 | 2230.9360456317945 | 2208.4409735954046 |
| 10 | [1., 1., 0.] | 0.9161094705850563 | 903.4052357173947 | 478.5442469169688 |
| 10 | [1., 0., 0.] | 0.5488491446806997 | 436.82749362361375 | 192.26853104013202 |

Table 5: Results of the income tax model using the Sen reward function with 50 agents and varying numbers of training iterations. The Gini coefficients were no longer strictly driven to 0 as they were with the wealth tax, but the learner was unable to successfully maximize the Sen.

With the income tax model, the Gini coefficients were no longer driven to 0 as they were with the wealth tax, allowing the economic model to function more realistically. The learner did not perform as well as expected when using the income tax. The tax policies produced were still entirely binary. When using the non-zero Gini reward function, the learner was unable to achieve the target Gini, though it was significantly closer than when using the wealth tax. When using the Sen reward function, the model was unable to maximize its reward and target a 0 Gini, with one trial actually having a final Gini over 0.9.

# 6 Conclusions

Though we were never able to make the model function as intended, this was an incredible learning experience. Since the start of this project, I have learned how to code numerous different machine learning algorithms, learned about progressive tax systems and social welfare functions, and learned the inner workings of this reinforcement learning model inside and out. The importance of the environment has been one of the most impactful lessons. Nearly every error in the code lay in the environment, and nearly every modification to avoid the binary tax policies affected the environment. The environment is where the learner is trained and run, so every detail must be perfect or it will affect the outcome of

the model. The model relies solely on input from the environment to train itself, so any mistake can ruin its ability to learn.

## References


[1] QuantEcon 0.5.3 documentation: Inequality. https://quanteconpy.readthedocs.io/en/latest/tools/inequality.html. Accessed: 04-06-2022.

[2] Jan Abrell, Mirjam Kosch, and Sebastian Rausch. How effective was the uk carbon tax? - a machine learning approach to policy evaluation. *CER-ETH – Center of Economic Research at ETH Zurich Working Paper*, 2019.

[3] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark, 2020.

[4] Pietro Battiston, Simona Gamba, and Alessandro Santoro. Optimizing tax administration policies with machine learning. *University of Milan Bicocca Department of Economics, Management and Statistics Working Paper*, 2020.

[5] A. Dragulescu and V.M. Yakovenko. Statistical mechanics of money. *The European Physical Journal B*, 17(4):723–729, 2000.

[6] Lukasz Kidzinski, Sharada Prasanna Mohanty, Carmichael F. Ong, Zhewei Huang, Shuchang Zhou, Anton Pechenko, Adam Stelmaszczyk, Piotr Jarosik, Mikhail Pavlov, Sergey Kolesnikov, Sergey M. Plis, Zhibo Chen, Zhizheng Zhang, Jiale Chen, Jun Shi, Zhuobin Zheng, Chun Yuan, Zhihui Lin, Henryk Michalewski, Piotr Milos, Blazej Osinski, Andrew Melnik, Malte Schilling, Helge J. Ritter, Sean F. Carroll, Jennifer L. Hicks, Sergey Levine, Marcel Salathé, and Scott L. Delp. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. *CoRR*, abs/1804.00361, 2018.

[7] Christopher W. Kulp, Michael Kurtz, Nathaniel Wilston, and Luke Quigley. The effect of various tax and redistribution models on the gini coefficient of simple exchange games. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(8):083118, 2019.

[8] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[9] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap,

and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[10] Martin Zumaya, Rita Guerrero, Eduardo Islas, Omar Pineda, Carlos Gershenson, Gerardo Iñiguez, Carlos Pineda, and José R. Nicolás-Carlock. *Identifying Tax Evasion in Mexico with Tools from Network Science and Machine Learning*, pages 89–113. Springer International Publishing, 2021.