University of Zurich[UZH]

Solving economic models for ranges of economic parameters using deep learning

Master's Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Arts in Banking and Finance

Author

Junlei Chen

Bülachstrasse 7F, 8057 Zürich

18-744-847

junlei.chen@uzh.ch

Supervisor

Prof. Dr. Felix Kübler

Sub-supervisor

Marlon Azinović

Date of Submission: March 10, 2021

Abstract

In this thesis, I introduce a method of solving macroeconomic models for entire ranges of economic parameters using deep learning. By constructing a loss function for training the neural network, taking into account the equilibrium conditions (Euler Equation) and other considerations, I apply the method to solve two different models.[1] They feature incomplete markets, idiosyncratic risk, occasionally binding constraint, non-stationary shock process, and asset pricing with non-trivial market-clearing conditions. The reached accuracy demonstrates that the implemented method can compute satisfactory approximations. Furthermore, I present a novel algorithm to calibrate the extended parameters matching with real-world economic indices using gradient descent, which can bring insights to public policy construction, economic decision-making process, etc.[2]

[1]The python code implementing the algorithm mentioned in the thesis is linked here: https://github.com/Junlei-Chen/Master-Thesis.git.

[2]I would like to thank Prof. Felix Kübler for the valuable comments and guidance. I also extremely appreciate Marlon Azinović's dedicated support and research suggestions during the thesis writing. Furthermore, I acknowledge my flatmates Nina Desboefs, Stefan Schöpf, Peter Szemraj for the exciting ideas exchange during our lunches and dinners. Finally, my deep and sincere gratitude to my fiancé Alexander Böhm for his continuous and unparalleled love.

Contents

1 Introduction

Solving macroeconomic models with stochastic, high-dimensional, and strong non-linear features has been an exceedingly challenging task, due to the notoriously difficult problem known as the "curse of dimensionality"[3]. However, with the development of state-of-the-art Deep Learning (DL) algorithms, new methods for addressing this issue are being developed in recent years (see e.g.[1][5][12]). Additionally, in the context of *functional rational expectations equilibrium* (FREE)[10], constructing neural networks, which approximate the equilibrium functions and are trained with gradient-descent based optimization seems fairly generic and suitable. The standard approach in the literature to obtain model solutions for different parameter values is solving the model from scratch several times using different economic parameters. It has been a rarely explored field where one could add the entire ranges of economic parameters to s neural network as the input variables and also calibrate the parameters matching with the targeted economic indices.

In this thesis I investigate the potential of using neural networks to approximate model solutions for entire ranges of economic parameters. This thesis mainly contributes to the scarce research on the parameter estimation process of the algorithms to solve economic models with deep learning. Additionally, I present a novel algorithm to calibrate the concerned economic parameters pinning down related economic indices. Using the obtained solution to fit real-world data would bring new insights to the process of policy construction and economic decision-making, etc.

Concretely, I solve two nonlinear dynamic stochastic general equilibrium models. The first benchmark model is introduced by Brock and Mirman (1972)[4], which introduces the first optimizing growth model with stochastic productivity shocks. I present the deep learning approach of solving this model with the input extension of discount factor β and the risk aversion parameter γ and calibrate the parameters with the economic target indices and also visualize the interaction between the economic factors. The second model is a modified version inspired by Heaton and Lucas (1996)[7], with two types of representative agents, incomplete markets, occasionally binding constraints, stochastic shocks and equilibrium asset pricing. For this model, I also investigate the potential of solving it with adding discount factor β into the input space and calibrating the parameter with real-world data.

The remaining part of the thesis is organized as follows: section 2 gives a brief review of the related literature. Section 3 describes the benchmark model from Brock and Mirman (1972)[4] and illustrates the methodology I utilize to solve the model. In section 4, I present the data sampling process, the loss function construction details, the training performance of the model, as well as the parameter calibration with discount factor β. In section 5 I showcase the extension of the algorithm input with risk aversion parameter γ and introduce the multiple parameter calibration algorithm. Section 6 presents this methodology adapted on a more complicated model inspired by Heaton and Lucas (1996)[7], featuring two types of agents as well as incomplete market assump-

tions. Section 7 discusses some practical issues that occurred when solving the models and programming the implementations which can bring values for further research. Section 8 is my conclusion and discusses remaining challenges and possible future research directions.

2 Literature Review

The approach of solving recursive equilibria numerically with computational methods can be dated back to the 1970s[11]. However, the research making explicit use of recent developments in machine learning to compute approximate equilibrium in dynamic models is still relatively rare. I summarize three highly relevant papers in this field as follows:

Duarte (2018)[5]:
By solving an economic model with structure estimation, this paper successfully shows how gradient-based optimization methods can be used to estimate stochastic dynamic models in the continuous time context. This paper leverages new powerful approximate dynamic programming techniques that allow for the global solution of high-dimensional stochastic dynamic problems by augmenting the state space and using deep neural network to learn the moments by observing raw outcomes of simulations. Different from this paper, the models I try to solve in my thesis are discrete-time based.

Maliar, Maliar, and Winant (2019)[12]:
This paper introduces a generic unsupervised learning approach that uses three fundamental objects of economic dynamics - the maximization of lifetime reward, the minimization of the Bellman residuals, and the minimization of the Euler equation residuals. This independent work with the use of Euler equation residuals as a cost function in this paper is similar to the approach of the algorithm I present in my thesis.

Azinovic, Gaegauf, and Scheidegger (2019)[1]:
This paper introduces the concept of deep equilibrium nets - neural networks that directly approximate all equilibrium functions and that are trained in an unsupervised fashion to satisfy all equilibrium conditions along simulated paths of the economy. The economic models solved are featured with a substantial amount of heterogeneity, significant uncertainty, and occasionally binding constraints. They show that deep equilibrium nets can solve rich and economically relevant models fast and efficiently.

Scheidegger, S. and Bilionis, I. (2019)[13]:
This paper presents a novel computational framework to compute global solutions of high-dimensional dynamic stochastic economic models on potentially irregularly-shaped geometries. Specifically, applying Gaussian process regression in combination with active subspaces and Bayesian Gaussian

mixture models, they demonstrate the ability of the framework to learn the value and policy functions as well as ergodic sets. They also show that their framework can address parameter uncertainty and can provide predictive confidence intervals for policies that correspond to the epistemic uncertainty induced by limited data, which has the similarity to my thesis which extends the deep learning input space with the economic parameters.

The papers discussed above all presented advanced computational approaches and efficient techniques used in solving high-dimensional stochastic dynamic models. What makes my thesis different from these papers' contribution is that I investigate the possibility of extending the deep learning input space with the entire ranges of economic parameters, as well as the parameter calibration with pinning down different real-world economic targets. Since time preference and risk aversion have always been prime components in designing effective policy, I mainly focus on the ranges of discount factor β and γ extension in this thesis.

3 The Benchmark Economic Model

3.1 Model Description

The benchmark economic model I choose to solve here is the Brock and Mirman (1972)[4] model. This model provided the first optimizing growth model with unpredictable (stochastic) shocks. Because this model has the features mentioned above while being not too complicated and can already be solved analytically I choose it as an appropriate benchmark model to start with.

The social planner's goal is to solve the problem:

$$\max \ \mathbb{E}\left[\sum_{n=0}^{\infty} \beta^n log C_{t+n}\right] \tag{1}$$

s.t.

$$K_{t+1} = Y_t - C_t$$
$$Y_{t+1} = A_{t+1} K_{t+1}^{\alpha}$$

where A_t is the level of productivity in period t, which is now allowed to be stochastic. In this model the capital stock is not useful as a state variable: Because capital has a 100 percent depreciation rate, all that matters to the consumer when choosing how much to consume is how much income they have at that moment, and not how that income breaks down into a part due to K and a part due to A.

While trying to solve this model, the first step is to rewrite the problem in Bellman equation form

$$V_t(Y_t) = \max_{C_t} \ log C_t + \beta \, \mathbb{E}_t[V_{t+1}(Y_{t+1})] \tag{2}$$

and take the first order condition:

$$u'(C_t) = \beta\, \mathbb{E}_t[A_{t+1}\alpha K_{t+1}^{\alpha-1} u'(C_{t+1})]$$

$$\frac{1}{C_t} = \beta\, \mathbb{E}_t \left[\frac{A_{t+1}\alpha K_{t+1}^{\alpha-1}}{C_{t+1}}\right]$$

$$1 = \beta\, \mathbb{E}_t \left[\alpha A_{t+1} K_{t+1}^{\alpha-1} \frac{C_t}{C_{t+1}}\right] \tag{3}$$

where $\alpha A_{t+1} K_{t+1}^{\alpha-1} \equiv R_{t+1}$. Here, the definition of R_{t+1} helps clarify the relationship of this equation to the usual consumption Euler equation.

Now I show that this FOC is satisfied by the consumption function $C_t = \kappa Y_t$, where $\kappa = 1 - \alpha\beta$. To see this, first note that the proposed consumption rule implies that $K_{t+1} = (1-\kappa)Y_t$.

The first order condition says

$$\begin{aligned}
1 &= \beta\, \mathbb{E}_t \left[\alpha \frac{A_{t+1}K_{t+1}^{\alpha}}{K_{t+1}} \frac{\kappa Y_t}{\kappa Y_{t+1}}\right] \\
&= \beta\, \mathbb{E}_t \left[\alpha \frac{Y_{t+1}}{K_{t+1}} \frac{\kappa Y_t}{\kappa Y_{t+1}}\right] \\
&= \beta \left[\alpha \frac{Y_t}{K_{t+1}}\right] \\
&= \beta \left[\alpha \frac{Y_t}{Y_t - C_t}\right] \\
&= \beta \left[\alpha \frac{Y_t}{Y_t(1-\kappa)}\right] \\
&= \beta \left[\alpha \frac{1}{(1-\kappa)}\right] \\
(1-\kappa) &= \alpha\beta \\
\kappa &= 1 - \alpha\beta
\end{aligned}$$

Therefore, the analytical solution to the Brock and Mirman (1972)[4] model is presented here. This serves as a benchmark to check the accuracy of the neural network's output.

3.2 Methodology

The approach I use to solve the model proposed above is to create an algorithm using a neural network as an approximator, featuring the function mapping the state of the economy to choices of savings and investment, which corresponds to the equilibrium conditions. Noticing that, should the solution be correct, the Euler Equation would hold. Therefore, the loss function is constructed ensuring this condition as well as some other considerations.

More specifically, I choose to use densely connected feedforward neural networks as function approximators since they have desirable features as follows. It has been shown

that as a type of universal function approximators, neural networks can leverage distinct local, highly non-linear features, process a large amount of high-dimensional input data, and have the potential to overcome the curse of dimensionality to some extent when solving for example partial differential equations and computing optimal stopping rules (see e.g.[6][2]).

Given hyper-parameters $\{K, \{m_i\}_{i=1}^K, \{\sigma_i(\cdot)\}_{i=1}^K\}$ and trainable parameters $\boldsymbol{\rho}$, a neural network $\mathcal{N}_{\boldsymbol{\rho}}$ encodes the mapping

$$\mathbf{x} \to \mathcal{N}_{\boldsymbol{\rho}}(\mathbf{x}) = \sigma_K(\mathbf{W}_K \dots \sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \dots + \mathbf{b}_K), \tag{4}$$

where $\mathbf{W}_i \in \mathbb{R}^{m_{i+1} \times m_i}$ are matrices often referred to as weight matrices, and $\mathbf{b}_i \in \mathbb{R}^{m_{i+1}}$ are vectors often referred to as bias vectors. The vector $\boldsymbol{\rho}$ represents all entries of the weight matrices and the bias vectors. K is referred to as the number of layers of the neural network within the model's construction, and m_i as the number of nodes in layer i. The nonlinear functions σ_i are referred to as activation functions and are applied element-wise to each entry of a vector: $\sigma_i(\mathbf{x}) = [\sigma_i(x_1), \dots, \sigma_i(x_{m_{i+1}})]^T$. Therefore, a densely connected feedforward neural network is created by a sequence of matrix-vector multiplications followed by the application of an activation function.

The goal of the algorithm is to approximate the equilibrium function $\boldsymbol{\theta}_K : \mathcal{Z} \times \mathbb{R}^2 \to \mathbb{R}$ denotes the capital investment functions, such that for all states $\mathbf{x} := [z, k, \beta]^T \in \mathcal{Z} \times \mathbb{R}^2$, where $z \in \mathcal{Z}$ denotes the exogenous shock, capital holding k, as well as the discount parameter β, the Euler equation 3 is always fulfilled.

In order to do so, I combine four elements which are given by: (i) an appropriate type of function approximators; (ii) a loss function measuring the accuracy of a given approximation at a given state; (iii) an updating mechanism to improve the approximation; and (iv) a sampling method for choosing states for updating and the evaluation of the approximation quality.

Specifically for my algorithm, I choose the multi-layer neural network as a function approximator. The loss function is constructed using the errors in the equilibrium condition (Euler equation) and the neural network parameters are updated using variants of gradient descent approaches. To update the parameters of the neural networks, as well as to evaluate the quality of approximation, I randomly sample state variables for ranges of feasible numbers to feed into the neural networks.

4 Deep Learning Implementation

4.1 Data Sampling

Within the context of TensorFlow 2.0, it is possible to generate the sample state variables randomly following some specified distribution. Firstly, I assign necessary parameters to the Brock-Mirman model as follows (the input extension with the relative risk aversion parameter γ will be shown in the next section):

Relative risk aversion γ	Capital share α	TFP η
1	0.3	{0.97,1.03}

Table 1: Parameterization of the economic parameters

The total factor productivity (TFP) η takes two values, $\eta \in \{0.97, 1.03\}$ and evolves with a transition matrix

$$\Pi^{\eta} = \begin{bmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{bmatrix} \tag{5}$$

where $\Pi^{\eta}_{i,j}$ denotes the probability of a transition from the exogenous shock i to exogenous shock j, where $i, j \in \mathcal{Z}$. This describes the economy's tendency to remain in a booming state rather than turning into a recession and vice versa

For the first period I set the probability for two exogenous shocks to be i.i.d., afterwards the shocks evolve with the persistence described above.

Moreover, the ranges for the input state variables and the parameter β is set as follows:

Capital holding k range	Discount factor β range
$[0.05,\ 0.8]$	$[0.90,\ 0.99]$

Table 2: Ranges for input endogenous state variable k and parameter β

4.2 Loss Function

Following Judd (1992)[8] and Azinović, Gaegauf, and Scheidegger (2019)[1], I take the relative error in the Euler equation, namely equation 3, for capital

$$e^{\mathrm{REE_{cap}}}_{\mathbf{x}}(\boldsymbol{\rho}) := \frac{\beta\, \mathbb{E}_t \left[\alpha A_{t+1} K_{t+1}^{\alpha-1} {C_{t+1}}^{-\gamma}\right]^{-\frac{1}{\gamma}}}{C_t} - 1. \tag{6}$$

Using either the relative error or the absolute error obtained from the Euler equation is equivalent when the equilibrium condition is satisfied. However, the advantage of using the relative Euler equation error is that its value's economic meaning is independent of the utility function. Concretely, it quantifies the consumption changes.

Generally, the unsupervised loss function depends on two components: the parameters of the function approximator $\boldsymbol{\rho}$ and the set of states $\mathbf{x}$ at which the desired conditions are evaluated. The latter is referred to as the *training set*, which I denote as $\mathcal{D}_{\text{train}}$. Given parameters $\boldsymbol{\rho}$ and a set of states $\mathcal{D}_{\text{train}}$, I define the loss function as the mean

squared error of the equilibrium condition:

$$\ell_{\mathcal{D}_{\text{train}}}(\boldsymbol{\rho}) := \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} \left(\frac{1}{N-1} \sum_{i=1}^{N-1} (e_{\mathbf{x}}^{i,\text{REE}_{\text{cap}}}(\boldsymbol{\rho}))^2\right) \tag{7}$$

The loss function is defined such that optimizing the trainable parameters $\boldsymbol{\rho}$ is made possible. Therefore, parameters are considered "good", if they minimize the loss function. Next I use the gradient descent to optimize the parameters $\boldsymbol{\rho}$. The gradient descent approach updates the parameters gradually in the direction in which the loss function decreases—which is:

$$\rho_k^{\text{new}} = \rho_k^{\text{old}} - \alpha^{\text{learn}} \frac{\partial \ell_{\mathcal{D}_{\text{train}}}(\boldsymbol{\rho}^{\text{old}})}{\partial \rho^{\text{old}}} \forall k \in \{1, \ldots, length(\boldsymbol{\rho})\}. \tag{8}$$

Here, I use the gradient descent optimizer Adaptive Moment Estimation (Adam)[9] which stabilizes the training of neural networks by continually adapting the learning rate and gradient. In Adam, adaptive learning rates are computed for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients similar to momentum. Whereas momentum can be seen as a ball rolling down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima on the error surface. It is shown empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

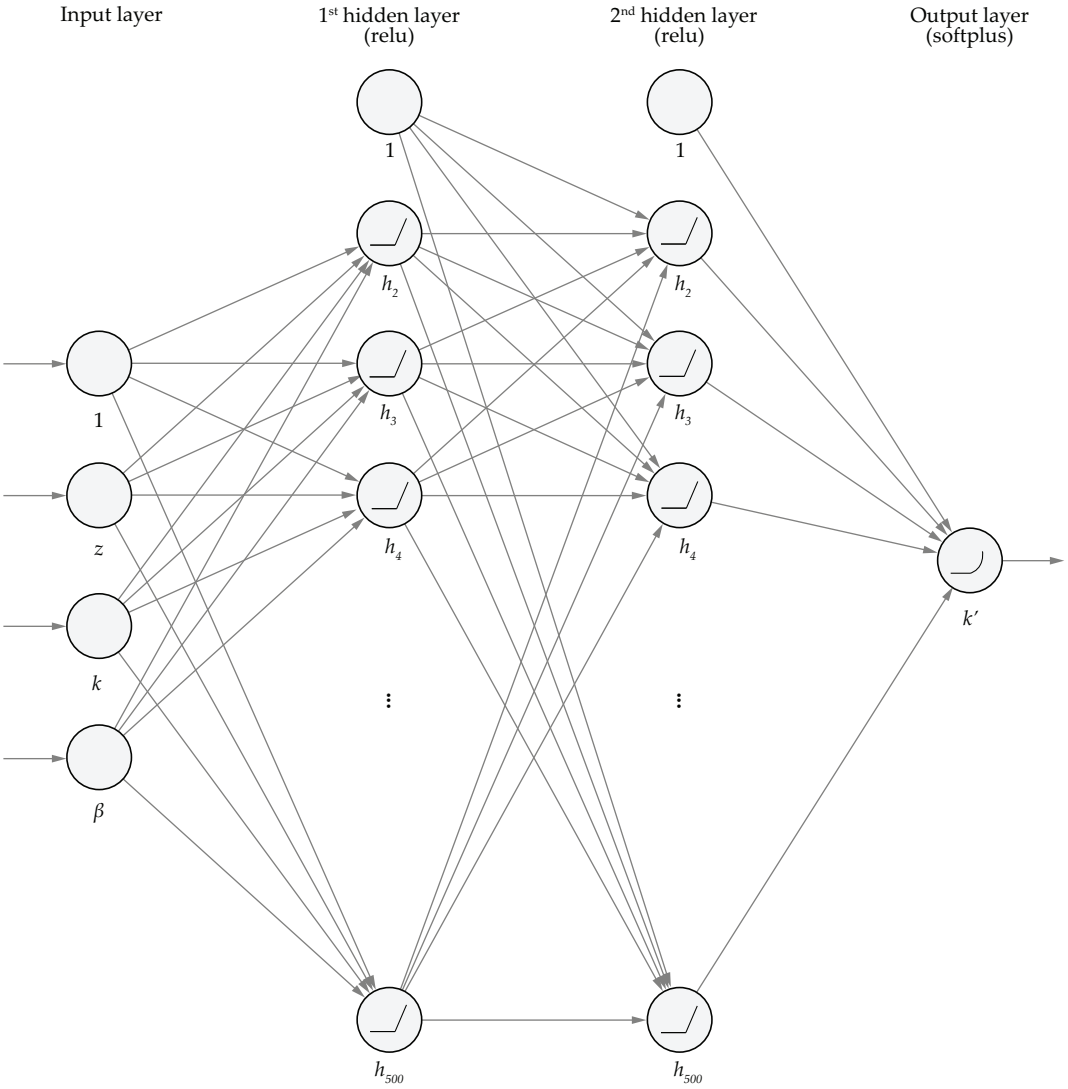

Figure 1: Schematic illustration of the neural network architecture for the deep learning process.

4.3 Model Training

I use a deep neural network with two hidden layers to solve the Brock-Mirman model. The input layer consists of $1+3$ input nodes, representing the constant 1, the exogenous shock z, the capital holding k, and the discount factor β. After the input layer, the neural network features two hidden layers with 500 and 500 relu-activated hidden nodes, respectively. The output layer consists of 1 node, activated with the softplus function to ensure that the non-negativity constraint is fulfilled. A schematic illustration of the neural network architecture is given in figure 1.

I found that it is very helpful to decrease the learning rate toward the end of the training procedure, in order to fine-tune the neural network parameters. Therefore, I switch from the learning rate of $\alpha^{\text{learn}} = 1 \times 10^{-4}$ for the first 10,000 episodes of training to $\alpha^{\text{learn}} = 5 \times 10^{-5}$ for the second 10,000 episodes of training. The chosen hyper-parameters are specified in table 3.

Episodes	Learning rate α^{learn}	Nodes hidden layers	Activations hidden layers	Activation output layer
1 - 10,000	1×10^{-4}	{500, 500}	{relu, relu}	{softplus}
10,000 - 20,000	5×10^{-5}	{500, 500}	{relu, relu}	{softplus}

Table 3: Parameterization of the hyper-parameters.

4.4 Model Performance

Next, I illustrate the performance of the neural network's approximation for the Brock-Mirman model. Figure 2 shows the evolution of the loss function during the 20,000 episodes of training. As shown, the loss function decreases quickly during the training and converges at approximately 10^{-6}. Since the plotted value is in the form of the squared error, the actual scale of the relative Euler equation is approximately 10^{-3}, which is quite accurate for a model with 3 state variables and a kinked solution, considering that the neural networks have not been trained for extremely many episodes. On a modern CPU, optimization should be done within 15 minutes. It would be dramatically faster on hardware adapted to deep-learning.

The relative Euler equation errors after training on 1,000 sampled state variables are summarized in table 4.

	mean	max	min	10	50	90
Relative Euler equation error (%)	0.590	1.103	0.000	0.000	0.041	0.122

Table 4: Relative Euler equation error. The columns show the mean, the max, and the min errors as well as the 10th, 50th, 90th percentile.

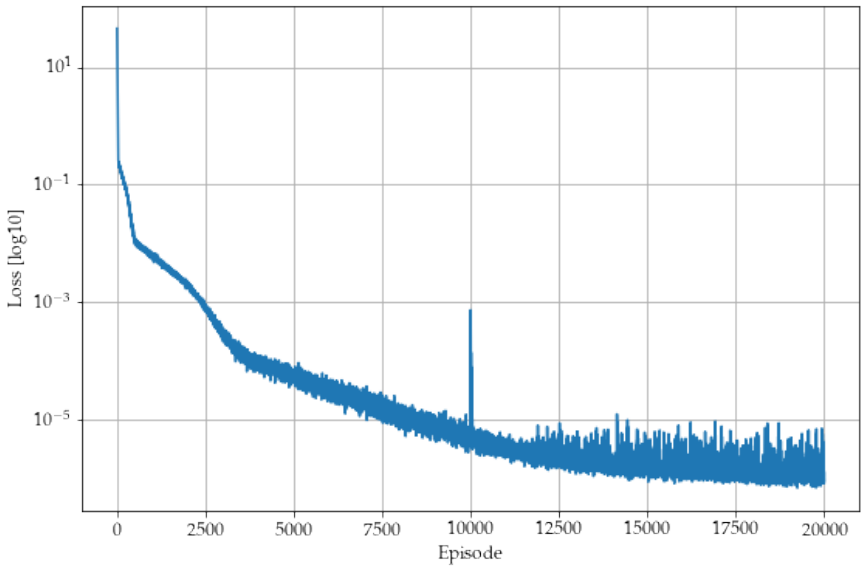

Figure 2: Evolution of the loss function during the first 10,000 episodes of training with learning rate $\alpha^{\text{learn}} = 1 \times 10^{-4}$ and the second 10,000 episodes with learning rate $\alpha^{\text{learn}} = 5 \times 10^{-5}$.

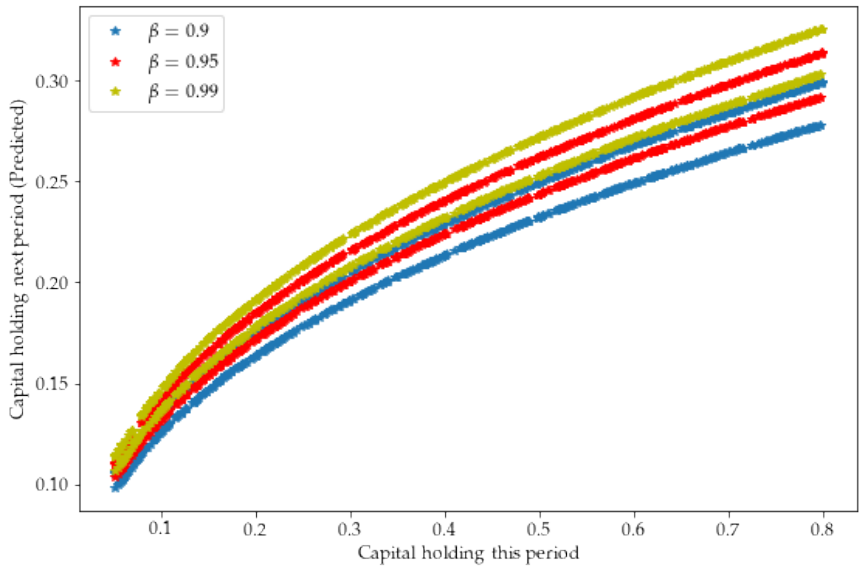

Figure 3: Capital holding this period *v.s.* Capital holding next period (Neural network prediction).

Additionally, we can see from figure 3 that the neural network is able to accurately predict the next period of capital holding for different given state variables and different

βs. In the figure it is shown that for each β, there are two different predicted corresponding capital holdings for the next period since I have two different exogenous shocks TFP $\eta = \{0.97, 1.03\}$. Also, it is intuitive that with larger discount factor β, people would have the tendency to save more for the next period. To confirm the prediction's quality, I also plot the capital holding this period K_t versus the consumption this period C_t showed in figure 4. This figure shows that with the realization of a bad shock TFP $\eta = 0.97$ today, people tend to consume less while with a good shock this period $\eta = 1.03$, people would tend to consume more which leads to a larger C_t.

Furthermore, the neural network achieves very satisfactory prediction within the feasible state variable range. As shown in figure 5, within the feasible range (the 45° line crosses the solutions part), the prediction for capital holding next period K_{t+1} and the analytical solution for the policy function $K_{t+1} = (1 - \kappa)Y_t$, where $\kappa = 1 - \alpha\beta$, are not visually distinguishable.

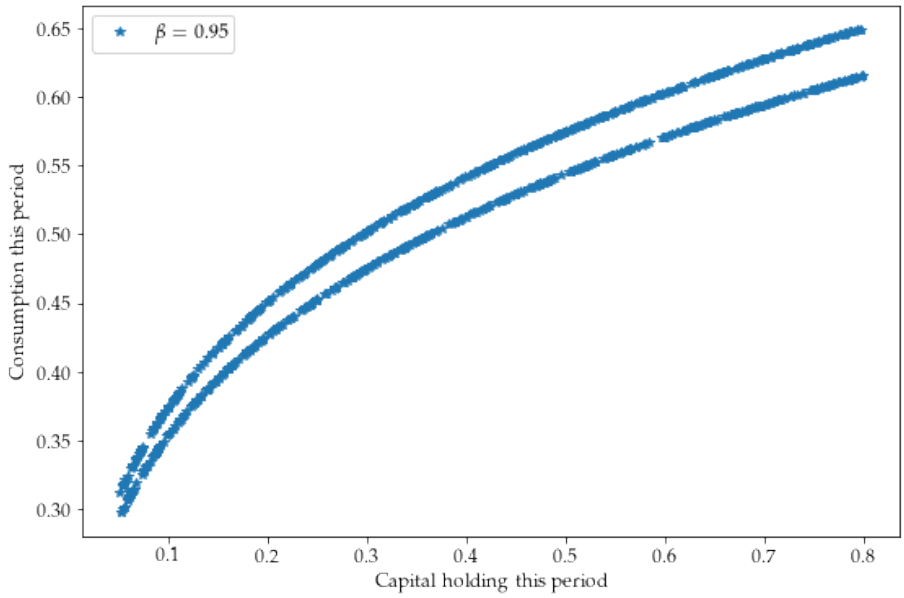

Figure 4: Capital holding this period *v.s.* Consumption this period (Simulated with neural network prediction).

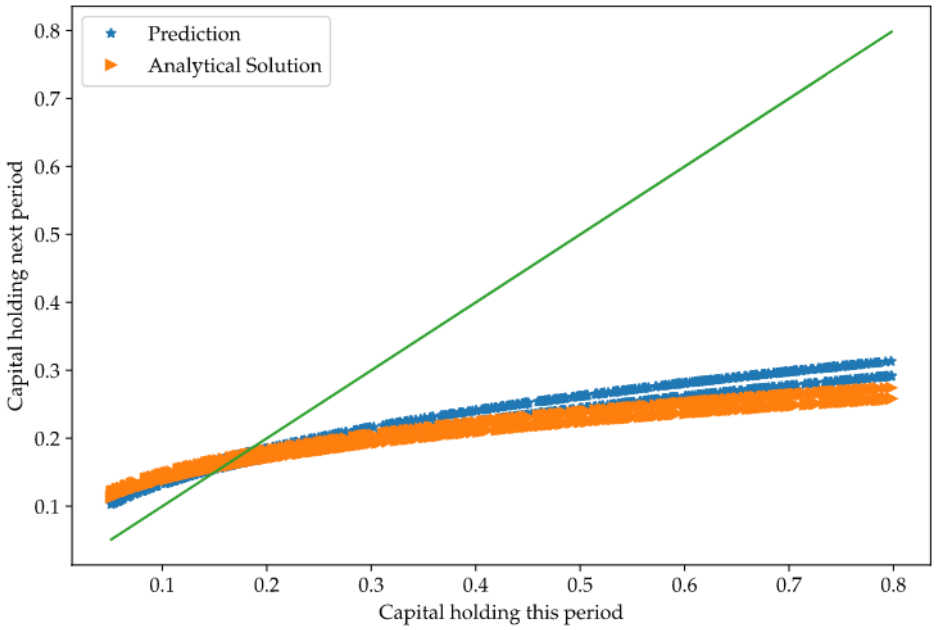

Figure 5: Prediction from neural network *v.s.* Analytical solution derived from section 3.1. The parameter β is set as 0.95. The green line shows the 45° line where $y = x$.

4.5 Parameter Calibration

As a next step, I introduce the calibration of parameter β to match real-world data. One important, common concern is the average interest rate R_t. I simulate 1,000 different states with 100 evolving periods, then I calculate the mean interest rate predicted by the neural network model. As we can see in figure 6, the average interest rate has a strong linear relationship with the range of the discount factor β.

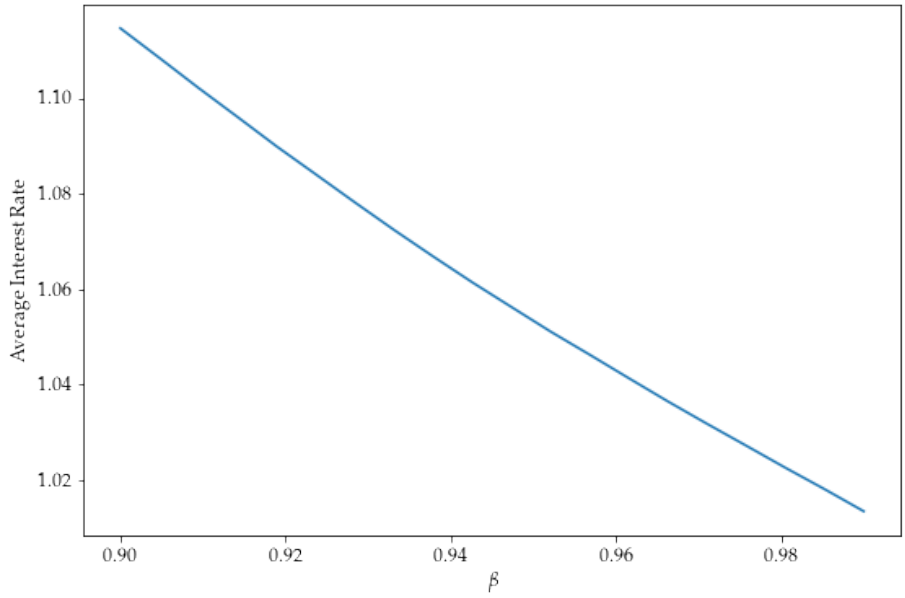

Figure 6: Average Interest Rate (Simulated with the prediction) *v.s.* Range of β.

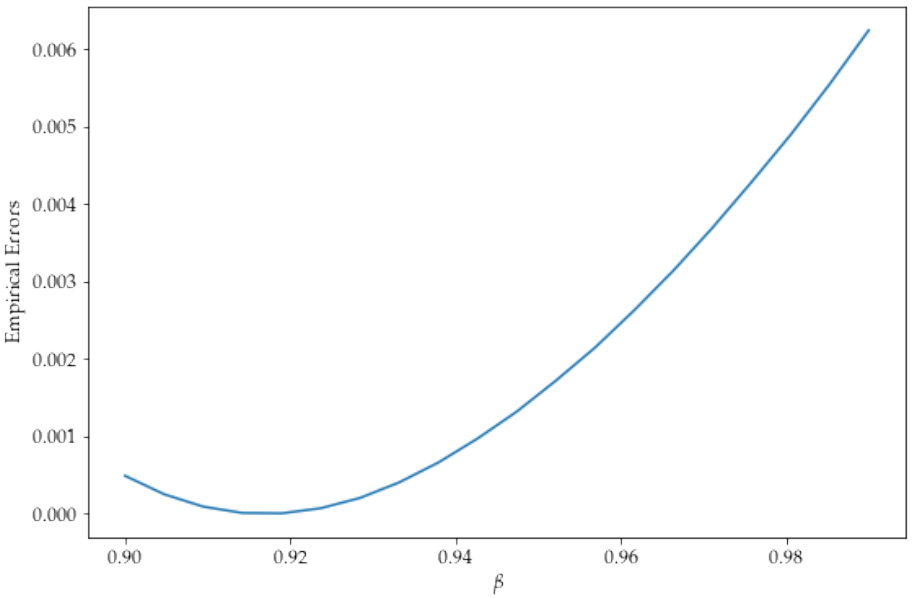

Figure 7: Empirical Errors *v.s.* Range of β with $r_{target} = 1.0925$.

Therefore, it is now ready to calibrate the parameter β with different average interest rates. Here I define an objective function:

$$objective(\beta_{guess},\ r_{target}) := (r_{sim} - r_{target})^2 \tag{9}$$

where r_{target} is the average interest rate I aim to achieve by calibrating the parameter β_{guess}, r_{sim} is the simulation that can be predicted using the neural network model. With this objective function, I can then use the gradient descent approach to find the right parameter β corresponding to the r_{target}.

According to historical records[3], the average annual return for S&P500 index since Jan,1 1871 until Dec,31 2020 is 1.0925. I generate the objective function (empirical errors) with $r_{target} = 1.0925$ and plot it against ranges of β in figure 7. The gradient descent approach (I let it run until the $objective(\beta_{guess},\ r_{target}) < 10^{-5}$) gives us the calibrated $\beta_{guess} = 0.9176$, as shown in figure 8 with the β_{guess} updating process.

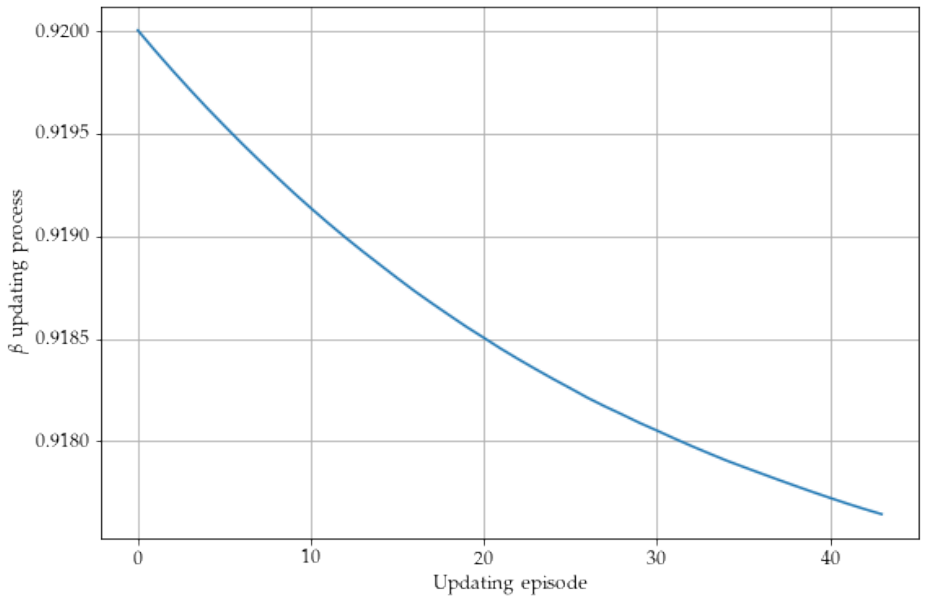

Figure 8: Evolving of β_{guess} using the gradient descent method.

5 Extending Benchmark Model Input with More Parameters

5.1 Input Extension

I extend the Deep Learning algorithm to solve the Brock-Mirman model input space with two economic parameters β and γ at the same time. To do this, I keep the parameterization of the economic parameters for capital share α and TFP η as shown in table 1, while extending the input space to $\mathbf{x} := [z, k, \beta, \gamma]^T \in \mathcal{Z} \times \mathbb{R}^3$, where $z \in \mathcal{Z}$

[3]Calculated using this definition: Compound Annual Growth Rate (See website calculation tool here: http://www.moneychimp.com/features/market_cagr.htm).

denotes the exogenous shock, capital holding k, as well as the discount parameter β and the relative risk aversion parameter γ.

The ranges for the input state variables and the parameters β and γ are set as follows:

Capital holding k	Discount factor β	Relative risk aversion γ
$[0.05,\ 0.8]$	$[0.90,\ 0.99]$	$[0.5,\ 3.0]$

Table 5: Ranges for input endogenous state variable k and parameters β and γ

As in the section 4.1, I sample the input data randomly following uniform distribution with ranges specified above. Moreover, since there is one more input dimension now, the input layer of the deep neural network now consists of $1+4$ input nodes, representing the constant 1, the exogenous shock z, the capital holding k, the discount factor β, and the relative risk aversion parameter γ, respectively. Following the input layer, the neural network still keeps two hidden layers with 500 and 500 relu-activated hidden nodes. The output layer remains 1 node activated with the softplus function, featuring the non-negative predicted capital holding next period k'.

Besides, since the input space contains more dimensions now, the training process is comparatively more "difficult" for the neural network to figure out what the correct policy function is. I decide to let the training run for longer in order to get better results. Therefore, I set the learning rate $\alpha^{\text{learn}} = 1 \times 10^{-4}$ for the first 30,000 episodes and switch to $\alpha^{\text{learn}} = 1 \times 10^{-5}$ for the second 30,000 episodes of training. The chosen hyper-parameters are specified in table 6.

Episodes	Learning rate α^{learn}	Nodes hidden layers	Activations hidden layers	Activation output layer
1 - 30,000	1×10^{-4}	{500, 500}	{relu, relu}	{softplus}
30,000 - 60,000	1×10^{-5}	{500, 500}	{relu, relu}	{softplus}

Table 6: Parameterization of the hyper-parameters.

5.2 Model Performance

I now illustrate the performance of the neural network approximation on the Brock-Mirman model with extended input. It is shown in figure 9 that after 60,000 episodes of training, the loss converges to below 10^{-6}, which implies that the actual scale of loss has decreases to around 1×10^{-3}, resulting in a fairly trustworthy prediction performance.

The relative Euler equation errors after training on 1,000 sampled state variables are summarized in table 7.

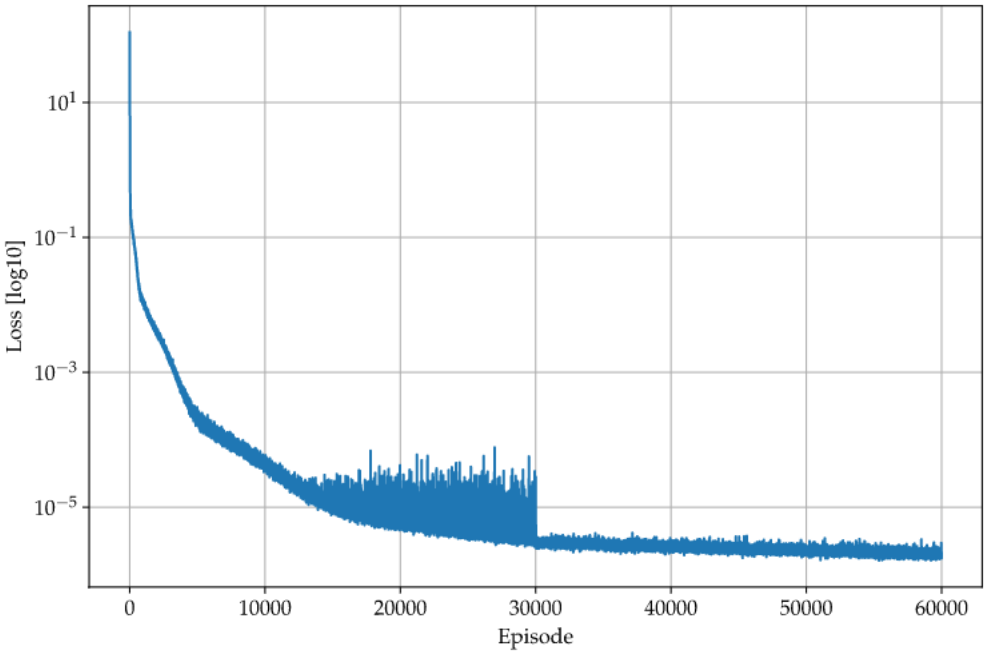

Figure 9: Evolution of the loss function during the first 30,000 episodes of training with learning rate $\alpha^{\text{learn}} = 1 \times 10^{-4}$ and the second 30,000 episodes with learning rate $\alpha^{\text{learn}} = 1 \times 10^{-5}$.

	mean	max	min	10	50	90
Relative Euler equation error (%)	0.069	0.518	0.000	0.000	0.054	0.147

Table 7: Relative Euler equation error. The columns show the mean, the max, and the min errors as well as the 10th, 50th, 90th percentile.

Furthermore, from the figure 10, it is shown that the policy function can be accurately predicted corresponding to different values of γ, keeping $\beta = 0.95$ unchanged. Additionally, in figure 11 one can see that the neural network predicts policy functions similarly well to figure 4, which is a confirmation of the accuracy of the model's training despite the extension of the input space dimension.

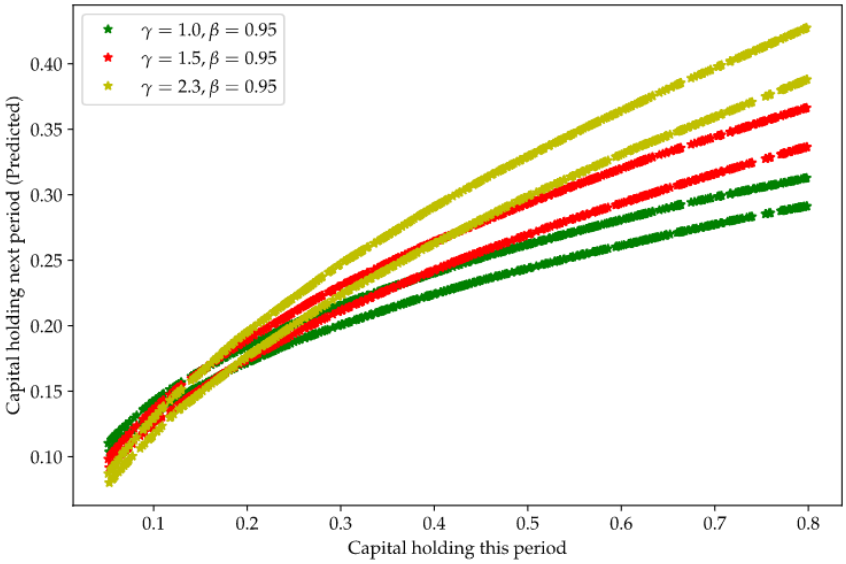

Figure 10: Capital holding this period *v.s.* Capital holding next period (Neural network prediction).

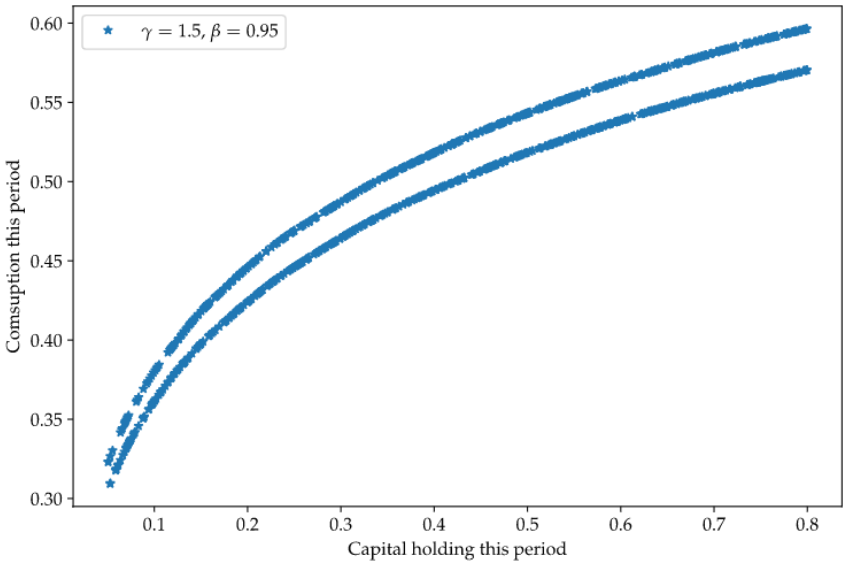

Figure 11: Capital holding this period *v.s.* Consumption this period (Simulated with neural network prediction).

5.3 Parameter Calibration

In this part, I showcase the potential to target two economic indices simultaneously with calibration of the parameters β and γ. The two chosen targets are (i) average interest rate R_t and (ii) relative consumption standard deviation $\sigma(\frac{C_{t+1}}{C_t})$. The reason for choosing these two targets is that, intuitively, the discount factor β captures the mean interest rate R_t while the relative risk aversion parameter γ captures the volatility of relative consumption evolving.

Based on this, I define the modified objective function:

$$\begin{aligned} objective(\beta_{guess},\ \gamma_{guess},\ r_{target},\ \sigma(c)_{target}) := (&\frac{\sigma(c)_{sim} - \sigma(c)_{target}}{\sigma(c)_{target}})^2 \\ &+(\frac{r_{sim} - r_{target}}{r_{target}})^2 \end{aligned} \tag{10}$$

where r_{target} and $\sigma(c)_{target}$ are the two economic targets: average interest rate R_t and relative consumption standard deviation $\sigma(\frac{C_{t+1}}{C_t})$ respectively. Specifically, I use the relative mean squared error of the two targets instead of the absolute error as in definition 9. This is due to the fact that using the absolute error would yield very small (almost indistinguishable) values for the objective function, resulting in challenges for the parameter calibration.

After simulating with 1,000 randomly distributed states with 500 evolving periods, using $\beta = 0.95$ and $\gamma = 1.0$, the two simulated targets are achieved: $r_{target} = 1.0552$, $\sigma(c)_{target} = 0.03396$. By setting these two targets in the objective function, as well as the initial values of $\beta_{guess} = 0.95$ and $\gamma_{guess} \in \big[0.5,\ 1.5\big]$, I present the figure 12 showing the cut-off surface of empirical errors against γ. It is apparent that the minimum point forms when $\gamma = 1.0$, therefore making it possible to calibrate the two parameters β and γ.

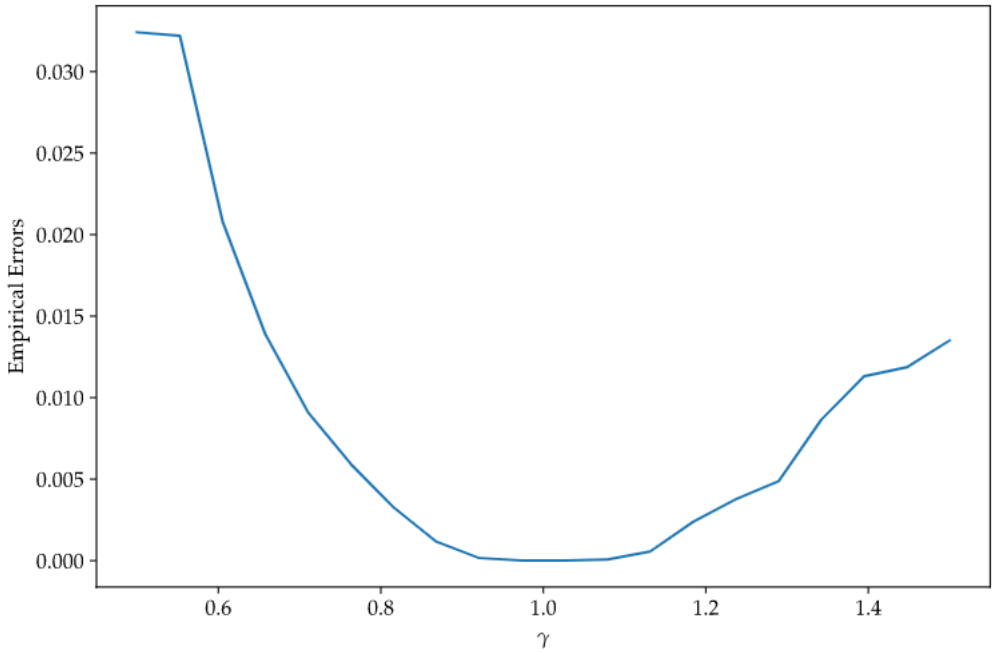

Figure 12: Empirical Errors *v.s.* Ranges of γ with $r_{target} = 1.0552$ and $\sigma(c)_{target} = 0.03396$, setting $\beta = 0.95$.

Following the same parameter calibration methodology as in section 4.5, I aim to use the gradient descent approach to find the correct parameters β and γ corresponding to the r_{target} and $\sigma(c)_{target}$. However, since it is an optimization problem with multiple objectives, the basic gradient descent approach has proven to be challenging for finding the minimal error value. Therefore, I utilize the Adam approach here as well to update the two parameters β and γ.

The parameter's updating evolution is shown in figure 13. We can see that after some volatile updating, the calibration algorithm can find the approximately correct values of the parameters $\beta = 0.946$ and $\gamma = 1.030$, which, given that the algorithm needs to pin down two targets at the same time, are quite close to the actual simulated values of $\beta = 0.95$ and $\gamma = 1.0$.

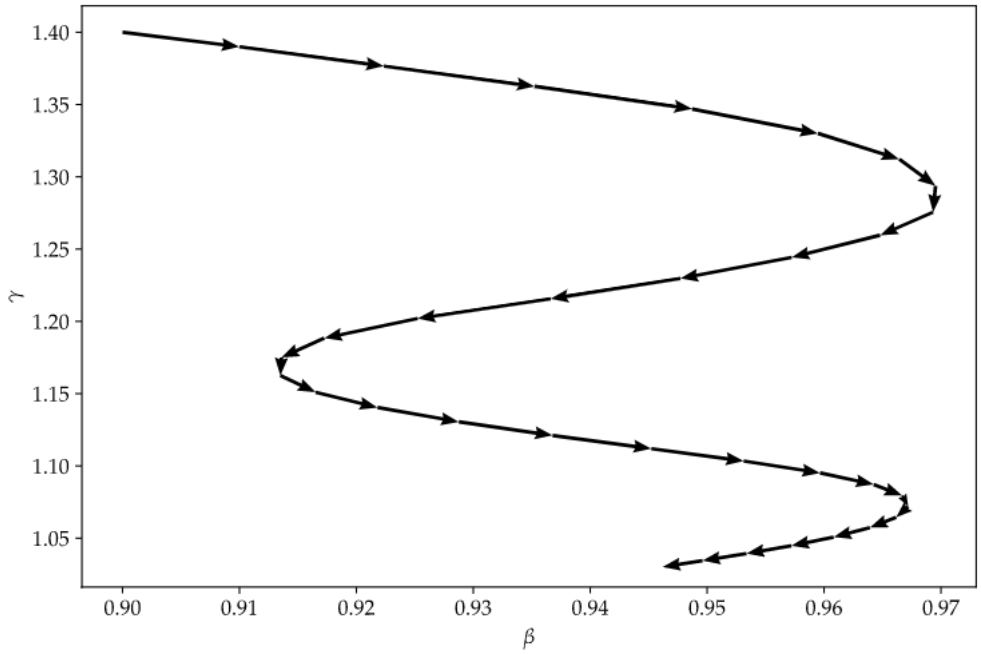

Figure 13: Parameter β and γ updating process, with initial values of $\beta = 0.9$ and $\gamma = 1.4$, aiming to find $\beta = 0.95$ and $\gamma = 1.0$.

Additionally, along with the parameter calibration, it is also convenient to construct the two target indices individually as a function of β and γ. The two functions are plotted in figure 14 and 15 as two 3D plots.

These two figures nicely correspond to the previous assumptions that the discount factor β captures the variation of the average interest rate, while the relative risk aversion parameter γ captures the variation of the volatility of the relative consumption. It is also economically interpretable that the discount factor β and the average interest rate R_t are negatively correlated, since intuitively interest rate should be the inverse of the discount factor β. Also, for relative risk aversion parameter γ, it is negatively correlated with the standard deviation of relative consumption $\sigma(\frac{C_{t+1}}{C_t})$ because this considers the fact that the more risk-averse one is, the more willingly one would like to smooth out their lifetime consumption choices, leading to less volatile relative consumption ratios, which is captured by $\sigma(\frac{C_{t+1}}{C_t})$.

Nevertheless, we can see that the function surfaces in the two figures are not perfectly smooth. I think this is possibly due to the fact that the average interest rate and the standard deviation of relative consumption are both subject to the simulation shocks generation randomness. Additionally, simulating more periods may help with the against the "waves" on the function surfaces.

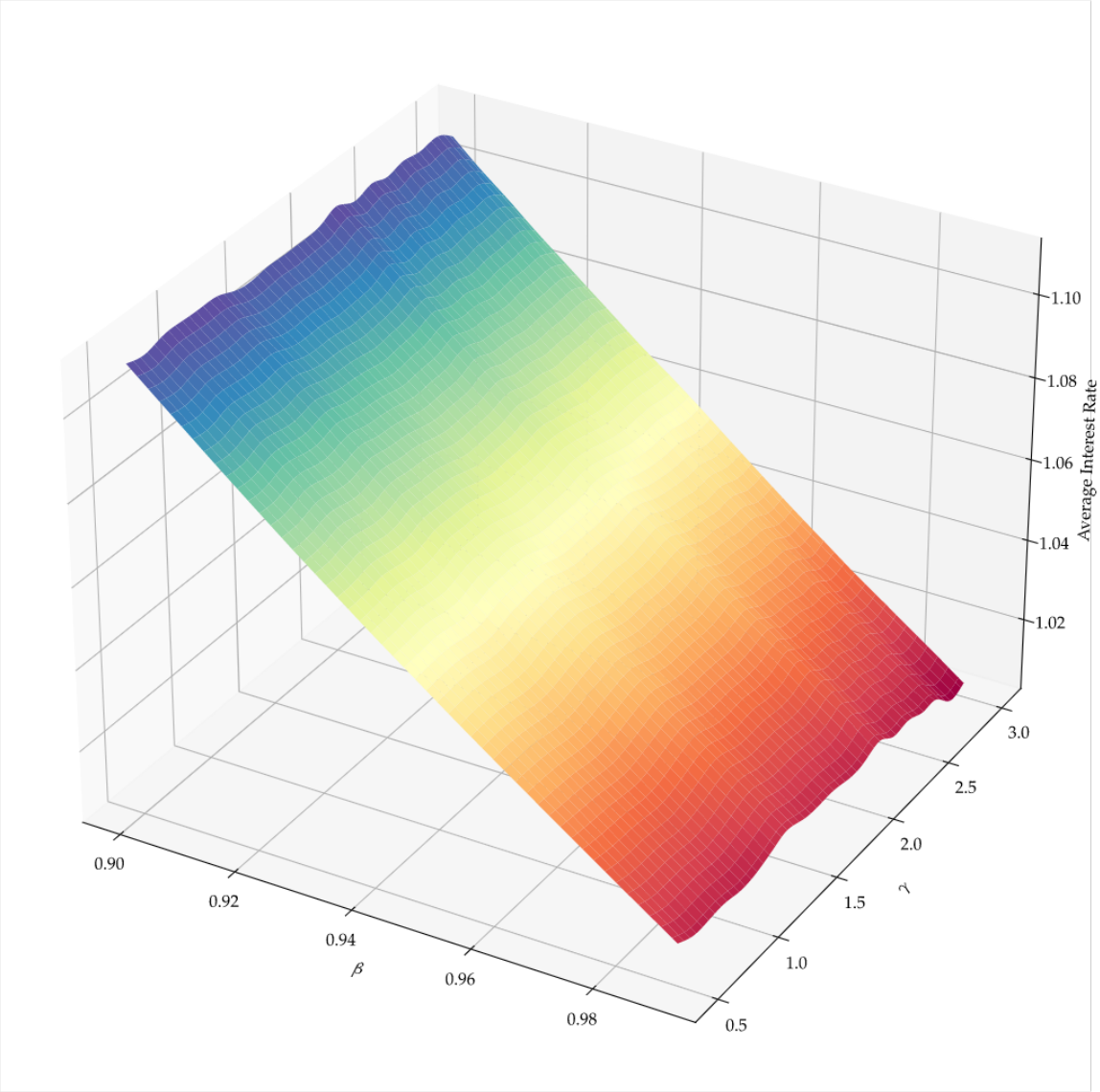

Figure 14: Average interest rate R_t as function of β and γ.

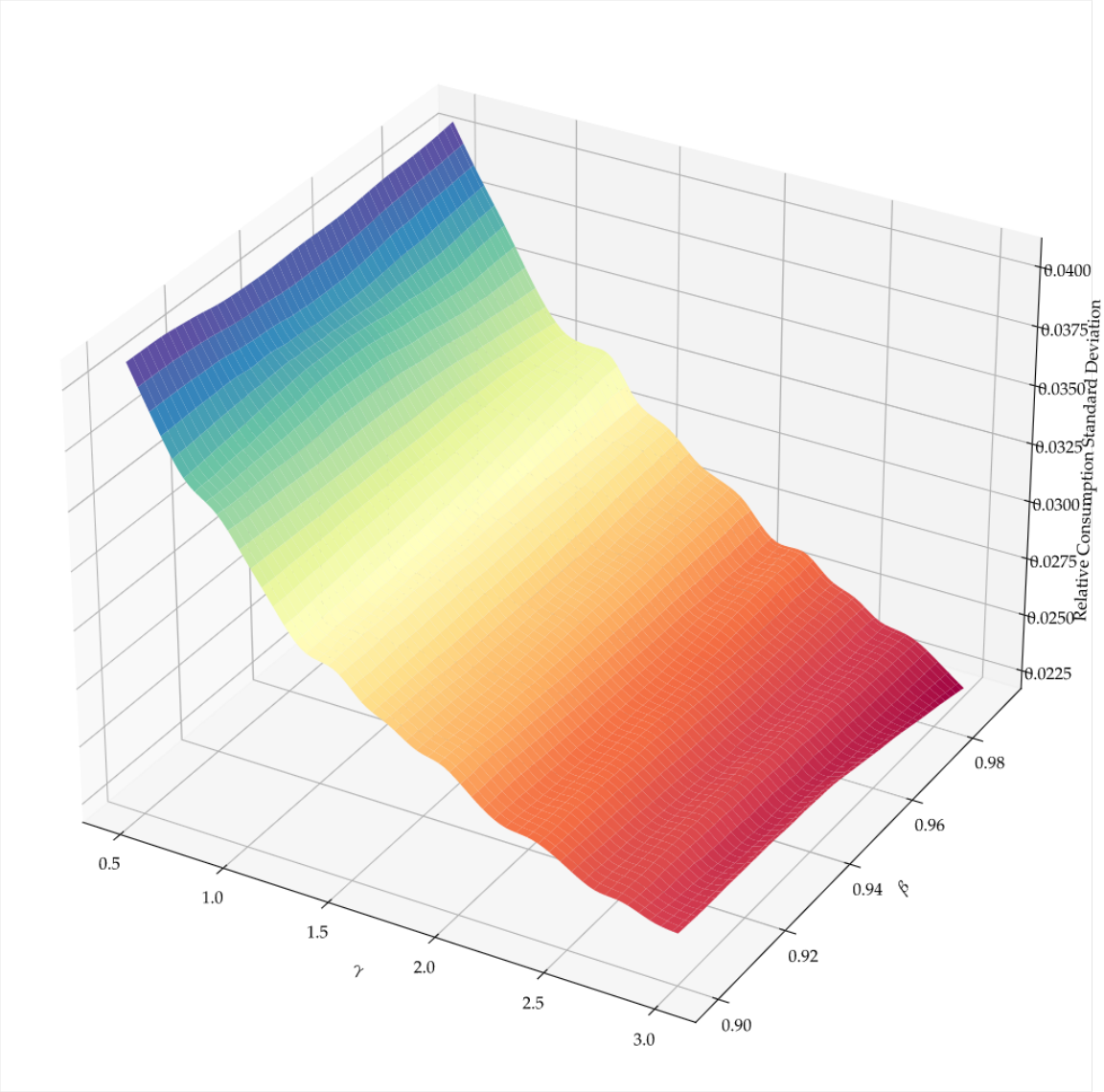

Figure 15: Relative consumption standard deviation $\sigma(\frac{C_{t+1}}{C_t})$ as function of β and γ.

6 A Model Featuring Incomplete Markets with Idiosyncratic Shocks

6.1 Model Description

After solving the benchmark Brock-Mirman model, I aim to solve a more complicated model featuring incomplete markets as well as idiosyncratic shocks. The model I am

using is a simplified version inspired by Heaton and Lucas (1996)[7]. This model encompasses many ingredients that appear in recent macroeconomic studies, such as incomplete markets, occasionally binding constraint, non-stationary shock process, and asset pricing with non-trivial market-clearing conditions. Therefore, I think it is a good model to solve after the benchmark model.

It is an incomplete-markets model with two representative agents $i \in \mathcal{I} = \{1, 2\}$ who trade in bond b. b_t^1 and b_t^2 denote the bond holding for the first and second agent at the current period, respectively. p_t^b denotes the bond price at time t. The aggregated state $z \in \mathcal{Z}$, which consists of agents' income h_t^i and aggregate endowment Y_t^a, follows a first-order Markov process.

Agent i takes the bond price as given and maximizes their inter-temporal expected utility:

$$\max \ \mathbb{E}_t \left[\sum_{n=0}^{\infty} \beta^n \frac{(c_{t+n}^i)^{1-\gamma}}{1-\gamma} \right] \tag{11}$$

s.t.

$$c_t^i + p_t^b b_t^i \leq b_t^i + h_t^i$$
$$b_{t+1}^i \geq K_t^b$$

where h_t^i denotes the agent's income, and $Y_t^a = h_t^1 + h_t^2$ denotes the aggregate endowment. The borrowing limit is set to be a constant fraction of the worst-case yearly income, i.e., $K_t^b = \bar{K}^b h_t^i(z_{worst})$.

In equilibrium, prices are determined such that markets clear in each shock history:

$$b_t^1 + b_t^2 = 0 \tag{12}$$

Therefore, the Euler Equation for agent i is written as follows:

$$-{c_t^i}^{-\gamma} p_t^b + \beta \, \mathbb{E} \left[{c_{t+1}^i}^{-\gamma} \right] + \mu_t^i = 0, i \in \mathcal{I} = \{1, 2\} \tag{13}$$

with the Karush-Kuhn-Tucker (KKT) conditions:

$$\mu_t^i (b_{t+1}^i - K_t^b) = 0 \tag{14}$$
$$\mu_t^i \geq 0$$

6.2 Data Sampling

Next, following the same data generation process, I use TensorFlow 2.0 to sample the state variables following specified distribution. The parameterization of the model is set in table 8:

Here, the idiosyncratic shock $z \in \mathcal{Z}$ takes four values $\{0, 1, 2, 3\}$, featuring four different situations where the representative agents are facing stochastic income shocks.

Relative risk aversion γ	Aggregate endowment Y_t^a	Per capita income h_t^1
1	{7.00, 6.65}	{3.75, 3.25, 3.60, 3.05}

Table 8: Parameterization of the economic parameters.

To simplify the problem and focus on the model solving with Deep Learning, I set the transaction matrix to be:

$$\Pi = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} \tag{15}$$

So that the evolution of the idiosyncratic shocks are set to be i.i.d., making it easier to implement for the time being.

Moreover, the ranges for the input variables are presented as followed:

Bond holding b_t^1	Discount factor β
$[-1.525,\ 1.525]$	$[0.90,\ 0.99]$

Table 9: Ranges for input endogenous state variable b_t^1 and parameter β.

Here, I set the borrowing limits to be half of the worst-case yearly income for one representative agent, which corresponds to the model setting that $K_t^b = \bar{K}^b h_t^i(z_3)$.

The neural network is to approximate the equilibrium function $\boldsymbol{\theta}_{b,p^b,\mu} : \mathcal{Z} \times \mathbb{R}^6 \to \mathbb{R}^4$ capturing the bond holding policy functions as well as the equilibrium price, such that for all states $\mathbf{x} := [z, Y^a, h^1, h^2, b^1, b^2, \beta]^T \in \mathcal{Z} \times \mathbb{R}^6$, where $z \in \mathcal{Z}$ denotes the exogenous shock, Y^a denotes the aggregate endowment, $h^i, i \in \{1, 2\}$ denotes the per capita yearly income for each agent, $b^i, i \in \{1, 2\}$ denotes the bond holding for the current period, as well as the discount parameter β, the output $\mathcal{N}_{\boldsymbol{\rho}}(\mathbf{x}) := [b_{next}^1, p^b, \mu^1, \mu^2]^T$ where b_{next}^1 denotes the bond holding next period, p^b denotes the equilibrium price for bond, and $\mu^i, i \in \{1, 2\}$ the KKT multipliers for each agent, the Euler Equation 13 and the KKT conditions 14 will always hold.

The training process of the algorithm takes longer than the previous ones to converge to a small enough loss, which is reasonable considering the enhanced complexity. Specifically, I set the learning rate $\alpha^{\text{learn}} = 1 \times 10^{-4}$ for the first 50,000 episodes, and switch to $\alpha^{\text{learn}} = 1 \times 10^{-5}$ for the second 50,000 episodes.

Noticing that the output $[b_{next}^1, p^b, \mu^1, \mu^2]^T$, where p^b, μ^1 and μ^2 are non-negative values, while b_{next}^1 is to be controlled within range $[-1.525, 1.525]$, I use softplus activation functions for the first three nodes and sigmoid activation function modification for the last one to ensure the prediction of the neural network is economically interpretable.

Episodes	Learning rate α^{learn}	Nodes hidden layers	Activations hidden layers	Activation output layer
1 - 50,000	1×10^{-4}	{300, 300}	{relu, relu}	{softplus, sigmoid}
50,000 - 100,000	1×10^{-5}	{300, 300}	{relu, relu}	{softplus, sigmoid}

Table 10: Parameterization of the hyper-parameters.

6.3 Model Performance

Next, I summarize the neural network training performance on the new model with idiosyncratic shocks. It can be seen from figure 16 that after the first 40,000 training episodes, the loss has already converged to around 10^{-6}, with very large volatility. After switching the learning rate to $\alpha^{\text{learn}} = 1 \times 10^{-5}$, the loss remains steadily below 10^{-6}.

The Euler equation errors as well as KKT condition errors after training on 1,000 sampled state variables are summarized in table 11.

	mean	max	min	10	50	90
Euler equation error agent 1 ($\times 10^{-2}$)	0.016	0.140	0.000	0.000	0.013	0.033
Euler equation error agent 2 ($\times 10^{-2}$)	0.016	0.111	0.000	0.000	0.013	0.033
KKT condition error agent 1 ($\times 10^{-2}$)	0.000	0.040	0.000	0.000	0.000	0.019
KKT condition error agent 2 ($\times 10^{-2}$)	0.000	0.097	0.000	0.000	0.000	0.016

Table 11: Euler equation errors and KKT condition errors. The columns show the mean, the max, and the min errors as well as the 10th, 50th, 90th percentile.

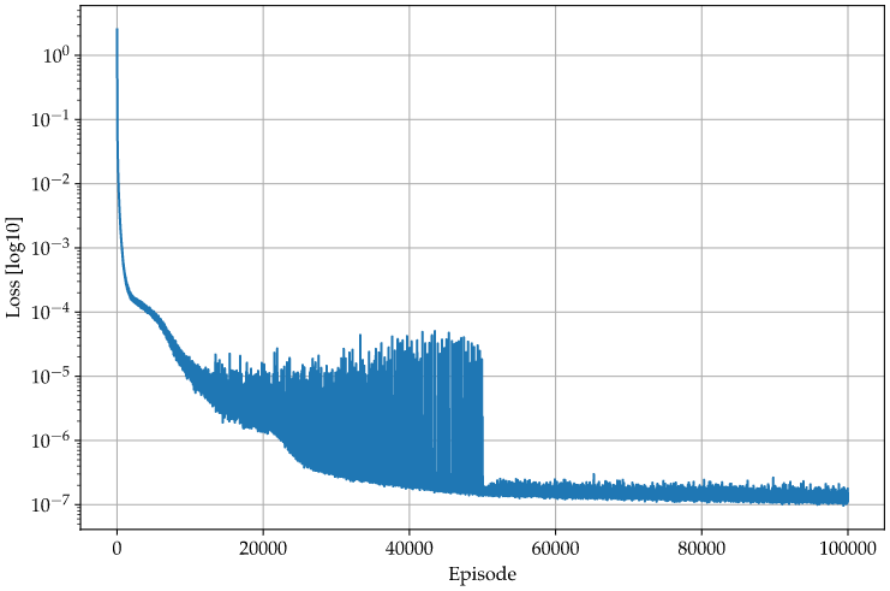

Figure 16: Evolution of the loss function during the first 50,000 episodes of training with learning rate $\alpha^{\text{learn}} = 1 \times 10^{-4}$ and the second 50,000 episodes with learning rate $\alpha^{\text{learn}} = 1 \times 10^{-5}$.

To see how the neural network's prediction evaluates the solution to this model, I plot neural network output figure 17, 18, 19, and 20. These figures serve as a good verification of the prediction accuracy.

Figure 17 presents the policy function $b_t^1 \to b_{t+1}^1$, which predicts that the bond holding next period is almost linearly related to bond holding this period with two different intercept values. More specifically, in the two "good" aggregate conditions where the agent's per capita income $h^1 \in \{3.75, 3.6\}$, the bond holding next period has larger values whereas in the two "bad" conditions where $h^1 \in \{3.25, 3.05\}$, the values for the bond holding next period will be smaller. This corresponds to the intuition that, if the realization of the shock today is positive, then one would have incentive to hold more assets in the future but not borrowing them, and vice versa.

Figure 18 shows the equilibrium bond prices with each shock. When the aggregated endowment $Y^a = 7.0$, the equilibrium bond prices are around 0.975, whereas when $Y^a = 6.65$, the equilibrium bond prices drop to below 0.94. Also, at the two ends of the b^1 limits, the bond prices go up with the spikes, which corresponds to the occasionally binding borrowing constraints.

Figure 19 and figure 20 show the prediction of KKT multipliers for both agents. According to these two figures, the KKT multipliers for agent 1 and agent 2 are perfectly symmetrical, due to the market clearing conditions. Besides, for agent 1, the borrowing constraint is mostly binding when the idiosyncratic shock $z \in \{1, 3\}$ realizes, which means the per capita income $h^1 \in \{3.25, 3.05\}$, namely, the "bad" conditions for agent

1. Therefore, they would be willing to borrow more bond assets while they would encounter the borrowing constraints. The "worse" the situation is, the more binding the borrowing constraints are.

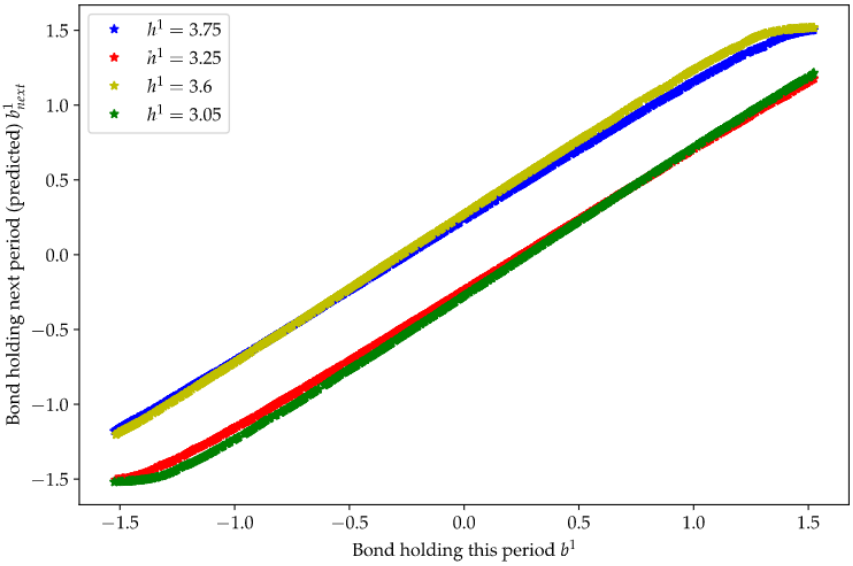

Figure 17: Bond holding this period b^1 *v.s.* Bond holding next period (Neural network prediction) b^1_{next}.

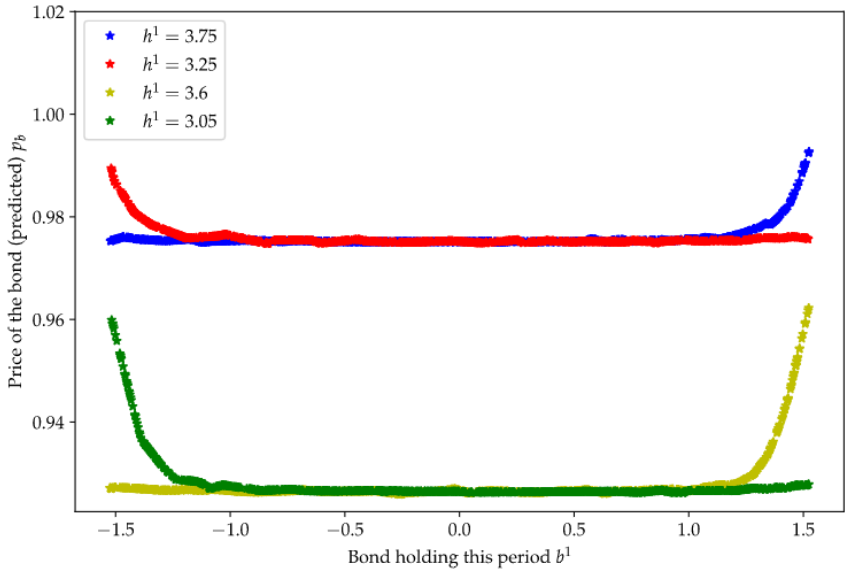

Figure 18: Bond holding this period b^1 *v.s.* Equilibrium bond price (Neural network prediction) p^b.

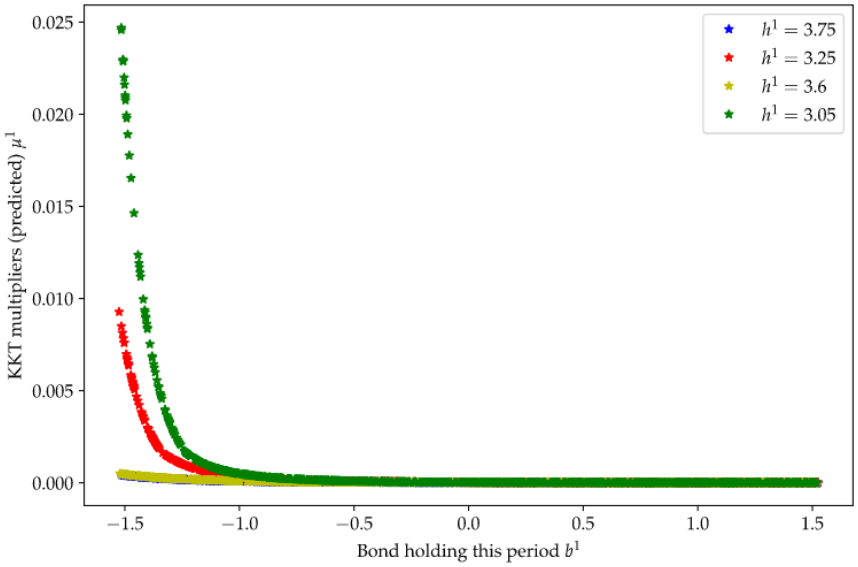

Figure 19: Bond holding this period b^1 *v.s.* KKT multiplier for agent 1 (Neural network prediction) μ^1.

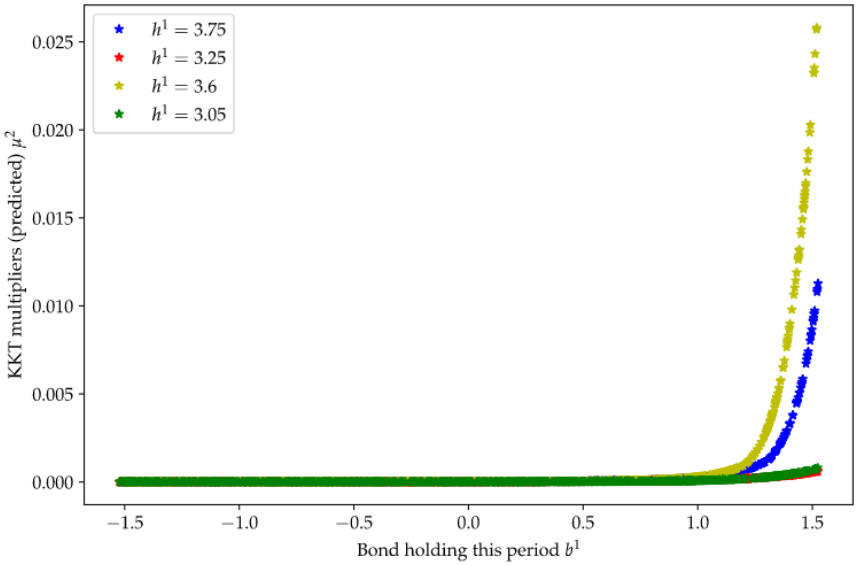

Figure 20: Bond holding this period b^1 *v.s.* KKT multiplier for agent 2 (Neural network prediction) μ^2.

Furthermore, figure 21, 22, 23, and 24 evaluate the Euler Equation errors as well as the KKT condition errors for each of the shocks generated by the neural network. The Euler Equation errors are mainly below 5×10^{-3}, and the KKT condition errors are all

below 2×10^{-3}. The reached accuracy demonstrates that the implemented method can compute satisfactory approximations.

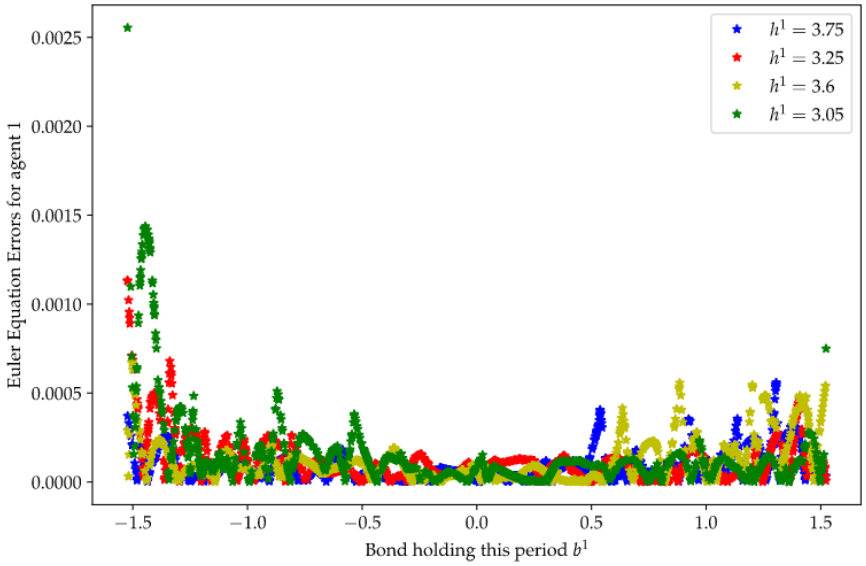

Figure 21: Bond holding this period b^1 *v.s.* Euler Equation error for agent 1.

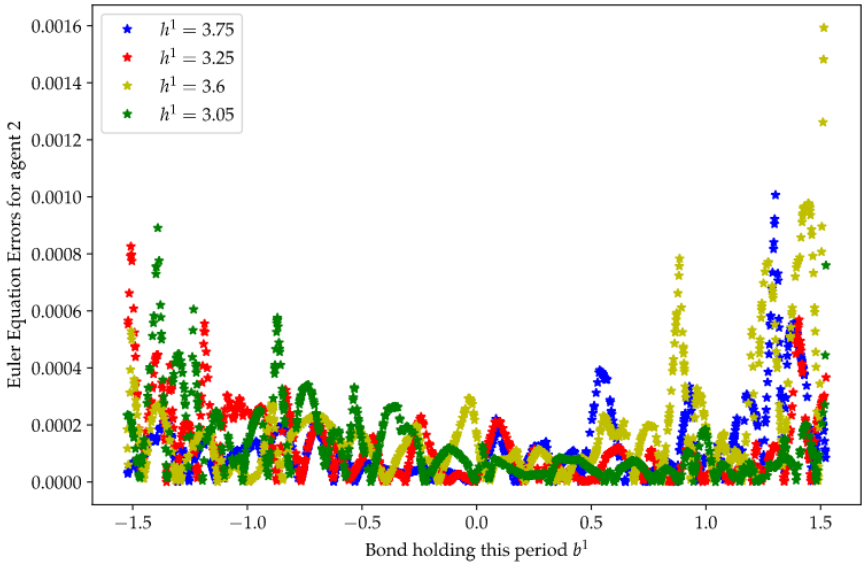

Figure 22: Bond holding this period b^1 *v.s.* Euler Equation error for agent 2.

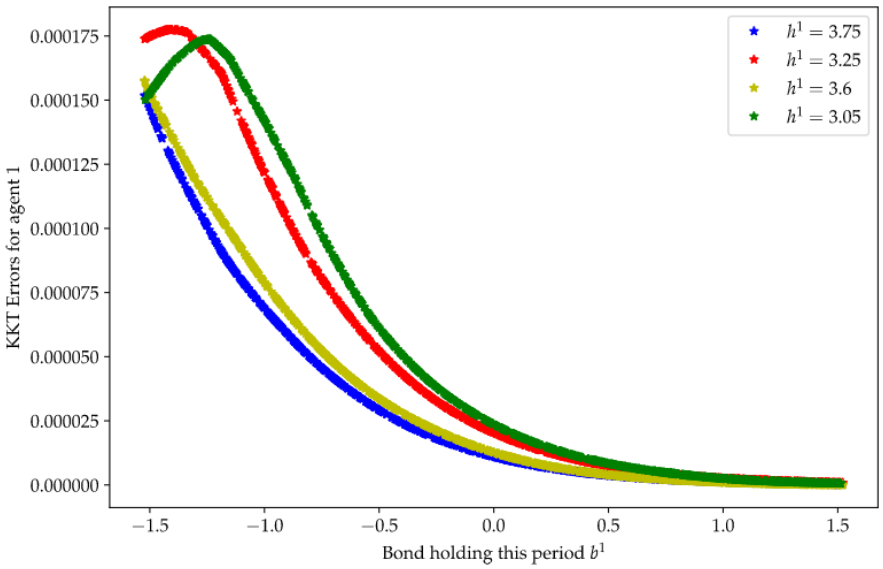

Figure 23: Bond holding this period b^1 *v.s.* KKT error for agent 1.

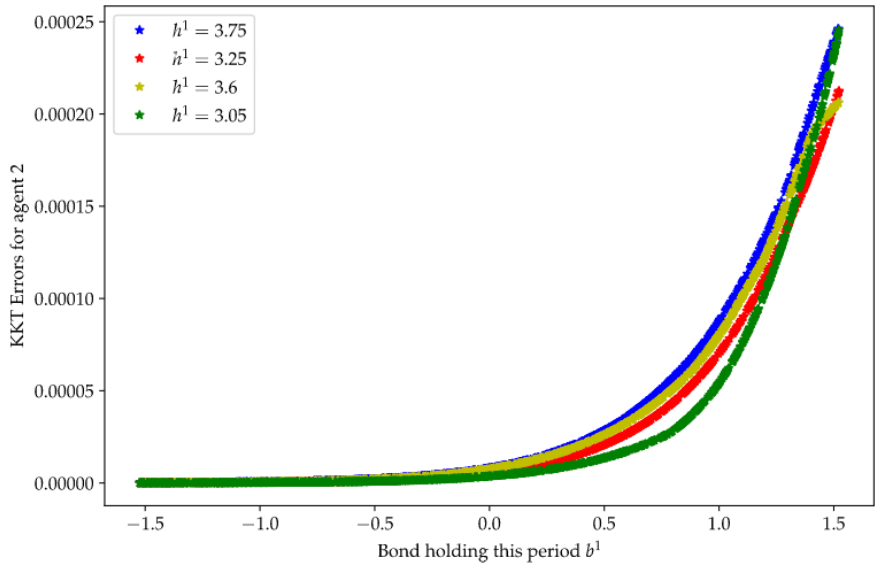

Figure 24: Bond holding this period b^1 *v.s.* KKT error for agent 2.

6.4 Parameter Calibration

I illustrate the parameter β calibration for this model to match real-world data. Similarly to section 4.5, I simulate 1,000 different states with 500 evolving periods, then compute the average interest rate $r = \frac{1}{p^b}$ predicted by the neural network model.

The figure 25 shows the simulated average interest rate versus the range of β, which is very similar to figure 6. The discount factor β and the average interest rate are negatively correlated.

To calibrate β with the appropriate interest rate which reflects the nature of the inverse relationship with the bond price, I decide to use the U.S. Treasury Yield Curve Rates with 30-year maturity[4], which is 2.17% at the time of writing. Therefore, I plot figure 26, confirming that the error function is perfectly convex and ready to be calibrated with parameter β.

Using the same structure of the objective function 9 previously defined, I set the initial value of $\beta_{guess} = 0.92$ and use the gradient descent approach to update the new parameter value until the objective function reaches below $1 \times 10^{-}5$. The result is shown in figure 27, which indicates that the true β value should be 0.978.

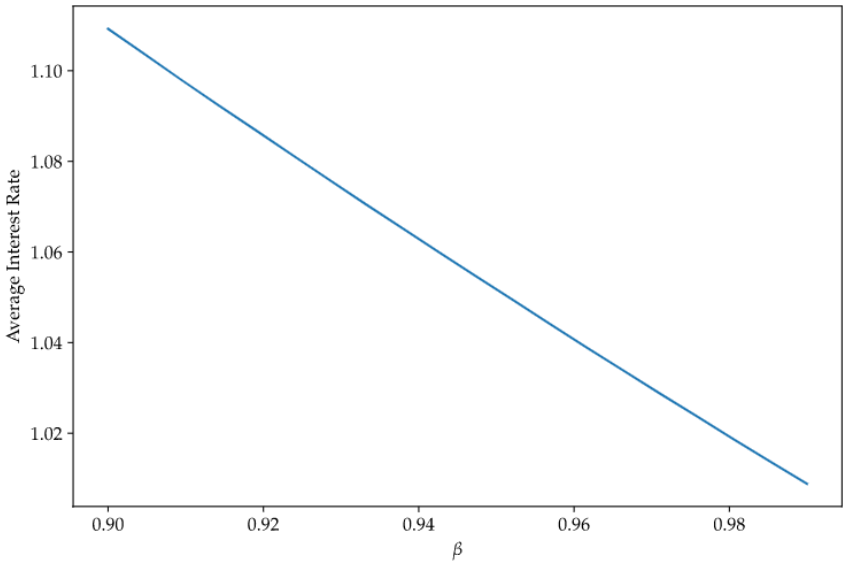

Figure 25: Average Interest Rate (Simulated with the prediction) *v.s.* Range of β.

[4]The Daily Treasury Yield Curve Rates can be found on the official website of U.S. Department of the Treasury. (See e.g. https://www.treasury.gov/resource-center/data-chart-center/interest-rates/pages/textview.aspx?data=yield)

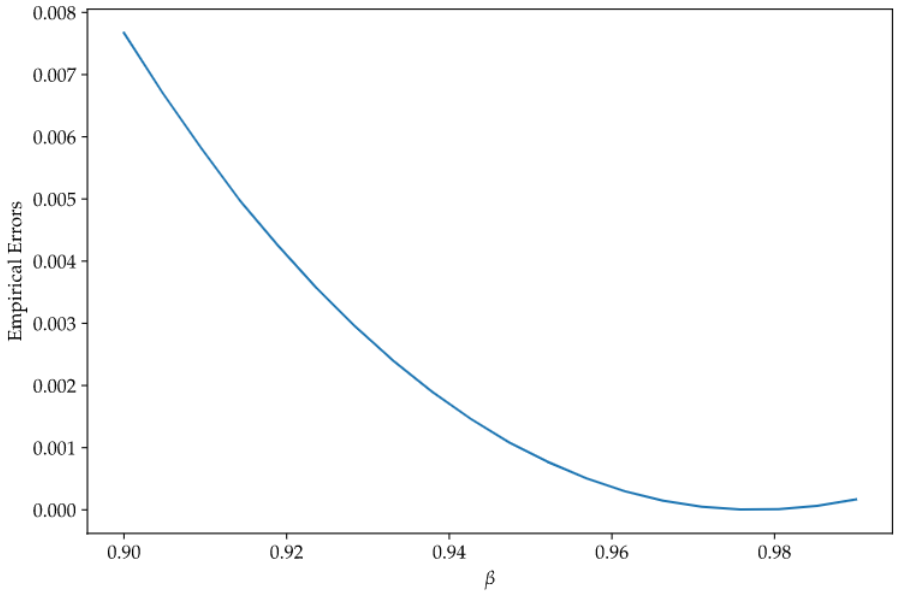

Figure 26: Empirical Errors $v.s.$ Range of β with $r_{target} = 1.0217$.

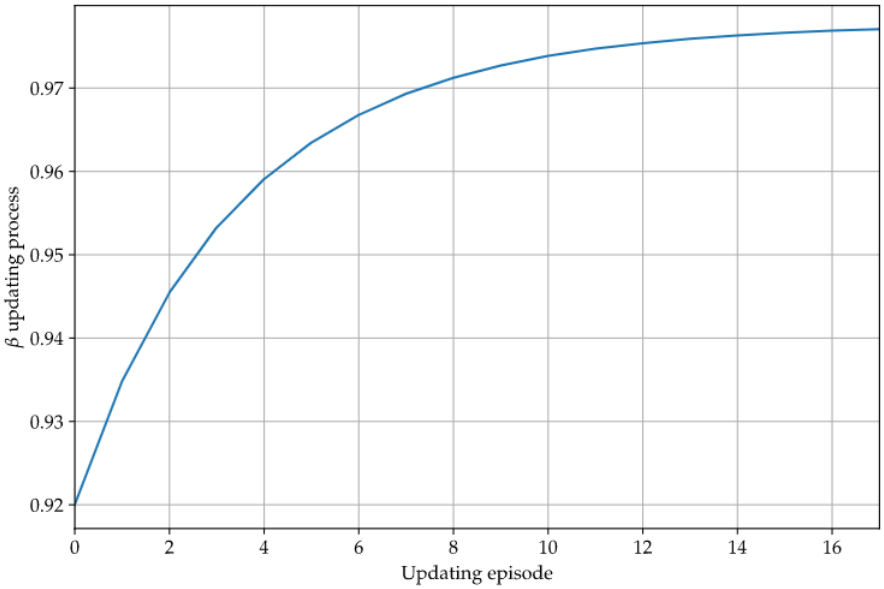

Figure 27: Evolution of β_{guess} using gradient descent approach.

7 Practical issues

7.1 Loss Function Implementation

The inverse of relative error While trying to solve the benchmark model with the extension of more parameters described in section 5.1, it has proven challenging to implement the normal relative consumption error $e_{\mathbf{x}}^{\mathrm{REE_{cap}}}(\boldsymbol{\rho})$ as described in equation 6 within the loss function construction. I think it is mainly due to the fact that the division calculation is handled poorly by the computer when the predicted C_t values, as the denominators, are sometimes not distinguishable from 0. Hence, the loss function tends to have very large spikes during the training process and is not able to find the correct policy function solutions. As shown in figure 28, the loss function cannot converge during the entire training process compared to the nicely converging loss function in figure 9. Also, as shown in figure 29, the neural network without doing the inverse implementation is not able to predict reasonably accurate policy functions as opposed to figure 11.

The solution I find helpful to overcome this issue is using the inverse of the relative consumption error, which is then defined as:

$$e_{\mathbf{x}}^{\mathrm{REE_{cap}}}(\boldsymbol{\rho}) := \frac{C_t}{\beta\,\mathbb{E}_t\left[\alpha A_{t+1} K_{t+1}^{\alpha-1} {C_{t+1}}^{-\gamma}\right]^{-\frac{1}{\gamma}}} - 1. \tag{16}$$

Naturally, this modification avoids the division calculation since it gets rid of it by just using multiplications while retaining the ability to capture the quantities of relative consumption errors.

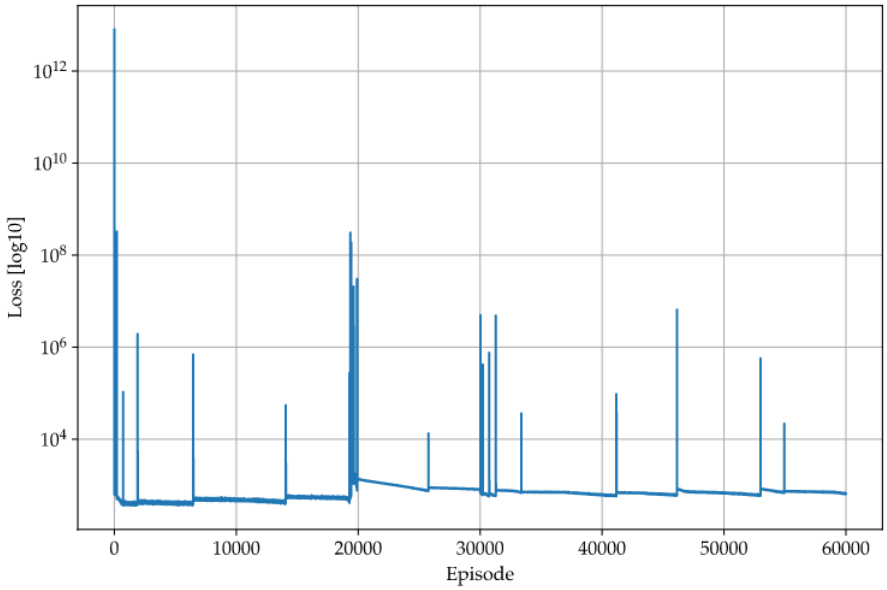

Figure 28: Evolution of the loss function with the same hyper-parameters as described in Table 6. See comparable loss function evolution figure 9.

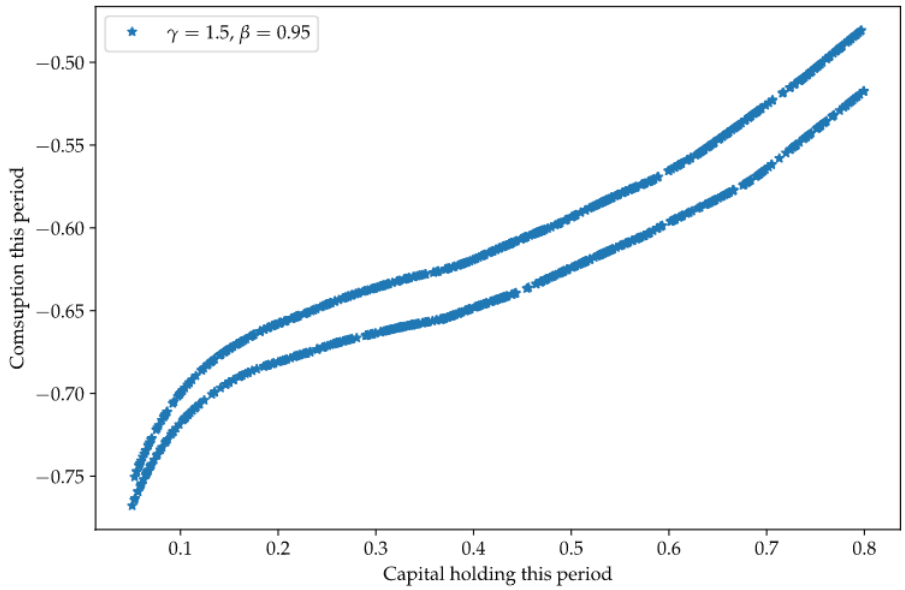

Figure 29: Capital holding this period *v.s.* Consumption this period (Simulated with neural network prediction).

7.2 Redundant Information for Neural Networks

Adding more information as input variables During the process of solving the model featuring incomplete markets with idiosyncratic shocks described in section 6.1, one finding is that adding more information as the input variables to feed neural networks will help improve the performance of the algorithm.

More concretely, before passing redundant information to the input space, the dimension of the input space $\mathbf{x} := [z, Y^a, h^1, b^1, \beta]^T$. Afterwards, during loss function construction I explicitly set $h^2 = Y^a - h^1$ as well as $b^2 = 0 - b^1$. Theoretically, this approach should be equivalent to setting the input space as $\mathbf{x} := [z, Y^a, h^1, h^2, b^1, b^2, \beta]^T$ while clarifying the relationships between h^1, h^2, b^1, and b^2 before passing them to the input space. However, empirically using the latter approach obtains a better performance of neural networks. As mentioned in the paper of Azinovic, Gaegauf, and Scheidegger (2019)[1], passing redundant information to the neural networks, increasing the dimension of the input space stabilizes the learning process.

As shown in figure 30, the loss function without adding h^2 and b^2 gets stuck between 10^{-5} and 10^{-6}, without the tendency to decrease, whereas in figure 16 it successfully decreases to below 10^{-6}. Additionally, it is shown in figure 31 that without increasing the input dimension, the neural network predicts very small KKT multipliers for agent 2, while predicting similar KKT multipliers for agent 1 correctly. This serves as a confirmation of the importance of passing redundant information as input variables.

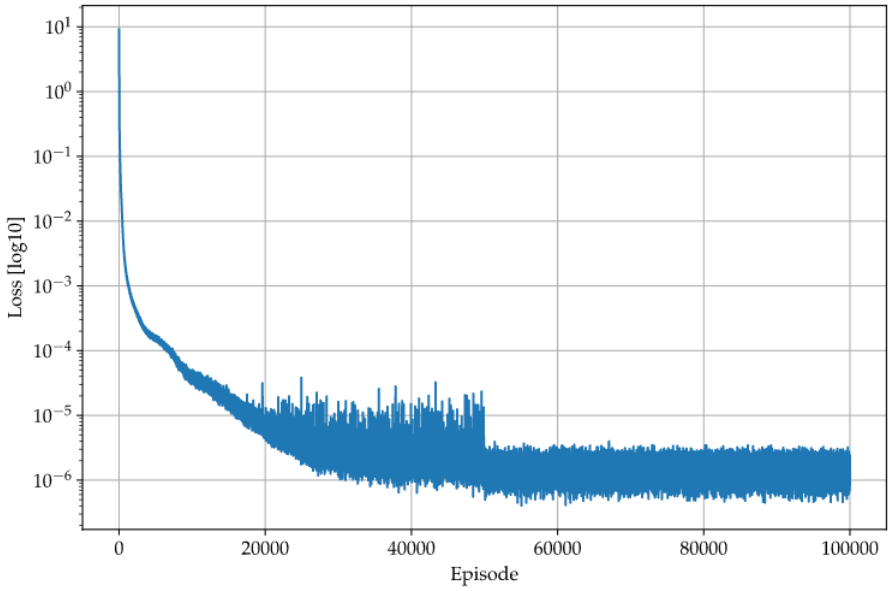

Figure 30: Evolution of the loss function with the same hyper-parameters as described in Table 10. See comparable loss function evolution figure 16.

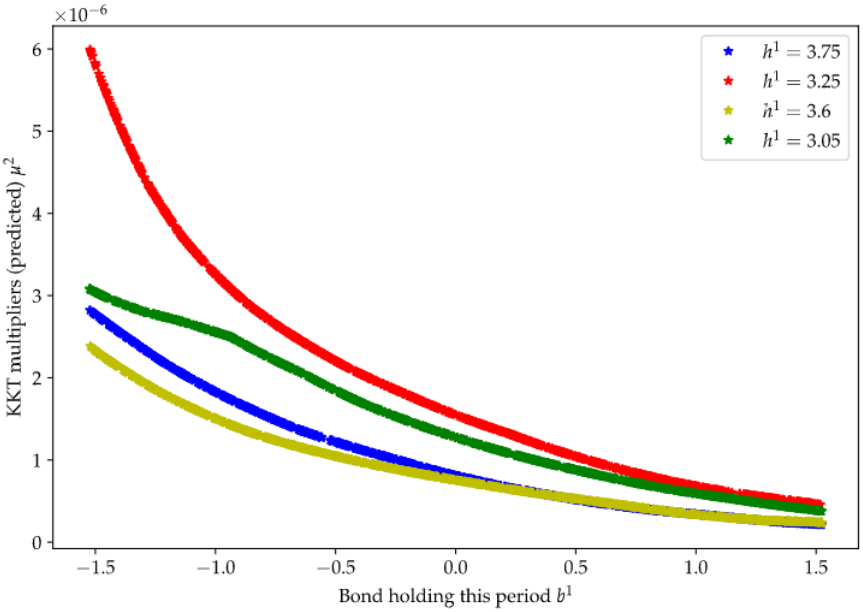

Figure 31: Prediction of the KKT multiplier for agent 2 μ^2.

Redundant equilibrium conditions While constructing the loss function for model with idiosyncratic shocks, I add redundant information about the KKT multiplier conditions for next period as well. Concretely, I include the KKT conditions for the current period:

$$\mu_t^i(b_{t+1}^i - K_t^b) = 0 \tag{17}$$

as well as the KKT conditions for next period together to the loss function:

$$\mu_{t+1}^i(b_{t+2}^i - K_{t+1}^b) = 0 \tag{18}$$

However, as figure 32 shows, the empirical loss also converges to below 1×10^{-6}, which implies the implementation is probability not that necessary. Comparing the loss function figures 32 and 16, it can be seen that within the same amount of episodes of training and hyper-parameter setting, adding redundant equilibrium conditions results in slightly larger errors, even though both cases have achieved satisfactory approximations. This indicates that adding redundant equilibrium conditions to the cost function does not play as important role as adding redundant information explicitly to the input space.

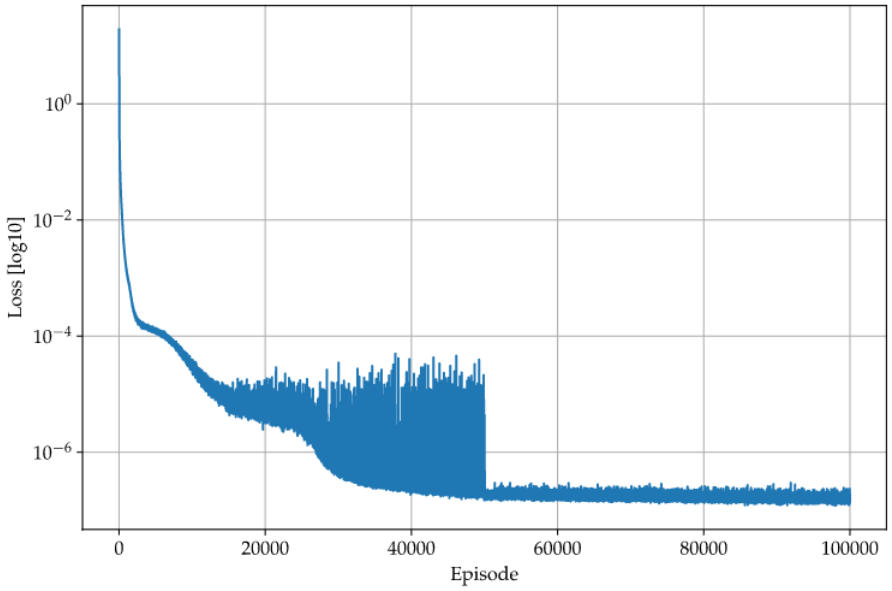

Figure 32: Evolution of the loss function with the same hyper-parameters as described in Table 10. See comparable loss function evolution figure 16.

8 Conclusion

In this thesis, I explore the possibility to solve the model from Brock and Mirman (1972)[4] as well as the simplified version of model from Heaton and Lucas (1996)[7] using deep learning with expanding of the input to ranges of model parameters within the context of TensorFlow 2.0. Additionally, I successfully accomplish the parameter calibration with the interaction of the real-world data, the average annual return of S&P500 index and the U.S. Treasury Yield Curve Rates. Furthermore, I investigate the relationships between different economic indices and economic model parameters.

Concretely, I solve the two models mentioned above using the method of minimizing Euler equation residuals on 1,000 sampled state variables and stochastic shocks, as well as ranges of economic parameters. The first model serves as a benchmark since it can already be solved analytically, and I showcase that, using deep learning, a satisfactory solution approximation can be achieved. The second model is more complicated, featuring incomplete markets, occasionally binding constraint, non-stationary shock process and asset pricing with non-trivial market-clearing conditions. Additionally, adding the economic parameters into the input space provides the possibility of parameter calibration, which will contribute to the economic policy construction, economic decision-making, etc.

My thesis serves as a possible starting point on further research on solving dynamic economic models with the extension of ranges of economic parameters as well as param-

eter calibration matching with real-world data. The future possible research directions include: exploring additional parameter's extension into neural networks, deeper understanding of how and why neural networks can overcome the "curse of dimensionality", and hyper-parameter tuning as well as neural network architecture performance comparison.

9 References

[1] Marlon Azinovic, Luca Gaegauf, and Simon Scheidegger. Deep equilibrium nets. *Available at SSRN 3393482*, 2019.

[2] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep optimal stopping. *Journal of Machine Learning Research*, 20:74, 2019.

[3] Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

[4] William A Brock and Leonard J Mirman. Optimal economic growth and uncertainty: The discounted case. *Journal OF Economic Theory*, 4:479–533, 1972.

[5] Victor Fonseca Duarte. Gradient-based structural estimation. Working paper, 2018.

[6] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

[7] John Heaton and Deborah J Lucas. Evaluating the effects of incomplete markets on risk sharing and asset pricing. *Journal of political Economy*, 104(3):443–487, 1996.

[8] Kenneth L Judd. Projection methods for solving aggregate growth models. *Journal of Economic theory*, 58(2):410–452, 1992.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Dirk Krueger and Felix Kubler. Computing equilibrium in olg models with stochastic production. *Journal of Economic Dynamics and Control*, 28(7):1411–1436, 2004.

[11] Michael JP Magill. A local analysis of n-sector capital accumulation under uncertainty. *Journal of Economic Theory*, 15(1):211–219, 1977.

[12] Lilia Maliar, Serguei Maliar, and Pablo Winant. Will artificial intelligence replace computational economists any time soon? 2019.

[13] Simon Scheidegger and Ilias Bilionis. Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science*, 33:68–82, 2019.